Andrés Iglesias
Nobuki Takayama (Eds.)

LNCS 4151

Mathematical Software – ICMS 2006

Second International Congress
on Mathematical Software
Castro Urdiales, Spain, September 2006, Proceedings

Springer

Lecture Notes in Computer Science 4151

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison
Lancaster University, UK
Takeo Kanade
Carnegie Mellon University, Pittsburgh, PA, USA
Josef Kittler
University of Surrey, Guildford, UK
Jon M. Kleinberg
Cornell University, Ithaca, NY, USA
Friedemann Mattern
ETH Zurich, Switzerland
John C. Mitchell
Stanford University, CA, USA
Moni Naor
Weizmann Institute of Science, Rehovot, Israel
Oscar Nierstrasz
University of Bern, Switzerland
C. Pandu Rangan
Indian Institute of Technology, Madras, India
Bernhard Steffen
University of Dortmund, Germany
Madhu Sudan
Massachusetts Institute of Technology, MA, USA
Demetri Terzopoulos
University of California, Los Angeles, CA, USA
Doug Tygar
University of California, Berkeley, CA, USA
Moshe Y. Vardi
Rice University, Houston, TX, USA
Gerhard Weikum
Max-Planck Institute of Computer Science, Saarbruecken, Germany

Andrés Iglesias   Nobuki Takayama (Eds.)

Mathematical Software – ICMS 2006

Second International Congress
on Mathematical Software
Castro Urdiales, Spain, September 1-3, 2006
Proceedings

Springer

Volume Editors

Andrés Iglesias
University of Cantabria
Department of Applied Mathematics and Computational Sciences
E.T.S. Ingenieros de Caminos
Av. de los Castros, s/n, 39005, Santander, Spain
E-mail: iglesias@unican.es

Nobuki Takayama
Kobe University
Faculty of Science
Department of Mathematics
1-1, Rokkodai, Nada-ku, Kobe 657-8501, Japan
E-mail: takayama@math.kobe-u.ac.jp

Library of Congress Control Number: 2006931195

CR Subject Classification (1998): G.1, G.2, G.4, F.2, I.1, I.3.5

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN	0302-9743
ISBN-10	3-540-38084-1 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-38084-9 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2006
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11832225 06/3142 5 4 3 2 1 0

Preface

This volume contains the outstanding collection of invited papers and refereed papers selected for the Second International Congress on Mathematical Software, ICMS 2006, held in Castro Urdiales, Spain, September 1-3, 2006. We cordially invite you to visit the ICMS 2006 website http://www.icms2006.unican.es where you can find all relevant information about this interesting event.

ICMS 2006 was the second edition of this congress, which follows up the successful ICMS 2002 held in Beijing, China. Since its inception, this congress has been a satellite event of the International Congress of Mathematicians - ICM, the world's largest conference on mathematics, celebrated every four years since the edition of 1900 in Paris, where David Hilbert presented his 23 famous problems. For the first time, this 2006 edition of ICM is held in Spain (see: http://www.icm2006.org for details), and so is ICMS 2006.

This congress was devoted to all aspects of mathematical software, whose appearance is — in our opinion — one of the most important events in mathematics. Mathematical software systems are used to construct examples, to prove theorems, and to find new mathematical phenomena. Conversely, mathematical research often motivates developments of new algorithms and new systems. Beyond mathematics, mathematical software systems are becoming indispensable tools in many branches of science and technology.

The development of mathematical software systems relies on the cooperation of mathematicians, algorithm designers, programmers, and the feedback from users. The main audiences of this conference are mathematical software developers and programming mathematicians, but we also intend to provide an opportunity to discuss these topics with mathematicians and users from applied areas. The congress focused on the following major themes:

1. Software engineering problems for mathematical software
2. Mathematics and media (including user interfaces and integration of documents and software systems)
3. Mathematics related to mathematical software (experiments, algorithms) as well as scientific computing
4. High-performance computing
5. Applications of mathematical software.

ICMS 2006 comprised ten sessions — devoted to different mathematical software issues — some plenary talks and a general track. We would like to thank all session organizers for their diligent work, which further enhanced the congress standing and to all reviewers for their expertise and generous effort which led to a very high-quality event with excellent papers and presentations. We specially recognize the contribution of the Program Committee and Advisory Program Committee members for their tremendous support and for making this congress a very sucessful event.

To keep all congress activities within the planned three days we scheduled three parallel sessions, with the remarkable exception of the plenary talks. We are thankful to the plenary and invited speakers for their kind acceptance to attend to this congress and deliver a talk.

All presentations were given at the congress venue, La Residencia, a recently rebuilt palace intended to become the new "Centro Internacional de Encuentros Matemáticos - CIEM" (International Center for Mathematical Meetings), an ambitious initiative of the Spanish mathematical community oriented towards the creation of an international center for mathematics in Spain. La Residencia offers advanced audio-visual equipment for conferences and presentations as well as a unique indoors–outdoors environment for both fruitful discussions and relaxing enjoyment. In addition, it is placed in Castro Urdiales, one of the most beautiful places of the north coast of Spain. Castro Urdiales — in the region of Cantabria — is an old fishermen's town with magnificient buildings, beautiful promenades, a very rich history and a lovely combination between the antique and the modern. We thank the Director of CIEM, Laureano González-Vega, for his support before and during the congress, and the Castro Urdiales Town Council for the facilities to organize the congress in this beautiful place.

ICMS 2006 was jointly organized by the Department of Mathematics, Statistics and Computation and the Department of Applied Mathematics and Computational Sciences of the University of Cantabria, Spain. We thank both departments for their encouranging support to this congress from the very beginning. We would like to express our gratitude to the Local Organizing Committee — and especially to the local Organizing Chair, Jaime Gutierrez — for their persistent and enthusiastic work towards the success of ICMS 2006. The smoothness of the organization was ensured by the cooperation of the Ph.D. students Alvar Ibeas and Akemi Gálvez, who took care of a number of tasks, including the congress website and the posters.

We owe special thanks to our sponsors: AddLink Software Científico, the University of Cantabria, the International Congress of Mathematicians, the Spanish Ministry of Education and Science, the Fundación Leonardo Torres Quevedo and the Centro Internacional de Encuentros Matemáticos, for their continuous support without which this congress would not be possible. We also thank our publisher, Springer — and especially LNCS editor Alfred Hofmann and his team — for their acceptance to publish the proceedings and for their kind assistance and cooperation during the editing process.

Finally, we thank all authors for their submissions and all congress attendants for making ICMS 2006 truly an excellent forum on mathematical software, facilitating exchange of ideas, fostering new collaborations and shaping the future of mathematical software. Last, but certainly not least, we wish to thank our readers for their interest in this volume. We really hope you find in these pages interesting material and fruitful ideas for your future work.

September 2006

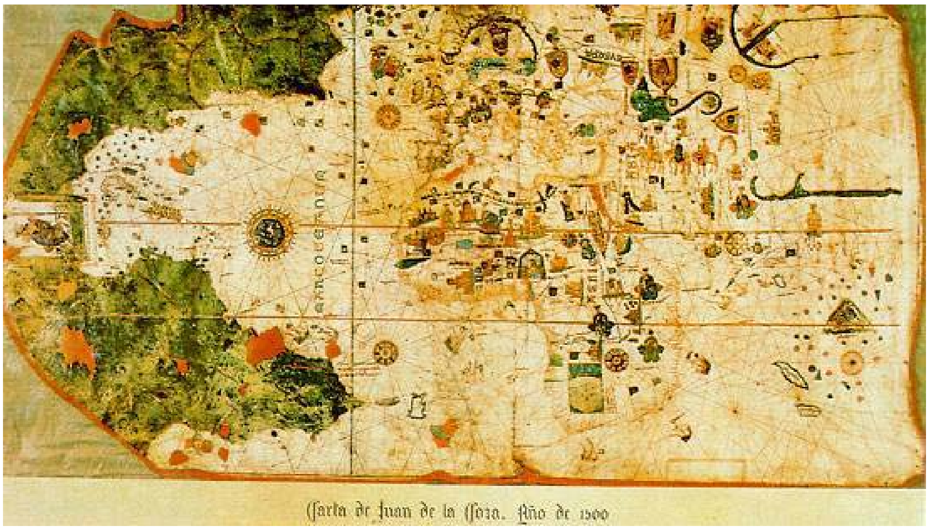
Nobuki Takayama
Andrés Iglesias

Note

The picture below shows the famous “Mappa Mundi” the world’s oldest representation of the New World, dating back to 1500 AD. It was made by Juan de La Cosa, a Spanish sea captain who sailed with Columbus to the new world on his first three voyages. Juan de la Cosa was born in Santoña — a small town on the north coast of Spain — near Castro Urdiales (ICMS 2006 congress venue). After several trips to the coast of Africa, he became a proficient navigator and map maker and when Christopher Columbus was planning his 1492 first voyage to the “New World,” he joined the expedition with his own vessel: Santa Maria.

Of special interest in this map is the outline of Cuba, which Christopher Columbus never believed to be an island. In fact, on Columbus’s third trip, Juan de la Cosa traveled alongside Columbus on the “La Niña” ship and a difference of opinion arose concerning the lands newly discovered. Juan de la Cosa was one of the signers to the Perez-Luna agreement, which stated that Cuba was a continent. He signed this under the orders of Columbus, although he was sure that the island now known as Cuba was *not* forming part of the continent. Amazingly, Columbus remarked: "Juan de la Cosa thinks he knows more than I do in the art of navigating".

The map is currently in the Naval Museum of Madrid (ICM 2006 congress venue) and it is a good example of the application of mathematical techniques to cartography.



Organization

ICMS 2006 was organized by the Department of Mathematics, Statistics and Computation and the Department of Applied Mathematics and Computational Sciences of the University of Cantabria, Spain.

Conference Chairs

Nobuki Takayama (Kobe University, Japan)
Andrés Iglesias (University of Cantabria, Spain)

Program Committee

Henk Barendregt (Nijmegen University, The Netherlands)
Komei Fukuda (ETH Zentrum, Zurich, Switzerland)
Xiaoshan Gao (Academia Sinica, Beijing, China)
Gert-Martin Greuel (University of Kaiserslautern, Germany)
Jaime Gutiérrez (University of Cantabria, Spain)
Steve Linton (University of St. Andrews, UK)
Tetsuo Ida (University of Tsukuba, Japan)
Andres Iglesias (University of Cantabria, Spain)
Michael Joswig (Technische Universität Darmstadt, Germany)
Ken Nakamula (Tokyo Metropolitan University, Japan)
Michael Pohst (Technische Universität Berlin, Germany)
Konrad Polthier (Konrad Zuse Zentrum, Berlin, Germany)
Rafael Sendra (University of Alcalá de Henares, Spain)
David Sevilla (Concordia University, Canada)
Nobuki Takayama (Kobe University, Japan)
Jan Verschelde (University of Illinois at Chicago, USA)
Freek Wiedijk (Nijmegen University, The Netherlands)
Joris Van der Hoeven (Université Paris-Sud, France)

Advisory Program Committee

Arjeh Cohen (Eindhoven University of Technology, The Netherlands)
Jack Dongarra (University of Tennessee,USA)
Jose M. Gutiérrez (University of Cantabria, Spain)
Bernd Sturmfels (University of California Berkeley, USA)
Dongming Wang (Université Pierre et Marie Curie and CNRS, France)

Local Organizing Committee

Jaime Gutiérrez (Chair) (University of Cantabria, Spain)
Andrés Iglesias (University of Cantabria, Spain)
Alvar Ibeas (University of Cantabria, Spain)
Akemi Gálvez (University of Cantabria, Spain)

Session Organizers

New Developments on Computer Algebra Packages
Andrés Iglesias (University of Cantabria, Spain)
Tetsuo Ida (University of Tsukuba, Japan)

Interfacing Computer Algebra and Mathematical Visualization
Konrad Polthier (Konrad Zuse Zentrum, Berlin, Germany)

Computer Mathematics
Freek Wiedijk (Nijmegen University, The Netherlands)
Henk Banrendregt (Advisory Organizer) (Nijmegen University, The Netherlands)

Software for Algebraic Geometry and Related Topics
Nobuki Takayama (Kobe University, Japan)
Gert-Martin Greuel (Advisory Organizer) (University of Kaiserslautern, Germany)

Number Theoretical Software
Ken Nakamula (Tokyo Metropolitan University, Japan)
Michael Pohst (Technische Universität Berlin, Germany)

Methods in Computational Number Theory
David Sevilla (Concordia University, Canada)
Jaime Gutiérrez (University of Cantabria, Spain)

Free Software for Computer Algebra
Joris Van der Hoeven (Université Paris-Sud, France)

Software for Optimization and Geometric Computation
Komei Fukuda (ETH Zentrum, Zurich, Switzerland)
Michael Joswig (Technische Universität Darmstadt, Germany)

Methods and Software for Computing Mathematical Functions
Amparo Gil (University of Cantabria, Spain)
Javier Segura (University of Cantabria, Spain)

Access to Mathematics on the Web
Paul Libbrecht (German Research Center for Artificial Intelligence, Germany)

Sponsoring Organizations

AddLink Scientific software

The University of Cantabria

The International Congress of Mathematicians, ICM 2006

Spanish Ministry of Education and Science

Fundación Leonardo Torres Quevedo

CIEM: International Center of Mathematical Meetings

Table of Contents

A General Computational Scheme for Testing Admissibility of Nilpotent Orbits of Real Lie Groups of Inner Type

Alfred G. Noël

Department of Mathematics University of Massachusetts
Boston, MA 02125-3393, USA
anoel@math.umb.edu

Abstract. One of the most fundamental problems in the field of Representation Theory is the description of all the unitary representations of a given group. For non-compact real reductive Lie groups, there is evidence that new unitary representations can be obtained from data provided by their *admissible* nilpotent orbits. In this paper, we describe a general scheme for determining the admissibility of a given real nilpotent orbit. We implement some parts of the scheme using the software system *LiE*. We give a detailed example and study the complexity of the algorithms.

1 Introduction

For real reductive Lie groups there is substantial evidence linking admissible orbits and unitary representations. Standard methods such as parabolic induction can be used to associate unitary representations to admissible semisimple orbits. However the theory is not well developed for nilpotent orbits. For example we do not have a clear strategy to attach a representation to a general admissible orbit. In many cases Vogan and Barbash have proven that the set of irreducible representations obtained from the admissible nilpotent orbits or the unipotent representations are the building blocks for all unitary representations of the group. In the case of real semisimple Lie groups the admissible nilpotent orbits seem to be very good candidates for which a general technique may be developed [10].

Unless otherwise specified, $\mathfrak{g}$ will be a real semisimple Lie algebra of inner type with adjoint group G and Cartan decomposition $\mathfrak{g} = \mathfrak{k} \oplus \mathfrak{p}$ relative to a Cartan involution θ. The term inner type means that rank($\mathfrak{g}$) = rank($\mathfrak{k}$). Some authors use the term equal rank to refer to such algebras. We will denote by $\mathfrak{g}_{\mathbb{C}}$ the complexification of $\mathfrak{g}$. Let σ be the conjugation of $\mathfrak{g}_{\mathbb{C}}$ with respect to $\mathfrak{g}$. Then $\mathfrak{g}_{\mathbb{C}} = \mathfrak{k}_{\mathbb{C}} \oplus \mathfrak{p}_{\mathbb{C}}$ where $\mathfrak{k}_{\mathbb{C}}$ and $\mathfrak{p}_{\mathbb{C}}$ are obtained by complexifying $\mathfrak{k}$ and $\mathfrak{p}$ respectively. K will be a maximal compact Lie subgroup of G with Lie algebra $\mathfrak{k}$ and $K_{\mathbb{C}}$ will be the connected subgroup of the adjoint group $G_{\mathbb{C}}$ of $\mathfrak{g}_{\mathbb{C}}$, with Lie algebra $\mathfrak{k}_{\mathbb{C}}$. It is well known that $K_{\mathbb{C}}$ acts on $\mathfrak{p}_{\mathbb{C}}$ and the number of nilpotent orbits of $K_{\mathbb{C}}$ in $\mathfrak{p}_{\mathbb{C}}$ is finite. Furthermore, for a nilpotent $e \in \mathfrak{p}_{\mathbb{C}}$, $K_{\mathbb{C}} \cdot e$ is a connected component of $G_{\mathbb{C}} \cdot e \cap \mathfrak{p}_{\mathbb{C}}$.

© Springer-Verlag Berlin Heidelberg 2006

2 The Kostant-Sekiguchi Correspondence

A triple (x, e, f) in $\mathfrak{g}_{\mathbb{C}}$ is called a *standard triple* if $[x, e] = 2e$, $[x, f] = -2f$ and $[e, f] = x$. If $x \in \mathfrak{k}_{\mathbb{C}}$ and e and $f \in \mathfrak{p}_{\mathbb{C}}$ then (x, e, f) is a *normal* triple. It is a result of Kostant and Rallis [2] that any nilpotent e of $\mathfrak{p}_{\mathbb{C}}$ can be embedded in a standard normal triple (x, e, f). Moreover e is $K_{\mathbb{C}}$-conjugate to a nilpotent e' inside of a normal triple (x', e', f') with $\sigma(e') = f'$ [7]. The triple (x', e', f') will be called a $Kostant - Sekiguchi$ or KS-triple.

Every nilpotent E' in $\mathfrak{g}$ is G-conjugate to a nilpotent E embedded in a triple (H, E, F) in $\mathfrak{g}$ with the property that $\theta(H) = -H$ and $\theta(E) = -F$ [7]. Such a triple will be called a KS-triple also.

Define a map c from the set of KS-triples of $\mathfrak{g}$ to the set of normal triples of $\mathfrak{g}_{\mathbb{C}}$ as follows:

$$x = c(H) = i(E - F)$$

$$e = c(E) = \frac{1}{2}(H + i(E + F))$$

$$f = c(F) = \frac{1}{2}(H - i(E + F))$$

The triple (x, e, f) is called the Cayley transform of (H, E, F). It is easy to verify that the triple (x, e, f) is a KS-triple and that $x \in i\mathfrak{k}$. The Kostant-Sekiguchi correspondence [7] gives a one to one map between the set of G-conjugacy classes of nilpotents in $\mathfrak{g}$ and the $K_{\mathbb{C}}$-conjugacy classes of nilpotents in $\mathfrak{p}_{\mathbb{C}}$. This correspondence sends the zero orbit to the zero orbit and the orbit through the nilpositive element of a KS-triple to the one through the nilpositive element of its Cayley transform. Michèle Vergne [8] has proved that there is in fact a diffeomorphism between a G-conjugacy class and the $\mathfrak{k}_{\mathbb{C}}$-conjugacy class asssociated to it by the Kostant-Sekiguchi correspondence.

3 Admissible Orbits

The *coadjoint representation* of G on $\mathfrak{g}^*$, the dual space of $\mathfrak{g}$, is defined as follows:

$$Ad_g^*(f)(E) = f(Ad_{g^{-1}}(E)) \qquad g \in G \quad \text{and} \quad E \in \mathfrak{g}.$$

The differential of the above representation on $\mathfrak{g}$ is:

$$ad_X^*(f)(E) = -f([X, E]) \qquad X \in \mathfrak{g}.$$

Let $\mathfrak{g} = \mathfrak{k} + \mathfrak{p}$ be a Cartan decomposition of $\mathfrak{g}$. It is known that there exists a G-invariant non-degenerate bilinear form $\langle , \rangle$ on $\mathfrak{g}$ which is negative definite on $\mathfrak{k}$ and positive definite on $\mathfrak{p}$. We will make use of such a form to identify coadjoint orbits in $\mathfrak{g}^*$ with adjoint orbits in $\mathfrak{g}$. Therefore we shall determine admissible nilpotent orbits of G in g. We will consider the cases where G is adjoint or simply connected.

For $\lambda \in \mathsf{g}^*$ define $G^\lambda = \{g \in G : g(\lambda) = \lambda\}$ and $\mathsf{g}^\lambda = \{X \in \mathsf{g} : \lambda([X, \mathsf{g}]) = 0\}$. Then there is a non-degenerate symplectic form ω_λ on $\mathsf{g}/\mathsf{g}^\lambda$ given by $\omega_\lambda(X + \mathsf{g}^\lambda, Y + \mathsf{g}^\lambda) = \lambda([X, Y])$. Moreover G^λ preserves ω_λ. Denote by $Sp(\omega_\lambda)$ the symplectic group defined by ω_λ. Define the group :

$$\tilde{G}^\lambda = \{(g, m) \in G^\lambda \times M(\omega_\lambda) : \psi(g) = \pi(m)\}$$

where $M(\omega_\lambda)$ is the *metaplectic group* associated with $Sp(\omega_\lambda)$, ψ is a natural homomorphism from G^λ to $Sp(\omega_\lambda)$ and π is defined as follows:

$$1 \longrightarrow \{1, \xi\} \longrightarrow M(\omega_\lambda) \longrightarrow Sp(\omega_\lambda) \longrightarrow 1$$

$M(\omega_\lambda)$ is a two-fold covering of $Sp(\omega_\lambda)$ which can be pulled back to give $\tilde{G}^\lambda$ as a double cover of G^λ.

Definition 1. *A representation (ρ, V) of $\tilde{G}^\lambda$ is called admissible if $\rho(\xi) = -1_V$ and $d\rho(E) = \sqrt{-1}\lambda(E)1_V$ for all $E \in \mathsf{g}^\lambda$. The linear functional λ is said to be admissible if $\tilde{G}^\lambda$ has at least one admissible representation.*

An element E of g is admissible if and only if its image λ_E under $\langle , \rangle$ is admissible in g^*.

We note that in the case of the trivial nilpotent orbit $\tilde{G}^\lambda$ is trivial. Hence the trivial nilpotent orbit is always admissible. This is the orbit to which one would want to attach the trivial representation in the Orbit Method scheme. From now on we will concern ourselves with non-zero nilpotent orbits.

J. Schwartz [6] translated the admissibility of real nilpotent orbits into the admissiblity of nilpotent elements in complex symmetric spaces via the so-called Kostant-Sekiguchi [7] correspondence. T. Ohta [5] used this technique to determine admissibility of nilpotent orbits in the classical Lie algebras. We should also point out that Schwartz has classified nilpotent admissible orbits of several classical Lie groups.

Let e be non zero nilpotent in $\mathsf{p}_{\mathbb{C}}$. Then $K^e_{\mathbb{C}}$ acts $\mathsf{k}_{\mathbb{C}} / \mathsf{k}^e_{\mathbb{C}}$ and $(\mathsf{k}_{\mathbb{C}}/\mathsf{k}^e_{\mathbb{C}})^*$. Define the character δ_e of $K^e_{\mathbb{C}}$ as follows:

$$\delta_e(g) = (det(g|_{\mathsf{k}_{\mathbb{C}}/\mathsf{k}^e_{\mathbb{C}}}))^{-1} \qquad g \in K^e_{\mathbb{C}}$$

Using δ_e and the homomorphism $s : \mathbb{C}^\times \to \mathbb{C}^\times$, with $s(z) = z^2$ we obtain the following double cover of $K^e_{\mathbb{C}}$:

$$\tilde{K}^e_{\mathbb{C}} = \{(g, z) \in K^e_{\mathbb{C}} \times \mathbb{C}^\times : \delta_e(g) = z^2\}$$

It turns out that this covering is precisely the one induced on K^E, when $E = c(e)$ and λ identified with E, by the metaplectic covering $M(\omega^\lambda)$. Schwartz used such a covering to show that admissibility of real nilpotent orbits is equivalent to admissibility of nilpotent orbits in complex symmetric spaces.

Definition 2. *A representation χ of $K^e_{\mathbb{C}}$ is said to be admissible if its differential is half the differential of δ_e. The nilpotent e is admissible if $K^e_{\mathbb{C}}$ has at least one admissible representation.*

The following theorem is due to J. Schwartz [6].

Theorem 1. *There is a natural bijection between the equivalence classes of nilpotent admissible G-orbits and the equivalence classes of nilpotent admissible $K_{\mathbb{C}}$-orbits.*

See [11] Lemma 7.8, Theorem 7.11, Theorem 7.14. for a proof.

In fact the question of admissibility of nilpotent orbits can be translated into a question on the representation of the identity component $(K^e_{\mathbb{C}})_\circ$ of the group $K^e_{\mathbb{C}}$. See [10], [11].

The character δ_e is difficult to compute explicitly from the above description. However Takuya Ohta **[O]** has found an explicit description of δ_e which we shall discuss below. We will need some notations. Let (x, e, f) to be a KS-triple with $x \in i\mathsf{k}$. From the representation theory of sl_2, $\mathsf{g}_{\mathbb{C}}$ has the following eigenspace decomposition:

$$\mathsf{g}_{\mathbb{C}} = \bigoplus_{j \in \mathbb{Z}} \mathsf{g}^{(j)}_{\mathbb{C}} \qquad \text{where } \mathsf{g}^{(j)}_{\mathbb{C}} = \{z \in \mathsf{g}_{\mathbb{C}} | [x, z] = jz\}.$$

Similarly we have:

$$\mathsf{k}_{\mathbb{C}} = \bigoplus_{j \in \mathbb{Z}} \mathsf{k}^{(j)}_{\mathbb{C}} \qquad \text{where } \mathsf{k}^{(j)}_{\mathbb{C}} = \{z \in \mathsf{k}_{\mathbb{C}} | [x, z] = jz\}$$

and

$$\mathsf{p}_{\mathbb{C}} = \bigoplus_{j \in \mathbb{Z}} \mathsf{p}^{(j)}_{\mathbb{C}} \qquad \text{where } \mathsf{p}^{(j)}_{\mathbb{C}} = \{z \in \mathsf{p}_{\mathbb{C}} | [x, z] = jz\}.$$

Moreover the centralizers of e in $\mathsf{k}_{\mathbb{C}}$ and $\mathsf{p}_{\mathbb{C}}$ decompose as:

$$\mathsf{k}^e_{\mathbb{C}} = \bigoplus_{j \in \mathbb{Z}} (\mathsf{k}^e_{\mathbb{C}} \cap \mathsf{k}^j_{\mathbb{C}}) = \bigoplus_{j \geq 0} (\mathsf{k}^e_{\mathbb{C}} \cap \mathsf{k}^j_{\mathbb{C}})$$

and

$$\mathsf{p}^e_{\mathbb{C}} = \bigoplus_{j \in \mathbb{Z}} (\mathsf{p}^e_{\mathbb{C}} \cap \mathsf{p}^j_{\mathbb{C}}) = \bigoplus_{j \geq 0} (\mathsf{p}^e_{\mathbb{C}} \cap \mathsf{p}^j_{\mathbb{C}})$$

It is a known fact that $\mathsf{k}^e_{\mathbb{C}} = \mathsf{k}^{(x,e,f)}_{\mathbb{C}} \oplus u_e$, where $\mathsf{k}^{(x,e,f)}_{\mathbb{C}} = \mathsf{k}^e_{\mathbb{C}} \cap \mathsf{k}^0_{\mathbb{C}}$ is a reductive subalgebra of $\mathsf{k}^e_{\mathbb{C}}$ and $u_e = \bigoplus_{j>0} (\mathsf{k}^e_{\mathbb{C}} \cap \mathsf{k}^j_{\mathbb{C}})$ consists of nilpotent elements. Denote by $d\delta_e$ the differential of δ_e then we have the next two lemmas from [5]:

Lemma 1. *The differential of δ_e is trivial on u_e. Suppose that $\mathsf{t}_{\mathbb{C}}$ is a toral subalgebra of $\mathsf{k}_{\mathbb{C}}$ containing x. Then the centralizer $\mathsf{t}^e_{\mathbb{C}}$ of e in $\mathsf{t}_{\mathbb{C}}$ is a toral subalgebra of $\mathsf{t}^{(x,e,f)}_{\mathbb{C}}$ and $d\delta_e$ on $\mathsf{t}^e_{\mathbb{C}}$ is given by*

$$d\delta_e(t) = \sum_{i\geq 1} tr(ad(t)|_{\mathsf{k}^i_{\mathbb{C}}}) - \sum_{i\geq 2} tr(ad(t)|_{\mathsf{p}^i_{\mathbb{C}}}) = -\sum_{i\geq 1} tr(ad(t)|_{\mathsf{p}^i_{\mathbb{C}}}) + \sum_{i\geq 2} tr(ad(t)|_{\mathsf{k}^i_{\mathbb{C}}})$$

The preceding lemma suggests that admissibility is a question on tori of $\mathsf{k}^{(x,e,f)}_{\mathbb{C}}$. This is indeed the case as described in the next two results of Ohta [5].

Lemma 2. *Let t_1 be a Cartan subalgebra of $\mathsf{k}^{(x,e,f)}_{\mathbb{C}}$ and T_1 the corresponding connected subgroup of $(K^{(x,e,f)}_{\mathbb{C}})_\circ$ the identity component of $K^{(x,e,f)}_{\mathbb{C}}$. Then e is admissible if and only if there exists a character, χ, of T_1 such that $\delta_e(g) = (\chi(g))^2$ for all $g \in T_1$.*

Corollary 1. *Suppose that $T_1 \simeq (\mathbb{C}^\times)^r$. Then $d\delta_e|_{t_1}$ is a linear map : $\mathbb{C}^r \longrightarrow \mathbb{C}$. From the previous lemma e is admissible if and only if each coefficient of z_i in the linear combination of $(z_1, z_2, \ldots z_r)$ is an even integer.*

4 Algorithms

From the previous section we see that in order to develop a computational scheme for classifying admissible nilpotent orbits in a real Lie algebra we need to :

1. implement algorithms for computing representatives of $K_{\mathbb{C}}$-conjugacy classes of maximal tori in the reductive centralizer $\mathsf{k}^{(x,e,f)}_{\mathbb{C}}$.
2. implement an algorithm for computing Otha's character δ_e.
3. Evaluate δ_e on a basis of a torus t_1 computed in (1) and use the above corollary to decide whether or not the given orbit is admissible.

Step 1 requires implementing an algorithm given in [3] for computing maximal tori in $\mathsf{k}^{(x,e,f)}_{\mathbb{C}}$. Next, we develop an algorithm written in pidgin *LiE* to compute δ_e. Readers who are not familiar with the software system *LiE* should consult [9].

4.1 Algorithm I

```
# This function computes the Ohta's character associated to
# a real nilpotent element via the Kostant-Sekiguchi
# correspondence.
# This function assumes that the real form of the
# algebra g is of inner-type. We also assume that the Vogan
# system of simple roots contains exactly one non-compact root.
```

```
#Author:   Alfred G. Noel
#Purpose: Classification of admissible nilpotent orbits.

# Date : January 10, 2006.

Ohta_character(int ncptindex; vec neutral; mat kroots;  grp g)=
{
# ncpt_index: indicate which roots are non compact.
# neutral: semisimple element in a normal  SL2-triple associated
# with the orbit.
# It should be expressed in the root system of the algebra g.
# kroots: a system of simple roots for K.
# g: the type of the simple group

setdefault(g); n  = n_pos_roots; l   = Lie_rank;
alpha = pos_roots; roots = alpha^-alpha;

k_index =0;
p_index =0;
k_sum = null(l);
p_sum = null(l);

#Compute degree positive roots

for k = 1 to  2*n  do
    degree = 0;
for i = 1 to l do
for j = 1 to l do
degree = degree + neutral[j]*roots[k][i]*
Cartan(alpha[i],alpha[j]);
od; od;
if (roots[k][ncptindex] == 1 || roots[k][ncptindex] == -1)
then if degree  > 0 then p_index = p_index +1;
               p_sum = p_sum + roots[k]; fi;
else
    if degree  > 1 then k_index = k_index +1;
               k_sum = k_sum + roots[k]; fi;
  fi; od;

# Compute delta
delta = k_sum - p_sum;
return(delta);
}
```

Once we have determined the Ohta's character of the given orbit we evaluate it on the torus t_1 obtained in step 1. We need a few more notations. We use the

Bourbaki simple root system $\Delta = \{\alpha_1, \dots \alpha_l\}$ for $\mathrm{g}_{\mathbb{C}}$. Hence $t_1 = \bigoplus_{\mu} H_\mu$ where $H_\mu = \sum_{i=1}^{l} \mu_i H_{\alpha_i}$ with $\mu_i \in \mathbb{Z}$.

4.2 Algorithm II

```
# This function evaluates the Ohta's character of a
# nilpotent element on  a maximal torus of the reductive
# centralizer associated with it.
# Author:   Alfred G. Noel
# Purpose:   Classification of admissible nilpotent orbits.

# Date : January 12, 2006.

Check_admissibility(vec delta;  mat torus; int dim_torus; grp g)=
{
# delta: one dimensional array describing the Ohta's character
# torus : a two dimensional array whose rows are basis element of
# the maximal torus described above and computed in step 1.
# dim_torus: dimension of the torus
# g: the type of the simple group

setdefault(g);
l  = Lie_rank;
coeff= null(dim_torus);
for k = 1 to dim_torus do
print (" ");
for i = 1 to l do
      for j = 1 to l do
            incr = delta[i]*torus[k][j];
            prod = Cartan(alpha[i], alpha[j]);
           coeff[k] = coeff[k] + incr*prod;
      od;
od;
od;
for i = 1 to dim_torus do
if coeff[i] % 2 !=0 then
 print( "This orbit not admissible under the Adjoint group");
 break;
 fi;
od;
return(coeff);
}
```

4.3 Example

Here is a *LiE* program for the Lie group E_8. This group is the largest exceptional group and serves as a benchmark case for computational schemes in Representation theory.

```
# EVIII a real form of E8 orbit 7
# Djokovic label: 11000001
# This program is designed and implemented
#  by Alfred G. Noel
# Purpose: Investigation of admissible representations

setdefault E8;
n  = n_pos_roots;
l   = Lie_rank;
alpha = pos_roots;
g = E8
# K_C simple root system #
beta = [[2,2,3,4,3,2,1,0], [0,0,0,0,0,0,0,1],
 [0,0,0,0,0,0,1,0],[0,0,0,0,0,1,0,0],
 [0,0,0,0,1,0,0,0],[0,0,0,1,0,0,0,0],
 [0,1,0,0,0,0,0,0],[0,0,1,0,0,0,0,0]];

#  The triple (x,e,f) : computed from a previous alogrithm
#  in LNCS. See reference

x=[3,  6, 7, 11, 9, 7, 5, 3];
e =  [[1,2,2,3,2,1,1,1], [1,2,2,3,3,3,2,1], [1,2,3,5,4,3,2,1]];
f = -e;

#Compute   delta:

delta =Ohta_character(1, x ,beta, g);

print(delta);

#Check admissibility:

# Compute the torus from a previous alogrithm in LNCS.
#   See reference
torus =[[0,0,1,0, 0, 0, 0, 0], [ 0,0, 0, 0,1, 0, 0,0],
[ 0,0, 0, 0, 0, 0,1,0],[ 0,1, 0, 1, 0,0, 0,-1],
[ 1, 0, 0, 1, 0,1, 0,1]];

coeff = null(5);
coeff = Check_admissibility(delta,torus,5,g);
print ( "FINISH");
```

Here is the corresponding *LiE* session:

```
LiE version 2.2 created on Nov 22 1997 at 16:50:29
Authors: Arjeh M. Cohen, Marc van Leeuwen, Bert Lisser.
Mac port by S. Grimm
Public distribution version

type '?help' for help information
type '?' for a list of help entries.
> read kdd
> read admis
> read mytest1
     [-17,-33,-39,-61,-50,-39,-28,-17]
       This orbit not admissible under the Adjoint group
     FINISH
> coeff
     [0,0,0,1,-1]
>
```

The functions are in the files “kdd” and “admis”. The example is in the file “mytest1”. As the reader can see this orbit is not admissible under the adjoint group since $\delta_e(z_1, z_2, z_3, z_4, z_5) = z_4 - z_5$.

We use two software systems *LiE* version 2.2 and Mathematica version 4.0 to implement the algorithms. The fact that Mathematica offers an environment for symbolic computation makes it possible to solve certain systems of equations in a nice way. We could have used Maple for that purpose also. We have tried to use some Mathematica packages for Coxeter groups but we were not satisfied with the results. Mathematica [12] is a well known software system. *LiE* is used mostly by mathematicians and physicists who perform computations of a Lie group theoretic nature. *LiE* does not provide an environment for symbolic computation. However using vectors and matrices with integer entries as basic computational objects it does allow the programmer to access and test many non trivial results and conjectures about complex reductive Lie groups, their representations and their Weyl groups. Moreover, it only works with integer numbers and does not have a polynomial system of equations solver. *LiE* is written in C and run mostly on Unix systems. There are also some executables for the classic Macintosh system. More information on *LiE* can be found in [9]. The computations were carried out on an iMac 1 GHz PowerPc G4 running Mac OS 10.3.9 with 1GB DDR SDRAM.

Both algorithms used the function *Cartan* as their main evaluator. In *LiE* this function is well implemented, very fast and quite stable. Let r be number of roots of $\mathfrak{g}$ and l, the semisimple rank of $\mathfrak{g}$. Then, in the worst case $Ohta\_character()$ is $\mathcal{O}(rl^2(\mathcal{O}(Cartan())))$ while $Check\_admissibility$ is $\mathcal{O}(l^3(\mathcal{O}(Cartan())))$ since the dimension of the torus is bounded by l.

5 Conclusion and New Directions

Mathematica is a proprietary multi-purpose software system and the source code is not available to the public. This is not the case for *LiE* which is designed specifically for computations in complex Lie groups and their representations. The mathematical algorithms are well conceived and their implementations seem to work well on average. In some cases where the Weyl group had to be processed, we observed some scalability problems. We are currently investigating the possibility of adding more functions which deal with real Lie groups and their representations. The scope and feasibility of such a project are being evaluated. An other approach is to realize *LiE* as a Mathematica package. We have not looked at this alternative. However we believed that such package may not work as well as the stand alone version. It is also desirable to design a good graphic user interface for *LiE. LiE* is maintained by M. A. A. van Leeuwen at l'Université de Poitiers in France. More information on *LiE* can be obtained at http://wwwmathlabo.univ-poitiers.fr/˜maavl/LiE/.

We should point out that John Stembridge from the University of Michigan USA has created a Maple software package called Coxeter/Weyl for manipulating weights and characters of irreducible representations of semisimple Lie algebras, including functions for computing weight multiplicities, tensor product decompositions, and branching. However, we found *LiE* to be faster and easier to use for the type of algorithms we sought to implement. Information on Coxeter/Weyl can be obtained at http://www.math.lsa.umich.edu/˜jrs/maple.html#coxeter.

Recently, the author has joined *The Atlas of Lie Groups and Representations.* This is a project to make available information about representations of semisimple Lie groups over real and p-adic fields. Of particular importance is the problem of the unitary dual: classifying all of the irreducible unitary representations of a given Lie group. It might be possible to implement the previous algorithms in this new and potentially very large C++ system. For more information on the atlas the reader should visit: http://atlas.math.umd.edu/.

Finally we need to add to the package the capability to study admissibility under the simply connected cover of the adjoint group. Let us consider the following data obtained for orbit 6 of $E_{8(8)}$ in [1] or [4]:

6. 10001000 $\mathsf{k}_{\mathbb{C}}^{(x,e,f)} \simeq A_3 + T_1$

$$x = 4H_{\alpha_1} + 7H_{\alpha_2} + 9H_{\alpha_3} + 14H_{\alpha_4} + 12H_{\alpha_5} + 9H_{\alpha_6} + 6H_{\alpha_7} + 3H_{\alpha_8}$$

$$\begin{aligned} e = {} & X_{\alpha_1+\alpha_2+2\alpha_3+3\alpha_4+3\alpha_5+2\alpha_6+\alpha_7} + X_{\alpha_1+2\alpha_2+2\alpha_3+3\alpha_4+3\alpha_5+2\alpha_6+\alpha_7+\alpha_8} \\ & + X_{\alpha_1+2\alpha_2+2\alpha_3+4\alpha_4+3\alpha_5+2\alpha_6+2\alpha_7+\alpha_8} + X_{\alpha_1+2\alpha_2+3\alpha_3+4\alpha_4+3\alpha_5+3\alpha_6+2\alpha_7+\alpha_8} \end{aligned}$$

$$\begin{aligned} \mathsf{t}_{\mathbb{C}}^1 = {} & \mathbb{C}(H_{\alpha_2} - H_{\alpha_8}) \oplus \mathbb{C}(H_{\alpha_4} - H_{\alpha_7}) \oplus \mathbb{C}(H_{\alpha_3} - H_{\alpha_6}) \\ & \oplus \mathbb{C}(H_{\alpha_1} + H_{\alpha_2} + 2H_{\alpha_3} + 2H_{\alpha_4} + H_{\alpha_5}) \end{aligned}$$

$$d\delta_e(z) = -z_4$$

Because the coefficient of $-z_4$ is -1 we know that the orbit labeled 10001000 is not admissible under the adjoint group G. Now we would like to determine whether or not it is admissible under the simply connected Lie group with Lie algebra $E_{8(8)}$. To answer this question we need to carry our computation within the weight spaces.

In this case Let $\mathsf{g} = E_{8(8)}$, a real form of $\mathsf{g}_{\mathbb{C}} = E_8$, and $\Delta = \{\alpha_1, \alpha_2, \ldots, \alpha_8\}$ the Bourbaki simple roots of $\mathsf{g}_{\mathbb{C}}$ then $\Delta_k = \{\beta_1, \ldots, \beta_8\}$, where $\beta_1 = 2\alpha_1 + 2\alpha_2 + 3\alpha_3 + 4\alpha_4 + 3\alpha_5 + 2\alpha_6 + \alpha_7$, $\beta_2 = \alpha_8$, $\beta_3 = \alpha_7$, $\beta_4 = \alpha_6$, $\beta_5 = \alpha_5$, $\beta_6 = \alpha_4$, $\beta_7 = \alpha_2$ and $\beta_8 = \alpha_3$, is a set of simple roots for $\mathsf{k}_{\mathbb{C}} = s0_{16}(\mathbb{C})$.

The fundamental weights of $\mathsf{k}_{\mathbb{C}}$ are $\lambda_1 = (\beta_1 + \beta_2 + \beta_3 + \beta_4 + \beta_5 + \beta_6 + 1/2(\beta_7 + \beta_8)$, $\lambda_2 = \beta_1 + 2\beta_2 + 2\beta_3 + 2\beta_4 + 2\beta_5 + 2\beta_6 + \beta_7 + \beta_8$, $\lambda_3 = (\beta_1 + 2\beta_2 + 3\beta_3 + 3\beta_4 + 3\beta_5 + 3\beta_6 + 3/2(\beta_7 + \beta_8)$, $\lambda_4 = \beta_1 + 2\beta_2 + 3\beta_3 + 4\beta_4 + 4\beta_5 + 4\beta_6 + 2(\beta_7 + \beta_8)$, $\lambda_5 = \beta_1 + 2\beta_2 + 3\beta_3 + 4\beta_4 + 5\beta_5 + 5\beta_6 + 5/2(\beta_7 + \beta_8)$, $\lambda_6 = \beta_1 + 2\beta_2 + 3\beta_3 + 4\beta_4 + 4\beta_5 + 6\beta_6 + 3(\beta_7 + \beta_8)$, $\lambda_7 = 1/2(\beta_1 + 2\beta_2 + 3\beta_3 + 4\beta_4 + 5\beta_5 + 6\beta_6 + 4\beta_7 + 3\beta_8)$ and $\lambda_8 = 1/2(\beta_1 + 2\beta_2 + 3\beta_3 + 4\beta_4 + 5\beta_5 + 6\beta_6 + 3\beta_7 + 4\beta_8)$.

A computation by hand shows that the character $\chi = -\lambda_1$ is actually equal to $\sqrt{\delta_e}$. Hence, according to the above lemma from Ohta, the preceding orbit is *admissible* under the simply-connected cover but fails to be admissible under the adjoint group. We shall automate the process so that the functions work with the weight space instead of the root space as they currently do. At the moment, we are trying to find the best way to deal with this problem and shall report on our progress in the near future.

References

1. Djoković D. *Classification of nilpotent elements in simple exceptional real Lie algebras of inner type and description of their centralizers* J. Alg. **112** (1988) 503-524.
2. Kostant B., Rallis S. *Orbits and representations associated with symmetric spaces* Amer. J. Math. **93** (1971) 753-809.
3. Noël A. G. *Computing maximal tori using LiE and Mathematica*, Lectures Notes in Computer Science, Springer-Verlag. **2657** (2003) 728-736.
4. Noël A. G. *Classification of Admissible Nilpotent Orbits in Simple Exceptional Lie Algebras of Inner Type*, American Mathematical Society Journal of Representation Theory, **5** (2001) 455-493.
5. Ohta T. *Classification of admissible nilpotent orbits in the classical real Lie algebras*, J. of Algebra **136**, N0. 1 (1991) 290-333.
6. Schwartz J. *The determination of the admissible nilpotent orbits in real classical groups* ,Ph. D. Thesis M.I.T. Cambridge, MA (1987)
7. Sekiguchi J. *Remarks on real nilpotent orbits of a symmetric pair* J. Math. Soc. Japan **39**, No. 1 (1987), 127-138.
8. Vergne M. *Instantons et correspondence de Kostant-Sekiguchi*, R. Acad. Sci. Paris Sér. I Math. **320** (1995), 901-906.
9. Van Leeuwen, M. A. A., Cohen, A. M., and Lisser, B. *LiE: A package for Lie group computations*, Computer Algebra Nederland, Amsterdam, Netherlands (1992).
10. Vogan D. *Unitary representations of reductive groups* Annals of Mathematical Studies, Princeton University Press Study **118** (1987)
11. Vogan D. *Associated varieties and unipotent representations* Harmonic Analysis on Reductive Groups Birkhäuser, Boston-Basel-Berlin (199)1 315-388
12. Wolfram S., *The Mathematica Book* Wolfram media, Cambridge University Press (1998)

Efficient Implementation of Polynomial Arithmetic in a Multiple-Level Programming Environment

Xin Li and Marc Moreno Maza

University of Western Ontario, London N6A 5B7, Canada

Abstract. The purpose of this study is to investigate implementation techniques for polynomial arithmetic in a multiple-level programming environment. Indeed, certain polynomial data types and algorithms can further take advantage of the features of lower level languages, such as their specialized data structures or direct access to machine arithmetic. Whereas, other polynomial operations, like Gröbner basis over an arbitrary field, are suitable for generic programming in a high-level language.

We are interested in the integration of polynomial data type implementations realized at different language levels, such as LISP, C and ASSEMBLY. In particular, we consider situations for which code from different levels can be combined together within the same application in order to achieve high-performance.

We have developed implementation techniques in the multiple-level programming environment provided by the computer algebra system AXIOM. For a given algorithm realizing a polynomial operation, available at the user level, we combine the strengths of each language level and the features of a specific machine architecture. Our experimentations show that this allows us to improve performances of this operation in a significant manner.

1 Introduction

In a general purpose computer algebra system, generic code implementing univariate and multivariate polynomial over an arbitrary ring or field is a central feature. This is the case, for instance, in the computer algebra systems AXIOM [13], ALDOR [2], MAGMA [3] and SINGULAR [11]. On the other hand, the quest for high-performance in critical areas, such as modular algorithms for polynomial GCDs, leads naturally to develop low-level specialized code for univariate and multivariate polynomials over finite fields. Such code is available in the software systems mentioned above and others.

The works of [14,7,8] present efficient implementations of asymptotically fast algorithms for polynomial arithmetic in a high-level programming environment. In the reported experiments, for FFT-based univariate polynomial multiplication and the Half-GCD algorithm, the speed-up ratio between the pure low-level specialized code and the pure high-level generic code is in the range $2 \cdots 4$. (We refer to [10] and [16] respectively for these algorithms.) Therefore, high-performance

© Springer-Verlag Berlin Heidelberg 2006

in polynomial arithmetic can be achieved in a high-level programming environment, such as AXIOM and Aldor.

However, linkage to specialized code is a substantial bonus when low-level implementation can take advantage of special software or hardware features. The purpose of this study is to investigate implementation techniques for polynomial arithmetic in a multiple-level programming environment. We are interested in the integration of polynomial data type implementations realized at the different code levels. In particular, we consider situations for which code from different levels can be combined together within the same application in order to achieve high-performance.

As a driving example, we use the modular algorithm of van Hoeij and Monagan [12]. We recall its specifications. Let $\mathbb{K} = \mathbb{Q}(a_1, a_2, \ldots, a_e)$ be an algebraic number field over the field $\mathbb{Q}$ of the rational numbers. Let $f_1, f_2 \in \mathbb{K}[y]$ be univariate polynomials over $\mathbb{K}$. The algorithm of van Hoeij and Monagan computes $\gcd(f_1, f_2)$. To do so, for several prime numbers p, a tower of simple algebraic extensions $\mathbb{K}_p$ of the prime field $\mathbb{Z}/p\mathbb{Z}$ is used. Arithmetic operations in $\mathbb{K}_p$ are performed by means of operations on multivariate polynomials over $\mathbb{Z}/p\mathbb{Z}$, whereas the operations on the images of f_1, f_2 modulo p are performed in the univariate polynomial ring $\mathbb{K}_p[y]$. Therefore, several types of polynomials are used simultaneously in this algorithm. This is why it is a good candidate for our study.

We chose AXIOM as our implementation environment based on the following observations. AXIOM has a high-level programming language, called SPAD, which possesses all the essential features of object-oriented languages. Libraries written in SPAD implement a hierarchy of algebraic structures (groups, rings, fields, ...) and a hierarchy of algebraic domains ($\mathbb{Q}$, $\mathbb{A}$[x] for a given ring $\mathbb{A}$, ...).

The SPAD compiler translates SPAD code into Common Lisp, then invokes the underlying Lisp compiler to generate machine code. Today, GCL [17] (GNU Common Lisp) is the underlying Lisp of AXIOM [1]. The design of GCL makes use of the native C compiler for compiling to native machine code. In addition, GCL employs the GNU Multi-Precision library (GMP) [9] for its arbitrary precision number arithmetic. Therefore, AXIOM is an efficient multiple language level system. Moreover, the complete AXIOM system is open-source. Hence, we can implement our packages at any language level and even modify the AXIOM kernel. This allows us to take advantage of each language level's strength and access machine arithmetic directly when necessary. Therefore, we believe that AXIOM, with its different implementation levels, all in open source, provides an exceptional development environment among all computer algebra systems, for the purpose of our study.

In Sections 2, 3 and 4 and 5 we discuss the strength (in view of our objectives) of the SPAD, Lisp, C and Assembly level, respectively, together with our implementation techniques. In Section 6, we report on our experimentation.

Our results suggest that choosing adequate and optimized data structures is essential for polynomial arithmetic with high-performance. At the same time, implementing them at a suitable language level or levels may impact the running time of an algorithm implementation in AXIOM by a factor in the range $2 \cdots 4$ for large input data.

2 The SPAD Level

The AXIOM computer algebra system possesses an interactive mode for user interactions and the SPAD language for building library modules. The SPAD language has a two-level object model of *categories* and *domains* that is similar to *interfaces* and *classes* in Java. For instance, `Ring` is the AXIOM category for all rings (in the algebraic sense, see for instance [5]) with units and `Integer` is the AXIOM domain for the ring of integer numbers (see [13] for more details about the AXIOM hierarchies of categories and domains).

The user can define a new category and domain, which, then, can be added to the library modules. To do so, this new SPAD code must implement an AXIOM type constructor (or a package), to be compiled with the SPAD compiler. An AXIOM *type constructor* is simply a function which returns an AXIOM type, that is a category or a domain. For instance, `SparseUnivariatePolynomial`, abbreviated to `SUP`, is a type constructor, which takes an argument `R` of type `Ring` and returns an implementation of the ring of univariate polynomials over `R`, with a sparse representation (see below). Actually, `SUP(R)` implements `Ring` and other operations specific to polynomials (evaluation, differentiation, ...). The whole interface of `SUP(R)` is `UnivariatePolynomialCategory(R)` where `UnivariatePolynomialCategory` is a category constructor.

The SPAD language supports *conditional exports*. This permits to implement the following statement: if `R` has type `Field` then `SUP(R)` implements `EuclideanDomain`. SPAD supports also *conditional implementation*. For instance, if `R` has type `FiniteFieldCategory`, one can use formulas such *Little Fermat Theorem* to speed up some operations of `SUP(R)`, such as exponentiation. These features of the SPAD language are important for combining different data types and achieving high-performance.

Implementing a new domain constructor requires the programmer choosing a data structure for representing the objects defined by this domain. After a newly defined domain or category is compiled, it becomes an AXIOM data type which can be used just like any system provided data type.

In the light of these properties of the SPAD language, we describe briefly the polynomial type constructors that we use in this study. Please, see [13] and [14] for more details. Let `R` be an AXIOM `Ring` and `V` be an AXIOM `OrderedSet`.

`SUP` or `UP`. As mentioned above, the domain `SUP(R)` implements the ring of univariate polynomials with coefficients in `R`. More precisely, it satisfies the AXIOM category `UnivariatePolynomialCategory(R)`. The representation of these polynomials is *sparse*, that is, only non-zero terms are encoded.

`DUP`. The domain `DUP(R)` implements `UnivariatePolynomialCategory(R)` as well. The representation is dense: all terms, null or not, are encoded.

`NSMP`. The domain `NSMP(R,V)` implements the ring of multivariate polynomials with coefficients in `R` and variables in `V`. (To be precise, it implements the

AXIOM category `RecursivePolynomialCategory(R, V)`.) A non-constant polynomial f of `NSMP(R)`, with greatest variable v, is regarded as a univariate polynomial in v implemented as an element of `SUP(NSMP(R))`. Therefore, the representation is recursive and sparse.

`DRMP`. The domain `DRMP(R,V)` implements the same category as `NSMP(R,V)`. The representation is also recursive. However, it is based on `DUP` rather than `SUP`.

The constructors `SUP` and `NSMP` are provided by the AXIOM standard distribution, whereas `DUP` and `DRMP` were implemented by us at SPAD level. Our motivation is the implementation of modular methods based on the Euclidean Algorithm (or its variants) which tend to "densify" the computations, even if the input polynomials are sparse. This is the case, for instance with the algorithm of van Hoeij and Monagan, that we have implemented for this study.

3 The Lisp Level

The domain constructors `SUP`, `DUP`, `NSMP` and `DRMP` allow the user to construct polynomials over any AXIOM `Ring`. So we say that their code is generic. Observe that `R` has no influences on the representation scheme of the objects of `SUP(R)` or `DUP(R)`.

Ideally, one would like to use also *conditional data representations*. For instance, one could think of a domain `U(R)` implementing univariate polynomials over `R`, an AXIOM `Ring`, such that sparse polynomials (polynomials with frequent null terms) have a sparse representation and dense polynomials (polynomials with few null terms) have a dense representation. In addition, if `R` implements a prime field $\mathbb{Z}/p\mathbb{Z}$ for a machine word size prime p, one could require encode each dense polynomial of `U(R)` by an array of machine words (such that the slot of index i contains the coefficient in the term of degree i). But the code of this ideal type constructor `U` would be quite complicated and harder to optimize from the compilation point of view. Indeed, many tests would be needed for selecting the appropriate representation. Instead, we believe that specialized domain constructors (say, dense univariate polynomials over a prime field) to be called by a package (implementing, for instance, the algorithm of van Hoeij and Monagan) are a better choice. Moreover, polynomials over prime fields are such an importance case, for modular methods, that they deserve an independent treatment.

For these reasons, we have defined at the SPAD level a polynomial type constructor `MultivariateModularArithmetic`, abbreviated to `MMA`, taking a prime integer `p` and `V`, an `OrderedSet`, as arguments, such that `MMA(p,V)` implements the same operations as `DRMP(PF(p),V)`. (The domain `PF(p)` implements the prime field of characteristic `p`.) In fact, `MMA(p,V)` is just a wrapper for an implementation in Lisp. At this inner level of AXIOM, we have realized two implementations of `MMA(p,V)`: one for the case where `p` fits in a machine word and one for the case where it does not.

In these implementations, we used the *vector-based recursive dense* representation proposed by Richard J. Fateman [6]: a multivariate polynomial f is encoded by a number (to be precise, an integer modulo `p`) if f is constant and, otherwise, by a LISP vector storing the coefficients of f w.r.t its leading variable. At the SPAD level, such disjunction would be implemented by a union type bringing an extra indirection. This can be avoided at the LISP level, thanks to the so-called predicate functions which can judge the type of an object.

In addition, we can tell the LISP compiler to use machine integer arithmetic, if `p` fits in a machine word and, otherwise, to use the functions for big integer arithmetic from the GMP library [9].

Similarly for univariate polynomials, we have defined at the SPAD level a univariate type constructor `UnivariateModularArithmetic`, abbreviated to `UMA`, taking a prime integer `p` as argument and implementing the same operations as `DUP(PF(p))`. It is also a wrapper for two LISP implementations: one for Small `p`'s and for one big `p`'s. Each of these univariate polynomials is implemented using `fixnum-array` (a C-like array) in GCL. It is possible direct using C arrays to encode univariate polynomials over $\mathbb{Z}/p\mathbb{Z}$, but we prefer the LISP level garbage collection system which is more efficient and convenient.

All these specialized implementations at the LISP level yield significant speed up, as reported in Section 6.

4 The C Level

GCL is implemented in C language and uses the native optimizing C compiler to generate native machine code. This allows us to extend the functionalities of the LISP level of AXIOM with a given C function by either integrating this function into the GCL kernel, or by integrating it into a GCL library.

This interoperability between LISP and C has at least two benefits for achieving high-performance in the AXIOM environment. First, ASSEMBLY code (written for some efficiency critical operation, see Section 5) can be into the LISP level via C. Second, we can use existing C libraries providing efficient implementations of polynomial and integer arithmetic, such as GMP library [9] or NTL [15]. We illustrate these two benefits by an important example: the implementation of dense univariate polynomials over the prime field $\mathbb{Z}/p\mathbb{Z}$.

Recall that we have two implementations for these polynomials at the LISP level: one for small primes p (that fit in a machine word) and one for big primes p. They are both available at the SPAD level via the wrapper domain constructor `UMA`. For both the small and big prime case, we have integrated in the GCL kernel:

– classical multiplication, addition and Chinese remaindering algorithm written in ASSEMBLY,
– FFT-based multiplication written in C with ASSEMBLY sub-routines.

Moreover, in the big prime case, we distinguish between the double precision case and the multiple precision case. In the former one, we have developed our ASSEMBLY routines which improve the performance of the GMP multiple precision functions. (See [14] for details.) In the latter one, we rely on the ASSEMBLY routines of GMP. This distinction is motivated by the importance of the double precision case. For instance, most prime numbers used in the modular method of [4] are of that size.

5 The Assembly Code Level

As mentioned in Section 4, we make use of the Assembly functions of the GMP library in order to accomplish low-level operations with univariate polynomial over $\mathbb{Z}/p\mathbb{Z}[x]$. There are two other reasons for taking advantage of Assembly code in our AXIOM environment. We discussed them below.

5.1 Controlling Register Allocation

In a modern computer architecture, CPU registers seat at the top level of the memory hierarchy. Although optimizing compilers devote special efforts to make good use of the target machine's register set, this effort can be constrained by numerous factors, such as:

– difficulty to estimate the execution frequencies of each part of the program,
– difficulty to allocate or evict ambiguous values,
– difficulty to take advantage of some new hardware features on specific platforms.

Therefore, a high-performance oriented application requires programmers to better exploit the power of registers on a target machine. In fact, we have spent a great effort in this direction in our implementation.

First, we directly program the efficiency-critical parts in Assembly language in order to explicitly manipulate data in registers. For example, for dense univariate polynomials over $\mathbb{Z}/p\mathbb{Z}$, we write the classical multiplication algorithm in both C and Assembly language. The Assembly version is faster than the C version since we always try to keep frequently used variables in registers instead of a memory location. Although in C we can declare a variable to be of "register" type, this does not guarantee that the register is reserved for this variable. According to our benchmark results, our explicit register allocation method is always faster than the C compiler's optimization.

Beside the general purpose registers, we also can use MMX, XMM registers when they are available. Keeping the working set in registers will yield significant performance improvement comparing to keeping them in main memory.

5.2 Using Architecture Specific Features

Polynomial arithmetic in $\mathbb{Z}/p\mathbb{Z}[x]$ makes an intensive use of integer division. This integer operation has a dominant cost in crucial polynomial operations like the FFT-based multiplication in $\mathbb{Z}/p\mathbb{Z}[x]$. Therefore, improving the performance of integer division is one of the key issues in our implementation.

In the NTL library [15], the single precision modular reduction is implemented by means of floating point arithmetic, based on the following formula

$$a \equiv a \lfloor a \ 1/p \rfloor \ p$$

This formula can be implemented directly in C in two or three lines of code. However, one can further improve the performance by writing assembly code.

We have also implemented this trick in Assembly language for the Pentium IA-32 with SSE2 support. The SSE/SSE2 instruction sets use XMM registers. Each of these registers is of 128-bit and can be used to pack 2 double precision floating point numbers. In fact, SSE/SSE2 instructions can compute on multiple data packed in one register, in parallel.

Our implementation of the FFT-based polynomial multiplication over $\mathbb{Z}/p\mathbb{Z}$ uses this technique. It is faster than using FPU unit, as reported in Section 6.

6 Experimentation

6.1 Benchmarks for the Lisp Level Implementation

The goal of these benchmarks is to measure the performance improvements obtained by our specialized multivariate polynomial domain constructor `MMA` implemented at the Lisp level and described in Section 3. We are also curious about measuring the practical benefit of dense recursive polynomial domains in a situation (polynomial GCD computations over algebraic number fields) where AXIOM libraries traditionally use sparse recursive polynomials.

As announced in the introduction, our test algorithm is that of van Hoeij and Monagan [12]. Recall that, given an algebraic number field $\mathbb{K} = \mathbb{Q}(a_1, a_2, \ldots, a_e)$, this algorithm computes GCDs in $\mathbb{K}[y]$ by means of a small prime modular algorithm, leading to computations over a tower of simple algebraic extensions $\mathbb{K}_p$ of $\mathbb{Z}/p\mathbb{Z}$. Recall also that the algorithm involves two polynomial data types:

– a multivariate one for the elements of $\mathbb{K}$ and $\mathbb{K}_p$,
– a univariate one for the polynomials in $\mathbb{K}[y]$ and $\mathbb{K}_p[y]$.

Figure 1 shows the different combinations that we have used.

$\mathbb{Q}(a_1, a_2, \ldots, a_e)$	$\mathbb{K}[y]$
`NSMP` in SPAD	`SUP` in SPAD
`DMPR` in SPAD	`DUP` in SPAD
`MMA` in Lisp	`SUP` in SPAD
`MMA` in Lisp	`DUP` in SPAD

Note that:

– the first two combinations, that is `NSMP + SUP` (sparse polynomial domains) and `DMPR + DUP` (dense polynomial domains), involve only SPAD code,

– the other two combinations use MMA - our dense multivariate polynomials implemented at the Lisp level and SUP/DUP - univariate polynomials written at the SPAD level.

We would like to stress the following facts:

– the algorithms for addition, multiplication, division of `DRMP` and `MMA` are identical,
– none of the above polynomial types uses fast arithmetic, such as FFT-based or Karatsuba multiplication.

Remember also that:

– the SPAD constructors `NSMP`, `DMPR`, `UP`, and `DUP` are generic constructors, i.e. they work over any AXIOM ring,
– however, our dense multivariate polynomials implemented at the Lisp level (provided by the `MMA` constructor) only work over a prime field.

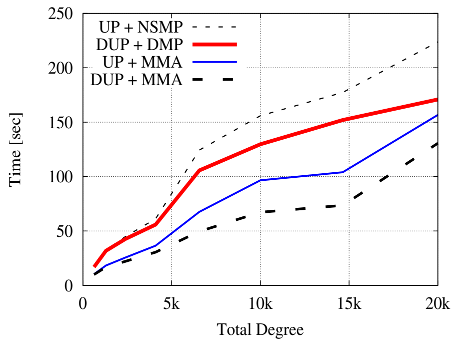
Therefore, we are comparing here is the performances of

– specialized code at the Lisp level versus generic code at the SPAD level,
– sparse representation versus dense representation.

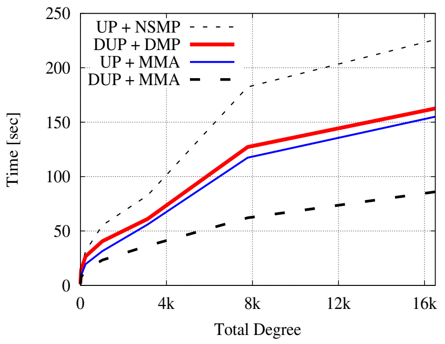
We have run the algorithm of van Hoeij and Monagan for different degrees of the extension $\mathbb{Q} \to \mathbb{K}$, different degrees of the input polynomials and different sizes for their coefficients. Figure 1 p. 20 shows our benchmark results. First, we fix the coefficient size bound to 5 and increase the total degree (degree of the extension plus maximum degree of an input polynomial). The charts (a), (b) and (c) correspond to towers of 3, 4 and 5 simple extensions respectively. Second, we fix the total degree to 2000 and increase the coefficient bound. The charts (d), (e) and (f) correspond to towers of 3, 4 and 5 simple extensions respectively. We observe the following facts.

`Charts (a), (b), (c).` For univariate polynomial data types, `DUP` outperforms `SUP` and, for the multivariate polynomial data types, `MMA` outperforms `DRMP`, which outperforms `NSMP`. For the largest degrees, the timing ratio between the best combination, `DUP + MMA`, and the worst one, `SUP + NSMP` is in the range $2 \cdots 3$.

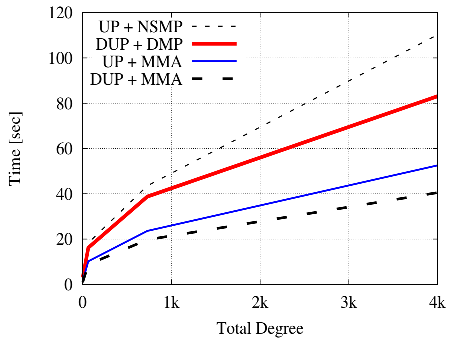
`Charts (d), (e), (f).` The best and the worst combinations are the same as above, however the timing ratio is in the range $3 \cdots 4$. Interestingly, the second best combination is `SUP + MMA` for small coefficients and `DUP + DRMP` for larger ones. This fact has probably the following double explanation. First, the `SUP` constructor relies on some fast routines which allows it to compete with the `DUP` constructor for small input data. Second, garbage collection of polynomials built with `DUP + DRMP` appears to be more efficient than for `SUP + MMA` polynomials, for large data.



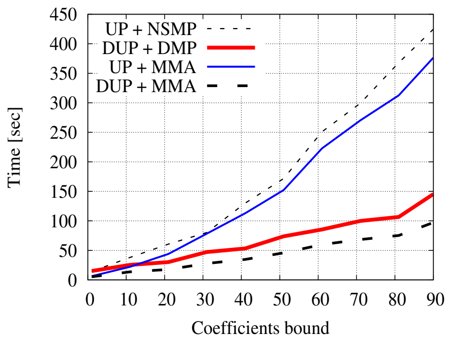
(a)



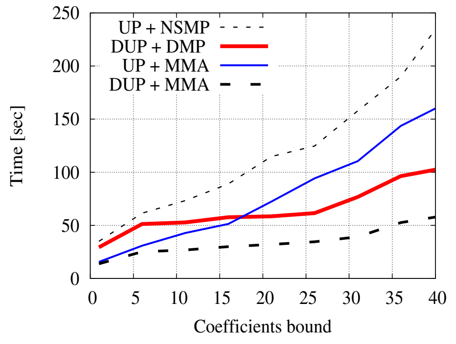
(b)



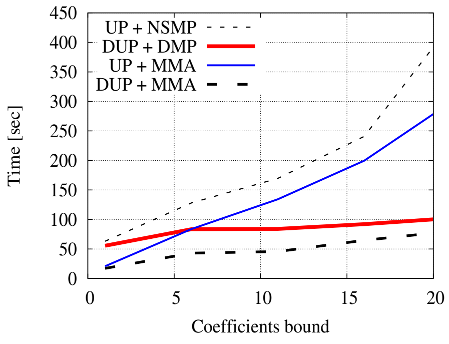
(c)



(d)



(e)



(f)

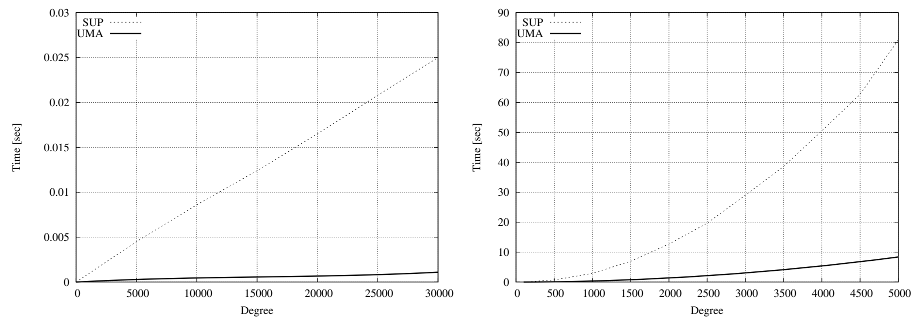
Fig. 1. In (a), (b) and (c) fixing the coefficient size bound, and increase the total degree of input polynomials. Conversely In (d), (d), and (f) fixing the total degree and increase the coefficient size bound.

6.2 Benchmark for the C and Assembly Level Implementation

The goal of these benchmarks is to measure the benefit provided by the C and Assembly levels to the SPAD level. Figure 2 shows benchmarks for addition and classical multiplication in $\mathbb{Z}/p\mathbb{Z}[x]$ for a 64-bit prime p, between

- the code of the `SUP` constructor (from the SPAD level), and
- the `UMA` constructor (written in Lisp with C and Assembly sub-routines).

This clearly illustrates the benefits of our implementation at Assembly level comparing to its SPAD level counterpart.



Assembly level polynomial addition. Assembly level polynomial multiplication.

Fig. 2.

Figure 3 illustrates the performance difference between

- the generic Assembly code using integer arithmetic and,
- the SSE2 Assembly code using floating point arithmetic.

The benchmark data of Figure 3 are obtained with our implementation of FFT-based univariate polynomial multiplication over $\mathbb{Z}/p\mathbb{Z}[x]$ for a 27-bit Fourier prime p. This benchmark shows that our SSE2-based implementation is significantly faster than our generic Assembly version.

6.3 Benchmark of the Multi-level Implementation

The goal of this benchmark is to compare an AXIOM function, involving code at all levels, SPAD, Lisp, C and Assembly, versus its counterpart in a similar computer algebra system, namely MAGMA. We choose multivariate polynomial multiplication based on Kronecker substitution (which for us is implemented at SPAD and Lisp levels, since it is independent from machine arithmetic) and FFT-based univariate multiplication (which for us is implemented at C and Assembly level, as shown in previous benchmark). Our implementation outperforms MAGMA's counterpart as shown in Figures 4.

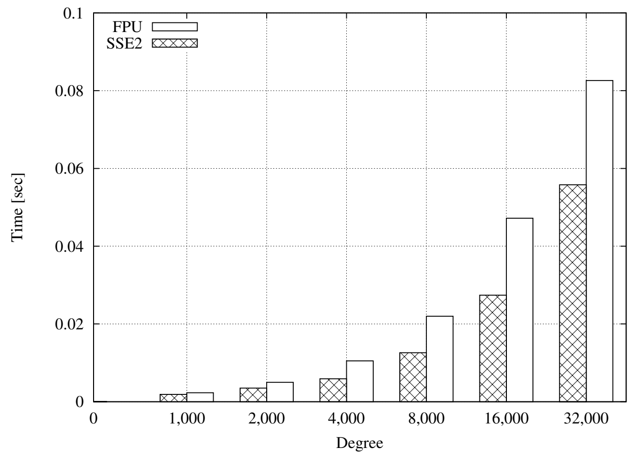


Fig. 3. Generic assembly vs. SSE2 version assembly of FFT-based univariate polynomial multiplication over $\mathbb{Z}/p\mathbb{Z}$

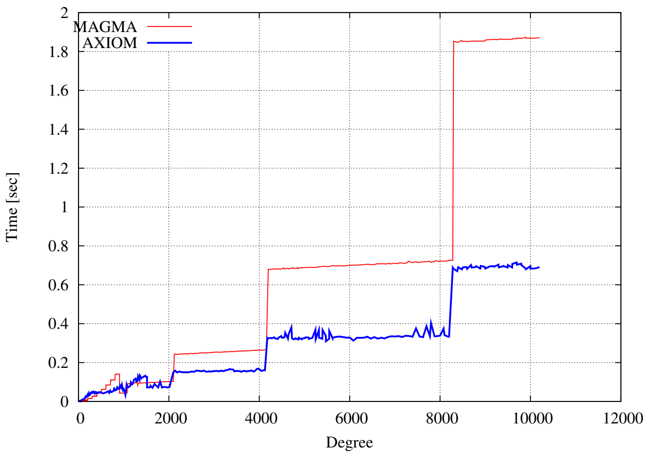


Fig. 4. Bivariate multiplication modulo a 64-bit prime

7 Conclusion

We have investigated implementation techniques for polynomial arithmetic in the multiple-level programming environment of the AXIOM computer algebra system. Our benchmark results show that careful integration of data structures and code from different levels can improve the performances in a significant manner (a ratio of 2...4 speed up reported in 6). The integration process requires deep understanding of polynomial arithmetic, machine arithmetic and compiler optimization techniques. However, we believe that it should be implemented in a transparent way for the end-user.

References

1. *The* AXIOM *developers web site.* `http://page.axiom-developer.org`.
2. aldor.org. *Aldor 1.0.2.* University of Western Ontario, Canada, 2004. http://www.aldor.org.
3. The Computational Algebra Group in the School of Mathematics and Statistics at the University of Sydney. *The MAGMA Computational Algebra System for Algebra, Number Theory and Geometry.* `http://magma.maths.usyd.edu.au/magma/`.
4. X. Dahan, M. Moreno Maza, É. Schost, W. Wu, and Y. Xie. Lifting techniques for triangular decompositions. In *ISSAC'05*, pages 108–115. ACM Press, 2005.
5. David S. Dummit and Richard M. Foote. *Abstract algebra.* Simon and Schuster, 1993.
6. R. J. Fateman. Vector-based polynomial recursive representation arithmetic. `http://www.norvig.com/ltd/test/poly.dylan`, 1999.
7. A. Filatei. Implementation of fast polynomial arithmetic in Aldor, 2006. University of Western Ontario.
8. A. Filatei, X. Li, M. Moreno Maza, and É Schost. Implementation techniques for fast polynomial arithmetic in a high-level programming environment. In *Proc. ISSAC'06*. ACM Press, 2006.
9. Free Software Foundation. *GNU Multiple Precision Arithmetic Library.* `http://swox.com/gmp/`.
10. J. von zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, 1999.
11. G.-M. Greuel, G. Pfister, and H. Schönemann. SINGULAR 3.0. A Computer Algebra System for Polynomial Computations, Centre for Computer Algebra, University of Kaiserslautern, 2005. `http://www.singular.uni-kl.de`.
12. M. van Hoeij and M. Monagan. A modular gcd algorithm over number fields presented with multiple extensions. In Teo Mora, editor, *Proc. ISSAC 2002*, pages 109–116. ACM Press, July 2002.
13. R. D. Jenks and R. S. Sutor. *AXIOM, The Scientific Computation System.* Springer-Verlag, 1992. AXIOM is a trade mark of NAG Ltd, Oxford UK.
14. X. Li. Efficient management of symbolic computations with polynomials, 2005. University of Western Ontario.
15. V. Shoup. *The Number Theory Library.* http://www.shoup.net/ntl.
16. C.K. Yap. *Fundamental Problems in Algorithmic Algebra.* Princeton University Press, 1993.
17. Taiichi Yuasa, Masami Hagiya, and William F. Schelter. *GNU Common Lisp.* `http://www.gnu.org/software/gcl`.

Development of a Maple Macro Package Suitable for Drawing Fine TeX-Pictures

Masayoshi Sekiguchi, Satoshi Yamashita, and Setsuo Takato

Kisarazu National College of Technology,
Kiyomidai-Higashi 2-11-1, Chiba 292-0041, Japan
masa@kisarazu.ac.jp

Abstract. The authors have developed a Maple macro package: *KETpic* which generates LaTeX source codes for clear drawings. KETpic enables us to draw every kind of complicated figure easily. Users are simply required to command Maple, with KETpic loaded, to plot graphs, to create LaTeX source codes by KETpic commands, and to embed them into LaTeX source files. Desired figures are obtained by the usual LaTeX compilation. Two versions of KETpic for Macintosh and Windows are provided, both of which are available for Linux users. Figures finally obtained, either on a PC display or on printed matter, are clear and possess the highest accuracy. KETpic does not require an expensive printer. Carefully prepared figures are significantly advantageous for mathematical education because they facilitate students' understanding of difficult mathematical notions. In this paper, we describe the advantages of KETpic with typical examples.

1 Problems We Have Encountered

In 2002, a mathematical textbook series revision committee[9,10,11,12,13,14], including one of the authors (T) as the chairman, had decided to employ LaTeX 2_ε equipped with *emath*[6] and *WinTpic*[4] for preparing their manuscripts.

To reduce various costs, LaTeX 2_ε was chosen. WinTpic is a popular software in Japan, which generates Tpic special codes. It supports GUI and is easy to handle, but has the following defects:

1. it runs only on Windows (95 and later),
2. it cannot uniformly distribute gaps in a dashed curve,
3. it cannot draw natural looking free closed curves,
4. it does not permit users to input a complicated function such as defined by a truncated series or integral calculus.

A LaTeX macro package, emath, can support the picture environment and is well-designed for preparing materials for mathematics classes, and its use has spread among teachers and publishers in Japan. Although emath is based on epic.sty, eepic.sty and Tpic specials, it has a problem:

5. Hatching fails to fill non-convex areas, or overflows its boundaries.

© Springer-Verlag Berlin Heidelberg 2006

Accordingly, WinTpic and emath are inadequate for editing textbooks. The committee needed to find an alternative.

In addition, it is necessary for us to prepare graphical materials for classes because mathematically accurate pictures help students effectively follow various discussions in every course topic. Preparing such materials requires effort, time and money. Therefore, we usually search for easier and cheaper ways of personal DTP. Fortunately, Maple and LaTeX provide us a suitable way, especially if they are combined. Maple can calculate values of any complicated function and write them in a file. LaTeX supports the picture environment where users can draw line segments, circles, and points with mathematical expressions or symbols. Although plain Maple creates LaTeX files, it does not optimize its output so as to be suitable for drawing graphs. Optimization of Maple outputs remained as a problem.

2 Solution: KETpic

Tpic is a graphic extension of TeX which uses a simpler set of embedded commands. Currently, Tpic is popular world-wide and only requires some special dvi drivers. When Tpic special codes are generated by some computer algebra software, they can be expected to produce very accurate graphs of any function.

A famous computer algebra system: Maple enables us to manipulate raw data of plots and strings. This feature is sufficient to generate Tpic special codes. Thus, we have developed a macro package of Maple (version V and later): *KETpic* (abbreviation of Kisarazu Educational Tpic). KETpic as well as Maple covers the major platforms. We have already released two versions of KETpic for Mac and Windows which generate codes with different end-of-line characters according to each platform. They are both available for Linux users(Redhat 2.4.5 and later)[8].

Table 1. Tpic special commands generated by KETpic

command	functions
`pn`	sets line width
`pa`	adds x, y to the path
`fp` (or `ip`)	draws a line on the current path
`ar` (or `ia`)	draws a circle or arc
`sh`	shades a closed object

2.1 Design Concept and Advantages of KETpic

Pursuing operability and economy for KETpic, we designed KETpic so as to carry out calculations on Maple as much as possible, and not to depend on the PC and the peripherals (such as a PostScript printer etc). KETpic generates only the primitive commands of Tpic specials listed in Table 1, except for the other commands in Table 2.

The commands in Table 2 assume the heavier processing ability of PostScript. Therefore, Tpic special codes containing such commands require a PostScript printer for clear printing. WinTpic, mentioned above, depends largely on the performance of printers because it generates all kinds of Tpic special commands. If no PostScript printer is available, then dvi-outputs can be directly printed by an inkjet printer which is the most popular method. Fig. 1 displays such a result, a graph of $y = \sin(1/x)$ printed by a non-PostScript printer. We can see that the gaps are not uniformly distributed over the curve. However, if printed by a PostScript printer, no such problem would result.

Table 2. Tpic special commands which KETpic does not generate

command	functions
`da`	similar to `fp` but with a dashed line
`dt`	similar to `fp` but with a dotted line
`sp`	draws a spline curve on the current path

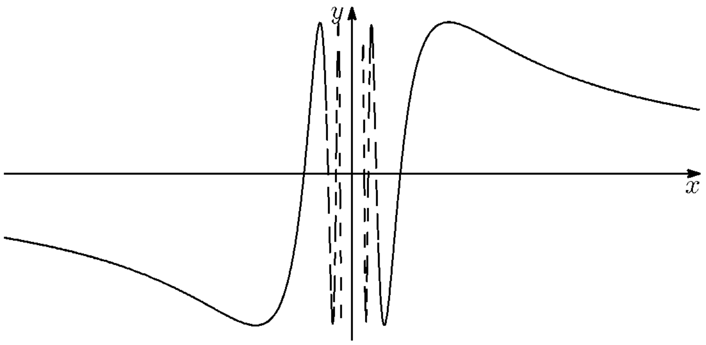


Fig. 1. A non-PostScript plotting of $y = \sin(1/x)$ of a LaTeX picture generated by WinTpic

Dashed/dotted lines/curves or spline curves can be created by Maple without `da`, `dt` and `sp`. Hereafter, we illustrate how KETpic draws dashed curves, and omit a description of how to draw dotted curves and spline curves. KETpic draws a dashed curve using the following procedures.

1. Calculate the length of curves,
2. Pick up points from the curve periodically,
3. Connect them alternately.

Plot data are written in a file as Tpic special codes. The codes generated by KETpic only plot points, line segments, and arcs. Dashed curves are drawn in this way. We present a picture of the same function by KETpic in Fig. 2. Even if we print it on an ordinary printer, we can see that the gaps are uniformly distributed over the curve. Accordingly, producing a clear picture by KETpic does not need expensive hardware.

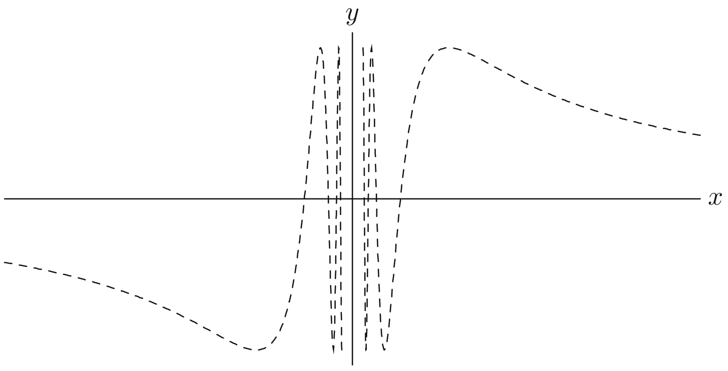


Fig. 2. Example for `dashline`. Gaps are uniformly distributed over the curve.

Although KETpic does not support GUI and permits only line inputting, it is superior in accuracy because it can treat correct (or approximate to as high an order as required) functions and numerals. The very fine figures, whether printed on printed matter or displayed on a monitor, possess the highest accuracy of any software package as far as we know.

2.2 Flow of Preparing Figures by KETpic

It is very easy for anyone to prepare LaTeX pictures by KETpic. The command reference, as well as KETpic packages, is currently downloadable from the web site[8]. Here, we briefly describe the flow of producing the picture displayed in Fig. 2: a graph of the function $y = \sin(1/x)$ in a dashed curve. We assume that readers have already installed Maple (version V or later) and LaTeX in their PCs.

1. First Step (Differences among platforms exist in symbols indicating folders or directories)
   Load some library packages and KETpic.

   ```
   > with(plots):
   > with(plottools):               # optional
   ```

   In the case of Macintosh,

   ```
   > folder:=`Macintosh_HD:work:`:
   > read cat(folder,`ketpic.m`):   # Load KETpic For Mac
   ```

   In the case of Windows,

   ```
   > folder:=`C:\\My Documents/`:
   > read cat(folder,`ketpicw.m`):  # Load KETpic For Windows
   ```

In the case of Linux,

```
> folder:=`/home/USERSNAME/`:
```

Linux users can utilize both `ketpic.m` and `ketpicw.m`. After loading KETpic, all commands can be used just as usual Maple commands.

2. Second Step (There are no differences among platforms after this step)
   Define a drawing window and axes, and command Maple to calculate the values of the function $\sin(1/x)$.

```
> setwindow(-2..2,-1.1..1.1):   # window sizes are declared
> setax("","","","","",""," ",""): # names of axes etc as default
> g1:=plot(sin(1/x),x=XMIN..XMAX,numpoints=200):
> frmdisp(g1);                  # optional: for check of data
```

3. Third Step
   Open an appropriate LaTeX file and write data into it.

```
> openfile(cat(folder,`f1.tex`)):# Open a file at a folder
> openpicture("1cm"):            # Define a unit length
> dashline(g1,0.5,0.5):          # last two parameters control
> closepicture():                #    the distribution of gaps
> closefile():                   # Close the file
```

Finally, insert the LaTeX file (here in the example above, `f1.tex`) into the user's source file as follows. This template LaTeX source is provided at the same site.

```
\documentclass[a4]{article}
\newlength{\Width}   % Define length commands of KETpic
\newlength{\Height}  %
\newlength{\Depth}   %
\begin{document}
\input{f1}           % Insert a LaTeX file
\end{document}
```

After the usual compilation of the LaTeX file, we can obtain Fig. 2.

3 Other Examples

In this section, we present other examples showing various advantages of KETpic.

3.1 Dashed Curves

Two drawing commands for dashed lines: `dashline` and `invdashline` are defined in KETpic. As shown in Fig. 3, using `invdashline`, users can draw dashed lines/curves which stop short of reaching other lines or curves. Usual dashed lines/curves which do reach other lines/curves are drawn by using `dashline`. The following codes generate Fig. 3.

```
> setwindow(-0.5..5.5,-0.5..2.5):
> pA:=[2,2]: pB:=[5,2]: pC:=[0,2]: pH:=[2,0]: pL:=[5,0]:
> g1:=plot([pA,pB]):    g2:=plot([pA,pH]):
> g3:=plot([pB,pL]):    g4:=plot([pA,pC]):
> openfile(‘f3.tex‘):   openpicture("1cm"):
> invdashline(g2,g4):   dashline(g3):         drwline(g1):
> letter([4.5,1.5],"w","{\\tt dasheline}"):
> letter([2.6,2.5],"e","{\\tt invdasheline}"):
> arrowline([2.5,2.5],[1,2]):
> arrowline([2.6,2.5],[2,1]):
> arrowline([4.5,1.5],[5,1]):
> closepicture():       closefile():
```

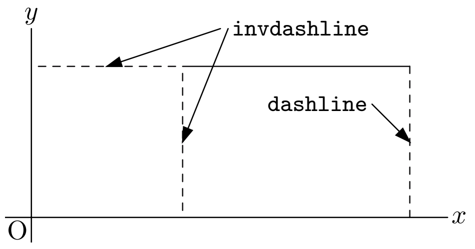


Fig. 3. An example illustrating the differences between `dashline` and `invdashline`. With `dashline`, the dash connects to the original line whereas with `invdashline`, a gap is created between the original line and the dashed line.

3.2 The Dirac Delta Function

The Dirac delta function $\delta(t)$ can be thought as a limit of a function $\varphi_\varepsilon(t)$ as $\varepsilon \to 0$, where

$$\varphi_\varepsilon(t) = \begin{cases} \dfrac{1}{\varepsilon} & (0 < t \le \varepsilon), \\ 0 & (t > \varepsilon). \end{cases}$$

Laplace transform of the sequence of functions $\varphi_\varepsilon(t)$

$$\mathcal{L}[\varphi_\varepsilon(t)] = \begin{cases} \dfrac{1 - e^{-\varepsilon s}}{\varepsilon s} & (s \neq 0), \\ 1 & (s = 0) \end{cases}$$

also converges to $\mathcal{L}[\delta(t)]$: Laplace transform of $\delta(t)$. Using this fact, we try to make an intuitive introduction of the Dirac delta function as an inverse Laplace transform of 1. We expect it to be effective for beginners in this course topic.

3.3 Polar Coordinate System

Many readers are familiar with the parallel projection of a hemisphere describing polar coordinates (see Fig. 5). However, this is not easy to draw because of the need for hidden line elimination. This picture has been inserted in a textbook[12].

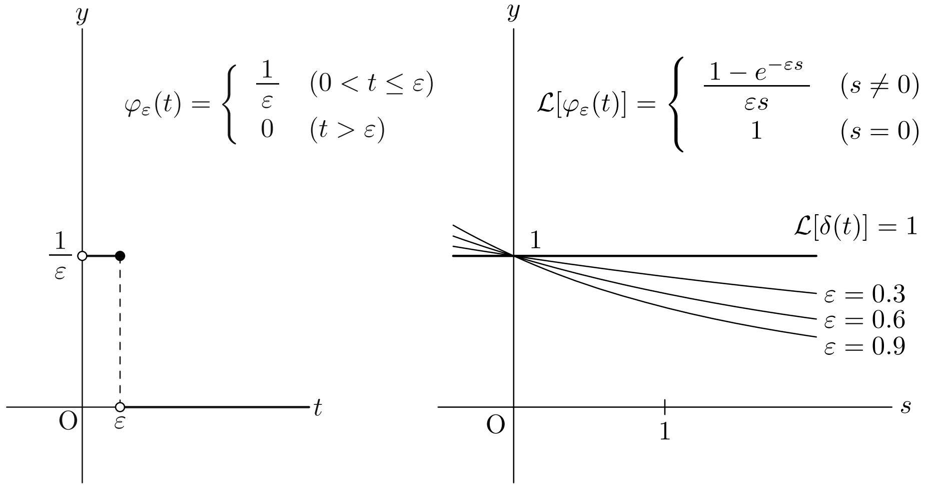


Fig. 4. A way of teaching the Dirac delta function as an inverse Laplace transform of unity

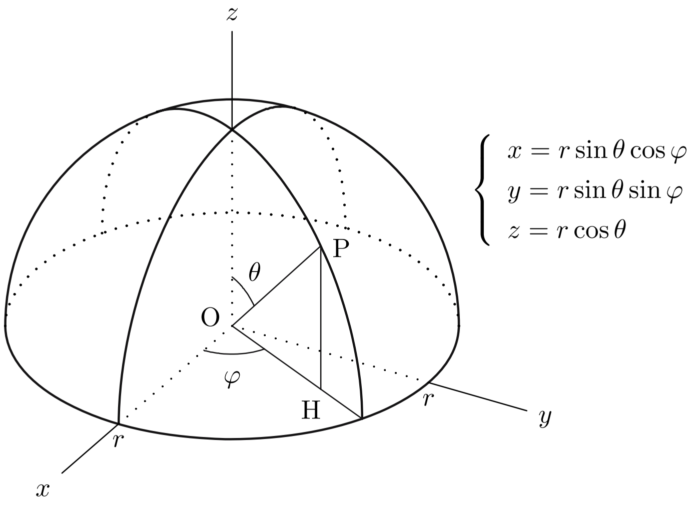

z
$\begin{cases} x = r\sin\theta\cos\varphi \\ y = r\sin\theta\sin\varphi \\ z = r\cos\theta \end{cases}$
P
θ
O
φ
H
r
y
r
x

Fig. 5. Clear description of the polar coordinate system

3.4 Nonlinear Differential Equation

The Lotka-Volterra equation, an ecological model, is one typical nonlinear differential equation, which is defined by

$$\begin{cases} \dfrac{dx}{dt} = x - xy, \\ \dfrac{dy}{dt} = -y + xy. \end{cases}$$

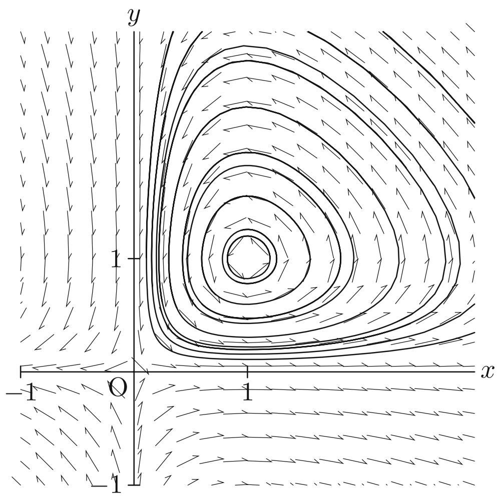


Fig. 6. Solution curves of the Lotka-Volterra equation and its vector field

Fig. 6 is a common picture to visualize the vector field, but is not easy to produce. For this picture, we used Maple commands `dsolve/numeric` which solves differential equations numerically and `dfieldplot` which plots the vector field.

3.5 Hatching Non-convex Regions

When we indicate a region where a function is defined, or a region satisfying inequalities, we hatch that region. KETpic command `hatch` realizes area-hatching. Fig. 7 shows the union of the asteroid $x^{2/3} + y^{2/3} = 1$ and its rotation by $\pi/4$.

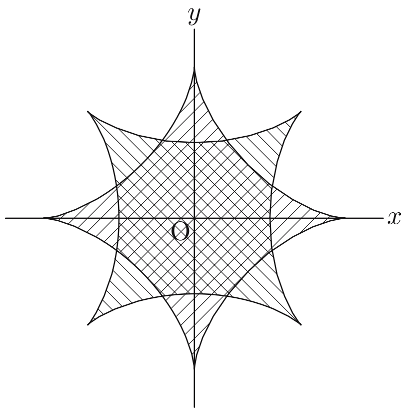


Fig. 7. Hatching inside a non-convex area

We will show how to hatch asteroids. First, define a boundary,

```
> g1:=plot([2*cos(t)^3,2*sin(t)^3,t=0..2*Pi]):
```

Second, hatch inside the boundary,

```
> gs1:=hatchdata([0,0],[g1],45):
```

KETpic successfully fills the non-convex region `g1`$\cup$`gs1`. Rotation of the asteroid: `g2` is easily obtained by the following commands.

```
> g2:=rotate(g1,Pi/4):
> gs2:=hatchdata([0,0],[g2],-45):
```

3.6 Cycloid

For the introduction of a cycloid, we wish to illustrate how to draw a cycloid. Explanation using animation through a video projector will be appealing to students. However, we think that the alternative way shown in Fig. 8 is also appealing.

First, we put a circle with its radius 1 and a point at the origin. We define `z3` as the union of the circle and the point. Next, we rotate it by `dt` and translate it by `dt`. Repeating this process, we obtain the picture of Fig. 8.

```
> z1:=point([0,0]):
> z2:=circle([0,1],1):
> z3:=frmdisp(z1,z2):              # z3 is the union of z1 and z2
> z4:={}: N:=20: dt:=2*Pi/N:
> for i from 1 to N do
>   tmp := rotate(z3,-i*dt,[0,1]):
>   tmp2:= translate(tmp,i*dt,0):
>   z4  := z4 union {tmp2}
> od:
> openfile(`f8.tex`):    openpicture("1cm"):
> dottedline(z3,op(z4)):
> closepicture():        closefile():
```

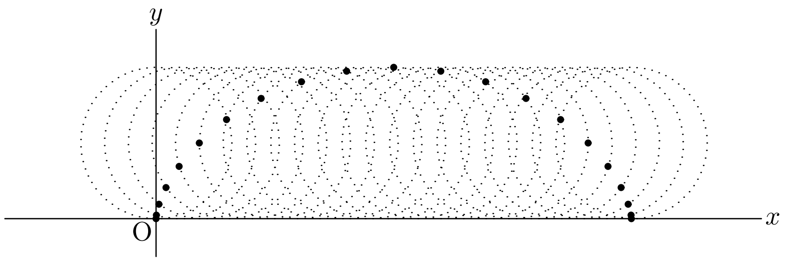


Fig. 8. How to draw a cycloid

3.7 Perspective Projection

Perspective projection is currently available with Maple. However, we cannot change slightly the view line and the distance between the object and our eyes. In order to realize a fine-tuning of output, we developed the command `projpers` by which we can change the parameters. Two results by `projpers` are shown in Fig. 9. These form a stereogram of the intersection of two cylinders. This stereogram realizes a 3D image in our brain by a cross-eyed view, that is, to see the left picture with the right eye and the right picture with the left eye.

The final example implies further possibilities of KETpic as an editing tool of academic papers as well as a good material for understanding strange attractors.

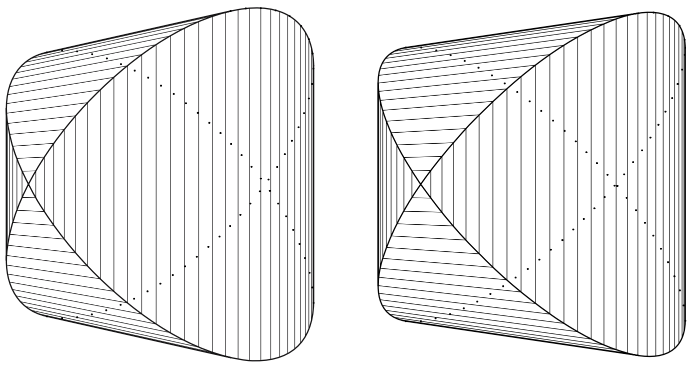


Fig. 9. A stereogram of the intersection of two cylinders realizes a 3D image by a cross-eyed view

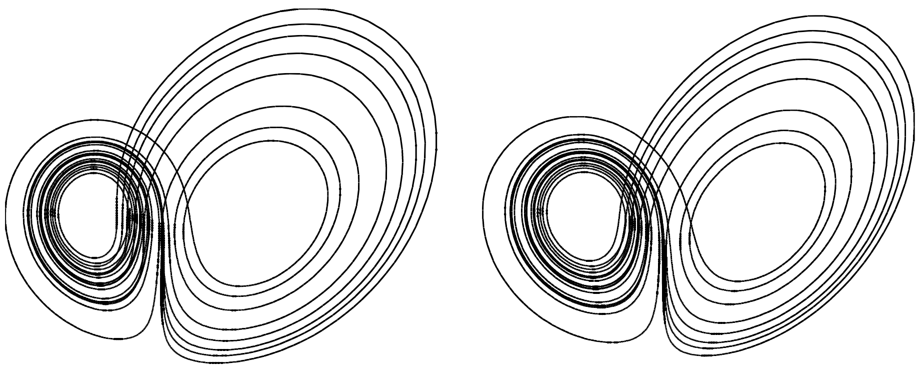


Fig. 10. A stereogram of the Lorentz Attractor realizes a 3D image by a cross-eyed view

Displayed in Fig. 10, the Lorentz Attractor, is a solution curve of

$$\frac{dx}{dt} = 10(y - x),\ \frac{dy}{dt} = 28x - y - xz,\ \frac{dz}{dt} = -\frac{8}{3}z + xy.$$

In order to obtain this picture, we used one of Maple commands, `DEplot3d`, which solves three-dimensional differential equations and plots them.

4 Summary

We have developed a Maple macro package: KETpic. KETpic takes full advantage of Maple, generates Tpic special codes, and enables us to draw clear figures with the highest accuracy. Other advantages of KETpic are operability, runability on major platforms, and portability.

KETpic and several examples are downloadable from the web site[8] where the manual is also available.

Not only book editors but also teachers of mathematics (sometimes physics as well) can utilize KETpic whenever they need an easy way to make supplementary materials to explain mathematical notions in their classes. Therefore, KETpic is one of the best choices for drawing graphs in every kind of mathematical material.

References

1. Char, B.W. and Geddes, K.O., et al: "Maple V Library Reference Manual", (1991), Springer-Verlag.
2. Goossens, M., Rahtz, S. and Mittelbach, F.: "The LaTeX Graphics Companion", (1997), Addison-Wesley.
3. Heal, K.M., Hansen, M.L. and Rickard, K.M.: "Maple V Learning Guide", (1996), Springer-Verlag.
4. Horii, M.: `http://www.vector.co.jp/soft/win95/writing/se061886.html` (in Japanese).
5. Mittelbach, F. and Goossens, M., et al: "The LaTeX Companion", (2004), Addison-Wesley.
6. Okuma, K.: `http://homepage3.nifty.com/emath/` (in Japanese).
7. Okumura, H.: `http://oku.edu.mie-u.ac.jp/~okumura/texwiki/` (in Japanese).
8. Sekiguchi, M.: `http://www.kisarazu.ac.jp/~masa/math/`
9. Takato, S., Saito, H., et al:"Fundamental Mathematics", (2003), Dainihon Tosho (in Japanese).
10. Takato, S., Saito, H., et al:"Differential and Integral I", (2003), Dainihon Tosho (in Japanese).
11. Takato, S., Saito, H., et al:"Linear Algebra", (2004), Dainihon Tosho (in Japanese).
12. Takato, S., Saito, H., et al:"Differential and Integral II", (2004), Dainihon Tosho (in Japanese).
13. Takato, S., Saito, H., et al:"Probability and Statistics", (2005), Dainihon Tosho (in Japanese).
14. Takato, S., Saito, H., et al:"Applied Mathematics", (2005), Dainihon Tosho (in Japanese).

Matlab-Based Problem-Solving Environment for Geometric Processing of Surfaces

A. Gálvez and A. Iglesias

Department of Applied Mathematics and Computational Sciences,
University of Cantabria, Avda. de los Castros,
s/n, E-39005, Santander, Spain
akemi.galvez@postgrado.unican.es, iglesias@unican.es
http://personales.unican.es/iglesias

Abstract. In this paper a new problem-solving environment (PSE) for geometric processing of surfaces is introduced. The PSE has been designed to be responsive to the needs of our collaboration with an industrial partner, the Spanish company CANDEMAT S.A., devoted to build moulds and dies for the automotive industry. The PSE has been implemented in *Matlab* and is aimed to support the full range of activities carried out by our partner in the field of geometric processing of surfaces for the automotive industry. Firstly, the paper describes the architecture of the system and some implementation details. Then, some examples of its application to critical problems in the automotive industry - such as the computation of the intersection curves of surfaces, the generation of tool-path trajectories for NC machining and the visualization of geometric entities stored in industrial files of several formats - are briefly described. The PSE has shown to provide our partner with accurate, reliable solutions to these and other problems and to serve as a communication channel for exchange of geometrical data as well as a platform for trial and research support.

1 Introduction

The work of this paper has been motivated by a collaboration agreement between our research group at the University of Cantabria and an industrial partner, the Spanish company CANDEMAT S.A., devoted to the automotive and aerospace industries. CANDEMAT is primarily focused on building moulds and dies of mechanical parts of car bodies, a domain that requires typically a number of advanced geometric processing techniques - and very often novel approaches to the bulk of problems arising in this field. Automotive industry is a sector in which the growing competition asks for continuous reduction of development and innovation cycles while the demands on quality, safety and comfort are increasing rapidly. Although the extraordinary advances in areas such as CAD/CAM and many others have contributed to comply with those requirements so far, recent moves in global markets and rising competition have evidenced the need for a change in way cars are built. A major issue is the dissemination of efforts among

© Springer-Verlag Berlin Heidelberg 2006

the great variety of suppliers and external engineering partners, each having its own methods, hardware, software, data formats, communication protocols and so on. This makes integration of all this technology much harder and eventually causes a mismatch in comparison to what the industry would expect if a smooth integration of the underlying data and processes would have been achieved [10].

On the other hand, the current tendency in carmakers' world is the integration of a crowd of many small manufacturers into a few large corporations. This ongoing transition also requires the integration of technical staff and computer hardware and software at full extent. One way to overcome these challenges - arguably the best one - is the intensive use of complex problem-solving environments. In fact, it has recently been reported that the use of problem-solving environments (PSE) is currently a common practice for product development in automotive and aerospace industries [8].

In our case, this means to create a scientific environment supporting the full range of computational tasks - from problem formulation and algorithm selection to numerical simulation and solution visualization - and being able to provide our industrial partner with reliable solutions to its problems. The ultimate goal of this process is to establish a strategic position in the market for our partner based on some kind of technological advantage and its ability to yield deliverables suitable for short and medium-term production. The PSE described in this paper is the computer tool created to meet these goals.

Tradicionally, scientific computation has been performed by using standard programming languages, such as Pascal or C. More recently, the use of interactive, user-friendly interfaces has become a standard requirement as well. Programming languages for graphical user interfaces (GUIs) and/or object-oriented programming, such as Delphi, Visual Basic, Visual C/C++, Python, Perl and others, have been intensively used for these purposes. Although these languages interact quite well with the numerical layer provided by standard programming languages, they cannot be embedded easily into a common programming environment. In addition, their versions differ depending on the platform and operating system considered. Fortunately, the last generation of computer algebra systems (CAS) overcomes these limitations. In particular, *Matlab* [14] is one of the first preferences because of its appealing features for numerical, graphical and user interface tasks. This assessment has been supported by several announcements about the use of *Matlab* by automotive corporations (like, for instance, by Daimler-Chrysler and Motor Ford Company in [18]). *Matlab* is a numerical-oriented package that also includes a number of symbolic computation features. In fact, its most recent versions incorporate a subset of the *Maple* kernel totally integrated into the numerical layer. Even although its symbolic capabilities are not as powerful as their counterpart in *Maple*, they are faster and very suitable for many purposes. In addition, the graphical *Matlab* capabilities (based on Open GL graphical routines) exceeds those commonly available in many other CAS.

In this paper we focus on problems related to the geometric processing of surfaces. *Geometric processing* is defined as the calculation of geometric properties of curves, surfaces and solids [2]. In its most comprehensive meaning, this term includes all algorithms that are applied to already constructed geometric entities [5]. Geometric processing is a key tool in the automotive industry, since many processes and operations in manufacturing rely heavily on the geometric properties of curves and surfaces. This work introduces a new problem-solving environment (PSE) for geometric processing of surfaces. The PSE has been implemented in *Matlab* and designed to be responsive to the needs of our collaboration with the company CANDEMAT. The next sections will describe its most interesting features and some examples of its application.

The structure of this paper is as follows: in Section 2 we describe the architecture of the system and some implementation details. Then, some examples of its application to critical problems in the automotive industry - such as the computation of the intersection curves of surfaces, the generation of tool-path trajectories for NC machining and the visualization of geometric entities stored in industrial files of several formats - are briefly described in Section 3. Conclusions and further remarks in Section 4 close the paper.

2 Our Matlab-Based Problem-Solving Environment

This section analyzes the architecture of the *Matlab*-based Problem-Solving Environment introduced in this paper. In addition, some implementation details are briefly described in this section. Lastly, we discuss the possibility of compiling and linking the resulting files for standalone applications.

2.1 System Architecture

Matlab's development environment consists of a command line displayed in one or several windows. The main window - the *Desktop* - has pulldown menus from an overhead menu bar and is subdivided into several resizable windows. These windows contain a command line, a command stack, and a directory tree. Programming code is written as either an M-file, consisting of *Matlab* commands, or a compiled version of an M-file created for a platform-specific target. Creating and testing most computational codes thus involve two separate operations: creating M-file text, and then executing it to test and debug.

An important factor to choose *Matlab* concerns the efficiency. Since *Matlab* is based on C/Java, it runs faster than other analyzed symbolic and numerical programs. Moreover, its basic element is an array that does not require dimensioning, so it takes less time to be computed. Although the symbolic operations are not suitable for these based-on-array advanced features, numerical and graphical operations (which are the key ingredient of many commands for scientific visualization) are greatly improved by using those features. *Matlab* also provides many connectivity tools for interfacing to external routines written in other languages, including Java, C, and Fortran, as well as to a variety of data objects and servers. Using most of the above-mentioned features, we have created a PSE comprised of the following components:

1. *a graphical user interface (GUI)*: *Matlab* provides a development environment for creating specialized GUIs (GUIDE), incorporating menus, push and radio buttons, text boxes and many other interface devices. In our system, the GUI consists of a set of several windows associated with different tasks. Some windows are for geometric processing of surfaces and include dialogue boxes to allow users to input the symbolic equations of the geometric entities. Other windows (such as those for visualizing industrial files of different formats: IGES, STEP, CATIA) are mostly intended for graphical output and include different commands for visualization. Several examples of those windows will be shown in Section 3.
2. *a library of commands (toolbox)*: to deal with the most popular geometric entities, more than 200 functions and commands have been implemented in our PSE. The resulting libraries are continuously updated and extended, so the system must be flexible enough to allow the programmer to improve the algorithms and codes in an efficient, quick and easy way. Users can deal with geometric entities in either numerical or symbolic way via the *Symbolic Math Toolbox*, which contains a subset of the *Maple*'s kernel. At their turn, the numerical functions invoke low-level functions from the numerical kernel, which can be given as *Matlab* functions, M-files, C/C++ MEX files, API files or built-in files (see [14] for details).
3. *a Matlab graphics library*: this module includes the graphical commands needed for our setup. For instance, in the case of PC platforms, it is a set of DLLs (Dynamic Link Libraries). In general, they are not included in the *Matlab* standard version and should be purchased separately. They allow users to export the graphical interfaces and simultaneously preserve the good interplay with the numerical commands.

2.2 Implementation Issues

Regarding the implementation, the PSE has been developed by the authors in *Matlab* v6.0 by using a Pentium III processor at 2.4 GHz. with 512 MB of RAM. However, the program (whose minimum requirements are listed in Table 1) supports many different platforms, such as PCs (with Windows 9x, 2000, NT, Me and XP) and UNIX workstations from Sun, Hewlett-Packard, IBM, Silicon Graphics and Digital. Figures in this paper correspond to the PC platform version.

The graphical tasks are performed by using the *Matlab* GUI for the higher-level functions (windowing, menus, or input) while the graphics library *Open GL* is used for rendering purposes. The numerical kernel has been implemented in the native *Matlab* programming language, and the symbolic kernel has been created by using the commands of the *Symbolic Math Toolbox*.

Another interesting feature of our PSE is the possibility of compiling and linking its different modules and windows along with their associated libraries and underlying code to generate standalone applications. Roughly, this process can be summarized as follows: firstly, the chosen module is compiled by using the *Matlab* compiler. The result is a set of multiple C or C++ source code modules

Table 1. Minimum hardware configuration to install the PSE

Hardware	Requirement
Operating System	Windows (9x, 2000, NT, Me, XP), UNIX
RAM	128 MB
Disk storage	25 MB (60 MB recommended)
Monitor	Super VGA monitor
Screen resolution	minimum of 640 × 480

that are actually versions in C/C++ of the initial M-files implemented in *Matlab*. These files are subsequently compiled in a C/C++ environment to generate the object files. Then, those files are linked with the C++ graphics library, M-file library, Built-In library, API library and ANSI C/C++ library files. The final result is an executable program running on our computer. This feature provides our industrial partner with the facilities for input/output operations in collaborative frameworks involving third parties - even though those parties use different software - via these standalone applications.

We should remark here the excellent integration of all these tools to generate optimized code that can easily be invoked from C/C++ via Dynamic Link Libraries (DLLs), providing both great portability and optimal communication between all modules. Complementary, we can construct an installer for the PSE by using either Visual C++ or Visual Basic environments. The authors have accomplished all tasks mentioned above during the process of creating the PSE.

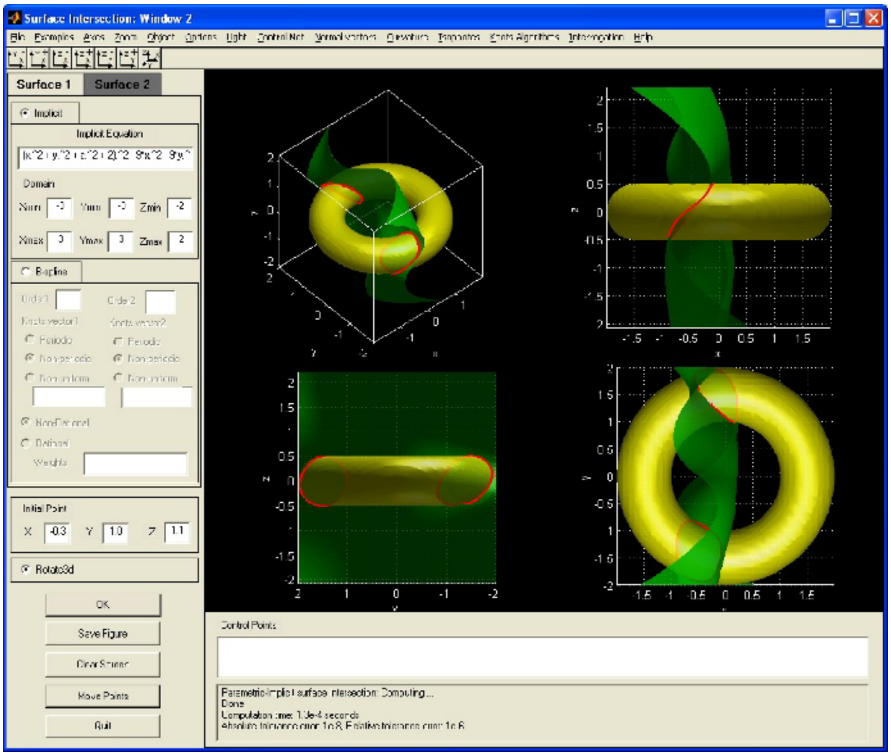
3 Examples of Application

In this section some simple yet illustrative examples of application of our PSE aimed to show the good performance of the program are briefly discussed. The chosen examples concern the issues of surface intersection, generation of tool-path trajectories for NC machining and the visualization of geometric entities stored in industrial files of several formats.

3.1 Surface Intersection

The intersection of surfaces is one of the most outstanding problems in many fields, such as computational geometry, solid modeling, geometric processing, visualization and manufacturing of 3D entities. It appears in the countouring of surfaces [9], in numerical-controlled machining [4] (for instance, the intersection of offset surfaces with series of parallel planes), in the boundary (B-rep) representation for Constructive Solid Geometry models of the objects, in manufacturing [2] (slicing operations for rapid prototyping, determination of collisions), etc.

During the last decades, a number of different methods to compute the intersection of surfaces have been described in the literature (see, for example, the excellent reviews on this topic in [11] (Chapter 12) or [16] (Chapter 5) and the



Implicit Equation: (x.^2+y.^2+z.^2+2).^2-9*x.^2-9*y.^

Domain: Xmin -3 Ymin -3 Zmin -2 Xmax 3 Ymax 3 Zmax 2

Initial Point: X -0.3 Y 1.0 Z 1.1

```
Parametric-implicit surface intersection: Computing...
Done
Computation time: 1.3e-4 seconds
Absolute tolerance error: 1e-8, Relative tolerance error: 1e-6
```

Fig. 1. Screenshot of the surface intersection window: example of the implicit-parametric case

references therein). Basically, they can be classified into analytical and numerical methods. Analytical methods seek exact solutions by finding some function describing the intersection curves. Although unaffected by robustness and efficiency limitations, they require many different algorithms designed *ad hoc* for each kind of surface involved. Furthermore, they cannot deal with non-algebraic surfaces, are quite slow and have huge requirements in terms of computation power and resources. Therefore, numerical methods are usually applied for industrial purposes, including combinations of algebraic and analytical methods [6], hybrid algorithms combining subdivision (based on the divide-and-conquer methodologies), tracing and numerical methods (mainly Newton's method) [13], etc. Unfortunately, they exhibit a substantial loss of accuracy making them unsuitable for practical applications. Finally, there is a family of methods known as marching methods based on generating a sequence of points of an intersection curve branch by stepping from a point on such a curve in a direction determined by some local differential geometry analysis.

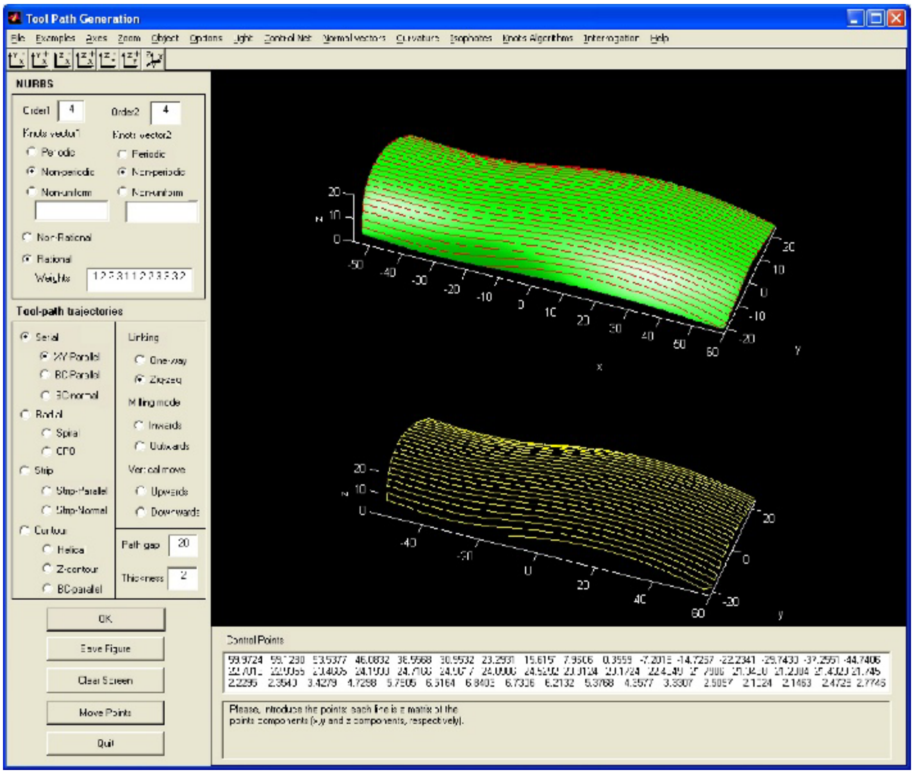
Our PSE has a special module devoted to compute the intersection curves of two surfaces. To this aim, we introduced a new algebraic-differential method based on formulating the problem in terms of an initial value problem of first-order ordinary differential equations (ODEs). The resulting system of ODEs is then numerically integrated through an adaptive 4-5-order Runge-Kutta method. This approach has shown to work properly for the cases of parametric and implicit surfaces [7,20]. The method requires a starting point on the intersection curve for each branch of the solution. To determine such a point we trace a path on those surfaces by following the direction indicated by either the gradient of the distance between both surfaces (for the parametric-parametric case) or the vector field of the implicit surface (for the parametric-implicit and the implicit-implicit cases) computed by following [22]. This procedure yields a starting point on the nearest branch of the intersection curve.

Figure 1 shows the surface intersection window of our system along with an example for the implicit-parametric case. The parametric surface is a NURBS, described by its control points, weights and knot vectors, while the implicit surface is described by its symbolic equation (as shown in the input box of surface 1). Our surface intersection algorithm is applied and the resulting intersection curves as well as the initial surfaces are displayed. The figure shows the 3D picture and its projections onto the three coordinate planes. The system allows the user to set up the threshold for the absolute and relative tolerance errors (10^{-8} and 10^{-6} respectively in this example). The computation time for those values is 1.3×10^{-4} seconds in this example.

3.2 Generation of Tool-Path Trajectories for NC-Machining

A major problem in manufacturing is the determination of the best trajectory for the cutting tools in NC (numerical controlled) machining. For example, it has been shown that there is no simple way to describe the guiding surface for the cutter of a five-axis machining tool [11]. On the contrary, there are many different possibilities, depending on the tool topology, surface shape, etc. (see [16] for details).

Figure 2 shows an example of the tool-path generation module of our PSE. As shown in this figure, the user can input a NURBS surface by specifying: (1) the order of the parametric variables, (2) the kind of knot vectors (periodic, non-periodic and non-uniform, according to the classification in [1]), (3) the control points and (4) their weights. Then, the type of tool-path trajectory (serial, radial, strip or contour, according to the classification in [4]) must be chosen. At its turn, the trajectory can be connected by either one-way or by zig-zag, while the milling mode and vertical move can be in-outwards and up-downwards, respectively. Other parameters such as the distance between back-to-back paths (path-gap) and the path thickness can also be given. The example in Fig. 2 displays a zig-zag serial XY-parallel trajectory on a bicubic NURBS surface (top) and the trajectory itself (bottom). Additional calculations for the given surface - such as the derivatives, fundamental forms, curvatures (Gauss, Median, principal), etc. - can also be performed. The most usual algorithms for NURBS surfaces - such as subdivision, degree raising, knot insertion and refinement,etc.- can also be applied at will.


Tool Path Generation
File Examples Axes Zoom Object Options Light Control Net Normal vectors Curvature Isophotes Knots Algorithms Interrogation Help
OK
Save Figure
Clear Screen
Move Points
Quit
x
y
z

NURBS

Order1 4 Order2 4

Knots vector1: Periodic [✓ Non-periodic] Non-uniform

Knots vector2: Periodic [✓ Non-periodic] Non-uniform

Non-Rational [✓ Rational]

Weights 1 2 3 3 1 1 2 3 3 2 3 2

Tool-path trajectories

[✓ Serial]: [✓ XY Parallel] BC Parallel BC normal

Radial: Spiral CPO

Strip: Strip-Parallel Strip-Normal

Contour: Helical Z-contour BC-parallel

Linking: One-way [✓ Zig-zag]

Milling mode: Inwards Outwards

Vertical move: Upwards Downwards

Path gap 20

Thickness 2

Control Points

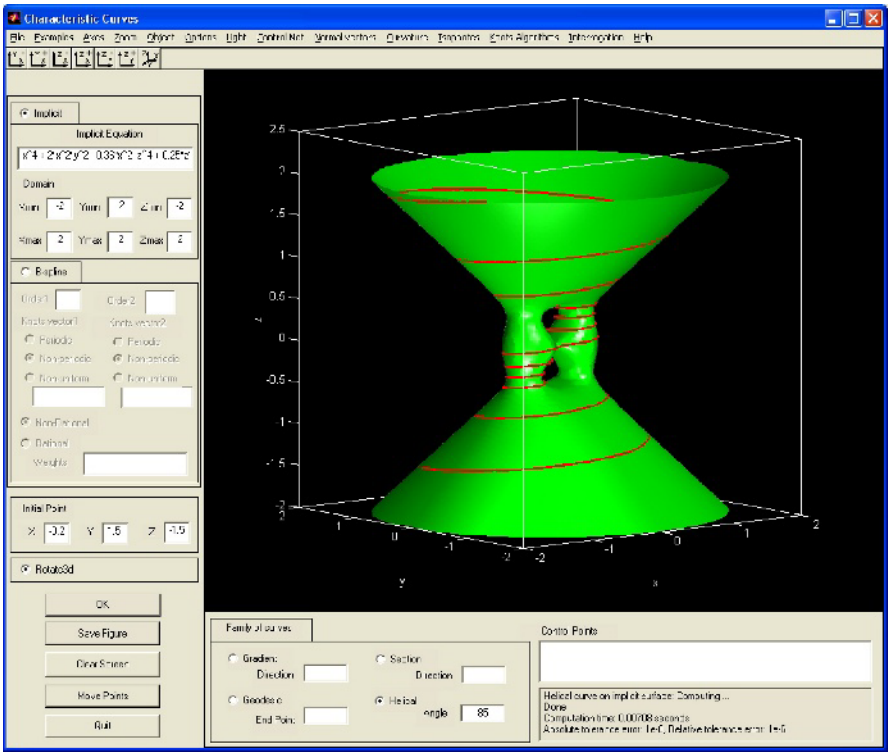
```
59.9724 59.[illegible]230 53.5377 46.0832 38.5568 30.9532 23.2931 15.615[illegible] 7.9506 0.3559 -7.2013 -14.7257 -22.2341 -29.7433 -37.2551 -44.7406
[illegible]
2.2295 2.3543 3.4279 4.7258 5.7505 6.5164 6.8403 6.7336 6.2132 5.3768 4.3577 3.3307 2.505[illegible] 2.1224 2.1463 2.4723 2.7746
```

Please, introduce the points: each line is a matrix of the points components (x,y and z components, respectively).

Fig. 2. Screenshot of the tool-path generation window: example of a zig-zag serial XY-parallel trajectory on a bicubic NURBS surface

Another reasonable approach is to consider the so-called *characteristic curves* on a surface. They are curves reflecting either the visual or the geometric properties of the surface. For the visual properties we can use the *reflection lines* [12] and the *isophotes* [17], which help to evaluate the behavior and aesthetics of the surface under illumination models, while the geometric properties can be accurately analyzed through the *contour lines* [3,23], the *lines of curvature* [3], *geodesic paths* [3,15], *asymptotic lines* [24], etc.

In our PSE we consider four kind of characteristic curves on surfaces (see Figure 3): gradient, section, geodesic and helical curves. For each kind of curves, novel especialized methods have been developed and implemented [19,21,22], so that users can apply them even with a minimal knowledge about the method and the PSE. These new methods yield efficient trajectories for cutting tools that have interesting mathematical properties and a geometrical meaning: for instance, the geodesic curves are used to minimize the distance on the surface between two end points and, hence, the time required for machining; the section curves are planar curves, so the vertical component of the main axis of the cutter tool does not oscillate; the helical topology is also highly used in



Implicit [✓]

Implicit Equation: x^4 + 2x^2y^2 0.36*x^2 z^4 + 0.25*z

Xmin -2 Ymin 2 Zmin -2

Xmax 2 Ymax 2 Zmax 2

Initial Point: X -0.2 Y 1.5 Z 1.5

Rotate3d [✓]

Helical [✓] angle 85

Helical curve on implicit surface: Computing ...
Done
Computation time: 0.00708 seconds
Absolute tolerance error: 1e-6, Relative tolerance error: 1e-6

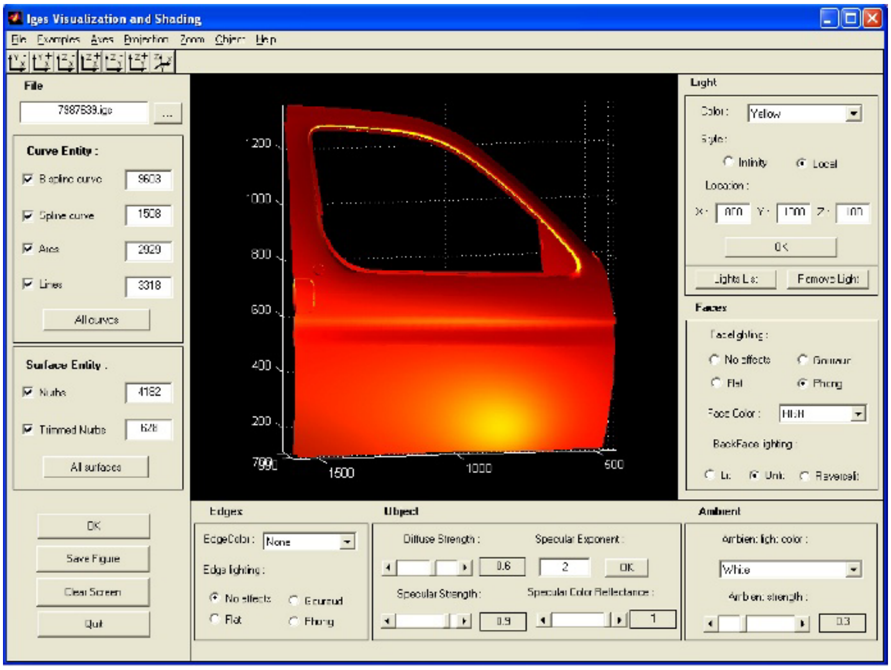
Fig. 3. Screenshot of the characteristic curves window: example of a helical curve on an implicit surface

high-speed machining, and so on. The methods can be applied to either implicit or parametric surfaces provided that they are differentiable.

Figure 3 shows an example of the generation of a helical curve on an implicit surface for five-axis NC machining. A helical curve is a curve whose tangent maintains a constant angle or slope with respect to a given direction (85 degrees with respect to the vertical axis in this example). This kind of problems appears quite often in three-axis, five-axis and high-speed NC machining [4,16].

3.3 Visualization of Industrial Files

Figure 4 shows an example of the application of the powerful *Matlab* graphical capabilities to display large files storing geometric entities in the most popular industrial formats. In particular, a mechanical part of a car body (the right front door) stored in an IGES-format industrial file is visualized. To this purpose, the geometrical information of the different entities for curves (Bspline curves, spline curves, arcs, lines) and surfaces (NURBS, trimmed NURBS) is read and computed. Then, suitable commands for computer graphics, including surface algorithms (wireframe, hidden-line removal or z-buffer), shaders (constant,



File: 7987539.igs

Curve Entity:
[✓] B-spline curve 9603
[✓] Spline curve 1508
[✓] Arcs 2929
[✓] Lines 3318

Surface Entity:
[✓] Nurbs 4182
[✓] Trimmed Nurbs 628

Light
Color: Yellow
Style: Infinity [✓ Local]
Location: X: 800 Y: 1700 Z: 100

Faces
Face lighting: No effects, Gouraud, Flat, [✓ Phong]
Face Color: HIGH
BackFace lighting: Lit [✓ Unlit] Reverselit

Edges
EdgeColor: None
Edge lighting: [✓ No effects] Gouraud Flat Phong

Object
Diffuse Strength: 0.6
Specular Exponent: 2
Specular Strength: 0.9
Specular Color Reflectance: 1

Ambient
Ambient light color: White
Ambient strength: 0.3

Fig. 4. Screenshot of the industrial files visualization window: example of the visualization of a mechanical part of a car body stored in IGES format

faceted, Gouraud, Phong, etc.), lighting techniques (ambient, diffuse, specular, etc.), texture mapping, light sources of different colors and positions, interactive manipulation (coloring, zooming, rotation), etc. are applied. These illumination effects allow us to simulate the mechanical part with a high level of quality. In addition to their aesthetic effect, the illumination models are important to detect irregularities on surfaces, for example, by using some illumination lines called isophotes [10]. This procedure does reproduce the real situation of looking for the different reflection lines on the surface of a car body.

4 Conclusions and Further Remarks

In this paper a new problem-solving environment for geometric processing of surfaces has been described. The PSE is aimed to support the full range of activities carried out by our partner in the field of geometric processing for the automotive industry. The PSE is comprised of several modules, each devoted to a specific task: surface intersection, tool-path generation, measurement of distances on surfaces, surface interrogation, illumination models, etc. This modular structure allows the programmer to introduce his/her own methods - even new modules - and perform trials at will. Usually, each module is associated with one or several windows on which the user can input data (symbolic equations, numerical parameters and values, industrial files), perform symbolic and numerical calculations and display the solution in either text-like or graphical way. Additionally, the user can compile the modules to generate standalone applications that can run independiently. This procedure can be applied to any extent, from individual functions to single modules (even the whole system). In these later cases, the procedure requires to link the modules with numerical and/or graphical libraries. Powerful communication and connectivity tools for a variety of programming languages and programs enhance the applicability of this PSE.

The PSE has shown to provide our partner with reliable solutions to its problems and to serve as a communication channel for exchange of geometrical data. The system can deal with implicit and parametric surfaces, including the case of NURBS surfaces (by far, the most common surfaces in industry). In fact, the most popular industrial formats for the geometrical description of mechanical parts of car bodies are properly handled and displayed (currently only IGES format is fully supported; STEP and CATIA formats are in progress). Finally, the excellent Open GL library can be applied for visualization purposes. This feature improves dramatically the quality of pictures and provide users with a powerful tool for surface interrogation and quality of shape analysis via illumination models.

The present work is still in progress and can be improved in many different ways. Additional modules as well as the improvement of the existing ones are the next goals for future work. Our experience with this system and the feedback from our partner have been very positive and encouranging and convinced us about the feasibility and usefulness of this work. Although this paper has been affected by strong limitations of space, we still hope we have provided enough details to make the manuscript helpful to those interested to follow a similar approach.

References

1. Anand, V.B.: Computer Graphics and Geometric Modeling for Engineers. John Wiley and Sons, New York (1993)
2. Barnhill, R.E.: Geometric Processing for Design and Manufacturing. SIAM, Philadelphia (1992)
3. Beck, J.M., Farouki, R.T., Hinds, J.K.: Surface analysis methods. IEEE Computer Graphics and Applications, Dec. (1986) 18-36
4. Choi, B.K., Jerard, R.B: Sculptured Surface Machining. Theory and Applications. Kluwer Academic Publishers, Dordrecht/Boston/London (1998)
5. Farin, G.E.: Curves and Surfaces for Computer Aided Geometric Design, Fourth Edition. Academic Press, San Diego (1996)
6. Farouki, R.T.: Direct surface section evaluation. In: Farin, G. (ed.), Geometric Modeling. Algorithms and New Trends, SIAM, Philadelphia (1987) 319-334
7. Gálvez, A., Puig-Pey, J., Iglesias, A.: A Differential Method for Parametric Surface Intersection. Lectures Notes in Computer Science, **3044** (2004) 651-660
8. Generic Enabling Application Technologies Working Group. Workshop on "Grids for Complex Problem-Solving" organized by the European Commission, DG INFSO-F2 in January 2003 (Web page:`http://www.cordis.lu/ist/grids`).

9. Grandine, T.A.: Applications of contouring, SIAM Review **42** (2000) 297–316
10. Grids for Integrated Problem-Solving Environments: Status and Research Perspectives vs. Requirements from an Industrial Viewpoint. Workshop organized by Fraunhofer SCAI. Sankt Augustin, Germany, April 29th-30th (2003)
11. Hoschek, J., Lasser, D.: Computer-Aided Geometric Design, A.K. Peters, Wellesley, MA (1993)
12. Klass, R.: Correction of local surface irregularities using reflection lines. Computer Aided Design **12**(2) (1980) 73-77
13. Kriezis, G.A., Patrikalakis, N.M., Wolters, F.E.: Topological and differential-equation methods for surface intersections, Computer Aided Design **24**(1) (1992) 41-55
14. The MathWorks Inc: Using Matlab, (1997); see also its Web Page: *http://www.mathworks.com*
15. Munchmeyer, F.C., Haw, R.: Applications of differental geometry to ship design. In: Computer Applications in the Automation of Shipyard Operation and Ship Design IV, Rogers, D.F., et al. (eds.) North Holland, Amsterdam (1982) 183-188
16. Patrikalakis, N.M., Maekawa, T.: Shape Interrogation for Computer Aided Design and Manufacturing. Springer-Verlag, New York, Berlin Heidelberg (2002)
17. Poeschl, T.: Detecting surface irregularities using isophotes. Computer Aided Geometric Design **1**(2) (1984) 163-168
18. Web Page: *http://www.mathworks.com/company/pressroom.*
19. Puig-Pey, J., Gálvez, A., Iglesias, A.: Polar Isodistance Curves on Parametric Surfaces. Lectures Notes in Computer Science, **2330** (2002) 161-170
20. Puig-Pey, J., Gálvez, A., Iglesias, A.: A New Differential Approach for Parametric-Implicit Surface Intersection. Lectures Notes in Computer Science, **2657** (2003) 897-906
21. Puig-Pey, J., Gálvez, A., Iglesias, A.: Helical Curves on Surfaces for Computer-Aided Geometric Design and Manufacturing. Lectures Notes in Computer Science, **3044** (2004) 771-778
22. Puig-Pey, J., Gálvez, A., Iglesias, A., Rodriguez, J., Corcuera, P., Gutierrez, F.: Some applications of scalar and vector fields to geometric processing of surfaces. Computers & Graphics, **29**(5) (2005) 723-729
23. Satterfield, S.G., Rogers, D.F.: A procedure for generating contour lines from a B-spline surface. IEEE Computer Graphics and Applications, Apr. (1985) 71-75
24. Theisel, H., Farin, G.E.: The curvature of characteristic curves on surfaces. IEEE Computer Graphics and Applications, Nov./Dec. (1997) 88-96

A *Mathematica* Notebook for Computing the Homology of Iterated Products of Groups

V. Álvarez, J.A. Armario, M.D. Frau, and P. Real*

Dpto. Matemática Aplicada I, Universidad de Sevilla, Avda. Reina Mercedes s/n
41012 Sevilla, Spain
{valvarez, armario, mdfrau, real}@us.es

Abstract. Let G be a group which admits the structure of an iterated product of central extensions and semidirect products of abelian groups G_i (both finite and infinite). We describe a *Mathematica 4.0* notebook for computing the homology of G, in terms of some homological models for the factor groups G_i and the products involved. Computational results provided by our program have allowed the simplification of some of the formulae involved in the calculation of $H_n(G)$. Consequently the efficiency of the method has been improved as well. We include some executions and examples.

1 Introduction

The calculation of the homology of a given group is in general a difficult task. From a theoretical point of view, spectral sequences and resolutions have been traditionally used to solve the question. But calculations could not be carried out in practice, due to the complexity of the processes involved in the computations.

In the past decade, the interest in explicitly compute both the (co)homology and the correspondent representative (co)cycles of a group, has increased surprisingly, accordingly to their applications to very different fields, such as coding theory and cryptography.

This situation has motivated that most of Computer Algebra Systems (CAS) are now concerned about the design of functions for achieving homological calculations.

For instance, GAP (*Group, algorithms, programming* [9]) includes a *homology package* [7], which is concerned with the calculation of the homology of simplicial complexes and the Smith's normal forms of (preferably sparse) integer matrices. MAGMA [18] includes a routine for computing the homology of p-groups [5]. The KENZO system [6] provides an environment for achieving calculations in the framework of *effective homology* [21]. More recently, Ellis is developing a homological algebra library for use with the GAP computer algebra system (termed HAP [11], *homological algebra programming*), which intends to be a complete tool to make basic calculations in the cohomology of finite and infinite groups.

* All authors are partially supported by the PAICYT research project FQM–296 from Junta de Andalucía (Spain).

© Springer-Verlag Berlin Heidelberg 2006

Though the KENZO system has incorporated some routines for the calculation of the homology of central extensions (in light of the work in [20]), as far as we know, none of these packages handles the class of groups that we are concerned with. We intend to cover this gap with the *Mathematica* notebook [1] which is presented here, for computing the homology of iterated products of central extensions and semidirect products of abelian groups.

The main reason for which we have decided to work on *Mathematica* instead of other CAS is that all the systems cited above work in terms of resolutions. We prefer to work in terms of "contractions", that is, at the level of reduced complexes of resolutions, following the philosophy in [8,21]. In most cases, both ways turn out to be equivalent [4]. Our preferences come from the fact that we want to elude the recursive formulae provided by the comparison theorem for resolutions. We prefer to work with explicit formulae from the first moment. For instance, the formulae in [19] for the maps involved in the Eilenberg-Zilber theorem, will allow us to perform explicit formulae for the maps describing a homological model for the iterated products that we are concerned with.

Furthermore, our method may be extended to cover iterated products of other groups for which homological models are known, such as finitely generated torsion-free nilpotent groups [17,15], finite p-groups [16,10], finitely generated two-step nilpotent groups [12] and metacyclic groups [13].

We have programmed the formulae of [3] for constructing a homological model for a semidirect product of abelian groups, and the formulae implicitly described in [20] for constructing a homological model for a central extension of abelian groups.

The term *homological model* refers to a contraction ${}^{\phi:}\bar{B}(\mathbb{Z}[G]) \underset{g}{\overset{f}{\rightleftharpoons}} hG$ from the *reduced bar construction* of the group G (i.e. the reduced complex associated to the standard bar resolution) to a differential graded module of finite type hG, so that

$$H_*(G) = H_*(\bar{B}(\mathbb{Z}[G])) = H_*(hG)$$

and the homology of hG may be effectively computed by means of Veblen's algorithm [22] (involving the Smith's normal forms of the matrices representing the differential operator).

Here ${}^{\phi:}\bar{B}(\mathbb{Z}[G]) \underset{g}{\overset{f}{\rightleftharpoons}} hG$ denotes a *contraction*, a special type of homotopy equivalence, where apart from the usual relations $fg = 1$, $1 - gf = d\phi + \phi d$, the annihilation properties $f\phi = 0$, $\phi g = 0$, $\phi\phi = 0$ are satisfied.

We must note that a routine in *Mathematica* which calculates the Smith's normal form of a matrix over the integers was provided a decade ago by David Jabon [14].

Using this package and the formulae above, the notebook finally computes the homology of the input group.

We organize the paper as follows. Section 2 is devoted to describe the formulae concerning the homological models for central extensions and semidirect products of abelian groups described in [20] and [3], respectively. The notebook itself is described in Section 3. In Section 4 we prove some results about simplifications on the formulae, which had been previously conjectured, attending to some output data provided by our program. Section 5 is devoted to show some executions and examples.

2 Describing Homological Models for the Factors

Let G be an iterated product of central extensions and semidirect products of abelian groups G_i, $1 \leq i \leq n$. In this section, we describe a homological model for G in terms of some homological models for each of the factor groups G_i.

2.1 A Homological Model for $\mathbb{Z}$

Let $E(u)$ denote the free DGA-algebra endowed with trivial differential and generators 1 (at degree 0) and u (at degree 1), so that $u \cdot u = 0$.

The comparison theorem for resolutions provides a homological model for $\mathbb{Z}$ (see [4] for details), ${}^{\phi_{\mathbf{Z}}}\!\!: \bar{B}(\mathbb{Z}[\mathbb{Z}]) \underset{g_{\mathbf{Z}}}{\overset{f_{\mathbf{Z}}}{\rightleftharpoons}} E(u)$, which is a subtle modification of that in [8] (they differ just in the homotopy operator $\phi_{\mathbf{Z}}$).

Here $g_{\mathbf{Z}}(u) = [1]$, $f_{\mathbf{Z}}([n_1|\cdots|n_q]) = \begin{cases} n_1\, u, & \text{if } q = 1 \\ 0, & \text{if } q > 1 \end{cases}$ and

$$\phi_{\mathbf{Z}}[n_1|\ldots|n_k] = \begin{cases} (-1)^k \displaystyle\sum_{i=1}^{n_k-1} [n_1|\ldots|n_{k-1}|i|1], & \text{if } n_k > 1 \\ 0, & \text{if } n_k = -1, 0, 1 \\ (-1)^k \displaystyle\sum_{i=1}^{-n_k-1} [n_1|\ldots|n_{k-1}|-i|1], & \text{if } n_k < 1 \end{cases} \tag{1}$$

2.2 A Homological Model for $\mathbb{Z}_n$

Let $\Gamma(v)$ denote the free DGA-algebra endowed with trivial differential and generators $\gamma_k(v)$ (at degree $2k$, $k \geq 0$, $\gamma_0(v) = 1$), such that

$$\gamma_k(v)\gamma_h(v) = \frac{(k+h)!}{k!h!}\gamma_{k+h}(v)$$

In [8] a homological model ${}^{\phi_{\mathbf{Z}_n}}\!\!: \bar{B}(\mathbb{Z}[\mathbb{Z}_n]) \underset{g_{\mathbf{Z}_n}}{\overset{f_{\mathbf{Z}_n}}{\rightleftharpoons}} (E(u) \otimes \Gamma(v), d)$ for $\mathbb{Z}_n$ is also described, such that $d(u) = 0$, $d(u \otimes v) = n \cdot u$ and $g_{\mathbf{Z}_n}(u) = [1]$,

$$g_{\mathbf{Z}_n}(\gamma_k(v)) = \sum_{x_i \in \mathbf{Z}_n} [1|x_1|\cdots|1|x_k], \qquad g_{\mathbf{Z}_n}(u\gamma_k(v)) = \sum_{x_i \in \mathbf{Z}_n} [1|x_1|\cdots|1|x_k|1]$$

$$\begin{aligned} f_{\mathbf{Z}_n}[x_1|y_1|\cdots|x_m|y_m] &= [\prod_{i=1}^{m} \delta_{x_i,y_i}]\gamma_m(v), \\ f_{\mathbf{Z}_n}[x_1|y_1|\cdots|x_m|y_m|z] &= [z\prod_{i=1}^{m} \delta_{x_i,y_i}]u\gamma_m(v), \end{aligned} \qquad \text{for } \delta_{x_i,y_i} = \begin{cases} 0, & x_i+y_i<n \\ 1, & x_i+y_i \geq n \end{cases}$$

and $\phi_{\mathbf{Z}_n}([x_1|\cdots|x_k]) = -\varphi_{\mathbf{Z}_n}([x_1|\cdots|x_k])$, for $\varphi_{\mathbf{Z}_n}[\,] = 0$, $\varphi_{\mathbf{Z}_n}[x] = \sum_{i=1}^{x-1}[1|i]$,

$$\varphi_{\mathbf{Z}_n}[x|y|\sigma] = \sum_{i=1}^{x-1}[1|i|\sigma] + \delta_{x,y}\sum_{k=1}^{n-1}[1|k|\varphi_{\mathbf{Z}_n}\sigma] \tag{2}$$

2.3 A Homological Model for a Central Extension

Here $A \ltimes_\alpha G$ denotes the central extension of A and G by means of the 2-cocycle $\alpha : G \times G \to A$, such that $(a,g)(a',g') = (a+a'+\alpha(g,g'), g+g')$. In case that α is a 2-coboundary, then $A \ltimes_\alpha G$ is isomorphic to the direct product $A \times G$.

A homological model ${}^{\phi_{AG}} \bar{B}(\mathbb{Z}[A \ltimes_\alpha G]) \underset{g_{AG}}{\overset{f_{AG}}{\rightleftharpoons}} (hAG, d_{AG})$ for a central extension $A \ltimes_f G$, for A being an abelian group, is described in [20], in terms of homological models ${}^{\phi_A} \bar{B}(\mathbb{Z}[A]) \underset{g_A}{\overset{f_A}{\rightleftharpoons}} (hA, d_A)$ and ${}^{\phi_G} \bar{B}(\mathbb{Z}[G]) \underset{g_G}{\overset{f_G}{\rightleftharpoons}} (hG, d_G)$ of the factor groups A and G. Explicitly,

$$\begin{aligned}
&f_{AG} = (f_A \otimes f_G)_{t\cap} AW_\delta \psi\varphi \\
&g_{AG} = \varphi\psi EML_\delta (g_A \otimes g_G)_{t\cap} \\
&\phi_{AG} = \varphi^{-1}\psi^{-1}(SHI_\delta + EML_\delta(1\otimes\phi_G + \phi_A \otimes g_G f_G)t\cap AW_\delta)\psi\varphi \\
&d_{AG} = d_A \otimes 1 + 1 \otimes d_G + (f_A \otimes f_G)t \cap \sum_{i\geq 0}(-1)^i[(1\otimes\phi_G + \phi_A \otimes g_G f_G)t\cap]^i (g_A \otimes g_G) \\
&(f_A \otimes f_G)_{t\cap} = (f_A \otimes f_G)(1 - t\cap \sum_{i\geq 0}(-1)^i[(1\otimes\phi_G + \phi_A\otimes g_G f_G)t\cap]^i)(1\otimes\phi_G + \phi_A \otimes g_G f_G) \\
&(g_A \otimes g_G)_{t\cap} = \sum_{i\geq 0}(-1)^i[(1\otimes\phi_G + \phi_A\otimes g_G f_G)t\cap]^i (g_A \otimes g_G) \\
&(1\otimes\phi_G + \phi_A \otimes g_G f_G)_{t\cap} = \sum_{i\geq 0}(-1)^i[(1\otimes\phi_G + \phi_A\otimes g_G f_G)t\cap]^i(1\otimes\phi_G + \phi_A\otimes g_G f_G) \\
&t\cap = AW\delta\sum_{i\geq 0}(-1)^i(SHI\delta)^i EML \\
&AW_\delta = AW(1 - \delta\sum_{i\geq 0}(-1)^i(SHI\delta)^i SHI) \\
&EML_\delta = \sum_{i\geq 0}(-1)^i(SHI\delta)^i EML \\
&SHI_\delta = \sum_{i\geq 0}(-1)^i(SHI\delta)^i SHI \\
&\delta((a_{n-1},\ldots,a_0),(g_{n-1},\ldots,g_0)) = -((a_{n-2},\ldots,a_0),(g_{n-2},\ldots,g_0)) + \\
&\qquad (-\alpha(g_{n-2},g_{n-1}) + a_{n-2}, -\alpha(g_{n-3},g_{n-2}+g_{n-1}) + \alpha(g_{n-3},g_{n-2}) + a_{n-3},\ldots, \\
&\qquad\qquad \ldots, -\alpha(g_0, g_1+\ldots+g_{n-1}) + \alpha(g_0, g_1+\ldots+g_{n-2}) + a_0), (g_{n-2},\ldots,g_0))
\end{aligned}$$

$$
\begin{array}{l}
\varphi([(a_0,g_0),\dots,(a_n,g_n)]) = \begin{cases} ((a_0,g_0),\dots,(a_n,g_n)), & \text{if } A\ltimes_\alpha G \text{ is abelian} \\ (-1)^{\lceil \frac{n}{2}\rceil+1}((a_n,g_n),\dots,(a_0,g_0)), & \text{otherwise} \end{cases} \\
\psi[(a_{n-1},g_{n-1}),\dots,(a_0,g_0)] = \\
\qquad ([a_{n-1},a_{n-2}+\alpha(g_{n-2},g_{n-1}),\dots,a_0+\alpha(g_0,g_1+\dots g_{n-2}+g_{n-1})],[g_{n-1},\dots,g_0]) \\
\psi^{-1}([a_{n-1},\dots,a_0],[g_{n-1},\dots,g_0]) = [(a_{n-1},g_{n-1}),(a_{n-2}-\alpha(g_{n-2},g_{n-1}),g_{n-2}),\dots, \\
\qquad \dots,(a_{n-i}-\alpha(g_{n-i},g_{n-i+1}+\dots+g_{n-2}+g_{n-1}),g_{n-i}), \\
\qquad\qquad \dots,(a_0-\alpha(g_0,g_1+\dots+g_{n-2}+g_{n-1}),g_0)] \\
AW((a_{n-1},\dots,a_0),(g_{n-1},\dots,g_0)) = \sum_{i=0}^{n}(a_{n-1},\dots,a_{n-i})\otimes(g_{n-i-1},\dots,g_0) \\
EML((a_{p-1},\dots,a_0)\otimes(g_{q-1},\dots g_0)) = (a_{p-1},\dots,a_0)\star(g_{q-1},\dots g_0) \\
SHI((a_{n-1},\dots,a_0),(g_{n-1},\dots,g_0)) = \\
\qquad \sum_{q=0}^{n-1}\sum_{p=0}^{n-p-q}((a_{n-1},\dots,a_{p+q+1},0),(g_{n-1},\dots,g_{p+q+1},g_{p+q}+\dots+g_q))|| \\
\qquad\qquad (a_{p+q},\dots,a_q)\star(q_{q-1},\dots,g_0)
\end{array}
$$

The symbol $\star$ refers to the shuffle product, so that the output of

$$(a_{p-1},\dots,a_0)\star(g_{q-1},\dots,g_0)$$

consists in the sum of all the different shuffles of the tuples, such that the inner order in the lists is preserved. The sign correspondent to a particular shuffle depends on the number of positions that elements a_i have got ahead of elements g_j.

2.4 A Homological Model for a Semidirect Product

Here $A\rtimes_\alpha G$ denotes the semidirect product of A and G by means of the homomorphism $\alpha : G \to \mathrm{Aut}(A)$, such that $(a,g)\cdot(a',g') = (a+\alpha(g)(a'),g+g')$. In case that α is the zero map, then $A\rtimes_\alpha G$ consists in the direct product $A\times G$.

A homological model ${}^{\phi_{AG}}\!:\bar{B}(\mathbb{Z}[A\rtimes_\alpha G]) \underset{g_{AG}}{\overset{f_{AG}}{\rightleftarrows}} (hAG,d_{AG})$ for a semidirect product $A\rtimes_\alpha G$, for G being an abelian group, is described in [3], in terms of homological models ${}^{\phi_A}\!:\bar{B}(\mathbb{Z}[A]) \underset{g_A}{\overset{f_A}{\rightleftarrows}} (hA,d_A)$ and ${}^{\phi_G}\!:\bar{B}(\mathbb{Z}[G]) \underset{g_G}{\overset{f_G}{\rightleftarrows}} (hG,d_G)$ of the factor groups A and G.

The formulae for central extensions given before also apply for a semidirect product, with the following exceptions,

$$
\begin{array}{l}
\delta((a_{n-1},\dots,a_0),(g_{n-1},\dots,g_0)) = -((a_{n-2},\dots,a_0),(g_{n-2},\dots,g_0))+ \\
\qquad +((\alpha(g_{n-1})(a_{n-2}),\dots,\alpha(g_{n-1})(a_0)),(g_{n-2},\dots,g_0)) \\
\psi[(a_{n-1},g_{n-1}),\dots,(a_0,g_0)] = \\
\qquad ((\alpha(g_{n-1}^{-1})(a_{n-1}),\dots,\alpha(g_{n-1}^{-1}+\dots+g_0^{-1})(a_0)),(g_{n-1},\dots,g_0)) \\
\psi^{-1}((a_{n-1},\dots,a_0),\ (g_{n-1},\dots,g_0)) = \\
\qquad ((\alpha(g_{n-1})(a_{n-1}),g_{n-1}),\dots,(\alpha(g_0+\dots g_{n-1})(a_0),g_0))
\end{array}
$$

2.5 A Homological Model for an Iterated Product

Let G be an iterated product of central extensions and semidirect products of abelian groups, which admits the form $A \underset{\alpha}{\times} G$, for A and G also being possibly iterated products. Here $A \underset{\alpha}{\times} G$ denotes a single central extension or a single semidirec product, as it is the case. This way, a homological model for G has been already described in the subsections above, provided some homological models for A and G are known. In fact, we do know these homological models for A and G. It suffices to iterate this scheme, until we arrive to homological models for $\mathbb{Z}$ or $\mathbb{Z}_n$.

3 The Notebook

3.1 Codifying the Group

An iterated product G of central extensions and semidirect products of abelian groups G_i, $1 \leq i \leq n$, is codified as a rooted binary tree, such that every inner vertex represents a product, their sons being the correspondent factor groups. As usual, a inner vertex contributes two sons in the level immediately below. This way, the number of leaf vertices coincides with the number of factor groups G_i. In order to obtain the group G, the inner vertices (i.e. single products) of the tree must be chosen from bottom to the top level (root vertex), from left to the right while staying at the same level.

For instance, the binary tree of Figure 1 represents an iterated product.

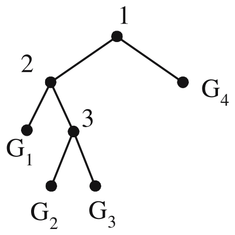


Fig. 1. A binary tree representing the product $(G_1 \times (G_2 \times G_3)) \times G_4$

We use a list for representing this binary tree, which we term *tree-list*. In fact, *Mathematica* is one of the most appropriate systems for handling with lists. Proceeding level by level, from top to the bottom, from left to the right, every vertex (but those placed on the last level) will be codified as an integer, attending to the correspondences in the table below.

Label	*Meaning*
0	leaf vertex
1	direct product
2	semidirect product $A \rtimes_\alpha G$
3	semidirect product $G_\alpha \ltimes A$
4	central extension $A \ltimes_\alpha G$
5	central extension $G_\alpha \rtimes A$

Example 1. Consider the dihedral group $D_{4t} = \mathbb{Z}_{2t} \rtimes_\alpha \mathbb{Z}_2$, for $\alpha : \mathbb{Z}_2 \to \mathrm{Aut}(\mathbb{Z}_{2t})$ such that $\alpha(0)(x) = x$ and $\alpha(1)(x) = -x$. The list codifying this group consists in $\{\{2\}\}$. There is no need to add the list $\{0, 0\}$ corresponding to the last level.

Example 2. Consider the central extension $\mathbb{Z}_{2t} \ltimes_\alpha \mathbb{Z}_2$, for $\alpha : \mathbb{Z}_2 \times \mathbb{Z}_2 \to \mathbb{Z}_{2t}$ being the 2-cocycle

$$\alpha(g_i, g_j) = \begin{cases} \lceil \frac{t}{2} \rceil + 1 & \text{if } g_i = g_j = 1 \\ 0 & \text{otherwise} \end{cases}$$

The list codifying this group consists in $\{\{4\}\}$. Once again, we do not take into account the list $\{0, 0\}$ corresponding to the last level.

Example 3. Consider the iterated product $(\mathbb{Z}_t \ltimes_{\alpha_2} \mathbb{Z}_2) \rtimes_{\alpha_1} \mathbb{Z}_2$, for α_2 being the 2-cocycle $\alpha_2 : \mathbb{Z}_2 \times \mathbb{Z}_2 \to \mathbb{Z}_t$ defined as

$$\alpha_2(g_i, g_j) = \begin{cases} \lceil \frac{t}{2} \rceil + 1 & \text{if } g_i = g_j = 1 \\ 0 & \text{otherwise} \end{cases}$$

and α_1 being the dihedral action

$$\alpha_1(a, b) = \begin{cases} -b & \text{if } a = 1 \\ b & \text{if } a = 0 \end{cases}$$

The list codifying this group consists in $\{\{2\}, \{4, 0\}\}$.

3.2 Codifying the Homological Models

The homological models described in the precedent section involve several structures, such as:

- Linear combinations.
- Products of exterior and divided power algebras.
- Elements in a cartesian simplicial product $X \times Y$, linear combinations of tuples $((x_1, y_1), \ldots, (x_n, y_n))$.
- Elements in a tensor product $X \otimes Y$, linear combinations of tuples $(x_1, \ldots, x_p) \otimes (y_1, \ldots, y_q)$.
- Formal series.

We now describe the way in which these structures are codified.

- The elimination of the attribute `Listable` on the addition and product functions provided by *Mathematica* supplies at once the possibility of making linear combinations with lists.
- Each of the groups $G_i = \mathbb{Z}_{n_i}$ gives rise to a product $P_i = E(u_i) \otimes \Gamma(w_i)$, whereas each group $G_j = \mathbb{Z}$ gives rise to a single $P_j = E(u_j)$. The elements in $hG = \prod_{i=1}^{n} P_i$ are codified as linear combinations of tuples of length n, such that if m is the j^{th} entry of a tuple, it refers to the generator $u^{m\ (mod\, 2)} \otimes \gamma_{\lfloor \frac{m}{2} \rfloor}(w)$ (notice that only 0 and 1 entries are permitted in case of $P_j = E(u_j)$ factors coming from $G_j = \mathbb{Z}$). These tuples are ordered as numbers of n digits. For instance, if $G = \mathbb{Z}_2 \times \mathbb{Z} \times \mathbb{Z}_2$, a basis for hG on degree 3 is given by $\{(0,0,3),(0,1,2),(1,0,2),(1,1,1),(2,0,1),(2,1,0),(3,0,0)\}$.
- Tuples are codified as lists, in a natural way.
- Formal series, which are always finite when applied on a concrete element, are codified in terms of the command `NestList`.

3.3 Calculating the Homology

As Veblen's algorithm indicates [22], in order to compute the homology $H_i(G)$ it is necessary to calculate the Smith's normal forms of the matrices M_i and M_{i+1} representing the differential operators d_i and d_{i+1}.

Our program firstly computes the matrices M_i and M_{i+1}. Afterwards, we use the `SmithNormalForm.na` package [14] and finally compute $H_i(G)$. Though the actual version of the notebook does not provide representative i-cycles, it could be straightforwardly adapted to this end. In fact, such an option was available in an earlier (not published) version of the notebook, which ran only over finite groups.

3.4 Input and Output Data

In these circumstances, we may now determine exactly what the input and output data are.

Input Data:

- The correspondent tree-list for the group G.
- The cardinalities of each elementary factor group $\mathbb{Z}$ or $\mathbb{Z}_n$. The notation is 0 for $\mathbb{Z}$ and $n > 1$ for $\mathbb{Z}_n$. Notice that k factor groups correspond to $k-1$ products, and vice versa.
- The maps α_i involved in the i^{th} product, $J_i \underset{\alpha_i}{\times} K_i$. The user should attend to the tree-list for the group G, in order to identify the index i corresponding to each product, as well as the syntax to use for codifying the elements in J_i and K_i, since they could be in turn iterated groups themselves.
- Finally, the desired degree k, in order to compute H_k.

Output data:

– The homology $H_k(G)$.

As soon as the computation has finished, the user is asked for going on computing H_{k+1}, since half of the computations (those corresponding to d_{k+1}) may be reused.

4 Simplifications on the Formulae

Calculations achieved with our program have provided some evidences of annihilation properties on some summands on the maps characterizing the homological models of the precedent sections. We include here the results that we have finally proved.

Proposition 1. *In the case of a homological model for a semidirect product, the morphism SHI_δ reduces to SHI, as well as EML_δ reduces to EML, and AW_δ reduces to $AW - AW\delta SHI$.*

Proposition 2. *In the case of a homological model for a semidirect product, the morphism $t\cap$ reduces to $t \cap ([a_{m-1}, \dots, a_0] \otimes [g_{n-1}, \dots, g_0]) =$ $(-1)^m([\alpha(g_{n-1})(a_{m-1}), \dots, \alpha(g_{n-1})(a_0)] - [a_{m-1}, \dots, a_0]) \otimes [g_{n-2}, \dots, g_0]$*

5 Executions and Examples

We compute here the matrices M_2 and M_3 corresponding to d_2 and d_3, as well as the homology groups H_1 and H_2 of some finite groups. These and other examples have provided essential information in order to calculate the total number of cocyclic Hadamard matrices on the correspondent groups, some of which seems to be new [2].

Example 4. Consider the family of groups $\mathbb{Z}_{2t} \ltimes_\alpha \mathbb{Z}_2$, for $t \in \mathbb{N}$ and α being the 2-cocycle $f(1,1) = \lceil \frac{t}{2} \rceil + 1$.

t	1	2	3	4	5
M_2	$\begin{pmatrix} 2 & 0 \\ 0 & 0 \\ 0 & 2 \end{pmatrix}$	$\begin{pmatrix} 2 & 2 \\ 0 & 0 \\ 0 & 4 \end{pmatrix}$	$\begin{pmatrix} 2 & 3 \\ 0 & 0 \\ 0 & 6 \end{pmatrix}$	$\begin{pmatrix} 2 & 5 \\ 0 & 0 \\ 0 & 8 \end{pmatrix}$	$\begin{pmatrix} 2 & 6 \\ 0 & 0 \\ 0 & 10 \end{pmatrix}$
M_3	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 & 0 \\ 0 & -2 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 3 & 0 \\ 0 & -2 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 5 & 0 \\ 0 & -2 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 6 & 0 \\ 0 & -2 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
H_1	$\mathbb{Z}_2 \oplus \mathbb{Z}_2$	$\mathbb{Z}_2 \oplus \mathbb{Z}_4$	$\mathbb{Z}_{12}$	$\mathbb{Z}_{16}$	$\mathbb{Z}_2 \oplus \mathbb{Z}_{10}$
H_2	$\mathbb{Z}_2$	$\mathbb{Z}_2$	0	0	$\mathbb{Z}_2$

Example 5. Consider the dihedral groups $D_{4t} = \mathbb{Z}_{2t} \rtimes_{\chi} \mathbb{Z}_2$, for $1 \leq t \leq 5$.

t	1	2	3	4	5
M_2	$\begin{pmatrix} 2 & 0 \\ 0 & 0 \\ 0 & 2 \end{pmatrix}$	$\begin{pmatrix} 2 & 0 \\ 0 & -2 \\ 0 & 4 \end{pmatrix}$	$\begin{pmatrix} 2 & 0 \\ 0 & -4 \\ 0 & 6 \end{pmatrix}$	$\begin{pmatrix} 2 & 0 \\ 0 & -6 \\ 0 & 8 \end{pmatrix}$	$\begin{pmatrix} 2 & 0 \\ 0 & -8 \\ 0 & 10 \end{pmatrix}$
M_3	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & -4 & -2 \\ 0 & 4 & 2 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & -6 & -4 \\ 0 & 6 & 4 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & -8 & -6 \\ 0 & 8 & 6 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & -10 & -8 \\ 0 & 10 & 8 \\ 0 & 0 & 0 \end{pmatrix}$
H_1	$\mathbb{Z}_2 \oplus \mathbb{Z}_2$	$\mathbb{Z}_2 \oplus \mathbb{Z}_2$	$\mathbb{Z}_2 \oplus \mathbb{Z}_2$	$\mathbb{Z}_2 \oplus \mathbb{Z}_2$	$\mathbb{Z}_2 \oplus \mathbb{Z}_2$
H_2	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2$

Example 6. Consider the family of iterated products $(\mathbb{Z}_t \ltimes_f \mathbb{Z}_2) \rtimes_{\chi} \mathbb{Z}_2$, for $2 \leq t \leq 5$, f being the 2-cocycle $f(1,1) = \lceil \frac{t}{2} \rceil + 1$ and χ being the dihedral action $\chi(1,b) = -b$.

t	2	3	4	5
M_2	$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 2 \end{pmatrix}$	$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 3 \end{pmatrix}$	$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 2 & 1 \\ 0 & 0 & -2 \\ 0 & 0 & 0 \\ 0 & 0 & 4 \end{pmatrix}$	$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 2 & 1 \\ 0 & 0 & -3 \\ 0 & 0 & 0 \\ 0 & 0 & 5 \end{pmatrix}$
M_3	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -3 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 3 & 0 & 1 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & -1 & 0 & -1 \\ 0 & 2 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -4 & 0 & -2 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 4 & 0 & 2 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & -1 & 0 & -1 \\ 0 & 2 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -5 & 0 & -3 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 5 & 0 & 3 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$
H_1	$\mathbb{Z}_2 \oplus \mathbb{Z}_2 \oplus \mathbb{Z}_2$	$\mathbb{Z}_2 \oplus \mathbb{Z}_2$	$\mathbb{Z}_2 \oplus \mathbb{Z}_2$	$\mathbb{Z}_2 \oplus \mathbb{Z}_2$
H_2	$\mathbb{Z}_2 \oplus \mathbb{Z}_2 \oplus \mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2$

References

1. V. Álvarez. http://mathworld.wolfram.com/HomologyIteratedGroups.html (2006). To appear.
2. V. Álvarez, J.A. Armario, M.D. Frau and P. Real. A genetic algorithm for cocyclic Hadamard matrices. *AAECC-16 Proceedings*, LNCS **3857**, 144–153, (2006).
3. V. Álvarez, J.A. Armario, M.D. Frau and P. Real. (Co)homology of iterated products of semidirect products of abelian groups. Preprint. Sent to the 9th International Workshop on Computer Algebra in Scientific Computing, CASC-06, Moldavia, (2006).

4. V. Álvarez, J.A. Armario, M.D. Frau and P. Real. Comparison maps for relatively free resolutions. Preprint. Sent to the 9th International Workshop on Computer Algebra in Scientific Computing, CASC-06, Moldavia, (2006).
5. J. Carlson, http://www.math.uga.edu/ jfc/
6. X. Dousson, J. Rubio, F. Sergeraert and Y. Siret. The KENZO program. Institute Fourier, Grenoble (19998). http://www-fourier.ujf-grenoble.fr/˜sergeraert/Kenzo/
7. J.G. Dumas, F. Heckenbach, D. Saunders and V. Welker. Computing simplicial homology based on efficient smith normal form algorithms. *Algebra, Geometry and Software Systems*, 177–204. Springer, Heidelberg (2003).
8. S. Eilenberg and S. Mac Lane. On the groups $H(\pi, n)$ II. *Annals of Math.*, **66**, 49–139, (1954).
9. The GAP group, 'GAP- Group, Algorithms and programming', School of Mathematical and Computational Sciences, University of St. Andrews, Scotland (1998).
10. J. Grabmeier and L.A. Lambe. Computing Resolutions Over Finite p-Groups. *Proceedings ALCOMA'99*, Eds. A. Betten, A. Kohnert, R. Lave, A. Wassermann, Springer Lecture Notes in Computational Science and Engineering, Springer-Verlag, Heidelberg (2000).
11. G. Ellis. GAP package HAP, 'Homological Algebra Programming', http://hamilton.nuigalway.ie/Hap/www/
12. J. Huebschmann. Cohomology of nilpotent groups of class 2. *J. Algebra*, **126**, 400–450, (1989).
13. J. Huebschmann. Cohomology of metacyclic groups. *Transactions of the American Mathematical Society*, **328**, n. 1, 1–72, (1991).
14. D. Jabon. http://mathworld.wolfram.com/SmithNormalForm.html (1994).
15. L.A. Lambe. Algorithms for the homology of nilpotent groups. *Conf. on applications of computers to Geom. and Top., Lecture Notes in Pure and Applied Math.*, **114**, Marcel Dekker Inc., N.Y., (1989).
16. L.A. Lambe. Homological perturbation theory, Hochschild homology and formal groups. *Proc. Conference on Deformation Theory and Quantization with Applications to Physics, Amherst, MA, June 1990, Cont. Math.*, **134**, 183–218, (1992).
17. L.A. Lambe and J.D. Stasheff. Applications of perturbation theory to iterated fibrations. *Manuscripta Math.*, **58**, 367–376, (1987).
18. The MAGMA computational algebra system, http://magma.maths.usyd.edu.au
19. P. Real. Homological Perturbation Theory and Associativity. *Homology, Homotopy and Applications*, **2**, 51–88, (2000).
20. J. Rubio. Integrating functional programming and symbolic computation. *Mathematics and computers in simulation*, **44**, 505-511, (1997).
21. F. Sergeraert. The computability problem in Algebraic Topology. *Advances in Math.*, **1104**, 1–29 (1994).
22. O. Veblen. Analisis situs. A.M.S. Publications, **5** (1931).

GCLC — A Tool for Constructive Euclidean Geometry and More Than That

Predrag Janičić

Faculty of Mathematics, University of Belgrade
Studentski trg 16, 11 000 Belgrade, Serbia
`janicic@matf.bg.ac.yu`

Abstract. We present GCLC /WINGCLC— a tool for visualizing geometrical (and not only geometrical) objects and notions, for teaching/studying mathematics, and for producing mathematical illustrations of high quality. GCLC uses a language GC for declarative representation of figures and for storing mathematical contents of visual nature in textual form. In GCLC, there is a build-in geometrical theorem prover which directly links visual and semantical geometrical information with deductive properties and machine–generated proofs.

1 Introduction

GCLC is a tool for visualizing objects and notions of geometry and other fields of mathematics (by generating figures and animations). It can be used for producing digital mathematical illustrations, for teaching and studying geometry (and not only geometry), and for storing visual mathematical contents in textual form — as figure descriptions in the GC language. GCLC provides easy-to-use support for many geometrical constructions, isometric transformations, and conics. The basic idea behind GCLC is that constructions are formal procedures, rather than drawings. Thus, in GCLC, producing mathematical illustrations is based on "describing figures" rather than of "drawing figures".[1] This approach stresses the fact that geometrical constructions are abstract, formal procedures and not figures. A figure can be generated on the basis of abstract description, in Cartesian model of a plane. A similar approach is used for illustrations for other supported fields. Figures can be displayed and exported as LaTeX files or bitmaps.

Although GCLC was initially built as a tool for converting formal descriptions of geometric constructions into LaTeX form (hence its name "Geometry Constructions $\rightarrow$ LaTeX Converter"), now it is much more than that. For instance, there is support for symbolic expressions, for drawing parametric curves, for program loops, etc; WINGCLC, a version with a Windows graphical interface, makes GCLC a dynamic geometry tool with a range of additional functionalities; a built-in geometry theorem prover can automatically prove a range of complex

[1] In a sense, this system is in spirit close to the TeX/LaTeX system [14,16], or is parallel to it. Within the TeX/LaTeX system, the author (explicitly) describes the layout of his/her text.

© Springer-Verlag Berlin Heidelberg 2006

theorems, etc. GCLC now links semantic information about a construction with its visual representation and with its deductive properties. Thus, it provides mathematical contents directly linked to visual information and supported by machine–generated proofs.

GCLC is under constant development from 1996. Some features of graphical interface of WinGCLC are presented in [13], some educational aspects of GCLC are presented in [7], the built-in theorem prover is described in [23,11], and the mathematical contents management issues are discussed in [24]. This paper is the first general overview of the system.

Overview of the Paper: The rest of the paper is organized as follows: in Section 2 we focus on formal geometrical constructions and illustrate the need for describing mathematical illustrations rather then drawing them; in Section 3 we give a brief overview of the language of the GCLC system; in Section 4 we describe basic features of the graphical interface; in Section 5 we describe the built-in geometry theorem prover; in Section 6 we present several examples, illustrating different features of GCLC; in Section 7 we briefly discuss applications of GCLC in producing mathematical illustrations, in storing mathematical contents of visual nature, and in teaching mathematics; in Section 8 we discuss some technical issues and give availability information; in Section 9 we give a short overview of the systems related to GCLC; in Section 10 we discuss potential directions for further work and in Section 11 we draw final conclusions. In Section A we give some additional examples.

2 Describing Formal Constructions

Geometrical constructions are the main area of GCLC. This type of mathematical problems is very relevant for the need to describe, and not draw images.

A geometrical construction is a sequence of specific, primitive construction steps. These primitive construction steps are also called *elementary constructions* and they are:

– construction (by *ruler*) of a line such that two given points belong to it;
– construction of a point such that it is the intersection of two lines (if such a point exist);
– construction (by *compass*) of a circle such that its center is one given point and such that the second given point belongs to it;
– construction (by *compass*) of a segment connecting two points;
– construction of intersections between a given line and a given circle (if such points exist).

By using the set of primitive constructions, one can define more involved, compound constructions (e.g., construction of right angle, construction of the segment midpoint, construction of the segment bisector etc.). In describing geometrical constructions, it is usual to use higher level constructions as well as the primitive ones.

GCLC follows the idea of formal constructions. It provides easy-to-use support for all primitive constructions, but also for a range of higher-level constructions. (Although motivated by the formal geometrical constructions, GCLC provides a support for some non-constructible objects too — for instance, in GCLC it is possible to determine/use a point obtained by rotation for 1°, although it is not possible to construct that point by ruler and compass).

There is a need of distinguishing abstract (i.e., formal, axiomatic) nature of geometrical objects and their semantics and usual models. A geometrical construction is a mere procedure of abstract steps and not a picture. However, for each (Euclidean) construction, there is its counterpart in the standard Cartesian model. While a construction is an abstract procedure, in order to make its usual representation in Cartesian plane (or, more precisely, in Cartesian model of Euclidean plane), one still has to make a link between these two. For instance, given three vertices of a triangle, one can construct a center of its circumcircle, but in order to visualize and represent this construction in Cartesian plane, he/she has to take three particular Cartesian points as vertices of the triangle (see the example given in Fig. 1). Thus, figure descriptions in GCLC are usually made by a list of definitions of several (usually very few) fixed points (defined in terms of Cartesian plane, i.e., by pairs of coordinates) and a list of construction steps based on these points.

3 GC Language

In GCLC, figures are described in the GC language. GC syntax is very simple, but, at the same time, it enables describing very complex figures in very few lines. Describing images in the GC language does not require programming skills. Descriptions via GC commands directly reflect mathematical objects to be visualized and are easily understandable to mathematicians. Therefore, GC is a higher-level language (with support for a number of advanced geometrical concepts) designed for mathematicians, and not a machine-oriented script language.

GC language consists of the following groups of commands (examples for different groups of commands are given in Section 6):

Basic definitions: these commands include commands for defining fixed points, for defining a line on the basis of two selected points, defining a circle, a numerical constant etc.

Basic constructions: these constructions include constructions of intersection points for two lines, and for a line and a circle, construction of the midpoint of a given segment, the bisector of an angle, the segment bisectors, perpendicular lines, parallel lines, etc.

Transformations: these commands include commands for translation, rotation, line-symmetry, half-turn, but also some non-isometric transformations like scaling, circle inversion etc.

Commands for calculations, expressions, and loops: there are commands for calculating angles determined by triples of points, distances between points, for generating random numbers, for calculating symbolic expressions and support for *while*-loops.

Drawing commands: there are commands for drawing lines, segments, circles, arcs, and ellipses in several modes.

Labelling and printing commands: points can be labelled, marked in a number of ways. In addition, a text can be attached to a particular point.

Cartesian commands: this group of commands provides support for direct access to a user–defined Cartesian system. A user can define a system, its unit, and, within it, he/she can define points, lines, conics, tangents etc. and can also draw curves given in parametric form.

Low level commands: there is support for changing line thickness, color, clipping area, figure dimensions etc.

Commands for describing animations: this group of commands provides support for making animations within Wingclc. Some points can be defined to move from one position to another; points can also be traced.

Commands for the geometry theorem prover: using support for the built-in geometry theorem prover, the user can provide the conjecture, can control a proof level and can limit a maximal number of proof steps.

4 Graphical Interface

Wingclc provides a range of interactive functionalities. In addition to tools for processing picture descriptions and locating errors, tools (*watch window*) for monitoring values of selected objects in a construction (so Wingclc can work as a *geometrical calculator*), there are also tools for easy and interactive moving of fixed points, updating pictures and making animations. (Animations and traced points can be defined both interactively and via gclc commands.) These interactive features can be very useful in teaching geometry, but can also help studying geometry or even help some research (with Wingclc serving as a machine assistant). Figure 5 illustrates some of the mentioned tools and devices (traces, animations, *watch windows*, etc.)

5 Theorem Prover

Automated theorem proving in geometry has two major lines of research: synthetic proof style and algebraic proof style (see, for instance, [18] for a survey). Algebraic proof style methods are based on reducing geometric properties to algebraic properties expressed in terms of Cartesian coordinates. These methods are usually very efficient, but the proofs they produce do not reflect the geometric nature of the problem and they give only a *yes* or *no* conclusion. Synthetic methods attempt to automate traditional geometry proof methods.

The geometry theorem prover built into GCLC is based on the area method [3,4,23].[2] This method belongs to the group of synthetic methods. It produces traditional, human-readable proofs, with a clear justification for each proof step. The main idea of the method is to express hypotheses of a theorem using a set of constructive statements, each of them introducing a new point, and to express a conclusion by an equality of expressions in *geometric quantities* (e.g., signed area of a triangle), without referring to Cartesian coordinates. The proof is then based on eliminating (in reverse order) the points introduced before, using for that purpose a set of appropriate lemmas. After eliminating all introduced points, the current goal becomes a trivial equality that can be simply tested for validity. In all stages, different expression simplifications are applied to the current goal. The method does not have any branching, which makes it very efficient. A wide range of geometric conjectures can be simply stated within GCLC and proved by the prover.

The prover is tightly integrated in GCLC. This means that one can use the prover to reason about a GCLC construction (i.e., about objects introduced in it), without changing and adapting it for the deduction process — the user only needs to add the conclusion he/she wants to prove. The proofs are generated in LaTeX form. For more details about the prover, see [23,11].

6 Examples

Geometrical Constructions. The example given in Fig. 1 illustrates one simple geometrical construction. Groups of commands are explained by comments (marked by the symbol `%`) within the description itself. As many other similar descriptions, this one has basically three parts (not necessarily separated): one with defining fixed points (with coordinates in Cartesian plane), one with construction steps, and one with labelling and drawing commands. By changing one of the three fixed points, the whole of the illustration is updated. In this example, three side bisectors of the triangle ABC are constructed. It is a simple fact that these three lines intersect at one point (at the center of the circumcircle). This can be also stated in the following form: pairwise intersections of the side bisectors, the points `O_1` and `O_2`, are identical. This property (as well as much more complex properties or hypotheses) can be, in a sense, explored within GCLC. Namely, `d` is defined to be the distance between `O_1` and `O_2`, and one can monitor the value of `d` to ensure that it is equal to zero (for these and for any other three particular vertices).

Cartesian Commands. Example given in Fig. 2 illustrates the support for a direct access to a user-defined Cartesian system. In this example, there is a description of one conic (parabola), via its canonical parameters, and one its tangent. This example also illustrates how a rather complex figure can be described in only a few lines.

[2] The theorem prover is developed in collaboration with prof. Pedro Quaresma from University of Coimbra.

```
% fixed points
point A 10 10
point B 50 10
point C 40 50

% side bisectors
med a B C
med b A C
med c B A

% intersections of bisectors
intersec O_1 a b
intersec O_2 a c
distance d O_1 O_2

% marking points
cmark_b A
cmark_b B
cmark_t C
cmark_lt O_1
cmark_rt O_2

% drawing the sides of the triangle ABC
drawsegment A B
drawsegment A C
drawsegment B C

% drawing the circumcircle of a triangle
drawcircle O_1 A
```

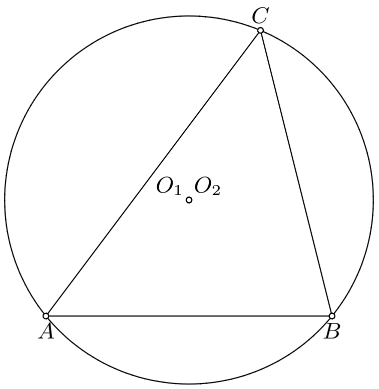


Fig. 1. Example of a GC description of a geometrical construction (left) and the corresponding (LaTeX) output (right)

Parametric Curves. Example given in Fig. 3 illustrates the support for parametric curves. The first curve, is drawn for parameter x ranging from -3 to 4, increased by the step 0.05.

While-loops. Example given in Fig. 4 illustrates while-loops. The construction described within this example shows that, for any line segment AB, the locus of all points L such that the angle ALB is right angle, is the circle with the perimeter AB. The point B is rotated (giving the point B') around the point A for the angle *phi* ranging from $0°$ to $70°$, and the point L is determined as a foot of the perpendicular from B to AB'. Points L for different values of *phi* are connected by line segments.

Animations. An animation in WINGCLC is defined as a formal construction with a set of fixed points that linearly move from an initial to a destination position. All positions of one selected point make *trace* (similar to *locus*), drawn in a selected color. The *watch window* is used for monitoring values of objects

```
% define and draw Cartesian axis
ang_picture 5 5 55 55
ang_origin 20 20
ang_drawsystem

% define a conic
ang_conic h 0 0 1 -1 0 -3

% construct a point P on the conic
% and the tangent in P
ang_point A1 2 2
ang_point A2 3 2
line l A1 A2
ang_intersec2 P P2 h l
ang_tangent p P h

% draw the conic and the tangent
cmark_t P
ang_drawline p
ang_drawconic h
```

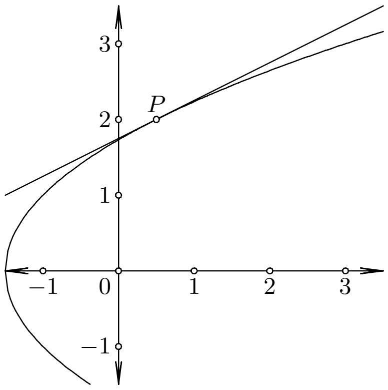


Fig. 2. Illustration for Cartesian commands

```
ang_picture 2 2 58 58
ang_origin 25 25
ang_unit 7
ang_drawsystem_a

ang_draw_parametric_curve x
        {-3; x<4; x+0.05}
        { x; sin(pow(x,2))*cos(x) }

% polar coordinates
number rho 2
ang_draw_parametric_curve phi
        { 0 ; phi<6; phi+0.1}
        { phi*rho*sin(phi)/5 ;
rho*cos(phi) }
```

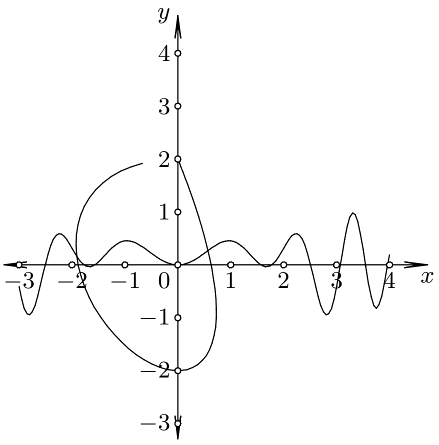


Fig. 3. Illustration for parametric curves

used in the construction. The screenshot shown in Fig. 5 illustrates some of the features and devices of WINGCLC (traces, animations, *watch windows*, etc.)

Theorem Prover. For the example shown in Figure 1, it can be checked that for any particular three points `A`, `B`, and `C`, the points `O_1` and `O_2` (pairwise intersections of the side bisectors) are identical. Using the prover, one can ensure

```
point A 5 5
point B 50 5

cmark_b A
cmark_b B

drawsegment A B
translate L_old B B B

number phi 0
while { phi<=70 }
{
       rotate B' A phi B
       line a B' A
       foot L B a

       drawsegment L L_old
       translate L_old L L L

       expression phi { phi+1 }
}

cmark_lt B'
drawdashsegment A B'
drawdashsegment B L
```

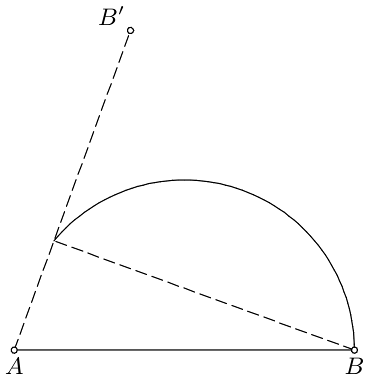


Fig. 4. Illustration for while-loops

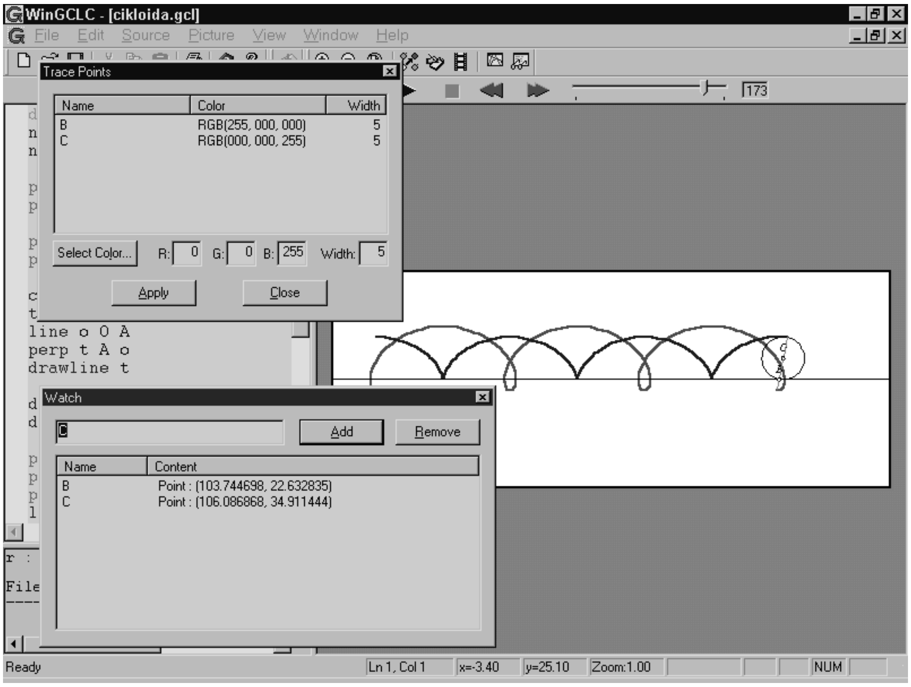
that this is valid statement, i.e., the distance between points `O_1` and `O_2` is always equal to zero. This statement can be given to the prover by simply adding the following line:

```
prove { equal { pythagoras_difference3 O_1 O_2 O_1 } 0 }
```

to the code given in Figure 1. The conjecture is stated within the command `prove` and via the geometric quantity *Pythagoras difference* (P_3). By definition, $P_3(A, B, C) = AB^2 + CB^2 - AC^2$, hence, the value $P_3(\texttt{O\_1},\texttt{O\_2},\texttt{O\_1})$ is equal to 0 if and only if the points `O_1` and `O_2` are identical (for more details, see [23]). The proof is exported to a LaTeX file, with explanations for each proof step. Figure 6 shows last steps of the proof made by the prover (the proof consists of 119 steps and it took 0.035 seconds of CPU time). This example illustrates how GCLC provides geometrical contents directly linked to visual information and supported by machine–generated proofs.

7 Applications

In this section we briefly discuss three main fields of application for GCLC: in producing mathematical illustrations, for storing mathematical contents, and in teaching mathematics.



Name	Color	Width
B	RGB(255, 000, 000)	5
C	RGB(000, 000, 255)	5

R: 0 G: 0 B: 255 Width: 5

```
line o O A
perp t A o
drawline t
```

Name	Content
B	Point : (103.744698, 22.632835)
C	Point : (106.086868, 34.911444)

Ln 1, Col 1 x=-3.40 y=25.10 Zoom:1.00 NUM

Fig. 5. *Trace* and *watch* windows with cycloid described in WinGCLC

(113)	$(0.062500 \cdot (P_{CBC} \cdot S_{BAC})) = \left(\frac{1}{4} \cdot \left(P_{CBM_a^0} \cdot S_{BAM_a^0}\right)\right)$	, by algebraic simplifications
(114)	$(0.062500 \cdot (P_{CBC} \cdot S_{BAC})) = \left(\frac{1}{4} \cdot \left(\left(P_{CBB} + \left(\frac{1}{2} \cdot (P_{CBC} + (-1 \cdot P_{CBB}))\right)\right) \cdot S_{BAM_a^0}\right)\right)$	, by Lemma 29 (point M_a^0 eliminated)
(115)	$(0.062500 \cdot (P_{CBC} \cdot S_{BAC})) = \left(\frac{1}{4} \cdot \left(\left(0 + \left(\frac{1}{2} \cdot (P_{CBC} + (-1 \cdot 0))\right)\right) \cdot S_{BAM_a^0}\right)\right)$	, by geometric simplifications
(116)	$(0.062500 \cdot S_{BAC}) = \left(\frac{1}{8} \cdot S_{BAM_a^0}\right)$	, by algebraic simplifications
(117)	$(0.062500 \cdot S_{BAC}) = \left(\frac{1}{8} \cdot \left(S_{BAB} + \left(\frac{1}{2} \cdot (S_{BAC} + (-1 \cdot S_{BAB}))\right)\right)\right)$	, by Lemma 29 (point M_a^0 eliminated)
(118)	$(0.062500 \cdot S_{BAC}) = \left(\frac{1}{8} \cdot \left(0 + \left(\frac{1}{2} \cdot (S_{BAC} + (-1 \cdot 0))\right)\right)\right)$	, by geometric simplifications
(119)	$0 = 0$	, by algebraic simplifications

28

Fig. 6. Last steps of the proof of the Circumcircle theorem

7.1 Producing Digital Illustrations

GCLC can serve as a tool for making digital illustrations of high quality. Descriptions made in GC language can be (internally) visualized or can be converted into some other format — LaTeX or bitmap format. Figures in LaTeX format produced by GCLC can be included directly in LaTeX documents, hence they use LaTeX fonts and formulae which is essential for good looking figures in LaTeX documents (while this is a problem for many other formats and tools). Pictures in bitmap format are suitable for different conversions and processing. Although these two picture formats (LaTeX and bitmap) have their advantages, GCLC figures are normally stored in their original, source form. This form is not only precise and sufficient for producing pictures, but also very concise: for instance, all figures from a university book with 120 illustrated geometrical problems [12] have together (in uncompressed, GCLC form) less than 130Kb. GCLC has been used for producing illustrations for a number of other books and articles.

7.2 Storing Mathematical Contents

We advocate for *describing* (rather than *drawing*) mathematical illustrations. Descriptions of images should be given in formal, but human-readable, easily understandable language, close to the intended mathematical meaning. Mathematical illustrations should be stored in such form (rather than in the form of images).

A lot of mathematical contents, both in education and in research, is of visual nature. In many (or most) lecture notes, books, and research articles there are mathematical illustrations. They carry mathematical information, some mathematical message that is represented visually rather than in textual or numerical form. Usually, such message is better understandable to a reader when represented visually. On the other hand, this visual information is usually not mathematically rigorous; it is usually approximation and/or interpretation of some mathematical objects, notions, concepts, numerical data, proofs, ideas etc. A reader interprets the visual information and in his/her mind creates a formal mathematical information. It is often assumed that the reader (with a support from the given textual explanations, earlier experience with illustrations, standard mathematical background, intuition, etc) can understand, “read” the right mathematical message from the illustration. Although a mathematical message is carried by an illustration, that message cannot always be reproduced from the illustration itself. In addition, a mathematician, the author or a reader of a mathematical text, may need to alter an image, to modify some of its characteristics (not only characteristics such as dimensions, but rather some characteristics implying the mathematical contents), to make it more general or more specific, and also to store it in a way that enables these sorts of transformations.

Let us consider geometrical illustrations: a complex geometrical construction may be illustrated by an image and can indeed make the understanding of the text easier. However, without a given context, without provided textual explanations of the problem, it is unlikely that one could guess the right nature and description of the construction. In addition, the Cartesian interpretation of Euclidean geometry is just one of possible interpretations and, hence, potentially misses some of the abstract geometrical meaning. The main point is: an image itself does not provide precise geometrical message. It is better to have a figure description that is formal and can be used for producing the required image, required mathematical illustration. Such information should be stored instead of images. Mathematical content stored in this way (via formal descriptions) is easy to understand, maintain, modify and process in different ways.

The program GCLC is developed along the lines of the given motivation. It is based on using the GC language, in which one can describe a number of geometrical constructions, but also other mathematical objects. Figure descriptions are declarative, precise and brief descriptions of mathematical content and from them corresponding illustrations can be generated. This way, GCLC can be seen as a tool for storing mathematical contents of visual natura in textual form.

7.3 Using GCLC in Teaching Mathematics

In teaching and studying geometry, students can interactively use GCLC to make different attempts in making constructions and/or exploring some mathematical (especially geometrical) objects, notions, ideas, problems, proofs, properties etc [7]. Formally describing mathematical objects is similar to programming, so this helps computer science students to better understand geometry notions and mathematics students to get familiar with programming. Interactive work makes this sort of studying more interesting and more fruitful. The built-in theorem prover can help students link semantic and deductive aspects of geometry.

Producing mathematical images and teaching/studying mathematics are not far from each other. For instance, within the geometry courses at the Faculty of Mathematics, University of Belgrade, prof. Zoran Lučić with his students made

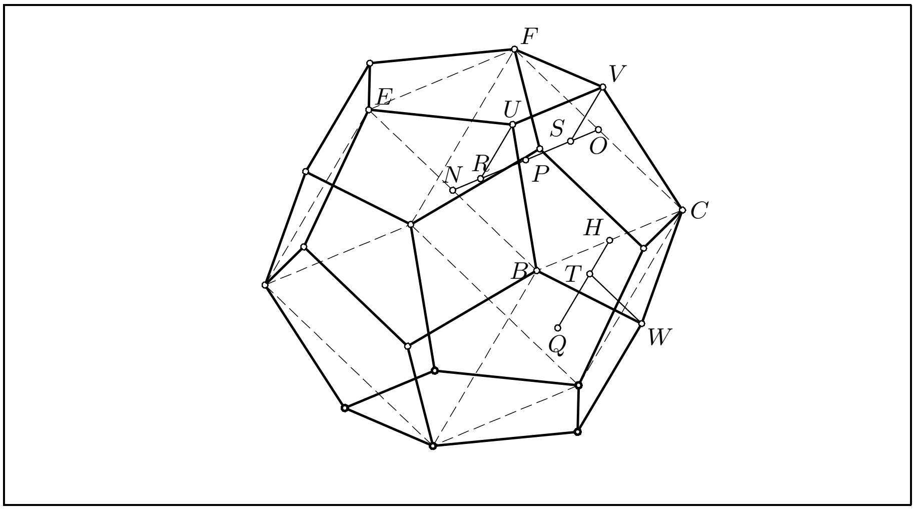


Fig. 7. Illustration for Euclid's construction of dodecahedron

an electronic version of Euclid's masterpiece of the classical mathematics — *The Elements* [17]. All figures in the book are described in the GC language and directly reflect the accompanying geometrical text. This is probably the first edition of *The Elements* that includes formal, rigorous description of all images, descriptions that directly reflect the accompanying mathematical text. Figure 7 shows Lučić's detailed illustration for Euclid's construction of dodecahedron. This figure is made by using the means of descriptive geometry and shows that three-dimensional objects can be also formally describe in GC language, despite the fact that it is basically designed for plane geometry.

8 Technical Issues, Versions and Availability

GCLC /WinGCLC programs are implemented in C++ programming language. The basic, command line version has around 18000 lines of code. GCLC programs are very small in size: the command line version of GCLC has around 350Kb, WinGCLC has around 700Kb.

There are command-line versions of GCLC for Windows and for Linux. WinGCLC is the version of GCLC for Windows, with a graphical user-friendly interface. As yet, there is no version with a graphical user interface for Linux.

GCLC package (with a manual file and sample files) is freely available from `www.matf.bg.ac.yu/~janicic/gclc/` and from EMIS (The European Mathematical Information Service) servers (`www.emis.de/misc/index.html`).

Figures in LaTeX format generated by GCLC are included in a LaTeX documents by the `\input` command. They use a simple package `gclc.sty`, with definitions that can be changed (so, for instance, can use some particular LaTeX drawing package).

9 Related Work

GCLC /WinGCLC is related to a family of similar, dynamic geometry tools such as *Cinderella*, [25,6], *Geometer's Sketchpad*, [10,8] *Eukleides* [20], *Cabri*, [2,15], *JavaView* [22,21]. GCLC share a number of features with these tools, but also have some specific features. The main features in which GCLC /WinGCLC differs from similar tools are:

– the deduction module (that directly links visual and semantical geometrical information with deductive properties and machine–generated proofs);
– features that go beyond Euclidean geometry; for instance, GCLC can be used for easily producing figures in Cartesian plane, including graphs of functions (see Figure 8); therefore, GCLC can substitute a wide range of tools for producing mathematical (not only geometrical) illustrations.

Some of the advantages of GCLC /WinGCLC (comparing to other tools) are also its free availability, its simplicity, small size of the program, its output files natively supported by LaTeX, interactive features such as animations and

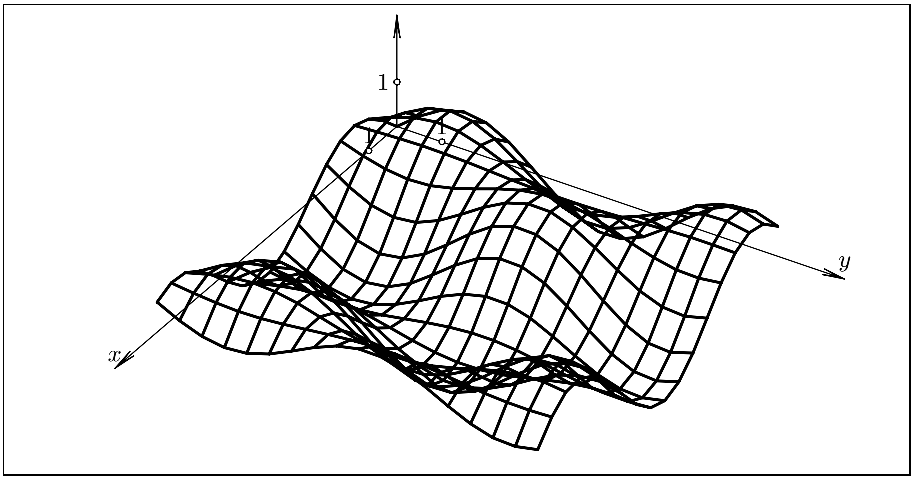


Fig. 8. Function of arity two drawn by GCLC

traces, etc. In contrast to some dynamic geometry tools, GCLC /WinGCLC focuses on explicitly and formally describing figures (instead of drawing figures) and thus, focusing on *meaning* rather than only on *layout* of figures. These explicit descriptions are easy to write and understand, and they directly reflect the mathematical meaning illustrated by figures.

There are links between GCLC and other tools: there are converters from *JavaView* `.jvx` code and from *Eukleides* code to GCLC code. This brings additional power to GCLC (by making available models created in other tools).

Concerning geometrical theorem proving, there are also some systems related to GCLC /WinGCLC. *Geometry Expert* [9] is a dynamic geometry system with algebraic based theorem prover. There are geometrical theorem provers based on the area method also within the systems Coq [19] and Theorema [1].

10 Further Work

The next version of GCLC /WinGCLC will have support for 3D Cartesian system, for plotting graphs, for exporting figures to EPS and SVG format, and for exporting figures descriptions and proofs to XML format. For future work, we also are planning to:

- implement additional geometrical theorem provers and build them into GCLC; one of the candidates is a prover based on Wu's algorithm [5];
- develop new additional modules to GCLC (for hyperbolical geometry, descriptive geometry, projective geometry etc.), so GCLC could work as a native platform for a range of geometries;

– develop new tools for linking GCLC with other mathematical software tools;
– make XML version of GC format, and link GCLC with some popular markup languages for mathematical contents.

11 Conclusions

In this paper we presented the software package GCLC /WINGCLC for visualizing mathematical objects and notions, for teaching/studying mathematics, and for producing mathematical illustrations of high quality. GCLC uses the GC language for declarative representation of figures, suitable for storing mathematical contents of visual nature. With such representation of information, the intended mathematical message and meaning of mathematical illustrations is possible to preserve and reconstruct.

After first ten years of development, GCLC is much more than a geometrical tool. There is support for symbolic expressions, for drawing parametric curves, for program loops, and WINGCLC makes GCLC an interactive, dynamic mathematical tool with a range of functionalities. The built-in geometry theorem prover can automatically prove a range of complex theorems. It links semantic information about a construction with its deductive properties. It provides mathematical contents directly linked to visual information and supported by machine–generated proofs.

The system is publicly available and is already being used by a number of mathematicians. We are planning to further improve it and extend it.

References

1. B. Buchberger et.al. Theorema: Towards computer-aided mathematical theory exploration. *Journal of Applied Logic*, 2006.
2. Cabri site. http://www.cabri.com.
3. C. C. Chou, OU X. S. Gao, and J. Z. Zhang. Automated production of traditional proofs for constructive geometry theorems. In *Eighth Annual IEEE Symposium on Logic in Computer Science*, 1993.
4. S.C. Chou, X.S. Gao, and J.Z. Zhang. *Machine Proofs in Geometry.* World Scientific, Singapore, 1994.
5. Shang-Ching Chou. *Mechanical Geometry Theorem Proving.* D.Reidel Publishing Company, Dordrecht, 1988.
6. Cinderella site. http://www.cinderella.de.
7. Mirjana Djorić and Predrag Janičić. Constructions, instructions, interactions . *Teaching Mathematics and its Applications*, 23(2):69–88, 2004.
8. Geometer's Sketchpad site. http://www.keypress.com/sketchpad/.
9. GEX site. http://woody.cs.wichita.edu/gex/7-10/gex.html.
10. Nicholas Jackiw. *The Geometer's Sketchpad v4.0.* Emeryville: Key Curriculum Press, 2001.
11. Predrag Janičić and Pedro Quaresma. System description: Gclcprover + geothms. *International Joint Conference on Automated Reasoning (IJCAR-2006).* LNAI 4130. Springer, 2006.

12. Predrag Janičić. *Zbirka zadataka iz geometrije*. Skripta Internacional, Beograd, 1st edition 1997, 6th edition 2005. Collection of problems in geometry (in Serbian).
13. Predrag Janičić and Ivan Trajković. WinGCLC — a Workbench for Formally Describing Figures. SCCG 2003, ACM Press.
14. Donald Knuth. *TeXBook*. Addison Wesley Professional, 1986.
15. J.-M. Laborde and R. Strasser. Cabri-géométre: A Microworld of Geometry for Guided Discovery Learning. *Zentrablatt Für Didactic der Matematik*, 22(5), 1990.
16. Leslie Lamport. *LaTeX: A Document Preparation System*. Addison Wesley Professional, 1994.
17. Zoran Lučić et. al. Euclid's *Elements*. http://www.matf.bg.ac.yu/nastavno/zlucic/.
18. Noboru Matsuda and Kurt Vanlehn. Gramy: A geometry theorem prover capable of construction. *Journal of Automated Reasoning*, 32:3–33, 2004.
19. Julien Narboux. A decision procedure for geometry in coq. In *TPHOLS 2004*, volume 3223 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.
20. Christian Obrecht. Eukleides. http://www.eukleides.org/.
21. Konrad Polthier et. al. JavaView. on-line at: http://http://javaview.de/.
22. K. Polthier, S. Khadem, E. Preuss, and U. Reitebuch. Publication of interactive visualizations with JavaView. *Multimedia Tools for Communicating Mathematics*. Springer, 2002.
23. P. Quaresma and P. Janičić. Framework for the Constructive Geometry. TR2006/001, Center for Informatics and Systems, University of Coimbra, 2006.
24. Pedro Quaresma and Predrag Janičić. Integrating dynamic geometry software, deduction systems, and theorem repositories. *Mathematical Knowledge Management (MKM-2006)*, LNAI 4108. Springer-Verlag, 2006.
25. Jürgen Richter-Gebert and Ulrich Kortenkamp. *Cinderella - The interactive geometry software*. Springer, 1999.

A Additional Examples

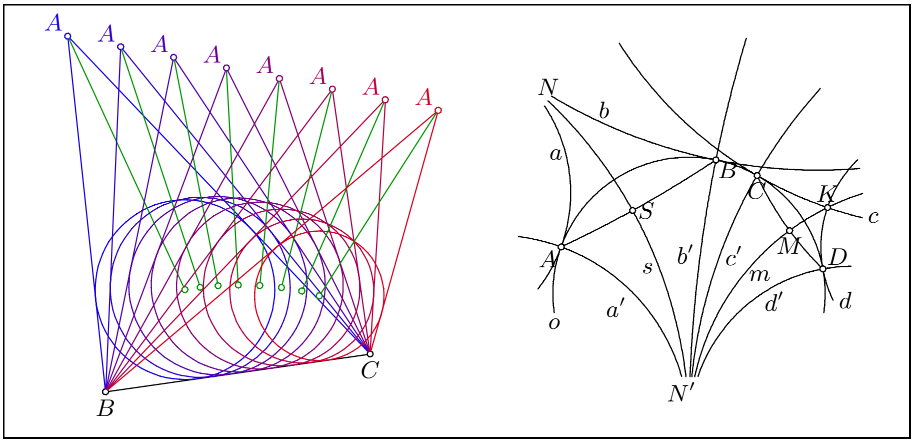


Fig. 9. Inscribed circle for different choices for the point A (left) and hyperbolical geometry figure represented in Poincare's disc model (right)

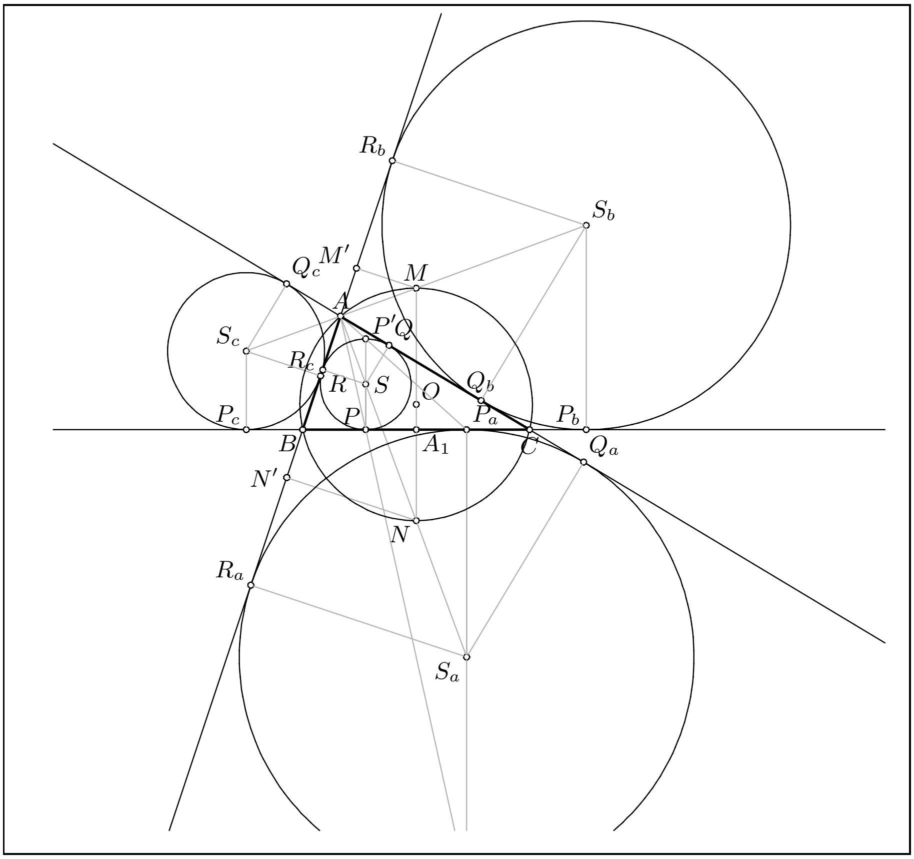


Fig. 10. Triangle with some characteristic points

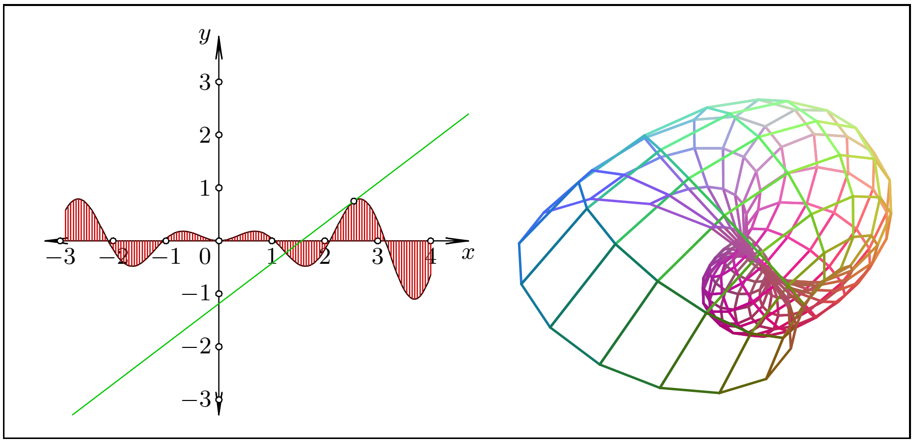


Fig. 11. Function and its derivation (left) and model imported from JavaView (right)

jReality, jtem, and Oorange — A Way to Do Math with Computers

Tim Hoffmann[1] and Markus Schmies[2]

[1] Mathematisches Institut der Universität München
Theresienstr. 39, D-80333 München, Germany
hoffmann@mathematik.uni-muenchen.de
[2] Technische Universität Berlin
Straße des 17. Juni 136, D-10623 Berlin, Germany
schmies@math.tu-berlin.de

Abstract. This paper presents our approach to the question of how to code mathematics (mostly experimantal and motivated from geometry) in Java. We are especially interested in the question how the development of mathematical software and the mathematics itself influence each other and how the design of programming tools and code can support this interrelationship.

1 Introduction

There are many ways computers are utilized to do mathematics. The area spreads from computer algebra and computer proofs to numerical analysis and scientific computing. For all of these applications there are powerfull tools available.

The software tools and design paradigms we want to present here can not be sorted easily into these categories:

Computer Algebra Systems (CAS) like MATHEMATICA [28], Maple [14], MuPad [20] or Gap, singular, and magma to name a few are all capable of powerfull symbolic manipulation and algebraic calculations. While there is a well working expression parser in the jtem project symbolic calculations are not the main focus of our system. Our components are designed to support a special way of software/application development which is not present in CAS systems, although they (usually) have powerfull programing languages built in. On the other hand one can not say that our development environment oorange is an integrated development environment (IDE) in the classical sense (like e.g. eclipse [6] or JBuilder for Java): It has for example no debugger but allows changes to a running program and it uses a graphical interface for the control flow (see section 4). Here GRAPE [9] shows the object oriented approach while AVS [3] has a similar graphical interface.

There is some similarity to dynamic geometry software like Cinderella [24,25], Cabri [5], or geometer's sketchpad [7] (again the list is not complete) in the sense that both types of systems can be used in a similar way: to investigate geometric facts, examples, and hypotheses by interactively changing configurations or

© Springer-Verlag Berlin Heidelberg 2006

"parameters". But even if there is a 2d viewer in jtem we are more concerned with 3d graphics than 2d which is the classical domain of the dynamic geometry (there is a 3d version of Cabri though) and do not relay on projective geometry and we do no automated prooving (like e.g. Cinderella).

This is again a complete field on its own (see for example the theorema project [13] which is build on top of MATHEMATICA).

Of course there are many well established systems and viewers for 3d graphics available (vtk [27], open scene graph [21], open inventor [22], aviatrix3d [2] and many more), and some of them are readily tailored for mathematical applications (like geomview [8] or javaview [10] and in some sense the 3DXplorMath [1]) but all of them missed at least one of the requirements we had for a viewer suited for our workflow ansatz (lacking e.g. thread saveness or hardware independence or hardware acceleration – see section 3).

The projects done in the spirit of our approach include numerically demanding ones (like [26] where workstation clusters were used for the search for heliciods with handles) but again while providing a set of standard algorithms for basic tasks we do not claim to be as effective or complete as many of the well established libraries and tools for numerical analysis or scientific computing [29,30]. This is traditionally an area where FORTRAN language shows its strength.

Our software and workflow arose from the needs and experiences our group has made in the last twenty years and the problems or tasks adressed by them origin mainly in discrete and differential geometry. Still they have a wide variety reaching from theory of discrete curves and surfaces, numerics on Riemann surfaces, and surface theory and their visualization. In these areas "mathematical computer experiments" became a common research tool early on, due to the visual character of the field (see e. g. [16]).

Mathematical computer experiments and programming mathematical concepts often do not follow the classical software engineering paradigms of modelling, specification, coding, and testing, but follow rather some concepts of extreme programming [4]. In particular setting up a mathematical experiment usually involves trying various approaches while inspecting the data and data flow closely. In short: these experiments often don't converge to a stable application that is ready for release. Instead the first results most likely give rise to new ideas of needs for modifications which hopefully will give new insight and so on. In these cases one can not any longer distinguish between the phases of development and usage of the software.

Another crucial factor is the ease of maintenance and deployment. Our experience is that in the long run it is not feasible to maintain and deploy software experiments with complicated library dependencies (after all math departments are no software development companies and do not have the manpower to support such developments). In addition it becomes more difficult for other people to run or verify the experiments. Whether an experiment is set up for teaching or for research purposes only, it seems most desirable to have the option of publishing it on the web. These general considerations already give rise to the following design paradigms:

1. maintainability
2. easy deployment and interoperability
3. adaption to mathematical work flow

The following sections feature our approach to a framework for the development of mathematical software which tries to address several of the above mentioned needs. In section 2 we discuss our observations of how experiments and applications grow and evolve and what is needed to support this life-cycle. After a short overview of the tools and packages we developed in section 3, section 4 presents an example that shows how our tools can be used to build fully functional applications.

2 Our Design Decisions

The design decisions presented here are driven by the experiences we gained from previous work. From 1994 to 1997 our group developed a prototyping environment for experimental mathematics: Oorange (Object ORiented Analysis Numerics and Graphics Environment) [15]. The Oorange project had a hybrid language scheme: numerics and graphics were implemented in Objective-C; the graphical programming environment used Tcl for scripting purposes. This decision was one of the reasons why the project finally came to the edge of maintainability. However in this process our design paradigms and a particular work flow emerged.

The first observation to make is that mathematical programming is neither data nor algorithm centered. This has several consequences: Firstly in an adapted programming environment both algorithms and (potentially rather complex) mathematical data objects need to be easily inspected and modified. Coding well defined tasks (like writing a library for a well understood piece of mathematics) is usually best done in a modern IDE like eclipse. But when doing an experiment we often come to a point where the current state might not be easily reproducible but one needs or wants to change the things one can do next in the program (this can be either changing or adding algorithms or equally important doing some visualization of the state). The paradigm for a design honoring this is: *"Changing the code while keeping the data (or state) alive"*.

Secondly one can divide the algorithms in experiments roughly into two types: something one might call "fundamental" or "library" algorithms that are usually well understood and best placed in their own class or framework and "control flow" or "experimental" algorithms that one wants to change modify and adapt repeatedly. Of course there is no easy distinction between the two types. As an example for the first kind one might take minimizers: If you are not in the stage of developing a particular implementation, you usually use them as a black box: you will either change some parameters of the minimizer or replace it with another already tested algorithm. An example for the second kind could be the following scenario: To find the fix-points of a generic Möbius transformation one can pick an algorithm from a numerics package to find the Eigenvectors. This might be unstable. But since the fix-points are generically either attractive or repelling one can replace finding the Eigenvectors with iterating the Möbius transformation a couple of times. This is not something one wants to create a separate package for but rather just do it "on the spot". Thus a design should *separate library and experimental code.*

Thirdly the (mathematical) objects should be prepared to work in a large variety of algorithm frameworks (maybe from different origin) without modification. Usually adapting the objects to as many frameworks as possible is not feasible as it makes the classes less maintainable, more difficult to understand, and adds a lot of mutual dependencies to the packages. So the "gluing code" should be placed outside the objects and if possible the creation of the glueing code should be deferred to the moment a particular interface is needed.

When writing code, reusability and maintainability are prominent goals, since the publication of code can be viewed as a scientific goal. Using only one language greatly simplifies maintainability. However, writing experimental code stands against maintainability. Thus keeping the development of experimental code and application out of the library is essential for keeping the code of the libraries stable, clean, and *orthogonal* [18].

Finally one should mention that we learned that ideally library development and scripting style programming should be done in the *same language.*

2.1 Software Life Cycle

As said above there is no strict border between library and experimental code. In our work flow code for problems that are well understood usually circulate in the group and become eventually (if general purpose enough) library code. But apart form that transition our experiments usually have the following life cycle.

1. create or gather classes or packages for "fundamental algorithms" and mathematical objects. Here "mathematical objects" can vary from complex numbers or quaternions to " a surface" and "fundamental algorithms" from say an ode solver to a special transformation of a surface.
2. connect the algorithms and objects in an experimental algorithm and make it a running application with gui components.
3. inspect the result and modify the experimental algorithms repeatedly.

This is a kind of extreme or micro extreme programming, since the turnaround times are within hours not weeks.

2.2 Implications for the Environment

Modern CAS like MATHEMATICA, Maple or MuPad can in principle follow our work flow paradigms, but neither the interface of the IDE (notebook) nor the design of the languages are specifically adapted for this use: Notebook interfaces do not clearly show the dependencies of different parts of the code and for example rule based programming does not separate code and data at all. Some classical IDEs have code replacing features but this usually not intuitively usable to archive say a particular visualization.

Most modern scripting languages (including CAS systems) can interface external libraries but we learned that ideally library development and scripting style programming should be done in the same language. While IDEs support reusability and component based programming if the language does, giving access to all relevant data using the IDE should not give the feeling/speed of using a debugger.

Ideally the IDE should be free (or even better open source).

Our approach to address at least some of the shortcomings is oorange (see next section).

2.3 Implications for the Programming Language

The projects presented here are developed using only pure Java as programming language (with the exception of jReality that has *optional* support for OpenGL to have hardware accelerated 3d graphics if needed or wanted). Java offers mainstream solutions for all major technical problems we had to deal with in the original Oorange project. This enables us to focus on design, which we broke up into three separated projects: Java Oorange, jReality, and jtem. We will discuss them in greater detail in the next section.

Besides of the modern design of the language Java has the advantage that the questions of easy deployment is virtually non existent anymore. The techniques of Java Webstart and Java applets allow to publish the finished experiments online.

Although the available code base is still one of the largest, FORTRAN never was an option for us since it is clearly not the first choice for graphics.

Using a CAS like MATHEMATICA or Maple as a language was no option either. There is virtually no easy way to do GUIs and although they have become quite performant already these languages have not the speed of Java or C++. Moreover there is only minimal graphical interactivity without add ons and many of them are not free making deployment over the net unfeasible.

Said that, many of the presented design concepts are not really language dependent.

3 A Tour Through the Packages

This section is not intended to be a tutorial or in depth description for any of the described packages but rather a short overview of some of the key concepts on which the packages are build.

jReality. A scene graph library for 3d graphics [11]. While being focused purely on rendering and displaying 3d graphics, the library provides support for all stages and tasks in the visualization of experimental mathematics. It can do OpenGL accelerated rendering on workstations with suitable graphics cards, software only rendering for use in web presentations and applets, multi wall stereoscopic display on CAVE like environments with head tracking for immersive virtual reality, and output to renderman RIB or SVG or Postscript for high quality images for videos or publications.

It should be stressed that jReality is not able and not intended to do complex manipulations on the data or combinatorics of its input. Instead it provides convenient methods to set or update the data it displays and which is provided from somewhere else.

jReality strictly separates the scene graph it maintains from the displaying "back-end". This has some useful consequences. As an example, the tool system (the part responsible for rotating geometry, picking a point or in general user interaction with the scene) can not make any assumptions on the available input devices: While virtually every desktop computer has a mouse (which provides 2-3 (relative) degrees of freedom plus some buttons) there is no such thing in a CAVE like virtual environment here the user usually has a "wand" with 6 (absolute) degrees of freedom (3 for the position and 3 for the orientation of the device in space) plus some buttons. jReality abstracts from the actual device and instead maps the available freedoms to different transformation matrices which then will be used by the tools. Thus the same tool (like a rotate tool) will work in both setups without any changes.

jReality does not depend on jtem or oorange. For readers familiar with Java here is a code fragment that shows everything needed to open a frame that displays Enneper's minimal surface:

```
sf = new ParametricSurfaceFactory();
sf.setGenerateVertexNormals( true );
sf.setImmersion(
  new ParametricSurfaceFactory.DefaultImmersion() {
    public void evaluate( double v, double u ) {
     x = u -u*u*u/3 + u*v*v;
     y = -v-u*u*v +v*v*v/3;
     z = u*u -v*v;
  }} );
sf.update();
ViewerApp.display(sf.getIndexedFaceSet());
```

jReality will see its release summer 06 [12].

jtem (Java Tools for Experimental Mathematics). The core of the jtem project [17,26] consists of a collection of scientific, mostly mathematical, libraries implemented in pure standard Java2.

The project currently offers almost a dozen different subprojects, but its origin are three numerical libraries: numericalMethods, mfc, and riemann.

On a core level jtem provides numerical methods like minimizers and root finders, matrix factorization or algorithms for mesh generation. On a higher level there are classes that model complex numbers, quaternions or special matrix groups and basic linear algebra (blas).

Here as well the idea of strict orthogonality is present since the dependencies are strictly one way: The high level classes will use the numerical methods, but the numerical methods can be used without any dependencies to the higher constructs. There is a package that provides elliptic functions and the library even has some gui components like a Möbius viewer (a 2d viewer that has Möbius transformations a fundamental transformation group) or java2d (a Swing based 2d viewer).

Oorange. The third component is more difficult to categorize: Oorange is a tool for rapid application development, that is especially designed for building applications (or experiments) while they run. The idea is that one can design the classes so that they model the mathematical objects they represent as closely as possible without the need to take care of interfaces or programming paradigms imposed by any other class or package that might be needed in a particular experiment. This interfacing (writing the glue) is done in – and partially by – oorange.

Again oorange does not depend on any of the previous packages but will work with any Java code available.

In oorange the dependencies of the objects are represented as a directed graph with the objects sitting in the nodes. Objects are created by dragging them from a tree view of the Java classpath on a canvas. Changes in the state are propagated along the edges of the graph. The actions to be performed when a change is signaled is specified in the node's "brain".

The brain of a node provides methods that manage the interaction of the object with other objects. For each incoming edge the brain of a node has an (initially empty) method that is called whenever the source of that edge changes its state. One can fill these methods with any Java code needed to perform the appropriate action like setting some values on the object with data read from the object of the source node. The source code for these methods can be written and changed directly in oorange. It gets compiled and loaded on the fly.

Once a node is adapted to a special purpose it can be saved for later reuse. There is a large library of pre-made nodes for both math and gui building.

3.1 Other Packages We Use

Our goal of interoperability would be void if we were not able (or would not profit) from external packages. Among the ones we frequently use are: ant, antlr, xstream, batik, jogl, jinput, javaview, javahelp, beanshell, xpp, mtj, and many more. And of course we use IDEs like eclipse and the jdk from Sun.

4 An Example: Geodesics on Tori of Revolution

We want to illustrate the work flow by sketching the creation process of an application, which shows geodesics on tori of revolution. Markus has build, tested, and deployed it within a couple of hours on the occasion of a regular math course in 2003. It can be found in the virtual math labs at TU Berlin (`www.math.tu-berlin.de/geometrie/labs`).

Even though the example is taken from an educational background it makes similar demands for an experimental environment than any of our typical research applications, e.g. complex user interface, visualization and graphical user interaction with two and three dimensional objects, computational intensive operations.

For such a setup it seems that major technical difficulties are inevitable, but we will show that in our environment these can be reduced to a minimum. This allows also the less experienced user to focus during the creation process of the application on the mathematics and still produces high performance code.

The path for creating the application is roughly as follows:

1. make an editable profile curve and display it.
   One needs an periodic graphically editable profile curve. It should be C^2 since the geodesic equation is second order. Cubic b-splines fulfill all the requirements and are available in jtem. For editing we will use a java2d viewer component.
2. make a torus of revolution from the curve and display it.
   We make the torus via parametric surface factory in jReality (we need to modify a node for that) and use an existing oorange node for the jReality viewer.
3. generate and draw geodesics on the torus.
   Adapt a node that utilize a numericalMethods ode solver to the geodesics equation, lift the result to the surface, and show it. It needs some more code and nodes, e.g. to edit the initial values and some parameters.
4. Finally make the application pretty.

The fundamental numerical and visualization tasks of this example can be considered as typical for our group and are therefore covered by the libraries.

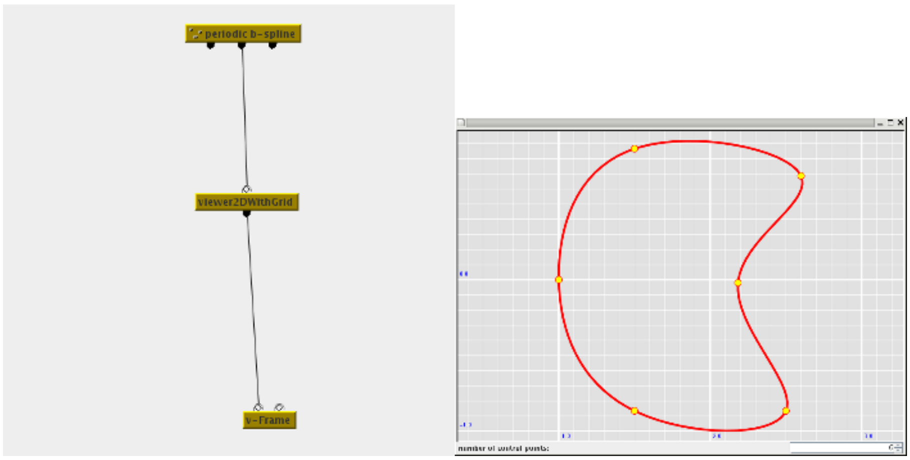


Fig. 1. oorange-network (left) and gui (right) of the application after Step 1

Planar curves appear in many of our applications: To get a graphical editable cubic b-spline curve one just needs to drag three existing oorange nodes into the oorange network editor and cable them. The top node is the actual b-spline. This node wraps the b-spline class available in jtem. The node in the middle is a viewer2d component that can display the spline. It gets the spline as an ingredient. Finally the 2d viewer must be placed in a frame to show it. This is done by cabling it as an ingredient for a node that holds a Java frame [19]. After a couple of seconds without writing a single line of code we finished the first task (Figure 1). Already at this stage we have fully functional application with mathematical content and gui components, which allows us to check right away.

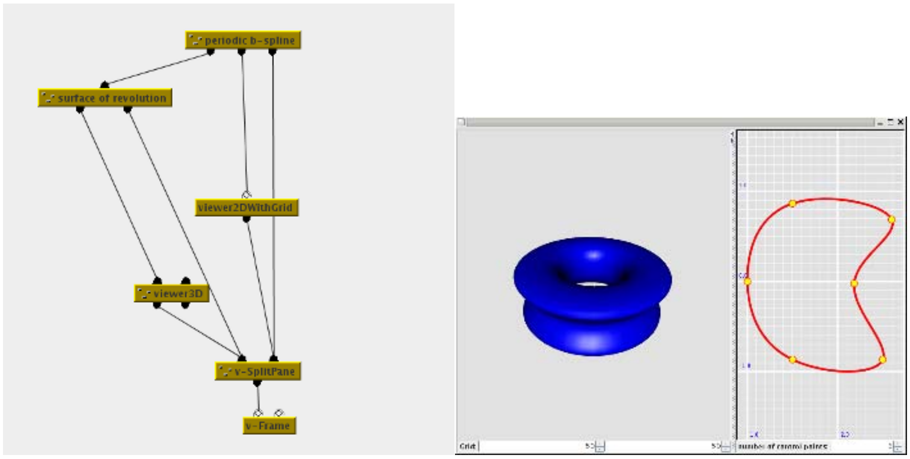


Fig. 2. oorange-network (left) and gui (right) of the application after Step 2

Also for a parametric surface there is a pre-made oorange node, which we have to adapt in order to get a surface of revolution. Therefore we cable the b-spline node with the parametric surface node; now we can adapt the formulas for the immersion referring to the b-spline. The lines to be adapted in the nodes brain are very similar to those presented as example jReality code on page 79:

```
self = new ParametricSurfaceFactory();
self.setClosedInUDirection( true );
self.setClosedInVDirection( true );
self.setGenerateVertexNormals( true );
self.setImmersion(
  new ParametricSurfaceFactory.DefaultImmersion() {
    public void evaluate( double v, double u ) {
     Complex Z = spline.valueAt(v);
     x = Z.re * Math.cos(2*Math.PI*u);
     y = Z.re * Math.sin(2*Math.PI*u);
     z = Z.im;
  } } );
self.update();
```

Now that the parametric surface is indeed a surface of revolution we rename the associated node. To visualize the surface we plug it into a 3d viewer node, which we could pack above the 2d viewer, showing the contour curve, by plugging it also into the frame. But the layout is nicer if we use a split pane [19]. The stage of the application is shown in Figure 2.

Again we can check that everything works as intended. We have already a nice application just the geodesic is missing.

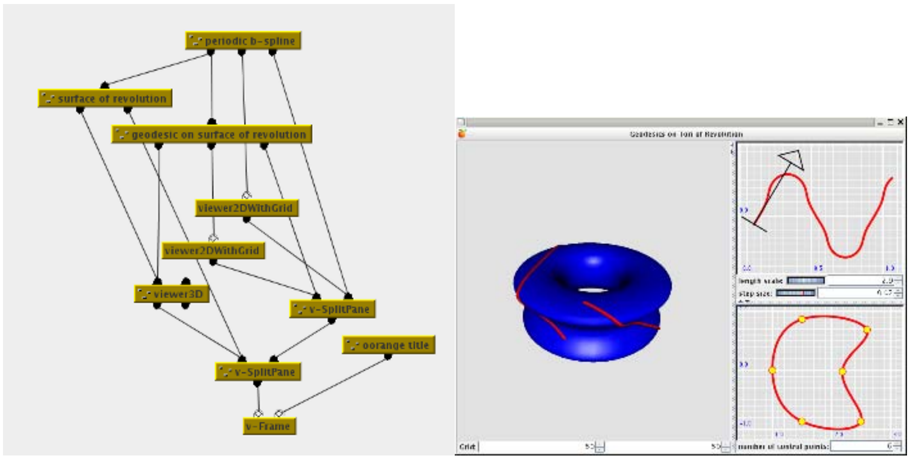


Fig. 3. oorange-network (left) and gui (right) of the application after Step 3

In the third step we adapt a general ode solver node that is based on a jtem solver to our needs. The numericalMethods solver needs an implementation of the interface “ODE” and the oorange node provides an empty implementation that needs to be filled. This is the implementation of the geodesic equation.

```
self = new de.jtem.numericalMethods.calculus.odeSolving.ODE() {
  public int getNumberOfEquations() { return 4; }
  final Real3 ddFTT = new Real3();
  public void eval( double t, double [] x, double [] y  ) {
    double u  = x[0];
    double v  = x[1];
    double du = x[2];
    double dv = x[3];
    Real3 duF   = du.valueAt(u,v);
    Real3 dvF   = dv.valueAt(u,v);
    Real3 dduuF = dduu.valueAt(u,v);
    Real3 dduvF = dduv.valueAt(u,v);
    Real3 ddvvF = ddvv.valueAt(u,v);
    double E = duF.dotProduct( duF );
    double F = dvF.dotProduct( duF );
    double G = dvF.dotProduct( dvF );
    double g = E*G - F*F;
    ddFTT.assignLinearCombination( du*du, dduuF, dv*dv, ddvvF );
    ddFTT.assignLinearCombination( 1, ddFTT,   2*dv*du, dduvF );
    double U = Real3.dotProduct( ddFTT, duF );
    double V = Real3.dotProduct( ddFTT, dvF );
    y[0] = du;
    y[1] = dv;
    y[2] = ( F * V - G * U ) / g;
    y[3] = ( F * U - E * V ) / g;
  }};
```

Entering the actual formulas that give the geodesic equation is obviously inevitable and the above code is just typical numerics code fragment in java which would look similar if it was written in say FORTRAN or C. But the important fact is here that this is by far the longest and the most complicated code which is needed for the whole application.

In the oorange network the component for the geodesic does a few more things. Besides the numerical part there is code that lifts the solution from the parameter domain to the surface, and there are gui components that allow the user to edit the initial values as well as to inspect the solution in the parameter domain. This component is visible in the above figure in the upper right. Figure 3 shows the final state of the network and the application as it was deployed (Java Webstart) for the math course.

The authors would like to thank Nadja Kutz for helpful discussions.

References

1. vmm.math.uci.edu/3D-XplorMath
2. aviatrix3d aviatrix3d.j3d.org
3. AVS www.avs.com/index_wf.html
4. K. Beck and C. Andres. *extreme programming explained: embrace change, second edition*, Addison-Wesley, ISBN 0321278658.
5. Cabri www-cabri.imag.fr/index-e.html
6. eclipse framework www.eclipse.org
7. geometer's sketchpad www.keypress.com/sketchpad
8. geomview www.geomview.org
9. GRAPE www.mathematik.uni-freiburg.de/IAM/Research/grape/GENERAL/
10. javaview www.javaview.de
11. P. Brinkmann, Ch. Gunn, T. Hoffmann, H. Pietsch, M. Schmies, and S. Weißmann. www.jreality.de
12. P. Brinkmann, Ch. Gunn, T. Hoffmann, H. Pietsch, M. Schmies, and S. Weißmann. *jReality — a thread-safe Java scene graph for mathematics. preprint*, (2006), 1011–1018.
13. B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, and W. Windsteiger. *The Theorema Project: A Progress Report.* newblock In M. Kerber and M. Kohlhase, editors, Symbolic Computation and Automated Reasoning, pages 98-113, 2000. newblock www.theorema.org.
14. F. Gravan. *the Maple book*, CRCPress, 2001 newblock www.maplesoft.com.

15. C. Gunn, A. Ortmann, U. Pinkall, K. Polthier, and U. Schwarz. *Oorange: A virtual laboratory for experimental mathematics.* In H.-Chr. Hege and K. Polthier, editors, *Visualization and Mathematics*, 249–265, Springer, 1997.
16. A. J. Hanson, T. Munzner, and G. Francis. *Interactive Methods for Visualizable Geometry* IEEE Computer, Vol. 27, No. 4, pp. 73-83, 1994.
17. T. Hoffmann, S. Khadem, U. Pinkall, and M. Schmies. `www.jtem.de`
18. A. Hunt and D. Thomas. *The Pragmatic Programmer: From Journeyman to Master.* Addison-Wesley, Oct 1999. ISBN: 020161622X
19. `javax.swing.JFrame` and `javax.swing.JSplitPane`. `java.sun.com/javase`.
20. `www.mupad.de`
21. open scenegraph `www.openscenegraph.org`
22. open inventor `www.tgs.com/`
23. H. Pietsch, U. Pinkall and M. Schmies. `www.oorange.de`
24. J. Richter-Gebert and U. Kortenkamp. *User Manual for the Interactive Geometry Software Cinderella* Springer 2000, ISBN: 3-540-67139-0 `www.cinderella.de`
25. J. Richter-Gebert and U. Kortenkamp *Euklidische und Nicht-Euklidische Geometrie in Cinderella.* Journal für Mathematikdidaktik, 22:303-324, 2000.
26. M. Schmies. *Computational Methods for Riemann Surfaces and Helicoids with Handles.* PhD thesis, Technische Universität Berlin, 2005.
27. visualization toolkit `public.kitware.com/VTK`
28. St. Wolfram. *the MATHEMATICA book*, Wolfram Media/Cambridge University Press, 1999 `www.wolfram.com`.
29. LAPACK `www.netlib.org/lapack`
30. MATLAB `www.mathworks.com`

MuPAD's Graphics System

Christopher Creutzig

SciFace Software GmbH & Co. KG
Technologiepark 11
33100 Paderborn
Germany
`www.sciface.com`

Abstract. Starting with MuPAD Pro 3.0, we introduced a new framework for 2D and 3D graphics including animations in computer algebra. Based around the concept of graphical objects that are fully manipulable from the programming level as well as interactively, the framework has proven to be well-designed and flexible. We will present both the users' and the developers' perspective, including how to implement new graphical primitives and a discussion of current limitations.

1 Introduction

Among other applications, computer algebra systems are widely used for research and teaching. For both of these applications, visualization is an important operation and most modern CAS support a variety of plotting functions in two and three dimensions. What follows is a description of one such system, the `plot` library of MuPAD.

MuPAD is a general purpose computer algebra system (CAS) developed by SciFace Software and available for Microsoft Windows, Linux, and Mac OS X. See [1,2] for details.

2 The Users' Perspective

2.1 Primitives

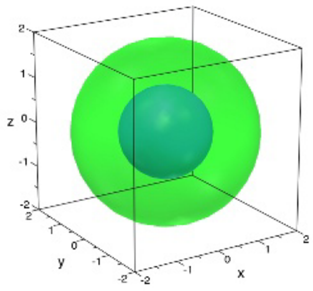
Whatever object-oriented may mean to you, it certainly includes objects which have a type, some form of data, maybe state, and properties. Our graphical objects have type such as 2D function plot, 3D ODE plot, Waterman polyhedron; they obviously have data such as the function to plot, the ode to trace (and from where) or the radius of the sphere defining the polyhedron; they also have properties such as color or line width. Attributes are mostly orthogonal to object types; an attribute such as `Mesh` is used by function and surface plots (including surfaces of revolution, spherical plots etc.), curves, conformal plots, vector fields, implicit plots etc. To make the roughly 400 attributes manageable, each object type can define its own defaults: A density plot by default has a finer mesh than a vector field. These defaults can be changed by the user, either globally or for the current plot or a specific part of it.

© Springer-Verlag Berlin Heidelberg 2006

2.2 Trees

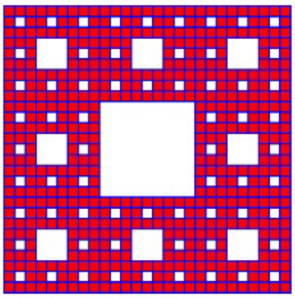
MuPAD's graphical system represents each plot as a tree of objects. Each such tree is rooted in a `Canvas` object which contains one or more `Scene2d` or `Scene3d` objects, which in turn contain coordinate systems, which contain primitives, groups of primitives, and transformations. Most of the time, the user does not need to care about these trees and can simply give the primitives to plot:[1]

```
plot(plot::Sphere(1),
     plot::Sphere(2, Color=RGB::Green.[0.5]))
```



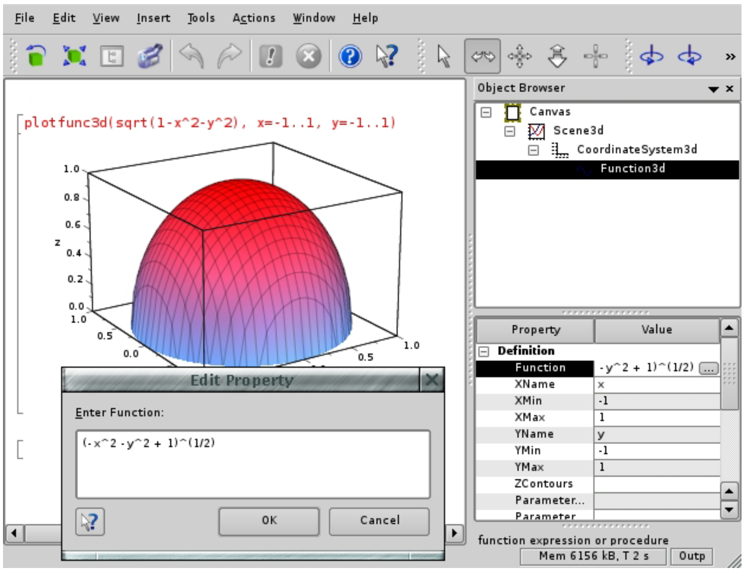
Still, the tree is useful on occasion. For example, linear transformations are nodes in this tree and can thus be applied to arbitrary subtrees:

```
sq := plot::Rectangle(0..1, 0..1, Filled, FillPattern=Solid):
for i from 1 to 3 do
  sq := plot::Scale2d([1/3, 1/3], sq);
  sq := plot::Translate2d([x,y], sq)
           $ x = 0..2/3 step 1/3
           $ y = 0..2/3 step 1/3;
  delete sq[5]; // remove center
end_for:
plot(sq, Axes=None, Scaling=Constrained)
```



1 All images have been scaled down to meet the size limit for this paper.

Many object property defaults, such as line colors or point sizes, are inherited through this tree structure. This is especially useful because *all* parameters can be changed in the graphic viewer:



```
plotfunc3d(sqrt(1-x^2-y^2), x=-1..1, y=-1..1)
```

Enter Function:

```
(-x^2 -y^2 + 1)^(1/2)
```

Property	Value
Definition	
Function	-y^2 + 1)^(1/2)
XName	x
XMin	-1
XMax	1
YName	y
YMin	-1
YMax	1
ZContours	
Parameter...	
Parameter	

function expression or procedure

Mem 6156 kB, T 2 s | Outp

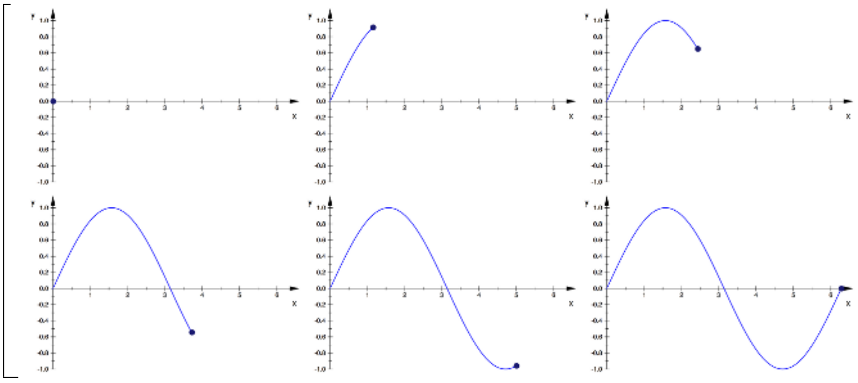
Apart from inheriting attributes, objects can also set hints, attribute values they suggest to objects further up in the tree. For example, a circle suggests that the whole scene should be displayed with constrained scaling, i.e., such that the line $x = y$ is at an angle of 45, functions hint at proper axis titles and a number of objects change the axis style to what is commonly and reasonably used with the type of plot in question.

Together, inheritance and hints allow for a very natural and rather free-form input, where users can in most cases supply values that feel global in almost any place.

2.3 Animation

If visualization of static objects helps mathematical work, animation brings another major step forwards. In MuPAD, animating almost *any* plot is simple (and done in a completely consistent way): At any place after all the required ranges (such as `x=0..10`), add an animation parameter and the range over which it should vary:

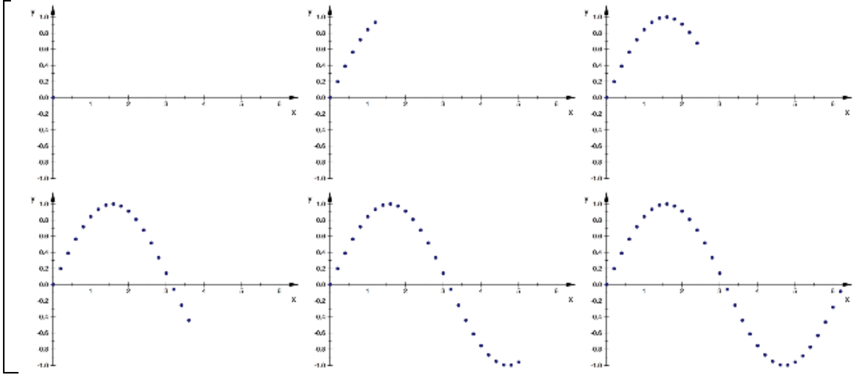
```
plot(plot::Curve2d([x, sin(x)], x=0..xmax, xmax=0..2*PI),
     plot::Point2d([x, sin(x)], x=0..2*PI, PointSize=2.5))
```



The animation parameter can be used in almost any formula, including function terms, color functions, parameter ranges, text definitions, or number of histogram cells, to name just a few. It cannot be used in inherited attributes and some objects, such as the Canvas container, cannot be animated at all.

Each object can have its own time range, which is the range (in seconds, with an arbitrary origin) over which the animation parameter should vary. Whether the object is visible (as a static object) before and after this range is configurable. This way, animations such as graphical proofs can be constructed without undue hassle. To keep the example input short, we use a rather random example here; more elaborate examples can be found in the gallery at the MuPAD web site.

```
plot(plot::Point2d([x, sin(x)], VisibleAfter = x)
      $ x = 0..2*PI step 0.2)
```



Prior to display, each animated object is rendered as a still object for a finite number of parameter values, just like cinema movies are recorded as a rapid succession of photographs. However, each object has its own setting for the number of frames to render. (The default is to render 50 frames for all object types, unless there is no animation parameter used, in which case only one frame is rendered.) The benefit is that slowly changing objects can be rendered more cheaply even in the presence of objects requiring a very fine time resolution.

2.4 Export Formats

Currently, MuPAD can export graphics in various bitmap formats (png, gif, jpeg, bmp), PostScript, svg, and JavaView. Animations can be exported in different avi formats, animated gif, or as a sequence of still images.

3 Technical Point of View

In this chapter, we will look at the technical side of MuPAD's plotting capabilities.

3.1 Programs Involved

As with many CAS, in MuPAD the computational engine (the kernel) and the user interface run in separate processes, with the UI sending commands to the kernel and the kernel sending responses and other data to the UI. Thus, the plot data is computed by one program (written in the MuPAD language) and interpreted by another program (written in C++, using OpenGL and the VRS system of the University of Potsdam ([3]) for 3D and custom routines for 2D), which is integrated in the user interface and also available as a standalone tool for OEM customers.

The viewer is capable of reading and writing the internal data format and performs all the export functionality (see below on limitations this causes) and can generate a MuPAD command which creates the plot as currently seen. The latter is required to allow interactive changing of all aspects of a primitive, including those that can only be interpreted by the kernel.

3.2 Data Format

Plot data is sent from the kernel to the UI in a custom xml format named xvc. The basic structure of these files looks as follows:

```
<Canvas Layout="Tabular" Spacing="1" ...>
  <Scene2dStyle BackgroundColor="#ffffff" .../> ...
  <ObjStyle Type="Default" AntiAliased="1" .../>
  <ObjStyle LegendColor="PointColor" Type="Point2d"/> ...
  <Scene2d Left="0" Bottom="0">
    <CoordinateSystem2d>
      <AxesOriginX Val="0">0</AxesOriginX>
      ...
```

```
<Obj2d Type="Curve2d" Visible="1">
  <Expr Opt="UName">x</Expr>
  <ParameterEnd Val="6.28319">2*PI</ParameterEnd>
  ...
  <Img2d Param="0">
    <Poly2d Filled="0" Closed="0"><P2>0 0</P2></Poly2d>
  </Img2d>
  <Img2d Param="0.128228">
    <Poly2d Filled="0" Closed="0">
      <P2>0 0</P2>
      <P2>0.00106857 0.00106857</P2>
      <P2>0.00213714 0.00213714</P2>
      ...
```

As we can see, most objects are written as `Obj2d` (or `Obj3d`) with a `Type` attribute. The reason for this is that new primitives can be defined at any time, without recompiling or even restarting the viewer. In fact, most of the primitives supplied with MuPAD have been implemented exactly this way. (The snippet above does not show how such types are communicated, because `Curve2d` and `Point2d` are known to the viewer.)

Inside each `Obj2d` element, we find one or more `Img2d` elements holding the rendering of the object for a specific parameter value. (The corresponding time value is calculated by the viewer and not included in the xvc file.) Inside each `Img2d` element, a static image is built from so-called low-level primitives. These cannot be extended without recompilation of the user interface. These low-level primitives are usually not visible to the user, unless they start peeking directly into xvc files.

Low-level primitives include `Poly2d`, `Arc2d`, `Pt2d`, `Pts2d`, `Line2d`, `Arrow2d`, `VerticalAsymptote`, `Text2d`, `ColorArray2d`, `DensityArray2d`, `Field2d`, `Mesh3d`, `Surf3d`, `Poly3d`, `Pts3d`, `Line3d`, `Lines3d`, `Arrow3d`, `Field3d`, `Box`, `Circle3d`, `Cone`, `Sphere`, `Ellipsoid`, `Text3d`, and special types like `Camera`, different types of light, and `ClippingBox`. It is not uncommon for a single `Img2d` to contain multiple different low-level primitives.

3.3 Rendering

While parsing, the viewer builds a tree representation of the xml file in memory. This representation follows the xml file rather closely, but with attribute defaults propagated to the relevant low-level primitives and the time values for the frames calculated. For plots with visible axes, the viewer then decides where to place the axes, their titles, and their tick marks, unless these have been set explicitly by the user.

Static objects are rendered by translating the low-level primitives into the primitives of the underlying rendering engine: VRS primitives for 3D graphics, primitives of the custom system in 2D. This translation is a very thin layer for most low-level primitives, but some primitives such as `Field3d` must be broken into a larger number of rendering objects.

For animations, the in-memory tree is queried for the next time value with a change and when that time is reached, a snapshot of the tree at this time position is rendered as a still image. (When exporting animations, equidistant time values are used instead, because most animation formats are built around the assumption of a constant frames per second rate.)

3.4 Recalculations

Some interactive changes, such as line color, take effect immediately. Others, such as changing the calculation mesh, must be sent to the kernel first. To avoid inconsistent states when editing multiple related attributes, this is done only when the user explicitly requests a recalculation. (By default, the user is alerted of this requirement upon the first non-immediate change.) This recalculation is performed by sending a complete plot command to the kernel and replacing the current image by the result of the command.

If the object plotted still exists at the library level (i.e., it has been assigned to some variable or stored in some other way), then after a recalculation the changes made interactively will be reflected at the library level. This means, among other applications, that the changes will carry over to subsequent plots involving the same object. (There are utility functions `plot::copy` and `plot::modify` to break this link for when it is not desired.)

4 Programmer's Point of View

As stated above, it is possible to introduce new graphical types by writing MuPAD code. In this chapter, we will write code for the strange attractor of the Hnon map, i.e., we are going to trace the behaviour of the discrete dynamical system given by the iteration $x_{n+1} = 1 - ax_n^2 + y_n$, $y_{n+1} = bx_n$.

4.1 Using the Framework

First of all, we have to tell the plot framework that we want to create a new object type, let's call it `Henon`. Our objects will be 2D and use the obvious attributes, such as `PointSize`, `PointColor`, `Iterations`, but also a new ones for a and b, which we will call `HenonA` and `HenonB` and for which we will need to tell the system a few things: They are supposed to be numerical values (we'll see later why we call them "Expr"), are mandatory (we could provide defaults, but not having any values is rather pointless) and belong to the attributes in the Definition section of the inspector. Additionally, we copy the definition of the `Iterations` attribute from `plot::Iteration`.

```
Henon := plot::createPlotDomain("Henon",
   // short description for status bar:
   "strange attractor of the Henon map",
   2, // dimension
   [PointSize, PointColor, PointStyle, Color,
    [HenonA, ["Mandatory", NIL], ["Definition", "Expr", FAIL,
      "First coefficient of the Henon iteration.", TRUE]],
    [HenonB, ["Mandatory", NIL], ["Definition", "Expr", FAIL,
      "Second coefficient of the Henon iteration.", TRUE]],
    [Iterations, ["Optional", 10], ["Definition", "Expr", FAIL,
      "Number of iteration steps.", TRUE]]]):
```

```
plot::setDefault(Henon::PointSize = 0.1,
  Henon::PointStyle = FilledCircles,
  Henon::Iterations = 3000):
```

As we have seen in the examples above, object creation and plotting are separated, and this is of course reflected in the code. The routine for object creation always uses `dom::checkArgs` for the argument parsing (to ensure a consistent interface), gets all the arguments the generic interface could not handle, plugs them into the right attributes, and finally declares the object as finished by calling `dom::checkObject`:

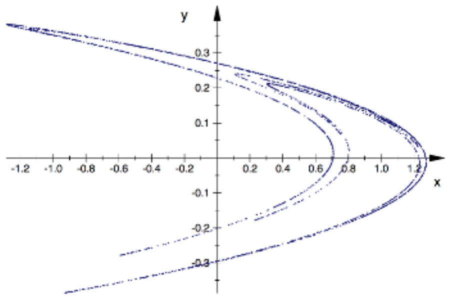
```
Henon::new :=
proc()
  local object, other;
begin
  object := dom::checkArgs([], args());

  other := object::other;

  case nops(other)
  of 2 do
    object::HenonA := other[1];
    object::HenonB := other[2];
    break;
  of 0 do break; // for recalc
  otherwise
    error("expecting two arguments");
  end_case;

  dom::checkObject(object);
end_proc:
```

The other half of the equation is the actual plotting. Following the discussion above, we will create a `Pts2d` low-level primitive and fill it with points. Note that we do not create any `Obj2d` or `Img2d` object: At the time our code is called, these have already started.

```
Henon::MuPlotML :=
proc(object, attributes, inheritedAttributes)
  local x, y, i, a, b, iter, xmin, xmax, ymin, ymax;
begin
  // extract all we need to use from attributes:
  a := float(attributes[HenonA]);
  b := float(attributes[HenonB]);
  n := attributes[Iterations];

  iter := (x, y) -> [1 - a*x^2 + y, b*x];

  // start the iteration without plotting first:
  [x, y] := [0.0, 0.0];
  for i from 1 to 100 do [x, y] := iter(x, y); end;

  // now, plot the next points,
  // remembering the min and max values:
  xmin := xmax := x;
  ymin := ymax := y;
  plot::MuPlotML::beginElem("Pts2d", "PointsVisible"=TRUE);
  for i from 1 to n do
    [x, y] := iter(x, y);
    xmin := min(xmin, x);  xmax := max(xmax, x);
    ymin := min(ymin, y);  ymax := max(ymax, y);
    plot::MuPlotML::prP2(x, y);
  end;
  plot::MuPlotML::endElem("Pts2d");
  // return the extent of our plot
  return([xmin..xmax, ymin..ymax]);
end_proc:
```

And we are done with our first new object, all properties interactively editable in the inspector and everything:

```
plot(Henon(1.4, 0.3))
```



Well, almost. A real-world implementation would need to detect iterations going off to infinity and stop there:

```
plot(Henon(1.5,0.5))

 Error: Overflow/underflow in arithmetical operation;
 during evaluation of 'iter'
```

If you start thinking about all the other forms of input validation that never show up in the code above: That's because the general framework takes care of them.

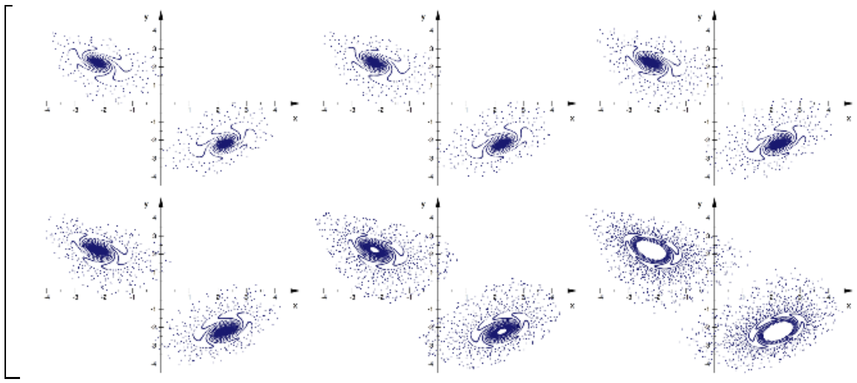
```
Henon(1.2, 0.2, FillColor=Blue, Color=Rot)

 Warning: 'FillColor' makes no sense in plot::Henon,
   ignored [Henon::new]
 Error: expecting an RGB or an RGBa color for attribute
   'Color' in Henon object [plot::checkOptions]
```

```
Henon(a, b)

 Error: unbound identifier(s) 'b, a' found [Henon::new]
```

4.2 Animation

With all our calculations stored safely away in the `MuPlotML` routine, animation do *not* need any extra work, since it is already provided by the framework:

```
plot(Henon(0.2, a, a=0.995..0.999))
```



5 Loose Ends

There are a number of things graphics in a CAS should provide which MuPAD currently does not offer.

5.1 Formula Typesetting

Currently, MuPAD graphics do not include typeset formulas. Since this is merely a matter of not done yet, not a principal problem (formulas to be typeset are stored as xml in other parts of the program already), there is little to discuss here.

5.2 More Degrees of Freedom

The current model permits a single degree of freedom for interactive parameter change, usually used as an animation parameter. In some applications, it would be helpful to have multiple degrees of freedom, such as changing two angles of a triangle or two or more parameters of a function. This is more difficult to achieve, since the design requires all numerical data to be computed at the time the display opens and our implementation separates the computation engine from the front-end. There are potential ways around this limitation, such as sending xml fragments or xml descriptions of the plot corresponding to the current parameter settings, potentially even exploring the surroundings of the current values in the background, for a faster response time. This is a research area.

5.3 On-Line Changes

It is currently not possible in MuPAD to let a long-running computation such as a simulation update an existing plot. The model obviously allows this, either by completely replacing an active plot (or individual objects in the plot identified by their xml ids) or by appending animation data.

5.4 User Feedback

In some cases, it would be helpful to just click on some part of the graphics and have something happen. MuPAD never aimed to be an interactive geometry package nor a fractal explorer, therefore this kind of feedback did never receive high priority. Since the graphical viewer is already able to make callbacks to the computing engine (e.g., when changing the term of a function plot), this is obviously not fundamentally impossible.

5.5 Export Formats

At least X3D (the successor to VRML, [4]) and the popular 3D model exchange format obj should be supported and for creating beautiful plots it would sometimes be helpful to export POVRay code ([5]) and/or to support the RenderMan standard ([6]).

There is currently no API to define new export formats. Such an API would work on the low-level trees of the xvc format.

In some cases (especially when exporting to POVRay), the internal xml format (xvc) contains less information than the exported code should ideally contain; as an example, triangulated parameterized surfaces in an xvc file do not contain parameter values, only triangles and their color information. For subsequent editing, this is insufficient. Also, the animation model at the user's side is much better suited to POVRay animations than the one used in the xvc files. There are two possible ways around this:

- Extend the xvc format
- Write POVRay code not from the xvc file, but directly from the library objects

The second way certainly gives more flexibility in the output and makes introducing new export formats easier for the user (assuming she is comfortable with writing MuPAD code), but would not automatically extend to new primitives. No complete solution has been drafted at the time of this writing.

6 Conclusion

We have presented one of many possible ways to implement easy-to-use, powerful, and readily extensible graphics capabilities in a general purpose computer algebra system. By restricting the programmers of such extensions, the framework has, since its conception in 2002, provided a stable and intuitive interface to the users.

Some areas which such a system should cover have not been implemented yet, but so far, the only wish that could not be fulfilled without major design changes was interactivity comparable to interactive geometry systems, which buy this flexibility by strictly limiting the type of objects that can be manipulated.

References

1. Creutzig, C. and Oevel, W.: MuPAD Tutorial, 2nd ed, Springer, 2004
2. Majewski, M.: Getting Started with MuPAD, Springer, 2005
3. Hasso-Plattner-Institute, University of Potsdam: VRS – The Virtual Rendering System, `www.vrs3d.org`
4. web3D Consortium: X3D International Standard, `http://www.web3d.org/x3d/specifications/`
5. POVRay team: POVRay raytracer, `www.povray.org`
6. Pixar: The RenderMan Interface, `https://renderman.pixar.com/products/rispec/`

An Efficient Implementation for Computing Gröbner Bases over Algebraic Number Fields

Masayuki Noro

Kobe University, Rokko, Kobe 657-8501, Japan
noro@math.kobe-u.ac.jp

Abstract. In this paper we discuss Gröbner basis computation over algebraic number fields. Buchberger algorithm can be executed over any computable field, but the computation is often inefficient if the field operations for algebraic numbers are directly used. Instead we can execute the algorithm over the rationals by adding the defining polynomials to the input ideal and by setting an elimination order. In this paper we propose another method, which is a combination of the two methods above. We implement it in a computer algebra system Risa/Asir and examine its efficiency.

1 Introduction

From the theoretical point of view, Gröbner basis computation can be done over any field by using Buchberger algorithm. In practice, however, the computational efficiency depends on the ground field. The difficulty of Gröbner basis computation over a finite field mainly comes from its combinatorial property because no coefficient swell occurs. But over the rationals, we often suffer from coefficient swell and various methods to avoid it have been investigated. The trace algorithm [4] and F_4 algorithm with modular computation [6] are successful ones. In this article we consider Gröbner basis computation over algebraic number fields. If the operations over an algebraic number field are provided, we can apply usual Buchberger algorithm over the field. Instead, an algebraic number field K can be represented as a residue class ring $\mathbf{Q}[x_1, \ldots, x_l]/J$, where J is a zero-dimensional maximal ideal and a Gröbner basis computation over K can be reduced to that over $\mathbf{Q}$ by joining J to the ideal to be considered. However these method are not satisfactory in view of efficiency. Here we give a simple but efficient method which is a combination of the two methods above.

Notation 1

$\mathrm{HT}(f)$	*: the highest term of f with respect to a term order*
$\mathrm{HC}(f)$	*: the coefficient of* $\mathrm{HT}(f)$
$\mathrm{GF}(p)$	*: the finite prime field of order p*
$\mathrm{NF}_G(f)$	*: a remainder of f with respect to a polynomial set G By fixing the method for choosing a reducer, the remainder is uniquely determined even if G is not a* Gröbner *basis.*
$\mathrm{S}(f,g)$	*: the S-polynomial of a pair $\{f,g\}$.*

© Springer-Verlag Berlin Heidelberg 2006

$\mathbf{Z}_{\langle p\rangle}$: $\{a/b \mid a \in \mathbf{Z}; b \in \mathbf{Z} \setminus p\mathbf{Z}\} \subset \mathbf{Q}$
ϕ_p : *the canonical projection from* $\mathbf{Z}_{\langle p\rangle}[X]$ *to* $\mathrm{GF}(p)[X]$

2 The Algorithm

Suppose that an algebraic number field $K = K_l$ is represented as a tower of simple extensions:

$$K_0 = \mathbf{Q}, K_i = K_{i-1}(\alpha_i) \quad (i = 1, 2, \ldots, l),$$

where α_i is a root of a monic irreducible polynomial over K_{i-1}:

$$m_i(\alpha_1, \ldots, \alpha_{i-1}, t_i) \in K_{i-1}[t_i] \quad (m_i(t_1, \ldots, t_i) \in R_i = \mathbf{Q}[t_1, \ldots, t_i])$$

and

$$K_i = \mathbf{Q}[t_1, \ldots, t_i]/J_i, \quad J_i = \langle m_1, \ldots, m_i \rangle.$$

Each J_i is a zero-dimensional maximal ideal of R_i. Set $S = K[x_1, \ldots, x_n]$, $T = \mathbf{Q}[x_1, \ldots, x_n, t_1, \ldots, t_l]$, $D = \{m_1, \ldots, m_l\}$, $J = \langle D \rangle$, $x = (x_1, \ldots, x_n)$, $t = (t_1, \ldots, t_l)$ and $\alpha = (\alpha_1, \ldots, \alpha_l)$. Let $\prec$ be a term order in S and $\prec_K$ a product term order of $\prec$ and the lexicographic order in R_l such that $t_1 \prec t_2 \prec \cdots \prec t_l$ and $t^a \prec_K x^b$.

Definition 1. $\mathrm{HC}_R(f) \in R$ *denotes the head coefficient of* f *as an element of* $R[x_1, \ldots, x_n]$ *with respect to* $\prec$. *We call* f *monic if* $\mathrm{HC}_R(f) = 1$.

Let $\tilde{B} = \{g_1(x,t), \ldots, g_d(x,t)\}$ be a subset of T and $I \subset S$ an ideal generated by $B = \{g_1(x,\alpha), \ldots, g_d(x,\alpha)\}$. The following theorem is well known.

Theorem 1. *Let* $\tilde{G}$ *be the reduced* Gröbner *basis of* $\tilde{I} = \langle \tilde{B} \cup D \rangle$ *with respect to* $\prec_K$. *Then* $(\tilde{G} \setminus D)|_{t=\alpha}$ *is the reduced* Gröbner *basis of* I *with respect to* $\prec$.

By this theorem, we can apply Buchberger algorithm over $\mathbf{Q}$ to compute the Gröbner basis of I over K. However, if we observe the execution of Buchberger algorithm, we notice that many intermediate basis elements of the form $t^b x^a +$ *lower* are generated before a monic element $x^a +$*lower* is generated. Once we have such a monic element, the intermediate basis elements become all redundant. This phenomenon is explained as follows. Suppose that $h(x,t) = t^b x^a +$*lower* $\in \tilde{I}$ is reduced with respect to D. We set $h_a(t) = \mathrm{HC}_R(h)$. Then $h_a(\alpha)$ is nonzero because $h_a(t)$ is reduced with respect to D. Then $h_a(\alpha)$ is invertible in K, which means that there exists $u(t) \in R$ such that $u(t)h_a(t) \equiv 1 \bmod J$. Then we have a monic element $u(t)h(x,t) = x^a +$ *lower* $\in \tilde{I}$. That is, the intermediate elements are those which have intermediate polynomials generated during the computation of the inverse of $h_a(\alpha)$ as their head terms. This affects the process of computation in various ways. For example, each generated basis element produces new S-pairs. It is well known that the efficiency of Buchberger algorithm is sensitive to the order of S-pairs to be processed, and the new S-pairs may make the subsequent computation inefficient. Or, the inverse computation implicitly appeared in our case is nothing but Euclid algorithm, and it is well known that computing the inverse of an algebraic number by Euclid algorithm tends to cause coefficient swell. Our remedy for this difficulty is very simple. We modify Buchberger algorithm as follows. We omit the selection strategy and the criteria for useless pair detection in the algorithm description.

Algorithm 1 (Buchberger algorithm over an algebraic number field)

```
$L \leftarrow \{\{f, g\} \mid f, g \in \tilde{B}, f \neq g\}$
$G \leftarrow \tilde{B}$
while $L \neq \emptyset$ do
    $\{f, g\} \leftarrow$ an element of $L$; $L \leftarrow L \setminus \{\{f, g\}\}$
    $\tilde{r}(x, t) \leftarrow \mathrm{NF}_{D \cup G}(\mathrm{S}(f, g))$
    if $\tilde{r} \neq 0$ then
      $u(t) \leftarrow$ the inverse of $\mathrm{HC}_R(\tilde{r}) \bmod J$
      $r \leftarrow \mathrm{NF}_D(u\tilde{r})$
      $L \leftarrow L \cup \{\{f, r\} \mid f \in G\}$
      $G \leftarrow G \cup \{r\}$
    end if
end while
return $G$
```

In Algorithm 1, G consists of monic polynomials because each normal form is made monic before added to G, therefore $\tilde{r}(x, \alpha)$ is a normal form with respect to $G|_{x=\alpha}$. So Algorithm 1 executes Buchberger algorithm over K and it returns a Gröbner basis of $\langle B \rangle$. Algorithm 1 avoids generating redundant basis elements. Furthermore the trace algorithm[4] and the efficient content reduction [5] can be applied because the computation itself is done over $\mathbf{Q}$.

3 The Implementation

We implemented Algorithm 1 in Risa/Asir [1]. We already have an implementation of Buchberger algorithm over $\mathbf{Q}$ with various optimization (the trace algorithm, the homogenized trace algorithm and the efficient content reduction) and the only function to be newly implemented is an efficient inverse computation of algebraic numbers. In addition, the normal form computation with respect to D, the set of defining polynomials greatly affects the whole efficiency.

3.1 Simplification of Algebraic Numbers

The normal form computation with respect to D is equivalent to the simplification of algebraic numbers by the defining polynomials. As D is the reduced Gröbner basis with respect to the lexicographic order, the result of simplification does not depend on the order of monomial reductions. However its cost depends on the order.

Example 1. Set

$$\begin{aligned} m_1(t_1) &= t_1^{20} + t_1^{19} + 2, \\ m_2(t_1, t_2) &= t_2^{30} + (t_1^{19} + t_1^{18} + 1)t_2^{29} + 1. \end{aligned}$$

$m_1(t_1)$ is irreducible over $\mathbf{Q}$. Let α_1 be a root of $m_1(t)$. Then $m_2(\alpha_1, t)$ is irreducible over $\mathbf{Q}(\alpha_1)$ and let α_2 be a root of $m_2(\alpha_1, t)$. Let us consider the simplification of α_2^{58}. If we always choose the reducer m_k with the smallest possible k during a simplification, it takes only 0.02 sec. But if we always choose m_k with the largest possible k, it takes 2 sec. In the latter strategy, the cost for simplification by m_2 is dominant. After simplifying all the occurrences of α_2^n for $n \geq 20$, the intermediate remainder contains 4674 monomials and its α_1 degree is $551 = 19 \times 29$, which is a kind of intermediate expression swell.

In our current implementation, the simplification is done by the following algorithm. Let $f \in R$ be a polynomial to be simplified.

Algorithm 2

```
$r \leftarrow 0$
*while* $f \neq 0$ *do*
    *if* $\mathrm{HT}(f)$ *is reduced with respect to* $D$ *then*
      $r \leftarrow r + \mathrm{HT}(f)$
      $f \leftarrow f - \mathrm{HT}(f)$
 *else*
      $k \leftarrow$ *the smallest* $k$ *such that* $\mathrm{HT}(m_k) | \mathrm{HT}(f)$
(*)   $f \leftarrow f - \mathrm{HC}(f) \cdot \frac{\mathrm{HT}(f)}{\mathrm{HT}(m_k)} m_k$
    *endif*
*end while*
*return* $r$
```

This is based on the following proposition.

Proposition 1. *Let f_1 be the right hand side of $(*)$ in Algorithm 2. Then we have*

$$\deg_{t_i}(f_1) \leq \begin{cases} \mathrm{MAX}(2(\deg_{t_i}(m_i) - 1), \deg_{t_i}(f)) & (i \leq k-1) \\ \deg_{t_i}(f) & (i \geq k) \end{cases}$$

Proof. By the property of k, $\deg_{t_i}(\mathrm{HT}(f)) \leq \deg_{t_i}(m_i) - 1$ for $i = 1, \ldots, k-1$ holds and we have

$$\deg_{t_i}(\frac{\mathrm{HT}(f)}{\mathrm{HT}(m_k)}(m_k)) \leq \begin{cases} 2(\deg_{t_i}(m_i) - 1) & (i \leq k-1) \\ \deg_{t_i}(\mathrm{HT}(f)) & (i \geq k) \end{cases}$$

because m_k is reduced with respect to m_i for $i \leq k-1$ and m_k does not contain t_i for $i \geq k+1$. This proves the assertion. □

In particular, if $f(t)$ and $g(t)$ is reduced with respect to D, then each t_i-degree of the intermediate reminders does not exceed $2(\deg_{t_i}(m_i) - 1)$ during the execution of Algorithm 2 and the remainder computation is expected to be efficient.

3.2 Computation of the Inverse of an Algebraic Number

Even if the ground field K is a simple extension, the computation of the inverse of an algebraic number is not an easy task. Non-modular extended Euclid algorithms often cause coefficient swell and it seems better to apply modular methods. We can apply Hensel lifting or Chinese remainder theorem (CRT) to compute the inverse of $f(\alpha)$ for $f(t) \in R$. Let $M = \{s_1, \ldots, s_d\}$ be the set of monomials which spans $R/\langle D\rangle$ over $\mathbf{Q}$, where $d = \dim_{\mathbf{Q}} R/\langle D\rangle$. Then we can compute $g(t) \in R$ such that $g(\alpha)f(\alpha) = 1$ as follows.

Algorithm 3

```
Convert $\sum_i c_i \mathrm{NF}_G(fs_i) = 1$ into a system of linear equations $Ac = b$
with respect to $c = (c_1, \ldots, c_d)^T$.
$a = (a_1, \ldots, a_d)^T \leftarrow$ the solution of $Ac = b$
                              (Apply a modular method such as Hensel lifting or CRT.)
return $\sum_i a_i s_i$
```

A more detailed explanation can be found in [7].

3.3 The Homogenized Trace Algorithm

Homogenization is very useful for avoiding intermediate coefficient swell over $\mathbf{Q}$, but it also increases the number of S-polynomials reduced to zero. By combining homogenization and the trace algorithm, we can cut the additional cost introduced by homogenization. Let B be a set of polynomials and p a prime. Algorithm 4 is a general algorithm to compute a Gröbner basis candidate.

Algorithm 4 (`Candidate`(B,p))

```
$L \leftarrow \{\{f,g\} \mid f,g \in B, f \neq g\}$
$G \leftarrow B$; $G_p \leftarrow \phi_p(B)$
while $L \neq \emptyset$ do
    $\{f,g\} \leftarrow$ an element of $L$
    $L \leftarrow L \setminus \{\{f,g\}\}$
    if $\mathrm{NF}_{G_p}(\phi_p(f), \phi_p(g)) \neq 0$ then
      $r \leftarrow \mathrm{NF}_G(\mathrm{S}(f,g))$
      if $\phi_p(\mathrm{HC}(r)) = 0$ then return **failure**
      $L \leftarrow L \cup \{\{f,r\} \mid f \in G\}$
      $G \leftarrow G \cup \{r\}$; $G_p \leftarrow G_p \cup \{\phi_p(r)\}$
    end if
end while
return $G$
```

Let S_d be the set of S-polynomials whose total degrees are equal to d in an execution of Buchberger algorithm. For a homogeneous input, we process S_d in the increasing order with respect to d. Then, after processing all S_i $(i < d)$, S_d produces all the Gröbner basis elements of the total degree d. We observe that the total efficiency is improved if we execute an inter-reduction after processing

S_d. Furthermore, we also observe that inter-reductions during processing S_d often improves the efficiency. Note that such inter-reductions are allowed because what we are doing on S_d is nothing but the computation of a linear basis of the homogeneous component I_d of the total degree d of the input ideal I. In the current implementation, an inter-reduction is executed every after k new basis elements are generated, where k can be set by users and the default value is 6.

Algorithm 5 (Homogenized trace algorithm)

```
$B_h \leftarrow$ the homogenization of $B$
loop
    $p \leftarrow$ a new prime
    $G_h \leftarrow$ Candidate($B_h, p$)
    if $G_h \neq$ failure then
        $G \leftarrow$ the dehomogenization of $G_h$
        $G \leftarrow \{g \in G \mid \mathrm{HT}(h) \not| \mathrm{HT}(g)$ for all $h \in G \setminus \{g\}\}$
        if $G$ is a Gröbner basis of $\langle G \rangle$ and $\mathrm{NF}_G(f) = 0$ for all $f \in B$ then return $G$
    endif
end loop
```

In Algorithm 5, a candidate of a Gröbner basis of $\langle B_h \rangle$ is computed, but the final check is done for the dehomogenized candidate. If B is not homogeneous, we usually have many redundant elements by dehomogenizing G_h. Except for finitely many p, G_h is a Gröbner basis of $\langle B_h \rangle$, in that case G is a Gröbner basis of $\langle B \rangle$. Then G is still a Gröbner basis after removing the redundant elements and the final checks are expected to be easy by the removal of redundancy [1].

Now we go back to our Gröbner basis computation over algebraic number fields. The normal forms in Algorithm 1 are computed over $\mathbf{Q}$, so we can apply Algorithm 5 by modifying Algorithm 4 as in Algorithm 1. We mention here the homogenization of elements in $\mathbf{Q}[x, t]$. We use the same notation as in Theorem 1. Algorithm 1 is essentially an algorithm over an algebraic number field K and it is natural to regard t-variables as parts of the coefficient field. Therefore, when we homogenize $\tilde{B}$ before entering Algorithm 4, the weights of t-variables, which are used to compute sugar in Algorithm 4, are set to 0. This setting seems natural, but it is not always optimal from the viewpoint of practical efficiency. This will be discussed later.

4 Experiments

4.1 Related Functions

We briefly explain Risa/Asir functions for Gröbner basis computation over algebraic number fields.

[1] In some cases, the check is hard because of the large coefficients of the final basis elements.

– `newalg`(*DefPoly*) generates a root of *DefPoly*, where *DefPoly* is a monic univariate polynomial whose coefficients are polynomials of already defined roots. That is, Risa/Asir can deal with multiple extension fields. Note that the system does not check whether *DefPoly* is irreducible over the field generated over **Q** by the roots contained in the polynomial. The irreducibility can be checked by `af`.
– `af`(*Poly*, *AlgList*) factorizes a univariate polynomial *Poly* over an algebraic number field generated by roots listed in *AlgList*.
– `nd_gr_trace`(*PolyList*, *VarList*, *Homo*, *Trace*, *Order*) computes the reduced Gröbner basis of $\langle PolyList\rangle \subset K[VarList]$ with respect to a term order specified by *Order*, where K is an algebraic number field generated over **Q** by all the roots appeared in *PolyList*. $Order = 0$ and $Order = 2$ mean the graded reverse lexicographic order (grlex) and the lexicographic order (lex) respectively. If $Trace = Homo = 1$, Algorithm 5 is executed.This function implements various improvements such as vectorized length exponent vector [9] and geobucket addition [8].

Remark 1. For comparison we also implemented a function which computes Gröbner bases over algebraic number fields by using addition, subtraction and multiplication over algebraic number fields. In this implementation, each normal form is appended to the intermediate basis without being made monic. Then we have to multiply the polynomial to be reduced by an algebraic number in each reduction step, which makes the coefficients larger. Furthermore, the trace algorithms cannot be applied because the ground field is an algebraic number field. Thus all the computations have to be done over the algebraic number field. We applied this function to several examples which are easily computed by `nd_gr_trace` and we found this function is useless for our purpose.

Example 2. Univariate polynomial GCD can be computed by Gröbner basis computation. In the following Risa/Asir session, `A1` is a root of `M1`. Then `M2` is defined over **Q**(`A1`) and `af` tells that it is irreducible over **Q**(`A1`). `A2` is a root of `M2`. `F1` and `F2` are polynomials over **Q**(`A1`, `A2`) and GCD(`F1`, `F2`) is computed by `nd_gr_trace`.

```
[0] load("sp")$
[101] M1=t1^6+6*t1^4+2*t1^3+9*t1^2+6*t1-4$
[102] A1=newalg(M1);
(#0)
[103] M2=t^3+3*t+A1^3+3*A1+2$
[104] af(M2,[A1]);
[[t^3+3*t+(#0^3+3*#0+2),1]]
[105] A2=newalg(M2);
(#1)
[106] F1 = x^6+(-6*A2+3*A1)*x^5+(15*A2^2-15*A1*A2+6*A1^2+27)*x^4
+(-20*A2^3+30*A1*A2^2+(-24*A1^2-108)*A2+7*A1^3+54*A1)*x^3+(15*A2^4
-30*A1*A2^3+(36*A1^2+162)*A2^2+(-21*A1^3-162*A1)*A2-27*A1^5+6*A1^4
-135*A1^3-216*A1)*x^2+(-6*A2^5+15*A1*A2^4+(-24*A1^2-108)*A2^3
+(21*A1^3+162*A1)*A2^2+(54*A1^5-12*A1^4+270*A1^3+432*A1)*A2+3*A1^5
+27*A1^4+27*A1^3+27*A1^2+162*A1-108)*x+(A2^6-3*A1*A2^5+(6*A1^2+27)
*A2^4+(-7*A1^3-54*A1)*A2^3+(-27*A1^5+6*A1^4-135*A1^3-216*A1)*A2^2
+(-3*A1^5-27*A1^4-27*A1^3-27*A1^2-162*A1+108)*A2-54*A1^5-6*A1^4
-218*A1^3-90*A1^2-276*A1+220)$
[107] F2 = x^4+(2*A2+2*A1)*x^3+(3*A2^2+3*A1*A2+3*A1^2+12)*x^2
+(3*A1*A2^2+(3*A1^2+6)*A2+6*A1-4)*x+(-2*A2-2*A1)$
[108] G=nd_gr_trace([F1,F2],[x],1,1,0);
[20*x^2+(20*#1+20*#0)*x+((-6*#0^5+3*#0^4-30*#0^3+3*#0^2-48*#0+8)
*#1^2+(3*#0^5+6*#0^4+15*#0^3+36*#0^2+34*#0+36)*#1-12*#0^5+6*#0^4
-60*#0^3+26*#0^2-96*#0+96)]
```

4.2 Timings

In the following examples, Algorithm 5 is applied over an algebraic number field or **Q**. Timings were measured on a PC with Intel Xeon 3.4GHz. In the tables, “total”, “#basis”, “check”, “monic” mean the total time, the number of the intermediate basis elements, the time for checking the Gröbner basis candidate and the time for making the normal forms monic. The last two are included in the total time. Timings are shown in seconds.

Example 3. Let C_7 be Cyclic-7 system with variables $c_1, \dots, c_7$. We compute the reduced Gröbner basis of $C_{7,\omega} = C_7|_{c_7=\omega}$ with respect to grlex order for $c_1 \succ \cdots \succ c_6$, where ω is a root of an irreducible factor $m(c_7)$ of the minimal polynomial of c_7 in $\mathbf{Q}[c_1, \dots, c_7]/\langle C_7 \rangle$.

1. $m(c_7) = c_7^6 + c_7^5 + c_7^4 + c_7^3 + c_7^2 + c_7 + 1$
2. $m(c_7) = c_7^2 + 5c_7 + 1$
   The Gröbner basis is very simple:
   $$\langle c_2 - 1, c_3 - 1, c_4 - 1, c_5 - 1, c_1 + c_6 + \omega + 4, c_6^2 + (\omega + 4)c_6 - \omega - 5 \rangle,$$
   but the computation is hard.
3. $m(c_7) = c_7^{12} - 5c_7^{11} + 24c_7^{10} - 115c_7^9 + 551c_7^8 - 2640c_7^7 + 12649c_7^6 - 2640c_7^5 + 551c_7^4 - 115c_7^3 + 24c_7^2 - 5c_7 + 1$

Table 1. Gröbner computations of $C_{7,\omega}$

	total	#basis	check	monic
1 over $\mathbf{Q}(\omega)$	9.3	268	0.2	1.4
1 over $\mathbf{Q}$	74	588	–	–
2 over $\mathbf{Q}(\omega)$	198	306	0	83
2 over $\mathbf{Q}$	119	675	–	–
3 over $\mathbf{Q}(\omega)$	256	306	0	128
3 over $\mathbf{Q}$	840	857	–	–

Example 4.

$$m_1(t_1) = t_1^7 - 7t_1 + 3,$$
$$m_2(t_1, t_2) = t_2^6 + t_1 t_2^5 + t_1^2 t_2^4 + t_1^3 t_2^3 + t_1^4 t_2^2 + t_1^5 t_2 + t_1^6 - 7.$$

$m_1(t_1)$ is irreducible over $\mathbf{Q}$. Let α_1 be a root of $m_1(t)$. $m_2(\alpha_1, t)$ is an irreducible factor of $m_1(t)$ over $\mathbf{Q}(\alpha_1)$ and let α_2 be a root of $m_2(\alpha_1, t)$.

$Cap = \{f_1, f_2, f_3, f_4\}$

$f_1 = (2ty - 2)x - (\alpha_1 + \alpha_2)zy^2 - z$

$f_2 = 2\alpha_2\alpha_1^4 zx^3 + (4ty + \alpha_2)x^2 + (4zy^2 + 4z)x + 2ty^3 - 10y^2 - 10ty + 2\alpha_1^2 + \alpha_2^2$

$f_3 = (t^2 - 1)x + (\alpha_2\alpha_1^4 + \alpha_2^3\alpha_1^3)tzy - 2z$

$f_4 = (-z^2 + 4t^2 + \alpha_2\alpha_1 + 2\alpha_2^3)zx + (4tz^2 + 2t^3 - 10t)y + 4z^2 - 10t^2 + \alpha_2\alpha_1^3$

Cap is constructed from *Caprasse* [2] by replacing its several coefficients by algebraic numbers in $\mathbf{Q}(\alpha_1, \alpha_2)$. In the computation over $\mathbf{Q}(\alpha_1, \alpha_2)$, almost all the time is spent on the check of the Gröbner basis candidate because the result contains many algebraic numbers with large integer coefficients. The computation over $\mathbf{Q}$ does not terminate within one hour.

Table 2. Gröbner computations of *Cap*

	total	#basis	check	monic
over $\mathbf{Q}(\alpha_1, \alpha_2)$	306	45	242	20
over $\mathbf{Q}$	$>$ 1hour	—	–	–

4.3 Discussion and Future Works

The results of our experiments show not only an advantage of our new method, but also show that further improvements are required. In Example 3-2, the computation via Theorem 1 is more efficient than the new method. We apply homogenization in both computations, but the results of homogenization differ according to the different settings of the weights, which affects the selection strategy and eventually the behaviors of the computations. The inefficiency of the new method in this example comes from large intermediate integer coefficients. They are still large even if we execute inter-reductions frequently, but they become small by the last inter-reduction executed after all S-pairs having the same total degree have been processed. Therefore it is possible to improve the new method if we have an efficient F_4-like implementation.

The current implementation requires that the each root is defined as a root of an irreducible polynomial and it is often hard to check the irreducibility. If we apply Dynamic Evaluation [3] we can weaken the requirement: the ideal generated by the defining polynomials is required to be only zero-dimensional and radical. Then the leading coefficient $\mathrm{HC}_R(\tilde{r})$ in Algorithm 1 is not necessarily invertible. If it is not invertible, we can split the ground ring by using the non-invertible element and the execution of Buchberger algorithm itself is split. Suppose that the ground ring R is represented as $\mathbf{Q}[t_1,\ldots,t_l]/J$ with a zero-dimensional radical ideal J. The following algorithm computes a set of pairs (J_i, r_i) such that $R = \oplus_i R_i$, $R_i = \mathbf{Q}[t_1,\ldots,t_l]/J_i$, $r_i \in R_i[x_1,\ldots,x_n]$ and $r_i = c_i\tilde{r}$ where c_i is the inverse of $\mathrm{HC}_{R_i}(\tilde{r})$ in R_i, thus r_i is monic.

Algorithm 6 (SPLIT_GROUND_RING$(J,\tilde{r})$)

```
loop
    $h \leftarrow \mathrm{HC}_R(\tilde{r})$
    if $h$ mod $J$ is invertible then
        $u(t) \leftarrow$ the inverse of $h$ mod $J$
        $r \leftarrow \mathrm{NF}_D(u\tilde{r})$
        return $\{(J,r)\}$
    else
        $J_1 \leftarrow J : h$
        $J' \leftarrow J + \langle h\rangle$
        $u_1(t) \leftarrow$ the inverse of $h$ mod $J_1$ ($h$ mod $J_1$ is invertible in $J_1$)
        $r_1 \leftarrow \mathrm{NF}_D(u_1\tilde{r})$
        $r' \leftarrow \tilde{r}$ mod $J'$
        $S' \leftarrow$SPLIT_GROUND_RING$(J', r')$
        return $\{(J_1,r_1)\} \cup S'$
    end if
end loop
```

By applying a new modular method proposed in [7], we can quickly check the invertibility of h mod J and compute $J : h$ and $J + \langle h\rangle$ if h mod J is not invertible. We plan to implement this method in near future.

References

1. Risa/Asir : A computer algebra system. http://www.math.kobe-u.ac.jp/Asir/asir.html.
2. SymbolicData. http://www.SymbolicData.org.
3. J. Della Dora, C. Discrescenso, D. Duval. About a new method for computing in algebraic number fields, In Proc. Eurocal'85 (LNCS 204), Springer-Verlag, 289-290, 1985.
4. C. Traverso. Gröbner trace algorithms. In Proc. ISSAC'88, LNCS **358**, Springer-Verlag 125-138, 1988.
5. M. Noro and J. McKay. Computation of replicable functions on Risa/Asir. In Proceedings of the Second International Symposium on Symbolic Computation PASCO'97, ACM Press, 130-138, 1997.
6. J. C. Faugère. A new efficient algorithm for computing Groebner bases (F_4). Journal of Pure and Applied Algebra (139) 1-3, 61-88, 1999.

7. M. Noro. Modular Dynamic Evaluation. To appear in Proc. ISSAC 2006, 2006.
8. T. Yan. The Geobucket Data Structure for Polynomials. J. Symb. Comp. 25,3, 285-294, 1998.
9. H. Schönemann. SINGULAR in a Framework for Polynomial Computations. M. Joswig and N. Takayama (eds.), Algebra, Geometry, and Software Systems, Springer, 163-176, 2003.

Tree Checking for Sparse Complexes

Massimo Caboara[1], Sara Faridi[2,⋆], and Peter Selinger[3,⋆]

[1] University of Pisa, Italy
caboara@dm.unipi.it
[2] Dalhousie University, Halifax, Canada
faridi@mathstat.dal.ca
[3] Dalhousie University, Halifax, Canada
selinger@mathstat.dal.ca

Abstract. We detail here the sparse variant of the algorithm sketched in [2] for checking if a simplicial complex is a tree. A full worst case complexity analysis is given and several optimizations are discussed. The practical complexity is discussed for some examples.

1 Introduction

The main goal of this paper is to give a detailed description and worst time complexity for a sparse variant of the tree-checking algorithm introduced in the paper [2]. For all the proofs in Sects. 2-3 we refer to [2].

Facet ideals were introduced in [4] as a method to study square-free monomial ideals, generalizing results in [9] and [8] on edge ideals of graphs. The idea is to associate a simplicial complex to a square-free monomial ideal, where each facet (maximal face) of the complex is the collection of variables that appear in a monomial in the minimal generating set of the ideal. The definition of a simplicial tree is a generalization of the concept of a graph-tree. Monomial ideals associated to trees have many properties that make them useful from an algebraic point of view [4,7].

In Sect. 2 we briefly recall the notation and results of [2]. In Sect. 3 we borrow from [2] a first version of the tree checking algorithm and we discuss its complexity. In Sect. 4 we present the main result of this paper, the full description of a variant of the tree checking algorithm optimized for sparse complexes and its worst case complexity. This variant has been briefly sketched in [2]. A brief subsection proposing further developments ends the paper.

Implementations. The algorithms described in this paper have first been coded in CoCoAL, the CoCoA system programming language (http://cocoa.dima.unige.it/). These prototypical implementations can be downloaded from [1]. Much more efficient (but less user friendly) C++ implementations have been developed for several versions of Algorithm 3.1 using the CoCoALib framework (http://cocoa.dima.unige.it/cocoalib/). The C++ code is also available at the website [1]. The code will be available in the CoCoA system from version 4.6 onwards. The code for the sparse variant is still a rough prototype, but already useful to test some example.

⋆ Research supported by NSERC.

© Springer-Verlag Berlin Heidelberg 2006

2 Simplicial Complexes and Trees

We define the basic notions related to facet ideals. More details and examples can be found in [4,5].

Definition 2.1 (Simplicial complex, facet). A *simplicial complex* $\varDelta$ over a finite set of vertices V is a collection of subsets of V, with the property that if $F \in \varDelta$ then all subsets of F are also in $\varDelta$. An element of $\varDelta$ is called a *face* of $\varDelta$, and the maximal faces are called *facets* of $\varDelta$.

Since we are usually only interested in the facets, rather than all faces, of a simplicial complex, it will be convenient to work with the following definition:

Definition 2.2 (Facet complex). A *facet complex* over a finite set of vertices V is a set $\varDelta$ of subsets of V, such that for all $F, G \in \varDelta$, $F \subseteq G$ implies $F = G$. Each $F \in \varDelta$ is called a *facet* of $\varDelta$.

The set of facets of a simplicial complex forms a facet complex. Conversely, the set of subsets of the facets of a facet complex is a simplicial complex. This defines a one-to-one correspondence between simplicial complexes and facet complexes. In this paper, we will work primarily with facet complexes.

Let k be a field. To a facet complex over a vertex set $\{v_1, \dots, v_n\}$, one can uniquely associate an ideal $\mathcal{F}(\varDelta)$ in the polynomial ring $k[x_1, \dots, x_n]$, where $\mathcal{F}(\varDelta)$ is generated by all the square-free monomials $x_{i_1} \dots x_{i_s}$, with $\{v_{i_1}, \dots, v_{i_s}\}$ a facet of $\varDelta$. This ideal is called the *facet ideal* of $\varDelta$.

From now on, we will often ease the notation by denoting facets of a complex by their corresponding monomials; for example, we write xyz for the facet $\{x, y, z\}$.

We now generalize some notions from graph theory to facet complexes. Note that a graph can be regarded as a special kind of facet complex, namely one in which each facet has cardinality 2.

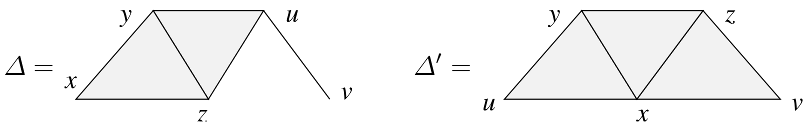
Definition 2.3 (Path, connected facet complex). Let $\varDelta$ be a facet complex. A sequence of facets $F_1, \dots, F_n$ is called a *path* if for all $i = 1, \dots, n-1$, $F_i \cap F_{i+1} \neq \emptyset$. We say that two facets F and G are *connected* in $\varDelta$ if there exists a path $F_1, \dots, F_n$ with $F_1 = F$ and $F_n = G$. Finally, we say that $\varDelta$ is *connected* if every pair of facets is connected.

Notation 2.1. If F, G and H are facets of $\varDelta$, $H \leqslant_F G$ means that $H \cap F \subseteq G \cap F$. The relation $\leqslant_F$ defines a preorder (reflexive and transitive relation) on the facet set of $\varDelta$.

Definition 2.4 (Leaf, joint). Let F be a facet of a facet complex $\varDelta$. Then F is called a *leaf* of $\varDelta$ if either F is the only facet of $\varDelta$, or else there exists some $G \in \varDelta \setminus \{F\}$ such that for all $H \in \varDelta \setminus \{F\}$, we have $H \leqslant_F G$.

It follows immediately from the definition that every leaf F contains at least one *free vertex*, i.e., a vertex that belongs to no other facet.

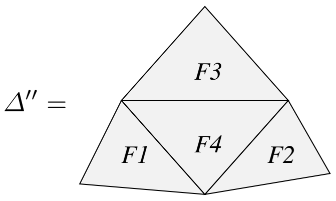
Example 2.1. In the facet complex $\Delta = \{xyz, yzu, uv\}$, xyz and uv are leaves, but yzu is not a leaf. Similarly, in $\Delta' = \{xyu, xyz, xzv\}$, the only leaves are xyu and xzv.



Definition 2.5 (Forest, tree). A facet complex Δ is a *forest* if every nonempty subset of Δ has a leaf. A connected forest is called a *tree* (or sometimes a *simplicial tree* to distinguish it from a tree in the graph-theoretic sense).

It is clear that any facet complex of cardinality one or two is a forest. When Δ is a graph, the notion of a simplicial tree coincides with that of a graph-theoretic tree.

Example 2.2. The facet complexes in Example 2.1 are trees. The facet complex Δ'' pictured below has three leaves F_1, F_2 and F_3; however, it is not a tree, because if one removes the facet F_4, the remaining facet complex has no leaf.



Alternatively, one could define a cycle and define a tree to be a connected complex that contains no cycles.

Definition 2.6 (Cycle). A *cycle* is a nonempty facet complex that has no leaf, but every proper subset of it has a leaf.

For example, the subcomplex $\{F_1, F_2, F_3\}$ is a cycle in Example 2.2, but the whole complex is not.

It follows immediately that a facet complex is a forest if and only if it contains no cycles.

2.1 Characterization of Trees

We now consider the problem of deciding whether or not a given facet complex is a tree.

Note that the naïve algorithm (namely, checking whether every non-empty subset has a leaf) is extremely inefficient: for a facet complex of n facets, there are $2^n - 1$ subsets to check. Also note that the definition of a tree is not inductive in any obvious way: for instance, attaching a single leaf to a tree need not yield a tree, as Example 2.2 shows. This seems to rule out an easy recursive algorithm.

Nevertheless, we demonstrate that the decision problem for simplicial trees can be solved efficiently. This is done via a characterization of trees given in this section.

Definition 2.7 (Paths and connectedness outside V). Let Δ be a facet complex, and let V be a set of vertices. We say that a sequence of facets $H_1, \dots, H_n \in \Delta$ is a *path outside* V in Δ if for all $i = 1, \dots, n-1$, $(H_i \cap H_{i+1}) \setminus V \neq \emptyset$. We say that two facets $F, G \in \Delta$ are *connected outside* V in Δ if there exists a path $H_1, \dots, H_n$ outside V in Δ such that $H_1 = F$ and $H_n = G$.

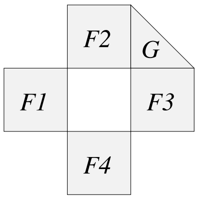
Note that in case $V = \emptyset$, this coincides with the definition of connectedness from Definition 2.3.

Notation 2.2. If F, G_1, G_2 are three distinct facets of Δ, then we define $\Delta_F^{G_1,G_2}$ to be the following subset of Δ:

$$\Delta_F^{G_1,G_2} = \{H \in \Delta \mid H \cap F = G_1 \cap G_2\} \cup \{G_1, G_2\}.$$

Definition 2.8 (Triple condition). Let Δ be a facet complex. We say a triple of facets $\langle F, G_1, G_2 \rangle$ satisfies the *triple condition* if $G_1 \not\leq_F G_2$ and $G_2 \not\leq_F G_1$, and if G_1 and G_2 are connected outside F in the facet complex $\Delta_F^{G_1,G_2}$.

Example 2.3. Consider the facet complex

$\Delta =$ 

The triple $\langle F_1, F_2, F_4 \rangle$ satisfies the triple condition. This is because $F_4 \not\leq_{F_1} F_2$ and $F_2 \not\leq_{F_1} F_4$. Moreover $\Delta_{F_1}^{F_2,F_4} = \{F_2, F_3, F_4, G\}$, and a path connecting F_2 and F_4 outside F_1 is F_2, F_3, F_4.

However, $\langle G, F_2, F_3 \rangle$ does not satisfy the triple condition, since $\Delta_G^{F_2,F_3} = \{F_2, F_3\}$, and F_2 and F_3 are not connected outside G.

Let Δ be a connected facet complex. Then the triple condition determines whether or not a triple of facets belongs to a cycle contained in Δ ([2] Proposition 4.5). In particular we have the following ([2] Theorem 4.6) criterion that determines if a given facet complex is a tree.

Theorem 2.3 (Main Theorem). *A connected facet complex Δ is a tree if and only if no triple of facets in Δ satisfies the triple condition.*

3 A Polynomial-Time Tree Decision Algorithm

By Theorem 2.3, to check if a facet complex $\Delta = \{G_1, \dots, G_l\}$ is a tree, we only need to check the triple condition for all triples of elements of Δ. The checks themselves are straightforward. Since the triple condition for $\langle F, G, G' \rangle$ is clearly unchanged if one switches G and G', we can limit triple checking to the elements of the set $\{\langle F, G_i, G_j \rangle \in \Delta^3 \mid G_i \neq F \neq G_j, i < j\}$. The procedures for the basic steps follow immediately from the earlier definitions.

Algorithm 3.1 (Standard algorithm)
Input: a connected facet complex $\Delta = \{G_1, \ldots, G_l\}$ with n vertices.
Output: **True** if Δ is a tree, **False** otherwise.

```
1. For each triple $\langle F, G, G' \rangle \in \{\langle F, G_i, G_j \rangle \in \Delta^3 \mid G_i \neq F \neq G_j, i < j\}$
   (a) If $G \leqslant_F G'$ or $G' \leqslant_F G$, continue with the next triple.
   (b) Build $\Delta_F^{G,G'}$.
   (c) If $G$ and $G'$ are connected outside $F$ in $\Delta_F^{G,G'}$, return False.
2. Return True.
```

The correctness of this algorithm is an immediate consequence of Theorem 2.3. The algorithm uses very little memory; the input Δ requires nl bits, and $\Delta_F^{G,G'} \subseteq \Delta$ requires l bits. The memory required to perform the connectedness check and to store the various counters is negligible. Thus, memory locality is good, and the computations can generally take place in the cache.

Remark 3.1. In the process of checking the triple condition for a triple $\langle F, G, G' \rangle$ that is part of a cycle, we build a connection path outside F. Clearly, any such path can be reduced to a *minimal* connected path $\{H_1, \ldots, H_n\}$ outside F for G, G', and $\{F, H_1, \ldots, H_n\}$ forms a cycle. Therefore, an easy modification of Algorithm 3.1 allow us to produce the set of all the facets $F \in \Delta$ that are part of some cycle, and a cycle $\Delta'_F \supseteq \{F\}$ for each of them.

3.1 Complexity

For each triple it is trivial to see that steps (a) and (b) can be performed with cost $O(n)$ and $O(nl)$ respectively. For step (c), the following holds.

Lemma 3.1 (Relation algorithm). *Let Δ be a facet complex with l facets over n variables such that F, G, G' are distinct facets of Δ. The connectedness outside F of $G, G' \in \Delta$ can be determined with time cost $O(nl)$.*

Proof. First of all we substitute Δ with the set $\{H - F \mid H \in \Delta\}$. We then define $n+1$ equivalence relations $P_0, \ldots, P_n$ on the set $\{1, \ldots, l\}$. P_0 is the identity relation, i.e., each equivalence class is a singleton. For each $j = 1, \ldots, n$, consider the vertex v_j and the set $X_j = \{i \mid v_j \in F_i\}$. Let P_j be the smallest equivalence relation such that $P_{j-1} \subseteq P_j$ and such that for all $i, i' \in X_j$, $(i, i') \in P_j$. Then facets F_i and $F_{i'}$ are connected if and only if $(i, i') \in P_n$. With a suitable data structure for representing equivalence relations (e.g., the relation associated to the partition $\{1\}$, $\{2, 3\}$,$\{4, 5, 6\}$ of $\{1, \ldots, 6\}$ can be represented by the array of integers $[1, 2, 2, 4, 4, 4]$), the complexity of the procedure above is $O(nl)$.

Consequently, step (c) of the tree decision algorithm can be performed at cost $O(nl)$. Thus, the total complexity of the tree decision algorithm is as follows: in the worst case we have to check $3 \cdot \binom{l}{3} = \frac{l(l-1)(l-2)}{2} = O\bigl(l^3\bigr)$ triples. The complexity of the steps (a)–(c) is $O(nl)$ and hence the total complexity of the algorithm is $O\bigl(nl^4\bigr)$.

Example 3.1. Consider the facet complex $\Delta = \{xy, xz, yz, yu, zt\}$. We have to check $3 \cdot \binom{5}{3} = 30$ triples. We start with the triple $\langle xy, xz, yz\rangle$.

- $xz \not\leq_{xy} yz$ since $xy \cap xz = x \not\subseteq y = xy \cap yz$. Similarly $yz \not\leq_{xy} xz$.
- xz and yz are connected outside xy in the complex $\Delta_{xy}^{xz,yz} = \{zt, xz, yz\}$.

We have hence discovered that Δ is not a tree. An unlucky choice of facets could have brought about the checking of 27 useless triples before this discovery, the other two useful triples being $\langle yz, xy, xz\rangle$ and $\langle xz, xy, yz\rangle$.

Remark 3.2. The relation algorithm for checking connectedness has very good worst case complexity. It is not so efficient in the average case, as shown below, see Table 2. Let us detail another algorithm for checking the connectedness of $F, G \in \Delta' = \{H_1, \ldots, H_l\}$: we examine $H_1, H_2, \ldots$ the elements of $\Delta - \{F, G\}$ and pose $F' = F$; if $H_i \cap F' \neq \emptyset$ we substitute F' with $F' \cup H_i$ and delete H_i from Δ. We repeat the previous procedure until $F' \cap G \neq \emptyset$ (F and G are connected) or $\Delta = \emptyset$ (F and G are not connected). Worst case complexity of this *list* algorithm is $O\bigl(nl^2\bigr)$, but we will see that it seems to be quite efficient in the average case.

3.2 Some Statistics

The facet complex $\{x_i x_{i+1} x_{i+2} \cdots x_{i+n} \mid i = 1, \ldots, m\}$ is trivially a tree. We call it the *line* n/m complex. The facet complex

$$\{yx_{11}x_{12}x_{13}, \ldots, yx_{1n-2}x_{1n-1}x_{1n}, \ldots, yx_{m1}x_{m2}x_{m3}, \ldots, yx_{mn-2}x_{mn-1}x_{mn}\}$$

is also trivially a tree, and we call it *star line* m/n. A random facet complex over $m + 10$ vertices, with m facets, each of which contains from n to k vertices, will be called *random* $m/n/k$ complex. It is extremely unlikely for such a complex to be a tree if $m > 30$ and $n, k > 5$.

Let us examine the connectedness check (Conn. Chk.) timings for the list and relation algorithms, compared to total timings, for some examples:

The list algorithm looks better in the average (sparse or almost sparse) case than the relation algorithm with respect to practical complexity. In the following examples, we will check connectedness with the list algorithm.

The following table gives the statistics for the checking of *every* triple for some random n/k examples. The Conn. Chk. and Triple. Cond. columns give the percentage of triples (against the total number of triples in both cases) that need a connectedness check or satisfy the triple condition respectively. The Tot. Time and Conn. Chk. Time columns give the total time spent and the time spent checking connectedness.

Table 1. List and Relation algorithms - Conn. Chk. Timings

Example	List algorithm Time Total/Conn. Chk.	Relation algorithm Time Total/Conn. Chk.
Rand.100/5/10	2.9s/0.5s	26.0s/23.6s
Rand. 200/5/10	21.2s/4.4s	467s/448s
Line 400/3	14.2s/1.1s	30.0s/16.3s
Line 400/40	115.1s/31.2s	2,418s/2,332s

Table 2. Random Examples

Example	Conn. Chk.	Triple. Cond.	Tot. Time	Conn. Chk. Time
Rand. 100/5/10	14%	14%	2.9s	0.6s
Rand. 100/20/40	100%	98%	9.4s	1.3s
Rand. 200/5/10	6%	6%	21.0s	4.4s
Rand. 200/20/40	94%	93%	132s	9.7s
Rand. 200/120/120	100%	100%	138s	10.2s
Rand. 400/5/20	7%	7%	625s	106s
Rand. 400/40/80	99%	99%	2,637s	90s

The more "dense" a random complex is, the higher the percentage of triples for which connectedness has to be checked and that satisfy the triple condition. It is exceedingly difficult for a random complex to be a tree, and the detection of a triple satisfying the triple condition is usually quite easy.

Tree examples are the hard cases, since every triple has to be checked.

Table 3. Tree Examples

Example	Conn. Chk.	Time
line 400/3	0.005%	14.2s
line 400/40	2%	115s
star line 4/100	0.01%	12.7s
star line 10/100	0.0003%	300s

3.3 Optimization

The runtime of Algorithm 3.1 can be improved by introducing some optimizations. First, note that if F is a facet such that no triple $\langle F, G, G'\rangle$ satisfies the triple condition, then F cannot be part of any cycle of Δ. Therefore, F can be removed from Δ, reducing the number of subsequent triple checks. We refer to this optimization as the *removal of useless facets*.

Remark 3.3. The facet order in a complex can be crucial when using the useless facet optimization, as shown in Table 4 below.

An important special case of a "useless facet" is a reducible leaf, as captured in the following definition:

Definition 3.1 (Reducible leaf). A facet F of a facet complex Δ is called a *reducible leaf* if for all $G, G' \in \Delta$, either $G \leqslant_F G'$ or $G' \leqslant_F G$.

A reducible leaf is called a "good leaf" by Zheng [10].

Remark 3.4. The facet F is a reducible leaf of Δ if and only if F is a leaf of every $\Delta' \subseteq \Delta$ with $F \in \Delta'$.

The remark immediately implies that a reducible leaf cannot be part of a cycle. Thus, it can be removed from Δ, and the algorithm can then be recursively applied to $\Delta \backslash \{F\}$. We were not able to find a tree without a reducible leaf; in fact, Zheng [10] conjectured that this is always the case. Checking whether a given facet F is a reducible leaf requires ordering all facets with respect to $\leqslant_F$, which takes $O(nl \log l)$ steps. A reducible leaf can thus be found in time $O\big(nl^2 \log l\big)$. Therefore, if Zheng's conjecture is true, the tree problem can be decided in time $O\big(nl^3 \log l\big)$. But even if the conjecture is not true, removing all reducible leaves at the beginning of Algorithm 3.1 is still a worthwhile optimization.

4 Optimization for Sparse Complexes

Let Δ be a facet complex with l facets. If every $F \in \Delta$ intersects a substantial ($\approx l$) number of facets, then the number of triples that satisfy the triple condition is probably high and our algorithm is usually able to detect one of them easily. If this does not happen, we can exploit the facet complex "sparseness" in our algorithm. For the remainder of this subsection, Δ will be a facet complex with l facets over n vertices.

4.1 Sparse Algorithm

Precomputing the incidence matrix for the graph describing the connectedness relation for the complex Δ allows us to use very efficient versions of all the sub procedures. The implementation of the sparse variant of the algorithm is still not complete, but we give here a full description, a complexity analysis and a prototype implementation, plus some examples.

Notation 4.1. Let Δ be a facet complex and $F \in \Delta$. We denote by d_F the cardinality of the set $\{H \in \Delta \mid H \cap F \neq \emptyset\}$. We denote by v_F the number of vertices in F.

Note that for $F \in \Delta$ we always have $d_F \leqslant l$ and $v_F \leqslant n$. Note also that the with a suitable implementation, costs for the intersection and equality/inequality operations for F, G are $O(\min\{v_F, v_G\})$.

Definition 4.1 (Connection block). Let $\varDelta = \{F_1, \ldots, F_l\}$ be a facet complex. The *connection block of* $\varDelta$, $(\mathrm{CB}_\varDelta)$, is the list of pairs $\langle i, \{j_1, \ldots, j_{d_{F_i}}\}\rangle$, $1 \leqslant i \leqslant l$ where $\{F_{j_1}, \ldots, F_{j_{d_{F_i}}}\}$ is the list of the facets connected to F_i. We call $\{j_1, \ldots, j_{d_{F_i}}\}$ in $\langle i, \{j_1, \ldots, j_{d_{F_i}}\}\rangle$ the i-th row of $CB_\varDelta$.

Note that $\mathrm{CB}_\varDelta$ is the incidence matrix for the graph describing the connectedness relation for the complex $\varDelta$. We denote the sum of the cardinality of the i-th rows of $\mathrm{CB}_\varDelta$ by E. Note that $E = \sum_{i=1}^{l} d_{F_i} = 2 \cdot \#(\text{edges in the graph})$. The space required to store $\mathrm{CB}_\varDelta$ is $O(E)$.

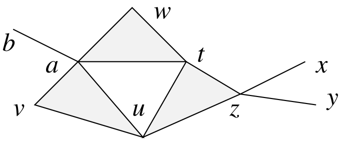
Notation 4.2. Let $\varDelta$ be a facet complex and $F \in \varDelta$. We denote by $\mathrm{CB}_\varDelta^F$ the connection block of the facet complex $\varDelta$ when the connectedness relation is replaced by the connectedness outside F relation.

The space required to store $\mathrm{CB}_\varDelta^F$ is less or equal than the space necessary to store $\mathrm{CB}_\varDelta$.

If we have $\varDelta, \mathrm{CB}_\varDelta$,and $\varDelta' \subset \varDelta$ and we want to build $\mathrm{CB}_{\varDelta'}$ we can do that efficiently by marking in $\varDelta$ all the elements in $\varDelta - \varDelta'$. When using $\mathrm{CB}_{\varDelta'}$ to check connectedness, we work in $\mathrm{CB}_\varDelta$ but we only consider indices whose associated facet in $\varDelta$ is not marked. Marking, mark erasing and mark checking for a facet F_i in $\varDelta$ can be done in constant time if we know the index i.

Example 4.1. Let $\varDelta$ be the facet complex

$$\{F_1 = xz, F_2 = yz, F_3 = ztu, F_4 = twa, F_5 = uva, F_6 = ab\}$$

$\varDelta =$ 

Then
$\mathrm{CB}_\varDelta = \{\langle 1, \{2, 3\}\rangle, \langle 2, \{1, 3\}\rangle, \langle 3, \{1, 2, 4, 5\}\rangle, \langle 4, \{3, 5, 6\}\rangle, \langle 5, \{3, 4, 6\}\rangle, \langle 6, \{4, 5\}\rangle\}$
$\mathrm{CB}_\varDelta^{F_3} = \{\langle 1, \emptyset\rangle, \langle 2, \emptyset\rangle, \langle 3, \emptyset\rangle, \langle 4, \{5, 6\}\rangle, \langle 5, \{4, 6\}\rangle, \langle 6, \{4, 5\}\rangle\}$
$\varDelta_{F_3}^{F_4, F_5} = \{F_4, F_5\}$ and $\mathrm{CB}_{\varDelta_{F_3}^{F_4, F_5}} = \{\langle 4, \{5\}\rangle, \langle 5, \{4\}\rangle\}$.
We have $F_6 \notin \varDelta_{F_3}^{F_4, F_5} = \{F_4, F_5\}$. Equivalently, we can mark F_6 in $\varDelta$ and avoid to consider 6 in $\mathrm{CB}_\varDelta^{F_3}$, the relation described by the incidence matrices is the same.

Algorithm 4.3 (Tree decision sparse algorithm)
Input: a connected facet complex $\varDelta = \{F_1, \ldots, F_l\}$ with n vertices.
Output: **True** if $\varDelta$ is a tree, **False** otherwise.

```
1. Build $\mathrm{CB}_\varDelta$.
2. For each $i$ s.t. $F_i \in \varDelta$
   (a) Build $\mathrm{CB}_\varDelta^{F_i}$.
```

   (b) For each $\langle G, G'\rangle \in \{\langle G_j, G_k\rangle \mid G_j, G_k \in i\text{-th row in } \mathrm{CB}_\Delta \text{ s.t. } j < k\}$
      i. If $G \leqslant_{F_i} G'$ or $G' \leqslant_{F_i} G$, continue with the next couple.
      ii. Build $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$ from $\mathrm{CB}_\Delta^{F_i}$ by marking Δ.
      iii. If G and G' are connected outside F_i in $\Delta_{F_i}^{G,G'}$ return **False**.
      iv. Erase all marks in Δ.
3. Return **True**.

Note that we build $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$ without building $\Delta_{F_i}^{G,G'}$ by using the previously computed incidence matrices.

Let us describe the algorithms for the sub procedures and their costs when not trivial.

1. Building CB_Δ: for every element $F_i \in \Delta$ we check if any other element $G \in (\Delta - F)$ is connected to F_i. Cost is $O\big(l^2 v_{F_i}\big)$.
2. Building $\mathrm{CB}_\Delta^{F_i}$ having CB_Δ: for every s-row of CB_Δ and every element in the row $(H_1, \ldots, H_{d_{F_s}})$ we check if the intersection holds outside F_i. This check can be done for every H with time cost v_H, and the total cost is hence $l \sum_{j=1}^{d_{F_s}} v_{H_j}$.
3. The cost of step 2(b).i is trivially $O(v_{F_i})$
4. Building $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$ having CB_Δ and $\mathrm{CB}_\Delta^{F_i}$: we don't actually build $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$ but we use $\mathrm{CB}_{\Delta_{F_i}}$ marking the facets in Δ that does not appear in $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$. Instead of looking for a connected path in $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$ outside F_i, we look equivalently for a connected path in $\mathrm{CB}_{\Delta_{F_i}}$ outside F_i whose links are not marked in Δ.
   There are two cases, $G \cap G' = \emptyset$ and $G \cap G' \neq \emptyset$.
   $G \cap G' = \emptyset$: we have to mark the facets $H \in \Delta$ for which $H \cap F_i \neq \emptyset$. These are the d_{F_i} elements in i-th row of CB_Δ. Cost is d_{F_i}.
   $G \cap G' \neq \emptyset$: we have to mark the facets $H \in \Delta$ for which $H \cap F_i \neq G \cap G'$. These are the elements outside i-th row of CB_Δ ($E - d_{F_i}$ markings) and the elements in the i-th row for which $H \cap F_i \neq G \cap G'$. ($d_{F_i}$ checks at cost v_{F_i} and at most d_{F_i} markings) The cost is hence $O(d_{F_i} v_{F_i} + E)$.
   Total cost for step 2(b).ii is thus $O(d_{F_i} v_{F_i} + E)$.
5. To check if G and G' are connected in $\Delta_{F_i}^{G,G'}$ outside F_i: having $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$ the problem reduces to check connectedness in a graph with $O(E)$ edges, and that can be done with the well known Breadth First Search technique at cost $O(E)$.
   Alternative: connection check using $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$: We start from the elements in the G row of $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$ in a dequeue list L. For every unmarked H there, we check if it is G', in which case we return true, we mark H in the complex and add to L the elements in the H row of $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$. At most l merging at unitary cost, at most E elements in L and E marking setting/checking. Total cost is hence $O(E)$.

This algorithm is a straightforward application of Theorem 2.3. Its complexity is as follows:

We build CB_Δ once (cost $O(lE)$), then for every $i \in \{1, \ldots, l\}$ we build $\mathrm{CB}_\Delta^{F_i}$ at cost $v_{F_i}E$ and we perform steps 2(b).i-iv $\binom{d_{F_i}}{2}$ times, at cost $O(d_{F_i}v_{F_i} + E)$ for every iteration. The dominant cost is this last step. Total complexity is hence

$$O\left(\sum_{i=1}^{l}(d_{F_i}^3 v_{F_i} + d_{F_i}^2 E)\right) = O\left(\sum_{i=1}^{l}\left(d_{F_i}^3 v_{F_i} + d_{F_i}^2 \sum_{s=1}^{l} d_{F_s}\right)\right)$$

– If the complex is not sparse ($v_{F_i} \approx n$, $d_{F_i} \approx l$ for every $i \in \{1, \ldots, l\}$) then Sparse Algorithm complexity is equal to Standard Algorithm complexity, $O(nl^4)$.
– If $d_{F_i}, v_{F_i} < \sqrt{l}$, a possible definition of sparseness for a complex, then Sparse Algorithm complexity is $O\left(l^3\sqrt{l}\right)$.

4.2 Some Statistics

The line 400/3 example is a sparse tree, and the line 400/40 example is an almost sparse tree. The star line examples are trees but they are not sparse. Permuted means that the facet are fed to the algorithm not in the "natural" order but after a random reshuffle. The R stands for the removal of useless facets optimization.

Table 4. Standard Alg./Sparse Alg. Comparison

Example	Standard	Standard+R	Sparse	Sparse+R
line 400-3	14.2s	4.3s	1.53s	0.01s
line 400-3 Permuted	12.8s	12.8s	0.86s	0.85s
line 400-40	115s	3.2s	100s	0.2s
line 400-40 Permuted	97s	89s	80s	80s
star-line 4/100	12.7s	3.7s	14.2s	4.5s
star-line 4/100 Permuted	12.4s	12.7s	14.1s	13.3s
star-line 10/100	300s	89s	318s	99s
star-line 10/100 Permuted	290s	279s	309s	296s

For sparse examples the sparse algorithm is clearly better than the standard algorithm; as expected, this is not the case when the sparseness is lacking. The useless facet removal optimization is very sensitive to facet ordering, but is very useful when the conditions are suitable.

5 Conclusions and Further Work

Large sparse trees are the hardest cases for our tree checking algorithm. The sparse algorithm looks very promising for these cases, especially from the point of view of its practical complexity.

We are working on the further optimization of the sparse algorithm. Using this algorithm for checking examples, we will work on Zheng's conjecture. We will then compare the sparse algorithm and the algorithm based on Zheng's conjecture performance for sparse complexes.

References

1. M. Caboara, S. Faridi, P. Selinger, Prototype implementation of tree algorithms, available at `http://www.dm.unipi.it/∼caboara/Research`.
2. M. Caboara, S. Faridi, P. Selinger. *Simplicial cycles and the computation of simplicial trees*. Journal of Symbolic Computation, to appear.
3. CoCoA Team, *CoCoA: a system for doing Computations in Commutative Algebra*, available at `http://cocoa.dima.unige.it/`.
4. S. Faridi, *The facet ideal of a simplicial complex*, Manuscripta Mathematica 109 (2002), 159–174.
5. S. Faridi, *Cohen-Macaulay properties of square-free monomial ideals*, Journal of Combinatorial Theory, Series A, 109 (2005), no. 2, 299–329.
6. S. Faridi, *Simplicial trees are sequentially Cohen-Macaulay*, J. Pure and Applied Algebra 190 (2004), 121–136.
7. S. Faridi, *Monomial ideals via square-free monomial ideals*, Lecture Notes in Pure and Applied Mathematics, to appear.
8. A. Simis, W. Vasconcelos, R. Villarreal, *On the ideal theory of graphs*, J. Algebra 167 (1994), no. 2, 389–416.
9. R. Villarreal, *Cohen-Macaulay graphs*, Manuscripta Math. 66 (1990), no. 3, 277–293.
10. X. Zheng, *Homological properties of monomial ideals associated to quasi-trees and lattices*, Ph.D. thesis, Universität Duisburg-Essen, August 2004.

The SARAG Library: Some Algorithms in Real Algebraic Geometry

Fabrizio Caruso

Università di Pisa, Dipartimento di Matematica "L. Tonelli",
Largo Bruno Portecorvo 5, 56127 Pisa, Italy
`caruso@posso.dm.unipi.it`

Abstract. In this paper we present *SARAG*, which is a software library for real algebraic geometry written in the free computer algebra system Maxima. *SARAG* stands for "Some Algorithms in Real Algebraic Geometry" and has two main applications: extending the capabilities of Maxima in the field of real algebraic geometry and being part of the interactive version of the book "Algorithms in Real Algebraic Geometry" by S. Basu, R. Pollack, M.-F. Roy, which can be now freely downloaded.

The routines of the library deal with: theory of signed sub-resultants, linear algebra, gcd computation, real roots counting, real roots isolation, sign determination, Thom encodings, study of the topology of curves. At the moment *SARAG* is being used as a tool to develop, implement and tune algorithms coming from new research results, e.g. an algorithm for faster gcd computation, an algorithm for the study of the topology of curves over non-Archimedian real closed fields.

Introduction

In this paper we present the *SARAG* software library, which stands for "Some Algorithms in Real Algebraic Geometry". Initially *SARAG* was developed as an add-on library for real algebraic geometry for the free and open source computer algebra system Maxima [13]. Now it is included in Maxima as a standard package.

The main goals of the package at the moment are

- extending Maxima with new routines in the field of real algebraic geometry,
- being used in the interactive version of the book "Algorithms in Real Algebraic Geometry" by S.Basu, R.Pollack, M.-F.Roy [2].

Therefore there are two ways to use it: either within Maxima or within the book [2]. Both the book and the library can be freely downloaded from http://name.math.univ-rennes1.fr/marie-francoise.roy/bpr-posted1.html.

The book uses TeXmacs [6] both as a viewer and as a front-end for Maxima [13] and SARAG. There have been somewhat similar books in the past. For example [11], which is a book on symbolic summation that in its electronic format (`pdf`) is freely downloadable but it is not interactive; and [4], which is a commercial interactive book with an algorithmic flavor on algebra for undergraduate students, where `html`, `Java` and the `GAP` [8] computer algebra system are used.

© Springer-Verlag Berlin Heidelberg 2006

The main routines of *SARAG* deal with problems related to computations with polynomials whose coefficients are in a real closed field (for a definition we refer to Sect. 2.1 of [2]), in particular: Gaussian elimination, characteristic polynomial, signed sub-resultants and signed remainder sequences, univariate gcd, Cauchy index and real roots counting, real roots isolation, sign determination, Thom encodings, topology of plane curves.

The library contains some linear algebra routines that perform better than standard Maxima (see Sect. 2.6).

At the moment *SARAG* is being used as a tool to develop, implement and tune algorithms coming from new research results, e.g. an algorithm for faster gcd computation, an algorithm for the study of the topology of curves over non-Archimedian real closed fields.

1 Some Problems and Algorithms

In this section we describe very briefly some of the problems and the algorithms implemented in the library. In particular we will consider the problems related to the topics of real roots counting, real roots isolation, sign determination, study of the topology of a curve.

1.1 Real Roots Counting

The problem of real roots counting is the problem of counting the real roots of a univariate polynomial disregarding the multiplicity.

A similar and closely related problem treated in the library is the problem of computing the *Tarski query.*

Definition 1. *Given the polynomials p, q and an open interval (a, b), the* Tarski query *for q with respect to p in (a, b) is*

$$\begin{aligned} TaQ(q, p, a, b) :=& |\{x \in (a, b) | p(x) = 0, q(x) > 0\}| - \\ & |\{x \in (a, b) | p(x) = 0, q(x) < 0\}|. \end{aligned} \tag{1}$$

In the library the following three methods for counting the real roots of a polynomial have been implemented:

1. signed remainder sequence,
2. signed sub-resultant sequence,
3. counting by isolation.

Signed Remainder Sequence. The first approach boils down to computing the Sturm's sequence, (i. e., a signed remainder sequence for the polynomial and its derivative) and counting the number of sign variations of the polynomials in the sequence evaluated at the ends of the considered interval. (see Theorem 2.50 in Sect. 2.2.2 of [2]).

Signed Sub-resultant Sequence. A more efficient approach is based on the principal signed sub-resultant coefficient sequence (see Sect. 4.2.2 in [2]). The number of roots is obtained by a generalized difference of the sign permanences and sign variations where the zero signs of the sequence are counted in a special way (for more details we refer to Chap. 8 and Chap. 9 of [2]).

Counting by Isolation. Counting can be done clearly also by first isolating the real roots with points and open intervals and then by counting the number of such points and intervals (see next section for a brief description of an algorithm for isolating the real roots).

1.2 Real Roots Isolation

The problem of real roots isolation is the problem of finding points and open intervals that describe all the real roots of a given univariate polynomial in an Archimedian real closed field.

The algorithm implemented in the package uses a conversion to the Bernstein basis for univariate polynomials with respect to a given interval and the *de Casteljau algorithm* for quickly producing polynomials in the Bernstein bases for sub-intervals.

Definition 2. *The Bernstein polynomials of degree p for the interval (l, r) are*

$$Bern_{p,i}(l,r) := \binom{p}{i}\frac{(x-l)^i(r-x)^{p-i}}{(r-l)^p},\ i = 0, \dots, p. \tag{2}$$

The criterion used to determine when an open interval contains exactly one real root is based on the following result:

Theorem 1. *Given a square free polynomial P in $R[x]$, where R is an Archimedian real closed field, we have:*

- *If P has no root in $C(l, r)$ then*

$$Var(P, (l, r)) = 0,$$

- *if P has exactly one root in $C_1(l, r) \cup C_2(l, r)$ then*

$$Var(P, (l, r)) = 1.$$

where

- *$Var(P, (l, r))$ is the number of sign variations in the list of coefficients in the Bernstein basis for P in the interval (l, r),*
- *$C(l, r)$ is the circle with diameter (l, r) and $C_1(l, r)$, $C_2(l, r)$ are the circles circumscribing the equilateral triangles based on (l, r) in which we identified R^2 with $C = R[i]$.*

For more details we refer to Sect. 10.1 and 10.2 of [2]. Similar problems and various applications of the Bernstein basis are treated extensively by [7].

1.3 Sign Determination

The problem of sign determination is the problem of studying the possible sign configurations of a set of polynomials at the roots of a given polynomial. The algorithm used in the library works for a general, possibly non-Archimedian real closed field.

The algorithm involves solving a linear system and the computation of *Tarski queries* (see Definition 1). For a description of this algorithm we refer to Sect. 10.3 of [2]. The basic idea of the algorithm appears in [1].

Thom Encodings. An application of sign determination is the computation of the Thom encodings for the roots of a polynomial in a general, possibly non-Archimedian real closed field, namely the sign determination applied to the study of the signs of all the derivatives of a polynomial at the roots of the polynomial. Thom encodings are important because they can be used to characterize and order the roots of a polynomial over a non-Archimedian real closed field (this idea appears in [5]).

1.4 Topology of a Curve

Given a bivariate polynomial $P[x, y]$ over an Archimedian real closed field, we want to compute a plane graph homeomorphic to the variety described by $P[x, y]$.

In particular we want to find

- the number of intersections that the curve has with vertical projections,
- if the curve has a *critical point*, i. e., a point $(\bar{x}, \bar{y})$ such that $\bar{y}$ is a multiple root of $P[\bar{x}, y]$, then we also want to find its relative position with respect to the other intersections with the vertical projection at $\bar{x}$.

We notice that the x-coordinates of the critical points are the roots of the *discriminant* (for a definition of *discriminant* see Sect. 4.1 in [2]).

The algorithm implemented in the library involves

- isolation of the roots of the discriminant $Discr(x)$ of $P[x, y]$ with respect to y by conversion into the Bernstein basis (see Definition 2) and by the *de Casteljau algorithm* (see Theorem 1),
- counting the number of intersections with vertical projections between roots of $Discr(x)$ and counting of the number of intersections with vertical projection above and below *critical points*, which is achieved by *Tarski query* computations (see Definition 1).

For some pictorial examples see Sect. 2.4. For a detailed description of this algorithm we refer to Sect. 11.6 of [2].

2 *SARAG* as Part of Maxima

SARAG as a software library can be used within Maxima. The first versions of the library were a separate Maxima package that the user had to download.

The latest versions are now included in standard Maxima (experimentally since version 5.9.3).

Here we give a few details on how to load, on the naming conventions, on the most important routines and on the plotting of topological graphs. For an extensive manual we refer to [3].

2.1 Loading the Library

The latest version of Maxima come with *SARAG* included as a standard package. In this case the library is loaded by simply starting Maxima and giving the command: `load("sarag/sarag");`. For previous versions of Maxima or to use the latest version of *SARAG* downloaded separately follow the instructions in [3].

2.2 The Commands

The names of the *SARAG* routines take into account: the algorithm or method used (*method*), what is to be computed (*problem*) and if a variation is to be used (*attribute*). This is achieved by composing the names of routines with the following structure: **method+problem+attribute**. For a detailed description of the commands we refer to [3].

2.3 Some Examples

Here we give a list of some commands of the library also used in [2]:

- `gaussDet(`*M*`)`
  computes the determinant of a matrix by Gaussian elimination
- `sSubResNumberOfRoots(`*pol*`,`*var*`)`
  counts the roots of *pol* by using the signed sub-resultant sequence
- `deCasteljauIsolateRoots(`*pol*`,`*var*`)`
  isolates the roots of *pol* by the de Casteljau method
- `smartSignDetermination(`*pol_sequence*`,`*pol*`,`*taQ*`,`*var*`)`
  it computes the sign configurations at the roots of *pol* by an optimized algorithm (see Sect. 10.3 of [2]) using *taQ* as subroutine for the computation of the *Tarski query*
- `archimedianTopology(`*pol*`,`*isol_algorithm*`,`*x_var*`,`*y_var*`)`
  it computes the topology of *pol* along the x-axis, using *isol_algorithm* as a subroutine for isolating roots
- `drawTopology(`*top_out*`)`
  draws the topological graph for *top_out* using *GnuPlot* [9], (where *top_out* is the second element in the output of `archimedianTopology`)

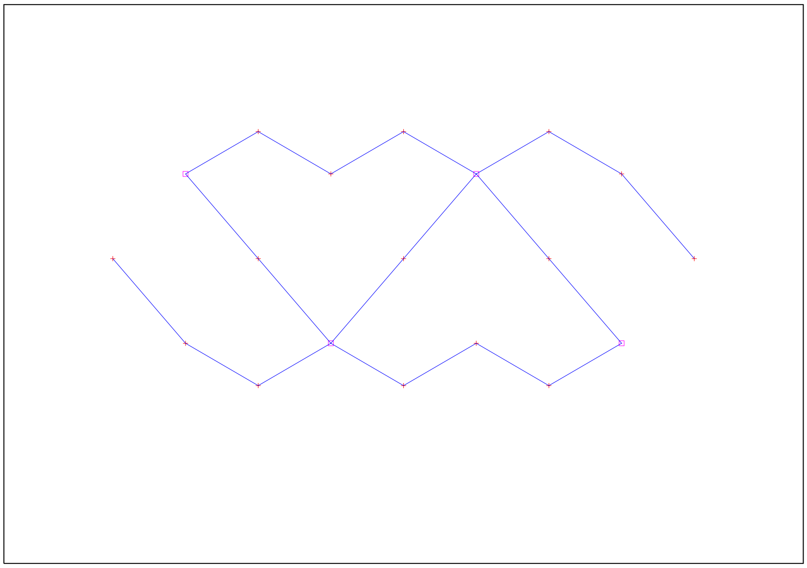
2.4 Pictures of Topological Graphs

The command `drawTopology` draws the topological graph from the output of `archimedianTopology` using *Gnuplot* [9].

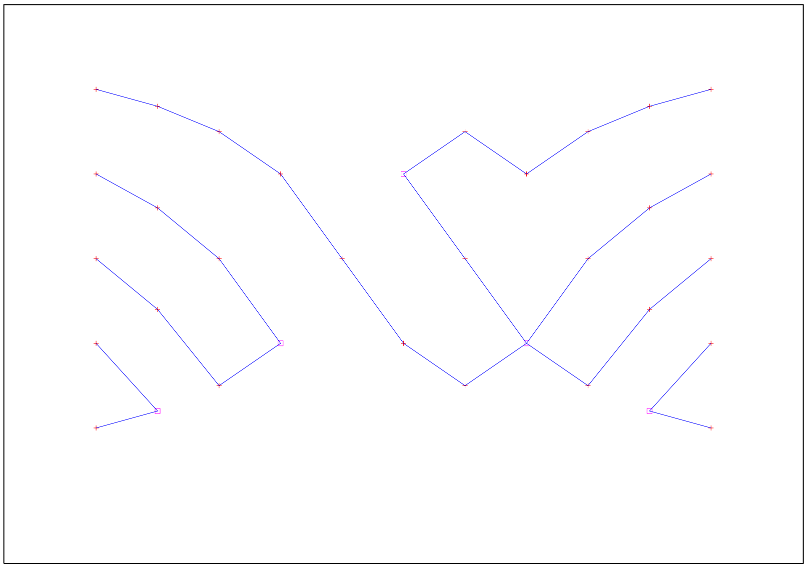
For $f := (x - y)(y^2 + x^2 - 1)$ the *SARAG* commands

```
t : archimedianTopology(f,deCasteljauIsolateRoots,x,y);
drawTopology(t[2]);
```

produce:

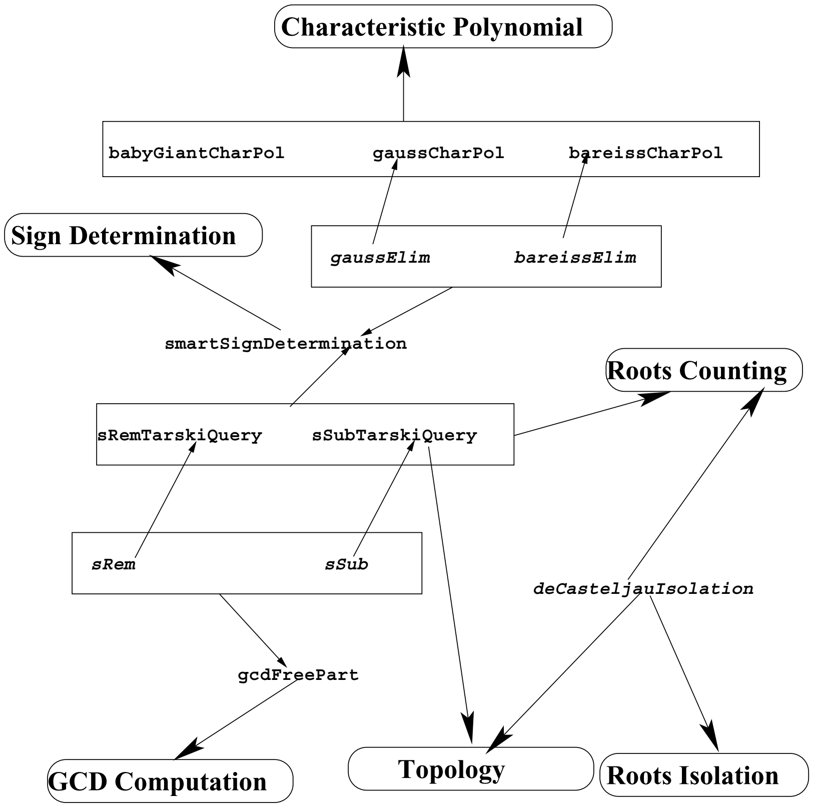


for $f := y^5 + (-x - 1)y^4 + (1 - 2x)y^3 + (2x + 1)y^2 + (2x - 1)y - x - 1$



2.5 The Structure of the Library

The structure of the library can be summarized by the following directed graph, whose sources (in oblique script) are the lowest level functions (Gaussian elimination, signed sub-resultants, signed remainder sequences, de Casteljau's algorithm), and where the circled nodes represent the applications:



2.6 Performance of Linear Algebra

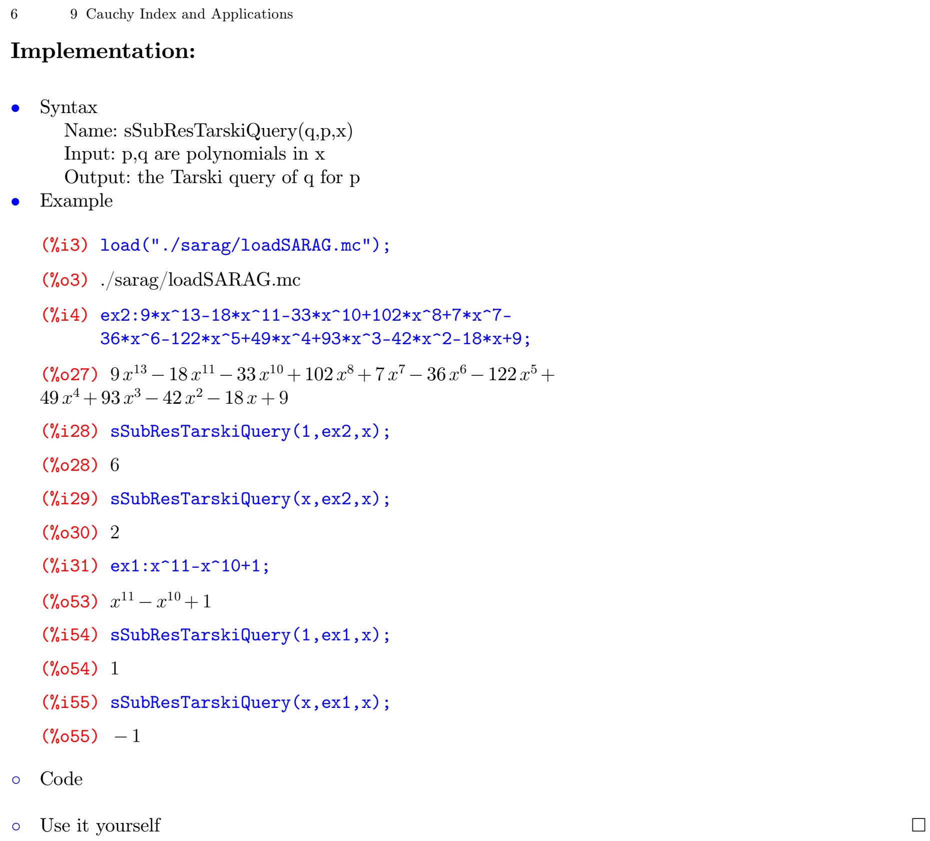
In the library we implement some routines for linear algebra that perform better than standard Maxima. For example *SARAG* performs the matrix triangulation[1] better for medium and large size matrices:

`Problem`	*SARAG*	standard built-in
`Hilbert(130)`	384.81s	359.01s
`Hilbert(140)`	481.88s	587.67s
`Hilbert(150)`	593.59s	934.76s
`Hilbert(160)`	717.16s	1407.36s

[1] The problem considered is the computation of the lower triangular form of Hilbert matrices. We used the command `triangularized` for *SARAG* and the command `gaussElim` for standard Maxima. The computer used was a Pentium IV, 2.5 Ghz, 1 Gb RAM running Maxima 5.9.2 under Linux.

3 *SARAG* as Part of a Book

An alternative way of using the package is through the interactive book "Algorithms in Real Algebraic Geometry" [2]. See the excerpt below:



6 9 Cauchy Index and Applications

Implementation:

- Syntax
  Name: sSubResTarskiQuery(q,p,x)
  Input: p,q are polynomials in x
  Output: the Tarski query of q for p
- Example

(%i3) `load("./sarag/loadSARAG.mc");`

(%o3) ./sarag/loadSARAG.mc

(%i4) `ex2:9*x^13-18*x^11-33*x^10+102*x^8+7*x^7-36*x^6-122*x^5+49*x^4+93*x^3-42*x^2-18*x+9;`

(%o27) $9\,x^{13}-18\,x^{11}-33\,x^{10}+102\,x^{8}+7\,x^{7}-36\,x^{6}-122\,x^{5}+49\,x^{4}+93\,x^{3}-42\,x^{2}-18\,x+9$

(%i28) `sSubResTarskiQuery(1,ex2,x);`

(%o28) 6

(%i29) `sSubResTarskiQuery(x,ex2,x);`

(%o30) 2

(%i31) `ex1:x^11-x^10+1;`

(%o53) $x^{11}-x^{10}+1$

(%i54) `sSubResTarskiQuery(1,ex1,x);`

(%o54) 1

(%i55) `sSubResTarskiQuery(x,ex1,x);`

(%o55) -1

- ○ Code
- ○ Use it yourself □

SARAG contains nearly all the algorithms contained in Chap. 8,9,10,11 of [2]. The book uses TeXmacs [6] as a viewer and as a front-end for Maxima[13] and *SARAG*. Therefore, in order to use the book it is necessary to install TeXmacs and Maxima. The TeXmacs file contains the text of the book as in the printed version and for each algorithm for which an implementation exists in the library an openable clickable window with the following fields: instruction on how to use the corresponding *SARAG* function, an example, a link to the code and a Maxima session with preloaded *SARAG* ready for the user/reader to try his/her own examples.

4 The Future

The future of this library is threefold:

1. include more algorithms contained in the interactive book,
2. provide Maxima with more functions in the field of real algebraic geometry,
3. use the library as a base for new algorithms.

4.1 More Algorithms from "Algorithms in Real Algebraic Geometry"

The library does not still contain all the algorithms in the book [2] covered by Chap. 8,9,10,11 like cylindrical algebraic decomposition. The library could also include algorithms from the remaining chapters.

4.2 Further Extend Maxima

The library could be further extended including algorithms which are not presented in [2] like

– alternative algorithms for real roots isolation, e. g., using the monomial basis (see [14] and [12]),
– semi-numerical computation of the topology for curves (see [10]),
– better linear algebra (optimized pivoting, improved Bareiss' algorithm).

Moreover *SARAG* could be better integrated in Maxima, e. g., including a test file and an in line manual in Maxima, substituting those parts of standard Maxima that are slower than *SARAG*, etc.

4.3 Support for New Algorithms

SARAG is now being used to investigate, benchmark and tune new algorithms in the areas of real algebraic geometry and computational algebra. In particular it is being used to develop and test new algorithms for fast univariate gcd computation and the study of the topology of plane curves over non-Archimedian real closed fields.

Acknowledgments

I would like to thank M.-F. Roy and L. Gonzalez-Vega for their friendly support and their very helpful suggestions.

References

1. M. Ben-Or, D. Kozen, and J. Reif. The complexity of elementary algebra and gemetry. *J. of Computer and Systems Sciences*, 18:251–264, 1986.
2. S. Basu, R. Pollack, and M-F. Roy. *Algorithms in Real Algebraic Geometry.* Springer, 2003. http://name.math.univ-rennes1.fr/marie-francoise.roy/.
3. F. Caruso. *The SARAG Library for Real Algebraic Geometry.* http://name.math.univ-rennes1.fr/marie-francoise.roy/bpr-posted1.html.
4. A.M. Cohen, H.Cuypers, and H.Sterk. *Algebra Interactive.* Springer, 1999.
5. M. Coste and M.-F. Roy. Thom's lemma, the coding of real algebraic numbers and the topology of semi-algebraic sets. *Journal of Symbolic Computation*, 5(1/2):121–129, 1988.

6. Joris Van der Hoeven. *TeXmacs*. Availabale at http://www.texmacs.org.
7. G. Farin. *Curves and surfaces for Computer Aided Design*. Academic Press, 1990.
8. *GAP*. http://www.gap-system.org/.
9. *Gnuplot*. http://www.gnuplot.info/.
10. L. Gonzalez-Vega and Iona Necula. Efficient topology determination of implicitly defined algebraic plane curves. *Computer Aided Geometric Design*, 19:719–743, 2002.
11. M. Petkovšek, H. Wilf, and D. Zeilberger. *A=B*. A K Peters, MA, 1997.
12. F. Roullier and P. Zimmermann. Efficient isolation of a polynomial real root. *Journal of Computational and Applied Mathematics*, 162(1):33–50, 2003.
13. W. F. Schelter. *Maxima on line documentation*. Available at http://maxima.sourceforge.net/.
14. J.V. Uspensky. *Theory of equations*. MacGraw Hill, 1948.

Algebraic Computation of Some Intersection D-Modules

F.J. Calderón Moreno and L. Narváez Macarro[⋆]

Universidad de Sevilla, P.O. Box 1160. 41080 Sevilla, Spain
{calderon, narvaez}@algebra.us.es

Abstract. Let $D \subset \mathbf{C}^n$ be a locally quasi-homogeneous free divisor (e.g. a free hyperplane arrangement), $\mathcal{E}$ an integrable logarithmic connection with respect to D and $\mathcal{L}$ the local system of horizontal sections of $\mathcal{E}$ on $X - D$. Let $\mathrm{IC}_X(\mathcal{E})$ be the holonomic regular $\mathcal{D}_X$-module whose de Rham complex is the intersection complex $\mathrm{IC}_X(\mathcal{L})$ of Deligne-Goresky-MacPherson. In this paper we show how to use our previous results on the algebraic description of $\mathrm{IC}_X(\mathcal{E})$ in order to obtain explicit presentations of it. Concrete examples for $n = 2$ are included.

Introduction

Intersection complexes can be constructed by an important operation: the intermediate direct image. Its description in terms of Verdier duality and usual derived direct images can be algebraically interpreted in the category of holonomic regular D-modules by using the deep properties of the de Rham functor. We need to compute localizations and D-duals.

This can be effectively done, in principle, by using the general available algorithms in [19,21,20], but in the case of integrable logarithmic connections along a locally quasi-homogeneous free divisor, we exploit the logarithmic point of view [2,4,5,7,8,23,24] and we use a general algebraic description of their associated intersection D-modules, from which we can easily derive effective computations.

The main ingredients we use are our previous results in [5] and [6].

The algorithmic treatment of the computations in this paper will be developed elsewhere.

Let us now comment on the content of this paper.

In section 1 we remind the reader of the basic notions and notations and we review our previous results on logarithmic $\mathcal{D}$-modules with respect to free divisors. We recall the logarithmic comparison theorem for arbitrary integrable logarithmic connections from [6], and we give the theorem describing the intersection D-module associated with an integrable logarithmic connection along a locally quasi-homogeneous free divisor.

In section 2, given a locally quasi-homogeneous free divisor D with a reduced local equation $f = 0$ and a cyclic integrable logarithmic connection $\mathcal{E}$ with respect to D, we explicitly describe a presentation of $\mathcal{D}[s] \cdot (\mathcal{E} f^s)$ over $\mathcal{D}[s]$ in terms

⋆ The authors are partially supported by MTM2004-07203-C02-01 and FEDER.

© Springer-Verlag Berlin Heidelberg 2006

of a presentation of $\mathcal{E}$ over the ring of logarithmic differential operators. This description will be useful in order to compute the Bernstein-Sato polynomials associated with $\mathcal{E}$.

In section 3, the general results of the previous section are explicitly written down in the case of a family of integrable logarithmic connections with respect to a quasi-homogeneous plane curves.

In section 4 we perform some explicit computations with respect to a cusp.

We wish to thank Hélène Esnault who, because of a question about our paper [5], drew our attention to computing intersection D-modules. We also thank Tristan Torrelli for helpful information about the Bernstein-Sato functional equations and for some comments on a previous version of this paper.

This paper is a condensed version of the preprint math.AG/0604287 with the same title.

1 Logarithmic Connections with Respect to a Free Divisor: Theoretical Set-Up

Let X be a n-dimensional complex analytic manifold and $D \subset X$ a hypersurface, and let us denote by $j : U = X - D \hookrightarrow X$ the corresponding open inclusion.

We say that D is a *free divisor* [22] if the $\mathcal{O}_X$-module $\mathrm{Der}(\log D)$ of logarithmic vector fields with respect to D is locally free (of rank n), or equivalently if the $\mathcal{O}_X$-module $\Omega^1_X(\log D)$ of logarithmic 1-forms with respect to D is locally free (of rank n).

Normal crossing divisors, plane curves, free hyperplane arrangements (e.g. the union of reflecting hyperplanes of a complex reflection group), discriminant of stable mappings or bifurcation sets are examples of free divisors.

We say that D is quasi-homogeneous at $p \in D$ if there is a system of local coordinates $\underline{x}$ centered at p such that the germ (D, p) has a reduced weighted homogeneous defining equation (with strictly positive weights) with respect to $\underline{x}$. We say that D is locally quasi-homogeneous if it is so at each point $p \in D$.

Let us denote by $\mathcal{D}_X(\log D)$ the 0-term of the Malgrange-Kashiwara filtration with respect to D on the sheaf $\mathcal{D}_X$ of linear differential operators on X. When D is a free divisor, the first author has proved in [2] that $\mathcal{D}_X(\log D)$ is the universal enveloping algebra of the Lie algebroid $\mathrm{Der}(\log D)$, and then it is coherent and has noetherian stalks of finite global homological dimension. Locally, if $\{\delta_1, \ldots, \delta_n\}$ is a local basis of the logarithmic vector fields on an open set V, any differential operator in $\Gamma(V, \mathcal{D}_X(\log D))$ can be written in a unique way as a finite sum

$$\sum_{\alpha \in \mathbf{N}^n, |\alpha| \leq d} a_\alpha \delta_1^{\alpha_1} \cdots \delta_n^{\alpha_n},$$

where the a_α are holomorphic functions on V.

From now on, let us assume that D is a free divisor.

We say that D is a *Koszul free* divisor [2] at a point $p \in D$ if the symbols of any (some) local basis $\{\delta_1, \ldots, \delta_n\}$ of $\mathrm{Der}(\log D)_p$ form a regular sequence in $Gr\mathcal{D}_{X,p}$. We say that D is a *Koszul free* divisor if it is so at any point $p \in D$.

Plane curves and locally quasi-homogeneous free divisors (e.g. free hyperplane arrangements or discriminant of stable mappings in Mather's "nice dimensions") are example of Koszul free divisors [3].

(1.1) An *integrable logarithmic connection* (ILC for short) with respect to D is a left $\mathcal{D}_X(\log D)$-modules which is locally free of finite rank over $\mathcal{O}_X$.

Let us denote by $\mathcal{O}_X(\star D)$ the sheaf of meromorphic functions with poles along D.

The first examples of integrable logarithmic connections are the invertible $\mathcal{O}_X$-modules $\mathcal{O}_X(mD) \subset \mathcal{O}_X(\star D)$, $m \in \mathbf{Z}$, formed by the meromorphic functions h such that $\operatorname{divi}(h) + mD \geq 0$.

For any ILC $\mathcal{E}$ and any integer m, the locally free $\mathcal{O}_X$-modules $\mathcal{E}(mD) := \mathcal{E} \otimes_{\mathcal{O}_X} \mathcal{O}_X(mD)$ and $\mathcal{E}^* := \operatorname{Hom}_{\mathcal{O}_X}(\mathcal{E}, \mathcal{O}_X)$ are ILC again. (See [5], §2 and [6], §2 for more details about ILC).

If $\mathcal{E}$ is an ILC, then $\mathcal{E}(\star D) = \mathcal{O}_X(\star D) \otimes_{\mathcal{O}_X} \mathcal{E}$ is a meromorphic connection (locally free of finite rank over $\mathcal{O}_X(\star D)$) and then it is a holonomic $\mathcal{D}_X$-module (cf. [15], th. 4.1.3). Actually, $\mathcal{E}(\star D)$ has regular singularities on the smooth part of D (it has logarithmic poles! [9]) and then it is regular everywhere [14], cor. 4.3-14, which means that if $\mathcal{L}$ is the local system of horizontal sections of $\mathcal{E}$ on $U = X - D$, the canonical morphism

$$\Omega_X^{\bullet}(\mathcal{E}(\star D)) \to Rj_*\mathcal{L}$$

is an isomorphism in the derived category.

For any ILC $\mathcal{E}$, one can define its logarithmic de Rham complex $\Omega_X^{\bullet}(\log D)(\mathcal{E})$ in the classical way (cf. [9, def. I.2.15]), which is a subcomplex of $\Omega_X^{\bullet}(\mathcal{E}(\star D))$. It is clear that both complexes coincide on U.

For any ILC $\mathcal{E}$ and any integer m, $\mathcal{E}(mD)$ is a sub-$\mathcal{D}_X(\log D)$-module of the regular holonomic $\mathcal{D}_X$-module $\mathcal{E}(\star D)$, and then we have a canonical morphism in the derived category of left $\mathcal{D}_X$-modules

$$\rho_{\mathcal{E},m} : \mathcal{D}_X \overset{L}{\otimes}_{\mathcal{D}_X(\log D)} \mathcal{E}(mD) \to \mathcal{E}(\star D),$$

given by $\rho_{\mathcal{E},m}(P \otimes e') = Pe'$.

We have the following theorem (see [5, th. 4.1] and [6, th. (2.1.1)]):

Theorem 1. *Let $\mathcal{E}$ be an ILC (with respect to the divisor D) and let $\mathcal{L}$ be the local system of its horizontal sections on $U = X - D$. The following properties are equivalent:*

1) *The canonical morphism $\Omega_X^{\bullet}(\log D)(\mathcal{E}) \to Rj_*\mathcal{L}$ is an isomorphism in the derived category of complexes of sheaves of complex vector spaces.*
3) *The morphism $\rho_{\mathcal{E},1} : \mathcal{D}_X \overset{L}{\otimes}_{\mathcal{D}_X(\log D)} \mathcal{E}(D) \to \mathcal{E}(\star D)$ is an isomorphism in the derived category of left $\mathcal{D}_X$-modules.*

(1.2) Let $\mathcal{E}$ be an ILC (with respect to D) and p a point in D. Let $f \in \mathcal{O} = \mathcal{O}_{X,p}$ be a reduced local equation of D and let us write $\mathcal{D} = \mathcal{D}_{X,p}$, $\mathcal{V}_0 =$

$\mathcal{D}_X(\log D)_p$ and $E = \mathcal{E}_p$. We know from [6, lemma (3.2.1)] that the ideal of polynomials $b(s) \in \mathbf{C}[s]$ such that

$$b(s)Ef^s \subset \mathcal{D}[s] \cdot \left(Ef^{s+1}\right) \left(\subset E[f^{-1}, s]f^s\right)$$

is generated by a non constant polynomial $b_{\mathcal{E},p}(s)$. By the coherence of the involved objects we deduce that $b_{\mathcal{E},q}(s) \mid b_{\mathcal{E},p}(s)$ for $q \in D$ close to p.

If $b_{\mathcal{E},p}(s)$ has some integer root, let us call $\kappa(\mathcal{E}, p)$ the minimum of those roots. If not, let us write $\kappa(\mathcal{E}, p) = +\infty$.

Let us call

$$\kappa(\mathcal{E}) = \inf\{\kappa(\mathcal{E}, p) \mid p \in D\} \in \mathbf{Z} \cup \{\pm\infty\}.$$

From now on let us suppose that D is a locally quasi-homogeneous free divisor.

Theorem 2. *Under the above hypothesis, if $\kappa(\mathcal{E}) > -\infty$, then the morphism*

$$\rho_{\mathcal{E},k} : \mathcal{D}_X \overset{L}{\otimes}_{\mathcal{D}_X(\log D)} \mathcal{E}(kD) \to \mathcal{E}(\star D) \tag{1}$$

is an isomorphism in the derived category of left $\mathcal{D}_X$-modules, for all $k \geq -\kappa(\mathcal{E})$.

Proof. It is a straightforward consequence of [3], [4, th. 5.6] and theorem (3.2.6) of [6] and its proof.

Let us note that the hypothesis $\kappa(\mathcal{E}) > -\infty$ in theorem 2 holds locally on X.

In the situation of theorem 2, if $\mathcal{L}$ is the local system of the horizontal sections of $\mathcal{E}$ on $U = X - D$, and $\mathrm{IC}_X(\mathcal{L})$ is the intersection complex of Deligne-Goresky-MacPherson associated with $\mathcal{L}$, which is described as the intermediate direct image $j_{!*}\mathcal{L}$, i.e. the image of $j_!\mathcal{L} \to Rj_*\mathcal{L}$ in the category of perverse sheaves (cf. [1], def. 1.4.22), we can apply the results of [6] §4, specially theorem (4.1), that we can rewrite as follows:

Theorem 3. *Under the above hypothesis, we have a canonical isomorphism in the category of perverse sheaves on X,*

$$\mathrm{IC}_X(\mathcal{L}) \simeq DR\left(\mathrm{Im}\varrho_{\mathcal{E},k,k'}\right),$$

for $k \geq -\kappa(\mathcal{E})$, $k' \geq -\kappa(\mathcal{E}^*)$ and $1 - k' \leq k$, with

$$\varrho_{\mathcal{E},k,k'} : \mathcal{D}_X \otimes_{\mathcal{D}_X(\log D)} \mathcal{E}((1-k')D) \to \mathcal{D}_X \otimes_{\mathcal{D}_X(\log D)} \mathcal{E}(kD) \tag{2}$$

the $\mathcal{D}_X$-morphism induced by the inclusion $\mathcal{E}((1-k')D) \subset \mathcal{E}(kD)$.

2 Bernstein-Sato Polynomials for Cyclic Integrable Logarithmic connections

In the situation of (1.2), let us assume that E is a cyclic $\mathcal{V}_0$-module generated by an element $e \in E$. The following result is proved in [6, prop. (3.2.3)].

Proposition 1. *Under the above conditions, the polynomial $b_{\mathcal{E},p}(s)$ coincides with the Bernstein-Sato polynomial $b_e(s)$ of e with respect to f, where e is considered to be an element of the holonomic $\mathcal{D}$-module $E[f^{-1}]$ (cf. [11]).*

Let $\Theta_{f,s} \subset \mathcal{D}[s]$ be the set of operators in $\operatorname{ann}_{\mathcal{D}[s]} f^s$ of total order (in s and in the derivatives) ≤ 1. The elements of $\Theta_{f,s}$ are of the form $\delta - \alpha s$ with $\delta \in \operatorname{Der}_{\mathbf{C}}(\mathcal{O})$, $\alpha \in \mathcal{O}$ and $\delta(f) = \alpha f$. In particular $\Theta_{f,s} \subset \mathcal{V}_0[s]$.

The $\mathcal{O}$-linear map $\delta \in \operatorname{Der}(\log D)_p \mapsto \delta - \frac{\delta(f)}{f} s \in \Theta_{f,s}$ is an isomorphism of Lie-Rinehart algebras over $(\mathbf{C}, \mathcal{O})$ and extends to a unique ring isomorphism $\Phi : \mathcal{V}_0[s] \to \mathcal{V}_0[s]$ with $\Phi(s) = s$ and $\Phi(a) = a$ for all $a \in \mathcal{O}$. Let us note that $\Phi^{-1}(\delta) = \delta + \frac{\delta(f)}{f} s$ for each $\delta \in \operatorname{Der}(\log D)_p$.

It is clear that $E[s]f^s$ is a sub-$\mathcal{V}_0[s]$-module of $E[s, f^{-1}]f^s$ and that for any $P \in \mathcal{V}_0[s]$ and any $e' \in E[s]$, the following relation holds

$$(Pe')f^s = \Phi(P)(e'f^s). \tag{3}$$

Proposition 2. *Under the above conditions, if D is a locally quasi-homogeneous free divisor, then* $\operatorname{ann}_{\mathcal{D}[s]}(ef^s) = \mathcal{D}[s] \cdot \Phi(\operatorname{ann}_{\mathcal{V}_0} e)$.

Proof. From (3) we know that $E[s]f^s = \mathcal{V}_0[s] \cdot (ef^s)$, and from [6, cor. (3.1.2)] we know that the morphism

$$\rho_{E,s} : P \otimes (e'f^s) \in \mathcal{D}[s] \otimes_{\mathcal{V}_0[s]} E[s]f^s \mapsto P(e'f^s) \in \mathcal{D}[s] \cdot (E[s]f^s) = \mathcal{D}[s] \cdot (ef^s)$$

is an isomorphism of left $\mathcal{D}[s]$-modules. Therefore

$$\operatorname{ann}_{\mathcal{D}[s]}(ef^s) = \mathcal{D}[s] \cdot \operatorname{ann}_{\mathcal{V}_0[s]}(ef^s).$$

The inclusion $\operatorname{ann}_{\mathcal{V}_0[s]}(ef^s) \supset \Phi(\operatorname{ann}_{\mathcal{V}_0} e)$ comes from (3). For the other inclusion, let $Q \in \operatorname{ann}_{\mathcal{V}_0[s]}(ef^s)$ and let us write $\Phi^{-1}(Q) = \sum_{i=1}^d P_i s^i$ with $P_i \in \mathcal{V}_0$. We have $0 = Q(ef^s) = \left(\Phi^{-1}(Q)e\right) f^s = \left(\sum_{i=1}^d (P_i e)s^i\right) f^s$ and $P_i \in \operatorname{ann}_{\mathcal{V}_0} e$. So, $Q = \Phi\left(\sum_{i=1}^d P_i s^i\right) = \sum_{i=1}^d \Phi(P_i)s^i \in \mathcal{V}_0[s] \cdot \Phi(\operatorname{ann}_{\mathcal{V}_0} e)$.

Remark 1. Theorems 2 and 3 and proposition 2 remain true if we only assume that our divisor D is of commutative linear type, i.e. its jacobian ideal is of linear type (see [6, §3]).

Remark 2. As we shall see in sections 3 and 4, theorem 3, proposition 1 and proposition 2 provide an effective method of computing the intersection $\mathcal{D}_X$-module corresponding to $\operatorname{IC}_X(\mathcal{L})$ in terms of the ILC $\mathcal{E}$, at least if D is a locally quasi-homogeneous free divisor, or more generally, if D is of commutative linear type (see remark 1).

3 Integrable Logarithmic Connections Along Quasi-homogeneous Plane Curves

Let $D \subset X = \mathbf{C}^2$ be a divisor defined by a reduced polynomial equation $h(x_1, x_2)$, which is quasi-homogeneous with respect to the strictly positive integer weights ω_1, ω_2 of the variables x_1, x_2. We denote by $\omega(f)$ the weight of a quasi-homogeneous polynomial $f(x_1, x_2)$. The divisor D is free, a global basis of $\mathrm{Der}(\log D)$ is $\{\delta_1, \delta_2\}$, where

$$\begin{pmatrix} \delta_1 \\ \delta_2 \end{pmatrix} = \begin{pmatrix} \omega_1 x_1 & \omega_2 x_2 \\ -h_{x_2} & h_{x_1} \end{pmatrix} \begin{pmatrix} \frac{\partial}{\partial x_1} \\ \frac{\partial}{\partial x_2} \end{pmatrix}.$$

We have:

-) $\delta_1(h) = \omega(h)h, \quad \delta_2(h) = 0,$

-) the determinant of the coefficient matrix is equal to $\omega(h)h$,

-) $[\delta_1, \delta_2] = c\delta_2$, with $c = \omega(h) - \omega_1 - \omega_2$.

We consider a logarithmic connection $\mathcal{E} = \oplus_{i=1}^n \mathcal{O}_X e_i$ given by actions:

$$\delta_1 \cdot \begin{pmatrix} e_1 \\ \vdots \\ e_n \end{pmatrix} = A_1 \begin{pmatrix} e_1 \\ \vdots \\ e_n \end{pmatrix}, \quad \delta_2 \cdot \begin{pmatrix} e_1 \\ \vdots \\ e_n \end{pmatrix} = A_2 \begin{pmatrix} e_1 \\ \vdots \\ e_n \end{pmatrix}.$$

For $\mathcal{E}$ to be integrable, the following integrability condition

$$\delta_1(A_2) - \delta_2(A_1) + [A_2, A_1] = cA_2 \tag{4}$$

must hold.

(3.1) We shall focus on the case where A_1, A_2 are $n \times n$ matrices satisfying (4) and of the form:

$$A_1 = \begin{pmatrix} -a & 0 & 0 & \cdots & 0 & 0 \\ -\delta_2(a) & -a+c & 0 & \cdots & 0 & 0 \\ -\delta_2^2(a) & -2\delta_2(a) & -a+2c & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -\delta_2^{n-2}(a) & -\binom{n-2}{1}\delta_2^{n-3}(a) & -\binom{n-2}{2}\delta_2^{n-4}(a) & \cdots & -a+(n-2)c & 0 \\ -\delta_2^{n-1}(a) & -\binom{n-1}{1}\delta_2^{n-2}(a) & -\binom{n-1}{2}\delta_2^{n-3}(a) & \cdots & -\binom{n-1}{n-2}\delta_2(a) & -a+(n-1)c \end{pmatrix},$$

$$A_2 = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & 1 \\ -b_0 & -b_1 & -b_2 & \cdots & -b_{n-1} \end{pmatrix}.$$

with $a, b_0, \ldots, b_{n-1}$ polynomials. Let us call $\mathcal{E}_{a,\underline{b}}$ the corresponding ILC.

Lemma 1. *The $\mathcal{D}_X(\log D)$-module $\mathcal{E}_{a,\underline{b}}$ is generated by e_1 (so it is cyclic) and the $\mathcal{D}_X(\log D)$-annihilator of e_1 is the left ideal $J_{a,\underline{b}}$ generated by $\delta_1 + a$ and $\delta_2^n + b_{n-1}\delta_2^{n-1} + \cdots + b_1\delta_2 + b_0$. So, the $\mathcal{D}_X(\log D)$-module $\mathcal{E}_{a,\underline{b}}$ is isomorphic to $\mathcal{D}_X(\log D)/J_{a,\underline{b}}$.*

Proof. The first part is clear since $\delta_2 \cdot e_i = e_{i+1}$ for $i = 1, \ldots, n-1$. For the second part, the inclusion $J_{a,\underline{b}} \subset \operatorname{ann}_{\mathcal{D}_X(\log D)}(e_1)$ is also clear. To prove the opposite inclusion, we use the fact that any germ of logarithmic differential operator P has a unique expression as a sum $P = \sum_{i,j} a_{i,j}\delta_1^i\delta_2^j$, where the $a_{i,j}$ are germs of holomorphic functions ([2], th. 2.1.4) and a division argument.

Remark 3. Theorem 2.1.4 in [2] says that $\mathcal{D}_X(\log D) = \mathcal{O}_X[\delta_1, \delta_2]$ with relations:

$$[\delta_1, f] = \delta_1(f), [\delta_2, f] = \delta_2(f), [\delta_1, \delta_2] = c\delta_2, \quad f \in \mathcal{O}_X.$$

In particular, we can define the *support* and the *exponent* of any germ of logarithmic differential operator P (or of any polynomial logarithmic differential operator in the Weyl algebra) by using the (unique) expression $P = \sum_{ijkl} a_{ijkl}x_1^k x_2^l \delta_1^i \delta_2^j$, and we obtain a division theorem and a notion of *Gröbner basis* for ideals. Under this scope, the integrability condition (4) reads out as the fact that the generators

$$g_1 = \delta_1 + a, \quad g_2 = \delta_2^n + b_{n-1}\delta_2^{n-1} + \cdots + b_0$$

of $J_{a,\underline{b}}$ satisfy Buchberger's criterion, i.e. that $\delta_2^n g_1 - \delta_1 g_2$ has a vanishing remainder with respect to the division by g_1, g_2, and then they form a Gröbner basis of $J_{a,\underline{b}}$.

In this way we can perform effective computations on the ring $\mathbf{W}_2(\log D) = \mathbf{C}[x_1, x_2, \delta_1, \delta_2]$ of polynomial logarithmic differential operators. For instance, we can treat the case of arbitrary ILC (not necessarily cyclic over $\mathcal{D}_X(\log D)$ –or over $\mathbf{W}_2(\log D)$–) in the sense that for arbitrary (polynomial) matrices A_1 and A_2 satisfying (4), we can effectively compute the annihilator over $\mathbf{W}_2(\log D)$ of any of the generators e_i. We plan to develop these ideas elsewhere.

Corollary 1. *The $\mathcal{D}_X$-module $\mathcal{D}_X \otimes_{\mathcal{D}_X(\log D)} \mathcal{E}_{a,\underline{b}}$ is isomorphic to $\mathcal{D}_X/I_{a,\underline{b}}$, where $I_{a,\underline{b}} = \mathcal{D}_X(\delta_1 + a, \delta_2^n + b_{n-1}\delta_2^{n-1} + \cdots + b_0)$.*

For any integer k, we can consider the logarithmic connections $\mathcal{E}_{a,\underline{b}}(kD)$ and $\mathcal{E}_{a,\underline{b}}^*$ (see section (1.1)).

Lemma 2. *With the above notations, the ILC $\mathcal{E}_{a,\underline{b}}(kD)$ and $\mathcal{E}_{a+\omega(h)k,\underline{b}}$ are isomorphic.*

Proof. An $\mathcal{O}_X$-basis of $\mathcal{E}_{a,\underline{b}}(kD)$ is $\{e_i^k = e_i \otimes h^{-k}\}_{i=1}^n$ and the action of $\operatorname{Der}(\log D)$ over this basis is given by

$$\delta_1 \cdot e_i^k = (\delta_1 \cdot e_i) \otimes h^{-k} + e_i \otimes (-\omega(h)kh^{-k}), \quad \delta_2 \cdot e_i^k = (\delta_2 \cdot e_i) \otimes h^{-k}.$$

Then, the isomorphism $\sum_{i=1}^n b_i e_i \mapsto \sum_{i=1}^n b_i e_i^k$ is $\mathcal{D}_X(\log D)$-linear.

The proof of the following proposition is clear.

Proposition 3. *The morphism*

$$\varrho_{\mathcal{E}_{a,\underline{b}},k,k'} : \mathcal{D}_X \otimes_{\mathcal{D}_X(\log D)} \mathcal{E}_{a,\underline{b}}((1-k')D) \to \mathcal{D}_X \otimes_{\mathcal{D}_X(\log D)} \mathcal{E}_{a,\underline{b}}(kD),$$

defined in (2), corresponds, through the isomorphisms in corollary 1 and lemma 2, to the morphism

$$\varrho'_{\mathcal{E}_{a,\underline{b}},k,k'} : \overline{P} \in \mathcal{D}_X / I_{a+\omega(h)(1-k'),\underline{b}} \mapsto \overline{Ph^{k+k'-1}} \in \mathcal{D}_X / I_{a+\omega(h)k}.$$

For the dual connection $\mathcal{E}^*_{a,\underline{b}}$, in order to simplify, let us concentrate on case $n=2$, where the integrability condition (4) reduces to:

$$(\delta_1 - c)(b_1) = 2\delta_2(a), \quad (\delta_1 - 2c)(b_0) = \delta_2^2(a) + b_1\delta_2(a). \tag{5}$$

Lemma 3. *With the above notations, the ILC $\mathcal{E}^*_{a,\underline{b}}$ and $\mathcal{E}_{c-a,\underline{b}^*}$, with $\underline{b} = (b_1, b_0)$ and $\underline{b}^* = (-b_1, b_0 - \delta_2(b_1))$, are isomorphic.*

Proof. The action of $\mathrm{Der}(\log D)$ over the dual basis $\{e_1^*, e_2^*\}$ in $\mathcal{E}^*_{a,\underline{b}}$ is given by:

$$(\delta_i \cdot e_j^*)(e_k) = \delta_i(e_j^*(e_k)) - e_j^*(\delta_i e_k) = -e_j^*(\delta_i e_k),$$

for $i = 1, 2$ and $j, k = 1, 2$. Then

$$\delta_1 \begin{pmatrix} e_1^* \\ e_2^* \end{pmatrix} = -A_1^t \begin{pmatrix} e_1^* \\ e_2^* \end{pmatrix}, \quad \delta_2 \begin{pmatrix} e_1^* \\ e_2^* \end{pmatrix} = -A_2^t \begin{pmatrix} e_1^* \\ e_2^* \end{pmatrix}.$$

Choosing the new basis $\{w_1 = e_2^*, w_2 = -e_1^* + b_1 e_2^*\}$ of $\mathcal{E}^*_{a,\underline{b}}$, we obtain

$$\delta_1 \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \cdots = \begin{pmatrix} a-c & 0 \\ \delta_2(a) & a \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}, \delta_2 \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \cdots = \begin{pmatrix} 0 & 1 \\ \delta_2(b_1) - b_0 & b_1 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$$

and the isomorphism $\sum_{i=1}^2 b_i w_i \mapsto \sum_{i=1}^2 b_i e_i$ is $\mathcal{D}_X(\log D)$-linear.

4 Some Explicit Examples

In this section we consider the case where $D \subset X = \mathbf{C}^2$ is defined by the reduced equation $h = x_1^2 - x_2^3$, and then $\omega(x_1) = 3$, $\omega(x_2) = 2$, $\omega(h) = 6$ and the basis of $\mathrm{Der}(\log D)$ is $\{\delta_1, \delta_2\}$, with

$$\begin{pmatrix} \delta_1 \\ \delta_2 \end{pmatrix} = \begin{pmatrix} 3x_1 & 2x_2 \\ 3x_2^2 & 2x_1 \end{pmatrix} \begin{pmatrix} \frac{\partial}{\partial x_1} \\ \frac{\partial}{\partial x_2} \end{pmatrix},$$

-) $\delta_1(h) = 6h, \quad \delta_2(h) = 0$,
-) the determinant of the coefficient matrix is equal to $6h$,
-) $[\delta_1, \delta_2] = \delta_2$ $(c = 1)$.

(4.1) Since the ILC $\mathcal{E}_{a,\underline{b}}$ and the ideals $I_{a,\underline{b}}$ in corollary 1 are defined globally by differential operators with polynomial coefficients and D has a global polynomial equation, the study of morphism

$$\rho_{\mathcal{E}_{a,\underline{b}},k} : \mathcal{D}_X \overset{L}{\otimes}_{\mathcal{D}_X(\log D)} \mathcal{E}_{a,\underline{b}}(kD) \to \mathcal{E}_{a,\underline{b}}(\star D)$$

can be done globally at the level of the Weyl algebra $\mathbf{W}_2 = \mathbf{C}[x_1, x_2, \frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}]$.

The integrability conditions in (5) (for $n = 2$) become in our case

$$(\delta_1 - 1)(b_1) = 2\delta_2(a), \quad (\delta_1 - 2)(b_0) = \delta_2^2(a) + b_1\delta_2(a). \tag{6}$$

Once a is fixed, it allows us to determine, uniquely, b_1 (the operator $\delta_1 - 1$ is injective), and to also determine b_0 up to a term ex_2, $e \in \mathbf{C}$ (the kernel of the operator $\delta_1 - 2$ is generated by x_2). In order to simplify, let us take

$$a = \lambda + mx_1 + nx_2, \quad b_1 = 2mx_2^2 + 2nx_1,$$

$$b_0 = ex_2 + 3nx_2^2 + 4mx_1x_2 + n^2x_1^2 + 2mnx_1x_2^2 + m^2x_2^4,$$

where λ, m, n, e are complex parameters. For convenience (see the rational factorization of $B(s)$ below), let us consider another complex parameter ν and make $e = \nu - \nu^2$.

Let us define, for $\underline{\mu} = (\lambda, m, n)$, the family of ILC of rank two, $\mathcal{F}_{\nu,\underline{\mu}} := \mathcal{E}_{a,\underline{b}}$ (see (3.1)), with a, b_0, b_1 as above. We have $\mathcal{F}_{\nu,\underline{\mu}} = \mathcal{D}_X(\log D) \cdot e_1$ and $\operatorname{ann}_{\mathcal{D}_X(\log D)} e_1 = \mathcal{D}_X(\log D)(g_1, g_2)$, with $g_1 = \delta_1 + a$ and $g_2 = \delta_2^2 + b_1\delta_2 + b_0$ (see lemma 1). It is clear that $\mathcal{F}_{\nu,\underline{\mu}} = \mathcal{F}_{1-\nu,\underline{\mu}}$.

The conclusion of proposition 2 can be globalized and we obtain

$$\operatorname{ann}_{\mathcal{D}_X[s]}(e_1 h^s) = \mathcal{D}_X[s](\Phi(g_1), \Phi(g_2)) = \mathcal{D}_X[s](\delta_1 + a - 6s, g_2)$$

$$\operatorname{ann}_{\mathbf{W}_2[s]}(e_1 h^s) = \mathbf{W}_2[s](\delta_1 + a - 6s, g_2).$$

Let us consider the left ideal I in the Weyl algebra with parameters

$$I = (h, \delta_1 + a - 6s, \delta_2^2 + b_1\delta_2 + b_0) \subset \mathbf{W}' = \mathbf{C}\left[\lambda, m, n, \nu, x_1, x_2, \frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}\right][s].$$

By a Gröbner basis computation with an elimination order, for example, with the help of [12], we compute the generator $B(s)$ of the ideal $I \cap \mathbf{C}[s]$ and operators $P(s), C(s), D(s) \in \mathbf{W}'$ such that

$$B(s) = P(s)h + C(s)(\delta_1 + a - 6s) + D(s)(\delta_2^2 + b_1\delta_2 + b_0).$$

We find

$$B(s) = \left(s - \frac{\lambda - 5}{6}\right)\left(s - \frac{\lambda - 8}{6}\right)\left(s - \frac{\lambda - \nu - 6}{6}\right)\left(s - \frac{\lambda + \nu - 7}{6}\right).$$

For $\lambda, \nu \in \mathbf{C}$, let us call $B_{\lambda,\nu}(s) \in \mathbf{C}[s]$ the polynomial obtained from $B(s)$ in the obvious way. We obtain then for each $\nu, \lambda, m, n \in \mathbf{C}$ the global Bernstein-Sato functional equation

$$B_{\lambda,\nu}(s) e_1 h^s = P(s)\left(e_1 h^{s+1}\right) \tag{7}$$

in $\mathcal{F}_{\nu,\underline{\mu}}[h^{-1}, s]h^s$. Therefore, $b_{\mathcal{F}_{\nu,\underline{\mu}},p}(s) \mid B_{\lambda,\nu}(s)$ (see prop. 1) for any $p \in D^1$ and

$$\kappa(\mathcal{F}_{\nu,\underline{\mu}}) \geq \tau(\lambda,\nu) := \min\{\text{integer roots of } B_{\lambda,\nu(s)}\} \in \mathbf{Z} \cup \{+\infty\}.$$

We can apply theorem 2 to deduce that morphism

$$\rho_{\mathcal{F}_{\nu,\underline{\mu}},k} : \mathcal{D}_X \otimes_{\mathcal{D}_X(\log D)} \mathcal{F}_{\nu,\underline{\mu}}(kD) \to \mathcal{F}_{\nu,\underline{\mu}}(\star D)$$

is an isomorphism for all $k \geq -\tau(\lambda,\nu)$. On the other hand, from lemma 3 we know that $(\mathcal{F}_{\nu,\lambda,m,n})^* = \mathcal{F}_{\nu,1-\lambda,-m,-n}$ and then morphism

$$\rho_{\mathcal{F}^*_{\nu,\underline{\mu}},k'} : \mathcal{D}_X \otimes_{\mathcal{D}_X(\log D)} \mathcal{F}^*_{\nu,\underline{\mu}}(k'D) \to \mathcal{F}^*_{\nu,\underline{\mu}}(\star D)$$

is an isomorphism for all $k' \geq -\tau(1-\lambda,\nu)$.

The above results can be rephrased in the following way:

1) Morphism $\rho_{\mathcal{F}_{\nu,\underline{\mu}},k}$ is an isomorphism if the four following conditions hold:

$\lambda + 6k \neq -1, -7, -13, -19, \ldots \quad \lambda + 6k \neq 2, -4, -10, -16, \ldots$
$\lambda + 6k - \nu \neq 0, -6, -12, -18, \ldots \quad \lambda + 6k + \nu \neq 1, -5, -11, -17, \ldots$

2) Morphism $\rho_{\mathcal{F}^*_{\nu,\underline{\mu}},k'}$ is an isomorphism if the four following conditions hold:

$\lambda - 6k' \neq 2, 8, 14, 20, \ldots \quad \lambda - 6k' \neq -1, 5, 11, 17, \ldots$
$\lambda + \nu - 6k' \neq 1, 7, 13, 19, \ldots \quad \lambda - \nu - 6k' \neq 1, -5, -11, -17, \ldots$

In particular, if the four following conditions:

(i) $\lambda \not\equiv 2 \pmod 6$ or $\lambda = 2$
(ii) $\lambda \not\equiv 5 \pmod 6$ or $\lambda = -1$
(iii) $\lambda + \nu \not\equiv 1 \pmod 6$ or $\lambda + \nu = 1$
(iv) $\lambda - \nu \not\equiv 0 \pmod 6$ or $\lambda - \nu = 0$

hold, both morphisms $\rho_{\mathcal{F}_{\nu,\underline{\mu}},1}$ $\rho_{\mathcal{F}^*_{\nu,\underline{\mu}},1}$ are isomorphisms.

Let us denote by $\mathcal{L}_{\nu,\underline{\mu}}$ the local system over $X - D$ of the horizontal sections of $\mathcal{F}_{\nu,\underline{\mu}}$. By theorem 3, provided that conditions (i)-(iv) are satisfied, we have

$$\mathrm{IC}_X(\mathcal{L}_{\nu,\underline{\mu}}) \simeq DR(\mathrm{Im}\varrho_{\mathcal{F}_{\nu,\underline{\mu}},1,1}).$$

Proposition 3 and (4.1) reduce the computation of $\mathrm{Im}\varrho_{\mathcal{F}_{\nu,\underline{\mu}},1,1}$ to the computation of the image of the map

$$\theta_{\nu,\underline{\mu}} : \overline{L} \in \mathbf{W}_2/\mathbf{W}_2(g_1,g_2) \mapsto \overline{Lh} \in \mathbf{W}_2/\mathbf{W}_2(g_1+6,g_2),$$

[1] In fact it is possible to show that $b_{\mathcal{F}_{\nu,\underline{\mu}},0}(s) = B_{\lambda,\nu}(s)$.

but $\mathrm{Im}\theta_{\nu,\underline{\mu}} = \mathbf{W}_2/K_{\nu,\underline{\mu}}$ where $K_{\nu,\underline{\mu}} = \{R \in \mathbf{W}_2 \mid Rh \in \mathbf{W}_2(g_1+6, g_2)\}$.

Now, in order to compute generators of $K_{\nu,\underline{\mu}}$, we proceed as follows. Since $[g_1, g_2] = 2g_2$ (for any $\nu, \underline{\mu}$) and the symbols $\sigma(g_1) = \sigma(\delta_1)$, $\sigma(g_2) = \sigma(\delta_2)^2$ form a regular sequence (D is Koszul free!), we deduce that

$$\sigma\left(\mathbf{W}_2(g_1+6, g_2)\right) = \left(\sigma(\delta_1), \sigma(\delta_2)^2\right)$$

and consequently $\sigma\left(K_{\nu,\underline{\mu}}\right) \subset (\sigma(\delta_1), \sigma(\delta_2)^2) : h$. A straightforward (commutative) computation shows that

$$(\sigma(\delta_1), \sigma(\delta_2)^2) : h = (\sigma(\delta_1), \sigma(Q_0))$$

with $Q_0 = 9x_2\frac{\partial^2}{\partial x_1^2} - 4\frac{\partial^2}{\partial x_2^2}$, and

$$\sigma(Q_0)h = x_2\sigma(\delta_1)^2 - \sigma(\delta_2)^2 = x_2\sigma(\delta_1)\sigma(g_1+6) - \sigma(g_2). \qquad (8)$$

Searching to lift the relation (8) to $\mathbf{W}_2$, we find

$$Qh = x_2(\delta_1 + mx_1 + nx_2 + 7 - \lambda)(g_1+6) - g_2 + (\lambda^2 - \lambda + \nu - \nu^2)x_2,$$

with $Q = Q_0 + 6mx_2\frac{\partial}{\partial x_1} - 4n\frac{\partial}{\partial x_2} + m^2x_2 - n^2$. In particular, if condition

$$\lambda^2 - \lambda + \nu - \nu^2 = 0 \quad (\Leftrightarrow \lambda - \nu = 0 \ \text{ or } \ \lambda + \nu = 1) \qquad (9)$$

holds, then $Q \in K_{\nu,\underline{\mu}}$.

Actually, by using the equality $[Q, g_1] = 4Q$ and the fact that $\sigma(Q) = \sigma(Q_0)$ and $\sigma(g_1) = \sigma(\delta_1)$ also form a regular sequence in $Gr\mathbf{W}_2$, condition (9) implies that

$$K_{\nu,\underline{\mu}} = \mathbf{W}_2(g_1, Q), \quad \sigma\left(K_{\nu,\underline{\mu}}\right) = (\sigma(\delta_1), \sigma(Q_0)).$$

On the other hand, since $\sigma(Q_0)$ is not contained in the ideal (x_1, x_2), we finally deduce the following result:

If parameters $\nu, \underline{\mu} = (\lambda, m, n)$ satisfy conditons (i)-(iv) and (9), then the conormal of the origin $T_0^*(X)$ does not appear as an irreducible component of the characteristic variety of $\mathrm{Im}\theta_{\nu,\underline{\mu}} = \mathbf{W}_2/K_{\nu,\underline{\mu}}$, and consequently

$$\mathrm{Ch}(\mathrm{IC}_X(\mathcal{L}_{\nu,\underline{\mu}})) = \mathrm{Ch}\left(\mathbf{W}_2/K_{\nu,\underline{\mu}}\right) = \{\sigma(\delta_1) = \sigma(Q_0) = 0\} = T_X^*(X) \cup T_D^*(X).$$

The existence of such an example has been suggested by [16], example (3.4), but the question on the values of the parameters $\nu, \underline{\mu}$ for which the local system $\mathcal{L}_{\nu,\underline{\mu}}$ is irreducible will be treated elsewhere.

If condition (9) does not hold, it is not clear that there exists a general expression for a system of generators of $K_{\nu,\underline{\mu}}$ as before.

Remark 4. The relationship between the preceding results and examples and the hypergeometric local systems (cf. [17,18]) is interesting and possibly deserves further work.

References

1. A.A. Beilinson, J. Bernstein, and P. Deligne. *Faisceaux pervers*, *Astérisque* 100. S.M.F., Paris, 1983.
2. F. J. Calderón-Moreno. Logarithmic differential operators and logarithmic de Rham complexes relative to a free divisor. *A. Sci. Éc. N. Sup. (4)*, 32(5) (1999), 701–714.
3. F. J. Calderón Moreno and L. Narváez Macarro. Locally quasi-homogeneous free divisors are Koszul free. *Proc. Steklov Inst. Math.*, 238 (2002), 72–77.
4. F. J. Calderón-Moreno and L. Narváez-Macarro. The module $\mathcal{D}f^s$ for locally quasi-homogeneous free divisors. *Compositio Math.*, 134(1) (2002), 59–74.
5. F. J. Calderón Moreno and L. Narváez Macarro. Dualité et comparaison sur les complexes de de Rham logarithmiques par rapport aux diviseurs libres. *Ann. Inst. Fourier (Grenoble)*, 55(1), 2005. (`math.AG/0411045`).
6. F. J. Calderón Moreno and L. Narváez Macarro. On the logarithmic comparison theorem for integrable logarithmic connections. Preprint, 2006. (`math.AG/0603003`).
7. F. J. Castro-Jiménez and J. M. Ucha-Enríquez. Free divisors and duality for $\mathcal{D}$-modules. *Proc. Steklov Inst. Math.*, 238 (2002), 88–96. (`math.AG/0103085`).
8. F. J. Castro-Jiménez and J. M. Ucha-Enríquez. Testing the logarithmic comparison theorem for free divisors. *Experiment. Math.*, 13(4) (2004), 441–449.
9. P. Deligne. *Equations Différentielles à Points Singuliers Réguliers*, *Lect. Notes in Math.* 163 Springer-Verlag, Berlin-Heidelberg, 1970.
10. D. R. Grayson and M. E. Stillman. Macaulay 2, a software system for research in algebraic geometry. Available at http://www.math.uiuc.edu/Macaulay2/.
11. M. Kashiwara. On the holonomic systems of linear differential equations II. *Invent. Math.*, 49 (1978), 121–135.
12. A. Leykin and H. Tsai. D-modules for Macaulay 2. Package included in [10].
13. Ph. Maisonobe and L. Narváez (editors). *Éléments de la théorie des systèmes différentiels géométriques*, *Séminaires et Congrès* 8. Soc. Math. France, Paris, 2004. Cours du CIMPA, École d'été de Séville (1996).
14. Z. Mebkhout. Le théorème de positivité, le théorème de comparaison et le théorème d'existence de Riemann. In [13], pages 165–308, 2004.
15. Z. Mebkhout, L. Narváez. La théorie du polynôme de Bernstein-Sato pour les algèbres de Tate et de Dwork-Monsky-Washnitzer. *A. S. E.N.S.*, 24, 227–256, 1991.
16. L. Narváez-Macarro. Cycles évanescents et faisceaux pervers: cas des courbes planes irréductibles. *Compositio Math.*, 65 (1988), 321–347.
17. O. Neto and P. C. Silva. Holonomic systems with solutions ramified along a cusp. *C. R. Math. Acad. Sci. Paris*, 335(2) (2002), 171–176.
18. O. Neto and P. C. Silva. On regular holonomic systems with solutions ramified along $y^k = x^n$. *Pacific J. Math.*, 207(2) (2002), 463–487.
19. T. Oaku. An algorithm of computing b-functions. *Duke M. J.*, 87(1), 1997, 115–132.
20. T. Oaku and N. Takayama. Algorithms for D-modules—restriction, tensor product, localization, and local cohomology groups. *J. P. Ap. Alg.*, 156(2-3) (2001), 267–308.
21. T. Oaku, N. Takayama, and U. Walther. A localization algorithm for D-modules. *J. Symbolic Comput.*, 29(4-5) (2000), 721–728.
22. K. Saito. Theory of logarithmic differential forms and logarithmic vector fields. *J. Fac. Sci. Univ. Tokyo*, 27 (1980), 265–291.
23. T. Torrelli. On meromorphic functions defined by a differential system of order 1. *Bull. Soc. Math. France*, 132 (2004), 591–612.
24. T. Torrelli. Logarithmic comparison theorem and D-modules: an overview. Preprint, 2005. (`math.AG/0510430`).

Plural, a Non–commutative Extension of Singular: Past, Present and Future

Viktor Levandovskyy

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University
Altenbergerstrasse 69, 4040 Linz, Austria
levandov@risc.uni-linz.ac.at

Abstract. We describe the non–commutative extension of the computer algebra system Singular, called Plural. In the system, we provide rich functionality for symbolic computation within a wide class of non–commutative algebras. We discuss the computational objects of Plural, the implementation of main algorithms, various aspects of software engineering and numerous applications.

Singular:Plural or, shortly, Plural [19] is a subsystem of a computer algebra system Singular [20]. It provides the framework for symbolic computations with one– and two–sided ideals and modules over non–commutative GR–algebras (Def. 2). Most of Gröbner basics (Sect. 2.5) are available in the kernel of the implementation, ranging from the elimination of variables to the free resolutions. Additional functions and libraries provide advanced algorithms and tools for non–commutative algebra. The powerful implementation and rich functionality make Plural a very helpful system for supporting the research in many fields of mathematics and its applications.

1 Past

In 1997, Gert–Martin Greuel and Yuriy Drozd proposed to modify the experimental branch of Singular, called SingularD, which contained implementations of Gröbner bases and syzygies for modules over Weyl and exterior algebras. One needed to extend the class of available algebras, and implement Gröbner bases and of related algorithms for these algebras as efficient as possible.

In the year 2000, the author defended his Master Thesis and presented the first version of Plural. The class of implemented algebras was bigger, than it was originally planned. Indeed, it constituted the class, studied by J. Apel under the name of *G–algebras* [1], and by A. Kandri–Rody and V. Weispfenning under the name *algebras of solvable type* [23]. T. Mora investigated these algebras among other in his works [34,35] without giving them a special name. It is important, that many quantum groups and different flavors of quantizations, applied to various algebras [5,26,32], are G–algebras (Def. 1) or their factor algebras, GR–algebras (Def. 2).

© Springer-Verlag Berlin Heidelberg 2006

As a name, PLURAL originates from a wordplay. In the funny informal discussion on the 1st of April 1999 (*the fool's day*), among other jokes around maths, it appeared suddenly as the contrary to the word "Singular" in the meaning of a grammar category. Therefore, the question "how to call the new–born SINGULAR extension" has got a quick answer.

Until the 2005, PLURAL was separated from SINGULAR de jure, but de facto PLURAL was included in the development structure of SINGULAR, although it was built in a different way, it kept its own separate documentation and so on. During 2001–2005 a standalone SINGULAR:PLURAL was released several times and used by the community. Many new algorithms were developed and implemented. A Gröbner basis algorithm was enhanced and profited from all the novelties in the kernel of SINGULAR like different kinds of geobuckets, fast internal maps etc. The development of the kernel of PLURAL was done by the author together with Hans Schönemann, and we have reported on some aspects of our work in [30].

Finally, in mid 2005 SINGULAR version 3-0-0 was released, with PLURAL as an integral part of it. Almost at the same time the Ph.D. Thesis [26] was defended by the author, where most of the theoretical and algorithmic research, connected to PLURAL, together with applications were described in detail.

2 Present

2.1 *GR*–Algebras and Their Properties

Let $\mathbb{K}$ be a field, and $T = T_n = \mathbb{K}\langle x_1, \dots, x_n\rangle$ a free associative $\mathbb{K}$–algebra, generated by $\{x_1, \dots, x_n\}$ over $\mathbb{K}$. Among the **monomials** $x_{i_1}x_{i_2}\dots x_{i_s}$, $1 \le i_1, i_2, \dots, i_s \le n$, spanning T as vector space over $\mathbb{K}$, we distinguish the **standard monomials** $x_{i_1}^{\alpha_1}x_{i_2}^{\alpha_2}\dots x_{i_m}^{\alpha_m}$, where $1 \le i_1 < i_2 < \dots < i_m \le n$ and $\alpha_k \in \mathbb{N}$. Via the correspondence $x^\alpha := x_1^{\alpha_1}x_2^{\alpha_2}\dots x_n^{\alpha_n} \mapsto (\alpha_1, \alpha_2, \dots, \alpha_n) =: \alpha$ the set of standard monomials is in bijection with the monoid $\mathbb{N}^n$.

Recall, that any finitely generated associative $\mathbb{K}$–algebra is isomorphic to T_n/I, for some n and some proper two–sided ideal $I \subset T_n$. If the set of standard monomials forms a $\mathbb{K}$–basis of an algebra $A = T/I$, we say that A has a Poincaré–Birkhoff–Witt (shortly, PBW) basis in the variables $x_1, \dots, x_n$. As one can immediately see, the commutative polynomial ring $\mathbb{K}[x_1, \dots, x_n]$ does have a PBW basis, while the free associative algebra $\mathbb{K}\langle x_1, \dots, x_n\rangle$ does not. The existence of a PBW basis is an important property of an algebra.

A total ordering $\prec$ on $\mathbb{N}^n$ is called a monomial ordering on the algebra A with the PBW basis $\{x^\alpha \mid \alpha \in \mathbb{N}^n\}$, if $\forall\, \alpha, \beta, \gamma \in \mathbb{N}^n$, $\alpha \prec \beta \;\Rightarrow\; x^\alpha \prec x^\beta \Rightarrow x^{\alpha+\gamma} \prec x^{\beta+\gamma}$. By $\mathrm{lm}(f)$ we denote the leading monomial of $f \in T$.

Definition 1. *Let $\mathbb{K}$ be a field, $T = \mathbb{K}\langle x_1, \dots, x_n\rangle$ and I be a two–sided ideal of T, generated by the set of elements $\{x_jx_i - c_{ij} \cdot x_ix_j - d_{ij},\quad 1 \le i < j \le n\}$, where $c_{ij} \in \mathbb{K} \setminus \{0\}$ and every $d_{ij} \in T$ is a polynomial, involving only standard*[1] *monomials of T. A $\mathbb{K}$–algebra $A = T/I$ is called* ***a G–algebra****, if the following conditions hold:*

[1] We assume this only for simplicity of presentation.

- ***Ordering condition:*** *there exists a monomial well–ordering $\prec$ on $\mathbb{N}^n$, such that $\forall\, 1 \le i < j \le n \;\; \operatorname{lm}(d_{ij}) \prec x_i x_j$.*
- ***Non–degeneracy condition:*** *$\forall\, 1 \le i < j < k \le n$, to the sets $\{c_{ij}\}$ and $\{d_{ij}\}$ we associate a polynomial $NDC_{ijk} = c_{ik}c_{jk} \cdot d_{ij}x_k - x_k d_{ij} + c_{jk} \cdot x_j d_{ik} - c_{ij} \cdot d_{ik}x_j + d_{jk}x_i - c_{ij}c_{ik} \cdot x_i d_{jk}$. A condition is satisfied, if each NDC_{ijk} reduces to zero with respect to the generators of I.*

The PBW Theorem (from e.g. [28]) generalizes the classical Poincaré–Birkhoff–Witt Theorem from the case of universal enveloping algebras of finite dimensional Lie algebras to the case of general G–algebras. Hence, a G–algebra in variables $x_1, \ldots, x_n$ has a canonical PBW basis $\{x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_n^{\alpha_n} \mid \alpha_k \in \mathbb{N}\}$.

Definition 2. *Let B be a G–algebra and $I \subset B$ be a proper nonzero two–sided ideal. Then, a factor algebra B/I is called a GR–**algebra**.*

Remark 1 (Setup for G–algebras). There are several ways to input a G–algebra in Plural. The Singular type `ring` is extended to the non–commutativity.
1) A generic way for setting up a G–algebra follows the definition above. First, one defines a commutative ring $\mathbb{K}[x_1, \ldots, x_n]$ with the monomial ordering. Then, one inputs two $n \times n$ matrices $C = (c_{ij})$ and $D = (d_{ij})$, and types `ncalgebra(C,D)`. The command `ncalgebra` accepts shortcuts for C or D, i.e. if one passes an argument of type `number` or `poly`, it is interpreted by `ncalgebra` as a matrix with entries of the upper triangle equal to the given argument.
2) Many families of algebras are predefined in Plural libraries like ncalg.lib, nctools.lib, and qmatrix.lib. Moreover, we add new algebras regularly.
3) We also provide the possibility to build tensor products of two GR–algebras over the field $\mathbb{K}$ and the construction of the opposite and the enveloping algebra (Sect. 2.3) from the given GR–algebra.

Remark 2 (Setup for GR–algebras). When the G–algebra has been set up, one can define a factor algebra modulo a two–sided ideal, that is a GR–algebra, which will be of the type `qring`. It is required, that a two–sided ideal must be given in its two–sided Gröbner basis, which can be achieved with the command `twostd`. The simplest syntax for defining a GR–algebra reads as `qring Q = twostd(I);`.

Theorem 1. *Let A be a G–algebra in n variables. Then*

1) *A is left and right Noetherian,*
2) *A is an integral domain,*
3) *A is Auslander–regular and Cohen–Macaulay,*
4) *the Gel'fand–Kirillov dimension* $\operatorname{GKdim}(A) = n + \operatorname{GKdim}(\mathbb{K})$,
5) *the global homological dimension* $\operatorname{gl.dim}(A) \le n$,
6) *the Krull dimension Kr.dim(A) $\le n$.*

We refer to [14], [26], [33] for corresponding definitions and proofs. There are examples, where the inequalities 5) and 6) are strict. In particular, 1) and 2) imply that every G–algebra satisfies a left and a right Ore conditions, hence there exist a total Ore localization, producing a left and a right quotient ring. It is known since [1], that one can use Gröbner bases on a G–algebra A for the arithmetic operations with fractions of its left or right quotient ring.

Remark 3. As for computation of dimensions, one can count only on the algorithm for the calculation of Gel'fand–Kirillov dimension [5], which is implemented in GKDIM.LIB (Lobillo and Rabelo, 2004). The generalized Krull dimension is known for its difficulty and, to the best of our knowledge, there is no algorithm for its computation for a general GR–algebra. We have proved in [26], that the global homological dimension $\operatorname{gl.dim}(A) = n$ provided there exist finite dimensional representations of A over $\mathbb{K}$. It is still an open question, whether the opposite direction is true. Another open problem is the exact computation of gl. dim of a given algebra in the case, when $\operatorname{gl.dim}(A) < n$. The phenomenon, demonstrated by n–th Weyl algebras W_n over a field of characteristic 0, is quite interesting. In this case $\operatorname{gl.dim}(W_n) = n$, while W_n is generated by $2n$ variables and is of Gel'fand–Kirillov dimension $2n$. This behavior is extremal in the sense that the global dimension of a G–algebra in $2n$ variables seems to be at least n.

The class of G–algebras unifies many very important and quite different algebras under one roof, among them quasi–commutative polynomial rings like multiparameter quantum affine spaces, universal enveloping algebras of finite dimensional Lie algebras, some iterated Ore extensions, many quantum groups and quantum deformations, many algebras associated to the classical operators.

One of the reasons for such unification lies in the common structural properties of these algebras. And the second reason is the Gröbner bases theory.

2.2 Gröbner Bases in *GR*–Algebras

We stress the similarities between G–algebras and commutative polynomial rings and use the similarities, when possible. We follow the approach to Gröbner bases, presented in [18]. Let A be a G–algebra in n variables. We say that a **monomial** of a free module A^r (involving component i) is an element of the form $x^\alpha e_i$, where $\alpha \in \mathbb{N}^n$ and e_i is the canonical i–th basis vector. We say, that $m_1 = x^\alpha e_j$ **divides** $m_2 = x^\beta e_k$ and denote it by $m_1 | m_2$, if $j = k$ and $\alpha_i \leq \beta_i \; \forall i = 1 \dots n$. Actually it is rather a pseudo–division on A, since if $m_1 | m_2$, then there exist $c \in \mathbb{K} \setminus \{0\}$, a monomial $p \in A$ and $q \in A^r$ such that $\operatorname{lm}(q) \prec m_1$ and $m_2 = c \cdot p \cdot m_1 + q$, where $q \neq 0$ in general.

From the properties of G–algebras it follows, that any $f \in A^r \setminus \{0\}$ can be written uniquely as $f = c_\alpha x^\alpha e_i + g$, with $c_\alpha \in \mathbb{K}^*$, and $x^\beta e_j \prec x^\alpha e_i$ for any nonzero term $c_\beta x^\beta e_j$ of g. Then we define in the usual fashion $\operatorname{lm}(f) = x^\alpha e_i$, the leading monomial of f, and $\operatorname{lc}(f) = c_\alpha$, the leading coefficient of f. Note, that $\forall \; \alpha, \beta \in \mathbb{N}^n$, $\operatorname{lm}(x^\alpha x^\beta) = \operatorname{lm}(x^{\alpha+\beta}) = \operatorname{lm}(x^\beta x^\alpha)$.

Definition 3. *Let $\prec$ be a monomial ordering on the free module A^r, $I \subset A^r$ a left submodule, and $G \subset I$ a finite subset. G is called a* ***left Gröbner basis*** *of I if and only if for any $f \in I \setminus \{0\}$ there exists $g \in G$, satisfying* $\operatorname{lm}(g) \mid \operatorname{lm}(f)$.

In order to come up with the more constructive definition, one has to use the notion of a monoideal of leading exponents [5] or a span of leading monomials [26] instead of the leading ideal. The latter works well in the commutative and even in the free associative algebras, but fails in general G–algebras for the reasons, which we discussed in detail in [26].

The normal form, the s–polynomial and the Buchberger's algorithm can be generalized for the left or right ideals in almost the same form as they appear in the literature for the commutative case. However, the proofs of main theorems in the Gröbner bases theory are different in spite of similarity. One has to develop a specific intuition, working with G–algebras, even though they are in many senses close to commutative algebras. As the simplest indication of the intrinsic difference we can take the Product Criterion: if the leading monomials of two polynomials f and g do not divide each other, we have $\text{spoly}(f, g) \to_{\{f,g\}} 0$. Hence, this is the easiest situation in the set of pairs, built in the Buchberger's algorithm: discard the pair (f, g) if the condition holds.

In G–algebras with some extra assumptions, we can show [30], that $\text{spoly}(f, g) \to_{\{f,g\}} g \cdot f - f \cdot g =: [g, f]$. Of course, it allows to discard the pair (f, g) from the pair set if f commutes with g. However, this happens rather rarely in general. Otherwise, the number of multiplications and reductions shows that we are perhaps in the worst situation, which might occur in the set of pairs.

On the contrary, the Chain Criterion and its variations generalize to G–algebras in its full generality [5,24,26,32]. The Chain Criterion is actually the most important criterion, used in Plural.

2.3 Left, Right and Two–Sided Structures

The three kinds of ideals and modules (left, right and two–sided) might make the life of a developer quite complicated. The two–sided ideals and, more generally, bimodules are very special structures. The notion of a two–sided Gröbner basis is different from the one of a one–sided Gröbner basis [1,23,30]. The two–sided Gröbner basis is computed with a special algorithm and is in general harder to compute, then the one–sided. A recent algorithm [12] shows superior performance, compared to the variations of the classical approach and will be used in the future. This algorithm utilizes the opposite algebras.

Let A be an associative algebra over $\mathbb{K}$. The **opposite algebra** A^{opp} is defined by taking the same vector space as of A, and by introducing a new "opposite" multiplication $*$ on it, defined by $f * g := g \cdot f$. Then, A^{opp} is an associative $\mathbb{K}$–algebra, and $(A^{\text{opp}})^{\text{opp}} = A$ holds. Moreover, $A \otimes_{\mathbb{K}} A^{\text{opp}}$ is called the **enveloping algebra** of A.

Lemma 1. *Let $B = A/I$ be a GR–algebra. Then B^{opp} is a GR–algebra, and $B^{\text{opp}} = A^{\text{opp}}/I^{\text{opp}}$.*

For right–sided computations with a module like a Gröbner basis, a syzygy module etc., it suffices to implement a left–sided functionality together with procedures for the effective treatment of opposite algebras and transfer of objects between an algebra and its opposite. In Plural, we provide the commands `opposite` and `envelope` for constructing the algebras and `oppose` for the objects transfer. There are several methods for representing the opposite algebra of a given algebra constructively, see [26] for their description.

2.4 Gröbner Trinity and Gröbner Engine

We can compute Gröbner basis of an ideal, Gröbner basis of its first syzygy module, and the transformation matrix between the original set of generators and the Gröbner basis (sometimes called a *lifting matrix*) basically with the same algorithm. We call these three powerful algorithms a **Gröbner trinity**. The same applies for one–sided Gröbner trinity for ideals over *GR*–algebras and is inherited by PLURAL from SINGULAR. The Gröbner trinity is extremely important for further applications of Gröbner bases. For example, a free resolution can be computed as the sequence of syzygies, while a lifting matrix allows to control the critical constellations of parameters, or, in other words, to observe the genericity of Gröbner basis computation [31] and so on.

The algorithm, which is able to compute all of the Gröbner trinity, is essentially the general version of Gröbner basis algorithm. It must be able to compute with free modules, hence it must accept monomial module orderings as input. Moreover, it is important to have the switch for dividing the set of module components into two disjoint groups. Having such a switch, one can compute Gröbner basis only of those vectors, which lie inside of one group and do not compute it for the other group, since the latter will be ignored at the end. Among other cases, this idea is used for computing both the syzygy module and the lifting matrix more easily. The same algorithm must be able to perform computations in a factor algebra, to use extra weights for the ordering or for the generators of a module, to interpret and to use on demand the supplemented information on Hilbert polynomial et cetera.

We call an implementation of the algorithm, which computes a (left) Gröbner basis and which complies with the requirements above, a **Gröbner engine**. The examples of Gröbner engines in SINGULAR are: Gröbner bases (non–negatively graded orderings), standard bases (local and mixed orderings), and PLURAL (left Gröbner bases for non–negatively graded orderings over *G*–algebras). All of these are called with the same command, namely `std`. Yet more methods for computing Gröbner bases are on their way to become someday Gröbner engines.

If the internal implementation of a variant of Gröbner basis algorithm is done in the form of Gröbner engine, one gets all the Gröbner basics (Sect. 2.5) available in a much shorter time, compared with the adjustment of every single application to the new Gröbner basis routine. Moreover, if the internal structure of the implementation of e.g. Gröbner basics is tuned for the use of generic Gröbner engine, one can use different engines for different applications.

The importance of having not only a fast Gröbner basis algorithm, but also fast Gröbner basics (for working with practice–relevant applications) is clear. The concept of Gröbner engine has been used implicitly in SINGULAR. Hans Schönemann and the author are working of the formalization and further development of this concept, providing an interface between Gröbner bases, Gröbner trinity and Gröbner basics. Our experience can be illustrated with two algorithms, available in SINGULAR, namely `janet` and `slimgb`.

`janet`. The possibility to compute Gröbner basis via *involutive basis* was proposed independently by Apel and Gerdt et. al. [13]. The corresponding algorithm has been implemented and enhanced by the group of V. P. Gerdt (`http://invo.jinr.ru`) for ideals of commutative rings, and demonstrated quite a good performance. With the help of the principal developer of the project JB ("Janet involutive bases"), Denis Yanovich, in 2003 we have incorporated their routines, written in C, into SINGULAR. We have learned a lot during that process; the amount of re-engineering we needed to do, together with several other factors, led us to the idea of Gröbner engine.

The SINGULAR command `janet` computes a Gröbner basis of an ideal through the computation of Janet basis and interreduction of the output. The same command, run in the G–algebra, returns a left Gröbner basis of a two–sided ideal. The cooperation with the group of Gerdt continues, and perhaps some day `janet` routines will evolve to the Gröbner engine.

`slimgb`. Slim Gröbner basis is the algorithm of M. Brickenstein [3,4]. It uses many interesting ideas and techniques, which have been proved to provide an impressive performance, especially over transcendental field extensions and also for elimination orderings. One of particular aims was to minimize, if possible, the intermediate coefficient swell. The methods, used in `slimgb`, were general enough to be applied for the non–commutative case. `slimgb` can compute a left Gröbner basis of a left module. Its performance has been successfully tested on many problems; using `slimgb` we obtained solutions for several long–standing computational challenges. Due to very good timings on examples, where elimination orderings were used, `slimgb` is the primary engine for the DMOD.LIB (Sect. 3.5). The development of `slimgb` goes further intensively and, as it seems, will lead to the Gröbner engine in the nearest future.

2.5 Gröbner Basics

Bernd Sturmfels called "Gröbner basics" the most important, yet basic applications of Gröbner bases. We adopt this notion to the non–commutative GR–algebras and remove from this list "too commutative" applications (such as Zariski closure of the image of a map, solving polynomial equations and radical membership). All the algorithms below have been generalized to the context of GR–algebras and implemented in PLURAL.

- Ideal (resp. module) membership problem
- Intersection with subrings (elimination of variables)
- Intersection of ideals (resp. submodules)
- Quotient and saturation of two–sided ideals
- Kernel of a module homomorphism
- Kernel of a ring homomorphism
- Algebraic relations between pairwise commuting polynomials

Definition 4. *Let A be a $\mathbb{K}$–algebra and $F \subseteq A$ a set. The subalgebra $C_A(F) = \{a \in A \mid [f, a] = 0 \; \forall f \in F\}$ is called the* ***centralizer of** F **in** A. Moreover, $Z(A) = C_A(A) = \{z \in A \mid za = az \; \forall a \in A\}$ is called the* ***center** of A.*

In addition to the classical Gröbner basics, there are typically non–commutative Gröbner basics (all of them are implemented in PLURAL):

- Two–sided Gröbner basis of a bimodule
- Gel'fand–Kirillov dimension of a module
- Annihilator of finite dimensional module
- Central quotient resp. saturation of ideals (if the center is non–trivial)
- Preimage of a left ideal under the morphism of algebras
- Graded Betti numbers (for graded modules over graded algebras)
- Left and right kernel of the presentation of a module
- Central Character Decomposition of the Module

It is interesting, whether it is possible to give an algorithm, which computes N–dimensional irreducible representations of a GR–algebra for a positive N. We have proposed an algorithm, which computes all the one–dimensional representations [27].

For a modern computer algebra system, specializing on the non–commutative algebras, it is quite important to have also non–Gröbner functionality, like the operations with opposite and enveloping algebras (described above), computations with centralizers and even more. Many applications (of e.g. representation theory) require an explicit knowledge of the generators of the center of a GR–algebra as well as the generators of centralizers of finite sets. These algorithms have been implemented in the library CENTER.LIB by O. Motsak. The implementation demonstrated quite a good performance.

While studying algebraic dependence of pairwise commuting polynomials, the method of Perron polynomial was widely used. It has been implemented in the library PERRON.LIB. With this library we have been able to compute several hard examples, which contributed to the progress in studying algebraic dependence in the situation, described in the Sect. 3.2.

3 Work in Progress and Future Development

3.1 Preimage of a Left Ideal

NCPREIMAGE.LIB is dedicated to the computation of the preimage of a left ideal under a morphism of GR–algebras, as it is described in [29]. The implementation of the main algorithm of the article requires, among other, the procedure for the computation of a tuple of strictly positive weights $(w_1, \dots, w_m)$, such that the elimination ordering with the extra weight vector $(w_1, \dots, w_m, 0, \dots, 0)$ satisfies the ordering condition of the Def. 1. If one works with a positively weighted degree ordering, a similar computation of weights can be achieved with the help of the method, described in e.g. [5]. It is implemented as the procedure `Gweights` in the library NCTOOLS.LIB.

3.2 Algebraic Dependence of Pairwise Commuting Polynomials

Consider the universal enveloping algebra A of a finite dimensional simple Lie algebra over a field $\mathbb{K}$. If $\operatorname{char}\mathbb{K} > 0$, it is known from the dimension argument assures, that the generators of the center are algebraically dependent. There are several open questions on the ideal of dependence polynomials which we investigate by using computer algebraic methods. We were able to compute the dependence polynomials explicitly for many prime p over the algebras $U(\mathfrak{sl}_2)$ (see [26]) and $U(\mathfrak{so}_3)$. Up to now, the case of $U(\mathfrak{sl}_3)$ remains unsolved and constitutes an important challenge.

There are more situations, when these methods can be applied. For instance, the algebraic dependence of the generators of the center appears also in quantum algebras, when one considers a quantum parameter q (usually assumed to be transcendental over $\mathbb{K}$) to be some primitive root of unity.

3.3 Homological Algebra in *GR*–Algebras

For two left A–modules M, N, $\operatorname{Ext}^i_A(M,N)$ for $i \geq 0$ carries no A–module structure in general. However, it turns out [5], that in the case, when either M or N is a *centralizing bimodule*, $\operatorname{Ext}^i_A(M,N)$ is an A–module and its presentation can be computed algorithmically. In many applications, one of the modules M, N is often appears to be a centralizing bimodule.

Together with G. Pfister we are working on the implementation of the methods above in the library NCHOMOLOG.LIB. It is planned to have procedures for the computation of Ext and Tor modules in the setup as above, accompanied by other useful tools for homological algebra. We will use these also for the algorithmic computation of Hochschild cohomology of bimodules. We need to compute left and right Gröbner bases, and two–sided bases for bimodules; the need for them motivated, among other, the deeper study and the enhanced implementation of opposite and enveloping algebras.

With the help of the library, we are going to check the long–standing conjecture, starting with algebras of rank 2 and 3:

> *for any simple weight module M over a complex finite–dimensional simple Lie algebra $\mathfrak{g}$, $\dim_{\mathbb{C}} H^i(\mathfrak{g}, M) < \infty$ holds for all i.*

All the computations, related to this conjecture can be done in the universal enveloping algebra $U(\mathfrak{g})$, which is a G–algebra. Among other, the library will be applied to the problems, arising in the systems and control theory.

3.4 Systems and Control Theory

The algorithmic methods of algebraic analysis can be applied to systems of equations involving linear operators like the (partial) differentiation, shift, difference and so on [8,9]. The algorithms for the case, when a system of equations involves only constant coefficients (hence, the system algebra is commutative), have been implemented in the library CONTROL.LIB (Becker, L., and Yena, 2004).

When treating systems with variable polynomial coefficients, the system algebra becomes a *GR*–algebra. Together with E. Zerz we are working on the library NCONTROL.LIB. This library will provide the procedures for the algebraic analysis of systems over not only *G*–algebras (like it is done in the package ORE-MODULES, [9]), but also in *GR*–algebras. The latter requires more efforts and a thorough inspection of the theory and its implementation.

In order to treat systems with rational coefficients, we have to provide Gröbner bases, Gröbner basics, and algorithmic homological algebra for modules over Ore–localized *G*–algebras (see Sect. 3.7).

3.5 *D*–Modules

The library DMOD.LIB (V. L. and J. Morales, 2006) contains procedures for computations with *D*–modules. Let $\operatorname{char}\mathbb{K}=0$. Given a polynomial $F\in\mathbb{K}[x_1,\dots,x_n]$, one is interested in computing the *D*–module structure of the localization $\mathbb{K}[x_1,\dots,x_n,F^s]$ for negative integer s. That is, one looks for the left ideal I in the Weyl algebra $D := A_n$ in $2n$ variables $\{x_1,\dots,x_n,\partial_1,\dots,\partial_n\}$, such that $\mathbb{K}[x_1,\dots,x_n,F^s] \cong A/I$ as *D*–modules. The algorithm for the computation of such I is often called $\operatorname{Ann} F^s$.

We have implemented two variants of this algorithm, namely the algorithm of Oaku and Takayama [6,36] in the procedure `annfsOT`, and the algorithm of Briançon and Maisonobe (e.g. [6]) in the procedure `annfsBM`. One can use both `std` and `slimgb` as underlying Gröbner engine for these complicated algorithms. With the current implementation of DMOD.LIB and `slimgb`, we were recently able to compute several hard examples, e.g. proposed by Castro and Ucha in [6]. In particular, the cases of F being a cusp $x^p - y^q$ (for coprime $p, q \in \mathbb{N}$), a *Reiffen curve* $x^p + y^q + xy^{q-1}$, $q \geq p+1 \geq 5$, or a hyperplane arrangement are studied. We plan to extend the functionality of the library in the direction, described in [36] and [38]. We are going to use the families of examples above as benchmarks and compare the performance of computer algebra systems such as KAN/SM1, MACAULAY2 and SINGULAR:PLURAL.

3.6 Applications to Algebraic Geometry

W. Decker, C. Lossen and G. Pfister created the library SHEAFCOH.LIB, devoted to the computation of the cohomology of coherent sheaves. The procedure `sheafCohBGG` utilizes the Bernstein–Gel'fand–Gel'fand (BGG) correspondence and the Tate resolution [11]. This algorithm, which uses computation of free resolutions over non–commutative exterior algebra (which is a *GR*–algebra), is sometimes much faster, than the commutative one, implemented in the procedure `sheafCoh`, which is based on local duality, following the ideas of Eisenbud.

D. Eisenbud and F.-O. Schreyer presented an algorithm for the computation on higher direct image complex of a coherent sheaf under a projective morphism. The implementation of this algorithm in SINGULAR will appear soon. Like in the SHEAFCOH.LIB, the BGG correspondence and hence, the computations over exterior algebras are used.

3.7 Directions of Future Work

Context–Based Multiplication. In [30] we have described our approaches to the multiplication of polynomials in general G–algebra. The next enhancement in this field is the implementation of formula–based multiplication for the simplest *contexts.* Namely, for an affine G–algebra with the relation $yx = q{\cdot}xy+ax+by+r$, $q, a, b, r \in \mathbb{K}$, $q \neq 0$, it seems possible to derive a symbolic formula in a closed form for the multiplication $y^s \cdot x^t = \sum c_{ij}x^i y^j$. To the best of our knowledge, no general closed–form formula is known yet. Using a formula instead of the updated tables will clearly require less memory, but eventually will consume more time. Subalgebras of affine type as above occur very often in big G–algebras, and the impact of the formula–based multiplication in such subalgebras on the overall performance of Gröbner basis algorithms is very interesting to investigate.

Combined Computations. Singular is one of the few systems, being able to perform *combined computations*, that is both commutative and non–commutative computations in one system. It is important to develop this ability further by implementing *context–based* operations, that is computations, which will derive the subalgebra, where the concrete input resides (e.g. a commutative subalgebra of a G–algebra), and provide the set of most optimized and relevant routines for the concrete computation. A similar method is implemented in Singular as so–called `p-Procs` for polynomial operations over different ground fields.

Ore Localizations. We are working on extending Plural to a bigger class of non–commutative algebras, connected with G–algebras by means of localization. Since from every G–algebra we can built left and right quotient rings, one can extend the machinery we have developed to partial localization of G–algebras. Let $B \subset A$ be two G–algebras, then we can perform the localization of A with respect to e.g. $B \setminus \{0\}$, by means of Ore. If B happens to be commutative, we can apply different localization, e.g. the localization with respect to a maximal ideal. Note, that variables, not belonging to B, remain polynomial. Such algebras are needed in many algebraic constructions and used in various applications.

For example, let R be a ring, containing $\mathbb{K}[x_1, \ldots, x_n]$ as a subring. Then the Weyl algebra with coefficients in R is defined to be $R\langle \partial_1, \ldots, \partial_n \mid [\partial_i, x_i] = 1, [\partial_j, x_k] = 0\rangle$. Very important examples are *rational Weyl algebras*, where $R = \mathbb{K}(x_1, \ldots, x_n)$ or *local polynomial Weyl algebras*, with $R = \mathbb{K}[x_1, \ldots, x_n]_{\langle x_1,\ldots,x_n\rangle}$. The standard basis algorithm for the latter has been recently discussed in [15].

PBW rings [5,24] constitute a general framework, describing such algebras and Gröbner bases for modules over them. Under some assumptions, which reflect the common setup for many applications, such an algebra is called an Ore algebra [8], which has nice properties and is much easier to implement, than a general PBW ring. However, Ore algebras do not cover various important cases of algebras. Therefore, we concentrate ourself on investigating the algorithmic aspects of computations in partial Ore localizations of G–algebras.

The computations in PBW rings are more complicated, than in G–algebras. Even basic arithmetics with one–sided fractions requires the computation of syzygies and hence Gröbner bases [1]. Therefore, the implementation of Gröbner bases in such algebras must be done quite carefully. On the other hand, the powerful implementation opens new perspectives for applications of symbolic computation in this segment of non–commutative algebra.

Non–commutative Computer Algebra Systems

We have reviewed in detail the modern Computer Algebra Systems with the non–commutative abilities in [26]. The following systems are designed for the computations in free associative algebras and path algebras:

- BERGMAN by J. Backelin et.al. [22] is a powerful and flexible tool to calculate Gröbner bases, Hilbert and Poincaré–Betti series, Anick resolution, and Betti numbers in non–commutative algebras and in modules over them,
- NCGB by J. W. Helton et.al. [21] is a MATHEMATICA package, being a part of the NCALGEBRA suite,
- OPAL by B. Keller et.al. [17] is the specialized standalone system for Gröbner bases in free and path algebras,
- GBNP (GROBNER) by A. Cohen and D. Gijsbers [10] is a package for GAP 4 with the implementation of non–commutative Gröbner bases for free and path algebras, following the algorithmic approach of Mora [34,35].

The systems below are mostly restricted to some classes of non–commutative associative algebras, but the computations with them are usually more efficient.

- FELIX by J. Apel and U. Klaus [2] provides generalizations of Buchberger's algorithm to free $\mathbb{K}$–algebras, polynomial rings and G–algebras. Also, the syzygy computations and basic ideal operations are implemented.
- MAS by H. Kredel and M. Pesch [25] contains a large library of Gröbner basis algorithms for computing in non–commutative polynomial rings,
- GROEBNER by F. Chyzak [7] is a MAPLE package, providing Gröbner basis algorithms (including elimination) for Ore algebras,
- a MAPLE package by R. Pearce [37] contains an implementation of Faugère's F4 algorithm for Ore algebras,
- KAN/SM1 by N. Takayama [39], distributed as a part of the system OPENXM, provides Gröbner basis computations in polynomial rings, rings of differential operators, rings of difference and q-difference operators.
- MACAULAY2 by D. Grayson and M. Stillman [16] includes Gröbner basis algorithms for exterior and Weyl algebras and a package for D–module theory.

Acknowledgments

I am grateful to Gert–Martin Greuel, Hans Schönemann and Gerhard Pfister for long and fruitful cooperation, and for their role in the development of PLURAL. I wish to thank Michael Brickenstein, Denis Yanovich, Javier Lobillo, and all other colleagues for their cooperation and contributions to SINGULAR:PLURAL.

References

1. Apel, J. Gröbnerbasen in nichtkommutativen Algebren und ihre Anwendung. *Dissertation, Universität Leipzig*, 1988.
2. Apel, J. and Klaus, U. FELIX, a Special Computer Algebra System for the Computation in Commutative and Non-commutative Rings and Modules, 1998. Available from `http://felix.hgb-leipzig.de/`.
3. Brickenstein, M. Neue Varianten zur Berechnung von Gröbnerbasen. *Diplomarbeit, Universität Kaiserslautern*, 2004.
4. Brickenstein, M. Slimgb: Gröbner Bases with Slim Polynomials. In *Reports On Computer Algebra No. 35*. Centre for Computer Algebra, University of Kaiserslautern, 2005. Available from `http://www.mathematik.uni-kl.de/~zca/`.
5. Bueso, J., Gómez–Torrecillas, J. and Verschoren, A. *Algorithmic methods in non-commutative algebra. Applications to quantum groups.* Kluwer Academic Publishers, 2003.
6. Castro-Jiménez, F.J. and Ucha, J.M. On the computation of Bernstein–Sato ideals. *J. of Symbolic Computation*, 37:629–639, 2004.
7. Chyzak, F. The GROEBNER Package for MAPLE, 2003. Available from `http://algo.inria.fr/libraries/`.
8. Chyzak, F. and Salvy, B. Non–commutative Elimination in Ore Algebras Proves Multivariate Identities. *J. of Symbolic Computation*, 26(2):187–227, 1998.
9. Chyzak, F., Quadrat, A. and Robertz, D. Linear control systems over Ore algebras. Effective algorithms for the computation of parametrizations. In *Proc. of Workshop on Time-Delay Systems (TDS03)*. INRIA, 2003.
10. Cohen, A.M. and Gijsbers D.A.H. GBNP, a Non–commutative Gröbner Bases Package for GAP 4, 2003. Available from `http://www.win.tue.nl/~amc/pub/grobner/`.
11. Eisenbud, D., Fløystad, G. and Schreyer, F.-O. Sheaf algorithms using the exterior algebra. *Trans. Am. Math. Soc.*, 355(11):4397–4426, 2003.
12. García Román, M. and García Román, S. Gröbner bases and syzygies on bimodules over PBW algebras. *J. of Symbolic Computation*, 40(3):1039–1052, 2005.
13. Gerdt, V.P. Involutive Algorithms for Computing Groebner Bases. In Pfister G., Cojocaru S. and Ufnarovski, V., editors, *Computational Commutative and Non-Commutative Algebraic Geometry.* IOS Press, 2005.
14. Gómez–Torrecillas, J. and Lobillo, F.J. Auslander-regular and Cohen-Macaulay quantum groups. *J. Algebr. Represent. Theory*, 7(1):35–42, 2004.
15. Granger, M. and Oaku, T. and Takayama, N. Tangent cone algorithm for homogenized differential operators. *J. of Symbolic Computation*, 39(3–4):417–431, 2005.
16. Grayson, D. and Stillman, M. MACAULAY 2, a Software System for Research in Algebraic Geometry, 2005. Available from `http://www.math.uiuc.edu/Macaulay2/`.
17. Green, E., Heath, L., and Keller, B. Opal: A System for Computing Noncommutative Gröbner Bases. In *RTA '97: Proceedings of the 8th International Conference on Rewriting Techniques and Applications*, pages 331–334. Springer, 1997.
18. Greuel, G.-M. and Pfister, G. with contributions by Bachmann, O., Lossen, C. and Schönemann, H. *A SINGULAR Introduction to Commutative Algebra.* Springer, 2002.
19. Greuel, G.-M., Levandovskyy, V., and Schönemann H. PLURAL. A SINGULAR 3.0 Subsystem for Computations with Non–commutative Polynomial Algebras. Centre for Computer Algebra, University of Kaiserslautern, 2006.

20. Greuel, G.-M., Pfister G., and Schönemann H. SINGULAR 3.0. A Computer Algebra System for Polynomial Computations. Centre for Computer Algebra, University of Kaiserslautern, 2006. Available from `http://www.singular.uni-kl.de`.
21. Helton, J.W. and Stankus, M. NCGB 3.1, a Noncommutative Gröbner Basis Package for MATHEMATICA, 2001. Available from `http://www.math.ucsd.edu/~ncalg/`.
22. J. Backelin et. al. The Gröbner basis calculator BERGMAN, 2006. Available from `http://servus.math.su.se/bergman/`.
23. Kandri-Rody, A. and Weispfenning, V. Non–commutative Gröbner bases in algebras of solvable type. *J. of Symbolic Computation*, 9(1):1–26, 1990.
24. Kredel, H. *Solvable polynomial rings.* Shaker, 1993.
25. Kredel, H. and Pesch, M. MAS, Modula-2 Algebra System, 1998. Available from `http://krum.rz.uni-mannheim.de/mas.html`.
26. Levandovskyy, V. Non–commutative computer algebra for polynomial algebras: Gröbner bases, applications and implementation. *Doctoral Thesis, Universität Kaiserslautern*, 2005. Available from `http://kluedo.ub.uni-kl.de/volltexte/2005/1883/`.
27. Levandovskyy, V. On preimages of ideals in certain non–commutative algebras. In Pfister G., Cojocaru S. and Ufnarovski, V., editors, *Computational Commutative and Non-Commutative Algebraic Geometry.* IOS Press, 2005.
28. Levandovskyy, V. PBW Bases, Non–Degeneracy Conditions and Applications. In Buchweitz, R.-O. and Lenzing, H., editors, *Representation of algebras and related topics. Proceedings of the ICRA X conference*, volume 45, pages 229–246. AMS. Fields Institute Communications, 2005.
29. Levandovskyy, V. Intersection of ideals with non–commutative subalgebras. In *Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC'06).* ACM Press, 2006, to appear.
30. Levandovskyy, V. and Schönemann, H. Plural — a computer algebra system for noncommutative polynomial algebras. In *Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC'03).* ACM Press, 2003.
31. Levandovskyy, V. and Zerz, E. Algebraic systems theory and computer algebraic methods for some classes of linear control systems. In *Proc. of the International Symposium on Mathematical Theory of Networks and Systems (MTNS'06)*, 2006.
32. Li, H. *Noncommutative Gröbner bases and filtered-graded transfer.* Springer, 2002.
33. McConnell, J.C. and Robson, J.C. *Noncommutative Noetherian rings.* AMS, 2001.
34. Mora, T. Groebner bases in non-commutative algebras. In *Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC'88)*, pages 150–161. LNCS 358, 1989.
35. Mora, T. An introduction to commutative and non-commutative Groebner bases. *Theor. Comp. Sci.*, 134:131–173, 1994.
36. Oaku, T. and Takayama, N. An algorithm for de Rham cohomology groups of the complement of an affine variety via D-module computation. *Journal of Pure and Applied Algebra*, 139(1–3):201–233, 1999.
37. Pearce, R. The F4 Algorithm for Gröbner Bases, Package for MAPLE, 2005. Available from `http://www.cecm.sfu.ca/~rpearcea/`.
38. Saito, S., Sturmfels, B. and Takayama, N. *Gröbner Deformations of Hypergeometric Differential Equations.* Springer, 2000.
39. Takayama, N. KAN/SM1, a Gröbner engine for the ring of differential and difference operators, 2003. Available from `http://www.math.kobe-u.ac.jp/KAN/index.html`.

Development of NZMATH

Matsui Tetsushi

Department of Mathematics, Tokyo Metropolitan University
`tetsushi@tnt.math.metro-u.ac.jp`

Abstract. NZMATH is a system oriented to calculations of number theory, based on Python. Currently, it has several basic data types and several modules for number theoretic computations. NZMATH has two key visions 1) user / developer fusion and 2) speed of development, and the system has been growing along the lines. The development is of open source by nature, and we are making effort to be as agile as possible. There are many areas to be developed, especially a module for algebraic numbers is awaited. Some experimental user interface construction is also discussed.

1 Introduction

There are several systems for number theory already. PARI formerly lead by H. Cohen and taken over by K. Belabas [19] and KANT lead by M. Pohst [6] are the most popular ones. Some more general systems such as MAGMA [9] also provide number theoretic functionality. SIMATH was another system founded by H. G. Zimmer, and had been taken over by Nakamula K. The author had experienced a few years of maintenance of SIMATH and ported it to 64 bit environment within Nakamula's group [10]. After the experience, we decided to make a new system with new concepts in 2003.

1.1 Visions of NZMATH

NZMATH is a system oriented to calculations of number theory, based on the scripting language Python [17]. It is the world's first system for number theory written in a so-called scripting language. The primary goal of the development is to implement various number theoretic algorithms.

The key visions of the NZMATH development are [13,12]:

1. user / developer fusion,
2. speed of development.

The first vision means that ideally there is no distinction between users and developers. From the user's view point, users should be able to be developers easily. Their programs should be merged into the system without difficulties. From the developer's view point, developers should be able to concentrate on implementing mathematical concepts, especially algorithms for number theory, on the system.

© Springer-Verlag Berlin Heidelberg 2006

The second vision means that we put emphasis on the development speed of system rather than the execution speed of resulting programs. It is a paraphrasing of a commonly accepted principle "too early optimization should be avoided." Later in section 3, there will be more detailed discussion about the visions.

In order to realize such visions, a scripting language is suitable. We chose the Python language as the implementation language. A brief introduction to the language is in the next subsection.

For your information, we would like to mention about the other project for mathematics employed Python. SAGE is the project gathering already existing systems into one system [23]. It uses Python as a glue language, and the concept is completely different to that of NZMATH.

1.2 Python

Python is a so-called scripting language. There is no clear definition about what is a scripting language, and even a Unix shell can be counted in with the most broad sense. We, however, restrict ourselves to mean by the word the languages with the following characteristics:

- executed on interpreter via intermediate language,
- without variable declarations,
- dynamically typed, and
- with garbage collection.

Recently, there are a few scripting languages comparative to Python; Perl is the most popular one, and Ruby is becoming more popular.

The development of Python began in 1989 by a Dutch programmer Guido van Rossum [20]. The language is well known with its clear syntax, object-oriented features and rich standard libraries. There are also a lot of third party libraries including mathematical libraries such as Numeric[1], SciPy [21], etc.

The reasons why we chose Python, described in [11], are as follows.

Python has clear syntax; i.e. a program source code looks regular and easy to understand. It is a better point compared to Perl.

Especially for our purpose, i.e. constructing a calculation system for number theory, the existence of a built-in multiprecision integer type is helpful. Python and Ruby have it, but Perl does not.

Object-oriented paradigm is familiar to mathematics. If there is a definition of a mathematical structure, there will be a class definition in a program. Moreover, when mathematician says that a field is a ring, the class inheritance relation will correspond to it. Besides such correspondences, allowing operator overloadings to provide infix notation of binary operations, such as '+' for addition, in programs is useful. Ruby is designed to be an object-oriented language. Python offers a

[1] There are three very similar libraries for numerical computations: Numeric, Numarray [15] and NumPy [16].

mechanism for object-oriented programming. And Perl also offers it but it does not fit well with other part of the language.

There is little difference between Python and Ruby in what they can provide. The choice was made because of the world wide popularity of Python.

1.3 Outline of the Paper

We will summarize the current status of NZMATH in the next section. Then, we will discuss in section 3 about our method of the development and how the visions are realized. Finally in section 4, we will present the future works.

2 Current Status

NZMATH has been developed for over two years. It is provided as a library package of Python. In the Python terminology, a file consists of classes or functions definitions is called "module", and a directory containing modules is called "package".

Currently, NZMATH has several basic data types and several modules for number theoretic computations. It runs with Python 2.3 or higher[2]. Here are the entire module descriptions of the latest release at the time of writing, version 0.5.0. Each heading of the paragraphs in the next few subsections are the names of the modules in NZMATH.

2.1 Basic Data Types

There is a multiprecision integer data type in Python by default, but there must be more data types for number theory.

rational provides rational numbers and rational integers. Although a multiprecision integer type is provided by Python, its result of division is either an integer, i.e. Euclidean quotient, or a floating point number. Our rational integer returns a rational number as a result of division. The module also provides the rational number field $\mathbb{Q}$ and the rational integer ring $\mathbb{Z}$.

integerResidueClass[3] provides integer residue classes. The class for rings of integer residue classes or $\mathbb{Z}/n\mathbb{Z}$ and the class for their elements are defined.

finitefield provides finite fields and their elements. There are six classes, three for fields and other three for their elements. The correspondence reflects the class hierarchy explained with the `ring` module below. The finite fields with prime cardinality are provided as `FinitePrimeField` and their elements are `FinitePrimeFieldElement`. The extended fields and their elements are provided as `FiniteExtendedField` and `FiniteExtendedFieldElement`, respectively. The remaining two classes are the abstract base classes for the classes above.

[2] The latest release at the time of writing is 2.4.3.

[3] The module names `integerResidueClass` and `rationalFunction` are slightly violating the naming convention and can be renamed in the future.

polynomial provides univariate or multivariate polynomials. The basic polynomial class can be used with any commutative rings as the coefficient ring. There are specialized ones for special coefficient rings; such as integers or rational numbers. Polynomial rings are also provided.

matrix, vector provides matrices, vectors and related functions such as determinant, trace, LU decomposition, etc.

rationalFunction[3] provides a class `RationalFunction` for rational functions. It is only for the result of a division of polynomials.

2.2 Number Theoretic Modules

The main part of the system for number theory. There are only a few modules now, but the development directed to enrich this category in the future.

multiplicative provides multiplicative number theoretic functions such as the Euler totient function φ, the Möbius function μ, and the sum of k^{th} powers of all divisors σ_k.

gcd provides some functions related to compute the greatest common divisors of integers.

prime provides primality testing functions from the trial division to the Jacobi sum method [4, section 9.1], and some functions related to prime generation.

factor provides integer factorization functions[4]. There are methods from the trial division to the multiple polynomial quadratic sieve method [8].

elliptic provides elliptic curves and related functions. Elliptic curves are defined over a finite prime field with any characteristic but 2 or 3, and their order and structure of the Mordell-Weil groups can be computed [7,1]. The structure computation is carried out via the Weil pairing.

quad provides class number computation of imaginary quadratic fields. The method used in the module is to count reduced quadratic forms [7,1]. Other methods will be implemented.

2.3 Miscellaneous Modules

The rests are auxiliary modules.

arith1 provides variety of functions: `floorsqrt` to compute the integer part of the square root of an integer, `log` to compute the integer part of the logarithm of an integer with specified integer base, etc.

[4] Actually, `factor` is not a plain module like others, but a sub-package consists of a few modules.

bigrandom provides random number generator for big numbers. It is based on the Python's default random number generator, which only provides random numbers in 32 bit.

combinatorial provides combinatorial functions including factorial, binomial coefficients, the Bernoulli number and the Catalan number.

ring provides abstract declaration of rings. There are two hierarchical trees of classes: one for rings and the other for their elements.

For example, `Ring` is the root class of ring classes. It declares that all descendants must have attributes `zero` and `one`, in other words, we call with 'ring' a ring with unity. Then, `CommutativeRing` inherits `Ring`. and so on.

The counter part of the `Ring` class in the other tree is `RingElement`. The definition of an element of a ring is too abstract to write down concrete code in the class, there is almost nothing in the body of the class definition. However, there is a commonly used method name to know a ring to which the element belongs: the `getRing` method.

There are also classes for fields, since field is a kind of commutative rings. We do not much care about skew fields and do not provide them.

lattice provides the LLL algorithm of lattice [4, section 2.6].

zassenhaus provides factorization of integer coefficient polynomials. The algorithms to do it are the Berlekamp-Zassenhaus method and van Hoeij's algorithm [5].

equation provides functions to solve algebraic equations $f(x) = 0$.

group provides finite abelian groups and methods to compute their group orders.

permute provides permutation groups.

real provides functions for real numbers. The module corresponds to the `math` standard module of Python. There are no definitions for real numbers or arbitrary precision floating point number. The results are in rational numbers and very slow.

imaginary provides functions for complex numbers, but since the name `complex` is already used by Python for the type name of complex numbers, it is named `imaginary`. Similar to the `real` module, the `imaginary` module corresponds to the `cmath` standard module of Python.

2.4 Manual

The manual is provided as HTML files included in the distribution. It is also available on the web[5]. With the Python interpreter, `help` function can extract

[5] NZMATH manual: `http://tnt.math.metro-u.ac.jp/nzmath/manual/`

documentation strings embedded in the source files. This functionality is a part of Python, and the extracted document is also viewable from shell command `pydoc`. However, documentation strings are not always written for all functions. Some programmers sometimes forget to write them, and some feel hard to write them in English. The `help` on the interpreter is, thus, not always helpful.

Instead of the documentation strings in the source code, HTML manual is maintained with a wiki[6] system to keep up to date. Someone who feel less hard to write English can help to write the wiki manual pages, if the programmer feels it is hard. Then, on the time of releases, the pages are converted to plain HTML files.

In addition to the process, the wiki approach has an advantage that translations of the manual into other languages than English are possible. It is recommended by many people that documentation should be in the code, but it is reasonable only for a domestic project. Having manuals in more than one languages in the code seems ridiculous. NZMATH must be an international project, and there are needs for translated manuals. Though English is the lingua franca in the academic world for the last half of a century, neglecting other languages is not justified.

2.5 Applications

There is a research using NZMATH; related to a quantum public key cryptosystem (QPKC), some computations have been carried out with NZMATH by Nishimoto and Nakamula [14]. The computations are about norms of quadratic or cubic field elements. Since there is no module for such elements, they write their own modules with the help of some NZMATH functions.

We are now implementing algebraic numbers on NZMATH, as we will discuss in section 4.1. Integration with their implementation is hoped.

3 Development

We have been developing NZMATH for over two years. The visions explained in section 1 remain unchanged through the development. We now have two more development concepts: “open source” and “agile development”.

3.1 User / Developer Fusion

One of the NZMATH’s visions is to get rid of the distinction between users and developers.

From the developer side, since we have chosen Python as the implementation language, developers do not have to think about memory management nor portability of code; such features are the subjects for the development of the language itself. In other words, developers can concentrate on mathematical problems. It is a direct benefit of choosing Python toward user / developer fusion.

[6] NZMATH Wiki: `http://hanaya.math.metro-u.ac.jp/nzmath/`

From the user side, the fact that the language they use and developers use is the same language makes contributions easy. The already existing systems have different languages for users: GP for PARI, KASH for KANT, simcalc for SIMATH. The language gap hardens for users to feed back their experience to the system development. In addition to it, since the Python language is widely used and general purpose, the knowledge is useful even apart from NZMATH.

It is true that user base is still too small to say anything. When the number of users increases, our claim shall be proven.

3.2 Speed of Development

Another one of the NZMATH's visions is to put emphasis on speed of development. The choice of Python as the implementation language helps to achieve the vision. Since the NZMATH project have started later, catching up faster to already existing systems is necessary to be accepted by a broader audience.

Python has been chosen partly because of reducing educational cost of students. The way of developing the software is influenced by resources. Our dominant resource is a human resource. There is no full-time staff; main developers are master course or undergraduate students. They leave from the development in a year or two on time of their graduations. It is, thus, desirable to choose a language easy to learn.

Almost all other systems are implemented in C or C++, because the resulting programs run fast, but it is "too early optimization." We would like to devote to implement algorithms for number theory. Writing in C like language, however, requires other knowledge such as memory management or hardware variations. Modern programming languages including Python offer automatic memory management and hide the hardware layer from programmers. Another feature of Python is that it is dynamically typed language. Statically typed languages can detect more inconsistency of types of objects at the compilation time. There are two strategies of static typing: variable declaration and type inference. C uses former strategy, in other words, programmers are forced to declare types for every variable. The latter strategy is mainly used for functional languages and we do not argue about it here. Spreading dynamically typed languages indicates that costs of such declarations are not worth to pay. Both features of Python enable the developers to concentrate on mathematics, as desired. If it is realized later that a part of the system needs more speed, then it is possible to reimplement the part by C with the experience of implementation in Python.

We achieved writing the modules explained in section 2 in less than 3 years. It shows that our choice of language was right.

3.3 Open Source

We distribute NZMATH with the BSD license. The license basically permits users to do anything. The program distributed under the license can be freely available, freely redistributed, and freely used. Since we have chosen a scripting language as the implementation language, users can always read the source code to be interpreted. It is, therefore, natural to make our source programs available freely. It means our system can be distributed as an open source software. Moreover, we believe that there is no need to restrict usages of the system; even including it to a closed source software. For such purpose, the GNU general public license, which is used by PARI for example, is too restrictive, because the license does not allow derived works without source code accessibility. Therefore, our choice of license is the BSD license.

3.4 Agile Development

Because the author is personally influenced by the ideas of agile paradigm [3] or, more precisely, extreme programming methodology [2], we shall discuss about the development here from such a point of view. It is clear that our way of development is not of "water fall" model, but rather incremental one. Therefore, there should be some hints for improvements from agile models.

The first point is about tests. In the context of a professional software development, the code coverage may have importance, but we do not require such a level of test for all developers. It is true that writing merely a test to see that the usual cases run correctly is effective to prevent most of ridiculous bugs. I do not know whether the other projects have explicit test codes or not, but it is worth writing.

The second point is about code reviews. By reviewing the code the other member have written, the members can learn about the module reviewed, acquire the sense of goodness of code, or detect minute bugs. There might be possibility to practice pair programming instead, since pair programming is a sort of continual reviewing. However, the development group has master course students as its main members, and they need their own result for their degrees, the activity that makes personal contributions ambiguous is not acceptable.

The third point is to release small. Our release schedules have not been so punctual, but the releases were, at least, the results of shortly scheduled plans. The plans include a module to be added or to be modified. If there is no such module because the development of it will take more than the usual release period, we plan a bug fix release. Such continual releases may stimulate the potential users, result some feed backs, and keep the motivations of the members.

The last point is about incremental design. Design should evolve day by day, not because demands for number theory system change, but because developers' understandings of the area progress. Regretfully, on this point, the activities such as discussing the design or writing the design explicitly on a paper before coding are rarely done. Moreover, refactorings after implementations are occasional. It is necessary to improve these practices.

4 Future of NZMATH

The development of NZMATH will continue further. We will discuss about some short term perspectives in the next subsections, then about longer term perspectives.

4.1 Algebraic Number Field

The algebraic number field is the next big subject for NZMATH development. The only thing NZMATH has had toward the area is the `quad` module, which computes the class numbers of imaginary quadratic fields.

Since algebraic number fields offer a fundamental tool to study number theory, NZMATH should handle them to be more "number theory oriented" system.

We have started writing programs for it after the release of 0.5.0 in February. The very first step is to define classes to represent algebraic numbers with arithmetic operations. The methods to compute norm or trace are also provided. The next step will be to define algebraic number fields and their integer rings. Then, invariants of the fields, such as discriminants, class numbers, unit groups, etc. will follow. Handling ideals and computing prime decompositions, will also be provided.

There will be specialized classes or modules for small degree fields similar to the `quad` module: real quadratic and/or cyclic cubic fields. Other kinds of fields may also have specialized modules, depending on the easiness of implementation.

4.2 Web UI

There has been no special interface for NZMATH other than the default Python interpreter. Though we think the interpreter remains as the primary interface, a possibility of another interface has been sought occasionally.

We are planning to make a web user interface. By "web" we mean that users will use their web browsers to start calculations. The most simple implementation looks like the Online MATH Calculator by Stein [22]. There can be, on the other hand, several choices how to construct the server side: a good old CGI, a plug-in module for web server or a dedicated web application server. Fortunately, there are plenty of choices with Python web servers.

The heavy calculations are allowed to be carried out only on users' desktop with or without the web interface. If users will be able to run the server on their own desktop easily, the web user interface will still be useful. Such installation of server / client pair on user's desktop is called "desktop server"[7].

The merits of constructing a user interface with the web technology comparing to with an independent client are: (1) the required knowledge for developer is smaller and (2) it is platform independent. It can be the first step towards using NZMATH for potential users, if the pages contain more introductory materials or integrated with manual pages.

The weakness of the web interface is lack of programmability. The Python interpreter, therefore, remains as the interface for programming, and we expect that almost all users will program anyway.

4.3 Speed of Execution

It has been and will be always the problem how to respond to the request of speed. We have chosen to put higher priority on easiness of programming and

[7] The author does not know who invented the word, but have learned the concept from PyDS blog system.

speed of development rather than on speed of execution. The formerly planned schedule about speed of execution is that at some point when the pace of additions of new modules will slow down, running time improvement will be a main topic of the development.

The strategy is endangered if execution is too slow to make the total time of programming and execution is slower with NZMATH than with other systems. We should, then, make some optimizations for speed of execution. There are requests of speeding up polynomial computation, for example. It is in fact the bottleneck of a certain kind of computations. Our polynomial implementation is designed for general purpose, i.e. choosing the variable and specifying coefficient ring are always required even if one only deals with polynomials in a specific ring; $\mathbb{Z}[X]$ for example. They are obviously eliminatable overheads if there is a specialized polynomial type. Use cases show that there are a few very commonly used types of polynomials. It is, thus, possible to make such special types to resolve the bottleneck of polynomial computations.

4.4 Outsourcing

The project uses a mailing list, a CVS repository, web pages and a wiki. The `nzmath-user` mailing list[8] is for user to user communications. The CVS repository is for version control of the source code. The web pages are for releases of NZMATH to the public, and for the user manual. And the wiki[6] is mainly for communications among developers, including maintenance of the user manual and bug tracking. Currently, all services are served on the machines in our laboratory. They, however, need not be served by ourselves. The Internet has grown rapidly during the last decade. Universities are not the only places to host such servers. Nowadays, there are several sites hosting free / open source softwares with version control softwares like CVS, mailing lists and web pages. We are planning to use one of such sites.

There are several merits and a few demerits. The first merit is that the project will be recognized widely as an open source project. Secondly, participation to the development will be easier. People may hesitate to edit wiki pages if the host is in a university domain. The last point has only a small importance, but it will reduce our daily routines as a side effect. One of the demerits is that we loose control of server softwares, but it is negligible compared to the merits.

4.5 Long Term Plans

Finally, we will discuss about long term future plans.

Of course, the implementation and improvement of wider range of algorithms will continue. Especially, construction of modules for invariant computations of algebraic number fields and elliptic curves continues to be in the main stream of the development. Elliptic curves over the rational field and hopefully algebraic

[8] `nzmath-user` mailing list: `nzmath-user@tnt.math.metro-u.ac.jp`

number fields should be a target. We hope to provide also some tools for analytic number theory.

Speeding up execution may be one of the main topics, when the development will be saturated with variety of modules or when we find a bottleneck of the speed of computations. The former seems very far from now, but the latter may start earlier.

Connecting to other systems is another future issue. Implementing some protocol stack will be in consideration. OpenXM [18] used in Risa/Asir is one of the candidates.

5 Conclusions

We have been developing the calculation system for number theory called NZMATH for over two years. The system is implemented in Python, a scripting language, in order to achieve the visions 1) user / developer fusion and 2) speed of development. The development has successfully been providing the wide variety of modules and the manuals of them, in the short period. It shows the correctness of the concepts.

There will be more topics to be covered by the system, including algebraic number fields. Web user interface is considered as a probable option to be implemented. Speed of execution is not ignorable factor of the development. Such problems arose in the development are discussed.

We hope the system be accepted by wider range of people, from number theorists to students of other areas.

Acknowledgment

We would like to express our thanks to Nakamula Ken, Uchiyama Shigenori, Hirano Takato and Nishimoto Keiichiro for helpful discussions and comments, the anonymous referee for the kind suggestions, and the members of the NZMATH development group and other contributors for their efforts of restless improvement of the software.

References

1. Antonio, C. A. M., Saito, K., Tanaka, S., Asuncion, J. S. and Nakamula, K.: Implementation of imaginary quadratic fields and elliptic or hyperelliptic curves over finite prime fields on the system NZMATH for number theory; AC2005 presentation, November 2005.
2. Beck, K. and Andres, C.: “Extreme programming explained: embrace change” (2nd ed.), Pearson Education Inc., 2005.
3. Beck, K., et al.: Manifesto for Agile Software Development; `http://www.agilemanifesto.org/`.
4. Cohen, H.: “A course in computational algebraic numbers”, Springer-Verlag, 1993.

5. van Hoeij, M.: Factoring polynomials and the knapsack problem; Journal of Number Theory 95 (2002), no. 2, pp.167–189.
6. The KANT Project: KANT / KASH; `http://www.math.tu-berlin.de/%7Ekant/kash.html`.
7. Komai, H.: Implementation of arithmetic of elliptic curves over finite prime fields on NZMATH; master thesis, Tokyo Metropolitan University, January 2005.
8. Kumaki, K.: Implementation of multiple polynomial quadratic sieve on the number theoretic system NZMATH and its analysis (Japanese); master thesis, Tokyo Metropolitan University, January 2005.
9. The Magma Computational Algebra System; `http://magma.maths.usyd.edu.au/magma/`.
10. Matsui, T., Kobayashi, D., Abe, M. and Nakamula, K.: SIMATH — Recent development in TMU; in Cohen, A.M., et al. (ed.) "Mathematical Software" Proceedings of ICMS2002, World Scientific, August 2002.
11. Matsui, T.: Development of computational number theory system by a scripting language (Japanese); in Noro M.(ed.) "Computer Algebra – Design of Algorithms, Implementations and Applications" RIMS Kokyuroku No.1395 pp.144-149, Kyoto University, October 2004.
12. Matsui, T.: NZMATH — past and future of the development; AC2005 presentation, November 2005.
13. Nakamula, K., Matsui, T.: Developing a system for number theory by script language — Announcement of the release of NZMATH 0.1.1; Algorithms and Number Theory, Dagstuhl, May 2004. `http://tnt.math.metro-u.ac.jp/%7Enakamula/talk/dag2004-a.pdf`.
14. Nishimoto, K., Nakamula, K.: Computer experiment on key generation for the quantum public key cryptosystem over quadratic fields; AC2005 presentation, November 2005.
15. Space Telescope Science Institute: Numarray; `http://www.stsci.edu/resources/software_hardware/numarray`.
16. NumPy; `http://www.numpy.org/`.
17. NZMATH development group: NZMATH; `http://tnt.math.metro-u.ac.jp/nzmath/`.
18. OpenXM, a project to integrate mathematical software systems; `http://www.openxm.org/`, 1998–2005.
19. the PARI group: PARI/GP Development Headquarter; `http://pari.math.u-bordeaux.fr/`.
20. van Rossum, G.: Foreword; "Programming Python" (1st ed.), O'Reilly, May 1996.
21. SciPy; `http://www.scipy.org/`
22. Stein, W.: Online MATH Calculator; `http://modular.math.washington.edu/calc/`.
23. Stein, W.: Software for Algebra and Geometry Experimentation; `http://modular.fas.harvard.edu/SAGE/`.

KASH: Recent Developments

Sebastian Freundt[1], Aneesh Karve[2], Anita Krahmann[1], and Sebastian Pauli[1]

[1] Institut für Mathematik, MA 8–1, Technische Universität Berlin
Straße des 17. Juni 136, 10623 Berlin, Germany
{freundt, krahmann, pauli}@math.tu-berlin.de
[2] Computer Sciences Department, University of Wisconsin - Madison
1210 W. Dayton St., Madison, WI 53706-1685, USA
karve@cs.wisc.edu

Abstract. In recent years the computer algebra system KASH/KANT for number theory has evolved considerably. We present its new features and introduce the related components, QaoS (Querying Algebraic Objects System) and GiANT (Graphical Algebraic Number Theory).

1 Introduction

KASH/KANT is a computer algebra system specialized for algebraic number theory and its applications. KANT stands for "Computational Algebraic Number Theory" with a slight hint of its German origin. It contains the following system components:

- The KANT C library with highly specialized algorithms for number theory
- KASH, the KANT shell, and its programming language
- The graphical user interface GiANT
- The QaoS databases for algebraic objects with access via the worldwide web, or from computer algebra systems

The KANT library for number theory has been in development since 1987 under the leadership of Michael Pohst. Development began in Düsseldorf, and since 1993, continues at Technische Universität Berlin. The following describes the evolution of the components of KASH/KANT.

1987	KANT V1 (Fortran library)
1992	KANT V2 (C library) built on the Cayley platform
1994	KANT V4 (C library) built on the Magma platform
1995	KASH 1.0 (KANT shell) based on GAP 3
1996	KASH 1.6, Database for number fields
1999	KASH 2.1
2004	KASH 2.4, WWW Database
2005	KASH 3, GiANT (GUI for KASH 2.x), QaoS Databases

The KANT V4 C library, built on the Magma [BCP97] platform, provides the core functionality of KASH. (KANT V4 functionality is also available in Magma.)

© Springer-Verlag Berlin Heidelberg 2006

The current version uses the GNU multi-precision library [GMP] for long integer arithmetic, and [MPFR] for arbitrary precision real numbers. KASH consists of a modified version of GAP 3, which is linked to the KANT V4 library, a variety of utility functions to provide an interface to GAP 3, and numerous functions written directly in the native KASH shell language, that make up the KASH library. Some of the GAP 3 library functions are also available in KASH. An overview of KASH functionality is found in section 2.

In recent years we have redesigned key components of KASH. Our main purpose was to create a system that was easier to maintain and extend. To this end, we added a number of object-oriented features to KASH 3, although we were limited since a ground-up rewrite was not feasible. With these new features, users can now write *generic functions* which can be used, for example, with global fields *and* number fields alike. To support generic functions we have introduced identifier overloading for function names (see section 5) and a new type system, which also allows for *user-defined types* (section 3).

In KASH 3, documentation (section 4) is written directly into the source code making documenting new functions simpler, and general maintenance easier. Similarly, the online help system and downloadable manuals are automatically generated from in-line comments in the source code.

In section 6 we introduce the redesigned and expanded KANT database for number fields (now called QaoS – Querying Algebraic Objects System).

While most algebra is done by writing text and formulas, diagrams have always been used to present structural information clearly and concisely. Text shells are the *de facto* interface for computational algebraic number theory, but they are incapable of presenting structural information graphically. In section 7 we present GiANT, a newly developed graphical interface for working with number fields. GiANT offers interactive diagrams, drag-and-drop functionality, and typeset formulas.

KASH and Other Computer Algebra Systems. As already mentioned, KASH/KANT shares parts of its C library with Magma. KASH also integrates easily with other computer algebra systems. The package Alnuth [AED05] allows access to the functionality of KASH 2.x from GAP 4. KASH 3 can be accessed through the Python based computer algebra system SAGE [St06].

The SCIEnce (Symbolic Computation Infrastructure for Europe) project, funded by the European Union, aims to link the computer algebra systems GAP 4 [GAP], KASH/KANT, MuPad [Mu05], and Maple [Ma05] in the context of Web and Grid services. In this scheme, each of these systems will run as a server or a client. The project will answer many questions about access/transport for complex objects. In particular the project team will develop an OpenMath [OM] XML format for the algebraic objects used by these systems.

Availability. KASH 3 is freely available. Binaries for Linux/x86, Mac OS X, and MS Windows can be downloaded from

`http://www.math.tu-berlin.de/~kant/kash.html`.

In addition KASH 2.x was ported to various UNIX versions (AIX, IRIX, OSF/1, Solaris).

2 Functionality

In the 1970s, Hans Zassenhaus postulated four principal tasks for computational algebraic number theory. Namely, the development of efficient algorithms for the computation of the maximal order, the unit group, the class group, and Galois group of algebraic number fields. We now have these algorithms, not only for number fields but also for global functions fields, and they are fast enough to make the computation of more complex objects possible, such as class fields.

KASH offers powerful functions for working with number fields, function fields, and local fields and for solving Diophantine equations. The only comparable systems in this regard are Magma [BCP97], which shares parts of its code with KASH, Pari-GP [BB$^+$05], and SAGE [St06], which contains Pari.

Number Fields

KASH allows base arithmetic and integral basis computation for extensions of the rationals and for relative extensions of number fields. Efficient algorithms for class groups and unit groups are applied in the computation of ray class groups [HPP03], which in turn enable the computation of class fields [Fi02]. As a further generalization of class groups, Picard groups of arbitrary orders can be computed. The computation of Galois groups is now possible for number fields of degree up to 23 [Ge05].

Example. We determine a ray class field over a number field.

```
kash% K := NumberField(X^3+6*X+3);
Number Field with defining polynomial X^3+6*X+3 over Q
kash% O := MaximalOrder(K);
Maximal Equation Order with defining polynomial X^3+6*X+3 over Z
```

Next we pick the index 3 subgroup of the ray class group with conductor (3). The composed types such as `grp^abl` are explained in section 3.

```
kash% r := RayClassGroup(3*O);
Abelian Group isomorphic to Z/6, extended by:
  ext1 := Mapping from: grp^abl: r to ids/ord^num: _AF
kash% q := Quotient(r,3*r.1);
Abelian Group isomorphic to Z/3, extended by:
  ext1 := Mapping from: grp^abl: r to grp^abl: q
```

The corresponding ray class field is given as an abstract extension. A representation is computed if and only if the user requests it.

```
kash% L := RayClassField(Inverse(q.ext1)*r.ext1);
Abelian Field, defined by (<[3, 0, 0]>, []) of structure: Z/3
kash% EquationOrder(L);
Equation Order with defining polynomial X^3-3*X-1 over o
```

Function Fields

Numerous algorithms for number fields have been generalized to global fields. For example, maximal order algorithms have been generalized to algorithms for the computation of finite and infinite maximal orders. Many tasks in the theory of function fields rely on the computation of Riemann-Roch spaces, which are applied to the computation of divisor class groups [He02]. Galois groups for function fields up to degree 23 can also be determined [Ge05].

Future work will provide increased support for elliptic curves, automorphism groups of function fields or curves, and isogeny and isomorphy computations. Functions for computing endomorphism rings of hyper elliptic curves and Deuring correspondences are also planned, as well as support for function fields over the complex numbers.

Example. We generate a function field `L` whose constant field is a function field `K`, and compute its finite maximal order.

```
kash% k := GF(7);; kx := FunctionField(k);;
kash% kxy := PolynomialAlgebra(kx);
Polynomial Ring over rational function field over GF(7)
kash% K := FunctionField(kxy.1^2+6*kx.1^7+2*kx.1^4+6*kx.1+3);
Algebraic function field defined over Univariate rational function
field over GF(7) by kxy.1^2 + 6*kx.1^7 + 2*kx.1^4 + 6*kx.1 + 3
kash% L :=ConstantFieldExtension(K,K);;O := MaximalOrderFinite(L);
Maximal Equation Order of L over Polynomial Ring over K
```

We generate a non-trivial Deuring correspondence of the function field `K` represented by the ideal `V`. We determine its Riemann-Roch space, which then could be applied to show that the correspondence is not trivial.

```
kash% BindNames_(CoefficientRing(O),["X"]);
kash% V := Ideal(O, [L.1-K.1,X^2+kx.1^2-2*kx.1*X-2*X-2*kx.1+1]);
Ideal of O, Basis:[X^2 +(5*kx.1+5)*X+kx.1^2+5*kx.1+1  0][6*K.1  1]
kash% M := RiemannRochSpace(2*Divisor(V));
KModule M of dimension 2 over K, extended by:
  ext1 := Mapping from: mdl/fld: M to fld^fun: L
```

Local Fields

KASH supports p-adic fields and their extensions as well as fields of Laurent series. Polynomials over local fields can also be factored; [Pa01]. Applications include the computation of integral bases, ideal decomposition over global fields, and completions of global fields. Support for computing unit groups and class fields over p-adic fields is scheduled for the next KASH release.

Example. We use an Eisenstein polynomial to generate a ramified extension over $\mathbb{Q}_3$. It is the completion of the number field `K`, from the number field example above, with respect to the ideal over 3.

```
kash% Q3 := pAdicField(3,10);
3-adic field mod 3^10
kash% V := TotallyRamifiedExtension(Q3,X^3+6*X+3);
Totally ramified extension defined by X^3+6*X+3 over Q3
```

We factor the generating polynomial of the class field `L/K` from the number field example over the field `V`.

```
kash% S := PolynomialRing(V);
Univariate Polynomial Ring over V
kash% Factorization(S.1^3-3*S.1-1,rec(Certificates:=TRUE));
[<S.1^3+((4*V.1^2-2*V.1 + 25)*V.1^3+O(V.1^30))*S.1-1+O(V.1^30),1>]
extended by certificates := [rec(F := 1, Rho := 1, E := 3,
Pi := (V.1^-1+O(F.1^29))*S.1+(V.1+2)*V.1^-1+O(F.1^29))]
```

Note that `V.1` denotes the uniformizer of the field `V` and that `S.1` is the indeterminate of the polynomial ring `S`. The polynomial $\texttt{S.1}^3 - 3 \cdot \texttt{S.1} - 1$ is irreducible, as expected. The certificates tell us that it generates a totally ramified extension of degree `E`=3 over `V`. Thus the ideal over 3 is totally ramified in `L` with ramification index 9.

Diophantine Equations

KASH contains efficient functions for solving absolute and relative norm equations [Fi97], Thue equations [BH96], and unit and index form equations [Wi00].

Example. We solve the Thue equation $X^3 + X^2Y - 2XY^2 - Y^3 = 7$.

```
kash% T := Thue(X^3+X^2-2*X-1);
Thue object with form:  X^3 + X^2 Y - 2 X Y^2 - Y^3
kash% Solutions(T,7);
[ [ -3, 2 ], [ 1, -3 ], [ 2, 1 ] ]
```

3 Type System

The type system of KASH 2.x was simple. When asked for the type of an object, KASH returned a string containing a description of the type. Most functions contained type information for their main argument and return value. Type checking was typically done using predicates, like `IsOrder()`. For KASH 3 we wanted to create a type system that met the following criteria:

– Represent mathematical categories in an intuitive way
– Allow efficient comparison of types (to facilitate overloading, inheritance)
– Allow the creation of user-defined types

That said, we needed to maintain backwards compatibility with the type system provided by the underlying KANT C libraries. We also needed to address the existence of data structures unique to the GAP language, such as lists. In creating the KASH 3 type system we studied the GAP and MAGMA type systems.

In the transition from GAP 3 [Sc$^+$93] to GAP 4 [GAP], the overloading of functions based on the predicate-driven type system was introduced. A function is installed as a method with a list of predicates that are required for the function to be called. This generalization of the predicate driven approach would have been difficult to combine with the types in the KANT C library.

Magma [BCP97] has a categorical type system. Types are internally organized in a category inclusion graph. A type is said to be related to another type if there is a path of inclusions from the one type to the other. Types in function signatures are matched using this relation. Recently, extended types have been introduced that make it possible, for instance, to distinguish the types of polynomial algebras over various rings. Magma does not support user-defined types.

Taking all this into consideration, we designed a new type system for KASH 3. Three kinds of types exist: simple types, typed aggregates, and composed types.

Simple Types include basic types like `char` and `type`, or more specialized types such as `thue` for Thue equations. Simple types also include data structures (aggregates) such as `list`, `dry` (lists with no duplicate entries), `set` (sorted lists with no duplicate entries), `record` , and `alist` (associative lists).

Typed Aggregates extend aggregates to include one or more additional type specifications. They are similar to parameterized types found in modern programming languages. The syntax for typed aggregates is *aggtype*`(`[`,`*type*]`)`. Typed aggregates can be created from aggregates: sequences (`seq()`), maps (`map()`), tuples (`tup()`), and `nof()`. The latter stands for "*n* of". It can be used in defining functions with a variable number of arguments. For example, a function that takes an arbitrary number of string arguments would have argument type `nof(string)`.

Composed Types represent complex mathematical objects. Composed types are composed of atoms. The first atom is either `elt-` or `str-`, which stand for element and structure respectively (elements are contained in structures). The first atom is followed by a *structure* atom, a caret (`^`), and a *specifier* atom. To take an example, `elt-fld^num` is the type of elements of number fields. The *structure* atoms describe the general algebraic structure, such as `fld` (field), `alg` (algebra), `grp` (group), etc. The *specifier* atoms give a more detailed description `fin` (finite), `pol` (polynomial), `abl` (abelian), etc.

Still more complex types can be constructed by appending a slash (`/`) followed by further *structure*`^`*specifier* pairs. The atom `str` can be left out unless it stands alone. In summary, the syntax for composed types is as follows: [`elt-`]*structure*`^`*specifier*[`/`...]. For examples `grp^abl` denotes the type of abelian groups, `fld^fun` the type of function fields, `ord^num` the type of orders of number fields, `elt-ord^num` the type of algebraic integers, and `alg^pol/fld^num` is the type of polynomial algebras over number fields.

Given the type of an element within a mathematical structure, we obtain the type of the parent structure by prepending `str-`, and vice versa by prepending `elt-` to the structure's type. Users can create new composed types by combining existing atoms, or by defining new atoms and combining those.

User-Defined Types Objects of user-defined types are represented as records whose `.type` field contains the type of the object. If a record `r` contains a `.base` component then `r` inherits most of the functionality of `r.base`. The functionality can be overwritten by methods installed for the type assigned in the `.type` field or by the `.operations` field which, as in GAP, can be used for overloading the function `Print` and the infix operations `+`, `-`, `*`, `/`, `^`, `in`, and `mod`.

Further special record fields are `.`n (where n is a positive integer) which contains the n-th generator of a structure and `.parent` for the parent structure of an object.

4 Documentation and Help

We designed the new documentation system for KASH with the following goals in mind.

- All documentation should be prepared from a single source. (This provides ease of maintenance, consistency, and up-to-date accuracy.)
- Documentation for new functions should be available to the user as soon as new functions are loaded into the system.
- The help system should support complex search patterns, including full text search (similar to Internet search engines).
- Function documentation should, whenever possible, be accompanied by examples written in working code.
- Output formats should be as follows: ASCII for help within the shell environment (we call this *online help*), XML/HTML for worldwide web documentation, and LaTeX/PDF for printed documentation.

Experience with previous releases of KASH, as well as other computer algebra systems, has taught us that users benefit tremendously from illustrative examples. We have therefore made an effort to provide an example for every standard KASH function. The system also includes special features to credit the authors of each example, and to provide cross-references to other parts of the system and the mathematical literature at large.

Implementation

We decided to represent entries in the documentation system as KASH records. As a result, KASH itself can be used to automatically process and generate documentation. All functions for converting the documentation to any of the desired formats are written in KASH. In this way future developers will not have to find their way through scripts written in other languages. A further advantage is that the documentation system can be easily extended.

The following is a list of the key fields present in a documentation record.

`kind` defines what is being documented.
`name` contains the name of the documented object.
`sin` is the input signature of a function as a list of the input-types.
`sou` is a list of the output-types of a function or constant.
`short` is a text describing the documented object.
`author` contains a list of authors.
`ex` contains examples illustrating how to use a function or generate an object of the given type.
`see` contains a list of references to related documentation records.

Each help entry is identified by a unique hash value computed from the `name` entry and, where applicable, the `sin` entry. The function `DocHash` returns this hash value. The hash values are used in the cross references in the `see` entry for example.

Passing a documentation record to the function `InstallDocumentation` automatically enters it in the online help system and notes the source of the documentation, which can be a file or the interactive shell session.

Online help is accessed by typing `?` into a KASH shell. The `?` may be followed by a variety of modifiers in order to generate more specific results for example `?*` for a full text search, `?(`*type*`)` for finding functions with *type* in the input signature, or `?->`*type* for functions returning objects of type *type*. If more than one result is found, a list is displayed with each item labeled with a four-digit integer, *n*. Their documentation is accessed with `?`*n*.

While online and worldwide web help are accessed in a random order, printed documentation is comprehensive in scope. Printed documentation is assembled from the documentation records described above. The hierarchical structure of the document is determined by a recursive structure of records and lists, where documentation records are referenced by their hash values. The result is translated into LaTeX by a KASH function.

5 Overloading and Type Matching

Like many modern programming languages, KASH 3 supports overloading of functions. Overloading allows users to define multiple functions with the same name but different parameter types. We call such overloaded functions *methods*. Overloading is made possible by KASH 3's strongly typed object system (see section 3) that replaces the predicate driven object system of prior releases.

Calling a method is syntactically equivalent to a function call. Precisely which method to execute is determined at run-time by the method's *input signature* (i.e. the type and order of the method's parameters).

Closely related is the concept of *type matching*. Type matching is the process by which individual function or method calls are matched against known signatures. As with other object-oriented languages, KASH 3 types form a transitive hierarchy. KASH 3 includes a handful of wild card atoms which can be used as simple types, or as parts of composed types. The wild card `any` matches any type or atom, `loc` (local) matches `ser` (power series) or `pad` (p-adic), and `rng` (ring) matches `ord` (order) or `fld` (field). As an example, to write a function that takes an algebraic integer or an algebraic number, we would use the type `elt-any^num` in its input signature.

The overloading mechanism is connected with the documentation system. The `name` entry of a documentation record specifies the method name which the signature `sin` should be added to. The installation of new methods resembles the installion of documentation with a function as a second parameter.

6 The QaoS Databases

Databases facilitate the discovery of constructive examples and broad, heuristic observations. The KANT database is one of the world's largest databases for algebraic number fields. In KASH 3, we wanted to generalise the database design and to improve accessibility. The result is QaoS (Querying Algebraic Objects System), an easily integrated, stand-alone solution for access to the KANT databases. QaoS is more versatile and extensible than its predecessors. An improved representation for elements and polynomials makes it possible to represent (relative) algebraic and transcendental extensions.

QaoS incorporates data from the KANT database [DW96] for number fields, and tables of transitive groups from [Hul05]. The latter are linked to the Galois group entries in the tables of field extensions. Currently QaoS provides the following information

- Algebraic Extensions of $\mathbb{Q}$ up to degree 9 (over 1.3 million number fields)
- All 40226 Transitive Groups up to degree 30
- Extensions of $\mathbb{F}_p(t)$ and $\mathbb{Q}(t)$ (experimental)

The number field records contain generating polynomials, signatures, discriminants, regulators, structure of class groups, and Galois groups.

Accessing the QaoS Databases. Designed as a client-server application, QaoS transfers information via the hypertext transport protocol (HTTP). Query formats are identical across clients. On the server side, a script translates the client query into SQL (structured query language) for use by the PostgreSQL [PSQL] database. Another script translates the result back into the format requested by the client. A web browser, for example, would select hypertext markup language (HTML); whereas a computer algebra system, would choose a format easily read by the system. In computer algebra systems the client functions are written in the system's native shell language. They call a non-interactive HTTP client for communicating with the server. The results are read using shell or string pipes, or a temporary file. If a new client can evaluate strings, porting the database interface is straightforward. Currently, client functions are available for GAP 4, KASH 2.56, KASH 3, Maple, and SAGE (contributed by Steven Simek).

Representation of Field Extensions. An algebraic extension L/K can be represented by a generating polynomial $f(x) \in K[x]$ such that $L \cong K[x]/(f(x))$. As, in general, databases do not support robust polynomial records, the polynomial f is represented as a list of coefficients. This works well if the coefficients of f are of a type that is supported by the database - integers, for example. In general however, this is not the case. If we consider the extension $\mathbb{F}_p(t)[y]/(g(y))$ with $g(y) \in \mathbb{F}_p(t)[y]$ of the rational function field $\mathbb{F}_p(t)$ over $\mathbb{F}_p$, we see that the polynomial $g(y)$ would have to be represented as a two-dimensional array of integers, but PostgreSQL does not easily support lists of lists of varying dimensions. We solve this problem by storing all elements needed to represent the generating polynomials as lists that contain either references to lists or references to integers. This approach can be used for towers of extensions containing arbitrarily many algebraic and transcendental steps.

Future Developments. We will allow registered users bidirectional access to the database. This will empower users to add data to the databases from within their client computer algebra systems. We also plan to provide QaoS clients for more computer algebra systems and support the OpenMath [OM] XML format for representing the algebraic objects developed in the SCIEnce project. Furthermore the tables of number fields will be extended and rechecked, and new tables of function fields and extensions of local fields will be created.

7 Graphical User Interface

Several general computer algebra systems, such as Maple [Ma05] and MuPad [Mu05], provide graphical interfaces. They offer typesetting and plotting, but they do not allow graphical manipulation of algebraic objects. For group theory the XGAP [CN04] package for GAP 4 [GAP] offers a tool for viewing and manipulating subgroup lattices and other structural information on UNIX systems running X Windows. It allows bidirectional communication between the interface and GAP. Working with elements of groups is not supported.

We introduce a graphical user interface for KASH called GiANT. GiANT differs from the preceding systems in that it offers direct, intuitive manipulation of algebraic structures, and it is specialized for working with number fields. In GiANT, number fields created by the user are incorporated into a tower of fields diagram that is updated in real time. The diagram is interactive and allows users to work with the elements, polynomials, and ideals of a number field simply by clicking on the number field's icon in the diagram. A specialized window containing the elements, polynomials, and ideals of the number field appears. All mathematical information is typeset using the same symbols and special characters that appear in modern math texts. The result is output which is easier to interpret than raw shell output.

Field elements, polynomials, and ideals can be manipulated by drag-and-drop operations, which may, for instance, reveal the minimal polynomial of an element, create the ideal generated by an element, or even move it to an extension field. In a similar way, the user can create relative field extensions by dropping irreducible polynomials from the ground field onto the field diagram.

GiANT is written in Java, but uses KASH to perform its computations. GiANT users may choose to work directly with KASH via a console. Any variables created with the graphical interface are also available in the console. The user may use the console to access features of KASH which are not graphically available. Alternatively, GiANT can be used as a KASH script generator, since all graphically-driven activities generate KASH code.

In the next step, we will provide a more flexible graphical user interface that, for example, displays lattices of subgroups using the same methods as for lattices of subfields. Furthermore, a bidirectional integration of graphical manipulation and a classic text based shell is needed, such that objects and structural information about the objects are displayed graphically as they are created in the shell.

For a more detailed description of GiANT see [KP06]. GiANT is released under the GNU General Public License (GPL) and available from

`http://giantsystem.sourceforge.net`.

This site also contains screen shots and videos of GiANT.

Acknowledgements

We would like to thank John Cannon for the long and fruitful collaboration with his Magma group. Special thanks to Allan Steel and Claus Fieker. Thanks to Martin Schönert for helping extend GAP 3 so that it would work with the KANT V4 C-library and Joachim Neubüser for making this possible.

Finally, a warm thanks to the following contributors to KANT and KASH. Michael Pohst (since 1987), Ulrich Schröter (1988–1992), Johannes von Schmettow (1989–1993), Bettina Arenz (1989–1992), Nicole Schröter (1990–1992), Max Jüntgen (1992–1996), Mario Daberkow (1992–1995), Klaus Wildanger (1993–1997), Martin Schörnig (1993–1996), Claus Fieker (since 1993), Jürgen Klüners (since 1994), Florian Heß (since 1995), Andreas Hoppe (1995–1997), Georg Baier (1996), Katharina Geissler (1996–2003), Carsten Friedrichs (1996–2000), Harald Bartel (1997–1999), Maike Henningsen (1999–2002), Hartmut Bauer (1998–2000), José Mendez (since 2000), Robert Fraatz (2002–2005), Marcus Wagner (since 2003), Lea Lieder (2005), Carolin Just (2005), and Osmanbey Uzunkol (since 2006).

References

AED05. B. Assmann, B. Eick, and A. Distler, *GAP package Alnuth: an interface to KANT*, `http://www.gap-system.org/Packages/alnuth.html`.

BB$^+$05. C. Batut, K. Belabas, D. Benardi, H. Cohen, and M. Olivier, *User's Guide to PARI-GP*, 2005, `http://pari.math.u-bordeaux.fr`.

BH96. Y. Bilu and G. Hanrot. *Solving Thue equations of high degree*, J. Number Th., 60:373–392, 1996.

BCP97. W. Bosma, J. J. Canon, and K. Playoust, *The Magma algebra system. I. The user language*, J. Symbolic Comput. 24 (1997), no. 3-4, 235–265, `http://magma.maths.usyd.edu.au`.

CN04. F. Celler, M. Neunhöffer, *GAP package XGAP: a graphical user interface for GAP*, 2004, `http://www-gap.mcs.st-and.ac.uk/Packages/xgap.html`.

DF$^+$97. M. Daberkow, C. Fieker, J. Klüners, M. Pohst, K. Roegner, M. Schörnig, and K. Wildanger, KANT V4, J. Symb. Comp. **11** (1997), 267–283

DW96. M. Daberkow and A. Weber, *A Database for Number Fields* in Jacques Calmet and Carla Limongelli (editors), *Design and Implementation of Symbolic Computation Systems: DISCO'96*, LNCS **1128**, Springer, 1996, 320–330

Fi97. C. Fieker, *Über relative Normgleichungen in algebraischen Zahlkörpern*, Dissertation, TU Berlin, 1997,

Fi02. C. Fieker, *Computing class fields via the Artin map*, Math. Comp. **70** (2001), no. 235, 1293–1303

GAP. *GAP: Groups, Algorithms, Programming*, `http://www.gap-system.org`.

GMP. *GMP: GNU Multiple Precision Arithmetic Library*, `http://www.swox.com/gmp`.

Ge05. K. Geissler, Berechnung von Galoisgruppen über Zahl- und Funktionenkörpern, Dissertation, TU Berlin, 2003.

He02. F. Hess, *Computing Riemann-Roch spaces in algebraic function fields and related topics*, J. Symbolic Comput. **33** (2002), no. 4, 425–445

HPP03. F. Hess, S. Pauli, and M.E. Pohst, *Computing the Multiplicative Group of Residue Class Rings*, Mathematics of Computation **72** (2003)

Hul05. A. Hulpke, *Constructing Transitive Permutation Groups*, J. Symb. Comp. **39** (2005), 1-30.

KP06. A. Karve and S. Pauli, *GiANT: Graphical Algebraic Number Theory*, preprint, 2006, `http://giantsystem.sourceforge.net`.

Ma05. Maplesoft, *Maple*, 2005, `http://www.maplesoft.com`.

Mu05. *MuPad: Multi Processing Algebra Data Tool*, `http://www.mupad.de`.

MPFR. *MPFR library for multiple precision floating point computation*, `http://www.mpfr.org`.

OM. *OpenMath: an extensible standard for representing the semantics of mathematical objects*, `http://www.openmath.org`.

Pa01. S. Pauli, *Factoring polynomials over local fields*, J. Symb. Comp. **32** (2001).

PSQL. *PostgreSQL*, `http://www.postgresql.org`.

Sc$^+$93. M. Schönert et. al. *GAP: Groups, Algorithms, Programming – version 3.27*, Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 1993, `http://www-gap.mcs.st-and.ac.uk/Gap3`.

St06. W. Stein et al, SAGE: Software for Algebra and Geometry Experimentation, 2006, `http://sage.scipy.org/sage`.

Wi00. K. Wildanger, *Über das Lösen von Einheiten- und Indexformgleichungen in algebraischen Zahlkörpern*, J. Number Theory **82** (2000), no. 2, 188–224.

Making Change and Finding Repfigits: Balancing a Knapsack

Daniel Lichtblau

Wolfram Research, Inc.
100 Trade Center Dr.
Champaign, IL 61820 USA
danl@wolfram.com

Abstract. We will discuss knapsack problems that arise in certain computational number theory settings. A common theme is that the search space for the standard real relaxation is large; in a sense this translates to a poor choice of variables. Lattice reduction methods have been developed in the past few years to improve handling of such problems. We show explicitly how they may be applied to computation of Frobenius instances, Keith numbers (also called "repfigits"), and as a first step in computation of Frobenius numbers.

Keywords: Frobenius instance solving, lattice reduction, integer linear programming, change-making problem, Frobenius numbers, Keith numbers, repfigits.

1 Introduction

Various problems in the realm of computational number theory have as key steps the solving of a linear equation or system over the integers, subject to some linear inequality constraints. The Frobenius instance problem (also known as the change-making problem) is a well-known example. The problem of finding what are called repfigits, to be described below, is another. These may be regarded as a class of knapsack problems wherein one is allowed to take only certain integer multiples of various items in forming a "valid" combination.. As such these fall into the category of integer linear programming (ILP).

Classical methods for solving such problems include branch-and-bound and cutting plane methods [8,16]. These approaches alone are often inadequate for certain classes of problems due to a phenomenon roughly described as "unbalanced bases". A method for dealing with this deficiency was developed in [1,2]. In essence it involves working with a basis that is reduced (in the sense of [11]) and ordered by size, in conjunction with standard branch-and-bound. We will describe and illustrate this for the types of problem mentioned above . We remark that the handling of Frobenius instances is by no means new, having been discussed in the aforementioned references. We also show how the method is applied in a new algorithm for finding Frobenius numbers, a task that is substantially harder than solving Frobenius instances.

© Springer-Verlag Berlin Heidelberg 2006

The algorithms described in this paper have been implemented in *Mathematica* [18]. Code is available from the author on request.

2 Frobenius Instances

Suppose we have a set of positive integers $A = (a_1, \ldots, a_n)$ and a target M, a positive integer. We seek nonnegative integer multipliers $X = (x_1, \ldots, x_n)$ such that $X \cdot A = M$. This is a standard problem in integer linear programming. A classical method [16] for solving this would be to solve the relaxed problem wherein we enforce all inequality constraints but all variables to be real rather than integer valued. If in the solution we encounter a variable a with value s that is not an integer then we spawn two subproblems where we enforce respectively that $a \leq \lfloor s \rfloor$ and $a \geq \lceil s \rceil$. We continue this process of solving relaxed subproblems, splitting when a variable has a noninteger value. It can be shown that eventually either we exhaust all possibilities or we obtain an integer valued solution [16]; in either case clearly the algorithm terminates.

A drawback to this approach is that the search space for the relaxed subproblems might appear to be "large", in the sense of having many points with not all coordinates integer valued. In particular it may be the case that a standard LP solver will find real valued solutions to the restricted subproblems without making rapid progress to an (entirely) integer valued solution, because it might be possible to subdivide the (real valued) solution polytope in such a way that integer points do not readily appear at corners.

The method of [1,2] was developed as a way to improve on this situation. Roughly it proceeds as follows. First we find a description of a solution set to our equations that is a priori integer valued but possibly does not satisfy the required inequalities. We arrange that this solution set has "good" basis vectors, such that when we solve relaxed problems with these we more rapidly walk through our (real valued) solution polytope. More correctly, the polytope is likely to intersect fewer hyperplanes orthogonal to larger direction vectors and spaced by integral multiples of those vectors.

With respect to the Frobenius instance problem it goes as follows. First we find a solution vector X over $\mathbb{Z}^n$ and a basis V for the integer null space (that is, $n-1$ independent vectors $V_j \in \mathbb{Z}^n$ with $V_j \cdot A = 0$; for this we use a method based on the Hermite normal form [5]. We use multiples of the basis vectors to find a "small" solution which we still call X. The tactic utilized to do this is sometimes called the embedding method. It apparently has been independently discovered several times; variants appear in [1,12,13,14]. In starting with a solution over $\mathbb{Z}^n$ rather than the nonnegatives $\mathbb{N}^n$ we are working with what is called an integer relaxation of the nonnegativity constraint. We will later enforce nonnegativity via the more common LP relaxations wherein integrality is not enforced.

We define variables $t = (t_1, \ldots, t_{n-1})$ so that solution vectors are given by $X + tV$. For purposes of finding a valid solution to the problem at hand we need to impose two requirements. The first is that all components are nonnegative and the second is that all are integers. The first can be met by standard linear programming. For the second, as in the classical approach, we will use branching on subproblems. Specifically we find solutions to relaxed LP problems wherein we now work over nonnegative reals. We then branch on noninteger values in those solutions. For example, if a solution has, say, $t_j = 5/4$, we create two subproblems identical to the one we just solved, but with the new constraints $t_j \leq \lfloor 5/4 \rfloor = 1$ and $t_j \geq \lceil 5/4 \rceil = 2$, respectively (though note that if other variables also had noninteger solutions then we need not have chosen this particular one). As observed above, this branching process will terminate eventually, with either a valid solution or the information that no such solution exists.

We explain again, in slightly different terms, why it is important to work with a small integer solution to the integer relaxation (that is, allowing negative values) and a lattice reduced basis for the null vectors. As our methodology is to take combinations of these null vectors, effectively they define directions in a solution polytope for the transformed problem. By working with a reduced basis we in effect change our coordinate system to one where the various search directions are roughly orthogonal. This helps us to avoid the possibility of taking many steps in similar directions in searching the polytope of nonnegative solutions for one that is integer valued. Thus we explore it far more efficiently. Moreover, in starting with a small solution we begin closer to the nonnegative orthant. Heuristically this seems to make the sought-for multipliers of the null vectors relatively small, and this is good for computational speed. This is discussed in section 2 of [1], with further explanation and illustrations found in [3].

A further efficiency, from [2], is to choose carefully the variable on which to branch. We order by increasing size the reduced lattice of null vectors. Branching will be done on the noninteger multiplier variable corresponding to the largest of these basis vectors. This has the effect of exploring the solution polytope in directions in which it is relatively thin, thus more quickly finding integer lattice points therein or exhausting the space. This refinement is important for handling pathological examples of the sort presented in [1] and [2].

As we have a constraint satisfaction problem we are also free to impose any linear integer objective function of our choosing. Thus an optimization is to obtain extremal values for linear forms with integer coefficients and use these results as simple cutting planes [16]. For example we can minimize or maximize the various coordinate values t_j, selecting one either in some specific order or at random for each subproblem. This process amounts to finding the width of the polytope along the directions of our lattice basis vectors, and enforcing integrality of the optimized variable helps to further restrict the search space.

As reported in [2] this method is very effective in solving Frobenius instances. We illustrate with an example from [17]

$$A = (10000000000, 10451674296, 18543816066, 27129592681, 27275963647, 29754323979, 31437595145, 34219677075, 36727009883, 43226644830, 47122613303, 57481379652, 73514433751, 74355454078, 78522678316, 86905143028, 89114826334, 91314621669, 92498011383, 93095723941)$$

We let the target be 862323776. In several seconds the instance solver returns the empty set. This has implications for bounding the Frobenius number of A; we will discuss that in a later section.

3 Keith Numbers

Keith numbers, also known as repfigits, were introduced in 1987 by Michael Keith [10] as a sort of computational novelty that relates a Fibonacci-like sequence to a linear equation involving its seed. They are defined as follows. Suppose we are given a number s of n digits (we work in base 10, but these can be defined with respect to arbitrary bases). We may form a sequence in Fibonacci style as follows. The first n elements are the digits themselves. The $(n+1)^{\text{th}}$ element is the sum of the first n digits. Subsequent elements are the sums of the preceding n elements. Then s is called a Keith number provided it appears in this sequence. As an example, the sequence for 197 is $\{1, 9, 7, 17, 33, 57, 107, 197, \ldots\}$ and so 197 is a Keith number. Keith originally referred to these as repfigits, for "replicating Fibonacci digits".

Keith numbers tend to be quite rare (there are only 71 of them below 10^{19}). Prior methods for finding them involved clever segmentation of an enumeration. While flawless (in the sense that they find all of them), these are limited in range due to algorithmic complexity and memory requirements. At the time the present work was begun the state of the art, from [10], was that all such numbers up to 19 digits had been found but no larger ones were known. We will take this substantially further.

To begin we must find equations to describe these things. If the digits are $\{d_0, d_1, \ldots, d_{n-1}\}$ then the number in question is $\sum_{j=0}^{n-1} d_j 10^{n-1-j}$. We form the appropriate sequence using a Fibonacci matrix of dimension n. This is simply a matrix that, when operating on a vector, replaces each element up to the last by its successor, and replaces the last by the sum of the elements. For example, for $n = 3$ it is $\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$.

If we multiply this matrix by itself $k-1$ times then the dot product of the bottom row with the digit sequence will give the $(n+k)^{\text{th}}$ term in the sequence. Some simple inequality considerations will give fairly tight bounds on how many such multiples can possibly work for a given number of digits n. We will use each possibility to form a homogeneous linear diophantine equation.

We demonstrate with a short example. We start by obtaining the set of candidate equation vectors for 5 digit examples. One of them is (−8207, 1705, 3069, 3395, 3524). That is, we will seek a solution to the system

$$-8207x_1 + 1705x_2 + 3069x_3 + 3395x_4 + 3524x_5 = 0$$

with each $x_j \in 0, 1, \ldots, 9$ and $x_1 \geq 1$.

As in the last section, the first step in the process of [1] and [2] is to find a full set of integer solutions to such a system. Since these are homogeneous equations we require only the integer null space. This can be obtained readily from the Hermite normal form for the matrix comprised of the vector for the homogeneous equation, augmented by an identity matrix. Again we want to work with vectors that are small and close to orthogonal so we apply lattice reduction to get a "good" set of vectors spanning the same solution set. We obtain $(-3,-1,-3,-3,-1)$, $(-2,-4,-3,3,-3)$, $(1,6,-5,6,-2)$, and $(7,-3,-15,5,26)$.

Notice that for any solution vector, its negative is also a solution vector. Looking at the first vector in our solution basis we thus see that 31331 is a Keith number of five digits. That was too easy; there is no guarantee we will have a solution vector with all components in the desired range. We look at a slightly larger example to see this. For six digits one candidate equation-defining vector is $(-96160, -4224, 5752, 7144, 7482, 7616)$. This time we get the following small null vectors: $(0,3,-3,0,4,0)$, $(-1,1,-3,-2,-4,-4)$, $(0,-6,-3,1,0,-2)$, $(0,-1,1,5,0,-6)$, and $(0,4,-12,15,-12,9)$. So we have a generating set of small null vectors none of which have entirely nonnegative or entirely nonpositive values (with the first being nonzero, in order that they give a legitimate six digit number). We now need a way to recombine these so that the first component is positive and the rest are nonnegative.

Again we have something we can tackle with standard branch-and-bound iterations [8,16]. Let $\{v_1,\ldots,v_n\}$ be our null space basis (here n is one less than the number of digits). We seek an integer vector of the form $a_1v_1+\ldots+a_nv_n$ such that all components lie in the range $\{0,1,\ldots,9\}$ with the first component strictly positive. For this we create variables $\{a_1,\ldots,a_n\}$ which will ultimately be required to take on integer values. In order to effect this we will solve relaxed linear programming problems with appropriate inequality constraints. As noted earlier, these are simply constraint satisfaction problems, so we can use arbitrary linear objective functions as a means of obtaining integer cuts cheaply. When some variables do not take on integer values in the solution we choose one such on which to branch and spawn a pair of subproblems. Our choice again is from [2]; we branch on that variable of noninteger solution value whose corresponding basis vector is largest.

To summarize, we first find the appropriate sets of integer equations. For each we find spanning sets of solutions that do not in general satisfy the digit inequality constraints. We lattice reduce these. We use ILP methods to find all possible solutions subject to the usual inequality constraints on digits.

We used this method to find all Keith numbers up through 29 digits. We show all repfigits between 20 and 29 digits below.

$\{1,2,7,6,3,3,1,4,4,7,9,4,6,1,3,8,4,2,7,9\}$
$\{2,7,8,4,7,6,5,2,5,7,7,9,0,5,7,9,3,4,1,3\}$
$\{4,5,4,1,9,2,6,6,4,1,4,4,9,5,6,0,1,9,0,3\}$
$\{8,5,5,1,9,1,3,2,4,3,3,0,8,0,2,3,9,7,9,8,9\}$
$\{7,6,5,7,2,3,0,8,8,2,2,5,9,5,4,8,7,2,3,5,9,3\}$
$\{2,6,8,4,2,9,9,4,4,2,2,6,3,7,1,1,2,5,2,3,3,3,7\}$
$\{3,6,8,9,9,2,7,7,5,9,3,8,5,2,6,0,9,9,9,7,4,0,3\}$
$\{6,1,3,3,3,8,5,3,6,0,2,1,2,9,8,1,9,1,8,9,6,6,8\}$

$\{2,2,9,1,4,6,4,1,3,1,3,6,5,8,5,5,5,8,4,6,1,2,2,7\}$
$\{9,8,3,8,6,7,8,6,8,7,9,1,5,1,9,8,5,9,9,2,0,0,6,0,4\}$
$\{1,8,3,5,4,9,7,2,5,8,5,2,2,5,3,5,8,0,6,7,7,1,8,2,6,6\}$
$\{1,9,8,7,6,2,3,4,9,2,6,4,5,7,2,8,8,5,1,1,9,4,7,9,4,5\}$
$\{9,8,9,3,8,1,9,1,2,1,4,2,2,0,7,1,8,0,5,0,3,0,1,3,1,2\}$
$\{1,5,3,6,6,9,3,5,4,4,5,5,4,8,2,5,6,0,9,8,7,1,7,8,3,4,2\}$
$\{1,5,4,6,7,7,8,8,1,4,0,1,0,0,7,7,9,9,9,7,4,5,6,4,3,3,6\}$
$\{1,3,3,1,1,8,4,1,1,1,7,4,0,5,9,6,8,8,3,9,1,0,4,5,9,5,5\}$
$\{1,5,4,1,4,0,2,7,5,4,2,8,3,3,9,9,4,9,8,9,9,9,2,2,6,5,0\}$
$\{2,9,5,7,6,8,2,3,7,3,6,1,2,9,1,7,0,8,6,4,5,2,2,7,4,7,4\}$
$\{9,5,6,6,3,3,7,2,0,4,6,4,1,1,4,5,1,5,8,9,0,3,1,8,4,1,0\}$
$\{9,8,8,2,4,2,3,1,0,3,9,3,8,6,0,3,9,0,0,6,6,9,1,1,4,1,4\}$
$\{9,4,9,3,9,7,6,8,4,0,3,9,0,2,6,5,8,6,8,5,2,2,0,6,7,2,0,0\}$
$\{4,1,7,9,6,2,0,5,7,6,5,1,4,7,4,2,6,9,7,4,7,0,4,7,9,1,5,2,8\}$
$\{7,0,2,6,7,3,7,5,5,1,0,2,0,7,8,8,5,2,4,2,2,1,8,8,3,7,4,0,4\}$

We remark that lattice methods alone can find sporadic large Keith numbers. One approach, from [15], improves the chances of getting a valid result from the lattice reduction step. The idea is to augment each null vector with a zero, and augment the lattice with a row consisting of some nonzero value (typically one) in the new column of zeros, and $\frac{9}{2}$ everywhere else. Thus if there is a valid solution then this augmented lattice contains the vector consisting of that nonzero value (or its negative) and the remaining entries in the range $\{-\frac{9}{2}, \frac{9}{2}\}$. As this would be a fairly "small" vector, one can hope that it will appear in the reduced basis (this is essentially the idea used by Schnorr and Euchner, in a binary setting, to raise the density at which one can typically solve subset sum problems). In practice we get a few Keith numbers this way as well as several more near misses.

It might be effective to combine this different lattice formulation with the branch-and-bound regimen described above. This is not entirely trivial as that required that the vectors span a solution space for a homogeneous equation, whereas the vectors in this modified lattice need not satisfy that equation.

Observe that the Keith numbers beginning at 24 digits but smaller than 29 digits all have leading digit in the set $\{1,2,9\}$. One might well wonder if there is a deep reason for this, and whether the trend returns after 29 digits. Also all known Keith numbers from 25 digits onward have a final digit that is either even or 5, and hence they cannot be prime. Again, one might wonder whether this trend continues, and, if so, whether there is an interesting reason behind it.

We will say a bit about the practical complexity of the method we have described for solving ILPs. While in principle branching can have very bad performance, in practice we find that the complexity scales reasonably well with problem size. Although we cannot claim that the methods described in this paper are polynomial time even in fixed dimension, in practice they do seem to scale that way. The behavior with respect to dimension is of course not so nice. For Keith number computations we find that, on average, the time spent for handling $n+1$ digits is roughly twice the time needed for n digits. Considering that each additional digit multiplies the search space by a factor of 10 this is still not so bad. To give an indication of computational speed, our implementation was able to handle 29 digits in around three days on a 3.0 GHz machine. We emphasize that this is but a crude measure both of complexity and actual performance, as various sorts of optimizations could have a significant impact on each. For example, preliminary experiments with the software in [6] indicate room for improvement in the handling of the ILP solving after the lattice reduction phase.

4 Frobenius Numbers

We are given a set $A = (a_1, \ldots, a_n)$ of positive integers with $gcd(A) = 1$. For later purposes we assume that the set is in ascending order. It can be shown that there are at most finitely many numbers not representable as a nonnegative integer combination of elements in A. The largest such nonrepresentable is called the Frobenius number of the set. The "Frobenius number problem" is to find it. Generally speaking this tends to be a different and usually harder problem than the Frobenius instance solving discussed earlier. We will give a much abbreviated discussion here in order to indicate how the sort of knapsack solving under discussion plays a role in computation of these numbers. References may be found in [4] and [17].

It turns out that this is trivial for $n = 2$ (this was shown by Sylvester in the late 1800's, even before the problem was popularized by Frobenius early in the twentieth century). Moreover in the 1980's some very good methods appeared for the case $n = 3$. For larger n, if one orders the elements by increasing size and restricts a_1 to be less than around 10^7 then there are effective algorithms to find the Frobenius number for A. This is roughly independent of the size of n; they can handle $n = 100$, for example. Our interest is in handling the case where a_1, the smallest element in A, is large (say, up to 10^{100}). As the problem is known to be intrinsically difficult we cannot hope to have both a_1 and n large. Hence we limit the latter to 10 or so.

In [17] one finds a definition of a "fundamental domain" which is a generalization that subsumes both lattice diagrams in earlier literature and a graph description from [4]. Similar ideas, expressed in the terminology of Minimal Distance Diagrams, appear in [9]. The Frobenius number will be determined by the furthest corner from the origin, in a suitably weighted l_1 norm. As we will see, an important domain feature is what we term "elbows". We give a quick idea of what is this fundamental domain for our set $A = (a_1, \ldots, a_n)$.

We start with the lattice of integer combinations of $\{a_2, \ldots, a_n\}$ that are zero modulo a_1. This is a full dimensional lattice in $\mathbb{Z}^{n-1}$. The set of residues of $\mathbb{Z}^{n-1}$ modulo this lattice gives rise to the fundamental domain, which lives in a space of dimension one less than the size of our set. It is not hard to see that there are a_1 distinct residue classes, so we know the cardinality of this domain. We now define the "weight" of a vector $v \in \mathbb{Z}^{n-1}$ as $v \cdot (a_2, \ldots, a_n)$. It can be shown that every residue class has at least one element with all nonnegative entries. From those we choose one of minimal weight. In case of a tie we choose the one that is lexicographically last. This uniquely defines the set of residues that we take to comprise the fundamental domain.

This domain has several interesting properties. (1) It is a staircase: if it contains a lattice element then it contains all nonnegative vectors with any coordinate strictly smaller. (2) It tiles $\mathbb{Z}^{n-1}$. (3) It is a cyclic group $\mathbb{Z}/a_1\mathbb{Z}$. (4) It can be given a circulant graph structure. It is this structure that was utilized in various shortest-path graph methods. Old and new approaches using such methods are discussed at length in [4]. In contrast, the method put forth in [17] primarily makes use of the staircase structure.

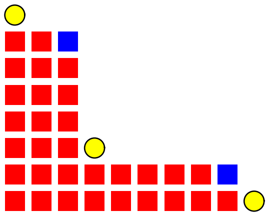
From the staircase property the fundamental domain has turning points we refer to as elbows. It has extremal points called corners. Specifically, a corner is a point c in the domain, such that $c + e_j$ is not in the domain, where e_j is the j^{th} coordinate vector. An elbow is a point x that is not in the domain, but is such that, for each j, either $x_j = 0$ or $x - e_j$ is in the domain. An elbow with all but one coordinate zero is called an "axial" elbow. It indicates how far one can go along a given axis and still remain inside the domain. There are two other definitions that play a role in the algorithm. We will not descibe them too carefully but, roughly, there are as follows.

(1) Protoelbows. These have both positive and negative coordinates and correspond to certain "minimal" equivalences (that is, reducing relations) in the lattice.

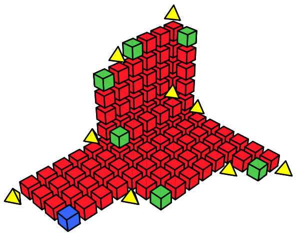
(2) Preelbows. These are the positive parts of the protoelbows. Elbows are minimal elements in the partially ordered (ascending by inclusion) set of preelbows.

With respect to these domains the Frobenius number corresponds to the farthest corner from the origin where distance is an l_1 metric weighted by element sizes. In brief one sees this as follows. Recall that each element in the domain corresponds to a residue modulo a_1 that satisfies a minimal nonnegativity property. Thus values in the same residue class but of smaller weight cannot be attained using nonnegative combinations of elements of A, whereas values of equal or larger weight are attained as such combinations. We conclude that the largest nonattainable value for the residue class of the element $X = (x_2, \ldots, x_n)$ is $a_2x_2 + \ldots + a_nx_n - a_1$. From this and the staircase property of the fundamental domain it is clear that the largest nonattainable value overall is a_1 less than the largest weight of a corner element.

Below are pictures (provided courtesy of Stan Wagon) of fundamental domains corresponding to $n = 3$ and $n = 4$ respectively (recall that the fundamental domain lives in a space of dimension one less than n). In the planar diagram the elbows are the lattice points on the axes that bound the diagram, and the lattice point in the interior just outside the "ell". The corners are the two extremal points reached by intersecting vertical and horizontal lines through the elbows. This picture tells the entire story as regards the $n = 3$ case because it can be shown that there is at most one interior elbow and two corners, and finding them is easy. The three elbows (two axial, one internal) are denoted by circles.



In the three dimensional diagram the elbows are again the axial bounding lattice points as well as bounding points where the staircase goes up in the coordinate planes and in the interior. They are demarcated by tetrahedra.



With this brief background we now say a bit about how knapsack solving can play a role in the computation of Frobenius numbers. First, it turns out that axial elbows are defined by an integer programming problem (see [17]) which, in complexity if not details of definition, is similar to the Frobenius instance problem. From the axial elbows we immediately get good lower and upper bounds on the Frobenius number (each elbow less one is in the domain, and the farthest corner is bounded by the vector with all components given by corresponding axial elbows less one). More importantly for our purposes is that they give a search space from which to find all elbows. They are particular integer points in a polyhedron that satisfy certain inequality conditions. Full details, including a method for finding them, are provided in [17]. From the elbows one can find all corners and in particular the one that gives the Frobenius number of the set.

Another tactic presented in [17] makes direct use of Frobenius instance solving to find axial elbows. One uses a bisection approach, working down from an a priori bound on the axial elbow values. The goal is to find the smallest value for which a certain set of Frobenius instances have no solution.

Still another point of overlap between Frobenius instances and numbers is the obvious fact that whenever a Frobenius instance solver returns an empty solution we automatically have a lower bound on the Frobenius number. One can test random values that are, say, an order of magnitude below the heuristic approximation for the Frobenius number presented in [4]. If any such test gives no solution we thereby establish a lower bound that is often better than a priori bounds to be found in the literature.

There is also a heuristic method in [17] for more efficiently "guessing" the likely Frobenius number from a restricted set of elbows. It gives an a priori upper bound, and a single Frobenius instance invocation can then verify whether it is in fact the actual value. In random examples this appears always to be the case.

We sketched above how efficient ILP knapsack solving, of the sort used for Frobenius instances, also may be applied to the (generally much harder) problem of finding Frobenius numbers. From the fundamental domain pictures one realizes a possible alternative approach. Working an axis at a time, a branch-and-bound strategy might be directly applied to get to extremal vertices (corners) in the domain. Thus we could have a bilevel branching algorithm, with the outer level iterating over these extrema, and the inner one using relaxed LPs to solve the ILPs needed to move to new vertices. This might become an alternative to the algorithm described in [17].

We remark that there is a connection between another knapsack solving technique and the problem of computing Frobenius numbers. It is well known that integer programming e.g. for knapsack problems can be done with toric Gröbner bases [7]. What is not so obvious is that they may also be used to deduce the stairway structure of the fundamental domain. An algorithm for this purpose is presented in [17]. The basic idea is to formulate a term ordering so that the staircase structure of the fundamental domain is captured by the staircase of the Gröbner basis lead monomials. This has the added virtue of finding all elbows at once, so no elaborate method is needed to search a bounding box defined by axial elbows. An implementation by the author has handled Frobenius number problems involving as many as 7 numbers of 40 digits. While it is not competitive with the main approach in [17] (which has handled sets of up to 11 numbers), Frobenius number problems of this size are apparently larger than what can be handled by other methods from the published literature.

Another link between Frobenius numbers and methods from ideal theory appears implicitly in [9] as well as other literature concerning what are called "multiple-loop networks". There, as in [4], one works with a circulant directed graph based on residues modulo a positive integer. Now, however, the modulus is the largest rather than smallest element of the given set. While the authors did not explicitly consider the connection to Frobenius numbers, some of the ideas are quite similar. In particular the maximum diameter of this graph, which is similar to the Frobenius number (though using an unweighted metric), plays an important role in their work. They discuss several aspects of the related domain (actually family of domains, as they do not impose uniqueness conditions) in terms of monomial ideals. They define a generalized ell shape and prove their domains are always of such a shape; this corresponds to the uniqueness of the interior elbow as shown in [17]. It would be interesting to understand better how their ideas, in particular regarding use of monomial ideals, relate to the toric ideal construction of the fundamental domain given in [17].

5 Summary

We have investigated several examples of integer linear programs with the common feature that a straightforward branch-and-bound approach, working with real relaxations, will tend to bog down in searching a large polytope. We utilize a method based on solving of an integer relaxation to the set of equality constraints. We reformulate the problem as one of adding combinations of null vectors to a specific solution. The vectors in question tend to be well suited to the problem at hand because we use lattice reduction to make them close to orthogonal. We then enforce inequality constraints via branch-and-bound on real relaxations of the new problem. We use a branching choice that tends to make the polytope thin in the search direction and thus helps to exhaust it efficiently.

We reviewed this approach as it was applied to the change-making problem [1,2]. We then used it to find all Keith numbers through 29 digits; previous methods had gone only through 19 digits. This brought us to a range where we could observe curious patterns in leading and trailing digits, something not present in the smaller Keith numbers. We also gave a brief idea of how this method is used in a new algorithm to compute Frobenius numbers.

A further direction would be to incorporate effective cutting planes. The code we used only attempts a very naive sort of cut (by using random coordinate variables as objective function). Preliminary experiments with an external library [6] indicate that more serious cutting plane efforts can give substantial speed improvement; we have seen ILP examples that improve by an order of magnitude. We emphasize that this still requires preprocessing with lattice reduction in the manner described in this paper.

Acknowledgments. I thank Stan Wagon for providing the diagrams of fundamental domains and for his careful reading and many suggestions that improved the exposition. I thank Victor Moll for inviting me to attend the 2005 Clifford Lectures conference at Tulane, where I presented an earlier version of this work. I thank the two anonymous referees for their several suggestions which improved readability, and the second referee in particular for pointing out the relevance of [9] and related literature on multiple-loop networks.

References

1. K. Aardal, C. A. J. Hurkens, and A. K. Lenstra. Solving a system of linear diophantine equations with lower and upper bounds on the variables. *Mathematics of Operations Research*, 25(3):427–442, 2000.
2. K. Aardal and A. K. Lenstra. Hard equality constrained integer knapsacks. In *Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization (IPCO 2002)*, volume 2337 of *Lecture Notes in Computer Science*, pages 350–366. Springer-Verlag, 2002.
3. K. Aardal, R. Weismantel, and L. A. Wolsey. Non-standard approaches to integer programming. *Discrete Appl. Math.*, 123(1-3):5–74, 2002.
4. D. Beihoffer, J. Hendry, A. Nijenhuis, and S. Wagon. Faster algorithms for frobenius numbers. *Elec. J. Combinatorics*, 12, 2005.
5. W. Blankenship. Algorithm 288: Solution of simultaneous linear diophantine equations. *Communications of the ACM*, 9(7):514, 1966.

6. COmputational INfrastructure for Operations Research (COIN-OR). http://www.coin-or.org/documentation.html.
7. P. Conti and C. Traverso. Buchberger algorithm and integer programming. In *AAECC-9: Proceedings of the 9th International Symposium on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 539 of *Lecture Notes in Computer Science*, pages 130–139. Springer-Verlag, 1991.
8. G. B. Dantzig. *Linear Programming - 1: Introduction.* Princeton Landmarks in Mathematics and Physics. Princeton University Press, Princeton, NJ, USA, 1963.
9. D. Gómez-Perez, J. Gutierrez, and Á. Ibeas. Circulant digraphs and monomial ideals. In *Proceedings of the 8th International Workshop on Computer Algebra in Scientific Computing (CASC 2005)*, volume 3718 of *Lecture Notes in Computer Science*, pages 196–207, 2005.
10. M. Keith. Determination of all keith numbers up to 10^{19}. Available from `http://users.aol.com/s6sj7gt/keithnum.htm`, 1998.
11. A. Lenstra, H. L. Jr., and L. Lovasz. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.
12. D. Lichtblau. Revisiting strong Grbner bases over Euclidean domains. Manuscript, 2003.
13. K. Matthews. Short solutions of a x=b using a lll-based hermite normal form algorithm. Manuscript, 2001.
14. P. Nguyen. Cryptanalysis of the goldreich-goldwasser-halevi cryptosystem from crypto '97. In *Proceedings of Advances in Cryptology - CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 288–304. Springer-Verlag, 1999.
15. C.-P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. In *Proceedings of the 8th International Conference on Fundamentals of Computation Theory*, volume 529 of *Lecture Notes in Computer Science*, pages 68–85. Springer-Verlag, 1991.
16. A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, Inc., New York, NY, USA, 1986.
17. S. Wagon, D. Einstein, D. Lichtblau, and A. Strzebonski. Frobenius numbers by lattice enumeration. Submitted, 2006.
18. S. Wolfram. *The Mathematica Book.* Wolfram Media, Champaign, IL, USA, 5th edition, 2003.

Robust HGCD with No Backup Steps

Niels Möller

KTH, School of Electrical Engineering
SE-100 44 Stockholm, Sweden
nisse@lysator.liu.se

Abstract. Subquadratic divide-and-conquer algorithms for computing the greatest common divisor have been studied for a couple of decades. The integer case has been notoriously difficult, with the need for "backup steps" in various forms. This paper explains why backup steps are necessary for algorithms based directly on the quotient sequence, and proposes a robustness criterion that can be used to construct a "half-gcd" algorithm without any backup steps.

1 Introduction

Euclid's algorithm for computation of the greatest common divisor is one of the oldest algorithms known. For inputs of size n, this algorithm runs in time $O(n^2)$, and since the total size of the remainders is $O(n^2)$, no algorithm which computes all the remainders can run asymptotically faster than this.

Lehmer's algorithm from 1938 cuts the running time of Euclid's algorithm by a constant factor [1]. A key to the first subquadratic algorithms by Knuth [2] and Schönhage [3], discovered in 1971, is that unlike the remainder sequence, the quotient sequence is of linear size, and it can be computed in subquadratic time. These algorithms are intimately tied to the quotient sequence and to the continued fractions expansion of a corresponding rational number.

Schönhage's 1971 algorithm is straightforward to apply to polynomial gcd, but, to quote [4]: "The integer HGCD algorithm turns out to be rather intricate". One of the main problems with the quotient-based gcd algorithms is that the recursive calls sometimes compute one or a few quotients too much, and the algorithm must check for this case and undo the corresponding divisions. I refer to these corrections as "backup steps". Both analysis and actual implementation is quite difficult and error prone. For the same reasons, the algorithm is seldom spelled out in detail in textbooks, and when it is, it has been plagued by errors. One variant of the algorithm, for both integers and polynomials, is described in [4], another one is tersely described in [5], while the author's implementation is described in [6].

The focus of the present paper is on the integer gcd, and on algorithms that work from the most significant end of the input. There is also a family of "binary" algorithms that work from the least significant end, from the classic binary gcd algorithm first published by Stein [7] in 1961, via improved quadratic algorithms by Sorenson [8] and Weber [9], to the subquadratic "binary recursive" algorithm

© Springer-Verlag Berlin Heidelberg 2006

by Stehlé and Zimmermann [10]. There are also subquadratic gcd algorithms for other algebraic contexts, such as Weilert's algorithm for gcd computations on Gaussian integers [11], and algorithms for computing the continued fraction expansion of algebraic numbers [12].

By formulating the stop and correctness conditions for the HGCD-function (which is the main work horse for subquadratic gcd) in terms that are not based on the quotient sequence, it is possible to construct gcd algorithms without backup steps, which yields a major improvement to algorithm complexity. The first publication of such a method (which however does not use the familiar HGCD form) that the author is aware of is in Weilert's diplomarbeit [13], where it is credited to an unpublished manuscript by Schönhage. Essentially the same algorithm has been rediscovered by Lucier Bradley [14], with inspiration from Schönhage's algorithm for reduction of binary quadratic forms [15], which uses similar ideas. See [6] for a comparison of the running time and implementation complexity of several subquadratic gcd algorithms.

The main contribution of this paper is to explain the need for "backup steps" as a lack of robustness in the quotient sequence, and the definition of a new robustness criterion, which aids the design of gcd algorithms without backup steps.

The rest of this paper is organized as follows. Sec. 2 explains the notation used, and describes the bounds that relate the sizes of the quantities in the algorithm. Sec. 3 describes the general structure of the HGCD algorithms. Sec. 4 and 5 examine the robustness, and lack thereof, of the quotient sequence, and Sec. 6 formulates a new criterion for robustness of a reduction matrix. Sec. 7 describes two HGCD algorithms which are based on the robustness criterion, and which don't need any backup steps.

2 Background and Notation

We work with vectors and matrices with elements that are integers, and usually non-negative. For compactness, column vectors are sometimes written as $(x_1; x_2)$ and 2×2-matrices as $(a_{11}, a_{12}; a_{21}, a_{22})$.

Sizes of numbers will be important, so we introduce the notation $\#x$ for the bit size of x. Define $\#x = \lceil \log_2(1 + |x|) \rceil$; then for $x \neq 0$, $\#x = s$ means that $2^{s-1} \leq |x| < 2^s$. When $\#$ is applied to a vector or matrix, it denotes the *maximum* bit size of the elements. Occasionally, we also need the minimum bit size, for which we use the notation $\underline{\#}(x, y) = \min(\#x, \#y)$.

The HGCD algorithms work with reductions of the form

$$\begin{pmatrix} a \\ b \end{pmatrix} = M \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{1}$$

where a, b, α, and β are positive integers, and $M = (u, u'; v, v')$ is a matrix of non-negative integer elements, such that $\det M = 1$. Then $\gcd(a, b) = \gcd(\alpha, \beta)$.

The non-negativity of all involved numbers imply the following bounds

$$|M|_\infty \min(\alpha, \beta) \leq \max(a, b) \leq |M|_\infty \max(\alpha, \beta) \tag{2}$$

```
HGCD($A, B$)
 1  $n \leftarrow \#(A, B)$
 2  Select $p_1 \approx n/2$
 3  $(\alpha, \beta, M_1) \leftarrow$ HGCD($\lfloor 2^{-p_1} A \rfloor, \lfloor 2^{-p_1} B \rfloor$)
 4  $(A, B) \leftarrow M_1^{-1}(A; B) = 2^{p_1}(\alpha; \beta) + M_1^{-1}[(A; B) \bmod 2^{p_1}]$
 5  Perform a small number of divisions or backup steps.
        $\triangleright$ $A$, $B$ are now of size $\approx 3n/4$
 6  Select $p_2 \approx n/4$
 7  $(\alpha, \beta, M_2) \leftarrow$ HGCD($\lfloor 2^{-p_2} A \rfloor, \lfloor 2^{-p_2} B \rfloor$)
 8  $(A, B) \leftarrow M_2^{-1}(A; B) = 2^{p_2}(\alpha; \beta) + M_2^{-1}[(A; B) \bmod 2^{p_2}]$
 9  Perform a small number of divisions or backup steps.
        $\triangleright$ $A$, $B$ are now of size $\approx n/2$
10  $M \leftarrow M_1 \cdot M_2$
11  Return $A, B, M$
```

Fig. 1. General structure of subquadratic HGCD

where $|M|_\infty = \max(u + u', v + v')$. Translated into bit sizes, we get

$$\#M \geq \#|M|_\infty - 1 \geq \#(a, b) - \#(\alpha, \beta) - 1 \tag{3}$$

$$\#M \leq \#(a, b) - \underline{\#}(\alpha, \beta) + 1 \tag{4}$$

3 General HGCD

Qualitatively speaking, the "half gcd" function, HGCD, takes a pair of integers $(a; b)$ as input, and produces a reduction matrix M and corresponding reduced numbers $(\alpha; \beta)$ which are all of roughly half the size of the inputs. Given an efficient HGCD function, gcd can be implemented efficiently by calling HGCD $O(\log n)$ times until the numbers are reduced to a comfortable size, after which a simpler quadratic gcd algorithm is used.

The key idea of subquadratic HGCD is that examining half of the input is sufficient for roughly half of the work. This leads naturally to a divide-and-conquer structure, illustrated in Fig. 1. All gcd alorithms using this structure have a running time of $O(n\,(\log n)^2 \log\log n)$, i.e., a factor $\log n$ slower than multiplication.

A less obvious benefit of an algorithm without backup steps, is that one gets the option of letting the algorithm return only M, and not compute or return the corresponding reduced numbers. Then Steps 8 and 9 above can be omitted.

At first look, it seems that the additional work that is required in Step 4 makes this a useless optimization; but careful study reveals that the availability of α and β in Step 4 does not save much work, at least not in the range where FFT-multiplication is used [16]. Omitting Step 9 also means that the returned reduction matrix will be slightly smaller, but the effect on the running time should be very small.

4 Quotient Sequence and Jebelean's Criterion

The first subquadratic gcd algorithms [2,3] actually compute the continued fractions expansion of a rational number. Hence, they work with the quotient sequence defined by Euclid's algorithm. It is then natural to formulate the requirements on the HGCD function as follows:

The input are two integers $a > b > 0$. The outputs are two remainders r_k and r_{k+1} and a matrix

$$M = \begin{pmatrix} q_1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} q_2 & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} q_k & 1 \\ 1 & 0 \end{pmatrix} \tag{5}$$

built up from the first k quotients in the quotient sequence for $(a;b)$. Then the remainders satisfy

$$\begin{pmatrix} a \\ b \end{pmatrix} = M \begin{pmatrix} r_k \\ r_{k+1} \end{pmatrix} \tag{6}$$

and $r_k > r_{k+1} \geq 0$. Furthermore, we require that $q_1, \dots, q_k$ are the initial quotients not only for $(a;b)$, but also for any two numbers with most significant parts a and b, i.e., $(A;B) = 2^p(a;b) + (A';B')$ where p is an arbitrary positive integer, and $0 \leq A', B' < 2^p$. Jebelean's criterion [17] gives a simple and precise condition for when this is the case. Here, we state the criterion only for the case of even k, and hence $\det M = 1$; the case of odd k is similar.

Theorem 1. *Let $a > b > 0$, with the kth remainders r_k and r_{k+1} and a corresponding matrix $M = (u, u'; v, v')$, with*

$$\begin{pmatrix} a \\ b \end{pmatrix} = M \begin{pmatrix} r_k \\ r_{k+1} \end{pmatrix} \tag{7}$$

Let $p > 0$ be an arbitrary positive integer, $0 \leq A', B' < 2^p$, and define

$$\begin{pmatrix} A \\ B \end{pmatrix} = 2^p \begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} A' \\ B' \end{pmatrix} \tag{8}$$

$$\begin{pmatrix} R_k \\ R_{k+1} \end{pmatrix} = M^{-1} \begin{pmatrix} A \\ B \end{pmatrix} = 2^p \begin{pmatrix} r_k \\ r_{k+1} \end{pmatrix} + M^{-1} \begin{pmatrix} A' \\ B' \end{pmatrix} \tag{9}$$

For even k, the following two statements are equivalent:

i. $r_{k+1} \geq v$ *and* $r_k - r_{k+1} \geq u + u'$
ii. *For any p and any A', B', the kth remainders of A and B are R_k and R_{k+1}.*

5 Quotient Sequence Is Not Robust

It is tempting to define HGCD based on quotient sequences and Jebelean's criterion. We then require that HGCD produces a reduction that corresponds to a prefix of the quotient sequence of the input, and which satisfies Jebelean's criterion. Preferably, the quotient sequence should also be as long as possible.

This approach leads to a HGCD where backup steps are essential for correctness. Consider the following example.

Assume that the input to the HGCD algorithm is $a = 858824$ and $b = 528747$, and that after Step 5 they have been reduced as

$$\begin{pmatrix} a \\ b \end{pmatrix} = M_1 \begin{pmatrix} C \\ D \end{pmatrix} \tag{10}$$

with $C = 64144$, $D = 3119$ and $M_1 = (13, 8; 8, 5)$, corresponding to quotients $q_1 = q_2 = \cdots q_6 = 1$. Now, these numbers are split as

$$\begin{pmatrix} C \\ D \end{pmatrix} = 16 \begin{pmatrix} 4009 \\ 194 \end{pmatrix} + \begin{pmatrix} 0 \\ 15 \end{pmatrix} \tag{11}$$

where the most significant halves are $c = 4009$ and $d = 194$. Now, the second recursive call, HGCD(c, d), produces the reduction

$$\begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} 21 & 20 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 129 \\ 65 \end{pmatrix} \tag{12}$$

This reduction of $(c; d)$ satisfies Jebelean's criterion. It corresponds to the two additional quotients $q_7 = 20$ and $q_8 = 1$, and these are also valid for (C, D). Form the full matrix

$$M = M_1 M_2 = \begin{pmatrix} 281 & 268 \\ 173 & 165 \end{pmatrix} \tag{13}$$

If we return this matrix as the value of HGCD(a, b), that corresponds to the reduction

$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 281 & 268 \\ 173 & 165 \end{pmatrix} \begin{pmatrix} 1764 \\ 1355 \end{pmatrix} \tag{14}$$

and the quotient sequence $1, 1, 1, 1, 1, 1, 1, 20, 1$. However, this reduction does *not* satisfy Jebelean's criterion, since $r_k - r_{k+1} = 409 < 549 = u + u'$.

If we form larger numbers with most significant bits a and b, we can find numbers such as

$$\begin{pmatrix} A \\ B \end{pmatrix} = 8 \begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} 1 \\ 7 \end{pmatrix} = \begin{pmatrix} 6870599 \\ 4229977 \end{pmatrix} \tag{15}$$

whose quotient sequence starts with $1, 1, 1, 1, 1, 1, 1, 20, 2$. To ensure correctness, the algorithm must check for this case, and sometimes discard the final quotient before returning.

In this example, the correct values for HGCD(a, b) to return are $\alpha = 3119$, $\beta = 1764$, and $M = (268, 13; 165, 8)$.

6 A Robustness Condition

As illustrated in the previous section, it is crucial that HGCD produces a reduction that is valid also when the input is disturbed. To make this requirement more precise, consider two input numbers a and b and a candidate matrix M of non-negative elements with $\det M = 1$. We then require that

$$M^{-1}\left\{\begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} x \\ y \end{pmatrix}\right\} > 0 \tag{16}$$

for all "small" disturbances x and y. It seems natural to consider disturbances in the interval $-1/2 \leq x, y < 1/2$ (for rounded inputs) or $0 \leq x, y < 1$ (for truncated inputs), but this is not sufficient and leads to similar problems as with Jebelean's criterion. But it is sufficient to widen these intervals by a factor two. In order to handle both truncated and rounded inputs, we allow the following set of disturbances:

$$S = \{(x; y) \in \mathbb{R}^2, |x| < 2, |y| < 2, |x - y| < 2\} \tag{17}$$

Note that $(x; y) \in S$ if and only if there exists an open interval of length 2 which contains the three points $\{0, x, y\}$, and in particular, S contains the open square defined by $-1/2 < x, y < 3/2$.

Theorem 2. *Assume that a, b, α and β are positive integers, such that*

$$\begin{pmatrix} a \\ b \end{pmatrix} = M \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{18}$$

for some matrix M with non-negative integer elements and unit determinant. Then the following two statements are equivalent:

i. $M^{-1}\{(a; b) + (x; y)\} > 0$ *for all* $(x; y) \in S$.
ii. $\alpha \geq 2\max(u', v')$ *and* $\beta \geq 2\max(u, v)$.

Proof. First assume that (i) holds. Then

$$\alpha > -v'x + u'y \tag{19}$$

for all $(x; y) \in S$. The point $(-2, 0)$ lies on the boundary of S, hence by continuity $\alpha \geq 2v'$. Similarly, $\alpha \geq 2u'$, $\beta \geq 2u$ and $\beta \geq 2v$, which shows that (ii) holds.

Conversely, assume that (ii) holds, and let $(x; y) \in S$ be arbitrary. By the definition of S, we can find a c such that $|c| < 1$ and $c - 1 < x, y < c + 1$. Then

$$\begin{aligned} \alpha + xv' - yu' &> \alpha + (-1 + c)v' - (1 + c)u' = \alpha - [(1 - c)v' + (1 + c)u'] \\ &\geq \alpha - 2\max(u', v') \geq 0 \end{aligned}$$

Similarly, $\beta - vx + uy \geq 0$, which concludes the proof by showing that (ii) implies (i). □

We can also derive two simpler sufficient conditions. It is clear that

$$\min(\alpha, \beta) \geq 2\max M \tag{20}$$

implies robustness, as defined above. It is useful, in particular for the base case, to have a sufficient condition based on bit sizes of the original and the reduced numbers. Put $n = \#(a, b)$. What is the smallest s such that $\underline{\#}(\alpha, \beta) > s$ is sufficient for robustness?

Combining the bound $\underline{\#}(\alpha, \beta) > s$ and (4), we see that the right hand side of (20) is of bit size at most $n - s + 1$. Then $\underline{\#}(\alpha, \beta) > s$ is a sufficient condition, provided that $n - s + 1 \leq s$, for which the smallest possible solution is $s = \lfloor n/2 \rfloor + 1$. This gives one more, even stricter, sufficient condition,

$$\underline{\#}(\alpha, \beta) > \left\lfloor \frac{n}{2} \right\rfloor + 1 \tag{21}$$

7 Robust HGCD Algorithms

This section describes two HGCD algorithm based on the robustness condition. We adopt the convention that HGCD returns only the matrix M, and not the corresponding reduced numbers. The objective is to return a reduction matrix $M = \text{HGCD}(A, B)$ which satisfies the robustness condition, and such that the reduced numbers, $(\alpha; \beta) = M^{-1}(A; B)$, have $|\alpha - \beta|$ small. For the inputs, we require that $\underline{\#}(A, B) > \lfloor n/2 \rfloor + 1$.

In principle, such an M can be computed by repeated subtraction, with the following algorithm.

```
HGCD-SIMPLE$(A, B)$
 1  $n \leftarrow \#(A, B)$
 2  $s \leftarrow \lfloor n/2 \rfloor + 1$
 3  $M = I$
 4  while $|A - B| \geq 2^s$
          do
 5            if $A > B$
                then
 6                  $A \leftarrow A - B$
 7                  $M \leftarrow M \cdot (1, 1; 0, 1)$
                else
 8                  $B \leftarrow B - A$
 9                  $M \leftarrow M \cdot (1, 0; 1, 1)$
10  return $M$
```

In practice, the base case, which handles all inputs below some threshold, should use an optimization similar Lehmer's algorithm to process one or two words at a time.

7.1 Strict Robustness

Enforcing the strictest of the robustness criteria, $\underline{\#}(\alpha, \beta) > s$, leads to the recursive algorithm in Fig. 2. Unlike the base case algorithm, it does not guarantee that $|\alpha - \beta| < 2^s$, but as we will see below, this difference will be at most a few bits larger.

```
HGCD($A, B$)
 1  $n \leftarrow \#(A, B)$
 2  $s \leftarrow \lfloor n/2 \rfloor + 1$
 3  $p_1 \leftarrow \lfloor n/2 \rfloor$
 4  if $\underline{\#}(A, B) > \lfloor (n + p_1)/2 \rfloor + 1$
 5      then
 6              $M_1 \leftarrow$ HGCD($\lfloor 2^{-p_1} A \rfloor, \lfloor 2^{-p_1} B \rfloor$)
 7      else  $M_1 \leftarrow I$
 8  $(C; D) \leftarrow M_1^{-1}(A; B)$
 9  Try a subtraction and division step on $(C; D)$.
    If that would result in $\underline{\#}(C, D) \leq s$, return $M_1$.
    Otherwise, update $M_1$ and $(C; D)$.
10  $p_2 \leftarrow 2s - \#(C, D) + 1$
11  if $\underline{\#}(C, D) > \lfloor (\#(C; D) + p_2)/2 \rfloor + 1$
12      then
13              $M_2 \leftarrow$ HGCD($\lfloor 2^{-p_2} C \rfloor, \lfloor 2^{-p_2} D \rfloor$)
14      else
15              $M_2 \leftarrow I$
16  $M \leftarrow M_1 M_2$
17  return $M$
```

Fig. 2. Robust HGCD, without backup steps

If the algorithm returns with $\#(\alpha - \beta) \leq s$, then the reduction is maximal. We assume that $\#(\alpha - \beta) \leq s + \epsilon$ for both recursive calls, i.e., the recursive calls are ϵ bits off from being maximal. The following lemma, which is a slight generalization of Lemma 6 in [6], explains how the algorithm works.

Lemma 1. *Let A and B be two given numbers of size $N = \#(A, B)$, and let $0 < p < N$. Partition A and B into the least significant p bits A' and B' and the $n = N - p$ most significant bits a and b, so that $A = 2^p a + A'$ and $B = 2^p b + B'$. Put $s = \lfloor n/2 \rfloor + 1$. and assume that $\underline{\#}(a, b) > s$. Let*

$$\begin{pmatrix} a \\ b \end{pmatrix} = M \begin{pmatrix} c \\ d \end{pmatrix} \tag{22}$$

with $\underline{\#}(c, d) > s$ and $\#(c - d) \leq s + \epsilon$. Form

$$\begin{pmatrix} C \\ D \end{pmatrix} = M^{-1} \begin{pmatrix} A \\ B \end{pmatrix} = 2^p \begin{pmatrix} a \\ b \end{pmatrix} + M^{-1} \begin{pmatrix} A' \\ B' \end{pmatrix} \tag{23}$$

Then $\underline{\#}(C, D) > p + s - 1$ and $\#(C - D) \leq p + s + \epsilon + 1$.

Proof. We have $\#M \leq n - s$. Note that $s > n/2 > n - s$. Let $M^{-1} = (v', -u'; -v, u)$. Then

$$\begin{aligned} C &= 2^p c + v' A' - u' B' \\ &> 2^p 2^s - 2^{n-s} 2^p \geq 2^p 2^{s-1} = 2^{p+s-1} \end{aligned}$$

which shows that $\#C > p + s - 1$ as claimed, and the same holds for D. Next,

$$\begin{aligned}|C - D| &= |2^p(c - d) + (v + v')A' - (u + u')B')| \\ &< 2^p 2^{s+\epsilon} + 2^{n-s}2^p \le 2^{p+s+\epsilon+1}\end{aligned}$$

□

With this lemma, we can derive the following facts about the algorithm.

– The input size for the first recursive calls is $\lceil n/2 \rceil$.
– Before the second recursive call, $\#(C, D) \le \lfloor 3n/4 \rfloor + \epsilon + 1$.
– The input size for the second recursive call is bounded by $\lfloor n/2 \rfloor + 2\epsilon + 2$.
– The final matrix corresponds to a reduction with $\#(\alpha - \beta) \le s + \epsilon + 2$.

If so desired, it is easy to modify the algorithm to always return with $\#(\alpha - \beta) \le s$, by computing α and β and performing a small number of extra reductions. See [6] for details on this variant of the algorithm.

This algorithm shares important features with the “controlled Euclidean descent” described in [13], although the organization as a “half-gcd” function is quite different.

7.2 Direct Robustness

The condition (21) is sufficient for robustness, but not necessary. It is a simple and convenient condition to use for the base case, but for the recursive recursive HGCD algorithm, this requirement can be dropped, and replaced by the conditions of Theorem 2 which are both sufficient and necessary. This corresponds to a slight change in the algorithm of Fig.2: Replace line 10 by

$$p_2 = \#M_1 + 2 \tag{24}$$

What can one gain by this change? The idea is that by allowing a larger set of reductions, and occasionally violating (21), we can produce better reductions. Maybe we can mitigate the two bits per recursive call build up of “non-maximality”, without paying anything?

To prove that the resulting algorithm returns a matrix that satisfies the robustness condition defined in Theorem 2, assume that the recursive calls return matrices that are robust, and let $(x; y) \in S$ be arbitrary.

For the robustness of the final matrix M, put $c = \lfloor 2^{-p_2}C \rfloor$, $\tilde{c} = 2^{-p_2}C - c$ and similarly for d and $\tilde{d}$, and form

$$\begin{aligned}M^{-1}\left\{\begin{pmatrix}A\\B\end{pmatrix} + \begin{pmatrix}x\\y\end{pmatrix}\right\} &= M_2^{-1}M_1^{-1}\left\{\begin{pmatrix}A\\B\end{pmatrix} + \begin{pmatrix}x\\y\end{pmatrix}\right\} \\ &= M_2^{-1}\left\{\begin{pmatrix}C\\D\end{pmatrix} + M_1^{-1}\begin{pmatrix}x\\y\end{pmatrix}\right\} \\ &= 2^{p_2}M_2^{-1}\left\{\begin{pmatrix}c\\d\end{pmatrix} + \underbrace{\begin{pmatrix}\tilde{c}\\\tilde{d}\end{pmatrix} + 2^{-p_2}M_1^{-1}\begin{pmatrix}x\\y\end{pmatrix}}_{\text{disturbance}}\right\}\end{aligned} \tag{25}$$

Since M_2 is robust with respect to $(c; d)$, all that remains to show is that the disturbance is in S. We have $0 \leq \tilde{c}, \tilde{d} < 1$ and by the choice of p_2, the elements of $2^{-p_2} M_1^{-1}(x; y)$ lie in the open interval $(-1/2, 1/2)$. It follows that the disturbance has elements in the open interval $(-1/2, 3/2)$, and hence $(\tilde{c}; \tilde{d}) + 2^{-p_2} M_1^{-1}(x; y) \in S$.

In this argument, there are two terms to the disturbance: The truncation error $(\tilde{c}; \tilde{d})$, and the input disturbance $(x; y)$, amplified by M_1^{-1}. In order to handle this, it is essential to allow a disturbance set S which is larger than a unit square.

8 Conclusions

HGCD algorithms based on the quotient sequence, and hence indirectly on Jebelean's criterion, are complex and require occasional backup steps to ensure correctness. These problems are exhibited as a lack of robustness of the quotient sequence.

A new robustness criterion is defined, which allows the design of HGCD algorithms without the need for backup steps. The strictest variant of the criterion leads to an algorithm (Fig. 2) of the same spirit as the work of Weilert and Schönhage, and which appears to be superior both in terms of running time and implementation complexity [6].

A more direct application of the same criterion leads to a novel HGCD algorithm. Further studies are needed to evaluate the properties and merits of these two variants.

There are also a number of optimizations that can be applied to all or most HGCD-like algorithms of the general structure of Fig. 1, including the binary recursive algorithm.

- The choice $p_1 \approx n/2$ makes sense if the recursive HGCD calls dominate the running time. But when the multiplications of Steps 4 and 10 are taken into account, it seems unlikely that $p_1 \approx n/2$ is optimal.
- In Step 4, bounds on the reduced numbers can be found a priori, and hence the calculations can be performed $\pmod{2^\ell}$ or $\pmod{2^\ell + 1}$, where the latter alternative is particularly attractive in the range of FFT multiplication.

References

1. Lehmer, D.H.: Euclid's algorithm for large numbers. American Mathematical Monthly **45** (1938) 227–233
2. Knuth, D.E.: The analysis of algorithms. Actes du Congrés International des Mathématiciens (1970) 269–274
3. Schönhage, A.: Schnelle Berechnung von Kettenbruchentwicklungen. Acta Informatica **1** (1971) 139–144
4. Thull, K., Yap, C.K.: A unified approach to HGCD algorithms for polynomials and integers (1990) Manuscript. Available from `http://cs.nyu.edu/cs/faculty/yap/papers`.

5. Pan, V.Y., Wang, X.: Acceleration of Euclidean algorithm and extensions. In: ISSAC '02: Proceedings of the 2002 international symposium on Symbolic and algebraic computation, New York, NY, USA, ACM Press (2002) 207–213
6. Möller, N.: On Schönhage's algorithm and subquadratic integer gcd computation. (2005) Preprint available at `http://www.lysator.liu.se/~nisse/archive/sgcd.pdf`
7. Stein, J.: Computational problems associated with Racah algebra. Journal of Computational Physics **1**(3) (1967) 397–405
8. Sorenson, J.P.: Two fast GCD algorithms. Journal of Algorithms **16**(1) (1994) 110–144
9. Weber, K.: The accelerated integer GCD algorithm. ACM Transactions on Mathematical Software **21** (1995) 111–122
10. Stehlé, D., Zimmermann, P.: A binary recursive GCD algorithm. In Buell, D., ed.: ANTS-VI. Volume 3076 of LCNS., Burlington, Springer-Verlag (2004)
11. Weilert, A.: Asymptotically fast GCD computation in $z[i]$. In: ANTS-IV: Proc. of 4th Intern. Symp. Algorithmic Number Theory. Number 1838 in LNCS, Leiden, Springer-Verlag (2000) 595–613 See page 602, in particular.
12. Brent, R.P., van der Poorten, A.J., te Riele, H.J.J.: A comparative study of algorithms for computing continued fractions of algebraic numbe rs. In Cohen, H., ed.: Algorithmic Number Theory. Volume 1122., Berlin (1996) 35–47
13. Weilert, A.: Ein schneller algorithmus zur berechnung des quartischen restsymbols. Diplomarbeit, Math. Inst. der Univ., Bonn (1999)
14. Lucier, B. Personal communication (2005)
15. Schönhage, A.: Fast reduction and composition of binary quadratic forms. In Watt, S.M., ed.: Proc. of Intern. Symp. on Symbolic and Algebraic Computation, Bonn, ACM Press (1991) 128–133
16. Stehlé, D., Zimmermann, P. Personal communication (2005–2006)
17. Jebelean, T.: A double-digit Lehmer-Euclid algorithm for finding the GCD of long integers. Journal of Symbolic Computation **19**(1-3) (1995) 145–157

The Design of CoCoALib

J. Abbott

Dipartimento di Matematica, Università di Genova, Italy

There are two ways of constructing a software design.
One way is to make it so simple
that there are obviously no deficiencies.
And the other way is to make it so complicated
that there are no obvious deficiencies.
The first method is far more difficult.
C.A.R. Hoare

Abstract. We describe some of the more important aspects of the design of CoCoALib, a new C++ library for effecting **Co**mputations in **Co**mmutative **A**lgebra. Special effort has been invested in making the code clean and portable while not neglecting run-time performance; one of the primary goals is to offer freely available reference implementations of the most important algorithms in the field.

1 Introduction

This document describes some of the more important design features of CoCoALib, a C++ library for computations in commutative algebra with particular focus on Gröbner bases. Theoretically the issue of computing Gröbner bases is resolved by Buchberger's algorithm, published about forty years ago. But, as commonly happens, there is a huge gulf between the simple elegance of a published algorithm, and the complex engineering hidden within a refined implementation. In the case of Buchberger's algorithm this gulf is even wider than usual: as witness we cite the numerous research papers published in the interim, and the fact that studies are still being actively pursued.

For almost 20 years the CoCoA project has been conducting research into computational commutative algebra, developing new algorithms and offering implementations in the interactive program "CoCoA". Recently we took the decision to rebuild the software from scratch with the specific aim of making excellent implementations available to all researchers. The implementations will be accessible in three distinct ways:

- as a standalone interactive system
- as a networked service (via an OpenMath-like communications channel)
- as a C++ library, called CoCoALib

CoCoALib, being the core of the project, is also its most evolved part, and is the part that we shall look at most closely. In keeping with the theme of ready accessibility the software is to be free and open in the sense of the GPL [6].

© Springer-Verlag Berlin Heidelberg 2006

2 Philosophy

Our aim is for CoCoALib to offer reference implementations of the principal algorithms in computational commutative algebra. The development of the library and the other components requires an enormous investment of time and resources. So that this investment is worthwhile we want to ensure that the software is widely available and will live for a long period of time. To attain these goals our implementations have to satisfy various design criteria:

- the code must be robust and reliable
- the code must exhibit good run-time performance
- the source code must be clear and well designed
- the source code must be well documented (both for users and maintainers)
- the source code must be clean and portable (excluding Microsoft quirkiness)
- the code must be easy and natural to use (whenever possible)

One implication of these criteria is that we regard clear and comprehensible code as being more desirable than arcanely convoluted code striving to achieve the utmost in run-time performance: clear code is more obviously correct and is easier to maintain, and so should live longer. Another implication is that the design should reflect the underlying mathematical structure since this will ensure that the library is natural to use.

Unfortunately, despite our best efforts in designing and building our software for reliability and longevity, there are potentially grave risks beyond our control: *e.g.* the threat of patents on algorithms casts an ominous shadow of doubt over the future of the CoCoA project (and many other similar open source projects).

2.1 Mathematical Underpinnings

The design of CoCoALib has been inspired and directed by the underlying mathematical structure: indeed, the development of the software design progressed symbiotically with that of the two-volume book of Robbiano and Kreuzer [2]. This inherent respect for the mathematical foundations makes CoCoALib natural and easy to use, while also making it clear that there are “obviously no deficiencies.

One of the main challenges of the design was to reconcile two traditionally conflicting goals: (mathematical) abstraction and efficiency. The inheritance mechanism of C++ plays a crucial role here. Our use of inheritance is exemplified by the way in which rings and their elements are implemented. An early design decision was to allow arbitrary (commutative) rings wherever possible: here “arbitrary” means that the precise nature of the ring can be known only at run time. Our design allows some knowledge of the ring to be expressed in the source code: for instance, we can specify that a routine is valid only for polynomial rings. The C++ inheritance mechanism is used to express the mathematical relationships between these various sorts of ring.

In contrast to the rich type structure used for rings, we use just a single C++ class to represent elements of rings. This is a compromise: it makes the library easier to create and to use, but also limits the amount of help the compiler can give us (since we are depriving it of information). We believe our choice offers the best combination of natural usability and future maintainability.

The implementation scheme chosen for rings works well, and analogous design patterns have also been employed elsewhere in the library: for instance in the code for modules and power product monoids.

Finally, to meet our run-time performance targets, in a few places in the very foundations of CoCoALib, we do use a little "quick and dirty hackery". Here credit is due to the design of C++ which allows such desperate code to be written where necessary, and which also permits it to fit seamlessly into a well-structured program.

3 What Is CoCoALib?

CoCoA is one of the main academic projects which produce free software specialized in computations in commutative algebra; two other similar systems are Singular [5] and Macaulay [4]. While there is much in common, the three groups do have differing and complementary emphases. Although a number of general purpose symbolic computation systems (e.g. REDUCE and Maple) do offer the possibility to compute Gröbner bases, typically their non-specialist nature implies a number of severe compromises which make them far less suitable to serve as a laboratory for researchers — most notably: relatively poor execution speed and limited control over the algorithm parameters.

Great care has been taken with the design of CoCoALib to make it easy and mathematically natural to use (within the constraints of C++); this in turn helps users to develop comprehensible programs whose correctness is self-evident. One notable problematic aspect is the use of "invisible homomorphisms", more commonly thought of as "natural identifications". After due consideration we opted to require that these be written out explicitly as this offers less scope for unexpected behaviour.

In deference to our human habit of erring, CoCoALib includes many checks for misuse, *e.g.* for erroneous or nonsensical input. If one of these checks reveals a problem, an indicative error is signalled (usually resulting in a helpful error message being printed). Of course, all these checks do incur potentially significant run-time overheads, so most basic operations can also be effected via a "fast and ugly" interface which avoids all the checks. Thus, in time critical subroutines, the programmer can choose to sacrifice safety (and readability) for more speed.

The usefulness of software is critically dependent on its documentation. So extensive documentation for the library has been prepared. Our documentation is divided into two major parts: one to aid and guide users of the library, and the other to help maintainers and contributors to the library (and perhaps to satisfy the more curious library users). Since it is often rather tedious to wade through volumes of precise technical description, there is also a good selection of example programs to illustrate the uses we anticipate for the various library components.

With some luck the CoCoALib user may find that much of the program writing can be achieved just by cutting-and-pasting from the examples.

3.1 CoCoALib in Comparison

CoCoALib has a number of features which distinguish it from other software able to compute Gröbner bases. The most obvious difference is that its principal component is a library, so the capabilities of CoCoALib are readily available to any other C++ program; this is not the case for programs which are standalone interactive systems. The library source code is GPLed (i.e. free and open, see [6]) and well-documented; we believe that the clarity of our code and the quality of our documentation exceed that of "our rivals".

Regarding the issue of execution speed, it is difficult to make a precise statement. An implementation of Buchberger's algorithm must specify two strategies: one for selecting pairs, and one for selecting reducers. The algorithm was proved correct for any choice of strategy. But the practical truth is that actual execution times (and memory requirements) depend enormously on the choice of strategies. So far no universally best strategies have been discovered; nevertheless, we believe the strategies employed in CoCoALib are generally pretty good. Bearing this in mind, we present a brief comparison between CoCoALib (version 0.96) and Singular (version 3.0.2-beta, see [5]).

In summary: a number of benchmarks suggest that CoCoALib is roughly comparable in speed to the `std` operation of Singular for computing Gröbner bases over small finite fields, while CoCoALib appears to be a bit faster than Singular when computing Gröbner bases over the rationals. Here are a few timings obtained on an Opteron 248 running GNU/Linux; unless otherwise indicated the coefficient ring was $\mathbb{F}_{32003}$.

Example	CoCoALib	Singular
Cyclic-7, drl	1	1
Cyclic-7, lex	53	55
Cyclic-7, drl, $\mathbb{Q}$	6	10
Cyclic-8, drl	100	67
Cyclic-8, lex	14000	14000
Cyclic-8, drl, $\mathbb{Q}$	920	14000

We have chosen Singular for the benchmark comparison because it was already installed on the computer. We would not expect qualitatively different results from a comparison with, say, Macaulay 2. A comprehensive, thorough and meaningful test of all programs capable of computing Gröbner bases would be a lengthy and arduous project. Indeed, several of the commercial algebra systems are so expensive that we question the sense in comparing free software against them.

4 Exploiting the Facilities of C++

In this section we look in some detail at the implementation of rings and their elements. The overall message is that use of inheritance and virtual member functions permits us to achieve both flexibility and efficiency.

4.1 Ring Types, Inheritance and Virtual Functions

It is quite clear that CoCoALib must offer ways of representing integers, rationals, and polynomials — all of these are elements of rings. An important early design decision was that it must also offer ways of representing the rings to which these values belong (*e.g.* $\mathbb{Z}, \mathbb{Q}, \mathbb{Q}[x, y, z]$). The representation of the ring elements seems not to pose any great problem as they are typically very "concrete" values.

In contrast, a mathematical ring is a "more abstract" concept, not the sort of value we expect a computer to manipulate. Yet for CoCoALib to be a useful mathematical research tool, it must have a means of representing certain sorts of commutative ring (with unit element). We also want CoCoALib to "know" about various special properties the rings may have (*e.g.* an integral domain) or to know how the ring was constructed (*e.g.* a ring of polynomials), so such information must be available. Thus the best way of representing a concrete ring in C++ is not entirely obvious. For instance, we need a way to tell in C++ which properties are possessed by that ring: one way would simply be to use some flags.

However, a well-known software design principle states that it is better to express as much "type information" as possible in the programming language, so that the compiler can help the programmer as much as possible (*e.g.* check that the types make sense even in rarely executed code). This maxim thus advises against encoding "type information" inside run-time flag values.

In CoCoALib we adopt a combined "compromise" approach: some properties are encoded as flags while others are expressed as C++ type information. Where possible the characteristics are expressed as C++ type information. Nevertheless one important characteristic has to be represented by a flag: whether the ring is in fact a field. This uncomfortable decision was dictated by the rules of C++: when creating a quotient ring the result might be a field or it might not (depending on the maximality of the ideal, a condition which can be tested only at run time) but the function which creates the quotient ring must produce a C++ object of type known precisely at compile time.

The approach taken in CoCoALib is as follows. There is a tree structure of abstract base classes related by (simple) inheritance, each base class represents a ring (commutative with unit) with certain properties:

```
RingBase             ← a commutative ring without special properties
  PolyRingBase       ← a polynomial ring
    SparsePolyRingBase ← a ring of sparse multivariate polynomials
  QuotientRingBase   ← a ring formed as a quotient
  FractionFieldBase  ← a ring formed as a fraction field
```

Each concrete class, whose objects will represent concrete rings, is derived simply from the most specific base class that characterises it. For example, a class which represents small prime finite fields is derived from `QuotientRingBase`, whereas the class which represents the ring of integers is derived directly from `RingBase`.

Those ring properties which are not expressed as C++ types are instead expressed as virtual member functions: for instance `IamField` returns `false` by default, but can be overridden in a concrete ring class to return `true`. Thus there is no visible flag indicating the presence or absence of this property; instead it is implicit in the virtual function mechanism. Here is the implementation of `IamField` for the class implementing quotient rings:

```
bool GeneralQuotientRingImpl::IamField() const
{  return IsMaximal(myReducingIdeal);  }
```

Note that the truth value can only be determined at run time, and that the implementation is directly inspired by the mathematical definition.

As the C++ virtual function mechanism works via pointers (here we ignore the subtle distinction between a reference and a pointer) the inheritance tree structure is mirrored in a second tree of concrete "smart pointer" classes: for each abstract ring class there is a corresponding "smart pointer" class which points to an object derived from that abstract class. These smart pointers disguise the fact that pointers are being used (thus avoiding an epidemic of asterisks in the code); they also implement a reference counting mechanism so that the concrete ring class is destroyed automatically at the right time.

Ring Elements. Previously the issue of representing values belonging to a ring was dismissed as seemingly trivial. While there are indeed no great technical difficulties, some care is required. Consider a simple case: the representation of the value 1 in the rings $\mathbb{Z}, \mathbb{Q}, \mathbb{Q}[x]$. In $\mathbb{Z}$ one would expect the value to be held as an arbitrary precision integer; in $\mathbb{Q}$ one would expect the value to held as as arbitrary precision rational; and in $\mathbb{Q}[x]$ the value would presumably be some representation of $1 \times x^0$. The main point is that the precise details of the representation depend on the ring; and these details should be kept hidden inside each ring (in C++ parlance the information is `private`).

Having recognised the "privateness" of the precise details of the representation of the value of a ring element, we defined the data structure `RingElem` for containing elements of rings. The data structure comprises two components: the identity of the ring to which the value belongs, and an opaque pointer to the value — the pointer must be opaque (*i.e.* `void*`) since the type of value pointed at depends on the ring.

Another consequence of the "privateness" of ring elements' representations is that in CoCoALib a concrete ring class does more than merely represent an abstract mathematical entity. The class also manages all values which belong to it. Indeed, the ring class defines (implicitly) the manner in which its values are codified in the computer; so only that ring is able to interpret properly the bit pattern in memory corresponding to the stored value. Thus when we do arithmetic with ring elements, the actual computation is conducted by a member function of the ring class. Fortunately, the virtual function mechanism in C++ lets most of this housekeeping happen invisibly — see below.

Given that all operations on ring elements have to be effected by the ring, it will come as no surprise that in the definition of the most basic abstract ring class, `RingBase`, there are many pure virtual member functions for creating, destroying, and performing arithmetic on ring elements. These functions are necessarily pure virtual since their implementation depends on how each concrete ring codifies its own values. An exception is `myPower`, the member function for computing powers, which has a default definition in terms of multiplication; nonetheless `myPower` is virtual so that it can be overridden in those rings where more efficient powering techniques exist (*e.g.* $\mathbb{F}_p$).

Let's see how rings and their elements interoperate. Almost every operation on ring elements is implemented as a C++ function or operator with a natural syntax which checks suitability of the arguments (*e.g.* belonging to the same ring, non-zero divisor), and if the checks pass then just the opaque pointers are passed to the relevant ring member functions. The ring member functions accept opaque pointers partly for "minimalism", and partly to emphasise that they do not check compatibility or suitability of their arguments. It is also possible to call directly the member function, but this is rarely worthwhile.

Let `R` denote a C++ variable of type `ring`, and let `x`, `a`, `b` all be C++ variables of type `RingElem` containing values belonging to `R`. Now consider the two following lines, which are essentially equivalent:

```
x = a + b;
R->myAdd(raw(x), raw(a), raw(b));
```

The first line checks that `x`, `a` and `b` do all belong to the same ring whereas the second line makes no such check; the second line just calls the function `raw` which extracts the opaque pointer and discards the information about the owning ring, then passes the pointers to the member function. If `x`, `a` and `b` do not all belong to `R` then the second line will result in "undefined behaviour" (*i.e.* probably a program crash). The first line will create a temporary value for the sum, the second line might not. Essentially the same choice exists for the other arithmetic operators.

Here is a similar situation which illustrates slightly more complicated behaviour:

```
x = gcd(a, b);
R->myGcd(raw(x), raw(a), raw(b));
```

Besides checking the compatibility of `x`, `a` and `b`, the first line also checks that the ring supports a gcd operation (since not all rings do); if not, an appropriate exception is signalled, *viz.* (`CoCoA::ERR::NotGCDDomain`). In contrast, if the ring `R` does not support a gcd operation then the second call will provoke a less helpful and more severe error by throwing `CoCoA::ERR::SERIOUS` because a direct call to a member function of `RingBase` should be made only when we know that the call is valid.

Small Finite Fields. At this point we note that elements of small prime finite fields are implemented exactly as described above despite the potential for accusations of scandalous inefficiency. Indeed, there is no doubt that significant gains in efficiency could be achieved by using inline operations rather than virtual function calls, and by storing the values in local variables rather than on the heap. CoCoALib offers the possibility of such an efficient implementation but at the cost of unnatural syntax and incompatibility with the fully generic ring element mechanism. This efficient but unnatural implementation is used in a few places in CoCoALib such as the special implementation of polynomials with coefficients in a small prime finite field — a normal user need know nothing about this special implementation as the library will automatically use the fast implementation if it can. Much the same idea can be used more widely, such as for matrices over small finite fields.

4.2 Templates and Inline Functions

The curious programmer might well be surprised to find that there are very few template functions or inline functions in CoCoALib. The lack of template functions is primarily due to the extensive use of inheritance to handle "polymorphism". Though using inheritance and virtual functions is not as fast as well-crafted template code, the difference in run-time efficiency proved to be remarkably small in our trials (here credit is due to the compiler writers), and the comprehensibility and maintainability of non-template code proved to be vastly superior.

One of the great attractions of C++ is the possibility of writing inline functions. Indeed CoCoALib contains a large number of very simple functions, yet only a small proportion of these are inline. Initially we defined almost all simple functions as inline, only to discover later that often this led to no measurable run-time gain compared to a normal "out of line" definition (while sometimes forcing public exposure in header files of private implementation details). The only functions worth making inline are those which are called an exceedingly large number of times, *e.g.* certain data member accessor functions, and the subset test for "div-masks" (see section 5).

There are several advantages of not making functions inline: the header files become simpler and so compilation gets faster, encapsulation can improve because private implementation details can be kept hidden, and fewer files need recompilation when the implementation is changed (*i.e.* debugged). We conduct regular profiling checks to make sure the choice of functions defined inline is wise.

5 Case Study: Divisibility Masks

The reason for giving this example particular prominence is that it demonstrates a way of combining inline operations with C++ inheritance to produce an implementation offering both speed and flexibility.

We start with a description of the problem we need to solve. One apparently trivial step in Buchberger's algorithm, which often turns out to be rather costly, is the search for a suitable reducer. In abstract, we start with a power product t (the LPP of the polynomial to be reduced) and a collection of further power products $t_1, t_2, \dots, t_r$ (the LPPs of the reducers) and must find whether some t_k divides t. Now, a power product is just a product of powers of indeterminates $\prod x_i^{e_i}$ which we can think of as being represented by the exponent vector $(e_1, e_2, \dots, e_n)$. Naturally, divisibility of power products boils down to simply comparing components of the two exponent vectors. Unfortunately this proves to be too slow.

A simple idea, which turns out to be rather effective in many cases, is to use bit masks as well as exponent vectors. If we set the j-th bit of the mask whenever the indeterminate x_j has non-zero exponent then we can quickly determine non-divisibility just by comparing bit masks: if a candidate factor contains an indeterminate not present in the power product t, we can immediately exclude the candidate. We call these bit-masks **div masks** since they are used for divisibility testing.

The idea described above is only a start, and needs to be refined; *e.g.* it works poorly when there are few indeterminates and high degrees. A little meditation rapidly leads us to realise that there are many different "div-mask rules" for setting bits in such a way that the div masks of every factor of a power product, t, are (non-strict) subsets of the div mask of t. Moreover, no single rule stands out as being obviously superior to all others; so we must allow the user to choose the rule to use.

In CoCoALib we use C++ inheritance to implement several sorts of div-mask rule. An essential observation is that the most common operation on div-masks is that of subset testing; filling of div-masks is second most common, but typically occurs far less often than subset testing. So it is vital to have a fast subset test, and no so important that filling be quick. We achieve this aim by using a compile-time fixed data structure for the div-mask values (a C++ `bit_set` of fixed size) which allows subset testing to be implemented as inline bit-twiddling operations. The filling of the div-mask is achieved through an out-of-line virtual function call to the appropriate div-mask rule class. Here we have achieved run-time selection of the div-mask rule at the same time as allowing inlined subset testing.

6 Twin Float Arithmetic

One feature apparently unique to CoCoALib is the possibility to compute with twin floats: these are limited precision floating point values employing probabilistic verification of computed values based on an idea of Traverso [1]. The idea was originally developed to compute quickly a good, probabilistically guaranteed approximation to a Gröbner basis. In CoCoALib the idea has been developed into a type of ring, and so can be easily used for any computation. In particular, it is possible to test two twin-float values for equality to obtain a result that is "probably correct".

Twin-float arithmetic is suitable for computations where the input data are known exactly, and where an approximate answer is acceptable. It cannot be applied directly to problems where the input is known only approximately. It works well when exact computation with rational numbers becomes excessively slow due to uncontrolled growth of the numerators and denominators of the rationals (*e.g.* during Gröbner basis computation).

Twin-float arithmetic is presented as a ring in CoCoALib, but this is not entirely honest. For instance, for any given twin-float "ring" it is possible to find a value $\epsilon > 0$ in the ring such that $1+\epsilon = 1$. Another quirk of twin-float arithmetic is that almost any arithmetic operation can fail due to "insufficient precision" (the most obvious case is when subtracting two close values). Even the equality test has three possible outcomes: true, false or "insufficient precision" (thus violating the Law of the Excluded Middle). The C++ exception mechanism is used to signal problems of insufficient precision, and some example programs show how to exploit this mechanism to retry computations at ever higher precisions until a sufficiently accurate result is obtained.

7 Other Features

CoCoALib offers more facilities than we can describe in this article. Here are a few comments on some of the more noteworthy features.

An essential operation for many algorithms in computational algebra is the ability to apply a homomorphism. CoCoALib allows ring homomorphisms to be used as "normal values" (*i.e.* stored in variables, passed as arguments, etc.). Once again the inheritance mechanism of C++ enables us to model cleanly the mathematical abstraction and to achieve good run-time performance at the same time.

A current collaboration between the CoCoA group and Shell International is studying applications of commutative algebra to problems with approximate coefficients. Consequently, CoCoALib will soon contain implementations of certain algorithms from this area.

Acknowledgment. CoCoALib relies on the GMP arbitrary precision arithmetic library (see `http://www.swox.com/`).

8 Availability of the Software

The website for the CoCoA project may be found at:

`http://cocoa.dima.unige.it/` $\leftarrow$ note that there is no `www`

CoCoALib is rapidly approaching the junction between "alpha testing" and "beta testing": we expect a full public release of version 1.0 later this year. The principal designer is John Abbott; further important contributions have been made by Anna Bigatti and Massimo Caboara.

A prototype server based on CoCoALib is supplied as part of the "old" system CoCoA 4.6. This offers access to several features of CoCoALib to those not wishing to use the C++ language. Eventually CoCoA 4.6 will be supplanted by a completely new interactive front end.

References

1. C. Traverso, A. Zanoni: *Numerical Stability and Stabilization of Groebner Basis Computation* Proc. ISSAC 2002, pp. 262-269
2. L. Robbiano, M. Kreuzer: *Computational Commutative Algebra (vols. 1 and 2)* Springer-Verlag, 2000 and 2005
3. The CoCoA Team: *The CoCoA Project* (includes both CoCoALib and the old CoCoA-4.6) Principal web site is `http://cocoa.dima.unige.it/`
4. D. Grayson, M. Stillman: *Macaulay 2 Computer Algebra System* Principal web site is `http://www.math.uiuc.edu/Macaulay2/`
5. G.-M. Greuel, G. Pfister, H. Schonemann: *Singular Computer Algebra System* Principal web site is `http://www.singular.uni-kl.de/`
6. The Free Software Foundation: *GNU General Public License* Main web site is `http://www.gnu.org/licenses/licenses.html`

Generation of Oriented Matroids Using Satisfiability Solvers

Lars Schewe[*]

TU Darmstadt, Fachbereich Mathematik, AG 7, Schlossgartenstraße 7, 64289 Darmstadt, Germany

Oriented matroids are a combinatorial abstraction of finite sets of points in $\mathbb{R}^n$. They have been used to study various problems in discrete and computational geometry (for more material on oriented matroids, see [1,2]). A number of methods to generate oriented matroids have been proposed (for instance in [4,8–10]) as these methods can be used as a building block for the algorithmic treatment of some hard problems: Algorithms to generate oriented matroids have for instance been used to decide whether certain 4-polytopes exist [6,10,14]. For these questions it is important to have effective algorithms for the generation of oriented matroids. We propose to use satisfiability solvers to generate oriented matroids. We have adapted this approach to generate oriented matroids that satisfy certain geometric constraints. Even though one can use the generated oriented matroids as a first step to find realizations (see for instance [2,6]), we will only focus on non-realizability results.

We applied our method to the following problem: Given an abstract simplicial complex Δ on a finite set E, we ask whether we can find, for given n, a polyhedral embedding of Δ in $\mathbb{R}^n$. A notorious special case of this question was described by Grünbaum: He asked, whether all triangulated orientable surfaces (that means that Δ is a closed, connected, orientable 2-manifold) admit a polyhedral embedding in $\mathbb{R}^3$. The first counter-example was found by Bokowski and Guedes de Oliveira [4]. Their idea was to show that the triangulation in question does not admit an oriented matroid. For this they used an algorithm that generates all admissible oriented matroids. In the case they investigated the algorithm yielded the result that no admissible oriented matroid exists. However, this approach needed a lot of CPU time; to apply this approach to more examples, one needs a better method to generate oriented matroids.

Following the approach of Bokowski and Guedes de Oliveira with our new generation method we obtained the following new results:

1. No triangulation of a surface of genus 6 using only 12 vertices admits a polyhedral embedding in $\mathbb{R}^3$.
2. There exist at least three triangulations of a surface of genus 5 using only 12 vertices that do not admit a polyhedral embedding.
3. For every $g \geq 5$ we can construct an infinite family of triangulations of a surface of genus g such that none of these admit a polyhedral embedding in $\mathbb{R}^3$.

[*] The author was supported by a scholarship of the Deutsche Telekom Foundation.

© Springer-Verlag Berlin Heidelberg 2006

We also used the same method to decide whether certain point-line-configurations exist [3, 5].

The method we propose can be outlined as follows: we generate all necessary constraints and delegate the backtracking to a satisfiability solver. We transform the problem of generating oriented matroids (subject to additional constraints) into an instance of the satisfiability problem. If we find a solution, we add a constraint excluding the solution just found and resolve. Thus, we can generate all admissible oriented matroids. This is similar to an approach proposed by Bremner – he modeled the problem as an integer programming feasibility problem.

We can translate the so called chirotope axioms for oriented matroids into an instance of the satisfiability problem. For an rank r oriented matroid on an n element set the number of variables is proportional to $\binom{n}{r}$. In the general case we need two variables for each r-element subset. If we restrict our attention to uniform oriented matroids – they correspond to the case where points are in general position – we need only one variable; the encoding of the oriented matroid axioms is also much simpler.

This set of constraints can be extended to generate oriented matroids subject to certain conditions. For instance, if we want to embed a simplicial complex Δ we get the additional constraint that the simplices of Δ may not intersect. We can express this by adding new clauses to our satisfiability instance. The clauses forbid certain *circuits* of the oriented matroid.

We rely on the additional feature of modern satisfiability solvers to add clauses to a problem instance later on. This allows us to generate *all* admissible oriented matroids. If we found a solution, we add a clause that forbids the solution just found and resolve. As the solver keeps its state, we do not lose the information of the previous run, which keeps the method effective.

We think one advantage of our method is that we can use 'off-the-shelf' software to solve the satisfiability problem. So far, we have sucessfully used the solvers MiniSat [11] and ZChaff [13]. With our approach any progress in solving satisifiablity problems will give us better methods to generate oriented matroids.

The solver ZChaff allows also to generate a 'proof log'. If an instance is unsatisfiable, the solver generates a proof of that fact. This gives us in principle a possibility to use a proof checker such as Coq [7] to give a formal proof for our results. The main problem is, however, that the proof generated by the solver is huge.

So far, the best method to show that no polyhedral embedding of a given simplicial exists was the method by Bokowski and Guedes de Oliveira. In 2000 it took them four month (using various computers in parallel) to decide that one triangulation with 12 vertices of the surface of genus 6 did not admit a polyhedral embedding. With our method the instance of the satisfiability problem has 495 variables and 226513 clauses. We can solve this on a a machine with two Pentium III processors (1 GHz) and 2 GB RAM in under 2 hours (the process used only one processor and only 47 MB RAM).

We have implemented the outlined method using a Haskell program to generate the constraints and solved the resulting instances using MiniSat and ZChaff. In both cases we wrote thin wrappers to directly call them from the Haskell program.

Right now, we try to implement this method as a topaz-client in the polymake-framework [12].

References

[1] Anders Björner, Michel Las Vergnas, Bernd Sturmfels, Neil White, and Günter M. Ziegler, *Oriented Matroids*, 2nd ed., Encyclopedia of Mathematics and its Applications, vol. 46, Cambridge University Press, Cambridge, 1999.

[2] Jürgen Bokowski, *Computational Oriented Matroids*, Cambridge University Press, 2006.

[3] Jürgen Bokowski, Branko Grünbaum, and Lars Schewe, *Topological configurations* (n_4) *exist for all* $n > 16$, submitted.

[4] Jürgen Bokowski and António Guedes de Oliveira, *On the generation of oriented matroids*, Discrete Comput. Geom. **24** (2000), no. 2-3, 197–208. The Branko Grünbaum birthday issue.

[5] Jürgen Bokowski and Lars Schewe, *There are no realizable* 15_4- *and* 16_4-*configurations*, Rev. Roumaine Math. Pures Appl. **50** (2005), no. 5–6, 483–493.

[6] Jürgen Bokowski and Bernd Sturmfels, *Computational Synthetic Geometry*, Lecture Notes in Mathematics, vol. 1355, Springer-Verlag, Berlin, 1989.

[7] The Coq development team, *The Coq proof assistant reference manual*, LogiCal Project, 2004. Version 8.0.

[8] Lukas Finschi and Komei Fukuda, *Generation of oriented matroids—a graph theoretical approach*, Discrete Comput. Geom. **27** (2002), no. 1, 117–136. Geometric combinatorics (San Francisco, CA/Davis, CA, 2000).

[9] Ralf Gugisch, *Konstruktion von Isomorphieklassen orientierter Matroide*, Dissertation, Universität Bayreuth, 2005 (German).

[10] David Bremner, *MPC – Matroid Polytope Completion*, June 2004. available at `http://www.cs.unb.ca/profs/bremner/mpc/`.

[11] Niklas Eén and Niklas Sörensson, *An extensible SAT-solver*, SAT (2003) (Enrico Giunchiglia and Armando Tacchella, eds.), Lecture Notes in Computer Science, vol. 2919, Springer, 2004, pp. 502–518.

[12] Ewgenij Gawrilow and Michael Joswig, *polymake: a Framework for Analyzing Convex Polytopes*, Polytopes — Combinatorics and Computation (Gil Kalai and Günter M. Ziegler, eds.), Birkhäuser, 2000, pp. 43–74.

[13] Yogesh S. Mahajan, Zhaohui Fu, and Sharad Malik, *Zchaff2004: An efficient SAT solver.*, SAT (2004) (Holger H. Hoos and David G. Mitchell, eds.), Lecture Notes in Computer Science, vol. 3542, Springer, 2005, pp. 360–375.

[14] Peter Schuchert, *Matroidpolytope und Einbettungen kombinatorischer Mannigfaltigkeiten*, Dissertation, TH Darmstadt, 1995 (German).

Flexible Object Hierarchies in Polymake
(Extended Abstract)

Ewgenij Gawrilow[1] and Michael Joswig[2,*]

[1] Institut für Mathematik, MA 6-1, TU Berlin, 10623 Berlin, Germany
gawrilow@math.tu-berlin.de
[2] Fachbereich Mathematik, AG 7, TU Darmstadt, 64289 Darmstadt, Germany
joswig@mathematik.tu-darmstadt.de

1 Introduction

Initially polymake [1,2,3] was conceived as a collection of tools for studying convex polyhedra only. The early versions of polymake had a very primitive data management, built around a single data type for polyhedra. However, as the time passed, more and more different discrete mathematical structures like graphs and simplicial complexes came along. This gave rise to a properly typed object hierarchy, which was strict enough to support established object-oriented (OO) software techniques, but, on the other side, flexible enough to allow for continuous extensions without breaking the compatibility.

These considerations have lead to a very abstract object model which will be sketched here. Most of the features were introduced in the release 2.1 published in 2005. A polymake object type is defined as a named collection of properties. The properties have to be declared, at least, with a name and data type. The declaration language is (a slight extension of) Perl; the object types are eventually translated into usual Perl classes. Unlike in the popular OO programming languages (e.g., C++, Java), the property declarations belonging to an object type do not have to be concentrated at one place. Instead, they can be spread over several sources. This key feature allows the user to extend the semantics of an object type by inventing new properties (and methods to compute them) without introducing a new, derived object type. Furthermore, one can easily merge several extensions coming from different sources. This technique is not generally supported in classical OO languages, as the merging of two derived types would lead to common base class ambiguity.

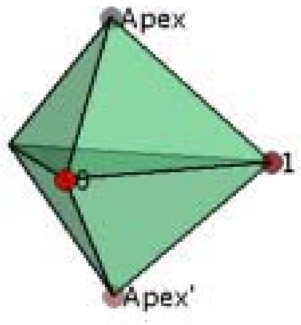
polymake object types do support derivation, though. The derived object type inherits all properties from its parent(s), there are no "private" properties. The conflicts which may arise by multiple inheritance (i.e., properties with identical names defined in two or more parent types) are, theoretically, allowed, since Perl resolves them via the depth-first search in the inheritance tree. Such conflicts should be avoided, however: They are a sure sign of a bad design of the type hierarchy.

[*] partially supported by Deutsche Forschungsgemeinschaft, DFG Research Group "Polyhedral Surfaces."

© Springer-Verlag Berlin Heidelberg 2006

2 Categorizing Object Properties

Throughout the following we will use polytopes in order to illustrate the concepts that we report on. A *(convex) polytope* is the convex hull of finitely many points in Euclidean space. As a general reference the reader is referred to Ziegler [4]. Let's look at a tiny polytope: a bipyramid over a triangle. Here you see a picture produced by JavaView [5] and the data file describing the polytope.



```
_application polytope
_version 2.2
_type RationalPolytope

VERTICES
1 0 0 0
1 1 0 0
1 0 1 0
1 1/3 1/3 2/3
1 1/3 1/3 -2/3

VERTEX_LABELS
0 1 2 Apex Apex'

FACETS
0 2 0 1
0 2 0 -1
2 -2 -2 -1
2 -2 -2 1
0 0 2 -1
0 0 2 1

VERTICES_IN_FACETS
{0 2 4}
{0 2 3}
{1 2 3}
{1 2 4}
{0 1 3}
{0 1 4}

DIM
3

BOUNDED
true

VIF_CYCLIC_NORMAL
{0 2 4}
{3 2 0}
{1 2 3}
{4 2 1}
{0 1 3}
{0 4 1}

TRIANGULATION.FACES
{0 1 2 3}
{0 1 2 4}

LP.OBJECTIVE
0 0 0 1

LP.MAXIMAL_FACE
{3}

LP.MAXIMAL_VALUE
1
```

The object properties can be classified according to various criteria. The classification is part of the property declaration.

Hierarchy. There are *atomic* properties and *sub-object* properties. Most of the properties shown above are atomic ones (VERTICES, BOUNDED, etc.) They are stored as native Perl scalars, arrays, or C++ objects with Perl interface, whichever is appropriate. TRIANGULATION is a sub-object of a different type SimplicialComplex, which comes with its own set of properties and methods. This is an interesting (mathematical) object of its own, and its properties (e.g., its f-vector) are not directly related to the polytope which is triangulated.

Multiplicity. A property can be *unique* or *multiple*. Atomic properties are always unique. A sub-object property may be declared multiple, if several instances of interest may differ in a non-trivial way. For example, there is more than one way to triangulate a given polytope. It is an important feature that a unique property, like the volume, may depend on a multiple one.

Variability. Most of the properties are *immutable*, they are assigned to the object in the course of its creation or computed later from its other properties. Some properties are, however, inessential (from the mathematical point of view), like `VERTEX_LABELS`, which is used solely for visualization. Such properties are declared *mutable* and are allowed to be changed during the whole object's lifespan. Other example of a mutable (and multiple) property is the sub-object `LP` (linear program) with the objective function specified in `LP.OBJECTIVE` (here: $z \rightarrow$max). It should be stressed, that despite the entire LP sub-object is mutable with respect to the containing polytope, the LP's own properties `OBJECTIVE`, `MAXIMAL_FACE`, etc. are immutable, as they define the LP or are computed from other LP's and polytope's properties.
Storage. The properties usually are costly to obtain. Hence objects with their *persistent* properties are stored in files. Once computed, polymake can retrieve them on a user's request directly from the file. Properties serving special purposes (e.g. visualization), which are cheap enough to be recomputed, can be declared *temporary*. Their lifespan is much shorter than that of the object. For example, `VIF_CYCLIC_NORMAL` repeats the incidence information, but with vertices being rearranged clockwise.

It may be worth mentioning that the object hierarchy is allowed to be cyclic. E.g., a linear program may be stored as a polytope object with a linear objective function as its sub-object or, vice versa, as a linear objective function with a polytope sub-object. This becomes useful with multiplicity taken into account: There may be one polytope with a number of linear objective functions or one linear objective function to be evaluated on several polytopes.

A final remark about the immutable nature of the objects. polymake treats objects much like one defines them in mathematics: "Let P be a polytope with ...". Once introduced, a polymake object should keep its semantics. The rule-driven computation of properties heavily relies on the consistency of all essential properties, and therefore polymake does not allow to modify them.

References

1. Gawrilow, E., Joswig, M.: `http://www.polymake.de/` polymake Version 2.2 (2006), with contributions by Thilo Schröder and Nikolaus Witte
2. Gawrilow, E., Joswig, M.: polymake: an approach to modular software design in computational geometry. Proc. of the 17th ACM Annual Symposium on Computational Geometry (2001) 222–231
3. Gawrilow, E., Joswig, M.: Geometric reasoning with polymake. Preprint `arxiv.org/math.CO/0507273` , 14 pages (2005)
4. Ziegler, Günter M.: Lectures on polytopes. Graduate Texts in Mathematics **152**, Springer-Verlag (1995)
5. Polthier, K. et al: `http://www.javaview.de/` JavaView - Interactive 3D Geometry and Visualization

A Presentation of the Gfan Software

Anders N. Jensen[⋆]

Department of Mathematical Sciences,
University of Aarhus, DK-8000 Århus, Denmark

Gfan [8] is a software package for computing Gröbner fans and tropical varieties of polynomial ideals. The Gröbner fan of an ideal $I \subset \mathbb{Q}[x_1, \ldots, x_n]$ is a polyhedral complex defined in [9]. For a homogeneous ideal the Gröbner fan is a complete fan and the normal fan of a polytope. Its cones are in bijection with the various initial ideals of I. In particular, the full dimensional cones are in bijection with the monomial initial ideals and thereby also in bijection with the reduced Gröbner bases of I. In [3] the local basis change of Gröbner bases was introduced. This method allows us to go from one Gröbner basis in the fan to a neighboring one, giving an effective algorithm for computing the Gröbner fan by traversing its maximal cones. The method can be refined by applying the reverse search technique [1]. This works even in the non-homogeneous case [5].

A computation of the Gröbner fan of I is useful when searching for an initial ideal with a particular property. The computer program TiGERS [7] is an earlier implementation of the Gröbner fan traversal for toric ideals. Gfan is more general as it can compute the Gröbner fan of any polynomial ideal $I \subset \mathbb{Q}[x_1, \ldots, x_n]$. Gfan even allows an interactive investigation of the fan.

Recently the field of tropical mathematics has received much attention. In tropical mathematics the semi-ring $(\mathbb{R}, \max, +)$ is considered. Here maximum takes the role of addition and plus the role of multiplication. Many combinatorial optimization problems are easily expressed in tropical notation. In tropical mathematics classical objects have tropical analogs. For example the tropical variety $\mathcal{T}(I)$ of a polynomial ideal $I \subset \mathbb{Q}[x_1, \ldots, x_n]$ is the analog of the usual variety of the ideal. There are many equivalent ways of defining the tropical variety of I. We prefer to state the definition in terms of initial ideals:

$$\mathcal{T}(I) := \{\omega \in \mathbb{R}^n : \text{in}_\omega(I) \text{ contains no monomials}\}$$

where $\text{in}_\omega(I)$ denotes the initial ideal of I with respect to the vector ω. The fact that the tropical variety is a union of Gröbner cones ensures that the tropical variety can be given the structure of a polyhedral complex making it a subfan of the Gröbner fan. A breadth first traversal algorithm for computing tropical varieties of prime ideals was developed in [2]. The algorithm relies on the relation

⋆ Research partially supported by the Faculty of Science, University of Aarhus, Danish Research Training Council (Forskeruddannelsesrådet, FUR) , Institute for Operations Research ETH, the Swiss National Science Foundation Project 200021-105202, grants DMS 0222452 and DMS 0100141 of the U.S. National Science Foundation and the American Institute of Mathematics.

© Springer-Verlag Berlin Heidelberg 2006

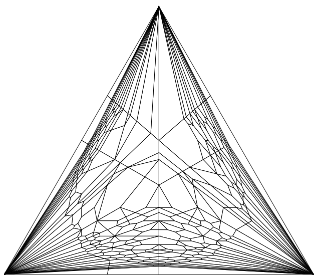


Fig. 1. The Gröbner fan of the ideal $\langle x_1^5+x_2^3+x_3^2-1, x_1^2+x_2^2+x_3-1, x_1^6+x_2^5+x_3^3-1\rangle$ intersected with the standard 2-simplex in $\mathbb{R}^3$

to Gröbner fans, a connectivity result, constructions of tropical bases, polyhedral computations and lifting results. Gfan contains the only existing implementation of this algorithm.

The Gfan package consists of several command line programs which may be combined to produce the desired result. For example:

```
gfan <input.txt | gfan_render > picture1.fig
```

will produce the drawing of the Gröbner fan of the ideal written in the file `input.txt`, see Fig. 1. Usually a list of the reduced Gröbner bases is a satisfactory output of a Gröbner fan computation while the actual combinatorics of a tropical variety often are of interest. Sometimes symmetry can be exploited and the output may be organized in symmetry classes making it more readable.

While Gröbner basis computations are doubly exponential in the worst case the Buchberger step of the local Gröbner basis change procedure is relatively easy in practice due to the homogeneity and other properties of the ideals in question. However, sometimes the Gröbner fan is simply too big to be computed or the conversion steps do take too much time. Here is a few examples of what can be computed. In Fig. 1 the 360 maximal cones were computed in 58 seconds using 5 megabytes of memory. It is easy to find four variable ideals where the Gröbner fan is too big to be traversed or the local steps are too time consuming. For some classes of ideals the program can handle much larger examples. The ideal in 16 variables generated by the 3 by 3 minors of a 4 by 4 matrix of variables has 163032 reduced Gröbner bases which, up to symmetry, can be computed in 7 minutes using 9 megabytes of memory. There are only 289 orbits to consider. Without exploiting symmetry the computation takes 14 hours.

Gfan [8] can be compiled with any newer version of gcc on a Linux or Mac OS X system with gmp [6] and cddlib [4] installed. These libraries are used for exact arithmetic and polyhedral computations, respectively.

In the talk we will give a brief overview of the implemented algorithms and show how to compute some interesting Gröbner fans and tropical varieties using the software.

References

1. Avis, D., Fukuda, K.: A basis enumeration algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. Discrete Computational Geometry **8** (1992) 295–313
2. Bogart, T., Jensen, A., Thomas, R., Speyer, D., Sturmfels, B.: Computing tropical varieties. J. Symb. Comput., to appear (2005)
3. Collart, S., Kalkbrener, M., Mall, D.: Converting bases with the Gröbner walk. J. Symb. Comput. **24**(3/4) (1997) 465–469
4. Fukuda, K.: cddlib reference manual, cddlib Version 094b. Swiss Federal Institute of Technology, Lausanne and Zürich, Switzerland. (2005) `http://www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html`
5. Fukuda, K., Jensen, A., Thomas, R.: Computing Gröbner fans. Mathematics of Computation, to appear (2005)
6. Granlund, T., et al.: GNU multiple precision arithmetic library 4.1.2 (2002) `http://swox.com/gmp/`.
7. Huber, B., Thomas, R.R.: Computing Gröbner fans of toric ideals. Experimental Mathematics **9**(3/4) (2000) 321–331
8. Jensen, A.N.: Gfan, a software system for Gröbner fans (2005) `http://home.imf.au.dk/ajensen/software/gfan/gfan.html`
9. Mora, T., Robbiano, L.: The Gröbner fan of an ideal. J. Symb. Comput. **6**(2/3) (1988) 183–208

Parallel Homotopy Algorithms to Solve Polynomial Systems[⋆]

Anton Leykin[1], Jan Verschelde[2], and Yan Zhuang[3]

Department of Mathematics, Statistics, and Computer Science
University of Illinois at Chicago, 851 South Morgan (M/C 249)
Chicago, IL 60607-7045, USA
[1]`leykin@math.uic.edu`
`http://www.math.uic.edu/~leykin`
[2]`jan@math.uic.edu`
`http://www.math.uic.edu/~jan`
[3]`yzhuan1@uic.edu`
`http://www2.uic.edu/~yzhuan1`

Abstract. Homotopy continuation methods to compute numerical approximations to all isolated solutions of a polynomial system are known as "embarrassingly parallel", i.e.: because of their low communication overhead, these methods scale very well for a large number of processors. Because so many important problems remain unsolved mainly due to their intrinsic computational complexity, it would be embarrassing not to develop parallel implementations of polynomial homotopy continuation methods. This paper concerns the development of "parallel PHCpack", a project which started a couple of years ago in collaboration with Yusong Wang, and which currently continues with Anton Leykin (parallel irreducible decomposition) and Yan Zhuang (parallel polyhedral homotopies). We report on our efforts to make PHCpack ready to solve large polynomial systems which arise in applications.

2000 Mathematics Subject Classification. Primary 65H10. Secondary 14Q99, 68W30.

Keywords and phrases: Continuation methods, high performance continuation, jumpstarting homotopies, linear-product systems, parallel computation, path following, polynomial systems, polyhedral homotopies, simplex system.

1 Motivation: We Want to Solve Large Systems

To solve a polynomial system $f(\mathbf{x}) = \mathbf{0}$, a homotopy $h(\mathbf{x}, t) = \mathbf{0}$ connects f to a start system $g(\mathbf{x}) = \mathbf{0}$ (g stands for *generic*, i.e.: all start solutions are regular), for example of the following form

[⋆] This material is based upon work supported by the National Science Foundation under Grant No. 0134611 and Grant No. 0410036. This work was partially supported by the National Center for Supercomputing Applications under DMS060008N and utilized the IBM pSeries 690 system copper. *Date:* 22 June 2006.

© Springer-Verlag Berlin Heidelberg 2006

$$h(\mathbf{x}, t) = \gamma(1-t)g(\mathbf{x}) + tf(\mathbf{x}) = \mathbf{0}, \quad \gamma \in \mathbb{C}, \tag{1}$$

where the random complex constant γ ensures with probability one that all solution of $h(\mathbf{x}, t) = \mathbf{0}$ are regular for all $t \in [0, 1)$. Thanks to this regularity, predictor-corrector methods can track all solution paths defined by $h(\mathbf{x}(t), t) = 0$, as t moves from 0 to 1, starting at $t = 0$ at the solution of $g(\mathbf{x}) = \mathbf{0}$ and ending at $t = 1$, at approximate isolated solutions of $f(\mathbf{x}) = \mathbf{0}$.

We say that the system f is *large* if the homotopy we use to solve it requires more than 100,000 solutions to track. Although large does not always automatically imply "difficult", numerical problems are more likely to occur. This paper is concerned with three issues:

– For efficiency, it is undesirable to keep all solutions in main memory.
– Numerical stabilities may occur as dimensions and degrees grow.
– Quality control on the computed solutions must be done fast.

Recent work produced two different software systems: PHoMpara [7] (a parallel version of PHoM [8]) and POLSYS_GLP [30] (based upon HOMPACK [39]). The first software system uses polyhedral homotopies, while the second one applies linear-product start systems to solve polynomial systems. Independent of the performance of these programs relative to PHCpack, it matters that PHCpack [33] offers both types of homotopies.

The parallel implementation of the path trackers in PHCpack started in a joint work with Yusong Wang [36] and yielded a parallel version of the Pieri homotopies [10] (refined in [12], see also [18]). Parallel path tracking is discussed in [1], [4], [5], [9], [20], and [21]. Our computational experiments showed that distributing all path tracking jobs at the start performs well when all paths require the same amount of work. Otherwise, dynamic load balancing is needed to achieve an optimal performance.

The parallel PHCpack project is currently continued in collaboration with Anton Leykin [16] (see also [14]) and Yan Zhuang [37]. The work in [14] reports on the parallel implementation of methods to decompose a positive dimensional solution set, using monodromy [24] and traces [25], as needed in a numerical irreducible decomposition [23]. The techniques presented in this paper provide efficient homotopies to create *witness sets* of these positive dimensional solution sets, see [26] and [27] for introductions to numerical algebraic geometry. The development of parallel polyhedral homotopies (described in [37]) will increase the capabilities of PHCpack to deal with solution sets of larger degrees.

The parallel software was developed using personal cluster computers from RocketCalc and ported to similar Beowulf clusters like UIC's supercomputer argo. Most recently, the parallel path tracking facilities of PHCpack were installed on NCSA's IBM pSeries 690 system running AIX 5.3. The use of MPI and a description of other interfaces to PHCpack can be found in [15].

In this paper we resolve the three issues raised above. To avoid the storage of all start solutions in the main memory, we propose to *jumpstart* homotopies, either by computing the roots whenever and wherever they are needed, or by reading the start solutions from file. For the sparsest class of polynomial systems, we discovered a numerically stable solver which computes the magnitudes of the roots separately, avoiding numerical overflow or underflow. Thirdly, for efficient quality control of the results, the programs are allowed only one linear sweep through the file which contain the solutions.

As noted before, what we call "large" does not automatically imply "difficult". The homotopies we consider in this paper are *optimal* (a notion introduced in [10]): no solution path diverges to infinity, so the overall cost of the solver is polynomial in the output size.

2 Jumpstarting Homotopies

Homotopy continuation methods compute one solution at a time. Keeping all start solutions in main memory may decrease the overall performance, or even be impossible. Assuming a manager/worker protocol, our solution is the following:

1. The manager reads a start solution from file "just in time" whenever a worker needs another path tracking job.
2. For total degree and linear-product start systems, it is simple to compute the solutions whenever needed.
3. As soon as a worker reports the end of a solution path back to the manager, the solution is written to a file.

Solutions to total degree start systems can be computed very fast, faster than they can be retrieved from file. A lexicographical indexing scheme allows the manager to dictate only which node has to track which path. As all nodes know how to solve total degree start systems, they only need a number, reducing the communication overhead.

For example, a typical total degree start system may look like

$$g(x_1, x_2, x_3) = \begin{cases} x_1^4 - 1 = 0 \\ x_2^5 - 1 = 0 \\ x_3^3 - 1 = 0. \end{cases} \tag{2}$$

It has $4 \times 5 \times 3 = 60$ solutions.

We can get the 25th solution via a decomposition of 24 (start counting from 0): $24 = 1(5 \times 3) + 3(3) + 0$. Let us verify this via lexicographic enumeration:

$$\begin{array}{l} 000 \to 001 \to 002 \to 010 \to 011 \to 012 \to 020 \to 021 \to 022 \to 030 \to 031 \to 032 \to 040 \to 041 \to 042 \\ 100 \to 101 \to 102 \to 110 \to 111 \to 112 \to 120 \to 121 \to 122 \to \boxed{130} \to 131 \to 132 \to 140 \to 141 \to 142 \\ 200 \to 201 \to 202 \to 210 \to 211 \to 212 \to 220 \to 221 \to 222 \to 230 \to 231 \to 232 \to 240 \to 241 \to 242 \\ 300 \to 301 \to 302 \to 310 \to 311 \to 312 \to 320 \to 321 \to 322 \to 330 \to 331 \to 332 \to 340 \to 341 \to 342 \end{array} \tag{3}$$

Although examples for which the total degree homotopy is optimal are fairly rare, one interesting application appears in magnetism, posed by Shigetoshi Katsura [13], see also [3]. This applications leads to a family of systems, which scales for up to any number of equations and variables. All equations in the systems are of degree two, except for one linear equation. The largest polynomial system in this family we considered has 21 equations and a total degree of $2^{20} = 1,048,576$. Recently, the `mpi2track` function in PHCpack tracked all 1,048,576 paths using a personal cluster of fourteen CPUs all running at a clockspeed of 2.4Ghz, in a traditional manager/worker dynamic load distribution model. It took 32 hours and 44 minutes to complete the tracking, leading to an output file of 1.3Gb.

While homotopies based on the total degree are rarely efficient, the simple principle of lexicographic enumeration of the start solutions applies to linear-product start systems. These start systems first occurred in [38], using multi-homogeneous homotopies [19] and were generalized in [34]. Compared to the total degree, homotopies using linear-product start systems typically follow far fewer solution paths than the total degree.

All equations in a linear-product start system are products of linear equations, of the form of the system in (4). Every $\cdots$ in (4) corresponds to a linear polynomial with randomly chosen coefficients.

$$g(\mathbf{x}) = \left\{ \begin{array}{c} (\cdots)\cdot(\cdots)\cdot(\cdots)\cdot(\cdots) = 0 \\ (\cdots)\cdot(\cdots)\cdot(\cdots)\cdot(\cdots)\cdot(\cdots) = 0 \\ (\cdots)\cdot(\cdots)\cdot(\cdots) = 0 \end{array} \right. \tag{4}$$

The random choice of the coefficients of the linear factors of the products in the linear-product start systems implies that the maximal number of isolated solutions is attained. Moreover, if every monomial in the target system $f(\mathbf{x}) = \mathbf{0}$ also occurs in the corresponding equation of the start system $g(\mathbf{x}) = \mathbf{0}$, then all isolated solutions of $f(\mathbf{x}) = \mathbf{0}$ lie at the end of some solution path defined by a homotopy using a linear-product start system $g(\mathbf{x}) = \mathbf{0}$, see [34]. Efficient implementations of this type of homotopies are described in [40] and [30].

Just like (2), the solution of the start system in (4) can be enumerated lexicographically. As the linear-product start system is stored on file in its product form, one does not need storing the start solutions on file. Moreover, any node in a parallel computer can solve for one particular solution. While the main motivation is to avoid to store the complete list of start solutions in main memory, an additional advantage is a reduced communication overhead: instead of passing the start solution vector from manager to path tracking worker, the manager simply has to pass out the label (or group of labels) to the nodes.

While there are as many candidates as the total degree, the number of start solutions (and the corresponding generalized Bézout number) is typically much less than the total degree. For efficiency – as the sequential root counting procedures in PHCpack already do – an incremental LU factorization of the coefficient matrices for each linear system leading to a start solution is an effective technique to prune the tree of all possible combinations of factors in the products of g.

3 A Numerically Stable Solver for "Simplex" Systems

Homotopies implementing Bernshteĭn's theorem [2] are described in [35]. What we now call *polyhedral homotopies* follows from the more general treatment in [11]. In [17] these methods are explained in greater detail. Bernshteĭn showed in [2] that the mixed volume of the Newton polytopes of the polynomial system bound the number of solutions (with all variables different from zero). For systems with randomly chosen coefficients, this bound is sharp. The first stage of a polyhedral homotopy method consists in the calculation of this mixed volume, see [8] and [6] for efficient programs to perform this task.

Polyhedral homotopies require in their second stage the solution of a polynomial system with random coefficients. Choosing all complex coefficients on the unit circle in the complex plane naturally leads to a well-conditioned polynomial system. Despite this good choice of the coefficients, previous versions of our software failed for some large examples used for testing the parallel polyhedral homotopies [37].

Consider for example the 12-dimensional polynomial system below in (5). It occurs as just one of the one of the 11,417 start systems generated by polyhedral homotopies to create a random coefficient start system occuring in the design of a robot (see [28], [29], [31], [32]):

$$\left\{\begin{array}{c} b_1 x_5 x_8 + b_2 x_6 x_9 = 0 \\ b_3 x_2^2 + b_4 = 0 \\ b_5 x_1 x_4 + b_6 x_2 x_5 = 0 \\ c_1^{(k)} x_1 x_4 x_7 x_{12} + c_2^{(k)} x_1 x_6 x_{10}^2 + c_3^{(k)} x_2 x_4 x_8 x_{10} + c_4^{(k)} x_2 x_4 x_{11}^2 \\ + c_5^{(k)} x_2 x_6 x_8 x_{11} + c_6^{(k)} x_3 x_4 x_9 x_{10} + c_7^{(k)} x_4^2 x_{12}^2 + c_8^{(k)} x_3 x_6 \\ + c_9^{(k)} x_4^2 + c_{10}^{(k)} x_9 = 0, \quad k = 1, 2, \ldots, 9. \end{array}\right. \tag{5}$$

The coefficients b_i, $i = 1, 2, \ldots, 6$, and $c_j^{(k)}$, $j = 1, 2, \ldots, 9$, $k = 1, 2, \ldots, 9$ are randomly chosen complex numbers, chosen so that $|b_i| = 1$ and $|c_j^{(k)}| = 1$. Because of this good choice of coefficients, all solutions are well conditioned. Despite the high degrees, there are only one hundred isolated solutions in $(\mathbb{C}^*)^{12}$, $\mathbb{C}^* = \mathbb{C} \setminus \{0\}$, because of the sparsity of the system: only 13 distinct monomials (after appropriate division).

We call such system a *simplex system*[1] and we can solve it fast, reducing it to *binomial system* using LU factorization on the coefficient matrix of the system. Every equation in a binomial system has exactly two monomials with nonzero coefficients. In compact form, we denote a binomial system by $\mathbf{x}^A = \mathbf{b}$ and solve if via the Hermite normal form of A, computing a unimodular matrix M ($\det(M) = \pm 1$), so that $MA = U$, with U is an upper triangular matrix and $|\det(U)| = |\det(A)|$. Let $\mathbf{x} = \mathbf{z}^M$, then $\mathbf{x}^A = \mathbf{z}^{MA} = \mathbf{z}^U$, so we have reduced $\mathbf{x}^A = \mathbf{b}$ to $\mathbf{z}^U = \mathbf{b}$.

[1] Because its support corresponds to a simplex. We thank the referee for suggesting this catchy term.

For example, for two variables we write

$$[z_1 \quad z_2] \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix} = [b_1 \quad b_2] \quad \text{for the system} \quad \begin{cases} z_1^{u_{11}} & = b_1 \\ z_1^{u_{12}} \, z_2^{u_{22}} & = b_2 \end{cases} . \tag{6}$$

Forward substitution on the triangular system shows that there are exactly $|\det(A)|$ distinct isolated solutions, and $|b_k| = 1$ implies $|z_k| = 1$, for every solution component, $k = 1, 2, \ldots, n$. So our binomial systems are numerically very well conditioned.

A simplex system is denoted by $C\mathbf{x}^A = \mathbf{b}$, where C is some coefficient matrix. The natural approach to reduce this simplex system to a binomial system is via a LU-factorization on C. Assuming $\det(C) \neq 0$, we compute a lower and an upper triangular matrix L and U so that $C = LU$ and solve two systems:

$$\begin{array}{l} (1)\ LU\mathbf{y} = \mathbf{b}, \text{ a linear system;} \\ (2)\ \ \mathbf{x}^A = \mathbf{y}, \ \text{ a binomial system.} \end{array} \tag{7}$$

However, this algorithm is numerically unstable! Even if all coefficients for C and $\mathbf{b}$ are chosen to lie on the complex unit circle, varying magnitudes in the intermediate values for $\mathbf{y}$ do occur. High powers, in the range of 50 and over occur in the Hermite normal form for larger systems and magnify the imbalance between the magnitudes in $\mathbf{y}$ up to the point where numerical underflow or overflow crashes the solver.

Our new solver separates the magnitudes of the solutions from their phases. Using the following notations $\mathbf{z} = |\mathbf{z}|\mathbf{e}_\mathbf{z}$, $\mathbf{e}_\mathbf{z} = \exp(i\theta_\mathbf{z})$, $\mathbf{y} = |\mathbf{y}|\mathbf{e}_\mathbf{y}$, $\mathbf{e}_\mathbf{y} = \exp(i\theta_\mathbf{y})$, $i = \sqrt{-1}$, we rewrite the binomial system and solve

$$\mathbf{z}^U = \mathbf{y} : |\mathbf{z}|^U \mathbf{e}_\mathbf{z}^U = |\mathbf{y}|\mathbf{e}_\mathbf{y} \Leftrightarrow \begin{cases} \mathbf{e}_\mathbf{z}^U = \mathbf{e}_\mathbf{y} \\ |\mathbf{z}|^U = |\mathbf{y}| \end{cases} \tag{8}$$

The first binomial system $\mathbf{e}_z^U = \mathbf{e}_\mathbf{y}$ is well conditioned because all components of the right hand side vector have modulus one. To find the magnitudes $|\mathbf{z}|$ we solve $|\mathbf{z}|^U = |\mathbf{y}|$, using a logarithmic scale, i.e.: $U \log(|\mathbf{z}|) = \log(|\mathbf{y}|)$. Even as the magnitude of the values $\mathbf{y}$ may be extreme, $\log(|\mathbf{y}|)$ will be modest in size.

Our new numerically stable solver to solve a simplex system $C\mathbf{x}^A = \mathbf{b}$ executes the following steps:

1. The LU factorization of C yields $\mathbf{x}^A = \mathbf{y}$, where $C\mathbf{y} = \mathbf{b}$.
2. Use the Hermite normal form of A, $MA = U$, $\det(M) = \pm 1$, to solve the binomial system $\mathbf{e}_\mathbf{z}^U = \mathbf{e}_\mathbf{y}$, $\mathbf{z} = |\mathbf{z}|\mathbf{e}_\mathbf{z}$, $\mathbf{y} = |\mathbf{y}|\mathbf{e}_\mathbf{y}$.
3. Solve the upper triangular linear system $U \log(|\mathbf{z}|) = \log(|\mathbf{y}|)$.
4. Compute the magnitude of $\mathbf{x} = \mathbf{z}^M$ via $\log(|\mathbf{x}|) = M \log(|\mathbf{z}|)$.
5. As $|\mathbf{e}_\mathbf{z}| = 1$, let $\mathbf{e}_\mathbf{x} = \mathbf{e}_\mathbf{z}^M$.

Even as $\mathbf{z}$ may be extreme, causing floating point overflow or underflow, we deal with $|\mathbf{z}|$ at a logarithmic scale and never raise small or large numbers to high powers. Only at the very end do we calculate $|\mathbf{x}| = 10^{\log(|\mathbf{x}|)}$ and $\mathbf{x} = |\mathbf{x}|\mathbf{e}_\mathbf{x}$.

For more on the parallel implementation of polyhedral homotopy methods in PHCpack, we refer to [37].

4 Scanning Solution Files into Frequency Tables

During runtime, we often want to monitor the progress of a large path tracking job, and get an impression about the "quality" of the solutions which have been already computed, but again, we do not want store all solutions in main memory. For each solution at the end of path, Newton's method reports three floating point numbers:

1. the magnitude of the last update to the solution vector;
2. an estimate for the inverse condition number of the Jacobian matrix at the solution;
3. the magnitude of the residual.

These three numbers determine the quality of a solution.

To determine the overall quality of the list of solutions, The program builds frequency tables, e.g.: counting #solutions with condition number between 10^{k-1} and 10^k, for some range of k. These frequency tables used to judge the quality of solution lists are a first step to employ so-called endgames, eventually with some reruns of the paths at tighter tolerances.

Recall the application posed by Shigetoshi Katsura [13] we mentioned in Section 2, which led to a homotopy of 2^{20} paths, leaving a file of 1.3Gb to process. Reading all solutions from file into main memory takes about 4 minutes and occupies more than 400Mb. While most modern workstations are well equipped with a large internal memory, to determine whether all solutions are distinct (no path crossing has happened) we do not need to occupy that much memory. The data compression of 400Mb into about 42Mb is by randomly projecting the solution vectors (of length 21) to the plane. The creation of a quadtree [22] (using as many levels till each leaf holds no more than 1,024 points) takes about 7 seconds and occupies about 58Mb. Sorting the leaves of the quadtree to determine path crossings takes less than a second.

Notice that the time to read all solutions from disk (4 minutes) dominates the time to create the quadtree (7 seconds).

As the quality analysis of solution lists can already be done while the lists are still incomplete, remedial action or more computationally demanding endgames (we refer to [27, Chapter 10] for an overview) will lead to extra jobs to be distributed among the worker nodes.

5 Towards High Performance Continuation ...

The polynomial systems we typically consider have a number n of variables which is relatively modest, averaging around 8 or 10. The nonlinearity results in a large number of solution paths, which we denote by R for the root count used in the homotopy. In this paper we considered R in the range of 100,000 and higher. Because $R \gg n$, several issues must be addressed to improve the performance of parallel homotopies. In particular, we avoid storing all start solutions in main memory by jumpstarting the homotopies. We discovered a numerical instability in the polyhedral homotopies which was not treated before and emphasized the need for fast quality control of large solution lists.

The “parallel PHCpack” effort has led to good speedups of running times on existing benchmark systems, essentially leaving the basic path tracking facilities and homotopy constructors intact, calling the routines in PHCpack in conjunction with message passing primitives. To solve large polynomial systems, an internal reorganization of PHCpack is needed, in an effort to turn Polynomial Homotopy Continuation into High Performance Continuation.

Acknowledgements. A first draft of this paper corresponds to a talk given by the second author at the AMS special session on Numerical Solution of Polynomial Systems held at the University of Notre Dame, 8-9 April 2006. We thank the organizers of this session, Chris Peterson and Andrew Sommese, for the opportunity to present this work. We thank the referee for valuable comments on the first draft.

References

1. D.C.S. Allison, A. Chakraborty, and L.T. Watson. Granularity issues for solving polynomial systems via globally convergent algorithms on a hypercube. *J. of Supercomputing*, 3:5–20, 1989.
2. D.N. Bernshteǐn. The number of roots of a system of equations. *Functional Anal. Appl.*, 9(3):183–185, 1975. Translated from *Funktsional. Anal. i Prilozhen.*, 9(3):1–4,1975.
3. W. Boege, R. Gebauer, and H. Kredel. Some examples for solving systems of algebraic equations by calculating groebner bases. *J. Symbolic Computation*, 2:83–98, 1986.
4. A. Chakraborty, D.C.S. Allison, C.J. Ribbens, and L.T. Watson. Note on unit tangent vector computation for homotopy curve tracking on a hypercube. *Parallel Computing*, 17(12):1385–1395, 1991.
5. A. Chakraborty, D.C.S. Allison, C.J. Ribbens, and L.T. Watson. The parallel complexity of embedding algorithms for the solution of systems of nonlinear equations. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):458–465, 1993.
6. T. Gao, T.Y. Li, and M. Wu. Algorithm 846: MixedVol: A software package for mixed volume computation. *ACM Trans. Math. Softw.*, 31(4):555–560, 2005.
7. T. Gunji, S. Kim, K. Fujisawa, and M. Kojima. PHoMpara – parallel implementation of the $\underline{\text{P}}$olyhedral $\underline{\text{H}}$omotopy continuation $\underline{\text{M}}$ethod for polynomial systems. Research report b-419, Tokyo Institute of Technology, 2005. Available via `http://www.is.titech.ac.jp/~kojima/sdp.html`.
8. T. Gunji, S. Kim, M. Kojima, A. Takeda, K. Fujisawa, and T. Mizutani. PHoM – a polyhedral homotopy continuation method for polynomial systems. *Computing*, 73(4):55–77, 2004.
9. S. Harimoto and L.T. Watson. The granularity of homotopy algorithms for polynomial systems of equations. In G. Rodrigue, editor, *Parallel processing for scientific computing*, pages 115–120. SIAM, 1989.
10. B. Huber, F. Sottile, and B. Sturmfels. Numerical Schubert calculus. *J. Symbolic Computation*, 26(6):767–788, 1998.

11. B. Huber and B. Sturmfels. A polyhedral method for solving sparse polynomial systems. *Math. Comp.*, 64(212):1541–1555, 1995.
12. B. Huber and J. Verschelde. Pieri homotopies for problems in enumerative geometry applied to pole placement in linear systems control. *SIAM J. Control Optim.*, 38(4):1265–1287, 2000.
13. S. Katsura. Users posing problems to PoSSO. In the PoSSO Newsletter, no. 2, July 1994, edited by L. Gonzelez-Vega and T. Recio.
14. A. Leykin and J. Verschelde. Decomposing solution sets of polynomial systems: a new parallel monodromy breakup algorithm. Accepted for publication in *The International Journal of Computational Science and Engineering.*
15. A. Leykin and J. Verschelde. Interfacing with the numerical homotopy algorithms in phcpack. Proceedings of ICMS'06, this volume.
16. A. Leykin and J. Verschelde. Factoring solution sets of polynomial systems in parallel. In Tor Skeie and Chu-Sing Yang, editors, *Proceedings of the 2005 International Conference on Parallel Processing Workshops. 14-17 June 2005. Oslo, Norway. High Performance Scientific and Engineering Computing*, pages 173–180. IEEE Computer Society, 2005.
17. T.Y. Li. Numerical solution of polynomial systems by homotopy continuation methods. In F. Cucker, editor, *Handbook of Numerical Analysis. Volume XI. Special Volume: Foundations of Computational Mathematics*, pages 209–304. North-Holland, 2003.
18. T.Y. Li, X. Wang, and M. Wu. Numerical schubert calculus by the pieri homotopy algorithm. *SIAM J. Numer. Anal.*, 40(2):578–600, 2002.
19. A. Morgan and A. Sommese. A homotopy for solving general polynomial systems that respects m-homogeneous structures. *Appl. Math. Comput.*, 24(2):101–113, 1987.
20. A.P. Morgan and L.T. Watson. A globally convergent parallel algorithm for zeros of polynomial systems. *Nonlinear Analysis*, 13(11):1339–1350, 1989.
21. W. Pelz and L.T. Watson. Message length effects for solving polynomial systems on a hypercube. *Parallel Computing*, 10(2):161–176, 1989.
22. H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2), 1984.
23. A.J. Sommese, J. Verschelde, and C.W. Wampler. Numerical decomposition of the solution sets of polynomial systems into irreducible components. *SIAM J. Numer. Anal.*, 38(6):2022–2046, 2001.
24. A.J. Sommese, J. Verschelde, and C.W. Wampler. Using monodromy to decompose solution sets of polynomial systems into irreducible components. In C. Ciliberto, F. Hirzebruch, R. Miranda, and M. Teicher, editors, *Application of Algebraic Geometry to Coding Theory, Physics and Computation*, pages 297–315. Kluwer Academic Publishers, 2001. Proceedings of a NATO Conference, February 25 - March 1, 2001, Eilat, Israel.
25. A.J. Sommese, J. Verschelde, and C.W. Wampler. Symmetric functions applied to decomposing solution sets of polynomial systems. *SIAM J. Numer. Anal.*, 40(6):2026–2046, 2002.
26. A.J. Sommese, J. Verschelde, and C.W. Wampler. Introduction to numerical algebraic geometry. In *Solving Polynomial Equations. Foundations, Algorithms and Applications*, volume 14 of *Algorithms and Computation in Mathematics*, pages 301–337. Springer–Verlag, 2005.
27. A.J. Sommese and C.W. Wampler. *The Numerical solution of systems of polynomials arising in engineering and science.* World Scientific, 2005.

28. H.-J. Su. *Computer-Aided Constrained Robot Design Using Mechanism Synthesis Theory.* PhD thesis, University of California, Irvine, 2004.
29. H.-J. Su and J.M. McCarthy. Kinematic synthesis of RPS serial chains. In the Proceedings of the ASME Design Engineering Technical Conferences (CDROM), Chicago, IL, Sep 2-6, 2003.
30. H.-J. Su, J.M. McCarthy, M. Sosonkina, and L.T. Watson. Algorithm 8xx: POLSYS_GLP: A parallel general linear product homotopy code for solving polynomial systems of equations. To appear in *ACM Trans. Math. Softw.*
31. H.-J. Su, J.M. McCarthy, and L.T. Watson. Generalized linear product homotopy algorithms and the computation of reachable surfaces. *ASME Journal of Information and Computer Sciences in Engineering*, 4(3):226–234, 2004.
32. H.-J. Su, C.W. Wampler, and J.M. McCarthy. Geometric design of cylindric PRS serial chains. *ASME Journal of Mechanical Design*, 126(2):269–277, 2004.
33. J. Verschelde. Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Trans. Math. Softw.*, 25(2):251–276, 1999. Software available at `http://www.math.uic.edu/~jan`.
34. J. Verschelde and R. Cools. Symbolic homotopy construction. *Applicable Algebra in Engineering, Communication and Computing*, 4(3):169–183, 1993.
35. J. Verschelde, P. Verlinden, and R. Cools. Homotopies exploiting Newton polytopes for solving sparse polynomial systems. *SIAM J. Numer. Anal.*, 31(3):915–930, 1994.
36. J. Verschelde and Y. Wang. Computing feedback laws for linear systems with a parallel Pieri homotopy. In Y. Yang, editor, *Proceedings of the 2004 International Conference on Parallel Processing Workshops, 15-18 August 2004, Montreal, Quebec, Canada. High Performance Scientific and Engineering Computing*, pages 222–229. IEEE Computer Society, 2004.
37. J. Verschelde and Y. Zhuang. Parallel implementation of the polyhedral homotopy method. To appear in the proceedings of *The 8th Workshop on High Performance Scientific and Engineering Computing (HPSEC-06)*, Columbus, Ohio, USA, August 18, 2006.
38. C.W. Wampler, A.P. Morgan, and A.J. Sommese. Numerical continuation methods for solving polynomial systems arising in kinematics. *ASME J. of Mechanical Design*, 112(1):59–68, 1990.
39. L.T. Watson, S.C. Billups, and A.P. Morgan. Algorithm 652: HOMPACK: a suite of codes for globally convergent homotopy algorithms. *ACM Trans. Math. Softw.*, 13(3):281–310, 1987.
40. S.M. Wise, A.J. Sommese, and L.T. Watson. Algorithm 801: POLSYS_PLP: a partitioned linear product homotopy code for solving polynomial systems of equations. *ACM Trans. Math. Softw.*, 26(1):176–200, 2000.

DEpthLAUNAY[⋆]

Manuel Abellanas[1] and Alfredo de las Vegas[2]

[1] Universidad Politécnica de Madrid, Facultad de Informática
mabellanas@fi.upm.es
http://www.dma.fi.upm.es/mabellanas
[2] Universidad Politécnica de Madrid, Facultad de Informática
alfredodelasvegas@gmail.com

Abstract. This paper describes the software DEpthLAUNAY. The main goal of the application is to compute Delaunay depth layers and levels of a planar point set [ACH]. Some other geometric structures can be computed as well (convex hull, convex layers and levels, Voronoi diagram and Voronoi levels, Delaunay triangulation, Delaunay empty circles, etc.) The application has been developed using CGAL [CGAL].

1 Introduction

In bivariate data analysis several notions of depth are used in order to classify and measure the centrality of data. Most of them are closely related to geometric structures and there are Computational Geometry algorithms to compute them efficiently. Convex hulls and convex layers are well known tools used for peeling outliers and for computing central elements of a set. Let $CH(S)$ be the convex hull of the set S. The *convex depth* of $p \in S$ with respect to the set S, is defined recursively as follows: if $p \in CH(S)$, $d_S^C(p) = 1$, else $d_S^C(p) = d_{S \setminus CH(S)}^C(p) + 1$.

The Delaunay triangulation is a structure that has been recently used for computing central elements and measuring the depth of points in a given set of points [ACH], [HRSS]. The *Delaunay depth* of p, $d_S^D(p)$, is defined to be $d+1$ when the graph theoretical distance from p to $CH(S)$ in the Delaunay triangulation $DT(S \cup \{p\})$ of $S \cup \{p\}$ is d.

The Delaunay layers of a set S are the subgraphs of the Delaunay triangulation of S induced by the subsets of S formed by points with the same Delaunay depth.

The Delaunay levels determined by a set S are the regions of the plane formed by points with the same depth with respect to S.

The software we are presenting here has been developed with three objectives: As a teaching tool, as a help in the research of combinatorial aspects of Delaunay depth and as an efficient tool for computing Delaunay depth layers and levels in practice.

DEpthLAUNAY has been developed using CGAL library. Several ways of input and output data have been implemented as well as the possibility of modifying data on line. The graphic output is one of the most valuable aspects of the application, allowing the user to interact and visualize the results. DEpthLAUNAY is freely available [Dpth].

[⋆] Partially supported by grants TIC2003-08933-C02-01 and CAM S0505/DPI/023.

© Springer-Verlag Berlin Heidelberg 2006

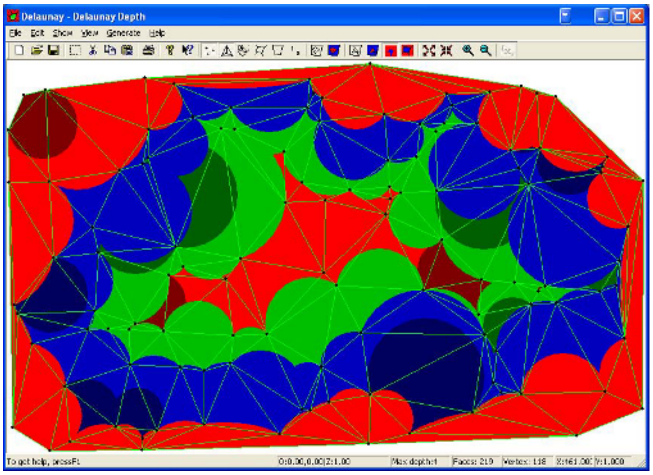


Fig. 1. View of application

2 Description of the Application

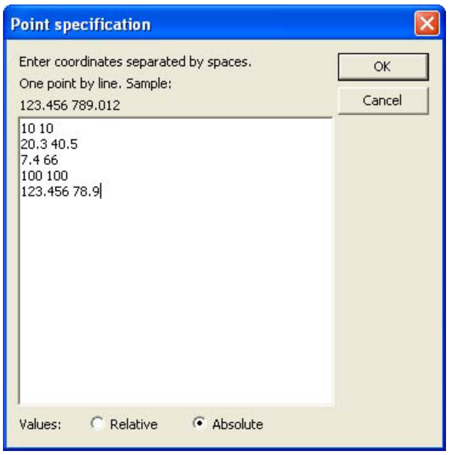
The aim of this application is to compute the Delaunay depth of a point with respect to a set of points, and some structures derived from Delaunay Depth as Delaunay Layers and Levels. Other well-known geometrical structures, like the Delaunay Triangulation, Voronoi Diagram, Convex Hull and Convex Hull Layers and levels can also be computed.

The program is developed using CGAL (Computational Geometry Algorithms Library) [CGAL]. In particular it uses its functions and objects to construct the Delaunay Triangulation of a point set in the plane. From the point of view of the programmer, the software has three layers of code:

– The inner is CGAL, a robust library that triangulates the points.
– The intermediate layer is an object named CNube, whose aim is to create a set of functions such that a programmer could use CGAL without knowing that library, using only CNube. As CGAL, CNube is independent of the operating system.
– The external layer is the part that shows the information in the screen. This layer depends on the operating system. At the moment it uses Microsoft Windows.

The program has been developed in C++ with OOP, using Microsoft Visual Studio .NET and Microsoft Windows XP as platforms.

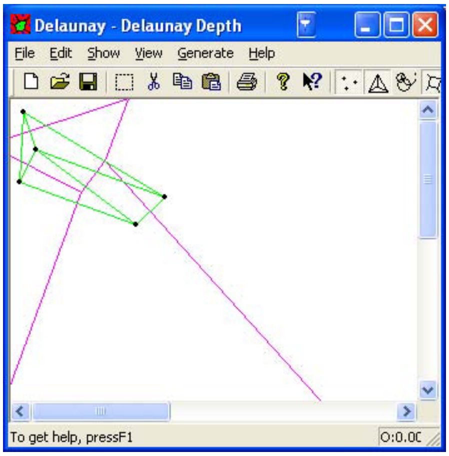
From the user's point of view, the software can be divided into three parts: the input of the data (points), the interaction and the output of the computed structures. Probably the most interesting part is the interaction, because computation and rendering of the triangulation, the Voronoi diagram, etc. are done in real time.



Enter coordinates separated by spaces.
One point by line. Sample:
123.456 789.012

```
10 10
20.3 40.5
7.4 66
100 100
123.456 78.9
```

Values: ( ) Relative [✓ Absolute]



Inserting points with absolute values Result

Fig. 2. Points inserted from keyboard

2.1 Entering Data

The easiest way to input points in the application is by using the mouse to point-and-click with the left button anywhere in the canvas. This way, the application introduces a point where the user desires. Points can be moved with the mouse while the left button is pressed. The right button will delete the point clicked. Points can also be added by typesetting their coordinates. These coordinates can be inserted in two ways (see figure 2 left):

– With approximate coordinates, for cases in which the user only want to test some properties of the points.
– With exact coordinates, for the cases in which the user needs to obtain an exact result.

Another input option is the automatic generation of points, both random and regular distributions. The random generated points can be constrained to a circle, a square or over a text. These random points can also be generated all along the canvas. The user can select the number of points to be generated. The regular distributions of points can be generated over a circumference, a spiral, a square, inside a text or conforming a mesh with the points arranged as squares, triangles or circles.

Another interesting method for adding points, connected to real world applications, is using a digital picture. A digital image can be loaded and, after selecting a color and a size for points, the application will look for groups of pixels with that color, and will convert them into coordinates of points, depending on the selected size. Figures 4, 5 and 6 show the selection of color and size of points; the generated set over the image; and the point set without the image.

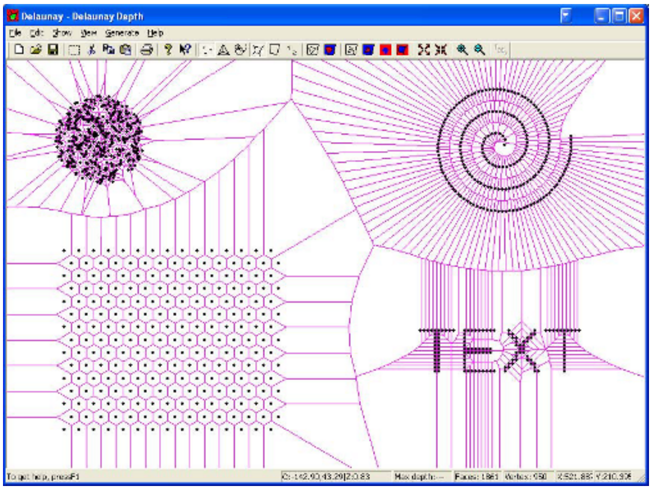


Fig. 3. Some points generated with DEpthLAUNAY

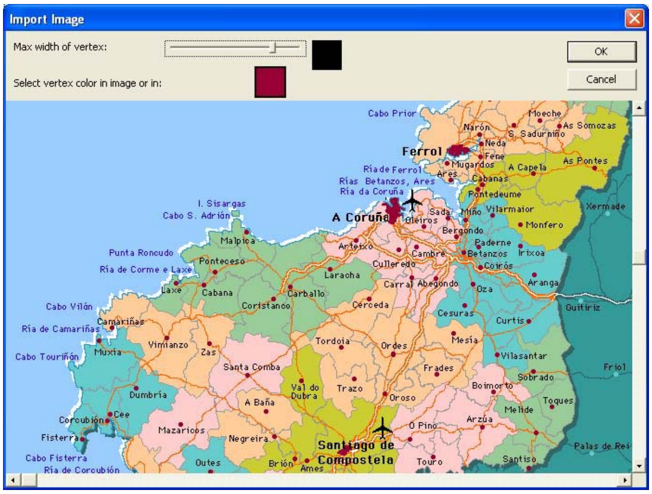


Fig. 4. Window to select color and size of points

2.2 Manipulating the Data

Once the data has been entered, the user can manipulate these points, basically inserting points, moving points or deleting points. All these operations are done in real time, allowing the user to see the result of the operation in real time.

The user can also manipulate groups of points using the clipboard and selections. Selections are made by capturing the points inside a rectangle . Once selected, the user can delete, cut or copy them to the clipboard. Once in the clipboard (like simple text), these points can be inserted to the program, and the

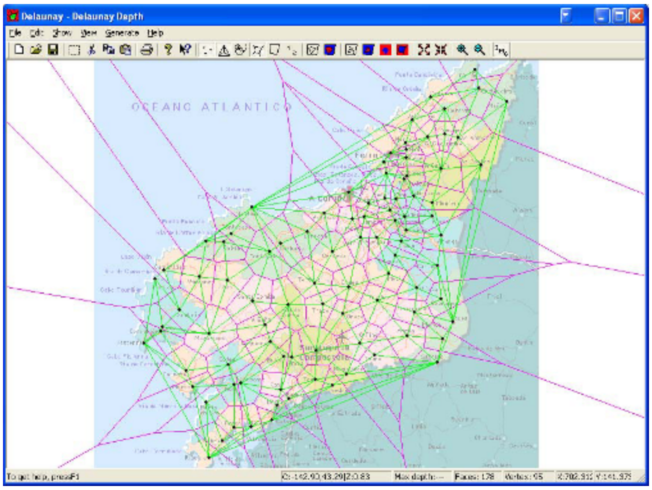


Fig. 5. Points extracted from the image, with the image in the background

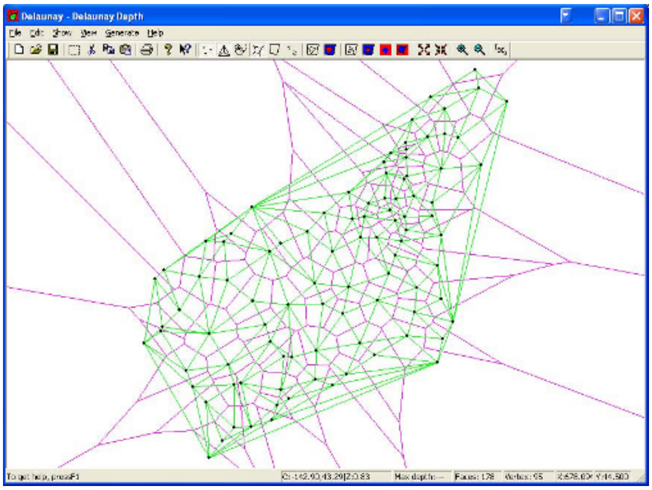
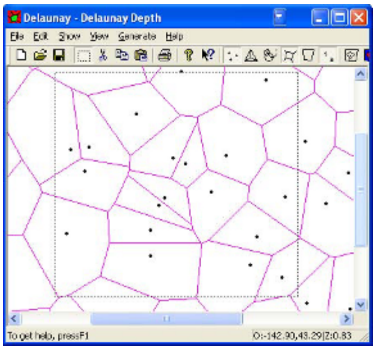


Fig. 6. Points extracted from the image

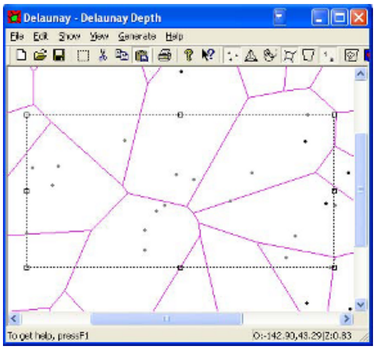
user can resize the pasting rectangle. Figure 7 shows a selection and a pasting of that selection from the clipboard.

2.3 Computing Structures

This software is mainly created to compute the Delaunay depth, but it is also capable of computing different diagrams and structures of the input set of points. These structures are (some of them can be used together):



Selected group of points



Pasting the selection from the clipboard

Fig. 7. Select and paste a group of points

- Vertices (See fig. 8)
- Delaunay triangulation (See fig. 8)
- Voronoi diagram (See fig. 8)
- Delaunay empty circles (See fig. 9)
- Convex hull (See fig. 8)
- Depth of every point (See fig. 8)
- Convex depth (See fig. 10)
- Convex layers (See fig. 10)
- Delaunay levels (See fig. 11)
- Delaunay layers (See fig. 12)
- Delaunay triangles colored depending on the depth of its vertices (See fig. 13)
- Voronoi diagram colored depending on the depth of its sites (See fig. 14)

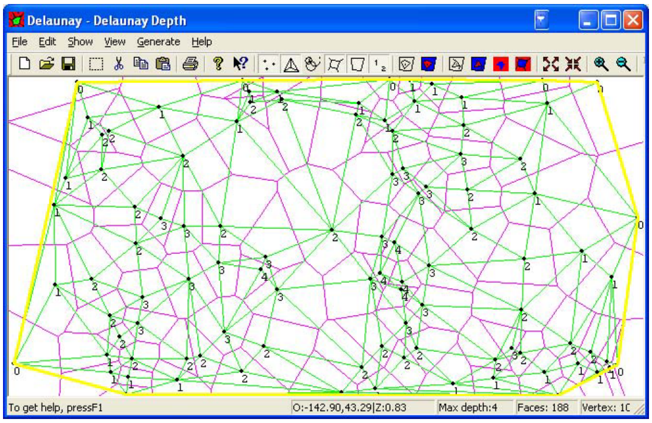


Fig. 8. Showing points, DT, VD, CH and depths

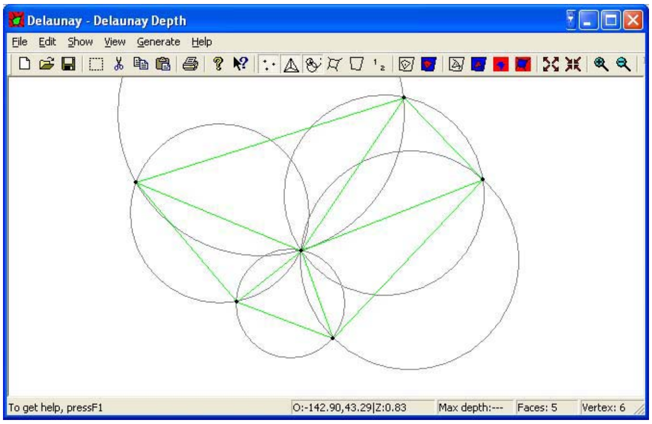


Fig. 9. Delaunay circles

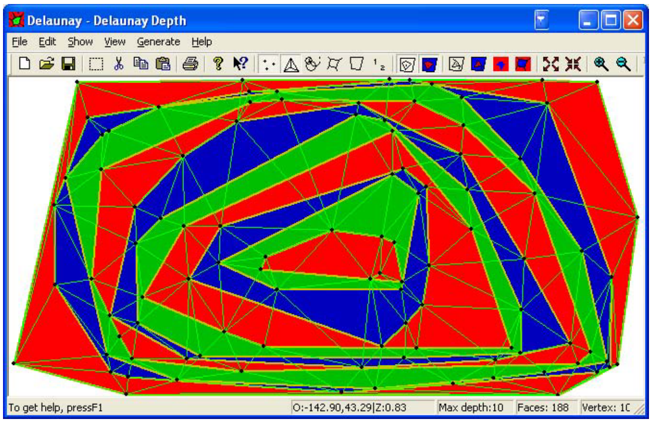


Fig. 10. Showing convex depth and levels

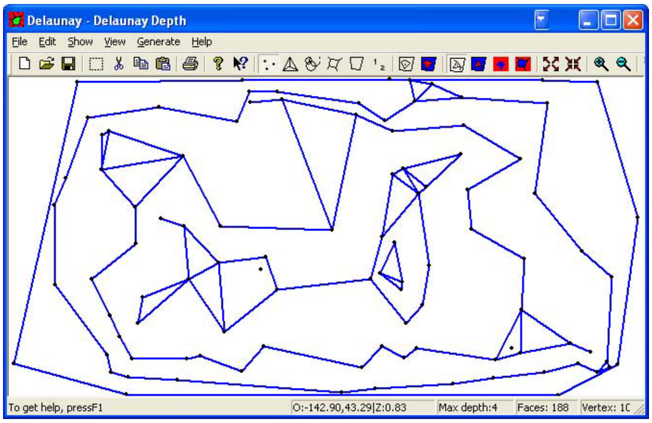


Fig. 11. Showing Delaunay layers

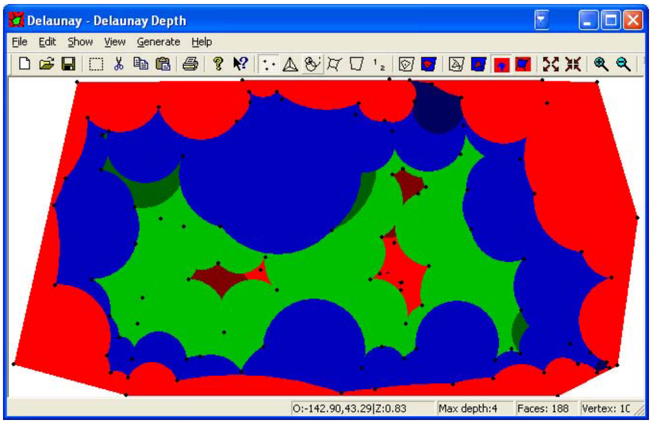


Fig. 12. Showing Delaunay levels

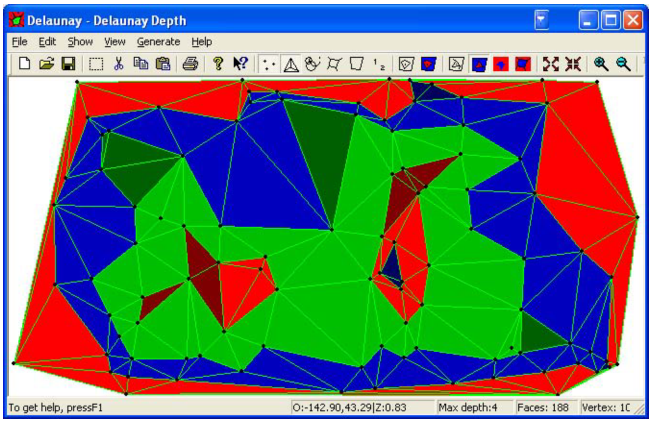


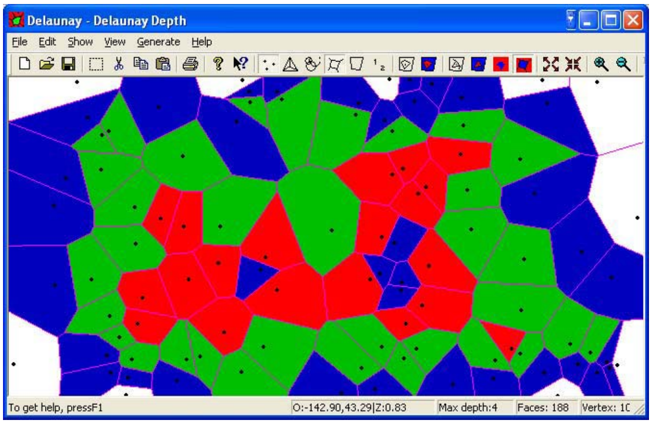
Fig. 13. Delaunay triangles colored depending on the depth of its vertices

The software can also zoom and unzoom part of the image, center the point set in the screen, and increase or decrease the size of the canvas, showing scroll bars when needed.

2.4 Exporting Data

The input points as well as some of the computed structures can be exported to files in several formats. This allows the user to process the data with other programs. The output formats are:

PNT: A simple list of points, with no triangulation information.
VTC: The format used by CGAL. Any program using CGAL can read it.
NIV: This format includes all the information about triangulation, depth levels and Voronoi diagram.



To get help, pressF1 | O:-142.90,43.29|Z:0.83 | Max depth:4 | Faces: 188 | Vertex: 1C

Fig. 14. Voronoi regions colored depending on the depth of sites

OFF: A 3D representation of depth, in which the height of a point corresponds to its depth in Geomview's format.

If the user prefers a graphical output, the software provides a snapshot of viewed data in several graphic formats.

3 Conclusions

DEpthLAUNAY was originally intended as a tool for statistical analysis of bivariate data, looking for a way to classify the points with respect to their Delaunay depth. This study can be seen in [ACH].

While the project was growing, its scope also grew, turning it into a valuable tool to investigate the properties of Delaunay triangulations, Delaunay depths and Voronoi diagrams. Non expert users have found it very easy to insert, move and delete points, and the online response of the program, even with a big number of points (a current PC can handle over 1000 points easily).

Due to the possibility to handle graphics as a way of obtaining input points, this program connects with real world. It can be used by non mathematicians as a helping tool in decision making. It is well known the relations between Voronoi diagrams and Delaunay triangulations and facility location problems.

This is an ongoing work and more options are going to be added to the application.

Acknowledgements

Authors thanks Belén Palop and the anonymous referees their suggestions for improving this paper. This work has been partially supported by grants TIC2003-08933-C02-01 and CAM S-0505/DPI/023.

References

[ACH] M. Abellanas, M. Claverol, F. Hurtado. Point set stratification and Delaunay depth (http://arxiv.org/abs/cs/0505017) to appear in Computational Statistics and Data Analysis, Elsevier.

[CGAL] Computational Geometry Algorithms Library, http://www.CGAL.org

[Dpth] DEpthLAUNAY, http://www.dma.fi.upm.es/mabellanas/delonedepth/

[HRSS] J. Hugg, E. Rafalin, K. Seyboth, D. Souvaine. An Experimental Study of Old and New Depth Measures, Workshop on Algorithm Engineering and Experiments (ALENEX06) , Springer-Verlag Lecture Notes in Computer Science , 2006.

iB4e: A Software Framework for Parametrizing Specialized LP Problems

Peter Huggins

University of California, Berkeley
phuggins@math.berkeley.edu

Given a polytope P, the classical linear programming (LP) problem asks us to find a point in P which attains maximal inner product with a given real objective vector c. When the objective is a vector of unknown parameters, the LP problem amounts to computing certain information about the polytope P, such as its vertices and normal fan.

In the sciences there are well-known problems which can be viewed as specialized instances of LP. One example is the pairwise sequence alignment problem in molecular biology. Often these specialized instances of classical LP admit specialized algorithms (such as the *Needleman–Wunsch algorithm* for sequence alignment [6]), under the assumption that the objective vector c is a given real vector. However, in such real-world applications, often the exact value of the objective vector is not known, and so an estimate is used instead. This raises some serious questions: If the value of the objective vector is changed slightly, will the optimal solution of the LP instance change drastically? Will that, in turn, also change a *qualitative* inferrence made from the obtained optimal solution? We might also want to know the set of all optimal solutions.

We could answer such questions if we knew the vertices and normal fan of the polytope for the LP instance. This leads us to ask: *Given a black box subroutine, which solves an LP instance for a given value of the objective vector c, can we compute the vertices and normal fan of the entire polytope for the LP instance?*

The answer is yes, and the algoirthm we use is essentially the *Beneath/Beyond (BB) method* [1, §3.4.2]. The algorithm builds the polytope incrementally, by systematically finding new vertices and facets. It does this by repeatedly choosing an objective vector c, and calling the black box subroutine to find a vertex of the polytope which solves the LP instance with objective c. The objective vectors are chosen so that after each iteration, either a new vertex of the polytope is found, or a new facet; hence the alogirthm requires no more than $O(V+F)$ calls to the black box LP solver, if the polytope has V vertices and F facets.

Our main contribution is the C++ library *iB4e*, which implements this Beneath/Beyond algorithm as part of an abstract base class *BBPolytope* with virtual member function *blackboxOptimize(c)*. For sequence alignment, *blackboxOptimize(c)* would be written to perform the standard Needleman–Wunsch algorithm with alignment scoring parameters given by c. For computing the Minkowski sum of V-polytopes (a problem studied in [2,3]), *blackboxOptimize(c)* would return the sum of each V-polytope's vertex that attains maximal inner product with c.

© Springer-Verlag Berlin Heidelberg 2006

Once the *blackboxOptimize(c)* function is written by the user, the derived *BBPolytope* class is ready to use. Polytopes are outputted in `polymake` format [7], so that *iB4e* can be interfaced with `polymake` for further polyhedral computations.

In addition to computing the polytope for an LP instance, *iB4e* can also give detailed information about particular vertices of interest. For instance, users can perform sensitivity analysis by asking which other vertices can be obtained by perturbing an objective vector within some specified confidence interval. Also, users can specify the "correct" vertices of a collection of computed polytopes, and *iB4e* will report the cone of all objective vectors which simultaneously yield the correct vertex of each polytope. Thus, the *iB4e* library can be used to infer all possible parameter values from a trusted collection of solved examples of a specialized LP problem. (In the case of pairwise sequence alignment, this gives the complete solution to the *inverse alignment problem.*)

The *iB4e* library supports both native arithmetic and arbitrary precision arithmetic (using the NTL/GMP libraries). The software computes polytopes of arbitrary dimension, but also has various optimized subroutines for low dimensions ($d \leq 5$). Thus *iB4e* is general and robust, but can also be suitable for intensive high-throughput computations such as those reported in [4]. We have observed that the software is particularly well-suited for computing vertices and facets in low dimensions and in general when $(V + F)$ is of reasonable size. However, when d and F are large and only computing vertices is desired, other methods are more appropriate (such as in [2] for the case of Minkowski sums).

We report on recent advances in whole genome parametric alignment [4] which used a pre-release version of *iB4e*, and we compare our software's performance to a polytope propagation implementation of the Needleman–Wunsch algorithm [5]. We also report on recent advances in computing Newton polytopes of binary hidden Markov models, which used the *iB4e* software to compute Minkowski sums. We give a hands-on example of how to use the library, from coding to execution. We finish by outlining future improvements of the software, such as reducing memory overhead via localized computation and secondary storage data structures.

All *iB4e* source and binaries are freely available under the GNU General Public License, and can be downloaded at the author's web address

`http://math.berkeley.edu/~phuggins/software/.`

References

1. F Preparata and MI Shamos. *Computational Geometry: An Introduction.* Texts and Monographs in Computer Science. Springer, New York, 1985.
2. K Fukuda and C Weibel. Computing All Faces of the Minkowski Sum of V-Polytopes. *Proceedings of the 17th Canadian Conference on Computational Geometry.* 2005.
3. P Gritzmann and B Sturmfels. Minkowski addition of polytopes: computational complexity and applications to Gröbner Bases. *SIAM J. Disc. Math.* 6:246-269, 1993.

4. C Dewey, P Huggins, K Woods, B Sturmfels, and L Pachter. Parametric alignment of Drosophila genomes. *PLoS Computational Biology, to appear*. 2006.
5. L Pachter and B Sturmfels. Parametric inference for biological sequence analysis. *Proceedings of the National Academy of Sciences, USA*. 101(46):16138–43, 2004.
6. SB Needleman and CD Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–445, 1970.
7. E Gawrilow and M Joswig. Polymake: A framework for analyzing convex polytopes. In G Kalai and GM Ziegler, eds., *Polytopes—Combinatorics and Computation*, pages 43–74. Birkhäuser, 2000.

Primal-Dual Enumeration for Multiparametric Linear Programming

Colin N. Jones[1] and Jan M. Maciejowski[2]

[1] Automatic Control Laboratory, Swiss Federal Institute of Technology
Physikstrasse 3, CH-8092 Zurich, Switzerland
cjones@ee.ethz.ch

[2] Control Group, Department of Engineering, University of Cambridge
Trumpington St, Cambridge, UK
jmm@eng.cam.ac.uk

Abstract. Optimal control problems for constrained linear systems with a linear cost can be posed as multiparametric linear programs (pLPs) and solved explicitly offline. Several algorithms have recently been proposed in the literature that solve these pLPs in a fairly efficient manner, all of which have as a base operation the computation and removal of redundant constraints. For many problems, it is this redundancy elimination that requires the vast majority of the computation time. This paper introduces a new solution technique for multiparametric linear programs based on the primal–dual paradigm. The proposed approach reposes the problem as the vertex enumeration of a linearly transformed polytope and then simultaneously computes both its vertex and halfspace representations. Exploitation of the halfspace representation allows, for smaller problems, a very significant reduction in the number of redundancy elimination operations required, resulting in many cases in a much faster algorithm.

1 Introduction

It is standard practice to implement a model predictive controller (MPC) by solving an optimisation problem on–line. For example, when the system is linear, the constraints are polyhedral and the cost is linear (e.g. $1-$ or $\infty-$norm), this amounts to computing a single linear program (LP) at each sampling instant. In recent years, it has become well-known that for this class of systems the optimal input is a piecewise affine function (PWA) defined over a polyhedral partition of the feasible states. By pre–computing this PWA function off–line, the on–line calculation of the control input then becomes one of evaluating the PWA function at the current measured state, which allows for significant improvements in sampling speed [1].

The computation of the optimal PWA function, mapping the measured state to the control input, can be posed as the following (multi)parametric linear program (pLP) [1]:

$$\min_u \left\{ c^T u \mid (x,u) \in P \right\}, \tag{1}$$

© Springer-Verlag Berlin Heidelberg 2006

where $x \in \mathbb{R}^d$ is the parameter, or state, $u \in \mathbb{R}^m$ is the optimiser, or control input and slack variables and P is a polyhedron, which incorporates the system constraints and is assumed bounded.

Several methods of computing the solution to pLP (1) can be found in the literature (e.g., [1,2,3]). All of these approaches enumerate the affine regions of the optimal PWA function one at a time. For each of the affine pieces of the function, the main computational burden is the determination of a minimal description of the polytope in which it is optimal. Computing this minimal representation is a so–called *redundancy elimination* operation, which requires the solution of a number of linear programs equal to the size of the input (the number of inequalities describing the polytope P). In most cases, these redundancy elimination LPs take the vast majority total computation time.

In this paper we present a new method of computing the optimiser of pLP (1) based on a *primal-dual* approach [4]. We first show that parametric linear programming can be posed as a vertex enumeration problem of an affine transform of the dual constraints in (1). The primal-dual algorithm then computes the convex hull of this transformed polytope as it is enumerating the vertices. The availability of these two descriptions of the same polytope then allow a significant reduction in the amount of work that the algorithm is required to do by removing the need to compute a large number of the redundancy elimination LPs.

The remainder of this paper is organised as follows. Section 2 provides required background on parametric linear programming. Section 3 introduces a polytope such that there is a one-to-one mapping from its vertices to the affine pieces of the solution. The primal-dual approach described in [4] is then adapted such that it can be applied to this polytope, and hence to the associated pLP in Section 3.2. Finally, examples and conclusions are given in Sections 4 and 5 respectively.

Notation

If $A \in \mathbb{R}^{m\times n}$ and $I \subseteq \{1,\dots,n\}$, then $A_{*,I} \in \mathbb{R}^{m\times|I|}$ is the matrix formed by the columns of A indexed by I. If $c \in \mathbb{R}^n$ is a vector then c_I is the vector formed by the elements of c in I. If $R \subseteq \{1,\dots,m\}$ then we will use the notation $A_{R,*} \in \mathbb{R}^{|R|\times n}$ to denote the matrix formed by the rows of A indexed by R.

A *polyhedron* is the intersection of a finite number of halfspaces and a *polytope* is a bounded polyhedron. If $P = \{x \mid Ax \le b\}$ is a polyhedron and $H = \{x \mid a^T x \le d\}$ is a halfspace such that $P \subseteq H$, then $P \cap \{x \mid a^T x = d\}$ is a *face* of P. One– and zero–dimensional faces are called *edges* and *vertices* respectively. Faces of dimension $\dim(P) - 1$ are called *facets* and $\dim(P) - 2$, *ridges*.

A vector $r \in \mathbb{R}^d$ defines a *ray* as $R = \{r\alpha \mid \alpha > 0\}$. A set C is called a cone if for every $x \in C$ and scalar $\alpha > 0$, we have $\alpha x \in C$. The columns of a matrix $F \in \mathbb{R}^{m\times n}$ are called the *generators* of the cone $C = \operatorname{cone}(F) \triangleq \{F\alpha \mid \alpha \ge 0\}$. The generator $F_{*,i}$ is called *redundant* if $F_{*,i} \in \operatorname{cone}(F_{*,\{1,\dots,n\}\setminus\{i\}})$ and *irredundant*, or *extreme* otherwise.

The *Minkowski sum* of two sets, denoted $A \oplus B$ is defined as $A \oplus B \triangleq \{x + y \mid x \in A,\ y \in B\}$.

2 Preliminaries

2.1 Linear Programming

Consider the following linear program:

$$\max_{\lambda} \left\{ c^T \lambda \mid \lambda \in D \right\} \ , \tag{2}$$

where $\lambda \in \mathbb{R}^n$ is the optimiser, $c \in \mathbb{R}^n$ is a vector and the constraint polytope D is defined by the matrix $A \in \mathbb{R}^{m \times n}$ and the vector $b \in \mathbb{R}^m$ as

$$D \triangleq \{ \lambda \mid A\lambda = b, \ \lambda \geq 0 \} \ . \tag{3}$$

Any set $B \subset \{1, \ldots, n\}$ such that $|B| = m$ and $\operatorname{rank} A_{*,B} = m$ is called a *basis* and we write $N = \{1, \ldots, n\} \setminus B$ for its complement and call λ_B and λ_N the *basic* and *non-basic variables* respectively. Every basis B defines a *basic solution*[1] λ^B to the linear equations in (3), which is given by restricting the non-basic constraints to zero $\lambda^B_B = A^{-1}_{*,B} b, \ \lambda^B_N = 0$. A basis is called *primal feasible* if the resulting solution also satisfies the inequality constraints in (3): $A^{-1}_{*,B} b \geq 0$. Note that all solutions represented by bases occur at a vertex of the constraint polyhedron D.

2.2 Optimality Conditions

Definition 1 (Tangent cone [5]). *Let λ be an element of the polyhedron $D \subseteq \mathbb{R}^n$. A vector $\gamma \in \mathbb{R}^n$ is said to be a* feasible direction *at λ if there exists a strictly positive scalar α for which $\lambda + \alpha\gamma \in D$. The set of all feasible directions at λ is called the* tangent cone *and is written $\mathcal{T}_D(\lambda)$. The* support cone *$\mathcal{S}_D(\lambda)$ is the translation of $\mathcal{T}_D(\lambda)$ by the vector λ, $\mathcal{S}_D(\lambda) \triangleq \mathcal{T}_D(\lambda) \oplus \{\lambda\}$. Note that strictly speaking, the support cone is not a cone as it has a vertex at λ rather than zero.*

Given a basis B, the extreme feasible directions at the solution λ^B are given by increasing each non-basic variable in a feasible (positive) direction:

$$\lambda_B = \lambda^B - A^{-1}_{*,B} A_{*,i} \lambda_i, \quad \lambda_i \geq 0, \quad \lambda_{N \setminus \{i\}} = 0, \quad \forall i \in N \ .$$

The set of all convex combinations of the extreme rays give the tangent cone:

$$\mathcal{T}_D(\lambda^B) = \operatorname{cone}(F) \ , \tag{4}$$

where $F_{B,*} \triangleq -A^{-1}_{*,B} A_{*,N}$, $F_{N,*} \triangleq I$.

Theorem 1 (Optimality Condition). *Let λ be an element of the polyhedron D. A necessary and sufficient condition for λ to be a global minimum of the linear program* (2) *is $c^T \gamma \geq 0$ for all feasible directions γ at λ.*

[1] Where clear from the context, we will refer to the basic solution as simply the solution.

Definition 2. *The* normal cone *to D at λ is the orthogonal complement of the tangent cone:*

$$\mathcal{N}_D(\lambda) \triangleq \left\{ v \;\middle|\; v^T\gamma \leq 0,\ \forall \gamma \in \mathcal{T}_D(\lambda) \right\} \ .$$

From the above definition and (4), the normal cone to the basic feasible solution λ^B is:

$$\mathcal{N}_D(\lambda^B) = \left\{ v \;\middle|\; F^T v \leq 0 \right\} \ , \tag{5}$$

where F is as defined in (4). A direct result of Theorem 1 and (5) is that the basic solution λ^B, and hence the basis B, is optimal if and only if[2]

$$-c \in \mathcal{N}_D(\lambda^B) \ . \tag{6}$$

2.3 Parametric Linear Programming

The problem we will consider in this paper is the following *parametric linear program*:

$$\max_{\lambda} \left\{ x^T E\lambda \mid \lambda \in D \right\} \ , \qquad\qquad x \in \mathcal{X} \tag{7}$$

where $x \in \mathcal{X}$ is the parameter, $\mathcal{X} \subseteq \mathbb{R}^d$ and $E^T \in \mathbb{R}^{d\times n}$ is a matrix of rank d, $d < n$. It is assumed throughout this paper that the set of feasible parameters $\mathcal{X}$ is full–dimensional, which is common to most pLP algorithms [1,2,3]. This assumption can be easily guaranteed through a pre-processing operation [1]. The standard assumption is also made that pLP (7) has an optimal bounded solution for every $x \in \mathcal{X}$.[3]

Definition 3 (Critical Region). *If B is a basis of pLP* (7)*, then the* critical region $\mathscr{R}_B$ *is defined as the set of all parameters $x_\circ \in \mathcal{X}$ such that B is optimal for $x = x_\circ$.*

From (6) and (7) that the critical region $\mathscr{R}_B$ is the polyhedral set $\mathscr{R}_B = \left\{ x \;\middle|\; F^T E^T x \geq 0 \right\} \cap \mathcal{X}$.

Our goal is to enumerate all full–dimensional critical regions. In [6] it was shown that by *lexicographically* perturbing the problem (7), the following properties hold:

1. Every full–dimensional critical region is uniquely defined by a single basis
2. The interiors of the full–dimensional critical regions do not overlap
3. The union of all full–dimensional critical regions is the set of feasible parameters $\mathcal{X}$.

[2] The vector $-F^T c$ is often referred to as the reduced cost $\bar{c}$ and condition (6) then becomes $\bar{c} \geq 0$.

[3] Note that this also implies that the dual solution is feasible and bounded.

In the remainder of this paper we will assume that the problem has been lexicographically perturbed and will therefore not discuss possible degeneracy of the solution.[4]

Remark 1. The standard parametric linear program resulting from model predictive control problems has a cost of the form $(x^T E + c)\lambda$ [1], which differs from that used here by the constant c. The procedure developed in this paper can be applied to such problems with no added complexity by first *homogenizing* the cost as detailed in [6].

3 pLP as Vertex Enumeration

The following theorem demonstrates that the goal of enumerating all bases that define full–dimensional critical regions can be re–posed as a vertex enumeration problem of an affine transform of the constraint polytope D.

Theorem 2. *If B is a feasible basis of pLP (7) and λ^B is the basic solution, then B defines a full–dimensional critical region if and only if $E\lambda^B$ is a vertex of the polytope $ED \triangleq \{E\lambda \mid A\lambda = b,\ \lambda \geq 0\}$.*

Proof. pLP (7) can be re-written as $\max_z \left\{x^T z \mid z \in ED\right\}$ through the change of variable $z \triangleq E\lambda$. It follows from (6) and Definition 3 that x is in the critical region $\mathscr{R}_B$ if and only if $-x$ is in the normal cone $\mathcal{N}_{ED}(E\lambda^B)$. Finally, the normal cone of a point $E\lambda^B$ in a polytope ED is full–dimensional if and only if $E\lambda^B$ is a vertex of ED [7, Sec. 3.2].

Two vertices of a polytope are called *neighbours*, or *adjacent*, if they are contained in the same edge, or one–dimensional face of the polytope. The proposed algorithm begins at a single vertex of ED and then recursively computes neighbours until all vertices have been found. In the following section we see that the adjacent vertices of a vertex $v \in ED$ are given by the intersection of ED and the extreme rays of the support cone $\mathcal{S}_{ED}(v)$. The proposed method is outlined as Algorithm 1 below.

Remark 2. Note that the extreme rays of the tangent cone (and hence the support cone) are given directly by the normals of the irredundant inequalities describing the normal cone. The negative normal cone of ED at a vertex $E\lambda^B$ is exactly the critical region of the basis B, $\mathscr{R}_B = -\mathcal{N}_{ED}(E\lambda^B)$ and therefore determining the extreme rays of the support cone is an operation that is entirely equivalent to the redundancy elimination operations that are done to compute the facets of the critical regions in other methods, e.g. [1, 2, 3].

Algorithm 1 below is similar to others presented in the literature [1, 2, 3], although the formulation is in the dual. The main contribution of this paper is

[4] Note that our definition of a critical region differs from that generally found in the literature. However, under the assumption that the problem is non–degenerate, or equivalently, lexicographically perturbed, the two definitions are equivalent.

in Step 4, where one must determine the extreme rays of the support cone of ED at the point $E\lambda^B$. In current algorithms, this requires a redundancy elimination operation in order to determine which rays are extreme. For problems that are of interest to control and are yet small enough to be computed, this redundancy elimination requires the majority of the computation time, as is illustrated in Section 4. Section 3.2 presents a new primal–dual approach that can significantly reduce the computation time for these smaller problems.

Remark 3. Algorithm 1 can be seen as a form of *gift-wrapping* algorithm in a polar dual context (also called *pivoting algorithms*) in which the vertices are not known *apriori* but are provided by an oracle.

Algorithm 1. Parametric Linear Programming

```
Require: Basis $B_0$ of pLP (7) such that $\dim \mathscr{R}_B = d$
Ensure: All bases $B$ such that $\dim \mathscr{R}_B = d$
 1: $\mathcal{L}_{unexplored} \longleftarrow \{B\}$,  $\mathcal{L}_{discovered} \longleftarrow \{B\}$
 2: while $\mathcal{L}_{unexplored}$ is not empty do
 3:   Select any basis $B$ from $\mathcal{L}_{unexplored}$
 4:   for all extreme rays $r$ of $E\mathcal{S}_D(\lambda^B)$ do          Section 3.2
 5:     $B' \longleftarrow$ neighbour$(r, B)$                              Section 3.1
 6:     $\mathcal{L}_{unexplored} \longleftarrow \mathcal{L}_{unexplored} \cup (\{B'\} \setminus \mathcal{L}_{discovered})$
 7:     $\mathcal{L}_{discovered} \longleftarrow \mathcal{L}_{discovered} \cup \{B'\}$
 8:   end for
 9: end while
10: Return list $\mathcal{L}_{discovered}$
```

3.1 Neighbour Function

This section outlines the function `neighbour`$(\cdot,\cdot)$, which is used in Step 5 of Algorithm 1. The edges of a polytope P that intersect at a vertex $v \in P$ are given by the intersection of P with the extreme rays of the support cone at v [5]. The following lemma describes the tangent cone of a basic solution of ED, where we recall that the support cone is equal to the tangent cone shifted by v.

Lemma 1. *If λ is an element in the polytope D, then $\mathcal{T}_{ED}(E\lambda) = E\mathcal{T}_D(\lambda)$.*

Proof. From Definition 1, $\gamma \in \mathbb{R}^d$ is in $E\mathcal{T}_D(\lambda)$ if and only if there exists a scalar $\alpha > 0$ and a vector g such that

$$\gamma = Eg, \quad \lambda + \alpha g \in D \ . \tag{8}$$

By assumption, E is rank d and therefore such a g always exists for each $\gamma \in E\mathcal{T}_D(\lambda)$. Under the mapping E, (8) becomes $E\lambda + \alpha\gamma \in ED$, which is true if and only if $\gamma \in \mathcal{T}_{ED}(E\lambda)$.

Lemma 1 and (4) give a description of the tangent cone to a vertex $E\lambda^B$ of ED defined by the basis B:

$$\mathcal{T}_{ED}(E\lambda^B) = \text{cone}(EF) \ , \tag{9}$$

where $F_{B,*} \triangleq -A_{*,B}^{-1}A_{*,N}$, $F_{N,*} \triangleq I$. Note however, that not every column $EF_{*,i}$ defines an extreme ray of $\mathcal{T}_{ED}(E\lambda^B)$, as some of them may well be redundant. Determining if a ray is redundant or not requires the majority of effort during the computation of a pLP and is the main topic of this paper. A new approach to determining redundancy will be introduced in Section 3.2.

We can now define the function $B' = \texttt{neighbour}(r, B)$, where $r = \left\{E(F_{*,i}\alpha + \lambda^B) \mid \alpha \geq 0\right\}$ is an extreme ray of the support cone $\mathcal{S}_{ED}(E\lambda^B)$. The neighbour function returns the basis B' such that $E\lambda^{B'}$ is the vertex of $r \cap ED$, which is different from $E\lambda^B$. Note that there are exactly two vertices on each edge.

The following new theorem provides an efficient method of computing the basis that represents the adjacent vertex given an irredundant ray of the tangent/support cone.

Theorem 3. *If B is a feasible basis of D, $E\lambda^B$ is a vertex of ED and $r = \left\{E(F_{*,i}\alpha + \lambda^B) \mid \alpha \geq 0\right\}$ is an extreme ray of the support cone $\mathcal{S}_{ED}(E\lambda^B)$, then the adjacent vertex of $E\lambda^B$ in the direction r is the optimal basis of the LP:*

$$\max_{\lambda} \left\{(EF_{*,i})^T E\lambda \mid \lambda \in D, \ \lambda_j = 0, \ \forall j \notin \mathcal{Q}\right\} \ , \tag{10}$$

where $\mathcal{Q} \triangleq \{j \mid \exists \rho \geq 0, \ EF_{*,i} = \rho EF_{*,j}\}$ and $F_{B,*} \triangleq -A_{*,B}^{-1}A_{*,N}$, $F_{N,*} \triangleq I$.

Proof. The adjacent vertex is reached by moving along the edge $r \cap ED$ away from $E\lambda^B$. Every point $\lambda \in D$ can be written as $\lambda = \lambda^B + F\gamma$, for some $\gamma \geq 0$, where F is as defined in the statement of the theorem, because every tangent cone is a superset of the polytope [7]. Consider the column $F_{*,j}$ and the resulting ray $\lambda = \lambda^B + F_{*,j}\gamma_j$, $\gamma_j \geq 0$. Clearly, $E\lambda \in r$ if and only if there exists a $\rho \geq 0$ such that $EF_{*,j} = \rho EF_{*,i}$. Therefore, the face P of D such that $EP = r \cap ED$ is given by $P = \{\lambda \mid \lambda_i = 0, \ \forall i \notin \mathcal{Q}\} \cap D$.

The LP given in the statement of the theorem then maximises in the direction of the ray r, while restricting λ to be in the face P.

Remark 4. Note that if the set $\mathcal{Q}$ in Theorem 3 contains only one element more than the basis B, then LP (10) will compute the adjacent basis in a single simplex pivot. This is a significant improvement over current methods [1,2,3], which always require the calculation of an LP of dimension equal to that of D.

3.2 Primal-Dual Enumeration

The standard method for redundancy elimination, or determining which rays are extreme in Step 3 of Algorithm 1 requires a single linear program of dimension d

per ray [8]. Testing if the i^{th} ray of the support cone $\mathcal{S}_{ED}(E\lambda^B)$ for some vertex $E\lambda^B$ of ED is redundant can be done using the following linear program [8]:

$$\begin{aligned} J(i) = \text{minimise} \quad & (-EF_{*,i})^T x \\ \text{subject to} \quad & \big(EF_{*,\{1:i-1,i+1,n\}}\big)^T x \geq 0 \\ & (EF_{*,i})^T x \geq -1 \end{aligned} \tag{11}$$

where the ray r^i is extreme if $J(i) < 0$. Current pLP methods reported in the literature [1, 2, 3] require the solution of LP (11) for each column of F at every vertex, resulting in the computation of a very large number of linear programs. This paper seeks to reduce this requirement through a heuristic based on [4], which can significantly reduce the work required to compute the extreme rays.

The approach presented in this section is called 'primal-dual' because both the primal (vertex) and dual (halfspace) representations of ED are computed. At the q^{th} step of the algorithm, q vertices of ED will have been found. At this point, the algorithm has a list of these q vertices $\{v^1, \ldots, v^q\}$; it also stores a halfspace representation of their convex hull $\mathcal{H}^q \triangleq \{z \mid G^q z \leq g^q\} = \operatorname{conv}\{v^1, \ldots, v^q\}$. When a new vertex v^{q+1} is found, the existing description of the convex hull is first extended to include it: a new matrix G^{q+1} and vector g^{q+1} are computed such that $\mathcal{H}^{q+1} = \{z \mid G^{q+1} z \leq g^{q+1}\} = \mathcal{H}^q \cup \{v^{q+1}\}$.

We are now able to use this information to improve the efficiency of redundancy elimination. Given a basis B, we begin by writing down the known description $\mathcal{S}_{ED}(E\lambda^B) = \{E(\lambda^B + F\gamma) \mid \gamma \geq 0\}$ of the support cone at the vertex $E\lambda^B$ from (9). The goal is now to test each ray $r^i \triangleq \{E(\lambda^B + F_{*,i}\gamma_i) \mid \gamma_i \geq 0\}$ to determine if it is an extreme ray of $\mathcal{S}_{ED}(E\lambda^B)$.

We note that the polytope $\mathcal{H}^{q+1}$ is an inner approximation of the set ED. It follows that if the ray r^i intersects the interior of $\mathcal{H}^{q+1}$, then it also intersects the interior of ED and is therefore not an extreme ray of $\mathcal{S}_{ED}(ED)$. This notion is formalised in the following theorem.

Theorem 4. *Let $\mathcal{H}^q = \operatorname{conv}\{v^1, \ldots, v^q\} = \{z \mid G^q z \leq g^q\}$, where v^i are q vertices of ED and $\dim(\mathcal{H}^q) = \dim(ED)$. If $E\lambda^B$ is a vertex of ED and of $\mathcal{H}^q$, then $r^i \triangleq \{E(\lambda^B + F_{*,i}\gamma_i) \mid \gamma_i \geq 0\}$ is a redundant ray of the support cone $\mathcal{S}_{ED}(E\lambda^B)$ if for each j such that $G_{j,*}v = g_j$ the condition $G_{j,*}EF_{*,i} < 0$ holds, where F is defined as in (4).*

Proof. The test is simply to check if a point on the ray r^i is internal to $\mathcal{H}^q$ for a strictly positive γ_i:

$$\begin{aligned} GE\big(\lambda^B + F_{*,i}\gamma_i\big) &\leq g \\ GEF_{*,i}\gamma_i &\leq g - GE\lambda^B \ . \end{aligned} \tag{12}$$

Recall that $E\lambda^B$ is a vertex of $\mathcal{H}^q$ and therefore $g - GE\lambda^B \geq 0$. For those constraints that are strictly greater than zero, (12) will clearly be satisfied for some $\gamma_i > 0$ and therefore we have only to test those constraints that are equal to zero. Clearly, there exists a strictly positive α such that (12) is satisfied if and only if $G_{j,*}EF_{*,i} < 0$ for all j such that $G_{j,*}E\lambda^B = g_j$.

Theorem 4 can now be used during Step 3 of Algorithm 1 to determine which, if any, of the rays of $\mathcal{S}_{ED}(E\lambda^B)$ are redundant. As this test can only prove redundancy and not irredundancy, if the conditions of the theorem are not met, then the linear program (11) must still be solved.

Convex Hull Computation. The use of Theorem 4 requires the computation of the convex hull $\mathcal{H}^q$ of the first q discovered vertices $\{v^1, \ldots, v^q\}$ of ED. While any convex hull algorithm could be used, ideally the algorithm should be incremental, or able to add one vertex at a time, and have the ability to quickly identify which inequalities are active at the most recently added vertex.

An incremental approach takes as input a full–dimensional polytope $\mathcal{H}^{q-1} = \{z \mid G^{q-1}z \leq g^{q-1}\}$ and computes the convex hull $\mathcal{H}^q = \mathcal{H}^{q-1} \cup \{v^q\}$ for a point v^q. The facet $\mathcal{H}^{q-1} \cap \left\{z \,\middle|\, G^{q-1}_{i,*}z = g^{q-1}_i\right\}$ is called visible from v^q if its supporting hyperplane separates $\mathcal{H}^{q-1}$ and v^q (i.e. $G^{q-1}_{i,*}v^q > g^{q-1}_i$), otherwise the facet is obscured. The set of facets of $\mathcal{H}^q$ then consists of the obscured facets of $\mathcal{H}^{q-1}$ as well as a new set of facets to replace the visible ones, which include the point v^q. Updating $\mathcal{H}^{q-1}$ to $\mathcal{H}^q$ therefore consists of two subproblems: finding all visible facets of $\mathcal{H}^{q-1}$ and computing new facets to replace them. The inequalities that must be tested in Theorem 4 is then exactly the set of new facets that are computed to replace the visible ones and are therefore computed as a side effect of the convex hull algorithm.

There are two approaches available for determining the set of visible facets that can be applied in the context of this paper, where the list of points v^i for i larger than q is not available while computing $\mathcal{H}^q$. The first is to simply check each facet of $\mathcal{H}^{q-1}$ to determine if v^q is on its positive (obscured) or negative (visible) side. This is the approach used in Kallay's beneath–beyond [9] and Motzkin's double description [10] methods and requires time linear in the number of facets of $\mathcal{H}^{q-1}$. An improvement on this is to store a so–called facet graph, whose nodes are facets and arcs connect facets if they share a common ridge. The set of visible facets then forms a subgraph of the facet graph and can be efficiently enumerated in time proportional to the number of visible facets [11].

Once the visible facets are computed and removed, the supporting hyperplanes of the new facets containing the point v^q can be efficiently computed by noting that they must contain the point v^q as well as the ridges formed by the intersection of the removed facets and the obscured facets.

The reader is referred to [12] for a more complete handling of incremental convex hull algorithms.

Complexity. Current methods of computing pLPs are in a sense output sensitive, in that they require a fixed number of redundancy elimination LPs (11) to be computed per critical region of the solution (one for each column of the matrix F). The approach introduced in this paper aims to reduce the number of redundancy elimination LPs through the use of Theorem 4. However, since Theorem 4 is a sufficient condition for redundancy and not a necessary one, no guarantee can be made that any of the rays will satisfy the conditions of the theorem, and as a result it may be the case that LP (11) must still be computed for each ray, which would therefore result in no improvement over current methods in the worst–case.

The additional cost of using the primal-dual test is the calculation and storage of the halfspace description of ED. While this convex hull can be computed efficiently in an incremental fashion as outlined above, the relationship between the number of vertices in ED and the number of inequalities can be exponential. As a result the applicability of this algorithm is limited to those polyhedra ED that can be described with both relatively few inequalities and few vertices. In Section 4 it will be seen that there are problems that are of a size and structure that are interesting in a control context and which satisfy these requirements.

4 Examples

The primary motivation for pLPs in control is the calculation of so–called closed–form or explicit Model Predictive Control (MPC) laws. In standard MPC an optimisation problem, which is a function of the current state, is solved at each sampling instant, whereas in closed-form MPC the problem is posed in multi-parametric form, with the state as a parameter, and solved offline.

The goal is to regulate the linear time invariant (LTI) system $x^+ = Ax + Bu$ to the origin, where $x \in \mathbb{R}^n$ is the state, x^+ is the successor state and $u \in \mathbb{R}^m$ is the input. A standard Model Predictive Controller (MPC) can be written as the solution to the following optimisation problem, in which the optimiser u_1 is the input that is applied to the system, given the measured state x:

$$\begin{aligned} J(x) = \underset{u_1,\dots,u_{N-1},x_1,\dots,x_N}{\text{minimise}} \quad & \textstyle\sum_{i=1}^{N-1} \|Ru_i\|_p + \sum_{i=1}^{N-1} \|Qx_i\|_p + \|Q_F x_N\|_p \\ \text{subject to} \quad & x_0 = x \\ & x_{i+1} = Ax_i + Bu_i, \quad i = 0,\dots,N-1 \\ & x_{i+1} \in \mathcal{X}, \quad u_i \in \mathcal{U} \quad i = 0,\dots,N-1 \end{aligned} \tag{13}$$

where x_i and u_i are future predicted states and inputs respectively, which are constrained to be in the polytopes $\mathcal{X}$ and $\mathcal{U}$. If the norm p is taken to be either the $1-$ or $\infty-$norm, then a linear program results, which is the case of interest in this paper. Conversion of this problem to the form of pLP (1) is discussed in [1] and requires the introduction of several slack variables.

4.1 Closed-Form MPC for a 4D System

Consider the problem (13) with the following randomly generated system, which is given as an example in the MPT toolbox [13]:

$$x^+ = \begin{bmatrix} 0.7 & -0.1 & 0.0 & 0.0 \\ 0.2 & -0.5 & 0.1 & 0.0 \\ 0.0 & 0.1 & 0.1 & 0.0 \\ 0.5 & 0.0 & 0.5 & 0.5 \end{bmatrix} x + \begin{bmatrix} 0.0 & 0.1 \\ 0.1 & 1.0 \\ 0.1 & 0.0 \\ 0.0 & 0.0 \end{bmatrix} u$$

with a prediction horizon $N = 5$ and the constraints $\|u_i\|_\infty \le 1$, $\|x_i\|_\infty \le 5$ on the input and state respectively. The cost is the minimisation of the $\infty-$norm of the states and inputs at each point in time and the matrices Q, Q_F and R are taken as the identity.

When the problem is written in the form of pLP (2), (3) the matrix A is in $\mathbb{R}^{20\times 120}$ and E is in $\mathbb{R}^{4\times 120}$ and the solution contains 12,128 critical regions. While this problem may seem fairly small, there are many interesting and useful control problems of this size and there are only a very small number of applications reported in the literature in which the parameter size is larger.

The proposed approach was compared against the two main methods for computing pLPs reported in the literature. The method used in the Multiparametric Toolbox (MPT [13]) is based on a similar exploration strategy as the proposed method and solves an LP of the form (11) for every possible redundant ray. The computation of adjacent critical regions is done using a linear program of dimension equal to that of D, and is therefore less efficient than that given in Theorem 3. The second method is that implemented in the Hybrid Toolbox [14] and is based on an entirely different exploration strategy. The reader is referred to [1] for details of this method.

From Table 1 one can see that the primal–dual algorithm offers a significant reduction in the number of pivots required to compute the solution. The computation of the convex hull for this example required 71.1 seconds to compute using qhull [15]. To give an idea of speed, a 3GHz Pentium IV machine using the Stanford Systems Optimization Laboratory (SOL) toolbox [16] can execute the required pivots for the primal–dual approach in 36.6 seconds, which when added to the time to compute the convex hull totals 107.7 seconds compared to a total of 280.1 seconds for the MPT [13] approach.

Table 1. Comparison of pLP Methods for Example 4.1

Method	**Simplex Pivots**	
	$\mathbb{R}^4$	$\mathbb{R}^{20}$
Primal–Dual	761,487	76,488
MPT [13]	6,409,503	670,940
Hybrid Toolbox [14]	$> 2GB$ RAM	

5 Conclusions

This paper has introduced a new method of enumerating the solution to a parametric linear program based on a primal–dual paradigm. It was shown that the proposed algorithm can significantly reduce the number of linear programs that need to be solved in order to determine irredundant descriptions of the critical regions. The code used in the paper is available as part of the Multiparametric Toolbox [13].

Acknowledgments

The authors would like to thank Eric Kerrigan and Komei Fukuda for their valuable discussions while preparing this paper.

References

1. Borrelli, F., Bemporad, A., Morari, M.: A Geometric Algorithm for Multi-Parametric Linear Programming. Journal of Optimization Theory and Applications **118**(3) (2003) 515–540
2. Tøndel, P., Johansen, T., Bemporad, A.: An algorithm for multi-parametric quadratic programming and explicit MPC solutions. Automatica (2003) 489–497
3. Bemporad, A., Morari, M., Dua, V., Pistikopoulos, E.: The explicit linear quadratic regulator for constrained systems. Automatica **38**(1) (2002) 3–20
4. Bremner, D., Fukuda, K., Marzetta, A.: Primal-dual methods for vertex and facet enumeration. Discrete and Computational Geometry **20** (1998) 333–357
5. Bertsekas, D., Tsitsiklis, J.: Introduction to Linear Optimization. Athena Scientific (1997)
6. Jones, C.: Polyhedral Tools for Control. PhD thesis, University of Cambridge (2005)
7. Ziegler, G.: Lectures on Polytopes. Springer-Verlag, New York (1995)
8. Fukuda, K.: Frequently asked questions in polyhedral computation. http://www.ifor.math.ethz.ch/fukuda/polyfaq/polyfaq.html (2000)
9. Preparatat, F., Shamos, M.: Computational Geometry: An Introduction. Springer-Verlag, New York (1985)
10. Motzkin, T., Raiffa, H., Thompson, G., Thrall, R.: The double description method. In Kuhn, H., Tucker, A., eds.: Contributions to the Theory of Games II. Volume 8 of Ann. of Math. Stud. Princeton University Press (1953) 51–73
11. Seidel, R.: A convex hull algorithm optimal for point sets in even dimension. Master's thesis, Dept. of Computer Science, University of British Columbia, Vancouver, Canada (1981)
12. Goodman, J.E., O'Rourke, J., eds.: Handbook of Discrete and Computational Geometry. CRC Press, New York (1997)
13. Kvasnica, M., Grieder, P., Baotić, M.: Multi-Parametric Toolbox (MPT) (2004) http://control.ee.ethz.ch/ mpt/.
14. Bemporad, A.: Hybrid toolbox (2005) Version 1.0.10, http://www.dii.unisi.it/hybrid/toolbox/.
15. Barber, C., Dobkin, D., Huhdanpaa, H.: The quickhull algorithm for convex hulls. ACM Trans. Math. Softw. **22**(4) (1996) 469–483
16. Murray, W., Saunders, M.: Systems Optimization Laboratory (SOL) (2006) http://www.sbsi-sol-optimize.com.

A Parallel, Asynchronous Method for Derivative-Free Nonlinear Programs

Joshua D. Griffin and Tamara G. Kolda

Sandia National Laboratories, Livermore, CA 94551, USA
{jgriffi, tgkolda}@sandia.gov
http://csmr.ca.sandia.gov/~{jgriffi, tgkolda}

Derivative-free optimization algorithms are needed to solve real-world engineering problems that have computationally expensive and noisy objective function and constraint evaluations. In particular, we are focused on problems that involve running cumbersome simulation codes with run times measured in hours. In such cases, attempts to compute derivatives can prove futile because analytical derivatives are typically unavailable and noise limits the accuracy of numerical approximations. Furthermore, the objective and constraint functions may be inherently nonsmooth, i.e., because the underlying model is nonsmooth.

Generating Set Search (GSS) methods [7] are particularly well-suited to such unwieldy optimization problems. GSS methods are a generalization of pattern search that derives its search directions from the generators of the ϵ-tangent cone of the linear constraints, i.e., a generating set. GSS methods offer several advantages:

- Because search direction are based upon the local geometry of the feasible region defined by the linear constraints, and not the objective or nonlinear constraint functions, they are well-suited for problems with noise.
- The function evaluations can be performed asynchronously in parallel [5, 10, 6].
- If the underlying objective function and constraints are smooth, GSS methods can bound the first-order optimality conditions in terms of step size.
- They can easily accommodate undefined points within the feasible region, i.e, points where the simulation unexpectedly fails.

The focus of this talk will be on the addition of constraint-handling abilities to APPSPACK, which is a C++ implementation of an asynchronous parallel GSS algorithm [4]. APPSPACK is a publicly available derivative-free software package whose handling of both linear and nonlinear constraints is based upon rigorous convergence theory. Specifically, we have added the ability to handle linear constraints using conforming search directions and nonlinear constraints using an augmented Lagrangian algorithm.

We are interested in solving a nonlinear program of the form

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & c(x) = d \\ & \ell \le Ax \le u. \end{aligned} \tag{1}$$

Here $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function, $c : \mathbb{R}^n \to \mathbb{R}^p$ denotes the nonlinear equality constraints, and A denotes an $n \times m$ linear constraint matrix. We allow

© Springer-Verlag Berlin Heidelberg 2006

for both linear equality and inequality constraints. We assume that evaluating $f(x)$ and $c(x)$ is expensive and derivatives are unavailable.

Consider first the problem of linear constraints. APPSPACK's linear constraint support is based on [11, 8]. We generate a core set of search directions, as outlined in [11], from generators of tangent cones corresponding to nearby constraints. The solid lines in 1 show the "conforming" search directions to the nearby boundary (as defined by the ϵ-ball that is drawn). The dashed lines indicate additional directions that we might choose to add to further accelerate the search. This choice of search directions guarantees that we have at least one feasible descent direction, if one exists. If the nearby constraints are nondegenerate, finding the search directions is a straightforward linear algebra problem; otherwise, more sophisticated machinery is required and we use `cddlib` [3] to find the appropriate generators.

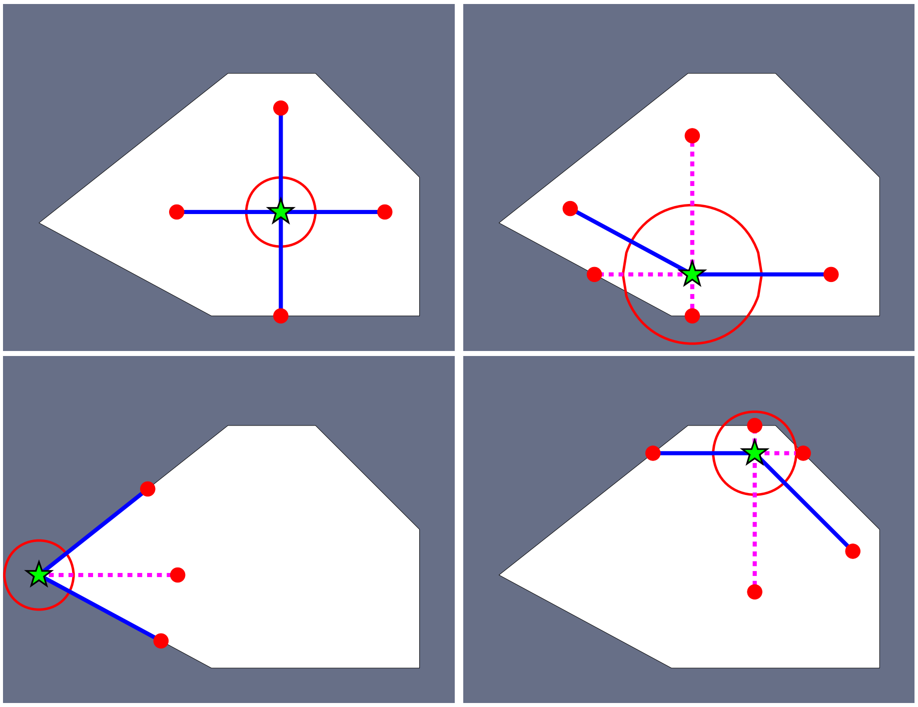


Fig. 1. At each iteration an ϵ-ball is formed about the current point to determine nearby constraints and corresponding search directions added. Extra directions (denoted by dashes) are generally added to facilitate movement toward and away from the boundary.

APPSPACK handles nonlinear equality constraints using an augmented Lagrangian method proposed in [2, 1] and extended to the directive-free case in [12, 9]. Remarkably, the derivative-free variant retains the same theoretical convergence properties as the original derivative-based augmented Lagrangian approach. This algorithm can be divided into inner and outer iterations; inner iterations being devoted to approximately solving the linearly-constrained subproblem

$$\begin{array}{ll} \underset{x \in \mathbb{R}^n}{\text{minimize}} & \phi_k(x) \equiv f(x) - \lambda_k^T c(x) + \frac{1}{2\mu_k}\|c(x)\|^2 \\ \text{subject to} & \ell \leq Ax \leq u \end{array} \tag{2}$$

for fixed λ_k and μ_k, while outer iteration are used to assess optimality and update the subproblem specific parameters.

This talk will begin with a brief background of GSS methods and follow with a description of how linear and nonlinear constraints are handled in APPSPACK. Theory and numerical results will be provided.

References

1. A. Conn, N. Gould, A. Sartenaer, and P. Toint, *Convergence properties of an augmented Lagrangian algorithm for optimization with a combination of general equality and linear constraints*, SIAM J. Optimiz., 6 (1996), pp. 674–703.
2. A. R. Conn, N. I. M. Gould, and Ph. L. Toint, *A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds*, SIAM J. Numer. Anal., 28 (1991), pp. 545–572.
3. K. Fukuda, *cdd and cddplus homepage*. From McGill University, Montreal, Canada Web page: `http://www.cs.mcgill.ca/~fukuda/soft/cdd_home/cdd.html`, 2005.
4. G. A. Gray and T. G. Kolda, *Algorithm 8xx: APPSPACK 4.0: Asynchronous parallel pattern search for derivative-free optimization*, ACM T. Math. Software. To appear.
5. P. D. Hough, T. G. Kolda, and V. J. Torczon, *Asynchronous parallel pattern search for nonlinear optimization*, SIAM J. Sci. Comput., 23 (2001), pp. 134–156.
6. T. G. Kolda, *Revisiting asynchronous parallel pattern search for nonlinear optimization*, SIAM J. Optimiz., 16 (2005), pp. 563–586.
7. T. G. Kolda, R. M. Lewis, and V. Torczon, *Optimization by direct search: new perspectives on some classical and modern methods*, SIAM Rev., 45 (2003), pp. 385–482.
8. ———, *Stationarity results for generating set search for linearly constrained optimization*, Tech. Rep. SAND2003-8550, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, Oct. 2003.
9. ———, *Convergence properties of an augmented Lagrangian direct search algorithm for optimization with a combination of general equality and linear constraints*. In preparation, 2005.
10. T. G. Kolda and V. Torczon, *On the convergence of asynchronous parallel pattern search*, SIAM J. Optimiz., 14 (2004), pp. 939–964.
11. R. M. Lewis, A. Shepherd, and V. Torczon, *Implementing generating set search methods for linearly constrained minimization*, Tech. Rep. WM-CS-2005-01, Department of Computer Science, College of William & Mary, Williamsburg, Virginia, July 2005.
12. R. M. Lewis and V. Torczon, *A globally convergent augmented Lagrangian pattern search algorithm for optimization with general constraints and simple bounds*, SIAM J. Optimiz., 12 (2002), pp. 1075–1089.

Convergent SDP-Relaxations for Polynomial Optimization with Sparsity

Jean B. Lasserre

LAAS-CNRS and Institute of Mathematics
LAAS, 7 avenue du Colonel Roche
31077 Toulouse Cédex 4, France
http://www.laas.fr/~lasserre
lasserre@laas.fr

Abstract. We consider a polynomial programming problem $\mathbf{P}$ on a compact basic semi-algebraic set $\mathbf{K} \subset \mathbb{R}^n$, described by m polynomial inequalities $g_j(X) \geq 0$, and with criterion $f \in \mathbb{R}[X]$. We propose a hierarchy of semidefinite relaxations that take sparsity of the original data into account, in the spirit of those of Waki et al. [7]. The novelty with respect to [7] is that we prove convergence to the global optimum of $\mathbf{P}$ when the sparsity pattern satisfies a condition often encountered in large size problems of practical applications, and known as the *running intersection property* in graph theory.

1 Introduction

In this paper we consider the polynomial programming problem:

$$\mathbf{P}: \quad \inf_{x\in\mathbb{R}^n} \{\, f(x) \mid \quad x \in \mathbf{K}\}, \tag{1}$$

where $f \in \mathbb{R}[X]$, and $\mathbf{K} \subset \mathbb{R}^n$ is the basic closed semi-algebraic set defined by

$$\mathbf{K} := \{x \in \mathbb{R}^n \mid \quad g_j(x) \geq 0, \quad j = 1, \ldots, m\,\}, \tag{2}$$

for some polynomials $\{g_j\}_{j=1}^m \subset \mathbb{R}[X]$.

The hierarchy of semidefinite programming (SDP) relaxations introduced in Lasserre [8] provides a sequence of SDPs of increasing size, whose associated sequence of optimal values converges to the global minimum of $\mathbf{P}$. Moreover, as proved in Schweighofer [13], convergence to a global minimizer of $\mathbf{P}$ (if unique) also holds. For more details, the reader is referred to [3,8,13] and the many references therein. In addition, practice reveals that convergence is usually fast, and often *finite* (up to machine precision); see e.g. Henrion and Lasserre [3].

However, despite these nice features, the size of the SDP-relaxations grows rapidly with the size of the original problem. Typically, the k^{th} SDP-relaxation has to handle at least one LMI of size $\binom{n+k}{n}$ and $\binom{n+2k}{n}$ variables, which clearly limits the applicability of the methodology to problems with small to medium

© Springer-Verlag Berlin Heidelberg 2006

size only. Therefore, validation of the above methodology for larger size problems (and even more, for large scale problems) is a real challenge of practical importance.

One way to extend the applicability of the methodology to problems of larger size, is to take into account *sparsity* in the original data, frequently encountered in practical cases. Indeed, as typical in many applications of interest, f as well as the polynomials $\{g_j\}$ that describe $\mathbf{K}$, are sparse, i.e., each monomial of f and each polynomial g_j are only concerned with a small subset of variables. This is the approach taken in Waki et al. [7] (extending Kim et al. [5] and Kojima et al. [6]), where the authors have built up a hierarchy of SDP-relaxations in the spirit of those in [8], but where sparsity is taken into account. In [7], the authors have proposed a systematic procedure to detect and structure sparsity in $\mathbf{P}$, via the so-called *chordal extension* of the *correlation sparsity pattern graph* (csp graph); the csp graph has as many nodes as variables, and a link beween two nodes (i.e., variables) means that these two variables both appear in a monomial of the objective function or in some inequality constraint $g_j \geq 0$ of $\mathbf{P}$. Once a sparsity pattern has been detected, they define a simplified "sparse" version of the SDP-relaxations of [8]; briefly, in the dual, the sum of squares (s.o.s.) multiplier associated with a constraint is now a polynomial in only those variables appearing in that constraint. In doing so, they have obtained impressive gains in the size of the resulting SDP-relaxations, as well as in the computational time needed for obtaining an optimal solution. However, and despite good approximations are obtained in most problems in their sample of experiments, convergence to the global minimum is *not* guaranteed.

Contribution. We propose a hierarchy of SDP-relaxations $\{\mathbf{Q}_r\}$ in the spirit of the original SDP-relaxations [8] and close to those defined in [7]. They are valid for arbitrary polynomial programming problems, and have the following three appealing features:

(a) In the SDP-relaxation $\mathbf{Q}_r$ of order r, the number of variables is $O(\kappa^{2r})$ where $\kappa = \max[\kappa_1, \kappa_2]$ witth κ_1 (resp. κ_2) being the maximum number of variables appearing in f (resp. in a single constraint $g_j(X) \geq 0$).

(b) The largest size of the LMI's (Linear Matrix Inequalities) is $O(\kappa^r)$.

This is to compare with the respective number of variables $O(n^{2r})$ and LMI size $O(n^r)$ in the original SDP-relaxations defined in [8].

(c) Under a certain condition on the sparsity pattern, the resulting sequence of their optimal value *converges* to the global minimum of $\mathbf{P}$.

So in view of (a) and (b), and when κ is small ($\kappa \ll n$), i.e., when sparsity is present, dramatic computational savings can be expected. In other words, these new SDP-relaxations are inherently exploiting sparsity in the data $\{f, g_j\}$ when present. Moreover, the size of the SDP-relaxation $\mathbf{Q}_r$ is in a sense *minimal*, at least when considering such types of SDP-relaxations, because one should at least handle moments involving κ variables, whenever some monomial of κ variables appears in the data $\{f, g_j\}$.

The condition under which such SDP-relaxations converge to the global minimum of $\mathbf{P}$ is easy to describe, and reflects a sparsity pattern frequently encountered in large scale problems. Namely, let $\{1,\dots,n\}$ be the union $\bigcup_{k=1}^{p} I_k$ of subsets $I_k \subset \{1,\dots,n\}$. Every polynomial g_j in the definition (2) of $\mathbf{K}$, is only concerned with variables $\{X_i \,|\, i \in I_k\}$ for some k. Next, $f \in \mathbb{R}[X]$ can be written $f = f_1 + \cdots + f_p$ where each f_k uses only variables $\{X_i \,|\, i \in I_k\}$. In cases where the subsets $\{I_k\}$ are not so easy to detect, one may use the procedure of Waki et al. [7] via the chordal extension of the csp graph.

Finally, the collection $\{I_1,\dots,I_p\}$ should obey the following condition: For every $k = 1,\dots,p-1$,

$$I_{k+1} \cap \bigcup_{j=1}^{k} I_j \subseteq I_s \quad \text{for some } s \leq k. \tag{3}$$

Notice that (3) is always satisfied when $p = 2$. Property (3) depends on the ordering and so, can be satisfied possibly after some relabelling of the I_k's. Moreover, if not satisfied, one may enforce (3) but at the price of enlarging some of the sets I_k. If $I_1,\dots,I_p$ are the maximal cliques of a chordal graph then (3) is satisfied possibly after some reordering of the cliques, and is known as the *running intersection property*; for more details on chordal graphs, the reader is referred to Fukuda et al. [2] and Nakata et al. [11].

Link with related literature. As already mentioned, our work is closely related to the recent work of Kojima et al. [6] and Waki et al. [7], in which they were the first to exploit sparsity of data and modify (or simplify) in an appropriate way the original SDP-relaxations defined in [8]. Our SDP-relaxations are very close to those defined in [7], but handle p additional quadratic constraints. These p additional constraints together with condition (3), are crucial to prove our convergence result. To summarize, our result implies that by a slight modification of the SDP-relaxations defined in [7], convergence is now guaranteed when the sparsity pattern satisfies (3).

2 Notation and Definitions

As common in algebra, variables of polynomials are denoted with capitals (e.g. X) whereas points in $\mathbb{R}^n$ are denoted with small letters (e.g. x). For a real symmetric matrix $A \in \mathbb{R}^{n\times n}$, the notation $A \succeq 0$ (resp. $A \succ 0$) stands for A is positive definite (resp. semidefinite), and for a vector x, let x' denote its transpose.

Let $\mathbb{R}[X]$ denote the ring of real polynomials in the variables $X_1,\dots,X_n$. In the usual canonical basis $v_\infty(X) = \{X^\alpha \,|\, \alpha \in \mathbb{N}^n\}$ of monomials, a polynomial $g \in \mathbb{R}[X]$ is written

$$g(X) = \sum_{\alpha\in\mathbb{N}^n} g_\alpha X^\alpha, \tag{4}$$

for some real vector $\mathbf{g} = \{g_\alpha\}$ with finitely many non zero coefficients.

With $\alpha \in \mathbb{N}^n$, let $|\alpha| := \sum_i \alpha_i$, and let $\mathbb{R}_r[X] \subset \mathbb{R}[X]$ be the $\mathbb{R}$-vector space of polynomials of degree at most r, with usual canonical basis of monomials $v_r(X) = \{X^\alpha \,|\, \alpha \in \mathbb{N}^n;\ |\alpha| \leq r\}$.

Let $I_0 := \{1, \ldots, n\}$ be the union $\cup_{k=1}^{p} I_k$ of p subsets I_k, $k = 1, \ldots, p$, with cardinal denoted n_k. Let $\mathbb{R}[X(I_k)]$ denote the ring of polynomials in the n_k variables $X(I_k) = \{X_i \,|\, i \in I_k\}$, and so $\mathbb{R}[X(I_0)] = \mathbb{R}[X]$.

For each $k = 0, 1, \ldots, p$, let $\mathcal{I}_k$ be the set of all subsets of I_k. Next, for every $\alpha \in \mathbb{N}^n$, let $\mathrm{supp}\,(\alpha) \in \mathcal{I}_0$ be the support of α, i.e.,

$$\mathrm{supp}\,(\alpha) \;:=\; \{\, i \in \{1, \ldots, n\} \;:\quad \alpha_i \neq 0 \,\}, \qquad \alpha \in \mathbb{N}^n.$$

For instance, with $n = 6$ and $\alpha := (004020)$, $\mathrm{supp}\,(\alpha) = \{3, 5\}$. Next, define

$$S_k := \{\, \alpha \in \mathbb{N}^n \;:\quad \mathrm{supp}\,(\alpha) \,\in\, \mathcal{I}_k \,\}, \qquad k = 1, \ldots, p. \tag{5}$$

A polynomial $h \in \mathbb{R}[X(I_k)]$ can be viewed as a member of $\mathbb{R}[X]$, and is written

$$h(X) \;=\; h(X(I_k)) \;=\; \sum_{\alpha \in S_k} h_\alpha \, X^\alpha \tag{6}$$

for some real vector $\mathbf{h} = \{h_\alpha\}$ with finitely many non zero coefficients.

2.1 Moment Matrix

Let $y = (y_\alpha)_{\alpha \in \mathbb{N}^n}$ (i.e. a sequence indexed in the canonical basis $v_\infty(X)$), and define the linear functional $L_y : \mathbb{R}[X] \to \mathbb{R}$ to be:

$$g \mapsto L_y(g) \;:=\; \sum_{\alpha \in \mathbb{N}^n} g_\alpha \, y_\alpha, \tag{7}$$

whenever g is as in (4).

As already presented in [8], given a sequence $y = (y_\alpha)_{\alpha \in \mathbb{N}^n}$, the *moment* matrix $M_r(y)$ associated with y, is the matrix with rows and columns indexed in $v_r(X)$, and such that

$$M_r(y)(\alpha, \beta) \;:=\; L_y(X^\alpha X^\beta) \;=\; y_{\alpha+\beta}, \quad \forall\, \alpha, \beta \in \mathbb{N}^n \text{ with } |\alpha|, |\beta| \leq r.$$

A sequence y is said to have a representing measure μ on $\mathbb{R}^n$ if

$$y_\alpha \;=\; \int_{\mathbb{R}^n} X^\alpha \, \mu(dX), \qquad \forall\, \alpha \in \mathbb{N}^n.$$

Let $s(r) := \binom{n+r}{r}$ be the dimension of vector space $\mathbb{R}_r[X]$. For a vector $\mathbf{u} \in \mathbb{R}^{s(r)}$, let $u \in \mathbb{R}[X]$ be the polynomial $u(X) = \langle \mathbf{u}, v_r(X) \rangle$. Then, one has

$$\langle \mathbf{u}, M_r(y)\mathbf{u} \rangle \;=\; L_y(u^2), \qquad \forall\, \mathbf{u} \in \mathbb{R}^{s(r)}.$$

Therefore, if y has a representing measure μ, then

$$\langle \mathbf{u}, M_r(y)\mathbf{u} \rangle \;=\; L_y(u^2) \;=\; \int_{\mathbb{R}^n} u(X)^2 \, \mu(dX) \;\geq 0,$$

which implies $M_r(y) \succeq 0$ (as $\mathbf{u} \in \mathbb{R}^{s(r)}$ was arbitrary).

Of course, in general, not every sequence y such that $M_r(y) \succeq 0$ for all $r \in \mathbb{N}$, has a representing measure. The $\mathbf{K}$-moment problem is precisely concerned with finding conditions on the sequence y, to ensure it is the moment sequence of some measure μ, with support contained in $\mathbf{K} \subset \mathbb{R}^n$.

2.2 Localizing Matrix

Let $h \in \mathbb{R}[X]$ be a given polynomial

$$h(X) \,=\, \sum_{\gamma\in\mathbb{N}^n} h_\gamma\, X^\gamma,$$

and let $y = (y_\alpha)_{\alpha\in\mathbb{N}^n}$ be given. The *localizing* matrix $M_r(h\,y)$ associated with h and y, is the matrix with rows and columns indexed in $v_r(X)$, obtained from the moment matrix $M_r(y)$ by:

$$M_r(h\,y)(\alpha,\beta) \,:=\, L_y(h(X)\,X^\alpha X^\beta) \,=\, \sum_{\gamma\in\mathbb{N}^n} h_\gamma\, y_{\gamma+\alpha+\beta},$$

for all $\alpha,\beta\in\mathbb{N}^n$, with $|\alpha|,|\beta|\le r$.

As before, let $\mathbf{u}\in\mathbb{R}^{s(r)}$, and let $u := \langle \mathbf{u}, v_r(X)\rangle \in \mathbb{R}_r[X]$. Then

$$\langle \mathbf{u}, M_r(h\,y)\mathbf{u}\rangle \,=\, L_y(h\,u^2), \qquad \forall\,\mathbf{u}\in\mathbb{R}^{s(r)},$$

and if y has a representing measure μ with support contained in the set $\{x\in\mathbb{R}^n \,:\, h(x)\ge 0\}$, then

$$\langle \mathbf{u}, M_r(h\,y)\mathbf{u}\rangle \,=\, L_y(h\,u^2) \,=\, \int_{\mathbb{R}^n} h(X)\,u(X)^2\,\mu(dX) \,\ge 0,$$

which implies $M_r(h\,y)\succeq 0$ (as $\mathbf{u}\in\mathbb{R}^{s(r)}$ was arbitrary).

Next, with $k\in\{1,\dots,p\}$ fixed, and $h\in\mathbb{R}[X(I_k)]$, let $M_r(y,I_k)$ (resp. $M_r(h\,y,I_k)$) be the moment (resp. localizing) submatrix obtained from $M_r(y)$ (resp. $M_r(h\,y)$) by retaining only those rows (and columns) $\alpha\in\mathbb{N}^n$ of $M_r(y)$ (resp. $M_r(h\,y)$) with $\mathrm{supp}\,(\alpha)\in\mathcal{I}_k$.

In doing so, $M_r(y,I_k)$ and $M_r(h\,y,I_k)$ can be viewed as moment and localizing matrices with rows and columns indexed in the canonical basis $v_r(X(I_k))$ of $\mathbb{R}_r[X(I_k)]$. Indeed, $M_r(y,I_k)$ contain only variables y_α with $\mathrm{supp}\,(\alpha)\in\mathcal{I}_k$, and so does $M_r(h\,y,I_k)$ because $h\in\mathbb{R}[X(I_k)]$. And for every polynomial $u\in\mathbb{R}_r[X(I_k)]$, with coefficient vector $\mathbf{u}$ in the basis $v_r(X(I_k))$, we also have

$$\begin{aligned}\langle \mathbf{u}, M_r(y,I_k)\mathbf{u}\rangle &= L_y(u^2), \qquad \forall\,u\in\mathbb{R}_r[X(I_k)]\\ \langle \mathbf{u}, M_r(h\,y,I_k)\mathbf{u}\rangle &= L_y(h\,u^2), \qquad \forall\,u\in\mathbb{R}_r[X(I_k)],\end{aligned}$$

and therefore,

$$M_r(y,I_k)\,\succeq 0 \Leftrightarrow L_y(u^2)\ge 0, \qquad \forall\,u\in\mathbb{R}_r[X(I_k)] \tag{8}$$

$$M_r(h\,y,I_k)\,\succeq 0 \Leftrightarrow L_y(h\,u^2)\ge 0, \qquad \forall\,u\in\mathbb{R}_r[X(I_k)]. \tag{9}$$

3 Main Result

Consider problem $\mathbf{P}$ as defined in (1), and recall that $I_0=\{1,\dots,n\}=\bigcup_{k=1}^p I_k$ for some subsets $I_k\subset\{1,\dots,n\}$, $k=1,\dots,p$. The subsets $\{I_k\}$ may be read directly from the data or may have been obtained by some procedure, e.g. the one described in Waki et al. [7].

With $\|x\|_\infty$ (resp. $\|x\|$) denoting the usual sup-norm (resp. euclidean norm) of a vector $x \in \mathbb{R}^n$, we make the following assumption.

Assumption 1. Let $\mathbf{K} \subset \mathbb{R}^n$ be as in (2). Then, there is $M > 0$ such that $\|x\|_\infty < M$ for all $x \in \mathbf{K}$.

In view of Assumption 1, one has $\|X(I_k)\|^2 \leq n_k M^2$, $k = 1, \ldots, p$, and therefore, in the definition (2) of $\mathbf{K}$, we add the p redundant quadratic constraints

$$g_{m+k}(X) := n_k M^2 - \|X(I_k)\|^2 \geq 0, \quad k = 1, \ldots, p, \tag{10}$$

and set $m' = m + p$, so that $\mathbf{K}$ is now defined by:

$$\mathbf{K} := \{x \in \mathbb{R}^n \mid \quad g_j(x) \geq 0, \quad j = 1, \ldots, m'\}. \tag{11}$$

Notice that $g_{m+k} \in \mathbb{R}[X(I_k)]$, for every all $k = 1, \ldots, p$.

Assumption 2. Let $\mathbf{K} \subset \mathbb{R}^n$ be as in (11). The index set $J = \{1, \ldots, m'\}$ is partitioned into p disjoint sets J_k, $k = 1, \ldots, p$, and the collections $\{I_k\}$ and $\{J_k\}$ satisfy:

(i) For every $j \in J_k$, $g_j \in \mathbb{R}[X(I_k)]$, that is, for every $j \in J_k$, the constraint $g_j(X) \geq 0$ is only concerned with the variables $X(I_k) = \{X_i \mid i \in I_k\}$. Equivalently, viewing g_j as a polynomial in $\mathbb{R}[X]$, $g_{j\alpha} \neq 0 \Rightarrow \operatorname{supp}(\alpha) \in \mathcal{I}_k$.

(ii) The objective function $f \in \mathbb{R}[X]$ can be written

$$f = \sum_{k=1}^{p} f_k, \quad \text{with } f_k \in \mathbb{R}[X(I_k)], \quad k = 1, \ldots, p. \tag{12}$$

Equivalently, $f_\alpha \neq 0 \Rightarrow \operatorname{supp}(\alpha) \in \cup_{k=1}^{p} \mathcal{I}_k$.

(iii) (3) holds.

As already mentioned, (3) always holds when $p \leq 2$.

Example 1. With $n = 6$, and $m = 6$, let

$$g_1(X) = X_1X_2 - 1; \quad g_2(X) = X_1^2 + X_2X_3 - 1; \quad g_3(X) = X_2 + X_3^2X_4,$$

and

$$g_4(X) = X_3 + X_5; \quad g_5(X) = X_3X_6; \quad g_6(X) = X_2X_3,$$

Then one may choose $p = 4$ with

$$I_1 = \{1, 2, 3\}; \; I_2 = \{2, 3, 4\}; \; I_3 = \{3, 5\}; \; I_4 = \{3, 6\},$$

and $J_1 = \{1, 2, 6\}, J_2 = \{3\}, J_3 = \{4\}, J_4 = \{4\}$.

So in Example 1, the objective function $f \in \mathbb{R}[X]$ should be a sum of polynomials in $\mathbb{R}[X_1, X_2, X_3]$, $\mathbb{R}[X_2, X_3, X_4]$, $\mathbb{R}[X_3, X_5]$ and $\mathbb{R}[X_3, X_6]$ (also considered as polynomials in $\mathbb{R}[X_1, \ldots, X_6]$).

Remark 1. For every $k = 1, \ldots, p$, let

$$\mathbf{K}_k := \{x \in \mathbb{R}^{n_k} : g_j(x) \geq 0, \quad \forall j \in J_k\}. \tag{13}$$

For every $k = 1, \ldots, p$, the set $\mathbf{K}_k \subset \mathbb{R}^{n_k}$ satisfies Putinar's condition, that is, there exists $u \in \mathbb{R}[X(I_k)]$ which can be written $u = u_0 + \sum_{l \in J_k} u_l \, g_l$ for some s.o.s. polynomials $\{u_0, u_l\} \subset \mathbb{R}[X(I_k)]$, and such that the level set $\{x \in \mathbb{R}^{n_k} : u \geq 0\}$ is compact. (Take $u = g_{m+k}$.) This property is crucial in the proof of Theorem 1.

3.1 Convergent SDP-Relaxations

For each $j = 1, \ldots, m'$, and depending on its parity, write $\deg g_j = 2r_j - 1$ or $2r_j$. Next, with $2r \geq 2r_0 := \max[\deg f, \max_j 2r_j]$, consider the following semidefinite program:

$$\mathbf{Q}_r : \quad \begin{cases} \inf\limits_{y} & L_y(f) \\ \text{s.t.} & M_r(y, I_k) \succeq 0, \, k = 1, \ldots, p \\ & M_{r-r_j}(g_j \, y, I_k) \succeq 0, \, j \in J_k; \quad k = 1, \ldots, p \\ & y_0 = 1 \end{cases}, \tag{14}$$

where the moment and localizing matrices $M_r(y, I_k)$, $M_r(g_j \, y, I_k)$ have been defined at the end of §2.2. Denote the optimal value of $\mathbf{Q}_r$ by $\inf \mathbf{Q}_r$, and $\min \mathbf{Q}_r$ if the infimum is attained.

Notice that $\mathbf{Q}_r$ is well-defined under Assumption 2(i)-(ii). Assumption 2(iii) is only useful to show convergence in Theorem 1 below.

The semidefinite program $\mathbf{Q}_r$ is a relaxation of $\mathbf{P}$. Indeed, with $x \in \mathbb{R}^n$ being a feasible solution of $\mathbf{P}$, the moment vector $y = \{y_\alpha\}$ of the Dirac measure $\mu = \delta_x$ at x, is feasible for $\mathbf{Q}_r$, with value $L_y(f) = \int f d\mu = f(x)$.

Under Assumption 2, and from the definition of $M_r(y, k)$ and $M_r(g_j \, y, k)$ in §2.2, the SDP-relaxation $\mathbf{Q}_r$ contains only variables y_α with α in the set

$$\Gamma_r := \{\, \alpha \in \mathbb{N}^n : \quad \mathrm{supp}\,(\alpha) \in \bigcup_{k=1}^{p} \mathcal{I}_k \, ; \quad |\alpha| \leq 2r \,\}. \tag{15}$$

Remark 2. Comparing with the SDP-relaxations of Waki et al. [7]. When the sets $\{I_k\}$ are just the *cliques* $\{C_k\}$ obtained from the chordal extension of the csp graph as defined in [7], then the SDP-relaxations (14) are basically the same as those defined in (32) in [7]. The only difference is in the definition of the feasible set $\mathbf{K}$ of $\mathbf{P}$, where we have now included the p redundant quadratic constraints (10). In this case, the SDP-relaxations (14) are thus stronger than (32) in [7], because they are more constrained.

In view of the definition of the moment matrix $M_r(y, I_k)$, write

$$M_r(y, I_k) \;=\; \sum_{\alpha\in\mathbb{N}^n} y_\alpha B^k_\alpha, \qquad k=1,\ldots,p,$$

for appropriate symmetric matrices $\{B^k_\alpha\}$, and notice that for every $k=1,\ldots,p$, one has $B^k_\alpha = 0$ whenever $\mathrm{supp}\,(\alpha) \notin \mathcal{I}_k$. Similarly, for every $k=1,\ldots,p$, and $j \in J_k$, write

$$M_{r-r_j}(g_j\,y, I_k) \;=\; \sum_{\alpha\in\mathbb{N}^n} y_\alpha C^{jk}_\alpha,$$

for appropriate symmetric matrices $\{C^{jk}_\alpha\}$, and notice that $C^{jk}_\alpha = 0$ whenever $\mathrm{supp}\,(\alpha) \notin \mathcal{I}_k$.

The dual SDP $\mathbf{Q}_r^*$ of $\mathbf{Q}_r$, reads

$$\left\{\begin{array}{ll} \displaystyle\sup_{\Omega_k, Z_{jk}, \lambda} \;\lambda & \\ \text{s.t.} & \displaystyle\sum_{k:\,\mathrm{supp}\,(\alpha)\in\mathcal{I}_k} [\,\langle \Omega_k, B^k_\alpha\rangle + \sum_{j\in J_k}\langle Z_{jk}, C^{jk}_\alpha\rangle\,] + \lambda\,\delta_{\alpha 0} \;=\; f_\alpha \\ & \text{for all } \alpha\in\Gamma_r \\ & \Omega_k, Z_{jk} \succeq 0, \qquad j\in J_k, \quad k=1,\ldots,p \end{array}\right. , \tag{16}$$

where Γ_r is defined in (15) and $\delta_{\alpha 0}$ is the usual Kronecker symbol. Proceding as in Lasserre [8], and using the spectral decomposition of matrices $\Omega_k, Z_{jk} \succeq 0$, the dual $\mathbf{Q}_r^*$ also reads:

$$\left\{\begin{array}{ll} \displaystyle\sup_{q_k, q_{jk}, \lambda} \;\lambda & \\ \text{s.t.} & f-\lambda \;=\; \displaystyle\sum_{k=1}^p (q_k + \sum_{j\in J_k} q_{jk}\,g_j\,) \\ & q_k,\, q_{jk} \in \mathbb{R}[X(I_k)] \text{ and s.o.s.}, \qquad j\in J_k, \quad k=1,\ldots,p \\ & \deg q_k,\, \deg q_{jk}g_j \le 2r, \qquad j\in J_k, \quad k=1,\ldots,p, \end{array}\right. \tag{17}$$

Theorem 1. *Let* $\mathbf{P}$ *be as defined in (1), with global minimum denoted* $\min\mathbf{P}$*, and let Assumption 1 and 2 hold. Let* $\{\mathbf{Q}_r\}$ *be the hierarchy of SDP-relaxations defined in (14). Then:*

(a) $\inf\mathbf{Q}_r \uparrow \min\mathbf{P}$ *as* $r\to\infty$*.*

(b) *If* $\mathbf{K}$ *has a nonempty interior, then there is no duality gap between* $\mathbf{Q}_r$ *and its dual* $\mathbf{Q}_r^*$*, and* $\mathbf{Q}_r^*$ *is solvable for sufficiently large* r*, i.e.,* $\inf\mathbf{Q}_r = \max\mathbf{Q}_r^*$*.*

(c) *Let y^r be a nearly optimal solution of $\mathbf{Q}_r$, with e.g.*

$$L_{y^r}(f) \leq \inf \mathbf{Q}_r + \frac{1}{r}, \qquad \forall\, r \geq r_0,$$

and let $\widehat{y}^r := \{y^r_\alpha : |\alpha| = 1\}$. If $\mathbf{P}$ has a unique global minimizer $x^* \in \mathbf{K}$, then

$$\widehat{y}^r \to x^* \qquad \textit{as } r \to \infty. \tag{18}$$

For a proof see [10]. Theorem 1 establishes convergence of the hierarchy of SDP-relaxations to the global minimum $\min \mathbf{P}$, as well as convergence to a global minimizer $x^* \in \mathbf{K}$ (if unique).

3.2 Computational Complexity

The number of variables for the SDP-relaxation $\mathbf{Q}_r$ defined in (14) is bounded by $\sum_{k=1}^{p} \binom{n_k+2r}{2r}$, and so, if all n_k's are *close* to each other, say $n_k \approx n/p$ for all k, then one has one has at most $O(p(\frac{n}{p})^{2r})$ variables, a big saving when compared with $O(n^{2r})$ in the original SDP-relaxations defined in [8] and implemented in [3].

In addition, one also has p LMI constraints of size $O((\frac{n}{p})^r)$ and $m+p$ LMI constraints of size $O((\frac{n}{p})^{r-r'})$ (where $2r'$ is the largest degree of the polynomials g_j's), to be compared with a single LMI constraint of size $O(n^r)$ and m LMI constraints of size $O(n^{r-r'})$ in [3,8]. So for instance, when using an interior point method, it is definitely better to handle p LMIs, each of size $(n/p)^r$, rather than a single LMI of size n^r.

4 Conclusion

We have provided a hierarchy of SDP-relaxations when the polynomial optimization problem $\mathbf{P}$ has some structured sparsity (which can be detected as in Waki et al. [7]). This hierarchy is of the same flavor (in fact a minor modification) as that in Waki et al. [7], for which excellent numerical results have been reported. Our contribution was to prove convergence of the optimal values to the global minimum of $\mathbf{P}$ when the sparsity pattern satisfies the condition (3), called the *running intersection property* in graph theory, and frequently encountered in practice. Therefore, this result together with [7], opens the door for the applicability of the general approach of SDP-relaxations to medium (and even large) scale polynomial optimization problems, at least when a certain sparsity pattern is present.

Acknowledgements. The author is indebted to Prof. M. Kojima for very interesting and helpful discussions on the topic of sparse SDP-relaxations. He also wishes to thank T. Netzer and M. Schweighofer from Konstanz University (Germany), who indicated a way to simplify the original SDP-relaxations of the author.

References

1. Curto, R.E., Fialkow, L.A.: The truncated complex K-moment problem. Trans. Amer. Math. Soc. 352 (2000) 2825–2855.
2. Fukuda, M., Kojima, M., Murota, K., Nakata, K.: Exploiting Sparsity in Semidefinite Programming via Matrix Completion I: General Framework. SIAM J. Optim. 11 (2001) 647–674.
3. Henrion, D., Lasserre, J.B.: GloptiPoly : Global Optimization over Polynomials with Matlab and SeDuMi. ACM Trans. Math. Soft. 29 (2003) 165–194.
4. Henrion, D., Lasserre, J.B.: Detecting global optimality and extracting solutions in GloptiPoly. In: Positive Polynomials in Control, D. Henrion and A. Garulli eds. Lecture Notes on Control and Information Sciences, Vol. 312, Springer Verlag, Berlin, 2005, pp. 293–310.
5. Kim, S., Kojima, M., Waki, H.: Generalized Lagrangian Duals and Sums of Squares Relaxations of Sparse Polynomial Optimization Problems. SIAM J. Optim. 15 (2005) 697–719.
6. Kojima, M., Kim, S., Waki, H.: Sparsity in sums of squares of polynomials. Math. Prog., to appear.
7. Waki, H., Kim, S., Kojima, M., Maramatsu, M.: Sums of squares and semidefinite programming relaxations for polynomial optimization problems witth structured sparsity. Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Tokyo, 2004.
8. Lasserre, J.B.: Global optimization with polynomials and the problem of moments. SIAM J. Optim. 11 (2001), 796–817.
9. Lasserre, J.B.: An explicit equivalent positive semidefinite program for nonliner 0-1 programs. SIAM J. Optim. 12 (2002) 756–769.
10. Lasserre, J.B.: Convergent SDP-relaxations for polynomial optimization with sparsity. SIAM J. Optim., to appear.
11. Nakata, K., Fujisawa, K., Fukuda, M., Kojima, M., Murota, K.: Exploiting Sparsity in Semidefinite Programming via Matrix Completion II: Implementation and Numerical Results. Math. Progr. 95 (2003) 303–327.
12. Putinar, M.: Positive polynomials on compact semi-algebraic sets. Indiana Univ. Math. J. 42 (1993) 969–984.
13. Schweighofer, M.: Optimization of polynomials on compact semialgebraic sets. SIAM J. Optim 15 (2005) 805–825.

Algorithm and Software for Integration over a Convex Polyhedron

Mark Korenblit[1] and Efraim Shmerling[2]

[1] Department of Computer Science
Holon Institute of Technology
52 Golomb Str., P.O. Box 305
Holon 58102, Israel
korenblit@hit.ac.il
[2] Department of Computer Science and Mathematics
The College of Judea and Samaria
Ariel 44837, Israel
efraimsh@yahoo.com

Abstract. We present a new efficient algorithm for numerical integration over a convex polyhedron in multi-dimensional Euclidian space defined by a system of linear inequalities. The software routines which implement this algorithm are described. A numerical example of calculating an integral using these routines is given.

1 Introduction and Description of the Algorithm

This article describes a new step in the development of algorithms and software for multiple integration.

Available standard numerical routines for multiple integration enable one to integrate over cubes of the form $\left[\ a_1^{(0)}; a_1^{(1)}\right] \times \left[\ a_2^{(0)}; a_2^{(1)}\right] \times ... \times \left[\ a_{n-1}^{(0)}; a_{n-1}^{(1)}\right] \times \left[\ a_n^{(0)}; a_n^{(1)}\right]$ in the n-dimensional Euclidean space E_n, where n is small, as well as over the sets in E_n defined by $\left[\ f_1^{(0)}; f_1^{(1)}\right] \times \left[f_2^{(0)}; f_2^{(1)}\right] \times ... \times [f_{n-1}^{(0)};\ f_{n-1}^{(1)}] \times \left[\ a_n^{(0)}; a_n^{(1)}\right]$, where $f_k^{(0)}$, $f_k^{(1)}$ $\left(k = \overline{1, n-1}\right)$ are functions, the only possible arguments of which are variables $x_{k-1}, x_{k-2}, ..., x_1$.

If the area of integration is more complicated, there are no available algorithms and software. There are only two universal methods that enable one to integrate over an arbitrary convex set (CS) in E_n, the boundary of which is defined by the equation $F(x_1, x_2, ..., x_n) = 0$. The first is the Monte-Carlo method [4]. The second is based on integration over a minimum n-dimensional cube C such that $CS \subset C$, and instead of the integral $\int\limits_{cs} f(x_1, x_2, ..., x_n)dx_1...dx_n$ the equivalent integral $\int\limits_{c} f_1(x_1, x_2, ..., x_n)dx_1...dx_n$ is calculated, where

$$f_1(x_1, x_2, ..., x_n) = \begin{cases} f(x_1, x_2, ..., x_n), & \text{if } (x_1, x_2, ..., x_n) \in CS \\ 0, & \text{otherwise} \end{cases} .$$

© Springer-Verlag Berlin Heidelberg 2006

Obviously, both methods are inefficient from the perspective of computation, and the second method requires a great number of "check on condition" operations which take a relatively long time. In the case where the domain of integration is a convex polyhedron much more efficient numerical methods can be developed.

Assuming that a convex polyhedron (CP) in E_n is defined by the system of linear inequalities over the variables $x_1, x_2, ..., x_n$, integration over the CP can be reduced to integration over a set of non-intersecting domains of the form $\left[f_1^{(0)}; f_1^{(1)} \right] \times \left[f_2^{(0)}; f_2^{(1)} \right] \times ... \times [f_{n-1}^{(0)};\ f_{n-1}^{(1)}] \times \left[a_n^{(0)}; a_n^{(1)} \right]$, where the functions $f_k^{(0)}, f_k^{(1)}$ $\left(k = \overline{1, n-1}\right)$ are linear combinations of variables $x_{k-2}, x_{k-1}, ..., x_1$.

Integration over such domains can be carried out utilizing available multiple integration routines (such routines are considered in [5] and [6]).

Integration over a convex polyhedron (and in the simplest case, calculating the volume of a convex polyhedron) has many applications. For example, it is required in solving certain probabilistic problems. Let's formulate one of them. For an n-th order random polynomial

$$a_0(\omega) + a_1(\omega)x + a_2(\omega)x^2 + ... + a_n(\omega)x^n = 0,$$

where $a_0(\omega), ..., a_n(\omega)$ are independent random variables uniformly distributed in the intervals $\left[a_1^{(0)}; a_1^{(1)} \right], \left[a_2^{(0)}; a_2^{(1)} \right], ..., \left[a_n^{(0)}; a_n^{(1)} \right]$, respectively, find the distribution of the number of real zeros of the polynomial belonging to a certain real interval $[a; b]$ (see [2]).

This problem can be easily reduced to the calculation of the volumes of a finite number of convex polyhedrons, defined by systems of linear inequalities.

An example of solving a problem of this kind with the help of the presented algorithm for multiple numerical integration is given in section 3 of the article.

In order to describe the presented method we need the following notations.

Notation 1. *Double inequality.*
Double inequality (DI) of j-th order is an ordered pair of j-th order vectors $LP = (LP_0, ..., LP_{j-1})$, $RP = (RP_0, ..., RP_{j-1})$ which correspond to the inequality

$$LP_0 + LP_1 \times x_1 + ... + LP_{j-1} \times x_{j-1} < x_j < RP_0 + RP_1 \times x_1 + ... + RP_{j-1} \times x_{j-1}.$$

Remark 1. From here on all the inequalities mentioned in the text are double inequalities. Assume that possible values of the variables x_i $\left(i = \overline{1, n}\right)$ are restricted to bounded intervals $[LConst_i; RConst_i]$ $\left(i = \overline{1, n}\right)$, respectively, this enables one to rewrite each linear inequality as a DI which is a necessary preliminary step for the algorithm.

Notation 2. *System of r-type.*
We say the system of inequalities belongs to r-type ($2 \leq r \leq n$) if it contains exactly one inequality of order i for each i such that $r < i \leq n$ and more than one r-th order inequalities. The system of inequalities belongs to a one-type if it has exactly one inequality of order i for each i such that $1 \leq i \leq n$.

Notation 3. *Transformation of an ordered pair (LV, RV) of vectors of i-th order into DI of order k $(k \le i-1)$.*

The inequality corresponding to the pair (LV, RV) is of the form

$$LV_0 + LV_1 \times x_1 + ... + LV_{i-1} \times x_{i-1} < RV_0 + RV_1 \times x_1 + ... + RV_{i-1} \times x_{i-1} \quad (1)$$

If $LV_{i-1} < RV_{i-1}$, we rewrite (1) in the form

$$RConst_{i-1} > x_{i-1} > \frac{LV_0 - RV_0}{RV_{i-1} - LV_{i-1}} + \frac{LV_1 - RV_1}{RV_{i-1} - LV_{i-1}} \times x_1 + \\ + \frac{LV_2 - RV_2}{RV_{i-1} - LV_{i-1}} \times x_2 + ... \frac{LV_{i-2} - RV_{i-2}}{RV_{i-1} - LV_{i-1}} \times x_{i-2}. \quad (2)$$

DI corresponding to (2) is then the result of the transformation.
If $LV_{i-1} > RV_{i-1}$, we rewrite (1) in the form

$$LConst < x_{i-1} < \frac{RV_0 - LV_0}{RV_{i-1} - LV_{i-1}} + \frac{RV_1 - LV_1}{RV_{i-1} - LV_{i-1}} \times x_1 + \\ + \frac{RV_2 - LV_2}{RV_{i-1} - LV_{i-1}} \times x_2 + ... + \frac{RVi -_2 - LV_{i-2}}{RV_{i-1} - LV_{i-1}} \times x_{i-2}. \quad (3)$$

If $LV_{i-k} = RV_{i-k}$, $k = 1, ..., k_1$, $LV_{i-k_1-1} \ne RV_{i-k_1-1}$,we transform the pair of vectors $(LV_0, LV_1, ..., LV_{i-k_1-1})$ of order $i - k_1$ into DI of order $i - k_1 - 1$ via formulae (2) or (3), where instead of i we substitute $i - k_1$.

Notation 4. *Decomposition.*
Assume that the initial system of inequalities (IS) belongs to r-type. The system has at least two inequalities of r-th order. These inequalities have ndlp distinct left parts and ndrp distinct right parts. So we have a pair of vectors (LP, RP), which consist of distinct left parts and distinct right parts, respectively, of r-th order inequalities belonging to the system.

The IS is decomposed into $ndlp \times ndrp$ systems S_{ij}, $i = \overline{1, ndlp}$, $j = \overline{1, ndrp}$. Each of the systems S_{ij} is formed in the following way:

(a) All inequalities (LP_i, RP_j) of orders different from r which belong to IS are included in S_{ij};

(b) Only one inequality (LP_i, RP_j) of order r is included in S_{ij};

(c) All the inequalities obtained via transformations of all pairs of vectors
(LP_i, RP_j), $s \ne i$, $s = \overline{1, ndlp}$
(RP_j, RP_p), $p \ne j$, $p = \overline{1, ndrp}$
are included in S_{ij};

(d) The inequality obtained via the transformation of the pair of vectors (LP_i, RP_j) is included in S_{ij}.

The systems S_{ij} which have no solutions (we call them invalid) are ignored.

It can be easily shown that:

– The system $\{S_{ij}\}_{\substack{i=\overline{ndlp}\\ j=\overline{ndrp}}}$ is equivalent to IS.
– The system $\{S_{ij}\}_{\substack{i=\overline{ndlp}\\ j=\overline{ndrp}}}$ consists of nonintersecting systems of inequalities.
– The types of systems S_{ij} are less than or equal to $r-1$.

Algorithm 1. *Algorithm for Integration over a Convex Polyhedron*
Step 1. Set Sum $= 0$ *(initialization of the calculated integral).*
Step 2. Check what is the type of IS. If the IS belongs to a one-type, go to step 3, otherwise go to step 4.
Step 3. Integrate over the set defined by the IS, add the calculated integral to Sum and terminate.
Step 4. Decompose the IS into nonintersecting systems and for each of them go to step 2 viewing it as IS.

Any available routine for numerical integration can be applied in step 3 of this algorithm.

2 Software Routines for the Algorithm Realization

The algorithm has been programmed in C++ language.

We define three basic structures named **Inequality**, **SOKOI** (system of k-th order inequalities), and **SOI** (system of inequalities). Their description is presented in Listing 1.

Structure **Inequality** implements an inequality of order k. Structure **SOKOI** realizes a system of m inequalities, each one of order k. Structure **SOI** includes n **SOKOI** structures, each of which presents a group of m_j inequalities of order j, $j = 1, 2, \ldots, n$. Thus, **SOI** forms a general system of inequalities.

Listing 1. Basic Structures

```
/* Implementation of inequality of order k */
struct Inequality
{
    int coefficients_num; // order of inequality
                          // (number of coefficients)
    double *left, *right; // pointers to left and right vectors
    Inequality (int k=0) // constructor
        : left (NULL),
          right (NULL)
    {
        coefficients_num = k; // set number of coefficients
        if (coefficients_num > 0)
        {
            // dynamic allocation of a left vector
            left = new double[coefficients_num];
            // dynamic allocation of a right vector
            right = new double[coefficients_num];
        }
    } // constructor Inequality
}; // struct Inequality

/* Implementation of a system of m inequalities,
   each one of order k */
struct SOKOI // SOKOI - system of k-th order inequalities
{
    int inequalities_num, // number of inequalities in a system
        coefficients_num; // order of inequality
                          // (number of coefficients in inequality)
    Inequality *inequalities_vector; // pointer to array
                                     // of inequalities
    SOKOI (int m=0, int k=0) // constructor
        : inequalities_vector (NULL)
    {
        inequalities_num = m; // set number of inequalities
        coefficients_num = k; // set number of coefficients
                              // in inequality
        if (inequalities_num > 0)
        {    // dynamic allocation of array of inequalities
            inequalities_vector = new Inequality[inequalities_num];
            // initialization of array of inequalities
            for (int i=0; i < inequalities_num; i++)
                inequalities_vector[i] =
                                Inequality (coefficients_num);
        }
    } // constructor SOKOI
}; // struct SOKOI

/* Implementation of n-th order system of inequalities */
struct SOI // SOI - system of inequalities
{
    int levels_num, // order of system (number of levels)
        *inequalities_num_in_level; // pointer to array of
                                    // numbers of inequalities
                                    // in each level
    SOKOI *systems_vector; // pointer to array of SOKOIs;
            // each SOKOI implements a subsystem consisting
            // of inequalities of equal orders
            // (inequalities which are in the same level)
    SOI (int n=0, int m[] = NULL) // constructor
        : systems_vector (NULL)
    {
        levels_num = n; // set number of levels
        inequalities_num_in_level = m; // set pointer to array of
                                       // numbers of inequalities
                                       // in each level
        if (levels_num > 0)
        {    // dynamic allocation of array of SOKOIs
            systems_vector = new SOKOI[levels_num + 1];
            // initialization of array of SOKOIs
            for (int j=1; j <= levels_num; j++)
                systems_vector[j] =
                        SOKOI (inequalities_num_in_level[j], j);
        }
    } // constructor SOI
}; // struct SOI
```

The following basic functions are used:

– double SKoAl (SOI system)
– double Recursion (SOI *father_system, int level)
– bool VectorsEqual (double vect1[], double vect2[], int size)
– bool Transform (int level, int left_num, int right_num, SOI old_system, SOI &new_system)
– void DeleteSoi (SOI *ptr_SOI)
– double Integration (SOI system)

In addition, the following auxiliary functions are utilized:

– void Auxiliary (double in_left[], double in_right[], int in_level, Inequality &new_inequality, int &out_level)
– void CopyIneq (Inequality in_inequality, Inequality &out_inequality)
– void DeleteIneq (Inequality inequality)
– bool FirstLevel (double old_left_0, double old_right_0, double old_left_1, double old_right_1, SOI &new_system)

SKoAl is the basic function which realizes the algorithm. Its input is a system of inequalities (type **SOI**) and its output is a numeric value of the integral.

As follows from Algorithm 1, the algorithm is an iterative process accompanied by the decreasing of parameter r (r determines the type of the system). This iterative process is provided by function **Recursion** (see Listing 2) which is a basis for function **SKoAl**.

Listing 2. `double Recursion (SOI *father_system, int level)`

```
{
    double sum = 0; // initialization of the calculated integral
    if (level > 1) // system is not one-type
    {
        /* Scanning all left and right parts */
        for (int j = 0; // scanning right parts
                j < (*father_system).inequalities_num_in_level
                                                    [level];
                j++)
        {
```

```
// scanning all previous right parts
for (int kj = 0; kj < j; kj++)
    // comparison of a right part with previous one
    if (VectorsEqual ( (*father_system).
                         systems_vector[level].
                           inequalities_vector[kj].
                             right,
                       (*father_system).
                          systems_vector[level].
                            inequalities_vector[j].
                              right,
                       level ) )
        break; // from "for kj"
// loop "for kj" wasn't completed because of
// finding identical right parts
if ( kj < j)
    continue; // "for j"
for (int i = 0; // scanning left parts
     i < (*father_system).inequalities_num_in_level
                                            [level];
         i++)
{
    // scanning all previous left parts
    for (int ki = 0; ki < i; ki++)
    // comparison of a left part with previous one
        if (VectorsEqual ( (*father_system).
                             systems_vector[level].
                               inequalities_vector[ki].
                                     left,
                               (*father_system).
                              systems_vector[level].
                                inequalities_vector[i].
                                      left,
                              level ) )
            break; // from "for ki"
    // loop "for ki" wasn't completed because of
    // finding identical left parts
    if (ki < i)
        continue; // "for i"
SOI *child_system = new SOI; // memory allocation
                             // for a new system
    // generating a new system that is
    // nearer to one-type
    if ( Transform (level, i, j,*father_system,
                                *child_system) )
        // recursive call on new system and adding
        // calculated integral to sum
        sum += Recursion (child_system, level - 1);
    DeleteSoi (child_system); // free a space
                              // of the new system
            } // "for i"
        } // "for j"
        return sum;
    } // if (level > 1)
    else // level == 1 (system is one-type)
        // integration over set defined by one-type system
        return Integration (*father_system);
}
```

Function **Recursion** constructs a tree in such a way that each node of this tree is an IS. The root of the tree is an original system of inequalities which enters function **SKoAl**. The tree leaves are systems of inequalities which belong to a one-type. Input parameters of **Recursion** are a pointer to **SOI** named **father_system** and an integer variable **level** that is equal to r.

In order to avoid generating duplicates of systems due to identical vectors, we use a boolean function, **VectorsEqual** in the course of the execution of **Recursion**. Input parameters of **VectorsEqual** are two numeric vectors and their size. This function compares the left/right part of an inequality currently under consideration with the left/right parts (respectively) of previously considered inequalities of the same system (as noted in Notation 4, we consider only distinct left parts and only distinct right parts.). If the corresponding parts are identical, then the next left/right part is compared with previously considered ones. If the corresponding parts are not identical, then the memory for a child of a node which is currently under consideration is allocated dynamically by means of a pointer to **SOI** named **child_system**.

The child node is intended for one of the new nonintersecting systems generated in step 4 of Algorithm 1. This system is constructed by means of a boolean function, **Transform**. The parameters of this function are (a) **level**; (b) integer variables **left_num** and **right_num** (which are substituted by the current numbers of the left and right parts, respectively, of the r-th order inequalities (see Notation 4)); and (c) two systems of inequalities (type **SOI**) named **old_system** and **new_system**. Function **Transform** is invoked by substituting structures pointed by pointers **father_system** and **child_system** instead of parameters **old_system** and **new_system**, respectively.

If a new system which is constructed by **Transform** is invalid, then the function returns $false$. Otherwise, **Transform** returns $true$, and the next recursion step of the algorithm is performed by the call of function **Recursion** with parameters **child_system** and **level** $-$ **1**. After return from this recursive call, the returned value is added to the local variable, **sum** (according to step 3 of Algorithm 1). Then (or immediately after the call of **Transform** if it returns $false$) function **DeleteSoi** frees a space allocated for a child of a currently considered node, and the value accumulated in **sum** is returned.

Recursive calls terminate when the value of **level** is equal to one, i.e., for the system belonging to a one-type. In this case, step 3 of Algorithm 1 is performed. Function **Integration** integrates over the set which is defined by the corresponding system and returns the calculated value to **Recursion**. Then, **Recursion** returns this value to its previous copy, into which this value is added to **sum**.

The final value of **sum** is returned by **Recursion** to function **SKoAl**, and is returned by **SKoAl** as a result.

Other functions are used in function **Transform**.

Function **Auxiliary** realizes transformation of an ordered pair of vectors into the double inequality (this procedure is described in Notation 3). Its input parameters named **in_left**, **in_right**, and **in_level**, correspond to the pair of vectors and to its order, respectively. The output parameters are structure **new_inequality** of type **Inequality** and integer variable **out_level**. They correspond to the constructed double inequality and to its order, respectively.

Function **CopyIneq** copies content of an input structure **in_inequality** (type **Inequality**) into a new structure **out_inequality**. This function is used for copying inequalities into **new_system** of function **Transform**.

Function **DeleteIneq** frees a space allocated for temporary objects of type **Inequality** in function **Transform**. This function is applied in function **DeleteSoi** as well.

A boolean function, **FirstLevel** constructs an inequality of the first order in a system which is generated by function **Transform**. If this inequality has no solutions, then **FirstLevel** returns $false$ into **Transform**, and the corresponding system is declared invalid.

The iterative process provided by function **Recursion** can be visualized as a recursion tree the nodes of which are recursive calls. And in this particular case, each recursion step includes a dynamic allocation. That is, as stated above, function **Recursion** also constructs a real tree the nodes of which are systems of inequalities implemented by structures of type **SOI**. But we need not keep all the nodes of this tree in the memory simultaneously. Before generating a new system of r-type, the previous r-type system is erased from memory. Thus, at most n nodes (a path from the the root to a leaf) are kept in memory simultaneously.

In fact, our algorithm is based on depth-first search (DFS) (see [1], [3]). However, in this particular case, a node is created directly before it is visited. The call of **Transform** corresponds to the visit. After all children of a node have been visited, the node is deleted by **DeleteSoi**.

Let's estimate the maximum number of nodes in our tree. Consider the n-th order system that has m inequalities. Suppose all m inequalities are of order n, i.e., the root of the tree has the maximum size which is possible. Also, suppose that a system of r-type generates only systems of $r-1$-type in every recursion step. The last supposition is that function **VectorsEqual** always returns $false$, i.e., $ndlp$ is equal to $ndrp$ all the time. In such a case, the number of nodes is

$$1+\sum_{i=2}^{n}\prod_{j=2}^{i}\left(2^{j-2}m-2^{j-2}+1\right)^2>\sum_{i=1}^{n}m^{2(i-1)}=\frac{m^{2n}-1}{m^2-1}.$$

However, the actual number of nodes which are kept in memory simultaneously is determined by the height of the tree and does not exceed n. It is clear that the sizes of these nodes are different. Corresponding computations show that in this case, the amount of required memory is $O\left(m2^n\right)$.

3 The Illustrative Example

In order to utilize the software described in section 2 for calculating volumes of convex polyhedrons in E_4, the simplified version of function **Integration** (mentioned in the previous section) which calculates the integral

$$\int\limits_{a_1}^{a_2} \int\limits_{a_3+a_4x_1}^{a_5+a_6x_1} \int\limits_{a_7+a_8x_1+a_9x_2}^{a_{10}+a_{11}x_1+a_{12}x_2} \int\limits_{a_{13}+a_{14}x_1+a_{15}x_2+a_{16}x_3}^{a_{17}+a_{18}x_1+a_{19}x_2+a_{20}x_3} 1\ dx_4dx_3dx_2dx_1$$

can be applied. This integral is the volume of the convex set corresponding to the following one-type system of inequalities:

$$\left\{ \begin{array}{c} ([a_{13}, a_{14}, a_{15}, a_{16}], [a_{17}, a_{18}, a_{19}, a_{20}]) \\ ([a_7, a_8, a_9], [a_{10}, a_{11}, a_{12}]) \\ ([a_3, a_4], [a_5, a_6]) \\ ([a_1], [a_2]) \end{array} \right. .$$

With the help of these tools, we solve the following simple probabilistic problem.

Problem 1. Calculate the probability that the polynomial equation with random coefficients

$$c_1(\omega) + c_2(\omega)x + c_3(\omega)x^2 + c_4(\omega)x^3 = 0,$$

where $c_1(\omega)$, $c_2(\omega)$, $c_3(\omega)$, $c_4(\omega)$ are independent random variables uniformly distributed in the interval $[-1;1]$, has one or three real roots belonging to the interval $[0;1]$.

Obviously, the problem can be reduced to calculating the volumes of two convex sets in E_4 (the volumes are equal). The first set is defined by the system of inequalities

$$\begin{cases} c_1 + c_2 + c_3 + c_4 > 0 \\ c_1 < 0 \\ -1 < c_1 < 1 \\ -1 < c_2 < 1 \\ -1 < c_3 < 1 \\ -1 < c_4 < 1 \end{cases} .$$

The second one is

$$\begin{cases} c_1 + c_2 + c_3 + c_4 < 0 \\ c_1 > 0 \\ -1 < c_1 < 1 \\ -1 < c_2 < 1 \\ -1 < c_3 < 1 \\ -1 < c_4 < 1 \end{cases} .$$

The first system can be rewritten in the following form:

$$\begin{cases} -1 < c_4 < 1 \\ -c_1 - c_2 - c_3 < c_4 < 1 \\ -1 < c_3 < 1 \\ -1 < c_2 < 1 \\ -1 < c_1 < 0 \end{cases} .$$

This system has been expressed as a system of double inequalities. The volume calculated by means of our software is equal to 2.79167. This result has been checked by the Monte-Carlo method and appears to be correct.

4 Conclusion and Future Work

We have presented a new algorithm for numerical integration over a convex polyhedron in the n-dimensional Euclidean space defined by a system of linear inequalities. We have described the software routines implementing the algorithm and estimated the memory costs required for their realization. We intend to estimate the running time of the algorithm as well. We also intend to determine maximum values of both n and the number of inequalities for which our algorithm is practically realizable. In addition, we plan to compare our algorithm with alternative methods based on various criteria.

References

1. *Algorithms and Theory of Computation Handbook*, edited by M. J. Atallah. CRC Press, Boca Raton (1999)
2. A. T. Bharucha-Reid and M. Sambandham: *Random Polynomials*. Academic Press (1986)
3. T. H. Cormen, C. E. Leiseron, and R. L. Rivest: *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts (1994)
4. M. H. Kalos and P. A. Whitlock: *Monte Carlo Methods*. John Wiley & Sons (1986)
5. W. H. Press, S. A. Tenkovsky, W. T. Vetterling, and B. P. Flannery: *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press (1992)
6. J. R. Price: *Numerical Methods, Software and Analysis*. Academic Press (1993)

A Matlab Implementation of an Algorithm for Computing Integrals of Products of Bessel Functions

Joris Van Deun and Ronald Cools

Dept. of Computer Science, K.U. Leuven, B-3001 Leuven, Belgium
{joris.vandeun, ronald.cools}@cs.kuleuven.be

Abstract. We present a Matlab program that computes infinite range integrals of an arbitrary product of Bessel functions of the first kind. The algorithm uses an integral representation of the upper incomplete Gamma function to integrate the tail of the integrand. This paper describes the algorithm and then focuses on some implementation aspects of the Matlab program. Finally we mention a generalisation that incorporates the Laplace transform of a product of Bessel functions.

1 Introduction

Integrals of products of Bessel functions arise in a wide variety of problems from physics and engineering. Many of the integrals that occur in the literature contain only one or two Bessel functions, e.g. in computing the filter loss coefficient in optical fibre technology [11], surface displacement in dynamic pavement testing [16], antenna theory [17], gravitational fields of astrophysical discs [5], crack problems in elasticity [22], particle motion in an unbounded rotating fluid [6,21], theoretical electromagnetics [8] or distortions of nearly circular lipid domains [20]. Techniques to compute this type of integrals are discussed in [2,14,22]. Analytic expressions for some special cases can be found in [19,26].

The more general case of a product of an arbitrary number of Bessel functions is far more difficult to handle and there are less analytic results known. For three or four factors we refer to [15,19,26]. The last reference also contains examples of integrals with an arbitrary number of factors. The numerical computation of these integrals has not gained much attention, even though they occur in several applications, ranging from Feynman diagrams in nuclear physics [10] to quantum field theory [7], speech enhancement [13] and scattering theory [9].

In [25] we presented an algorithm to compute integrals of the form

$$I(\boldsymbol{a},\boldsymbol{\nu},m) = \int_0^\infty x^m \prod_{i=1}^k J_{\nu_i}(a_i x)dx \qquad (1)$$

where the coefficients $a_i \in \mathbf{R}_0^+$ (we use $\boldsymbol{a}$ to denote the vector which contains these coefficients), the orders $\nu_i \in \mathbf{R}$, the power $m \in \mathbf{R}$ and $\sum_{i=1}^k \nu_i + m > -1$. This last condition assures that a possible singularity in zero is integrable. However, if some of the ν_i are negative integers, this condition may not be satisfied, even though the integral exists. This follows from formula 9.1.5 in [1] which reads as follows,

$$J_{-n}(x) = (-1)^n J_n(x).$$

© Springer-Verlag Berlin Heidelberg 2006

This case is detected by the program and the formula is used to transform negative integer orders to positive integer orders.

The case of negative coefficients a_i can always be reduced to the case of positive coefficients using [1, p. 360]

$$J_\nu(-z) = e^{\nu\pi\mathbf{i}} J_\nu(z),$$

where $\mathbf{i}$ is the imaginary unit, $\mathbf{i}^2 = -1$. We leave it to the user to do this transformation, so that the output of our program is always a real number.

Our algorithm has been implemented in the Matlab program `besselint`, which also works under Octave. For more information about the different versions of Matlab and Octave we tested, we refer to the section about backward compatibility. We also mention that our program solves one of the additional problems arising from Trefethen's 100-Digit Challenge [23]. Problem 8 in Appendix D of the book [4] concerns the evaluation of the integral

$$\int_0^\infty x J_0(x\sqrt{2}) J_0(x\sqrt{3}) J_0(x\sqrt{5}) J_0(x\sqrt{7}) J_0(x\sqrt{11}) dx, \tag{2}$$

which is just a special case of the more general formula (1).

In the next section we describe how the algorithm works. In the rest of the paper we then discuss some implementation issues particularly related to the fact that the program is meant to be used under Matlab or Octave and we mention some possible improvements and generalisations.

2 Description of the Algorithm

We provide a brief overview of the algorithm. For more details we refer the reader to [25].

The main difficulties in computing (1) come from the irregular oscillatory behaviour and possible slow decay of the integrand. We cannot just truncate the integral at a finite value and assume that the remaining contribution is negligible; the infinite part needs special treatment. This is illustrated in Figures 1 and 2, which show the integrand of formula (2).

The approach we take is to split the integral at a breakpoint x_0 and approximate the finite and infinite part separately. The determination of this breakpoint will be discussed below.

For the finite part, the interval $[0, x_0]$ is divided into a number of subintervals roughly proportional to the number of zeros in the integrand. Each subinterval is then integrated numerically using a Gauss-Legendre quadrature rule. The weights and nodes have been hard-coded into the program to speed up performance. In most cases we need a 15–point rule to reach full precision, followed by a 19–point rule to obtain the error estimate. From formula 9.1.7 in [1] it follows that the integrand $f(x)$ satisfies

$$f(x) = x^p \sum_{i=0}^{\infty} \alpha_i x^i, \quad x \to 0,$$

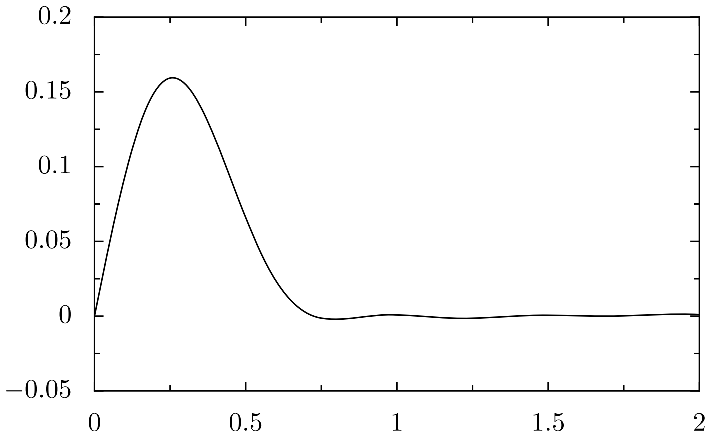


Fig. 1. Plot of the integrand of formula (2) near $x = 0$

(the exact values of α_i are not relevant), where $p = \sum_{i=1}^{k} \nu_i + m$. If p is not a positive integer, there is an algebraic singularity in 0 which we can remove by extrapolating in the sense of Richardson. This has also been incorporated in the program.

For the infinite part we use the asymptotic expansion of $J_\nu(x)$ as given in [26, p. 199]. Taking $n + 1$ terms in each expansion leads to an approximation of the form

$$x^m \prod_{i=1}^{k} J_{\nu_i}(a_i x) \sim 2\Re \left\{ x^m \sum_j e^{\mathbf{i}\eta_j x} F_{n,j}(\boldsymbol{\nu}, \boldsymbol{a}x) \right\}$$

where $\Re$ denotes the real part, $\eta_j = a_1 \pm a_2 \pm \cdots \pm a_k$ and the sum is over all possible combinations. The functions $F_{n,j}$ consist of polynomials of degree $k(2n+1)$ in $1/x$ times $x^{-k/2}$. Integrating this approximation, we have to compute

$$\int_{x_0}^{\infty} e^{\mathbf{i}\eta_i x} x^{m-k/2-j} dx, \quad i = 1, 2, \ldots, 2^{k-1}, \quad j = 0, 1, \ldots, k(2n+1). \tag{3}$$

which we can do analytically, using the upper incomplete Gamma function. This function is defined as [1, p. 260]

$$\Gamma(a, x) = \int_x^{\infty} t^{a-1} e^{-t} dt,$$

and can be extended to arbitrary complex a and x by analytic continuation. This way we may write

$$\int_{x_0}^{\infty} e^{\mathbf{i}\eta_i x} x^{m-k/2-j} dx = \left(\frac{\mathbf{i}}{\eta_i} \right)^{m-k/2-j+1} \Gamma(m - k/2 - j + 1, -\mathbf{i}\eta_i x_0). \tag{4}$$

The accuracy of the approximation for the infinite part depends on the breakpoint x_0 and on the order of the asymptotic expansion n, and we can estimate this accuracy

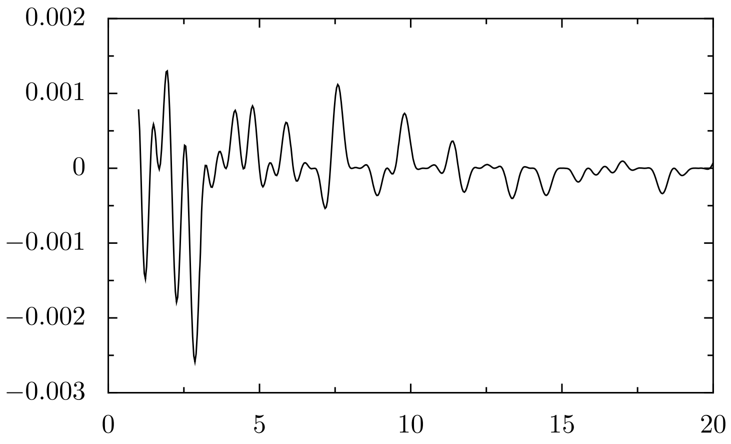


Fig. 2. Plot of the tail of the integrand of formula (2)

(as a function of x_0 and n) based on a first order error analysis. It then turns out that for a fixed accuracy there are always infinitely many (x_0, n)-pairs we can choose from. We therefore introduced a cost function $\chi(x_0, n)$ which describes the computational effort of the algorithm in terms of the number of function evaluations of the Bessel and incomplete Gamma functions,

$$\chi(x_0, n) = 2^k n t_{GJ} + \frac{x_0}{\pi}\frac{N}{2}\sum_{j=1}^{k} a_j.$$

The parameter N is the average length of the vector argument in a call to the Bessel function. This will be explained in the next section. The (machine-dependent) constant t_{GJ} is the relative efficiency of computing the incomplete Gamma function compared to computing the Bessel function. We look at this constant in more detail in the next section. The parameters x_0 and n are now chosen such that they minimise $\chi(x_0, n)$ under the constraint that the (estimated) error does not exceed a required threshold. It is up to the user to specify the absolute or relative error tolerance.

3 Determining the Parameter t_{GJ}

When introducing the cost function $\chi(x_0, n)$, we assumed that the computational effort is dominated by the evaluations of the Bessel and incomplete Gamma function. In our program, we use Matlab's `besselj` function, based on Amos' original Fortran code [3], to compute the Bessel functions. For the incomplete Gamma function we use `igamma`, a Fortran-to-Matlab conversion of the program from [12], which computes a continued fraction approximation by forward recurrence. To test the assumption that the cost is dominated by these two functions, we run `besselint` on a set of 21 examples which are thought to be representative (in the sense that they do not use extremely large values for the parameters or the number of factors in the integrand). The values of $\boldsymbol{a}$,

Table 1. Values of $\boldsymbol{a}$, $\boldsymbol{\nu}$, m and $I(\boldsymbol{a},\boldsymbol{\nu},m)$ for the examples from the test set. If more than 10 digits are shown, then the least significant one is properly rounded; otherwise the given value is exact

	$\boldsymbol{a}$	$\boldsymbol{\nu}$	m	$I(\boldsymbol{a},\boldsymbol{\nu},m)$
1	1	$\frac{1}{3}$	0	1
2	1	$-\frac{1}{4}$	$\frac{1}{3}$	0.4699242939646020
3	1	0	2	-1
4	1	0	4	9
5	$[1,5]$	$[0,1]$	0	0.2
6	$[5,1]$	$[0,1]$	0	0
7	$[1,3]$	$[-\frac{1}{2}\,\frac{1}{3}]$	$\frac{1}{6}$	0.4875332490256343
8	$[3,1]$	$[-\frac{1}{2}\,\frac{1}{3}]$	$\frac{1}{6}$	0
9	$[\sqrt{2},\sqrt{3},\sqrt{5}]$	0	1	0.1299494668722794
10	$[\sqrt{2},\sqrt{3},\sqrt{11}]$	0	1	0
11	$[\sqrt{2},\sqrt{3},\sqrt{5}]$	1	0	0.1423525086834354
12	$[\sqrt{2},\sqrt{3},\sqrt{5}]$	$[1,2,-3]$	1	-0.1150621628914800
13	$[\sqrt{2},\sqrt{3},\sqrt{11}]$	$[1,2,-3]$	1	0
14	$[\sqrt{2},\sqrt{3},\sqrt{5},\sqrt{7}]$	0	1	0.1104110282210471
15	$[1,1,1,1]$	$\frac{5}{4}$	$-\frac{3}{2}$	0.05133002738452328
16	$[\sqrt{2},\sqrt{3},\sqrt{5},\sqrt{7},\sqrt{11}]$	0	1	0.06106434990872167
17	$[\sqrt{2},\sqrt{3},\sqrt{5},\sqrt{7},\sqrt{11}]$	0	2	0.017024879933914
18	$[8,2.5,2,1.5,1]$	1	-2	0
19	$[8,2.5,2,1.5,1,0.5]$	$[\frac{1}{3} - \frac{1}{4} - \frac{1}{4} - \frac{1}{4} - \frac{1}{4} - \frac{1}{4}]$	$\frac{7}{12}$	0.5219234259420822
20	$[1,2,3]$	0	0	0.4752701735935373
21	$[\sqrt{2},\sqrt{3},\sqrt{2}+\sqrt{3}]$	0	0	0.4437109037960439

$\boldsymbol{\nu}$ and m for each of the examples, together with the value of the integral, are given in Table 1. For t_{GJ} we take the value 16, for reasons explained below. We used Matlab's profiler tool to analyse the distribution of CPU time among the various subfunctions in `besselint` (the main program which contains the 21 calls to `besselint` is called `experiment`, which is the first line in the profiler report). Table 2 reproduces the output report provided by the profiler. Computations were done in Matlab 7.0 on an Intel processor running at 2.80 GHz. Note that `besselj` and `igamma` together take up 3.45s of the total 4.53s of execution time, which is more than 76%. It seems our assumption is justified.

As we mentioned in the previous section, the parameter t_{GJ} is the relative efficiency of computing the incomplete Gamma function compared to computing the Bessel function. It is defined as the ratio t_Γ/t_J where t_Γ and t_J are the average execution times for a call to the incomplete Gamma function and Bessel function respectively. The determination of this parameter, however, presents some problems which we discuss next.

Table 2. Matlab's profiler output after calling `besselint` with the test set

Profile Summary
Generated 21-Oct-2005 13:10:04 using CPU time.

Function Name	**Calls**	**Total Time**	**Self Time**
experiment	1	4.530 s	0.040 s
besselint	21	4.490 s	0.060 s
besselint>fri	21	2.710 s	0.040 s
besselint>nqf	66	2.670 s	0.310 s
besselint>fun	1343	2.360 s	0.180 s
besselj	5470	2.180 s	0.770 s
besselint>ira	21	1.570 s	0.280 s
igamma	2027	1.270 s	0.080 s
igamma>cdh	2027	1.190 s	0.060 s
igamma>cdhs	2027	1.130 s	1.130 s
besselmx (MEX-function)	5470	0.820 s	0.820 s
besschk	5470	0.590 s	0.590 s
besselint>gop	21	0.150 s	0.050 s
besselint>optimfun	84	0.100 s	0.000 s
besselint>auxfun	84	0.100 s	0.050 s
conv	464	0.010 s	0.010 s
gammaln	252	0.050 s	0.050 s
repmat	21	0.010 s	0.010 s
datenummx (MEX-function)	2	0 s	0.000 s
profile>ParseInputs	1	0 s	0.000 s

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

The first problem in determining t_{GJ} comes from the fact that Matlab and Octave support vector operations. This means that we can call the function `besselj` with a vector argument to evaluate the Bessel function in N points at the same time. This will generally be faster than N separate calls. Obviously, we exploited this in our program. This has to be taken into account when trying to determine t_J. To determine t_{GJ} we timed 2040 calls to `igamma` and 120 calls with a vector argument of length $17 = 2040/120$ to `besselj` (for most representative examples, an average call to `besselj` in the numerical quadrature formula would contain a vector of this size). Averaging over 22 runs gives a ratio of more or less $t_{GJ} \approx 16$ in Matlab 7.0 and $t_{GJ} \approx 1357$ in Octave 2.1.69.

Another problem comes from the fact that our cost function is not exact. Even though it is clear that the dominant contribution to the computational effort comes from evaluating the incomplete Gamma function and the Bessel function, the other operations are not negligible. Apart from all this, it needs pointing out that the Matlab compiler and JIT (Just In Time) accelerator have become so sophisticated that it becomes nearly impossible to minimise execution times based on operation count. To illustrate this discussion

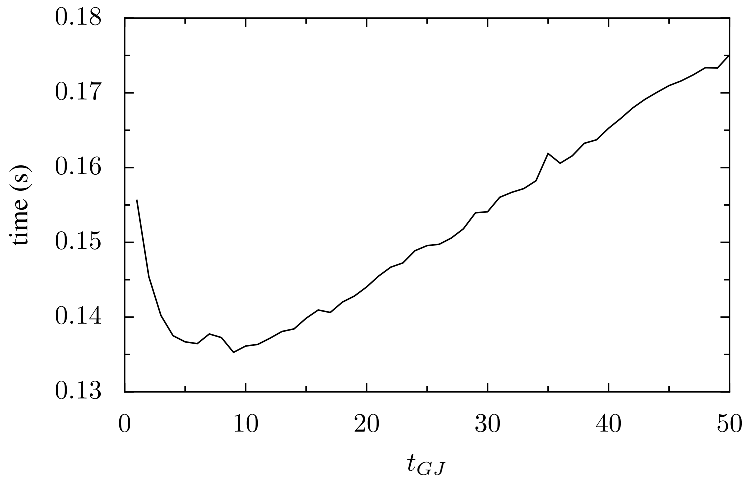


Fig. 3. Average execution time (s) for a call to `besselint` vs. t_{GJ} (Matlab)

we run `besselint` on the previous set of 21 examples, but this time for different values of t_{GJ}. Averaging over 10 runs gives the results from Figures 3 and 4. These figures give the average execution time for one (representative) call to `besselint` as a function of the parameter t_{GJ}. It can be seen that the predicted 'theoretical' values of t_{GJ} differ from the experimentally determined 'optimal' values. For the Matlab case, the difference in execution time is small, but if speed is a real concern, the user might want to repeat this test on his own machine to obtain the optimal t_{GJ}. Note that our code runs much faster under Matlab than under Octave. It is not clear what causes this difference.

4 Backward Compatibility

Our program was written to work in Matlab 7.0 and Octave 2.1.69. However, we specifically implemented it to be backward compatible with earlier versions of Matlab. Especially for a large commercial software package such as Matlab, one cannot simply assume that every user has access to the latest available version. We extensively tested our code under

- Matlab 7.0.1.24074 (R14) Service Pack 1 for Linux,
- Matlab 6.5.0.180913a (R13) for Linux and Windows,
- Matlab 6.1.0.450 (R12.1) for Windows, and
- Octave 2.1.69.

However, improving backward compatibility usually means losing some efficiency. This is particularly true when it comes to logical operations in Matlab, as we illustrate in this section.

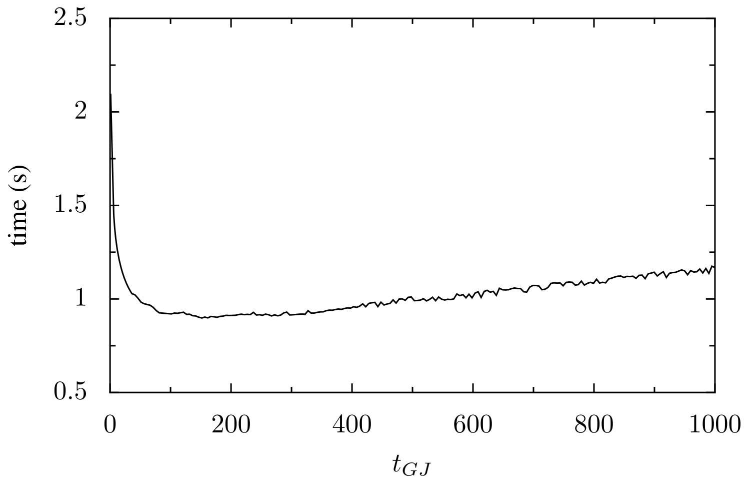


Fig. 4. Average execution time (s) for a call to `besselint` vs. t_{GJ} (Octave)

Logical expressions such as $x \vee y$ (where x and y are booleans and $\vee$ is the logical 'or') are very common in numerical software and they very often appear in conditional statements to control the flow of a program. For both the logical 'or' and 'and' there are situations where only one of the operands needs to be evaluated to determine the result. For example, in the expression $x \vee y$, if x is 'true', the result will be 'true' regardless of the value of y. In Matlab terminology, this is called 'short-circuit' logical evaluation. When there are many logical expressions in a program, this can be much more efficient than evaluating the entire expression (called 'element-wise' evaluation in Matlab). Unfortunately, there is a difference in syntax between earlier and recent versions of Matlab. From version 6.5 on, Matlab explicitly provides two different sets of logical operators, i.e. element-wise operators denoted by $\&$ and $|$, and short-circuit operators, $\&\&$ and $||$ (which is also the convention used by Octave from the beginning). However, in earlier versions of Matlab, the explicit short-circuit operators were not available and short-circuit evaluation was done automatically in certain situations (e.g. in the condition following an `if`-statement). Because of backward compatibility, we have to use the operators $\&$ and $|$, which dramatically reduces efficiency when working in more recent versions.

Table 3 was obtained in exactly the same way as Table 2, but with the only difference that all logical operators were replaced by their short-circuit versions. The total execution time for the test examples has changed from 4.69s to 3.93s, a reduction of over 13%, but if we compare the timings for `igamma`, then we note a reduction from 1.27s to 0.6s, more than 50%! The number of incomplete Gamma function evaluations increases exponentially with k (the number of Bessel functions in the integrand), so especially for integrals which contain many Bessel functions, we are sacrificing a lot of efficiency to the cause of backward compatibility. We do not know why Mathworks decided to introduce this syntactic anomaly for logical operators.

Table 3. Same as Table 2, but using short-circuit logical evaluation

Profile Summary
Generated 21-Oct-2005 13:53:21 using CPU time.

Function Name	Calls	Total Time	Self Time
experiment	1	3.930 s	0.070 s
besselint	21	3.860 s	0.070 s
besselint>fri	21	2.620 s	0.030 s
besselint>nqf	66	2.590 s	0.360 s
besselint>fun	1343	2.230 s	0.180 s
besselj	5470	2.050 s	0.540 s
besselint>ira	21	0.980 s	0.340 s
besselmx (MEX-function)	5470	0.820 s	0.820 s
besschk	5470	0.690 s	0.690 s
igamma	2027	0.600 s	0.100 s
igamma>cdh	2027	0.500 s	0.090 s
igamma>cdhs	2027	0.410 s	0.410 s
besselint>gop	21	0.190 s	0.020 s
besselint>optimfun	84	0.170 s	0.010 s
besselint>auxfun	84	0.160 s	0.080 s
gammaln	252	0.080 s	0.080 s
conv	464	0.030 s	0.030 s
repmat	21	0.010 s	0.010 s
datenummx (MEX-function)	2	0 s	0.000 s
profile>ParseInputs	1	0 s	0.000 s

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

5 Further Improvements and Generalisations

The function `besselj` essentially consists of two function calls, one to `besschk`, which checks the input arguments, and then a call to `besselmx`, which is a MEX file containing the actual Amos algorithm to compute $J_\nu(x)$. According to the profiler report, both calls take up approximately 40% of the total time for this function. Since the arguments to the Bessel functions are computed inside `besselint` and are not given by the user, this means we could speed up performance by calling `besselmx` directly instead of `besselj`, thereby skipping the call to `besschk`. This effectively reduces the computation time for `besselj` from 2.18s to 0.78s for `besselmx` (backward compatible version). Also the overhead associated with function calls is reduced. The problem with this approach, however, is that it only works in Matlab and not in Octave, which uses a different implementation. Since we want to provide a program that works both under Matlab and Octave, we have not performed this optimisation.

For the other bottleneck, `igamma`, there are two things we can do to speed up the computations. The most obvious improvement is to compute the continued fraction approximation by backward recurrence instead of forward, which is usually faster and also more stable. However, then we need a good estimate of the tail to be able to predict at what point to start the recurrence. The formulas from [27] are useless in our case, because they give asymptotic estimates which only become valid when extremely high precisions (working in multiprecision) are needed. In a Matlab environment, which uses IEEE double precision, they give unreliable results.

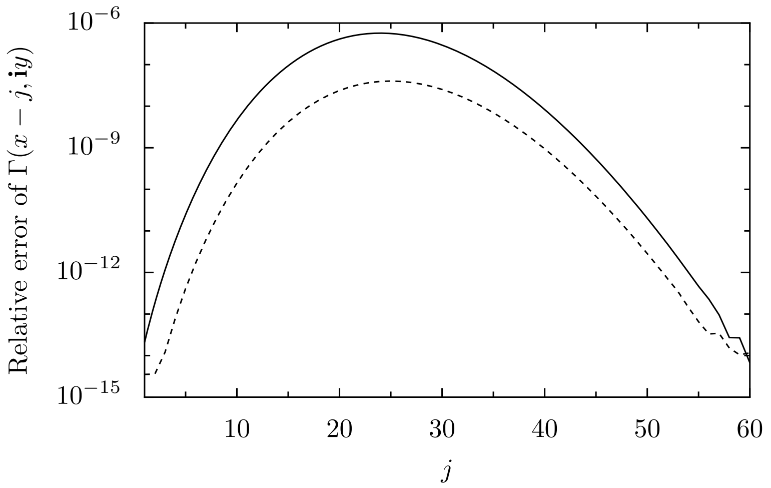


Fig. 5. Error in recurrence for $\Gamma(x-j,\mathbf{i}y)$ for $x=-1$, $j=0,\ldots,60$ and $y=25$. The solid line shows the relative error when the recurrence is computed starting from the value at the left and going to the right, the dashed line corresponds to going from right to left.

The second improvement to `igamma` comes from the recurrence relation satisfied by this function itself. Looking at equations (3) and (4), we see that a range of incomplete Gamma function evaluations are needed of the form

$$\Gamma(x-j,\mathbf{i}y), \quad j=0,1,\ldots,s$$

for certain values of x, y and s. But it follows from [1], formulas 6.5.2, 6.5.3 and 6.5.21 that we have

$$\Gamma(x-j+1,\mathbf{i}y)=(x-j)\Gamma(x-j,\mathbf{i}y)+(\mathbf{i}y)^{x-j}e^{-\mathbf{i}y}.$$

So instead of evaluating the incomplete Gamma function $2^{k-1}(k(2n+1)+1)$ times using the continued fraction expansion, we could suffice with 2^{k-1} evaluations and compute the remaining values recursively. However, it turns out that for several values of x and y, this recurrence can become unstable *in either direction*. This means we can loose (too many) digits both using forward or backward recurrence (of course, it will be *asymptotically* stable in one direction, but that is of little use when we want all values to have maximum precision). This is illustrated in Figure 5, which shows the relative error as a function of j for $x=-1$, $y=25$ and $s=60$. The correct approach here is to find the value of j for which the error reaches its maximum value (in our example

$j \approx 24$) and then use the recurrence relation in both directions, starting from that point. This is explained in more detail in [24].

Instead of using the continued fraction expansion to compute the incomplete gamma function, another possibility would be to use asymptotic expansions (at least when one or both of the parameters are large). Since no such implementation is available and since the computation of the incomplete gamma function was not our main concern, we have not yet considered this.

The present algorithm can be generalised to compute integrals of the form

$$\int_0^\infty e^{-cx} \frac{x^m}{1+dx^2} \prod_{i=1}^{k} J_{\nu_i}(a_i x)dx$$

where c is a positive real number and d is an arbitrary real number. The case $d = 0$ corresponds to the Laplace transform of a product of Bessel functions. Integrals of this type occur in several applications, e.g. in [18]. This generalisation, however, is not so straightforward (especially when $d \neq 0$) and the details will be presented elsewhere.

6 Conclusion

In the development of numerical software there is often a tradeoff to be made between speed and efficiency on the one hand, and compatibility on the other hand. We wanted to provide an implementation of our algorithm that works for older versions of Matlab and for Octave as well, but the experiments in this article clearly indicate that we can gain a lot of speed with a program that only runs under recent versions of Matlab. Based on our timings, we expect that it is possible to obtain an implementation which is (on average) roughly twice as fast as the current one. However, more research needs to be done regarding the recursive computation of the incomplete Gamma function. Theoretically, the approach presented in [24] should work fine, but a robust implementation has not yet been undertaken.

References

1. M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions With Formulas, Graphs, and Mathematical Tables*, volume 55 of *Applied Mathematics Series*. National Bureau of Standards, Washington, D.C., 1964.
2. V. Adamchik. The evaluation of integrals of Bessel functions via G–function identities. *J. Comput. Appl. Math.*, 64:283–290, 1995.
3. D. E. Amos. Algorithm 644: A portable package for Bessel functions of a complex argument and nonnegative order. *ACM Trans. Math. Software*, 12(3):265–273, 1986.
4. F. Bornemann, D. Laurie, S. Wagon, and J. Waldvogel. *The SIAM 100-Digit Challenge. A Study in High-Accuracy Numerical Computing*. SIAM, Philadelphia, 2004.
5. J. T. Conway. Analytical solutions for the newtonian gravitational field induced by matter within axisymmetric boundaries. *Mon. Not. R. Astron. Soc.*, 316:540–554, 2000.
6. A. M. J. Davis. Drag modifications for a sphere in a rotational motion at small non-zero Reynolds and Taylor numbers: wake interference and possible coriolis effects. *J. Fluid Mech.*, 237:13–22, 1992.

7. S. Davis. Scalar field theory and the definition of momentum in curved space. *Class. Quantum Grav.*, 18:3395–3425, 2001.
8. G. Fikioris, P. G. Cottis, and A. D. Panagopoulos. On an integral related to biaxially anisotropic media. *J. Comput. Appl. Math.*, 146:343–360, 2002.
9. R. Gaspard and D. Alonso Ramirez. Ruelle classical resonances and dynamical chaos: the three- and four-disk scatterers. *Physical Review A*, 45(12):8383–8397, 1992.
10. S. Groote, J. G. Körner, and A. A. Pivovarov. On the evaluation of sunset-type Feynman diagrams. *Nucl. Phys. B*, 542:515–547, 1999.
11. M. J. Holmes, R. Kashyap, and R. Wyatt. Physical properties of optical fiber sidetap grating filters: free space model. *IEEE Journal on Selected Topics in Quantum Electronics*, 5(5):1353–1365, 1999.
12. E. Kostlan and D. Gokhman. A program for calculating the incomplete Gamma function. Technical report, Dept. of Mathematics, Univ. of California, Berkeley, 1987.
13. T. Lotter, C. Benien, and P. Vary. Multichannel direction-independent speech enhancement using spectral amplitude estimation. *EURASIP Journal on Applied Signal Processing*, 11:1147–1156, 2003.
14. S. K. Lukas. Evaluating infinite integrals involving products of Bessel functions of arbitrary order. *J. Comput. Appl. Math.*, 64:269–282, 1995.
15. J. W. Nicholson. Generalisation of a theorem due to Sonine. *Quart. J. Math.*, 48:321–329, 1920.
16. J. M. Roesset. Nondestructive dynamic testing of soils and pavements. *Tamkang Journal of Science and Engineering*, 1(2):61–81, 1998.
17. S. V. Savov. An efficient solution of a class of integrals arising in antenna theory. *IEEE Antenna's and Propagation Magazine*, 44(5):98–101, 2002.
18. N. P. Singh and T. Mogi. Electromagnetic response of a large circular loop source on a layered earth: a new computation method. *Pure Appl. Geophys.*, 162:181–200, 2005.
19. N. J. Sonine. Recherches sur les fonctions cylindriques et le développement des fonctions continues en séries. *Math. Ann.*, 16:1–80, 1880.
20. H. A. Stone and H. M. McConnell. Hydrodynamics of quantized shape transitions of lipid domains. *Royal Society of London Proceedings Series A*, 448:97–111, 1995.
21. J. Tanzosh and H. A. Stone. Motion of a rigid particle in a rotating viscous flow: an integral equation approach. *J. Fluid Mech.*, 275:225–256, 1994.
22. M. Tezer. On the numerical evaluation of an oscillating infinite series. *J. Comput. Appl. Math.*, 28:383–390, 1989.
23. L. N. Trefethen. The $100, 100-Digit Challenge. *SIAM News*, 35(6):1–3, 2002.
24. J. Van Deun and R. Cools. A stable recurrence for the incomplete Gamma function with imaginary second argument. Technical Report TW441, Department of Computer Science, K.U.Leuven, November 2005.
25. J. Van Deun and R. Cools. Algorithm 8XX: Computing infinite range integrals of an arbitrary product of Bessel functions. *ACM Trans. Math. Software*, 2006. To appear.
26. G. N. Watson. *A treatise on the Theory of Bessel Functions*. Cambridge University Press, 1966.
27. S. Winitzki. Computing the incomplete Gamma function to arbitrary precision. In V. Kumar, M. L. Gavrilova, C. J. K. Tan, and P. L'Ecuyer, editors, *Computational Science and Its Applications – ICCSA*, volume 2667 of *Lecture Notes in Computer Science*, pages 790–798. Springer Verlag, 2003.

Computation of the Real Zeros of the Kummer Function $M(a; c; x)$

Alfredo Deaño[1], Amparo Gil[2], and Javier Segura[2]

[1] Departamento de Matemáticas. Univ. Carlos III de Madrid. 28911, Leganés (Madrid), Spain
[2] Departamento de Matemáticas, Estadística y Computación. Univ. de Cantabria. 39005, Santander, Spain

Abstract. An algorithm for computing the real zeros of the Kummer function $M(a;c;x)$ is presented. The computation of ratios of functions of the type $M(a+1;c+1;x)/M(a;c;x)$, $M(a+1;c;x)/M(a;c;x)$ plays a key role in the algorithm, which is based on global fixed-point iterations. We analyse the accuracy and efficiency of three continued fraction representations converging to these ratios as a function of the parameter values. The condition of the change of variables appearing in the fixed point method is also studied. Comparison with implicit Maple functions is provided, including the Laguerre polynomial case.

1 Introduction

Our algorithm is based on global fixed point iterations which apply to families of functions satisfying first order linear difference differential equations with continuous coefficients. The methods were described in [1,2]. The starting point of the methods is the construction of a first order system of differential equations:

$$\begin{aligned} y'(x) &= \alpha(x)y(x) + \delta(x)w(x) \\ w'(x) &= \beta(x)w(x) + \gamma(x)y(x), \end{aligned} \tag{1}$$

with continuous coefficients $\alpha(x)$, $\beta(x)$, $\gamma(x)$ and $\delta(x)$ in the interval of interest I, relating our problem function $y(x)$ with a contrast function $w(x)$, whose zeros are interlaced with those of $y(x)$. The coefficients $\delta(x)$ and $\gamma(x)$ satisfy the condition $\delta(x)\gamma(x) < 0$, which has to be met when $y(x)$ or $w(x)$ have at least two zeros in the interval I.

With these restrictions, we introduce new functions and a new variable as follows. First, we consider a change of the dependent functions:

$$y(x) = \lambda_y(x)\bar{y}(x)\,,\ w(x) = \lambda_w(x)\bar{w}(x), \tag{2}$$

with $\lambda_y(x) \neq 0$, $\lambda_w(x) \neq 0$ $\forall x \in I$ in such a way that $\bar{y}$ and $\bar{w}$ satisfy:

$$\begin{aligned} \bar{y}' &= \bar{\alpha}\,\bar{y} + \bar{\delta}\,\bar{w} \\ \bar{w}' &= \bar{\beta}\,\bar{w} + \bar{\gamma}\,\bar{y} \end{aligned} \tag{3}$$

© Springer-Verlag Berlin Heidelberg 2006

with $\bar{\delta} > 0$ and $\bar{\delta} = -\bar{\gamma}$. This is accomplished by choosing:

$$\lambda_y = \operatorname{sign}(\delta)\lambda_w \sqrt{-\delta/\gamma} \tag{4}$$

It is obvious that the new functions $\bar{y}(x)$ and $\bar{w}(x)$ have the same zeros as $y(x)$ and $w(x)$. Considering now a change of variables:

$$z(x) = \int \bar{\delta}(x)dx, \tag{5}$$

the system reads:

$$\begin{pmatrix} \dot{\bar{y}} \\ \dot{\bar{w}} \end{pmatrix} = \begin{pmatrix} \bar{a} & 1 \\ -1 & \bar{b} \end{pmatrix} \begin{pmatrix} \bar{y} \\ \bar{w} \end{pmatrix}, \tag{6}$$

where $\bar{a} = \bar{\alpha}/\bar{\delta}$, $\bar{b} = \bar{\beta}/\bar{\delta}$ and dots mean derivative with respect to z. Then, the ratio $H(z) = \bar{y}/\bar{w}$ satisfies the first order non-linear ODE:

$$\dot{H} = 1 + H^2 - 2\eta H, \tag{7}$$

where $\eta = (\bar{b} - \bar{a})/2$.

From Eq.(7) it is possible to build a fixed point iteration [1]:

$$T(z) = z - \arctan\left(H(z)\right), \tag{8}$$

which converges globally to the zeros of $H(z)$ in intervals where the function η does not change sign [1]. These zeros are the same as those of the function $\bar{y}(z)$, and undoing the change of variable we obtain the zeros of $y(x)$ in the original variable x.

2 The Confluent Hypergeometric Function $M(a;c;x)$

We consider the Kummer differential equation:

$$xy''(x) + (c - x)y'(x) - ay(x) = 0. \tag{9}$$

This equation has a regular singular point at the origin and an irregular singularity at infinity. The regular solution around the origin reads

$$M(a;c;x) = \sum_{n=0}^{\infty} \frac{(a)_n}{(c)_n} \frac{x^n}{n!}, \tag{10}$$

where $(a)_n = a(a+1)(a+2)\ldots(a+n-1)$ is the usual Pochhammer symbol. This power series has radius of convergence equal to infinity, and defines for all complex values of a, c, x (except if $c = 0, -1, -2, \ldots$) a function which is a hypergeometric function of type ${}_1F_1$ and it is known as *confluent hypergeometric function of the first kind* or *Kummer function*.

In this paper we will focus on real values of a, c and x. Moreover, we can restrict ourselves to $x > 0$, because by Kummer transformations [3]:

$$M(a;c;x) = e^x M(c-a;c;-x). \tag{11}$$

The positive real zeros of the function $M(a;c;x)$ are bounded, a property that can be proved by writing (9) in normal form. Indeed, the functions $w(x) = x^{c/2}e^{-x/2}y(x)$ with $y(x)$ solution of (9), satisfy

$$w''(x) + \left(-\frac{1}{4} + \frac{c-2a}{2x} + \frac{c(2-c)}{4x^2}\right)w(x) = 0. \tag{12}$$

It is straightforward to check that the turning points are:

$$x_{\pm} = c - 2a \pm \sqrt{(c-2a)^2 + c(2-c)}. \tag{13}$$

When $x > x_+$ and when $x < x_-$ the independent term in (12) is negative, and as a consequence of Sturm theorems [10] the function $M(a;c;x)$ can only have at most one zero in that region. More precisely, in [4] it is shown that if $M(a;c;x)$ has at least two positive real zeros then the parameters verify the so called *oscillatory conditions*:

$$a < 0, \qquad c - a > 1 \tag{14}$$

It is worth noticing that if $a = -n$ is a negative integer we obtain the classical Laguerre polynomials $L_n^{(\alpha)}(x)$, where $c = \alpha + 1$.

2.1 Systems of DDEs for $M(a;c;x)$

Let us illustrate the construction of a system of DDEs for the functions $M(a;c;x)$. These functions satisfy the following relation (Eq. 13.4.11 of [3]):

$$xM'(a;c;x) = (a+x-c)M(a;c;x) + (c-a)M(a-1;c;x) \tag{15}$$

and the following three-term recurrence relation (TTRR) (Eq.13.4.1 of [3]):

$$aM(a+1;c;x) = (2a-c+x)M(a;c;x) + (c-a)M(a-1;c;x) \tag{16}$$

Now, using the notation $y_n \equiv M(a+n;c;x)$, we can write the following system of difference-differential equations (1), as stated in [7]:

$$\begin{aligned} y_n' &= \frac{a+n+x-c}{x}y_n + \frac{c-a-n}{x}y_{n-1} \\ y_{n-1}' &= -\frac{a+n-1}{x}y_{n-1} + \frac{a+n-1}{x}y_n, \end{aligned} \tag{17}$$

where the second DDE can be obtained from the first DDE (15) by replacing $n \to n-1$ and using the TTRR (16) for expressing y_{n-2} in terms of y_n and y_{n-1}.

In this example the dependence on n is located in the first parameter of the hypergeometric function, but it is clear that other choices are available. By denoting $a_n \equiv a + k\,n$, $c_n \equiv c + m\,n$ and $y_n \equiv M(a_n;c_n;x)$ we will have different sets of DDEs for different selections of (k,m). As explained in the introduction, these DDEs are the starting point for building the fixed point iterations.

3 Ratios of Hypergeometric Functions and Continued Fractions

In order to apply the fixed point method for the real zeros of the function $M(a;c;x)$ we will need to compute the following ratios:

$$R_{1,1}(x) := \frac{M(a+1;c+1;x)}{M(a;c;x)}, \tag{18}$$

when $x < c-a$, and

$$R_{1,0}(x) := \frac{M(a+1;c;x)}{M(a;c;x)}, \tag{19}$$

when $x > c-a$. The selection of the x-range is based on the efficiency of the associated fixed point methods, as explained in [7].

As it is well known, confluent hypergeometric functions satisfy three-term recurrence relations in any direction of increasing (decreasing) parameters (with integer values), and the associated continued fractions converge to the ratio of minimal solutions of the recurrence by Pincherle's theorem [8,10] and when the recurrence has a minimal solution. However, great care is needed in order to avoid situations of pseudoconvergence of the continued fractions, as explained in [5], that may result in a loss of precision in the computation or even in the computation of a wrong ratio of functions.

When $x < c-a$ we have to use the iteration $(1,1)$ (increasing both parameters), and the function $M(a;c;x)$ is minimal in that direction. No pseudoconvergence is expected, since this phenomenon is present when x is larger than c, as can be seen in [5]. Therefore we can use the continued fraction that stems from the recursion:

$$H^{(1)}(x) := \frac{c}{c-x+}\;\frac{(a+1)x}{c+1-x+}\;\frac{(a+2)x}{c+2-x+}\;\cdots, \tag{20}$$

This continued fraction converges to the ratio $R_{1,1}$ for $x \in \mathbb{R}$. We note that if the parameter a is a negative integer then the confluent hypergeometric function reduces to a polynomial of Laguerre type, and therefore the continued fraction is finite.

When $x > c-a$ we have to use the ratio corresponding to the $(1,0)$ iteration. In this case the CF from the $(1,0)$ recursion can not be used for computing zeros of $M(a;c;x)$, because the function $M(a+n;c;x)$ is dominant when $n \to \infty$. However, we can use the QD algorithm [8] to construct the following C-fraction from the power series expansion of (19):

$$H^{(2)}(x) := a_0 + \frac{a_1 x}{1+}\;\frac{a_2 x}{1+}\;\frac{a_3 x}{1+}\cdots, \tag{21}$$

where $a_0 = 1$, $a_1 = 1/c$, and:

$$a_{2m} = \frac{a+1-c-m}{(c+2m-2)(c+2m-1)}, \qquad m \geq 1 \tag{22}$$

$$a_{2m+1} = \frac{a+m}{(c+2m-1)(c+2m)}, \qquad m \geq 1 \tag{23}$$

This CF converges to the ratio $M(a+1;c;x)/M(a;c;x)$ on compact subsets of $\mathbb{R}$ except for the zeros of $M(a;c;x)$. We refer the reader to [8, pg. 313] for a similar result for the ratio in the $(1,1)$ direction. However, it is important to note that this continued fraction exhibits pseudoconvergence [5] for large x, and therefore should not be used to compute the largest zeros. For moderate values of x, on the other hand, is a useful expression, as we show in Section 4.

When the variable x is large it is possible to use the fact that the function $M(a;c;x)$ is minimal in the $(0,1)$ direction, that is, when we increase the parameter c. This leads to the following continued fraction:

$$\frac{c}{c+x+}\;\frac{-(c+1-a)x}{c+1+x+}\;\frac{-(c+2-a)x}{c+2+x+}\;\ldots, \tag{24}$$

This continued fraction converges to the ratio $M(a;c+1;x)/M(a;c;x)$, and it does not present pseudoconvergence if the oscillatory conditions are fulfilled. Once computed, we can obtain the ratio in the $(1,0)$ direction by means of a three-term recurrence relation [3]:

$$\frac{M(a+1;c;x)}{M(a;c;x)} = \frac{c}{c - x\dfrac{M(a+1;c+1;x)}{M(a+1;c;x)}} \tag{25}$$

This gives the following continued fraction for the ratio $R_{1,0}(x)$:

$$H^{(3)}(x) := \frac{1}{1+}\;\frac{-x}{c+x+}\;\frac{-(c-a)x}{c+1+x+}\;\frac{-(c+1-a)x}{c+2+x+}\;\frac{-(c+2-a)x}{c+3+x+}\;\ldots, \tag{26}$$

4 Computational Aspects of Continued Fractions

The algorithm uses the modified Lentz-Thompson method [9] to compute the continued fractions (20), (21) and (26). The first one is used when $x < c-a$, and the other two when $x > c-a$.

First, we perform an analysis of the accuracy of the continued fractions used in the algorithm, comparing with the internal Maple subroutine `KummerM(a,c,x)`. In the following plots we have fixed several values of the parameter a and we have used a random sweep in the plane (c,x). The threshold of accuracy has been set to five digits less than the working precision (40 digits). Dark dots denote points where the relative error between the continued fraction used and Maple is smaller than this quantity, and grey dots when it is larger. We have also plotted the line $x = c-a$ in the case of the first continued fraction, and both $x = c-a$ and $x = c-2a$ in the other two cases.

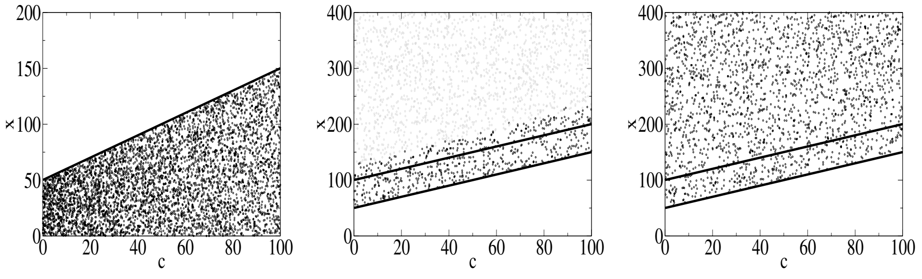


Fig. 1. Plots for $a = -50.1$, with a sweep of 5000 random points. Left: accuracy test for the CF (20) when $x < c - a$. Center: accuracy test for the CF (21) when $x > c - a$. We observe a loss of accura cy of the CF when x becomes large. Right: accuracy test for the CF (24) when $x > c - a$. We include the lines $x = c - a$ (left) and both $x = c - a$ and $x = c - 2a$ (center and right).

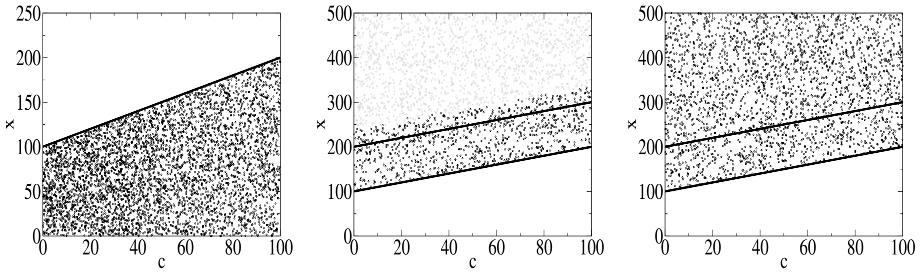


Fig. 2. Plots for $a = -100.1$, with a sweep of 5000 random points. Left: accuracy test for the (20) when $x < c - a$. Center: accuracy test for the CF (21) when $x > c - a$. We observe a loss of accuracy of the CF when x becomes large. Right: accuracy test for the CF (24) when $x > c - a$. We include the lines $x = c - a$ (left) and both $x = c - a$ and $x = c - 2a$ (center and right).

The graphics suggest that the first continued fraction (20) can be used when $x < c - a$. When $x > c - a$ and x is not too large the second one is correct, but the attainable accuracy deteriorates when x becomes large. This loss of accuracy corresponds to the continued fraction (21), because the third continued fraction (24) agrees with Maple subroutine in the whole region, as can be seen in the graphics on the right.

In the general algorithm we will change from (21) to (24) when $x = c - 2a$. This choice seems to be safe according to numerical experiments, and it can be justified by performing a canonical contraction of (21), as explained in [8, pg.83]. In the resulting continued fraction both the numerators and the denominators change of sign (from negative to positive), taking into account (14). When $x < c-2a+1$ the change in the denominator occurs when the numerators are still negative, and therefore there is no risk of pseudoconvergence (see [5] for the criterion

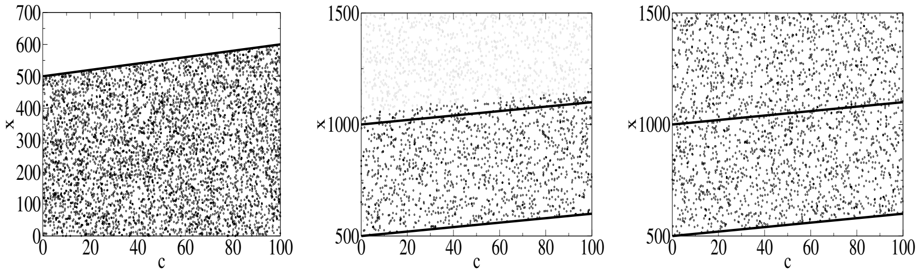


Fig. 3. Plots for $a = -500.1$, with a sweep of 5000 random points. Left: accuracy test for the CF (20) when $x < c - a$. Center: accuarcy test of the CF (21) when $x > c - a$. We observe a loss of accuracy of the CF when x becomes large. Right: accuracy test for the CF (24) when $x > c - a$. We include the lines $x = c - a$ (left) and both $x = c - a$ and $x = c - 2a$ (center and right).

of signs). On the other hand, when $x > c - 2a + 1$ the denominator changes from negative to positive when the numerators are already positive. This is a disctintive sign of pseudoconvergence and causes the loss of accuracy observed in the plots.

4.1 Timings

From the point of view of accuracy, numerical tests indicate that it is possible to use the third continued fraction (24) when $x > c - a$. However, the continued fraction (21) seems to be more efficient for moderate values of x in terms of CPU time, so we have kept the three continued fractions in the program to test CPU time. If we compare both continued fractions in the zone $c - a < x < c - 2a$ we obtain that (21) is generally faster, as shown in Figure 4.

Once accuracy is tested we have to compare CPU times for continued fraction evaluation and Maple, in order to establish which is the most efficient method depending on the parameters.

Timing tests have been performed again fixing different values of a, carrying out random sweeps in c and x and comparing the result from the continued fraction method with the intrinsic Maple procedure (Figure 5). In order to ensure a significant number of calls to both algorithms repeating loops are included, which use slightly different values of the parameters in each call, in such a way that the values are actually recomputed, and not stored in memory. Black dots represent points where the continued fraction is faster, whereas grey dots indicate that Maple is more efficient.

After several tests for fixed values of a the continued fraction method seems to be superior when x and/or a are large, whereas Maple is faster for large values of c. A final least squares fit is calculated to obtain a condition in terms of the parameters of the function that will enable us to choose the best method in the general algorithm.

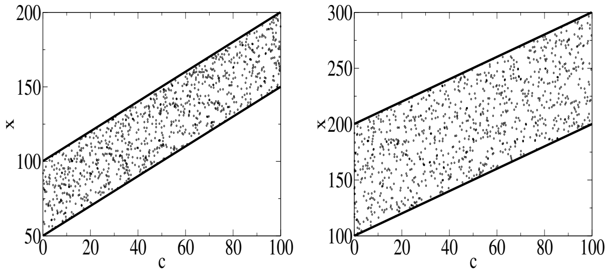


Fig. 4. Timings of continued fractions (21) and (24) in the region $c-a<x<c-2a$. We include both lines. Left, $a=-50.1$. Right: $a=-100.1$. 1000 random points have been used in the region $c-a<x<c-2a$, and points indicate where (21) is faster.

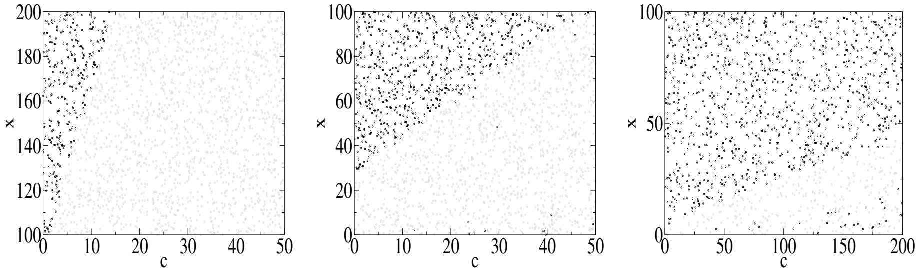


Fig. 5. Comparison between continued fractions (20), (21), (24) and Maple in the final scheme. Left: $a=-50.1$. Center: $a=-100.1$. Right: $a=-500.1$. 2000 random points have been used in each case. Black points indicate where continued fraction is faster, grey points where Maple is more efficient.

Apart from the numerical results shown before in the plots, similar graphics have been generated in the cases $a=-75.1$, -125.1, -150.1, -200.1, -250.1, -300.1, -325.1, -350.1, -400.1, -425.1, -450.1, using random sweeps in the plane (c,x), in order to obtain a finer fit.

For each value of a, a straight line in the plane (c,x) divides approximately the zones where Maple or the continued fraction are superior. These lines have the form $x=mc+l$, where m,l are positive and decreasing when $|a|$ increases (see Figure 5). We have used a least squares fit to adjust the values of m and l, for values of $|a|$ between 50 and 500, and we have obtained the following function:

$$f(a,c,x)=-x+\frac{A_1}{(-a)^{A_2}}c+\frac{A_3}{(-a)^{A_4}}, \tag{27}$$

where the constants A_1, A_2, A_3 and A_4 are:

$$A_1=2408.3405,\ A_2=1.5448166,\ A_3=5182.0407,\ A_4=1.0234031\,.$$

Given a, c and x, the continued fraction method should be used if $f(a,c,x)<0$. Several tests have been carried out to check if this fit is correct: we have computed random values of a, c and x in the intervals $(-500,-50)$, $(0,200)$ and $(0,1000)$ respectively, and we have checked again the timings of the continued fraction method and Maple whenever $f(a,c,x) < 0$. If the fit is correct, this test should be favourable to the continued fraction method. In all cases, the proportion of points in the regions considered where the continued fraction is actually faster than Maple is larger than 90%.

5 Building the Full Algorithm

The main ingredients of the algorithm for computing the real zeros of hypergeometric functions were discussed in [7]. The resulting algorithm for computing the real zeros of the Kummer function $M(a;c;x)$, in a given interval $I=(x_a,x_b)$ reads as follows:

- Consider the following functions:

$$\begin{aligned} z_1(x) &= 2\sqrt{(1-a)x}\,,\\ H_1(x) &= \sqrt{\left|\frac{(1-a)x}{(c-1)^2}\right|}\frac{M(a;c;x)}{M(a-1;c-1;x)}\,,\\ \eta_1(x) &= -\frac{2x+3-2c}{4\sqrt{(1-a)x}}\,, \end{aligned}$$

  and

$$\begin{aligned} z_2(x) &= \sqrt{(c-a)(1-a)}\log x\,,\\ H_2(x) &= \sqrt{\left|\frac{1-a}{c-a}\right|}\frac{M(a;c;x)}{M(a-1;c;x)}\,,\\ \eta_2(x) &= -\frac{2a+c+x}{2\sqrt{(c-a)(1-a)}}\,. \end{aligned}$$

- Let $x_{tran}=c-a$.

  If $x_b < x_{tran}$, consider $z(x)=z_1(x)$, $H(x)=H_1(x)$, $\eta(x)=\eta_1(x)$.
  If $x_b > x_{tran}$, let $I_1=(x_a,x_{tran})$, $I_2=(x_{tran},x_b)$. Then consider
  $z(x)=z_1(x)$, $H(x)=H_1(x)$, $\eta(x)=\eta_1(x)$ in I_1 and $z(x)=z_2(x)$, $H(x)=H_2(x)$, $\eta(x)=\eta_2(x)$ in I_2.

- Divide the interval I (or the corresponding subintervals I_1, I_2) in subintervals where $\eta(x)$ does not change sign.

- Apply the following routine (*SWEEP*) in those subintervals where $\eta(x)$ does not change sign, but replacing the variable x by $z(x)$ and $H(x)$ by $H(z)=H(x(z))$. The zeros are computed in the z variable. Let $[z_1,z_2]$ be an interval where η does not change sign:

```
SUBROUTINE SWEEP($j$,$z_1$,$z_2$,$\epsilon$,$i$,$z(i)$)
**Algorithm: forward and backward sweeps.**
**Input**: $j = -sign(\eta)$ (+1 forward sweep; -1 backward); $z_1$;$z_2$;$\epsilon \equiv$ relative precision
**Output**: $i$ (number of zeros);$z(1), ..., z(i)$: zeros in the interval
NOTERM=1
$\bar{z}_1 = \frac{z_1 + z_2}{2} + j(\frac{z_1 - z_2}{2})$
$z = \bar{z}_1$
$\bar{z}_2 = \frac{z_1 + z_2}{2} - j(\frac{z_1 - z_2}{2})$
IF ($jH(z) > 0$) THEN $z = z + j\pi/2$
$i = 0$
DO WHILE ($j(z - \bar{z}_2) < 0$)
      CALL FIXEDPOINT($z$,$\bar{z}_2$,$\epsilon$,$z_n$,NOTERM)
      IF(NOTERM=1) THEN
            $z(j\,i) = z_n$
            $i = i + 1$
            $z = z_n + j\pi/2$
   ELSE
            $z = \bar{z}_2 + j$
   ENDIF
END WHILE
END

SUBROUTINE FIXEDPOINT($z$,$\bar{z}_2$,$\epsilon$,$z_n$,NOTERM)
Err=$1 + \epsilon$
DO WHILE (NOTERM=1) AND (Err> $\epsilon$)
     $z_p = z$
     $z = z - \arctan(H(z))$
     Err= $|1 - z/z_p|$
     IF ($j(z - \bar{z}_2) > 0$) THEN NOTERM=0
END WHILE
$z_n = z$
END
```

and the zeros generated in a forward sweep are stored in the positive positions of the array $z(i)$ ($z(1)$, $z(2)$,...) while those generated in a backward sweep are stored in the negative positions ($z(-1)$, $z(-2)$,...).

∘ Invert the change of variable $z(x)$ to obtain the zeros in the original variable.

A relevant issue for the stability of the full algorithm, which was not analysed in previous references [1,2,4], is the condition of the change of variables associated to the recurrences used in the algorithm. We discuss this point briefly in the following section.

5.1 Condition of the Change of Variables

An important issue in the accuracy of the computed zeros is the condition of the change of variable $z = z(x)$ explained in section 1. Indeed, once the zeros are computed in the variable z by means of the fixed point iteration, it is necessary to undo the change of variable in order to obtain the zeros in the original variable x and evaluate $x = x(z)$ at those points. In this section we examine the two changes used for Kummer function.

As explained in [7], the changes of variable associated with the iterations $(1, 1)$ and $(1, 0)$ are, respectively:

$$z_1(x) = 2\sqrt{(1-a)x}, \qquad z_2(x) = \sqrt{(c-a)(1-a)}\log(x). \tag{28}$$

The second change maps the interval $(0, +\infty)$ onto $(-\infty, +\infty)$, and the first one onto itself. When transforming from z to x the relative condition number is:

$$\kappa_{rel} = \left| z\,\frac{\dot{x}(z)}{x(z)} \right| . \tag{29}$$

In the first case:

$$\kappa_{rel} = 2, \tag{30}$$

so the problem is well conditioned. In the second case:

$$\kappa_{rel} = \left| \frac{z}{\sqrt{(c-a)(1-a)}} \right| . \tag{31}$$

Hence the condition number grows linearly with the variable z. In terms of the variable x we have that $\kappa_{rel} = |\log(x)|$. Taking into account (Eq.13), when the parameter $-a$ is large the largest zero is $x \sim -4a$, and we can estimate $\kappa_{rel} \sim \log(-4a)$.

For instance, for Laguerre polynomials $L_n^{(\alpha)}(x)$ the condition number can be estimated as $\kappa_{rel} \sim \log(4n)$ when the degree of the polynomial is large, and the loss of significant digits is mild for the large zeros. Because the corresponding fixed point iteration is only used for $x > c - a = n + \alpha + 1 > 1$, there is no bad condition due to x close to zero and $\kappa_{rel} \sim \log(4n)$ in the worst case.

5.2 Timings

The algorithm has been coded in Maple. In Table 1 we show examples of typical CPU-times spent by the algorithm on a standard configuration PC (Intel Pentium M processor at 1.5GHz and RAM memory of 512MB) under Windows XP. We consider non polynomial and polynomial cases. In the polynomial case, we show the relative CPU-times in comparison with the computation of the zeros using the Maple functions **KummerM** and **fsolve**. The interval for computing the zeros has been fixed to $I = [0.001, 50]$.

Table 1. Typical CPU-times spent by the algorithm for computing the real zeros of the Kummer function $M(a;c;x)$. $^{(*)}$ The number of digits for the computation with the function **fsolve** has been set to 35 in order to obtain at least 14 digits correct for the largest zero. $^{(**)}$ The number of digits for the computation with the function **fsolve** has been set to 50 in order to obtain at least 14 digits correct for the largest zero.

a	c	N_{zeros}	CPU-time	$\mathrm{Rel}_{\mathrm{CPU}}$
-50.1	0.1	31	$1.6\,s$	
-100.1	0.1	44	$2.3\,s$	
-500.1	0.1	99	$8.6\,s$	
-50	0.1	31	$1.3\,s$	$13.6^{(*)}$
-100	0.1	44	$2.2\,s$	$55.7^{(**)}$

Acknowledgments

The authors thank the referee for comments which have helped in improving the final manuscript. A. Gil acknowledges financial support from *Ministerio de Educación y Ciencia (programa Ramón y Cajal)*. J. Segura acknowledges financial support from project BFM2003-06335-C03-02. The authors acknowledge financial support from *Ministerio de Educación y Ciencia* (Project MTM2004–01367).

References

1. J. Segura. *The zeros of special functions from a fixed point method.*, SIAM J. Numer. Anal. 40 (2002) 114-133.
2. A. Gil and J. Segura. *Computing zeros and turning points of linear homogeneous second order ODEs*, SIAM J. Numer. Anal. 41 (2003) 827-855.
3. M. Abramowitz, I. A. Stegun. *Handbook of Mathematical functions, with formulas, graphs and mathematical tables.* Dover, 1972.
4. A. Deaño, A. Gil, J. Segura. *New inequalities from classical Sturm theorems.* J. Approx. Theory 131 (2004), 208-230.
5. A. Deaño, J. Segura. *Transitory minimal solutions of hypergeometric recursions and pseudoconvergence of associated continued fractions.* Accepted for publication in Math. Comp.
6. E. Frank. *A new class of continued fraction expansions for the ratios of hypergeometric functions.* Transactions of the American Mathematical Society, Vol. 81, n. 2, (1956) 453-476.
7. A. Gil, W. Koepf, J. Segura. *Computing the real zeros of hypergeometric functions.* Numer. Algorithms 36 (2004) 113-134.
8. L. Lorentzen, H. Waadeland. *Continued fractions with applications.* North Holland, 1992.
9. I.J. Thompson, A. R. Barnett. *Coulomb and Bessel functions of complex arguments and order.* J. Comput. Phys. 64 (1986) 490-509.
10. N.M. Temme. *Special functions. An introduction to the classical functions of Mathematical Physics.* John Wiley and Sons. 1996.

Towards Reliable Software for the Evaluation of a Class of Special Functions

Annie Cuyt and Stefan Becuwe

Universiteit Antwerpen
Departement Wiskunde en Informatica
Middelheimlaan 1, BE-2020 Antwerpen, Belgium
{annie.cuyt, stefan.becuwe}@ua.ac.be

Abstract. Special functions are pervasive in all fields of science. The most well-known application areas are in physics, engineering, chemistry, computer science and statistics. Because of their importance, several books and a large collection of papers have been devoted to the numerical computation of these functions. But up to this date, even environments such as Maple, Mathematica, MATLAB and libraries such as IMSL, CERN and NAG offer no routines for the reliable evaluation of special functions. Here the notion reliable indicates that, together with the function evaluation a guaranteed upper bound on the total error or, equivalently, an enclosure for the exact result, is computed.

We point out how limit-periodic continued fraction representations of these functions can be helpful in this respect. The newly developed (and implemented) scalable precision technique is mainly based on the use of sharpened a priori truncation error and round-off error upper bounds for real continued fraction representations of special functions of a real variable. The implementation is reliable in the sense that it returns a sharp interval enclosure for the requested function evaluation, at the same cost as the evaluation.

1 Basic Continued Fraction Material

Let us consider a continued fraction representation of the form

$$f = \cfrac{a_1}{1 + \cfrac{a_2}{1 + \dots}} = \frac{a_1|}{|1} + \frac{a_2|}{|1} + \dots = \sum_{n=1}^{\infty} \frac{a_n|}{|1}, \quad a_n := a_n(x), \quad f := f(x). \quad (1)$$

Here a_n is called the n-th partial numerator. We use the notation f and $f(x)$ interchangeably. The latter is preferred when the dependence on x needs to be emphasized. We respectively denote by the N-th approximant $f_N(w_N)$ or $f_N(x; w_N)$, and N-th tail t_N or $t_N(x)$ of (1), the values

$$f_N(w_N) = f_N(x; w_N) = \sum_{n=1}^{N-1} \frac{a_n|}{|1} + \frac{a_N|}{|1 + w_N}, \quad (2)$$

$$t_N = t_N(x) = \sum_{n=N+1}^{\infty} \frac{a_n|}{|1}, \qquad t_0 = f. \tag{3}$$

We also need approximants of continued fraction tails and therefore introduce the notation $f_N^{(k)}(w_{N+k})$ or $f_N^{(k)}(x; w_{N+k})$ for

$$f_N^{(k)}(w_{N+k}) = f_N^{(k)}(x; w_{N+k}) = \sum_{n=k+1}^{k+N-1} \frac{a_n|}{|1} + \frac{a_{N+k}|}{|1 + w_{N+k}}.$$

Sometimes the notation $f^{(k)}$ is used for the tail t_k. A continued fraction is said to converge if $\lim_{N\to\infty} f_N(0)$ exists. Note that convergence to ∞ is allowed. In the present paper we assume the continued fractions (1) to converge. Moreover, we restrict ourselves to the case where some $w_N \neq 0$ can be chosen such that

$$\lim_{N\to\infty} f_N(w_N) = \lim_{N\to\infty} f_N(0).$$

The N-th approximant of a continued fraction can also be written as

$$f_N(w_N) = (s_1 \circ \ldots \circ s_N)(w_N), \qquad s_n(w) = \frac{a_n}{1+w}, \qquad n = N, \ldots, 1.$$

Using the linear fractional transformations s_n, one can define a sequence $\{V_n\}_{n\in\mathbb{N}}$ of value sets for f by:

$$s_n(V_n) = \frac{a_n}{1+V_n} \subseteq V_{n-1}, \qquad n \geq 1. \tag{4}$$

The importance of such a sequence of sets lies in the fact that these sets keep track of where certain values lie. For instance, if $w_N \in V_N$ then $f_N(w_N) \in V_0$ and $f_{N-k}^{(k)}(w_N) \in V_k$. Also $t_N \in \overline{V}_N$ and $f \in \overline{V}_0$. An equally important role is played by a sequence of convergence sets $\{E_n\}_{n\in\mathbb{N}}$, of which the elements guarantee convergence of the continued fraction (1) as long as each partial numerator a_n belongs to the respective set E_n:

$$\forall n \geq 1 : a_n \in E_n \Rightarrow \sum_{n=1}^{\infty} \frac{a_n|}{|1} \text{ converges.}$$

A sequence $\{V_n\}_{n\in\mathbb{N}}$ is called a sequence of value sets for a sequence $\{E_n\}_{n\in\mathbb{N}}$ of convergence sets if (4) holds for all $a_n \in E_n$. Value sets can also be defined for non-convergent continued fractions (then the E_n are called element sets), but in the current context this form of generality does not interest us.

It is well-known that the tail or rest term of a convergent Taylor series expansion converges to zero. It is less well-known that the tail of a convergent continued fraction representation does not need to converge to zero; it does not even need to converge at all. We give an example for each of the cases. Take for instance the continued fraction expansion

$$\frac{\sqrt{1+4x}-1}{2} = \sum_{n=1}^{\infty} \frac{x|}{|1}, \qquad x \geq -\frac{1}{4}.$$

Each tail t_N converges to $1/2(\sqrt{1+4x}-1)$ as well. More remarkable is that the even-numbered tails of the convergent continued fraction

$$\sqrt{2}-1 = \sum_{n=1}^{\infty}\left(\frac{(3+(-1)^n)/2}{1}\right) = \frac{1|}{|1} + \frac{2|}{|1} + \frac{1|}{|1} + \frac{2|}{|1} + \dots$$

converge to $\sqrt{2}-1$ while the odd-numbered tails converge to $\sqrt{2}$ (hence the sequence of tails does not converge), and that the sequence of tails $\{t_N\}_{N\geq 1} = \{N+1\}_{N\geq 1}$ of

$$1 = \sum_{n=1}^{\infty}\frac{n(n+2)|}{|1}$$

converges to $+\infty$. Very accurate approximants $f_N(w_N)$ for f can be computed by making an appropriate choice for the tail estimate $w_N \approx t_N$.

We call a continued fraction of the form (1) limit-periodic with period k, if

$$\lim_{p\to\infty} a_{pk+q} = \tilde{a}_q, \qquad q = 1,\dots,k.$$

More can be said about tails of limit-periodic continued fractions with period one, also called limit-periodic continued fractions. Let (1) converge and be limit-periodic with $a_n \geq -1/4$ and $\lim_{n\to\infty} a_n = \tilde{a} < \infty$. If $\tilde{w}$ is the in modulus smaller fixpoint of the linear fractional transformation $s(w) = \tilde{a}/(1+w)$, then

$$\tilde{w} = -\frac{1}{2} + \sqrt{\tilde{a} + \frac{1}{4}} = \lim_{N\to\infty} t_N$$

and, according to [8],

$$\lim_{N\to\infty}\left|\frac{f(x) - f_N(x;\tilde{w})}{f(x) - f_N(x;0)}\right| = 0.$$

Hence a suitable choice of w in (2) may result in more rapid convergence of the approximants ($w = 0$ is usually used as a reference).

In this paper we further restrict the condition that (1) converges, in the case of limit-periodic continued fractions, to the condition $a_n \geq -1/4$ and $\{a_n\}_{n\in\mathbb{N}}$ bounded [7, pp. 150–159]. This condition automatically implies that $\tilde{a} \geq -1/4$ and $\tilde{w}$ is real.

2 Truncation Error

Most truncation error upper bounds for $|f(x) - f_N(x;w_N)|$ are given for the classical choice $w_N = 0$. For continued fractions with partial numerators of the form $a_n(x) = \alpha_n x$ with $\alpha_n > 0$ we refer among others to the a priori Gragg-Warner bound

$$|f(x) - f_N(x;0)| \leq 2\frac{|a_1|}{\cos\phi}\prod_{k=2}^{N}\frac{\sqrt{1+4|a_k|/\cos^2(\phi)}-1}{\sqrt{1+4|a_k|/\cos^2(\phi)}+1}, \quad -\pi < 2\phi = \arg(x) < \pi.$$

which holds for $N \geq 2$ and the a posteriori Henrici-Pfluger bound

$$|f(x) - f_N(x;0)| \leq \begin{cases} |f_N(x;0) - f_{N-1}(x;0)|, & |\arg(x)| \leq \pi/2, \\ \dfrac{|f_N(x;0) - f_{N-1}(x;0)|}{|\sin(\arg(x))|}, & \pi/2 < |\arg(x)| < \pi. \end{cases}$$

In [4] we prove a practical and sharp truncation error bound for the case $w_N \neq 0$, which is valid for all continued fractions with real partial numerators $a_n(x)$. This result departs from the Oval Sequence Theorem [7, pp. 145–147], which holds in the complex plane, from which a priori truncation error estimates can be obtained in case $w_N \neq 0$. In the real case the involved value sets V_n and convergence sets E_n are intervals and the theorem can be simplified and sharpened to the real Interval Sequence Theorem [4], here Theorem 1.

Theorem 1. *Let for all n the values L_n and R_n satisfy $-1/2 \leq L_n \leq R_n < \infty$ and let*

$$b_n := (1 + \operatorname{sign}(L_n) \max(|L_n|, |R_n|))\, L_{n-1},$$
$$c_n := (1 + \operatorname{sign}(L_n) \min(|L_n|, |R_n|))\, R_{n-1},$$

satisfy $b_n \leq c_n$ and $0 \leq b_n c_n$. Then the sequence $\{V_n\}_{n\in\mathbb{N}}$ with $V_n = [L_n, R_n]$ is a sequence of value sets for the sequence $\{E_n\}_{n\in\mathbb{N}}$ of convergence sets given by

$$E_n = [b_n, c_n] = \begin{cases} [(1+R_n)L_{n-1}, (1+L_n)R_{n-1}], & b_n \geq 0, \\ [(1+L_n)L_{n-1}, (1+R_n)R_{n-1}], & b_n \leq 0. \end{cases}$$

For $w_N \in V_N$ the relative truncation error $|f(x) - f_N(x;w_N)|/|f(x)|$ is bounded by

$$\left| \frac{f(x) - f_N(x;w_N)}{f(x)} \right| \leq \frac{R_N - L_N}{1 + L_N} \prod_{k=1}^{N-1} M_k \tag{5}$$

where $M_k = \max\{|u/(1+u)| : u \in V_k\} = \max\{|L_k/(1+L_k)|, |R_k/(1+R_k)|\}$.

In Theorem 1 the sets E_n are deduced from the intervals $V_n = [L_n, R_n]$ and the bounds of E_n are formulated in terms of L_n and R_n. In the following Lemma 1 [4] we formulate L_n and R_n in terms of the bounds on a_n in E_n and associate intervals V_n with given intervals E_n, instead of the other way around. Let $E_n = [b_n, c_n]$ with $-1/4 \leq b_n \leq c_n$ and $b_n c_n \geq 0$. The condition that b_n and c_n have the same sign means nothing more than that at least $\operatorname{sign}(a_n)$ is kept fixed in E_n.

Lemma 1. *If the sequence of convergence sets $\{E_n\}_{n\in\mathbb{N}}$ is given by $E_n = [b_n, c_n]$ with $b_n \geq -1/4$ and $0 \leq b_n c_n$, then the corresponding sequence of value sets $\{V_n\}_{n\in\mathbb{N}}$ is given by $V_n = [L_n, R_n]$ where L_n and R_n are particular tails of the continued fractions*

$$\hat{D} = \frac{b_1|}{|1} + \frac{c_2|}{|1} + \frac{b_3|}{|1} + \frac{c_4|}{|1} + \ldots,$$
$$\hat{U} = \frac{c_1|}{|1} + \frac{b_2|}{|1} + \frac{c_3|}{|1} + \frac{b_4|}{|1} + \ldots,$$

and

$$\check{D} = \frac{b_1|}{|1} + \frac{b_2|}{|1} + \frac{b_3|}{|1} + \frac{b_4|}{|1} + \ldots,$$
$$\check{U} = \frac{c_1|}{|1} + \frac{c_2|}{|1} + \frac{c_3|}{|1} + \frac{c_4|}{|1} + \ldots,$$

More precisely, denoting the tails of $\hat{D}, \check{D}$ and $\hat{U}, \check{U}$ respectively by $\hat{D}^{(n)}, \check{D}^{(n)}$ and $\hat{U}^{(n)}, \check{U}^{(n)}$ we have when all $b_n \geq 0$:

$$\begin{aligned} L_{2j} &= \hat{D}^{(2j)}, \qquad & L_{2j-1} &= \hat{U}^{(2j-1)}, \\ R_{2j} &= \hat{U}^{(2j)}, \qquad & R_{2j-1} &= \hat{D}^{(2j-1)}, \end{aligned} \tag{6}$$

and when all $b_n \leq 0$:

$$L_n = \check{D}^{(n)}, \qquad R_n = \check{U}^{(n)}. \tag{7}$$

3 Round-Off Error

Several algorithms exist for the computation of $f_N(w)$, the most stable [5] being the backward recurrence algorithm

$$\begin{aligned} F_{N+1}^{(N)} &= w_N \\ F_n^{(N)} &= \frac{a_n}{1 + F_{n+1}^{(N)}}, \qquad n = N, N-1, \ldots, 1 \\ f_N(w) &= F_1^{(N)} \end{aligned}$$

For the backward recurrence algorithm to be useful in a scalable precision context, it must be possible to determine N rather easily a priori, in other words which approximant to compute.

When actually implementing $f_N(w_N)$, we need to take into account that each basic operation $* \in \{+, -, \times, \div\}$ is being replaced by an (IEEE compliant) floating-point operation $\circledast \in \{\oplus, \ominus, \otimes, \oslash\}$. Such a floating-point implementation is characterized by four parameters, being the base β used for the internal number representation, the precision or amount p of β-digits, and the exponent range $[e_{\min}, e_{\max}]$ allowed in the floating-point notation. Usually the rounding mode in use is round-to-nearest (with the proper tie break). Each basic floating-point operation $x \circledast y$ is then subject to a relative error of at most $1/2$ `ulp` [3] where one `ulp` or unit-in-the-last-place equals β^{-p+1}. Also each partial numerator a_n needs to be converted to a floating-point number $\breve{a}_n$, hence entailing a relative rounding error ϵ_n given by

$$\breve{a}_n = a_n(1 + \epsilon_n).$$

Here $|\epsilon_n|$ is usually no more than a few `ulp`. Without loss of generality, we assume that $w_N \in V_N$ is a floating-point number estimating t_N. When executing the backward recurrence, each computed $\breve{F}_n^{(N)}$ then differs from the true $F_n^{(N)}$ by a rounding error $\epsilon_n^{(N)}$, and this for $n = N, \dots, 1$, in other words

$$\begin{aligned}
\breve{F}_{N+1}^{(N)} &= w_N, \qquad \epsilon_{N+1}^{(N)} = 0, \\
\breve{F}_n^{(N)} &= \breve{a}_n \oslash \left(1 \oplus \breve{F}_{n+1}^{(N)}\right), \qquad n = N, \dots, 1 \\
&= \frac{\breve{a}_n}{1 + \breve{F}_{n+1}^{(N)}}(1 + \delta_n) \\
&= F_n^{(N)}(1 + \epsilon_n^{(N)}) \\
\breve{F}_1^{(N)} &= F_1^{(N)}(1 + \epsilon_1^{(N)})
\end{aligned}$$

Here δ_n is the relative rounding error introduced in step n of the algorithm. The question how large $|\epsilon_1^{(N)}|$ is, is answered in Lemma 2 [6] and Theorem 2, the latter being a slight generalization of a result proved in [6]. Let us introduce the notation

$$\gamma_n^{(N)} = F_{n+1}^{(N)}/(1 + F_{n+1}^{(N)}), \qquad n = 1, \dots, N.$$

Lemma 2. *Let $\{V_n\}_{n=1}^{\infty}$ be a sequence of value sets for (1). If $F_{N+1}^{(N)} = w_N \in V_N$, then for $1 \le n \le N$,*

$$|\gamma_n^{(N)}| = \left| \frac{F_{n+1}^{(N)}}{1 + F_{n+1}^{(N)}} \right| \le M = \max_{n=1,\dots,N} M_n.$$

Theorem 2. *Let $F_{N+1}^{(N)} = w_N$ be a floating-point number and let for $n = 1, \dots, N$,*

$$\begin{aligned}
|\epsilon_n| &\le \epsilon \; \texttt{ulp}, \\
|\delta_n| &\le \delta \; \texttt{ulp}, \\
|\gamma_n^{(N)}| &\le M.
\end{aligned}$$

Let the base β and precision p of the IEEE arithmetic in use satisfy

$$\left(1 + M(1 + 2\epsilon + 2\delta)\frac{M^{N-1} - 1}{M - 1}\right) \texttt{ulp} < 1.$$

Then $|\epsilon_1^{(N)}|$ is bounded by

$$|\epsilon_1^{(N)}| \le \frac{1}{2}(1 + 2\epsilon + 2\delta)\frac{M^N - 1}{M - 1} \; \texttt{ulp}.$$

From Theorem 2 we obtain for the relative round-off error:

$$\frac{|f_N(x; w_N) - \breve{F}_1^{(N)}|}{|f(x)|} = \left|\epsilon_1^{(N)}\right| \frac{|F_1^{(N)}|}{|f(x)|} \le \frac{1 + 2\epsilon + 2\delta}{2} \; \frac{M^N - 1}{M - 1} \, \frac{|F_1^{(N)}|}{|f(x)|} \beta^{-p+1}. \quad (8)$$

4 Towards a Reliable Implementation

Let us denote the right hand side of (5) by ϵ_T and the right hand side of (8) by ϵ_R. Clearly

$$\epsilon_T = \epsilon_T(N, b_1, \ldots, b_N, c_1, \ldots, c_N)$$

and

$$\epsilon_R = \epsilon_R(N, \beta, p, M_1, \ldots, M_N).$$

In order to guarantee that $\breve{F}_1^{(N)}$ has s significant β-digits, meaning that

$$\frac{\left|f - \breve{F}_1^{(N)}\right|}{|f|} \leq \epsilon_T + \epsilon_R \leq \sigma = \frac{\beta}{2}\beta^{-s}$$

we proceed as follows:

– we determine N (and $w_N \in V_N$) from the condition

$$\epsilon_T(N, b_1, \ldots, b_N, c_1, \ldots, c_N) \leq \tau < \sigma \tag{9}$$

– we determine a suitable precision p (for chosen β) from the condition

$$\epsilon_R(N, \beta, p, M_1, \ldots, M_N) \leq \rho < \sigma \tag{10}$$

with $\tau \geq 0, \rho \geq 0, \tau + \rho = \sigma$. The former condition directly involves the inaccuracy $|c_n - b_n|$ that we allow for the partial numerators a_n. The latter condition depends on the sequence of values M_n, hence on the L_n and R_n which can be obtained from the b_n and c_n.

Obtaining a useful value w_N is the remaining crucial step. To this end we need to establish a few new results. We further distinguish between

– limit-periodic continued fractions where $a_n \to \tilde{a}$ from one side, say $\{a_n\}_{n\in\mathbb{N}}$ is a decreasing (or increasing) sequence with $\lim_{n\to\infty} a_n = \tilde{a}$,
– and limit-periodic fractions where $a_n \to \tilde{a}$ in an alternating fashion, say the sequences $\{a_{2n+1}\}_{n\in\mathbb{N}}$ and $\{a_{2n}\}_{n\in\mathbb{N}}$ respectively decrease and increase towards their mutual limit $\tilde{a}$.

Let us denote the j-th approximants of R_k and L_k as given by (6) and (7) in Lemma 1, by $R_{k,j}(\omega_j)$ and $L_{k,j}(\omega_j)$ respectively. For the tail estimates in (6) and (7) we switch to the notation ω_j instead of the traditional w_j used in (2) in order to avoid confusion between the different tails. Detailed proofs of the new results will be given in future work [2]. For the time being we focus on the role of these results in a procedure for the reliable evaluation of special functions that allow a limit-periodic continued fraction representation (in a certain region of the real variable x).

For the accurate computation of a suitable N from (9) we need to know $|R_k - L_k|$ for $k = 1, \ldots, N$, in other words an upper bound for R_k and a lower bound for L_k. In order to obtain a suitable w_N, meaning a value $w_N \in V_N$, we need to know the interior of $[L_N, R_N]$ or an upper bound for L_N and a lower bound for R_N. Both can be realized by computing enclosures for the values L_k and R_k. These upper and lower bounds for L_k and R_k are given in the Lemmas 3, 4 and 5. Some additional care needs to be taken, but for the moment we restrict ourselves to the headlines of the technique. More details will be given in [2].

4.1 Case a_n Positive

When (1) has positive partial numerators a_n, then the values M_k in Theorem 1 equal

$$M_k = \frac{R_k}{1+R_k}, \qquad k = 1, \ldots, N-1.$$

In Lemma 3 we explicit the bounds on L_k and R_k in case the partial numerators a_n show an oscillatory behaviour towards the limit $\tilde{a}$. In Lemma 4 we treat the case where the a_n decrease monotonically to $\tilde{a}$.

Lemma 3. *Let the sequences $\{a_{2n+1}\}_{n\in\mathbb{N}}, \{b_{2n+1}\}_{n\in\mathbb{N}}, \{c_{2n+1}\}_{n\in\mathbb{N}}$ and the sequences $\{a_{2n}\}_{n\in\mathbb{N}}, \{b_{2n}\}_{n\in\mathbb{N}}, \{c_{2n}\}_{n\in\mathbb{N}}$ respectively decrease and increase to their mutual limit $\tilde{a}$. With*

$$\begin{aligned}
2\omega &= -1 + \sqrt{4\tilde{a}+1}, \\
2\omega^{(\ell)}_{k,2j-1} &= c_{k+2j} - b_{k+2j+1} - 1 + \sqrt{4c_{k+2j} + (c_{k+2j} - b_{k+2j+1} - 1)^2}, \\
2\omega^{(\ell)}_{k,2j} &= b_{k+2j+1} - c_{k+2j+2} - 1 + \sqrt{4b_{k+2j+1} + (b_{k+2j+1} - c_{k+2j+2} - 1)^2}, \\
2\omega^{(r)}_{k,2j} &= c_{k+2j+1} - b_{k+2j+2} - 1 + \sqrt{4c_{k+2j+1} + (c_{k+2j+1} - b_{k+2j+2} - 1)^2}, \\
2\omega^{(r)}_{k,2j-1} &= b_{k+2j} - c_{k+2j+1} - 1 + \sqrt{4b_{k+2j} + (b_{k+2j} - c_{k+2j+1} - 1)^2},
\end{aligned}$$

the following bounds can be given for L_k and R_k where $\ell \geq 1$ and $j \geq 0$:

$$\begin{aligned}
L_{2\ell-1,2j}(\omega^{(\ell)}_{2\ell-1,2j}) \leq L_{2\ell-1} &\leq L_{2\ell-1,2j}(\omega), \\
L_{2\ell-1,2j+1}(\omega^{(\ell)}_{2\ell-1,2j+1}) \leq L_{2\ell-1} &\leq L_{2\ell-1,2j+1}(\omega), \\
L_{2\ell,2j}(\omega) \leq L_{2\ell} &\leq L_{2\ell,2j}(\omega^{(\ell)}_{2\ell,2j}), \\
L_{2\ell,2j+1}(\omega) \leq L_{2\ell} &\leq L_{2\ell,2j+1}(\omega^{(\ell)}_{2\ell,2j+1}),
\end{aligned}$$

and

$$\begin{aligned}
R_{2\ell-1,2j}(\omega^{(r)}_{2\ell-1,2j}) \leq R_{2\ell-1} &\leq R_{2\ell-1,2j}(\omega), \\
R_{2\ell-1,2j+1}(\omega^{(r)}_{2\ell-1,2j+1}) \leq R_{2\ell-1} &\leq R_{2\ell-1,2j+1}(\omega), \\
R_{2\ell,2j}(\omega) \leq R_{2\ell} &\leq R_{2\ell,2j}(\omega^{(r)}_{2\ell,2j}), \\
R_{2\ell,2j+1}(\omega) \leq R_{2\ell} &\leq R_{2\ell,2j+1}(\omega^{(r)}_{2\ell,2j+1}).
\end{aligned}$$

Lemma 4. *Let the sequences* $\{a_n\}_{n\in\mathbb{N}}, \{b_n\}_{n\in\mathbb{N}}, \{c_n\}_{n\in\mathbb{N}}$ *all decrease to* $\tilde{a} \geq 0$. *With*

$$\begin{aligned}
2\omega_{k,j}^{(01)} &= \tilde{a} - c_{k+j+2} - 1 + \sqrt{4\tilde{a} + (\tilde{a} - c_{k+j+2} - 1)^2},\\
2\omega_{k,j}^{(10)} &= b_{k+j+1} - \tilde{a} - 1 + \sqrt{4b_{k+j+1} + (b_{k+j+1} - \tilde{a} - 1)^2},\\
2\omega_{k,j}^{(20)} &= c_{k+j+1} - \tilde{a} - 1 + \sqrt{4c_{k+j+1} + (c_{k+j+1} - \tilde{a} - 1)^2},\\
2\omega_{k,j}^{(02)} &= \tilde{a} - b_{k+j+2} - 1 + \sqrt{4\tilde{a} + (\tilde{a} - b_{k+j+2} - 1)^2},
\end{aligned}$$

the following bounds can be given for L_k *and* R_k *where* $k \geq 1$ *and* $j \geq 0$:

$$\begin{aligned}
L_{k,2j}(\omega_{k,2j}^{(02)}) \leq L_k \leq L_{k,2j}(\omega_{k,2j}^{(10)}),\\
L_{k,2j+1}(\omega_{k,2j+1}^{(20)}) \leq L_k \leq L_{k,2j+1}(\omega_{k,2j+1}^{(01)}),
\end{aligned}$$

and

$$\begin{aligned}
R_{k,2j}(\omega_{k,2j}^{(01)}) \leq R_k \leq R_{k,2j}(\omega_{k,2j}^{(20)}),\\
R_{k,2j+1}(\omega_{k,2j+1}^{(10)}) \leq R_k \leq R_{k,2j+1}(\omega_{k,2j+1}^{(02)}),
\end{aligned}$$

4.2 Case a_n Negative

When (1) has negative partial numerators a_n, then the values M_k in Theorem 1 equal

$$M_k = \frac{|L_k|}{1 + L_k}, \qquad k = 1, \ldots, N-1.$$

In Lemma 5 we explicit the bounds on L_k and R_k in case the partial numerators a_n form a monotonic sequence towards the limit $\tilde{a}$, either decreasing or increasing.

Lemma 5. *Let* $k \geq 1, j \geq 0$ *and*

$$\begin{aligned}
2\omega &= -1 + \sqrt{4\tilde{a} + 1},\\
2\omega_{k,j}^{(\ell)} &= -1 + \sqrt{4b_{k+j+1} + 1},\\
2\omega_{k,j}^{(r)} &= -1 + \sqrt{4c_{k+j+1} + 1}.
\end{aligned}$$

If the sequences $\{a_n\}_{n\in\mathbb{N}}, \{b_n\}_{n\in\mathbb{N}}, \{c_n\}_{n\in\mathbb{N}}$ *are decreasing with* $\lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n = \lim_{n\to\infty} c_n = \tilde{a}$, *then*

$$\begin{aligned}
L_{k,j}(\omega) \leq L_k \leq L_{k,j}(\omega_{k,j}^{(\ell)}),\\
R_{k,j}(\omega) \leq R_k \leq R_{k,j}(\omega_{k,j}^{(r)}).
\end{aligned}$$

If the sequences $\{a_n\}_{n\in\mathbb{N}}, \{b_n\}_{n\in\mathbb{N}}, \{c_n\}_{n\in\mathbb{N}}$ *are increasing with* $\lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n = \lim_{n\to\infty} c_n = \tilde{a}$, *then*

$$\begin{aligned}
L_{k,j}(\omega_{k,j}^{(\ell)}) \leq L_k \leq L_{k,j}(\omega),\\
R_{k,j}(\omega_{k,j}^{(r)}) \leq R_k \leq R_{k,j}(\omega).
\end{aligned}$$

4.3 Mixed Case

The condition that (1) has either only positive or only negative partial numerators a_n can be relaxed, as long as it is satisfied from a certain n on. If the number of terms with mixed behaviour is small, we can proceed as in [4]. If it is larger, then an alternative technique based on a combination of a small number of predictions and corrections, can be used [1]. The latter uses the same estimates as given in the Lemmas 3, 4 and 5.

5 Numerical Illustration

The collection of functions that can be evaluated reliably using this technique is impressive. It essentially includes all functions that have a known limit-periodic continued fraction representation. If the behaviour of the partial numerators a_n (increasing, decreasing, oscillating) is known then the current technique can be applied. If the behaviour varies as n grows, like in some hypergeometric functions, the related technique explained in [1] can be applied.

Without the ambition of being exhaustive, we are currently working at implementations for:

– the (lower and upper) incomplete gamma functions $\gamma(a,x)$ and $\Gamma(a,x)$,
– the error and complementary error function, Dawson's integral, the exponential integrals and several probability distributions that can be expressed in terms of these functions,
– the hypergeometric and confluent hypergeometric functions ${}_2F_1(a,1;c;x)$ and ${}_1F_1(1;b;x)$, and several ratios of hypergeometric and confluent hypergeometric functions,
– particular ratios of Bessel, spherical Bessel, modified Bessel, modified spherical Bessel, Whittaker and parabolic cylinder functions.

Here we give two numerical examples, one where the continued fraction representation (1) has positive a_n and one where the partial numerators a_n are negative.

5.1 Positive a_n

We consider

$$f(a,x) = \frac{a\gamma(a,x)e^x}{x^a} = \frac{\frac{a}{a-x}}{1} + \sum_{n=2}^{\infty} \left\lfloor \frac{\frac{(n-1)x}{(a+n-1-x)(a+n-2-x)}}{1} \right\rceil \tag{11}$$

where $\gamma(a,x)$ is the (lower) incomplete gamma function. The sequence $\{a_n\}_{n\in\mathbb{N}}$ is decreasing with $\tilde{a}=0$. Then L_k and R_k simplify to $L_k = 0$ and $R_k = a_{k+1}$ for $[b_n, c_n] = [0, a_n]$. We take $x = 1$ and $a = 9/2$ and require $f(a,x)$ to be evaluated with

$$\epsilon_T + \epsilon_R \leq 10^{-d+1}, \qquad d = 73, 74, \ldots, 80.$$

where τ and ρ in (9) and (10) are both taken equal to 5×10^{-d}. The results can be found in Table 1. Let us zoom in on the first line of output. For $d = 73$, the bound ϵ_T given by (5) is less than 2.0×10^{-73} if $N \geq 49$. Subsequently we choose our working precision p in (8) so as to keep ϵ_R below 5.0×10^{-73}. Here we take $\beta = 10$ because we are going to compare our evaluation with that given by the multiprecision implementation of Maple. From Theorem 1 we learn that all w_N satisfying

$$L_N = 0 \leq w_N \leq 1.812 \times 10^{-2} < R_N = a_{N+1}$$

are valid choices as a tail estimate, the easiest being $w_N = 0$.

In Table 2 we have set `Digits` in Maple to d and printed the result for the evaluation of $f(a, x)$ delivered by this computer algebra system. Clearly the evaluation in Maple is subject to a much larger error (2 or 3 trailing decimal digits are inaccurate in this case).

Table 1. Continued fraction library output

73	1.214009591773512617777498734645198390079596056622283491877162409691879700
74	1.2140095917735126177774987346451983900795960566222834918771624096918797000
75	1.21400959177351261777749873464519839007959605662228349187716240969187969998
76	1.214009591773512617777498734645198390079596056622283491877162409691879699983
77	1.2140095917735126177774987346451983900795960566222834918771624096918796999829
78	1.21400959177351261777749873464519839007959605662228349187716240969187969998292
79	1.214009591773512617777498734645198390079596056622283491877162409691879699982919
80	1.2140095917735126177774987346451983900795960566222834918771624096918796999829190

Table 2. Maple output

73	1.214009591773512617777498734645198390079596056622283491877162409691879774
74	1.2140095917735126177774987346451983900795960566222834918771624096918797015
75	1.21400959177351261777749873464519839007959605662228349187716240969187969966
76	1.214009591773512617777498734645198390079596056622283491877162409691879700001
77	1.2140095917735126177774987346451983900795960566222834918771624096918796999764
78	1.21400959177351261777749873464519839007959605662228349187716240969187969998223
79	1.214009591773512617777498734645198390079596056622283491877162409691879699982930
80	1.2140095917735126177774987346451983900795960566222834918771624096918796999829239

5.2 Negative a_n

Let us consider the function

$$f(x) = \frac{\exp(-x^2)}{2\sqrt{\pi}x(2x^2+1)\mathrm{erfc}(x)} - 1 = \sum_{n=1}^{\infty} \frac{\left.\frac{-(2n+1)(2n+2)}{(2x^2+5+4n)(2x^2+1+4n)}\right|}{\left|\ 1 \right.}$$

and $x = 2$. The partial numerators are negative and decrease to $\tilde{a} = -1/4$. We target $\epsilon_T \leq 2^{-79} \approx 1.65 \times 10^{-24}$ and use exact arithmetic for a change (hence $\epsilon_R = 0$).

The bound (5) is less than 2^{-79} for $N \geq 59$. For $j = 12$ we obtain in addition that

$$t_{N+1} = L_N < L_{N,j}\left(\frac{-1+\sqrt{4a_{N+2}+1}}{2}\right) < R_{N,j}(-1/2) < R_N = t_N$$

and hence that all w_N satisfying

$$L_{N,j}\left(\frac{-1+\sqrt{4a_{N+2}+1}}{2}\right) < -0.37621 \leq w_N \leq -0.37527 < R_{N,j}(-1/2)$$

are valid choices for the approximation of $f(x)$ by $f_N(x; w_N)$, since they belong to V_N guaranteed.

References

1. Colman, M., Cuyt, A.: Gauss and confluent hypergeometric functions accurate to the last digit. ACM TOMS (2006) (in preparation).
2. Cuyt, A., Becuwe, S.: Reliable software for the evaluation of several special functions. ACM TOMS (2006) (in preparation).
3. Cuyt, A., Verdonk, B.: Computer arithmetic: basic theory. SIAM, Philadelphia (2006) (in preparation).
4. Cuyt, A., Verdonk, B., Waadeland, H.: Efficient and reliable multiprecision implementation of elementary and special functions. SIAM Journal on Scientific Computing (2006) (to appear).
5. Gautschi, W.: Computational aspects of three-term recurrence relations. SIAM Rev. **9** (1987) 24–82
6. Jones, W.B., Thron, W.J.: Numerical stability in evaluating continued fractions. Math. Comp. **28** (1974) 795–810
7. Lorentzen, L., Waadeland, H.: Continued fractions with applications. North-Holland Publishing Company, Amsterdam (1992)
8. Thron, W.J., Waadeland, H.: Accelerating convergence of limit periodic continued fractions $K(a_n/1)$. Numer. Math. **34** (1980) 155–170

Multimedia Prototype of a Bilingual Model Within Technology Based Learning Environment: An Implementation of a Mathematics Learning Framework

Zur'aini Dahlan[1], Noraimi Shafie[1], and Rozeha A. Rashid[2]

[1] Science and Technology College, Universiti Teknologi Malaysia, City Campus, Kuala Lumpur, Malaysia
[2] Department of Telematics and Optics, Faculty of Electrical Engineering, Universiti Teknologi Malaysia, Skudai, Johor Bharu, Malaysia
zuraini@citycampus.utm.my, noraimi@citycampus.utm.my, rozeha@utm.my

Abstract. In response to the current globalization in the educational arena, and to the new policy of change in the medium of instruction for teaching mathematics and science in English as implemented by the Malaysian Government in 2003, we introduce the e-learning Bilingual Model which has been designed and used at the university. The multimedia prototype of the model consists of text-based content for first-year mathematics subjects with exact forms available in both the English language and the native language. Content is identified to provide descriptions of core concepts dynamically using, audio, video and graphics, and also constructed to provide bilingual glossaries. The combination of Information Communication Technologies (ICT) such as Short Messaging Service (SMS), MOODLE e-learning, and Freeware Online-portals for Group-websites (FrOG) become the setting for an integrated framework for Technology Based Learning Environment, and work as instructional delivery tools in addition to the traditional method of teaching mathematics.

Keywords: bilingual model, multimedia prototype, and technology based learning environment.

1 Introduction

The government agenda in achieving excellence in education as announced in *Belanjawan 2003* (Budget 2003) where the policy to change the medium of instruction in teaching mathematics and science from Bahasa Melayu to English presents an important innovation affecting instructions in mathematics and science not only in school, but in institutions of higher-learning as well. As the implementation for changing the medium of instruction from Bahasa Melayu to English is taking place immediately at the university level, students who are exposed to learning mathematics in the Malay medium all this while will definitely face

© Springer-Verlag Berlin Heidelberg 2006

several language learning difficulties as indicated by several studies when two kinds of language conventions, that is the social language and the academic language, are now taking place in the classroom.

Researchers have found that students who are non-native English speakers attempt to read and write mathematics sentences the same way that they read and write standard narrative text. Usually they try to translate word-for word between a mathematical concept expressed in words and the concept expressed in symbols. However they do not realize that mathematical concept expressed in words often differs in its order from the way the concept is expressed in symbols. Dale and Cuevas (1992) offer as example the phrase *eight divided by two* which might be incorrectly translated to

$$8\overline{)2} \text{ rather than } 2\overline{)8}\,,$$

or the algebraic phrase, *the number a is five less than the number b*, which might be mistakenly restated as

$$a = 5 - b, \text{ when it should be } a = b - 5.$$

Other difficulties that students may encounter are in the understanding of the specialized vocabulary and discourse features in the language of mathematics, and also in interpreting the meaning of logical connectors in mathematics discourse (Jarret, 1999).

Also, researchers believed that English as a second language (ESL) is best taught in natural situations, with the second language used in meaningful contexts rather than in repetitious drill of grammar and vocabulary. One variant of ESL, known as “sheltered subject-matter instruction”, adapts lessons according to students’ level of English proficiency. This approach is common in bilingual education programs, in which lessons are coordinated in students’ native language (Crawford, 1998).

In order to decide on the most appropriate approach to learning, some circumstances surrounding the learning was taken into account. For example, the bilingual, text-based mathematics prototype has an organization of instruction in which concepts are structured in increasing order of complexity. The learner can be introduced to the main concepts of a course and then move on to more of a self directed study that is meaningful to them and their particular context.

The prototype is actually a complementary tool for students to acquire mathematical understanding; meant for the use of the first year Diploma of Engineering students in University Teknologi Malaysia, since they are required to take mathematics subjects such as Algebra, Calculus, Geometry and Trigonometry, as prerequisites to their Engineering subjects. When students have been previously exposed to learning all these subjects in the medium of the native language, the prototype becomes the transitional device for them to derive mathematical meaning, assist comprehension and understanding faster by using the English language. This enables them to experience more reading, increased comprehension, achieve higher literacy and academic development within the mathematics courses.

2 Designing the Bilingual Model

The construction of the prototype is based on several aspects including: the syllabus, the content experts' opinion, the result of the Questionnaires and the Learning Style Inventory, as well as the learning theories and instructional design theories. Needs analysis was carried out in order to identify students' learning characteristics, language preference, perceived helpfulness of e-learning, and preferred language of instruction as well. These will determine their preferences on the major features of the courseware (Dahlan, Mohd. Mahzir, 2004).

To ensure that the teaching and learning of mathematics in English will be implemented effectively, the learning system using multimedia tools and applications on the internet is developed to have features such as

- Presentation of mathematics problem and statement using simple and declarative sentence in English
- Having the Bahasa Melayu that is the native language version available for the same notes and example in English
- Video clips of selected lesson
- Narration for concept expressed in words and concept expressed in symbols
- Link of specialized vocabulary of mathematics in the notes with the corresponding social and mathematical meaning

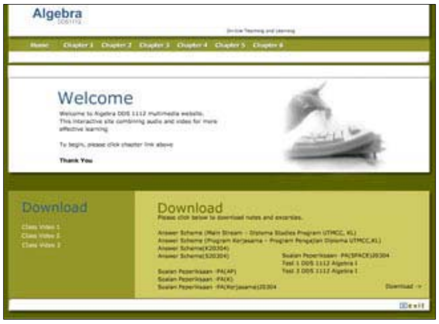
The web-based learning system fully uses Macromedia Flash and Authorware as the foundation for the construction of online notes which is composed in the English version and the Bahasa Melayu version. For the glossary, PHP programming language and MySQL database are two components used in the search engine in order to display user's search query. In addition, at the administrator's site, PHP is also used to insert text and graphics in the glossary database; and similar components are used for constructing and managing quizzes. These multimedia tools together with the Apache web server are then linked in an existing university MOODLE e-learning system.

3 Multimedia Prototype

The multimedia web page for the subject is divided into three main pages. The first page contains the introduction and links to the second and third page. Website view of the first page is shown in Figure 1.

The second page contains subtopic of the access from the first page. Access of notes in both languages is available (as shown in Fig. 2 and Fig. 3); the contents are structured in order to comply to the learning of any individual being exposed to the topic for the first time. Video clips of selected lessons taught in English and also Bahasa Melayu are added for further understanding of any topics.

In addition to the notes given, are example of questions together with the solutions and also collection of previous years examination questions. Furthermore, students are required to take interactive quizzes ranging from basic, intermediate to advance for building foundations of their mathematics skill and ability.



Welcome

Welcome to Algebra DDS 1112 multimedia website.
This interactive site combining audio and video for more effective learning.

To begin, please click chapter link above

Thank You

Download

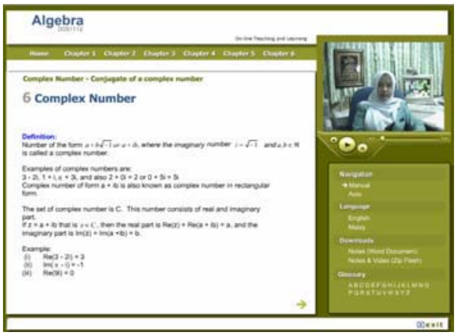
Please click below to download notes and exercises.

Answer Scheme (Main Stream - Diploma Studies Program UTMCC, KL)
Answer Scheme (Program Kerjasama - Program Pengajian Diploma UTMCC,KL)
Answer Scheme(K20304)
Answer Scheme(S20304)

Soalan Peperiksaan -PA(AP)
Soalan Peperiksaan -PA(K)
Soalan Peperiksaan -PA(Kerjasama)20304

Soalan Peperiksaan -PA(SPACE)20304
Test 1 DDS 1112 Algebra I
Test 2 DDS 1112 Algebra I

Fig. 1. Website view



Complex Number - Conjugate of a complex number

6 Complex Number

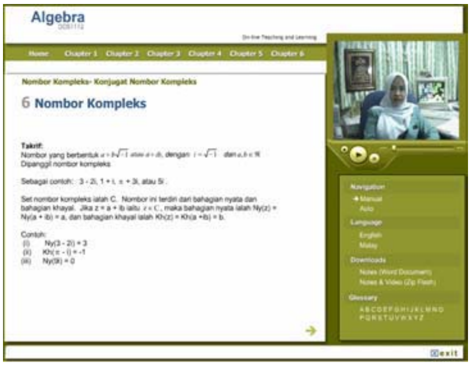
Definition:
Number of the form $a + b\sqrt{-1}$ or $a + ib$, where the imaginary number $i = \sqrt{-1}$ and $a, b \in \Re$ is called a complex number.

Examples of complex numbers are:
3 - 2i, 1 + i, π + 3i, and also 2 + 0i = 2 or 0 + 5i = 5i
Complex number of form a + ib is also known as complex number in rectangular form.

The set of complex number is C. This number consists of real and imaginary part.
If z = a + ib that is $z \in C$, then the real part is Re(z) = Re(a + ib) = a, and the imaginary part is Im(z) = Im(a +ib) = b.

Example:
(i) Re(3 - 2i) = 3
(ii) Im(π - i) = -1
(iii) Re(5i) = 0

Fig. 2. Interface of content in English



Nombor Kompleks- Konjugat Nombor Kompleks

6 Nombor Kompleks

Takrif:
Nombor yang berbentuk $a + b\sqrt{-1}$ atau $a + ib$, dengan $i = \sqrt{-1}$ dan $a, b \in \Re$
Dipanggil nombor kompleks

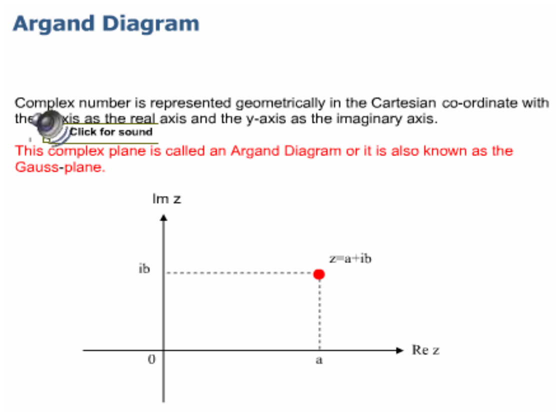
Sebagai contoh: 3 - 2i, 1 + i, π + 3i, atau 5i .

Set nombor kompleks ialah C. Nombor ini terdiri dari bahagian nyata dan bahagian khayal. Jika z = a + ib iaitu $z \in C$, maka bahagian nyata ialah Ny(z) = Ny(a + ib) = a, dan bahagian khayal ialah Kh(z) = Kh(a +ib) = b.

Contoh:
(i) Ny(3 - 2i) = 3
(ii) Kh(π - i) = -1
(iii) Ny(5i) = 0

Fig. 3. Interface of content in Bahasa Melayu

In Fig. 4, selected sentence is highlighted and added with the display of an audio icon to indicate that narration is provided. This is to help user to identify and to pronounce the words and mathematics symbols correctly. The demonstration on how to recite the mathematics sentence and symbols by narration is quite helpful for



Argand Diagram

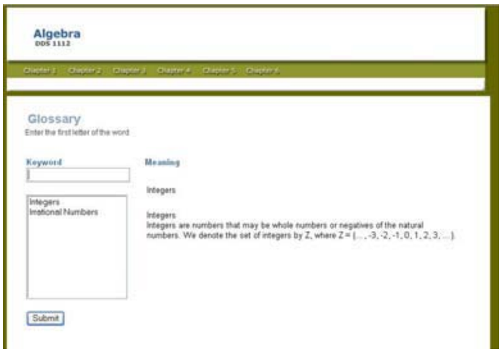
Complex number is represented geometrically in the Cartesian co-ordinate with the x-axis as the real axis and the y-axis as the imaginary axis.

This complex plane is called an Argand Diagram or it is also known as the Gauss-plane.

Fig. 4. Audio sample for selected sentence

learners to realize that mathematical concept when expressed in words often differs in its order from the way when the concept is expressed in symbols.

The third page as shown in Fig.5 contains the search for words or mathematical terms which is hypertext from the notes or accessed directly, from the database of the second page. The explanation of the mathematical term is given in both English and Bahasa Melayu. In obliging learners to understand the meaning of a word without any difficulty, short and simple descriptive phrase and presentation that is more acceptable to the general Malaysian scholar at the diploma level was prepared. The glossary section is most important in assimilating students with the use and the recognition of keywords. In reading sentence for mathematics subject in English, keyword recognition is one of the method that will help them to rapidly comprehend the context that is being discussed.



Algebra
DDS 1112

Glossary
Enter the first letter of the word

Keyword

Integers
Irrational Numbers

Meaning

Integers

Integers
Integers are numbers that may be whole numbers or negatives of the natural numbers. We denote the set of integers by Z, where Z = {..., -3, -2, -1, 0, 1, 2, 3, ...}

Fig. 5. Site for glossary search

4 Implementation

The Mathematics Learning Framework includes the Short Messaging Service (SMS), MOODLE e-learning and Freeware Online-portals for Group-websites (FrOG), are examples of Information Communication Technologies (ICT) used as instructional delivery tools within the technology based instructional learning, in addition to the traditional method of teaching mathematics. The multimedia prototype, as a part of the e-learning package, is used as a supplementary learning tool after each lecture, in helping students to review mathematics easily and systematically. It is also enhanced with the combination of active and cooperative learning among learners.

Now that various forms of computer/mobile-mediated communication is available, learners have access to large amount of current information, which is constantly being added with a high rate of new information. The state of having excess of information and unable to make a decision or remain informed about a topic is often referred to information overload or "technostress" (Wikipedia, 2006). In order to avoid users from being controlled by ICT rather than being empowered by it, they need to be guided, coached, motivated, and trained to use the system effectively.

Hussin and Dahlan in 2005 selected the SMS/FrOG as additional instructional delivery tools since it deals directly with both learner and instructor as individuals and it penetrates or breaks all barriers between learner and instructor. It defies conventional definition of task execution and allows learning to occur in a multitude of on-job situations, thus the need to monitor learners and to get feedback from learners, or to give encouragement and motivation becomes simple and more efficient. It even challenges and redefines the roles or "learner" and "instructor", by offering empowerment to anyone who masters the science (and art) of its application.

5 Finding and Discussions

All mature speakers of a language have knowledge about their language that is highly abstract, in other words they are able to distinguish between grammatical and ungrammatical sentences that differ only along this abstract dimension. This ability could not have been induced from simple exposure to the surface patterns of language but could have been acquired through a critical *a prori* knowledge about a language. For second language acquisition, the extension is that if learners successfully make similar distinctions, they must be able to follow their innate knowledge (Hakuta, McLaughlin, 1996).

From the linguistic perspectives, the rules of any grammar are highly abstract and do not reflect the surface properties of the language. Universal Grammar involves a set of principles with certain parameters which remain "open" until they are set by experience with the environment. Language acquisition is a process where the learner discovers how the principles operate in the target language and what parameter value apply. Furthermore, the grammar of a language is the set of values it assigns to various parameters (Hakuta, McLaughlin, 1996).

Given in the next paragraph is the sample of content for the chapter: Complex Number from Figure 2 and Figure 3. In this example, the statement in English is constructed using simple and declarative sentence which is actually almost an exact translation from the original statement in Bahasa Melayu. The reasoning behind this is when reading in English, the first year diploma students have the inclination towards translating word (in English) for word (into Bahasa Melayu), such that when they are reading mathematics reference text from an English publication the meaning of a sentence is lost in "their" translation.

Complex Number

Definition:

Number of the form $a + b\sqrt{-1}$ *or* $a + ib$*, with the imaginary number* $i = \sqrt{-1}$ *and* $a,b \in \Re$ *, is called a complex number.*

Examples of complex numbers are:
$3 - 2i$*,* $1 + i$*,* $\pi + 3i$*, and also* $2 + 0i = 2$ *or* $0 + 5i = 5i$ *.*

Complex number of form $a + ib$ *is also known as complex number in rectangular form.*

The set of complex number is C. This number consists of real and imaginary part.
If $z = a + ib$ *that is* $z \in \mathrm{C}$*, then the real part is* $Re(z) = Re(a + ib) = a$*, and the imaginary part is* $Im(z) = Im(a + ib) = b$*.*

The translation given below addressed the issue on how the parameters that have been set in the first language need to be reset or readjusted for the second language. Natural translation which refers to the cognitive skills involves, is applied here in order to enhance the contextual meaning and the link between the comprehension and meaning of the translation. The corresponding content in Bahasa Melayu statement is:

Nombor Kompleks

Takrif:

Nombor dalam bentuk $a + b\sqrt{-1}$ *atau* $a + ib$*, dengan nombor khayal* $i = \sqrt{-1}$ *dan* $a,b \in \Re$ *dipanggil nombor kompleks.*

Contoh-contoh nombor kompleks ialah:
$3 - 2i$*,* $1 + i$*,* $\pi + 3i$ *,* $2 + 0i = 2$ *, atau* $0 + 5i = 5i$ *.*

Nombor kompleks dalam bentuk $a + ib$ *juga dikenali sebagai nombor kompleks dalam bentuk segiempat.*

Set nombor kompleks ialah C. Nombor ini terdiri dari bahagian nyata dan bahagian khayal.
Jika $z = a + ib$ *iaitu* $z \in C$*, maka bahagian nyata ialah* $Ny(z) = Ny(a + ib) = a$*, dan bahagian khayal ialah* $Kh(z) = Kh(a + ib) = b$*.*

In this section, where the introduction to the imaginary number and the set of complex number is given, the learners have prior knowledge on the imaginary number. It was obtained when they were being taught the formula in finding roots of the quadratic equation. Once the learners have understood the construct of the imaginary number, they are able to apply the knowledge to other area that they have learned such as in the algebraic operations and exponential rule pertaining to the complex numbers without any difficulty.

Malakoff and Hakuta (1991) stated that bilingual has linguistic experience that is spread over two languages that are used in alternation. Experience is encoded in either one of the two languages and can be expressed in both languages. Similarly information representation can be switched between the languages. Most learning that is carried out in the first language readily transfers to English not only in content areas like mathematics, science and social studies, but also in skills, in speaking, reading and writing.

For type of content which involves concept or abstraction, such as in the definition of limit in calculus, the explanation on the concept in both languages is very extensive. The learner found it difficult to comprehend the meaning or understand the concept which is given in either the English or the native language at the first introduction. In this case, visual explanation is more appropriate and efficient in illustrating the idea.

When more complex challenges are given in mathematics, the Malaysian learners usually switch to the more dominant language that they have in order to understand or meet the challenges. Peal and Lambert in 1962 introduced the concept of "balanced" bilingual for those with equal proficiency in both languages in order to distinguish "pseudo-bilingual" from the truly bilingual. In this case, the majority of learners who are pseudo-bilingual/multilingual usually select the native language Bahasa Melayu as their domain. Further studies in determining the correlation between the preference or selection of language and the item difficulty of the mathematics content is being done and will be discussed later.

Mathematical notations need to have exactly one to one stringent translation in both languages with respect to their meaning and order. As an example, in the Binomial equation:

$$(a+b)^n = \sum_{r=0}^{n} {}^nC_r a^{n-r} b^r \tag{1}$$

we have the symbol $\sum$ that denotes "summation" in English or *"hasiltambah"* in Bahasa Melayu, and also the notations ${}^nC_r$ that is read as "n choose r" in English or *"n pilih r"* in Bahasa Melayu.

When "reading" the formula in both languages according in the meaning or order of the mathematical notations, significant change does not appear since the translation of mathematical notations is exact and does not involve any grammatical or parametric differences discussed earlier. However, when the algebraic phrase is stated differently from the arrangement of the mathematical notations or when some discourse feature in mathematics is encountered, then mistakes in the interpretation and comprehension may occur.

It was also observed that learners are able to comprehend mentally the visual interpretation of the more complex mathematical notations or the symbolic language of mathematics, but most of them do not have the same ability and quickness at demonstrating their oral presentation or communication in either languages (both Bahasa Melayu and English). In other words, they are able to show their work on mathematics, but they have difficulties in explaining the solution of their work to others. This disadvantage is more obvious when learners are required to construct mathematical conjectures, develop and evaluate some mathematical arguments, and also to select various types of representations of a certain mathematical problem. Action that has been taken to resolve this disadvantage is to practice the method of active and cooperative learning to cultivate the mathematics communication skill and to enhance their understanding on mathematics discourse during class.

6 Conclusion and Future Works

The introduction of the mathematics learning framework with the combination of MOODLE e-learning, SMS technology and free online group concept enable the instructors to monitor learners' progress and performance better. It is hoped that this can react as a catalyst in the process of generating higher order thinking and cultivating multi-tasking skills among learners.

The existence of multilingual communities in Malaysia enrich the culture and lifestyle, nevertheless the selection of English as a medium of instruction in the education system conforms the people of every descendent in the country to a minority in the global world. Therefore, condition that is conducive to learning which promotes additive bilingualism, where the native language and the English language support both academically and emotionally by the community and society, is most important to ensure positive effect on cognitive development.

Mathematics is a universal symbolic language that stands alone in any medium of communication. Traditional studies in mathematics usually involve the course syllabus, which is content oriented, and exercise on the oral and written interpretation. The visual interpretation of mathematics, such as using images to stimulate thinking, need to be explored since instant messages can be conveyed in visual form. As the saying goes "a picture speaks a thousand words".

The shift into the new paradigm of teaching mathematics, due to the changing world especially in ICT contributes to the high demand of new and innovative ideas in mathematics education for the future. Hence, there is an urgent need to train students to use the available technology effectively, and for the instructor to come up with quicker and more sophisticated tools and method for teaching and learning. In consequence, that learners are able to gather, sort, select, and experience benefit from the vast amount of information and overflow of knowledge that exist in the virtual atmosphere.

Acknowledgments

1. Professor Dato'Dr Haji Kamel Ariffin Atan. Director at the Institute of Mathematical Research, Universiti Putra Malaysia. For his effervescence and delight towards all research in mathematics, and especially in giving me confidence on the importance of this research.
2. This research is supported by the Research Management Center, Universiti Teknologi Malaysia. VOT # 75046.

References

1. Crawford, J.W. (1998). Issues in U.S. Language Policy : Bilingual Eucation. Retrieved Dec 13, 2004 from http://ourworld.compuserve.com/homepages/ JWCRAWFORD/ biling.htm.
2. Cummins, J. (1999). Beyond Adversarial discourse: Searching for common ground in the education of bilingual students. In I.A. Heath & C. J. Serrano(Eds.), Annual editions: Teaching English as a second language (pp.204-224). Guildford, CT: Dushkin/McGraw-Hill. [Online] Available at http://www.iteachilearn.com/cummins
3. Dale, T.C., and Cuevas,G.J. (1992). Integrating mathematics and language learning. In P.A. Richard-Amato & M.A. Snow (Es.), *The multicultural classroom: Readings for content-area teachers* White Plains, NY: Longman.
4. Dahlan, Z., Mohd Mahzir, A., (2004). The Construction of an E-Learning Bilingual Model for an Engineering Mathematics Subject at Universiti Teknologi Malaysia City Campus, Conference on Engineering Education., K.L., Malaysia, 14-15 Dec 2004.
5. Dahlan, Z., Hussin, F. H., (2005). Inculcating Generic Skills Among Students at University Teknologi Malaysia City Campus Through Technology Based Osmosis Learning, Proceedings of the 2005 Regional Conference on Engineering Education., Johor, Malaysia, 12-13 Dec 2005.
6. Ertmer, P. A., Newby, T. J. (1993). Behaviorism, cognitivism, constructivism: Comparing critical features from an instructional design perspective. *Performance Improvement Quarterly,* 6 (4), 50-70.
7. Genetic epistemology (J.Piaget). [On-line]. Available: http://www.gwu.edu/~tip/ piaget. html
8. Hong, K.-S., Ridzuan, A. A., & Kuek, M.-K. (2003). Students' attitudes toward the use of the Internet for learning: A study at a university in Malaysia. Educational Technology & Society, 6(2), 45-49*, Available at http://ifets.ieee.org/periodical/6-2/5.html (ISSN 1436-4522)*
9. Hakuta, K., McLaughlin, B. (1996). Bilingualism and second language learning: Seven tensions that define the research. In Berliner & R. Calfree (eds.), Handbook of Educational Psychology (pp. 603-621). New York: Macmillan Publish Co.
10. Hakuta, K., Ferdman, B.M., Diaz, R.M. (1987). Bilingualism and cognitive development: three perspectives. In S. Rosenberg (ed.), Advances in Applied Psycholinguistics Volume II: Reading, Writing and Language Learning (pp 284-319). Cambridge: Cambridge University Press.
11. Hakuta, K., Snow, C.E.(1986). Summary of research in bilingual education. California School Boards Journal, 44(7), 2 4.

12. Hussin, F.H., Dahlan, Z.(2005). Alternative framework for on-job immersion action research: Case studies in Technology Based Osmosis Learning. Proceedings of the 2005 Regional Conference on Engineering Education, Johor, Malaysia, 12-13 Dec 2005.
13. Jarrett, D. (1999). Teaching mathematics and science to english-language learners. *Northwest Regional Educational Laboratory*.[On-line] Available at http://nwrel.org/ msec/.
14. Malakoff, M., Hakuta, K. (1991). Translation skill and metalinguistic awareness in bilinguals. In E. Bialystok (Ed.), Language processing by bilingual children (pp.141-166). Oxford: Oxford University Press.
15. Mergel, B.(1998). Instructional Design and Learning Theory . [On-line] Available at http://www.usask.ca/education/coursework/820 papers /mergel/brenda.htm
16. Peal, E., Lambert, W. (1962) The relation of bilingualism to intelligence. *Psychological Mongraphs*, 76(No.546).
17. Rogerson, A. (1986). The mathematics in society project: a new conception of mathematics. International Journal of Math Education, Science and Technology. Vol 17, No 5, pp 611-616.
18. Schwier, R. A. (1995). Issues in emerging interactive technologies. In G.J. Anglin (Ed.), *Instructional technology: Past, present and future.* (2nd ed., pp. 119-127)., Englewood, CO: Libraries Unlimited, Inc.
19. Wikipedia contributors (2006). Wikipedia, The Free Encyclopedia. Retrieved May 23, 2006 from http://en.wikipedia.org/wiki/Information_overload

Methods to Access and Retrieve Mathematical Content in ActiveMath

Paul Libbrecht and Erica Melis

German Research Center for Artificial Intelligence, Saarbrücken, Germany

Abstract. This article describes how mathematical content items and formulæ are processed, retrieved, and accessed in ActiveMath. Central to the retrieval and access is a search tool which allows for searching text, attributes, relations and formulæ, and presenting items. The search tool has been evaluated according to the standard measures of precision and recall as well as for usability. We report results of these evaluations.

1 Introduction

Increasingly, (mathematical) content is enriched with semantic information in order to make it interoperable and better accessible for men and machines. To employ the semantics, new retrieval techniques have to be developed. Since our learning environment, ActiveMath, works with semantically represented maths content, we developed new techniques to make the semantics of items and of mathematical formulæ accessible with common information retrieval (IR) technologies. The developed techniques convert formulæ to indices and allow sub-expressions to be matched (with wild cards) while relying on the semantic representation of OpenMath [BCC$^+$04]. Moreover, we contribute the implementation of a search tool that does not only retrieve maths content items but also provides access to related items, ranks them, and presents them in an advanced human-readable rendering.

1.1 ActiveMath

ActiveMath [MAB$^+$01] is an integrated learning environment on the Web. Its content uses an extension of the `OMDoc` language [Koh00], which itself is an extension for mathematical documents of the OpenMath [BCC$^+$04] mathematical objects encoding. `OMDoc`'s item granularity is that of definitions, examples, exercises etc., and it is the level which is mostly used for management, referencing, and search in ActiveMath. Each *content item* can contain text with links and semantically encoded formulæ and is annotated with mathematical and pedagogical attributes and relations [MAF$^+$03].

ActiveMath supports learners in many ways including:

– generation of courses/books adapted to the user's learning goals, scenarios and knowledge
– interactive exercises with mathematical input, evaluation, and feedback

© Springer-Verlag Berlin Heidelberg 2006

– various learning support tools
– an open learner model.

The presentation of courses in the form of books provides an intuitive navigation paradigm. This and the search facility are two methods to access the content items on an ActiveMath server, allowing a learner to see the content item, reference it in communication, and find it.

A new ActiveMath facility for multi-dimensional search is presented in this paper. It searches through large `OMDoc` content repositories for text, formulæ, and items' characteristics.

The paper first recapitulates the learning and authoring situations in which content items are presented, searched for. This is followed by a description of the search tool's components and evaluation results. Finally related research and future works are presented.

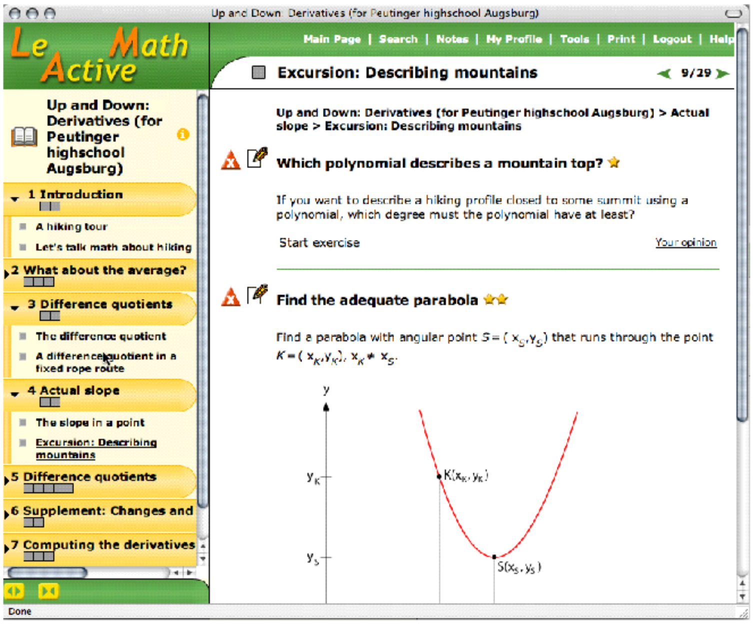
2 Access to Content Items

This section reviews common practices for mathematical knowledge management and indicates what ActiveMath is doing in this direction. We use the term *access* very broadly for the ability of a user to reach a given item, symbol, or formula. Accessing an item means to have it presented in a browser, to be able to reference it or to let other programmes download it. Access can be granted through linking or through search.

The search for mathematical texts or formulæ has the same purposes as general search and can be included into general search engines with additional components or via preprocessing. In ActiveMath, search can serve the system, e.g. adaptive course generation, or it can serve the learner to retrieve information for learning, to learn about the relationship among knowledge items, mathematical symbols, etc., to communicate search results, or to enquire about communicated information and mathematical expressions. Search can also serve authors and tutors which will search content with textual, formal, and attribute queries in many situations. For example, when authors are writing new content, are reviewing it, or are assembling new new courses from existing materials they search for items by their content, pedagogical attributes, or relations.

In ActiveMath, access to mathematical content may start when opening a "book" that contains items previously assembled by an author. An example page of a book is in Figure 1. While the learner is reading she may be wondering about a concept or symbol that is present on the page. Content items for this concept or symbol, e.g. a definition of it, may be directly linked by the author in which case the learner could simply click on it and the item would be displayed. The concept behind a mathematical symbol is presented via a link. The concept can also be searched by its name.

Situations in which a learner may wish to use links or search include the process of active learning exercising in which, e.g., a rule, definition, or theorem has to be recalled and applied. Such links can be offered in feedback of an exercise step. With her own initiative, the learner could use a search tool for


LeActiveMath
Main Page | Search | Notes | My Profile | Tools | Print | Logout | Help
9/29
Start exercise
Your opinion
Done

Up and Down Derivatives (for Peutinger highschool Augsburg)

Up and Down: Derivatives (for Peutinger highschool Augsburg)

1 Introduction
A hiking tour
Let's talk math about hiking
2 What about the average?
3 Difference quotients
The difference quotient
A difference quotient in a fixed rope route
4 Actual slope
The slope in a point
Excursion: Describing mountains
5 Difference quotients
6 Supplement: Changes and
7 Computing the derivatives

Excursion: Describing mountains

Up and Down: Derivatives (for Peutinger highschool Augsburg) > Actual slope > Excursion: Describing mountains

Which polynomial describes a mountain top?

If you want to describe a hiking profile closed to some summit using a polynomial, which degree must the polynomial have at least?

Find the adequate parabola

Find a parabola with anguler point $S=(x_S,y_S)$ that runs through the point $K=(x_K,y_K)$, $x_K \neq x_S$.

Fig. 1. A book presentation in ActiveMath

textual search or formulæ search in order to recall and/or copy semantics to a particular application, e.g. a concept mapping tool.

3 Ingredients of a Search Tool

In the sketch of the ingredients needed for a search tool we follow [You05] but our focus is on the mathematical content items and formulæ as opposed to the mere function orientation of [You05]. We present additional ingredients that are required for a system that does not only search but also presents the content.

Identifiers and References Content items need to be addressable in order to be extracted and referenced. References to items should be exchangeable in e-mail communications. Therefore, they need to be context independent and short.

Storage, Extraction, and Presentation Mathematical content items are embedded within larger documents. The documents need to be stored in repositories and ways for their extraction are needed, so that they can be queried and presented.

Query Input Queries for textual fragments are well known, being the focus of Information Retrieval. Queries for item attributes can be input using form-based interfaces or using a dedicated syntax. Queries for mathematical expressions is less developed. A usage of a facility to input the formulæ is important.

Indexing, Analysis, and Back-End Query Information Retrieval as in [vR79] provides powerful methods to convert rows of tokens to an index for which search results can be computed efficiently. This relies on a tokenization process called *analysis* which, among others, *stems* words (e.g. remove plural), removes too frequent words, etc.

We developed a special treatment for the highly structured nature of mathematical formulæ in the indexing process as well as in the analysis process transforming (queries for) mathematical expressions into (back-end queries for) rows of tokens.

Results Presentation The way a search tool displays a result is very important. For users who often only visit the first few matches, information retrieval has introduced the notion of relevance of a match, a score that is assigned to a document matching a query; results with highest relevance should be presented first. The results' list should provide visual hints about the type of mathematical item presented, its title, its ranking.

4 The ActiveMath Search Tool

The search tool of ActiveMath searches (an index produced from) the `OMDoc` sources. It provides a friendly user-interface that combines plain-text search with item-attributes and formulæ search. It presents the results of the queries as well as individual items and their relations.

A coarse architecture of the tool is depicted in Figure 2 which presents the flow of information at indexing time (on the left), at query time (right bottom), and at item presentation time (right top).

The description of the ActiveMath search tool follows the structure of §3.

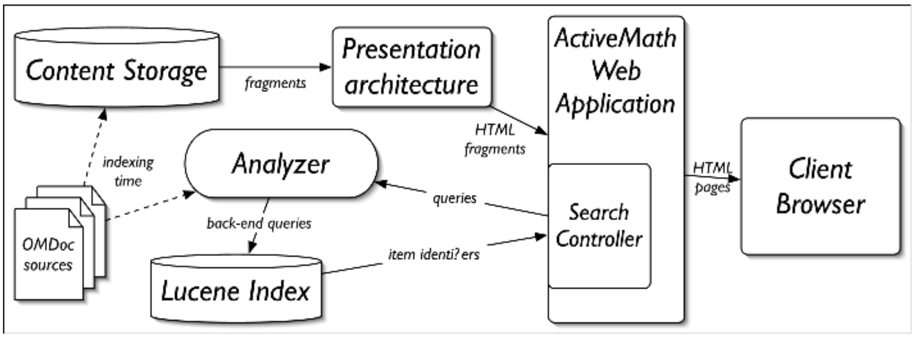
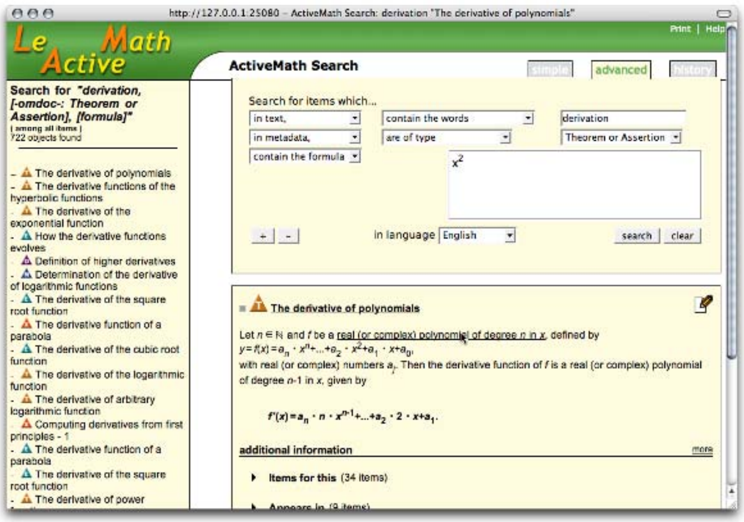


Fig. 2. Architecture of the ActiveMath search tool

Identifiers and References Each content item is marked by a content identifier. References to content items are presented as links within the browser presentations which makes them exchangeable by most desktop applications. Since these


http://127.0.0.1:25080 – ActiveMath Search: derivation "The derivative of polynomials"
Print | Help
Le Active Math
ActiveMath Search
simple
advanced
history
+
-
search
clear
more

Search for "derivation, [-omdoc-: Theorem or Assertion], [formula]"
(among all items)
722 objects found

- The derivative of polynomials
- The derivative functions of the hyperbolic functions
- The derivative of the exponential function
- How the derivative functions evolves
- Definition of higher derivatives
- Determination of the derivative of logarithmic functions
- The derivative of the square root function
- The derivative function of a parabola
- The derivative of the cubic root function
- The derivative of the logarithmic function
- The derivative of arbitrary logarithmic function
- Computing derivatives from first principles - 1
- The derivative function of a parabola
- The derivative of the square root function
- The derivative of power

Search for items which...

in text,	contain the words	derivation
in metadata,	are of type	Theorem or Assertion
contain the formula		x^2

in language English

The derivative of polynomials

Let $n \in \mathbb{N}$ and f be a real (or complex) polynomial of degree n in x, defined by
$y = f(x) = a_n \cdot x^n + \ldots + a_2 \cdot x^2 + a_1 \cdot x + a_0$,
with real (or complex) numbers a_i. Then the derivative function of f is a real (or complex) polynomial of degree n-1 in x, given by

$$f'(x) = a_n \cdot n \cdot x^{n-1} + \ldots + a_2 \cdot 2 \cdot x + a_1.$$

additional information

- **Items for this** (34 items)
- Appears in (9 items)

Fig. 3. The advanced search user interface

URIs are also downloadable, they provide an entry point to Web-robots. Among others, we used this in a comparison between the Google and ActiveMath search engines described in §5.

Storage and Extraction The OMDoc files are loaded in a content storage which splits the items and prepares them to be served individually. The search and presentation engines request the content of items, their metadata and the relations between them.

Query Input The user interface of the ActiveMath search tool allows for a set of queries which are expanded to low-level queries referring to the index. ActiveMath offers several kinds of queries: text queries, attribute queries, and formulæ queries. Simple queries allows the input of a conjunction of text and attribute queries.

In the advanced search mode users edit a Boolean combination of text, attribute, and formulæ queries: simple text fields are used for text queries, item attributes' queries are input using pop-up-menus, while formulæ queries are input with the Wiris input-editor.[1] A screenshot of the advanced search user-interface is in Figure 3.

Indexing, Analysis, and Back-end Queries The indexing process follows the Information Retrieval approach: the content is read from the OMDoc sources and decomposed by the *analysis process* in parallel streams of tokens. These streams are indexed by the Lucene library.[2] Each token is stored on the disk along with

[1] See `http://wiris.com/` for more information about the OpenMath input-editor that is provided in ActiveMath.

[2] Lucene is a Java library for high-performance retrieval at the Apache Software Foundation, `http://lucene.apache.org/`.

its position in the stream in a way that allows queries for (rows of) tokens to be efficiently matched.

We have developed an analysis process which converts the title, annotations metadata, formal and textual content (including formulæ) to tokens as follows:

– Annotations' attributes are stored as key-value pair tokens in their own fields.
– Word analysis applies the classical stemming and stop-words filtering following Martin Porter's algorithm [Por05].
– Words are also converted into a row of *phonetic tokens* which converts two phonetically equivalent words in the target language to the same phonetic tokens. This is a language-dependent process.
– Mathematical formulæ are converted from OpenMath to a row of tokens following the depth-first walk of the expression tree. This enables sub-terms to be matched.

4.1 Example Tokenization and Queries

Consider the following content item:

Trigonometric exercise

Let us assume $a + b = k$*.*

Our analysis process decomposes its content in named fields each populated with a sequence of tokens passed to the Lucene library for indexing. For the content item above, the following tokens are provided to the index:

```
id:                trigExo
attr:              type:exercise
title-en:          trigonometr exercis
text-en:           let us assum _(_1 _OMS_relation1/eq
                    _(_2 _OMS_arith1/plus _OMV_a _OMV_b _)
                    _2 _OMV_k _)_1
text-phonetic-en : LT US ASMN
```

With these token-streams in the index, queries for exact text, fuzzy text, item attributes, simple formulæ and formulæ with wild cards can be performed. They match the item each with a particular relevance score computed on the basis of the field and type of match (e.g., matches in titles are *boosted* by a factor of 10 as we expect them to be more relevant than matches in text):

– If the user enters "trigonometry" while working in English, the analysis converts this word to a query for token `trigonometr`, which is exactly matched to the field `title-en` of our item yielding score 10.0. If the user enters the word "assume", the analysis converts it the token "assum" which is exactly matched to the field `text-en` yielding a score of 1.0.
– If the user enters "asuming", the tokenization converts it to a query for the token "ASMN" in the *text-phonetic-en* field which is matched to the content of the phonetic english field (with score 0.8).

- A query for the type exercise would be reformulated as an index-query for the token `type:exercise` in the field `attr` which is matched to our item with score 1.0.
- If the user inputs the formula $a + b$ as formula query, it is translated to a query for the row of tokens

  `_(_`i `_OMS_arith1/plus _OMV_a _OMV_b _)_`i

  where i ranges from 1 to the maximum-depth in the index. This is exactly matched, with $i = 2$, to the tokens of our OpenMath representation of $a+b$ and thus yields a score of 1.0.
- the formula $? = k$ can be input as query by assigning the wild card role to the ? sign. It is translated to a query for the token sequence

  `_(_`i `_OMS_relation1/eq _? _OMV_k _)_`i

  where i ranges from 1 to the maximum-depth in the index and where `_?` is a wild card match of any row of tokens with the exception of the token `_(_`i. This can be exactly matched, with $i = 1$ to the OpenMath representation of our formula. The score of such a match is 1/6 which is computed on the length of the matched wild card.

Results Presentation The ActiveMath search tool returns the first page of results. Each result is displayed with bullets indicating the score, an icon of its type, and its title. The user can click it to obtain a display of the item.

The plain-text search, by default, returns conceptual content items of the current book with a sufficient relevance; a click can generalize this query. For a mathematical symbol only its definitions is presented. The tool is complemented by links that trigger an equivalent query to external sources of mathematical content on the Web.

Clicking on an item in the result list presents the item view. The ActiveMath presentation architecture converts the semantic `OMDoc` source into a format that is highly readable and is linked to other functionalities of ActiveMath; for example references to other items in the `OMDoc` source are transformed to HtML anchors linked to its item view. The system uses XSLT, caches, and a templating language to provide these presentations. It can render in HtML, xHtML +MathML, and pdf. To support the rendering of mathematical symbols, authorable notations are provided, see [MLUM06]. The *item view* that presents single items is accessible in each presentation of the item.

The description we have provided above shows that the ActiveMath search tool can be used to search for text with reasonable tolerance, for item attributes, as well as for formulæ with wild cards, that is, placeholders that are matched with any term.

5 Evaluation of the Search Tool

The search tool of ActiveMath has been evaluated along two methodologies. The first is a formative user testing. The second is a typical search evaluation with measures for precision and recall which is additionally complemented by a comparison to the Google search engine.

The tests were preformed with the LeActiveMath calculus content [LG06] a corpus equivalent to a typeset book of about 500 pages in English with full translations to German and Spanish. The index contains 2761 documents made of 560'259 tokens: 182765 words and 377'494 mathematical tokens spread in 36'389 formulæ, and 13'681 attribute tokens. On disk, this index takes about 10% of the size of its OMDoc sources.

Performance The simple text queries for learners are the slowest, ranging from 500 to 1500 milliseconds which is due mostly to the time taken to verify types of each result and replace symbols by their definitions. The advanced queries for mathematical terms, attributes, and words (both fuzzy and not) take between 50 and 150 milliseconds.

Formative evaluations At the University of Edinburgh 12 mathematics students were invited to discover and use the ActiveMath learning environment under the supervision of an expert. The *Think-aloud* protocols were taken. The evaluation indicated the learners found ActiveMath search tool quite useful and enjoyed an acceptable ease of use. It suggested the importance of labelling content items by types and indicated that learners start grasping the structure of content items, when using the search tool. The evaluation revealed a few usability glitches, most importantly the incomprehensibility of the word *metadata* to qualify queries for items' attributes.

Three German high school classes have used the same tools and content for several weeks. First observations from log files indicate that 95% of the spontaneous usage of the search tool are simple text queries.

Precision and Recall Evaluation This sample of the queries enriched with expected matches was tested as it is commonly done for a precision-and-recall evaluation [Mah06]. Some queries with their precision and recall measures are displayed in Table 1.

Table 1. A few example queries, their precision and recall

type	**query**	**evaluation**	**precision**	**recall**
plain-text	durchschnittliche	14 matches, 4 correct, no miss	0.286	1.0
plain-text	tangant	two results correct no miss	1.0	1.0
plain-text	sin	more than 20 matches, all wrong	0.0	0.0
formula	x^2	1 result correct, no miss	1.0	1.0
plain-text	theorem about quotient	1 correct match, no misses	1.0	1.0

The mean recall value is 0.93 (very high). The mean precision is 0.63, which is low since the fuzzy matching uses both the phonetic and edit-distance[3] matching approaches.

[3] The Lucene library offers a form of fuzzy queries which applies elementary modifications to the query words and recompute the matches. They are returned with a lower score based on the *edit-distance*, the amount of elementary modifications applied. Fuzzy textual matches in the ActiveMath search tool also use these queries.

Table 2. A few example queries with the number of matches in ActiveMath and Google search tools

query	ActiveMath amount matches	Google amount matches
whenever function differential quotient	1	1
tangent convex concave	15	8
parabola maximum	6	17
water maxmimum	39	0
tangant maximum	31	0
sin maximum	48	0
inflection point	243	27
inflection point (no fuzzy)	20	27
sketch	69	1
cauchy sequence	20	65

Since it requires a priori knowledge of the expected matches this sample of 40 queries in a book of 30 pages is small. However, for mathematical material, there seems to be no classical sample collection available such as the ones gathered for the TREC competitions.[4]

Google Comparison Another approach to evaluate a search engine is to compare the engine with another one which can also retrieve the content. Since ActiveMath is on the Web, it can be visited by Web-robots. We used the ability of the Google search engine to restrict its search on a given Web-server to compare Google results to ours.

A Web-server needs to be linked online in order to be accessible to Web-robots. It also requires to respond appropriately to robots' requests, that is, avoid HtML-frames and cookies which are both basic ingredients of rich browser-based Web-applications. In the versions of ActiveMath that was evaluated a special access was arranged for Web-robots, listing the content items. As a result, the Google search engine could retrieve the presentation of individual content items.

We gathered another set of queries, expected to be matched in the complete content of the collection realized in LeActiveMath [LG06] and compared the number of matches. Selected results can be found in Table 2. The Student-T-test comparison between the two columns indicated a t-score of 5.41 which indicates a significant difference. Distribution graphs of the number of matches are depicted in Figure 4 which indicate a broader variation of the ActiveMath search tool. One of the main differences of ActiveMath search tool compared to Google is the matches (good and bad) introduced by the fuzzy matches which the Google search engine does not provide. Two other factors led to the differences: the dates of the index construction differ; and the fact that Google searches the text of

[4] The TREC competition is a yearly competition organized along the TREC conferences by NIST where large collections of texts are given to participants, followed by queries. The result is evaluated, among others for precision and recall, by NIST. See `http://trec.nist.gov/` and the description in [Mah06].

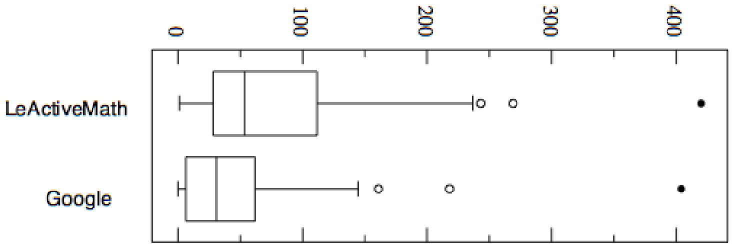


Fig. 4. The distribution graph of both ActiveMath and Google matches

the HtML item views which means, for example, that the text of the relations from the items, which are shown along the presented item, are considered to be part of the item or that the words of a presented formula, such as the word *sin* in $\sin x$, are matched as well.

6 Related Work

A few search tools allow for the query of mathematical terms, e.g. [AGC$^+$04] or [Urb04]. Because of the lack of a formal library, the ActiveMath search tool cannot manipulate the formulæ applying formal knowledge, for example term-ordering normalizations or symbol generalization. This results in a relatively low tolerance in formulæ search.

The search tool of the Digital Library of Mathematical Functions explained in [You05] is dedicated to functions. As a result it makes several normalizations of mathematical terms such as the conversion of $ab^{-1}cd^{-1}$ in $\frac{ac}{bd}$. Such normalizations introduce tolerance within the search tool. The usage of a simple type system for OpenMath objects could enable ActiveMath search tool to perform normalizations which would, otherwise, be abusive with mathematical objects such as group elements or matrices. The ActiveMath search tool also differentiates itself from the DLMF search tool by the user interface: while the DLMF search tool defines a plain-text input syntax, queries in the ActiveMath search tool are realized by input of concrete words, attribute-value, or formulæ.

Another avenue has been explored by Paul Cairns in [Cai04] where Latent Semantic Analysis can be used to provide a *semantic* distance between token-vectors, including mathematical terms. We started to explore LSA usage.

MathWebSearch [KS06] is a Web-crawler for documents with MathML content. This tool uses the term indexing techniques of [Gra96] to index formulæ collected on the Web. Compared to this search engine, the ActiveMath search tool is more learner-oriented but still lacks the ability to perform queries for formulæ with variables that occur several times. Since it uses the Lucene library, the ActiveMath search tool appears better scalable compared to the term indexing technique which needs an in-memory representation.

From the overall access point-of-view, the ActiveMath learning environment seems to be one of the rare mathematical content items presentation servers which manages items with a fine granularity. A few similar projects are the Thesaurus at `http://thesaurus.maths.org/`, whose goal is an international dictionary of mathematical concepts, or the encyclopedia projects. ActiveMath is the only one among them which offers search by attributes and formulæ and a few other features which are consequences of the semantic nature of the content encoding.

7 Conclusion

In this article, we described the access to mathematical content items in ActiveMath in particular, through its search tool and presented evaluations of this search tool. The main contribution of this research is the development of an analysis process for mathematical formulæ which converts them to streams of tokens on which information retrieval techniques can be applied. Using the Lucene library for the indexing matching makes the search tool efficient We also developed a dedicated ranking scheme for search results which allowed us to order most fuzzy matches below exact matches. The search interface has been designed for learners and it has been evaluated for usability. Two main conclusions can be drawn from the evaluations:

Not surprisingly, fuzziness introduces noise into the search results. Conversely, the fuzziness introduces tolerance to the queries.

Future Work The comparison with other search engines will be refined, on the one hand by enlarging the size of the sample and on the other hand, by using automated methods to construct the sample, perform the test, and to invoke queries on the Web-robot.

We are investigating the benefits of alternative indexing and matching mechanisms [Gra96] that may be more suitable for mathematical expressions.

We started working on access by Web-robots that can understand the mathematical semantic documents. They will be crawling `OMDoc` documents, or semantically rich xHtML documents in which mathematical formulæ are given in MathML whose semantic is given in parallel markup.

Acknowledgements

This publication is partly a result of work in the context of the LeActiveMath project, funded under the 6th Framework Program of the European Community – (Contract IST-2003-507826). The authors are solely responsible for its content.

References

AGC+04. A. Asperti, F. Guidi, C. Sacerdoti Coen, E. Tassi, and S. Zacchiroli. A content based mathematical search engine: Whelp. In J.-C. Filliatre, C. Paulin-Mohring, and B. Werner, editors, *Proceedings of the TYPES 2004*, LNCS 3839, pages 17–32. Springer-Verlag, 2004. See also `http://www.cs.UniBo.it/helm/`.

BCC+04. S. Buswell, O. Caprotti, D. Carlisle, M. Dewar, M. Gaëtano, and M. Kohlhase. The OpenMath standard, version 2.0. Technical report, The OpenMath Society, June 2004. Available at `http://www.openmath.org/`.

Cai04. P. Cairns. Informalising formal mathematics. In A. Asperti, G. Bancerek, and A. Trybulec, editors, *Proceedings of MKM 2004*, volume 3119 of *LNCS*, pages 58–72. Springer-Verlag, 2004. Available at `http://www.uclic.ucl.ac.uk/paul/research/MizarLSI.pdf`.

Gra96. P. Graf. *Term Indexing*, volume 1053 of *LNCS*. Springer-Verlag, 1996.

Koh00. M. Kohlhase. `OMDoc`: Towards an OPENMATH representation of mathematical documents. Seki Report SR-00-02, Fachbereich Informatik, Universität des Saarlandes, 2000. See also `http://www.mathweb.org/omdoc`.

KS06. M. Kohlhase. and I. Sucan, A Search Engine for Mathematical Formulæ. Proceedings of AISC 06, LNAI 4120, See also `http://kwarc.eecs.iu-bremen.de/software/ mmlsearch/`.

LG06. P. Libbrecht and C. Gross. Experience report writing LeActiveMath Calculus. In Proceedings of MKM 2006, LNAI 4108, Springer Verlag,

MAB+01. E. Melis, E. Andrès, J. Büdenbender, A. Frischauf, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. ActiveMath: A Generic and Adaptive Web-Based Learning Environment. *International Journal of Artificial Intelligence in Education*, 12(4):385–407, 2001.

MAF+03. E. Melis, J. Büdenbender E. Andrès, A. Frischauf, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. Knowledge Representation and Management in ACTIVEMATH. *Annals of Mathematics and Artificial Intelligence*, 38(1-3):47–64, 2003. Volume is accessible from `http://monet.nag.co.uk/mkm/amai/index.html`.

Mah06. K. Mahesh. Text retrieval quality: A primer. Technical report, Oracle Coproration, 2006. See `http://www.oracle.com/technology/products/text/htdocs/imt_quality.htm`.

MLUM06. S. Manzoor, P. Libbrecht, C. Ullrich, and E. Melis. Authoring presentation for OpenMath. In Michael Kohlhase, editor, Proceedings of MKM 2005, volume 3863 of *LNCS*, pages 33–48, Heidelberg, 2006. Springer.

Por05. Martin Porter. The Porter stemming algorithm, 2005. See `http://www.tartarus.org/ martin/PorterStemmer/`.

Urb04. J. Urban. Momm - fast interreduction and retrieval in large libraries of formalized mathematics. In *Proceedings of the ESFOR 2004 workshop at IJCAR'04, 2004*. More information on the project at `http://wiki.mizar.org/cgi-bin/twiki/view/Mizar/MoMM`

vR79. C.J. van Rijsbergen. *Information Retrieval*. Butterworths, 1979. Available at `http://www.dcs.gla.ac.uk/~iain/keith/`.

You05. A. Youssef. Information search and retrieval of mathematical contents: Issues and methods. In *proceedings of IASSE-2005*, Toronto, Canada, 2005.

Logiweb - A System for Web Publication of Mathematics

Klaus Grue

Dept.Comp.Sci, Univ.of Copenhagen (DIKU)
grue@diku.dk

Abstract. Logiweb is a system for electronic publication and archival of machine checked mathematics of high typographic quality. It can verify the formal correctness of *pages*, i.e. mathematical papers expressed suitably. The present paper is an example of such a Logiweb page and the present paper is formally correct in the sense that it has been verified by Logiweb. The paper may of course contain informal errors like any other paper. Logiweb is neutral with respect to choice of logic and choice of notation and can support any kind of formal reasoning.

Logiweb uses the World Wide Web to publish Logiweb pages and Logiweb pages can be viewed by ordinary Web browsers. Logiweb pages can reference definitions, lemmas, and proofs on previously referenced Logiweb pages across the Internet. When Logiweb verifies a Logiweb page, it takes all transitively referenced pages into account.

1 Introduction

Logiweb is a web-like system that allows mathematicians and computer scientists to web-publish pages with high typographic quality and high human readability which are also machine verifiable. Among other, Logiweb allows pages to contain definitions of formal theories, definitions of new constructs, programs, lemmas, conjectures, and proofs. Furthermore, Logiweb allows pages to refer to each other across the Internet, and allows proof checking of proofs that span several pages that reside different places in the world. As an example, a lemma on one page may refer to a construct which is defined on another page situated elsewhere, in which case the proof checker must access both pages to establish the correctness of the proof.

Logiweb is accumulative and provides a medium for archived mathematics, c.f. Section 2.2. In contrast, the World Wide Web is a medium suited for information in flux, i.e. information which may be updated at any time without notice. There are many formats like MathML and OMDoc [1,2] which are suited for mathematics on the web. After many experiments and investigations, html, pdf, and TEX have been chosen as interface formats of Logiweb until further, but Logiweb is an open system for which individual users may add support for further formats. Logiweb also has its own, internal format and referencing system which is particularly suited for archived mathematic.

© Springer-Verlag Berlin Heidelberg 2006

Like the Internet and the WWW, Logiweb is a robust, 'anarchistic' system that runs without any central authority.

Logiweb gives complete notational freedom to its users as well as complete freedom to choose any axiomatic theory (e.g. ZFC) as basis for their work. Logiweb also allows different notational systems and theories to co-exist and interact smoothly.

Logiweb may be used as it is but also has the potential to support other systems like Mizar [3,4]. Logiweb was originally designed to support Map Theory [5,6,7,8] which has the same power as ZFC but relies on very different foundations in that, e.g., it relies on λ-calculus *instead* of first order predicate calculus. However, Logiweb has been designed such that it supports all axiomatic theories equally well so the ability to support Map Theory should be seen as a widening rather than a narrowing of the scope.

Logiweb supports classical as well as intuitionistic logic, it supports theories built on first order predicate calculus as well as other brands of theories, and it supports theories (such as Map Theory) which admits general recursive definitions.

The absence of restrictions on the choice of logic of course makes it impossible to supply a general code-from-theorems extraction facility like `term_of` of Nuprl [9], but functions for manipulation of theorems and proofs of individual theories are expressible in the programming language of Logiweb.

One goal of Logiweb was to design a simple proof system which allows to cope with the complexity of mathematical textbooks. To ensure that the system can cope with the complexity of a full, mathematical textbook in a human readable style, two books [10,11] have been developed 1992-2002 to test the system.

Reference [10] is a discrete math book for first year university students and is of interest here because it has been possible to test the human readability of the book in practice. The associated course has been given ten times with a total of more than a thousand students. The course has been a success and runs as the first course on the computer science curriculum at DIKU in parallel with a course on ML.

Reference [11] is a treatise on Map Theory and is of interest here because it contains a substantial proof (a proof of the consistency of ZFC expressed in Map Theory) that can stress test Logiweb. To allow comparison with other proof systems and to ensure correctness, [11] has been ported by hand to Isabelle [12,13,7].

During 2005 and 2006, Logiweb has been used on two graduate courses in logic (c.f. 'Student reports' at `http://logiweb.eu/`) and Logiweb is being adapted according to user requests. After that, it is the intension to run first [11] and then [10] through the system. Running those two books through Logiweb requires adaption of the books to the current syntax of the Logiweb compiler plus programming of a number of proof tactics that are described but not formally defined in the books. Running [11] through Logiweb will also allow a comparison with Isabelle.

Logiweb includes a programming language, macro expansion facilities, proof checking facilities, means for expressing proof tactics, a protocol for exchanging information about pages, a format for storing and transmitting pages, machinery for rendering, machinery for compiling programs, machinery for referencing, and many other facilities. Each facility is simple, but the sum of features makes it impossible to cover everything here. For a comprehensive introduction to Logiweb, consult Logiweb itself at `http://logiweb.eu/` and read the 'base' page.

Section 3 gives examples of theories, lemmas and proofs expressed using Logiweb. Section 2 gives some background information about the structure of Logiweb and Section 4 gives a short description of how Logiweb has been used in teaching mathematics. Section 5 gives an overview of the implementation of the system.

2 Description of Logiweb

2.1 Vectors and References

Logiweb stores and transmits Logiweb pages in its own, internal format named the *Logiweb interchange format* and interacts with other systems using standardized formats like html, PDF, Lisp S-expressions, and many other formats. We shall refer to pages expressed in the Logiweb interchange format as *vectors*. Vectors are sequences of bytes.

Each Logiweb page has a unique Logiweb reference. A reference is a sequence of about thirty bytes. When expressed base 64 [14], the reference of the present paper is BokG7o6Za5JH5cFpE6nduncQ-rtRhuYtj-e_iigBB. The reference contains a protocol version number (always one, reserved for future extensions), a RIPEMD-160 [15] hash key, and a time stamp.

One may look up the present page on Logiweb following the link http://logiweb.eu:8080/logiweb/server/relay/64/BokG7o6Za5JH5cFpE6nduncQ-rtRhuYtj-e_iigBB/2/index.html. When clicking that link, a CGI-program at http://logiweb.eu:8080/logiweb/server/relay/ relays the reference BokG7o6Za5JH5cFpE6nduncQ-rtRhuYtj-e_iigBB to a Logiweb *server*, which resolves the reference and directs the users browser to the page.

Once a Logiweb page is submitted to Logiweb, its Logiweb reference remains fixed whereas its location may change. At any time there may exist many copies of each page on Logiweb and all the copies may be moved around, but all Logiweb servers cooperate on keeping track of all Logiweb pages at any time.

2.2 Immutability

Logiweb pages are *immutable* in the sense that if a user looks up the same Logiweb reference twice, even with years in between, and if Logiweb still has a copy of the page somewhere in the world, then the user can be sure to receive exactly the same Logiweb page the two times. If there are no copies left, then the user can be sure to get a message that the page does not exist anymore. Any user of Logiweb can mirror any Logiweb page and thereby ensure its continued existence. In this way, Logiweb pages are as stable as papers published in paper journals: If all copies of a paper journal burn, the contents is lost, but as long as at least one copy remains, the contents is still available.

An important feature of paper journals is that one cannot change a paper after publication. That makes it safe to refer e.g. to a definition in a published paper since the definition is fixed after publication. Logiweb mimics this feature through immutability.

2.3 Rendering

Given a Logiweb vector, i.e. a Logiweb page expressed in the Logiweb interchange format, Logiweb is able to *render* and *verify* the page.

Logiweb may render pages in arbitrary formats. The present page is rendered as one html and two PDF files, located the following places:

– http://logiweb.eu:8080/logiweb/server/relay/64/BokG7o6Za5JH5cFpE6nduncQ-rtRhuYtj-e_iigBB/2/body/tex/page.pdf
– http://logiweb.eu:8080/logiweb/server/relay/64/BokG7o6Za5JH5cFpE6nduncQ-rtRhuYtj-e_iigBB/2/body/tex/appendix.pdf [16]
– http://logiweb.eu:8080/logiweb/server/relay/64/BokG7o6Za5JH5cFpE6nduncQ-rtRhuYtj-e_iigBB/2/body/tex/page.html

The first link points to the present paper. The second link points to an electronic appendix with definitions that would bore most readers (such as definitions of how each construct should be rendered). The second link is included in the BibTeX bibliography [16] for easy reference. The third link points to a table of contents.

2.4 Verification

When verifying a page, Logiweb collects all definitions and verifies all claims found on the page. As an example, one may define

$$[\mathsf{n}! \doteq \textbf{if } \mathsf{n} = 0 \textbf{ then } 1 \textbf{ else } \mathsf{n} \cdot (\mathsf{n} - 1)!]$$

and one may claim

$$[3! = 6]^{\cdot}$$

which makes Logiweb verify that 3! equals 6 according to the given definition of the factorial function.

The claim above is a rendering of the bytes in the vector whose correctness is established by the verifier. This ensures that a human reader and the machine verifier see the same claims.

Verification ignores all text that has no formal meaning. The $\doteq$ and $[\cdots]^{\cdot}$ constructs have formal meaning in that they introduce definitions and claims, respectively. Verification ignores running text such as the present sentence and also ignores formulas like 3! which neither form a definition nor a claim. Section 3 gives examples of more advanced definitions like definitions of axiomatic systems and more advanced claims like mathematical proofs (where the claim to be verified is that the proof is correct).

2.5 Bibliographies

Each Logiweb page contains a Logiweb bibliography i.e. a list of references to other Logiweb pages. The Logiweb bibliography of the present paper (not to be confused with the BibTeX bibliography at the end of the paper) references two other Logiweb pages:

- http://logiweb.eu:8080/logiweb/server/relay/64/BAGTyrgl5Qmkb-DUmGqAWUda0vx9aHftYSe_iigBB/2/ [17]
- http://logiweb.eu:8080/logiweb/server/relay/64/BsPa2QtKsY6_LULMdLl-THnDv58AVac-Mbe_iigBB/2/ [18]

The references above have been included in the BibTeX bibliography for easy reference but there is in general no link between the Logiweb and the BibTeX bibliographies.

The definition

$$[\mathsf{n}! \doteq \textbf{if } \mathsf{n} = 0 \textbf{ then } 1 \textbf{ else } \mathsf{n} \cdot (\mathsf{n} - 1)!]$$

of the factorial function refers, among other, to multiplication $\mathsf{x} \cdot \mathsf{y}$. [17] both defines how to evaluate and how to render multiplication, and multiplication is available on the present page because [17] is included in the Logiweb bibliography of the present page. The immutability of [17] ensures that $\mathsf{x} \cdot \mathsf{y}$ denotes multiplication every time $[3! = 6]^{\cdot}$ is verified.

3 Example Proofs

3.1 Theories

We now give a very simple example of a theory, a lemma, and a proof. The theory is classical propositional calculus L as defined in [19]:

- [**Theory** L]
- [L **rule** MP: $\Pi\mathcal{A}\colon \Pi\mathcal{B}\colon \mathcal{A} \vdash \mathcal{A} \Rightarrow \mathcal{B} \vdash \mathcal{B}$]
- [L **rule** A1: $\Pi\mathcal{A}\colon \Pi\mathcal{B}\colon \mathcal{A} \Rightarrow \mathcal{B} \Rightarrow \mathcal{A}$]
- [L **rule** A2: $\Pi\mathcal{A}\colon \Pi\mathcal{B}\colon \Pi\mathcal{C}\colon (\mathcal{A} \Rightarrow \mathcal{B} \Rightarrow \mathcal{C}) \Rightarrow (\mathcal{A} \Rightarrow \mathcal{B}) \Rightarrow \mathcal{A} \Rightarrow \mathcal{C}$]
- [L **rule** A3: $\Pi\mathcal{A}\colon \Pi\mathcal{B}\colon (\neg\mathcal{B} \Rightarrow \neg\mathcal{A}) \Rightarrow (\neg\mathcal{B} \Rightarrow \mathcal{A}) \Rightarrow \mathcal{B}$]

Logiweb macro expands [**Theory** L] into a definition of L. That definition defines L as the meta-conjunction of all rules attributed to L.

Axiom A1 above says that $\mathcal{A} \Rightarrow \mathcal{B} \Rightarrow \mathcal{A}$ holds for all terms $\mathcal{A}$ and $\mathcal{B}$. The meta-quantifier $\Pi\mathcal{A}\colon \mathcal{B}$ states that $\mathcal{B}$ holds for all *terms* $\mathcal{A}$ as opposed to the object quantifier $\forall \mathsf{x}\colon \mathcal{B}$ which states that $\mathcal{B}$ holds for all *values* x.

The electronic appendix [16] defines priority and associativity such that $\Pi\mathcal{A}\colon\Pi\mathcal{B}\colon\mathcal{A}\Rightarrow\mathcal{B}\Rightarrow\mathcal{A}$ means $\Pi\mathcal{A}\colon\Pi\mathcal{B}\colon(\mathcal{A}\Rightarrow(\mathcal{B}\Rightarrow\mathcal{A}))$.

Inference rule L above expresses the rule of modus ponens. $\Pi\mathcal{A}\colon\Pi\mathcal{B}\colon\mathcal{A}\vdash\mathcal{A}\Rightarrow\mathcal{B}\vdash\mathcal{B}$ means $\Pi\mathcal{A}\colon\Pi\mathcal{B}\colon(\mathcal{A}\vdash((\mathcal{A}\Rightarrow\mathcal{B})\vdash\mathcal{B}))$.

The rules of a theory may be stated together as above or may be scattered throughout one Logiweb page. Presentations of mathematical logic often scatters the axioms of a theory over several chapters (c.f. the definition of set theory in [19]). Support for scattering is not included in Logiweb itself; it is implemented in [17] and is available to all users of Logiweb who reference that page. Users who want another style of presentation may replace [17] by their own alternative.

3.2 Lemmas and Proofs

The first formal proof in [19] proves the following lemma:

[L **lemma** I: $\Pi\mathcal{A}\colon\mathcal{A}\Rightarrow\mathcal{A}$]

The lemma says that $\mathcal{A}\Rightarrow\mathcal{A}$ holds for all terms $\mathcal{A}$ in propositional calculus. The classical proof of that reads:

L **proof of** I:

L01:	Arbitrary $\gg$	$\mathcal{A}$	;
L02:	A1 $\gg$	$\mathcal{A}\Rightarrow(\mathcal{A}\Rightarrow\mathcal{A})\Rightarrow\mathcal{A}$	;
L03:	A2 $\gg$	$(\mathcal{A}\Rightarrow(\mathcal{A}\Rightarrow\mathcal{A})\Rightarrow\mathcal{A})\Rightarrow$ $(\mathcal{A}\Rightarrow\mathcal{A}\Rightarrow\mathcal{A})\Rightarrow(\mathcal{A}\Rightarrow\mathcal{A})$	;
L04:	MP $\rhd$ L02 $\rhd$ L03 $\gg$	$(\mathcal{A}\Rightarrow\mathcal{A}\Rightarrow\mathcal{A})\Rightarrow(\mathcal{A}\Rightarrow\mathcal{A})$	;
L05:	A1 $\gg$	$\mathcal{A}\Rightarrow\mathcal{A}\Rightarrow\mathcal{A}$	;
L06:	MP $\rhd$ L05 $\rhd$ L04 $\gg$	$\mathcal{A}\Rightarrow\mathcal{A}$	$\square$

The proof above is rendered in a formalistic, tabular style close to that of [19]. Other styles, such as rendering as running text is also possible.

3.3 A Proof by Induction

Reference [18] defines Peano arithmetic S as follows:

- [**Theory** S]
- [S **rule** S1: $\Pi\mathcal{A},\mathcal{B},\mathcal{C}\colon\mathcal{A}=\mathcal{B}\vdash\mathcal{A}=\mathcal{C}\vdash\mathcal{B}=\mathcal{C}$]
- [S **rule** S2: $\Pi\mathcal{A},\mathcal{B}\colon\mathcal{A}=\mathcal{B}\vdash\mathcal{A}'=\mathcal{B}'$]
- [S **rule** S3: $\Pi\mathcal{A}\colon\neg 0=\mathcal{A}'$]
- [S **rule** S4: $\Pi\mathcal{A},\mathcal{B}\colon\mathcal{A}'=\mathcal{B}'\vdash\mathcal{A}=\mathcal{B}$]
- [S **rule** S5: $\Pi\mathcal{A}\colon\mathcal{A}+0=\mathcal{A}$]
- [S **rule** S6: $\Pi\mathcal{A},\mathcal{B}\colon\mathcal{A}+\mathcal{B}'=(\mathcal{A}+\mathcal{B})'$]
- [S **rule** S7: $\Pi\mathcal{A}\colon\mathcal{A}\cdot 0=0$]
- [S **rule** S8: $\Pi\mathcal{A},\mathcal{B}\colon\mathcal{A}\cdot(\mathcal{B}')=(\mathcal{A}\cdot\mathcal{B})+\mathcal{A}$]
- [S **rule** S9: $\Pi\mathcal{X},\mathcal{A},\mathcal{B},\mathcal{C}\colon\langle\mathcal{B}\equiv\mathcal{A}|\mathcal{X}{:=}0\rangle\Vdash\langle\mathcal{C}\equiv\mathcal{A}|\mathcal{X}{:=}\mathcal{X}'\rangle\Vdash\mathcal{B}\vdash\mathcal{A}\Rightarrow\mathcal{C}\vdash\mathcal{A}$][1]

[1] $\langle\mathcal{A}\equiv\mathcal{B}|\mathcal{X}{:=}\mathcal{C}\rangle$ says '$\mathcal{A}$ is identical (except for naming of bound variables) to $\mathcal{B}$ where $\mathcal{X}$ is replaced by $\mathcal{C}$ in $\mathcal{B}$, c.f. [16].

– [S **rule** MP: $\Pi\mathcal{A},\mathcal{B}\colon \mathcal{A} \Rightarrow \mathcal{B} \vdash \mathcal{A} \vdash \mathcal{B}$]
– [S **rule** Gen: $\Pi\mathcal{X},\mathcal{A}\colon \mathcal{A} \vdash \forall\mathcal{X}\colon \mathcal{A}$]
– [S **rule** Ded: $\Pi\mathcal{A},\mathcal{B}\colon \mathrm{Ded}(\mathcal{A},\mathcal{B}) \Vdash \mathcal{A} \vdash \mathcal{B}$]
– [S **rule** Neg: $\Pi\mathcal{A}\colon \Pi\mathcal{B}\colon \neg\mathcal{B} \Rightarrow \neg\mathcal{A} \vdash \neg\mathcal{B} \Rightarrow \mathcal{A} \vdash \mathcal{B}$]

Furthermore, [18] proves the following lemma taken from [19]:

[S **lemma** Prop 3.2c: $\Pi\mathcal{A},\mathcal{B},\mathcal{C}\colon \mathcal{A} = \mathcal{B} \vdash \mathcal{B} = \mathcal{C} \vdash \mathcal{A} = \mathcal{C}$]

Since the Logiweb bibliography of the present page points to [18], system S and the lemma above are available in the present paper. We may use that to prove $\forall x\colon 0 \cdot x = 0$ by induction in x. The induction step reads:

[S **lemma** Prop 3.2l′: $\Pi\mathcal{A}\colon 0 \cdot \mathcal{A} = 0 \Rightarrow 0 \cdot \mathcal{A}' = 0$]
S **proof of** Prop 3.2l′:

L01:	Arbitrary $\gg$	$\mathcal{A}$	;
L02:	Block $\gg$	Begin	;
L03:	Arbitrary $\gg$	$\mathcal{A}$	;
L04:	Premise $\gg$	$0 \cdot \mathcal{A} = 0$	;
L05:	S8 $\gg$	$0 \cdot \mathcal{A}' = 0 \cdot \mathcal{A} + 0$	;
L06:	S5 $\gg$	$0 \cdot \mathcal{A} + 0 = 0 \cdot \mathcal{A}$	;
L07:	Prop 3.2c $\rhd$ L05 $\rhd$ L06 $\gg$	$0 \cdot \mathcal{A}' = 0 \cdot \mathcal{A}$	;
L08:	Prop 3.2c $\rhd$ L07 $\rhd$ L04 $\gg$	$0 \cdot \mathcal{A}' = 0$	;
L09:	Block $\gg$	End	;
L10:	Ded $\rhd$ L09 $\gg$	$0 \cdot \mathcal{A} = 0 \Rightarrow 0 \cdot \mathcal{A}' = 0$	□

In the proof above, the begin-end-block in Line L02-L09 proves $0 \cdot \mathcal{A}' = 0$ under the assumption $0 \cdot \mathcal{A} = 0$. The assumption in L04 involves the meta-variable $\mathcal{A}$, which has an arbitrary, fixed value according to line L03. Line L10 then uses deduction to conclude $0 \cdot \mathcal{A} = 0 \Rightarrow 0 \cdot \mathcal{A}' = 0$ where $\mathcal{A}$ is arbitrary according to L01 (L03 has no effect outside the block).

The premise in L04 has to be expressed using a meta-variable $\mathcal{A}$. A statement with a free object variable x is understood to hold for all values of the object variable so that e.g. $0 \cdot x = 0$ implicitly means $\forall x\colon 0 \cdot x = 0$ which is not suitable in line L04 above. Ordinary text books on mathematics typically leave it to the reader to guess the scope and kind of each variable, but we have to be more pedantic here to make the proofs machine verifiable.

We may now prove [S **lemma** Prop 3.2l: $\forall x\colon 0 \cdot x = 0$]: S **proof of** Prop 3.2l:

L01:	S7 $\gg$	$0 \cdot 0 = 0$	;
L02:	Prop 3.2l′ $\gg$	$0 \cdot x = 0 \Rightarrow 0 \cdot x' = 0$	;
L03:	S9 @ x $\rhd$ L01 $\rhd$ L02 $\gg$	$0 \cdot x = 0$	;
L04:	Gen $\rhd$ L03 $\gg$	$\forall x\colon 0 \cdot x = 0$	□

The proofs given so far are rather trivial but still illustrate what the Logiweb system can do. For more complex examples, consult `http://logiweb.eu/` and [10,11].

4 Uses of Logiweb

Until further, Logiweb has been used in a course on mathematical logic for graduate students. The course was given in the spring of 2005 and again in the spring of 2006. Predecessors of the course, using predecessors of Logiweb, have been given a number of times since 1986.

During the above mentioned course on mathematical logic, the students form groups and choose a theorem they want to prove. After that each group writes a report which formally proves the theorem, verifies and publishes the report using Logiweb, and hands in the Logiweb reference of the report by e-mail.

Immutability (as guaranteed by RIPEMD-160 [15]) ensures that the report is not changed after it is handed in. Students will typically publish many attempts before they produce a version they want to hand in and, if they want, they may continue publishing new versions after handing in the report, but the students get a mark based on the version that matches the reference they send by e-mail.

Students typically base their reports on other Logiweb pages such as [17] and [18] which define elementary notions. The students typically locate the pages they want to reference by ordinary html browsing e.g. starting at `http://logiweb.eu/` or the home page `http://www.diku.dk/undervisning/2006s/202/` of the course.

It is the intension to use Logiweb also for teaching of mathematics for computer science based on [10]. In particular, it is planned to let the students answer exam questions using Logiweb during a four hour written exam. This scenario has been tested with success on a small group of first year students in 2004 using an old version of Logiweb.

5 Implementation Overview

A user may use the World Wide Web as shown in Figure 1. In the figure, the user may use the text editor to construct an html page and store it in the file system within reach of the http server. Then the user (or another user) may use the html browser to request the html page from the http server which in turn retrieves the html page from the file system.

Figure 2 shows how a user may use Logiweb. To write a Logiweb page, the user prepares a source text and invokes the Logiweb compiler on it. This is similar to running TEX on a TEX source [20]. Actually, much of a Logiweb source consists of TEX source code.

If the compiler succeeds in interpreting the source, then the compiler translates the source to a Logiweb vector, checks the correctness of the vector, and stores the vector back in the file system within reach of the http server. The compiler also renders the page in PDF so that users without a genuine Logiweb browser can view it. After that, any user that knows the url of the page can retrieve it using an html browser.

When the compiler succeeds in translating a Logiweb page, it also computes the Logiweb reference of the page and notifies the Logiweb server (c.f. Figure 2).

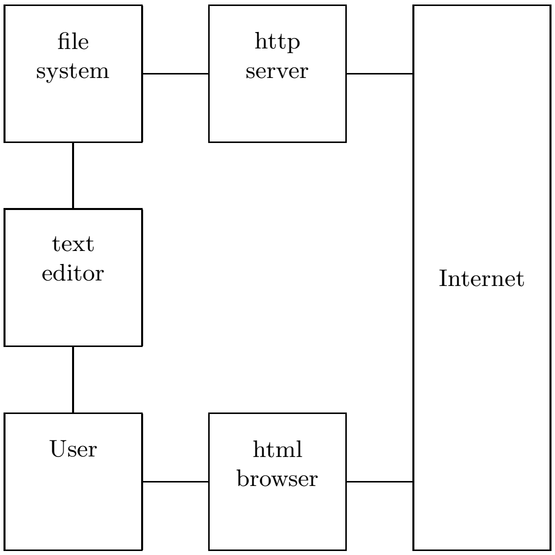


Fig. 1. World Wide Web

The Logiweb server keeps track of the relationship between http urls and Logiweb references and makes the relationship available via the Internet using the Logiweb protocol. The Logiweb protocol allows Logiweb servers to cooperate on indexing pages such that each server merely has to keep track of local pages plus some information about which other Logiweb servers to refer non-local requests to.

When the compiler translates a Logiweb page that references other Logiweb pages (which is the normal case), it uses the Logiweb server to locate the references and then transitively loads the referenced pages so that all definitions on transitively referenced pages are available.

A Logiweb relay (c.f. Figure 2) is a CGI-program which, given a reference, contacts the nearest Logiweb server, translates the reference to an ordinary url, and returns an html indirection to that url. This instructs the html browser of the user to fetch the associated page. The net experience for the user is that clicking a Logiweb reference in an html page makes the html browser navigate to the referenced Logiweb page. One may construct a reference by appending the following:

– an url like http://logiweb.eu:8080/logiweb/server/relay/ of a relay,
– a Logiweb reference like 64/BokG7o6Za5JH5cFpE6nduncQ-rtRhuYtj-e_iig BBin base 16, 32, or 64,
– a number of levels like /2/ to go upwards in the directory structure (/2/ corresponds to /../../), and
– a relative reference like index.html

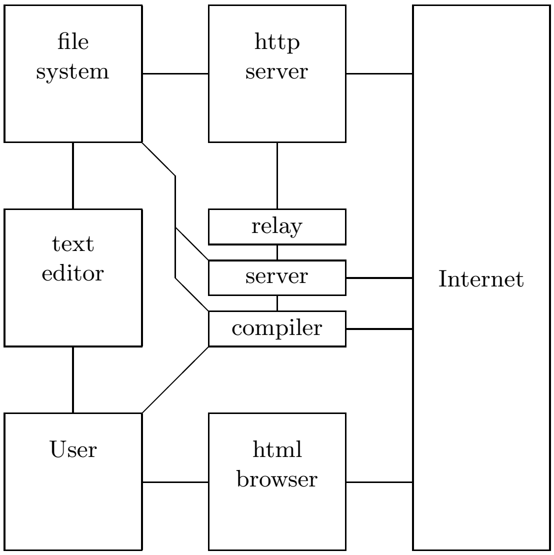


Fig. 2. Logiweb

Referencing from Logiweb pages to html pages is trivial but not necessarily advisable since the immutability of Logiweb pages makes it impossible to repair broken links.

For more details on Logiweb see `http://logiweb.eu/` or [21].

6 Status

Logiweb allows users to author, render, publish, verify, retrieve, and read pages that contain formal mathematics. At the time of writing, more than 600 kilobyte of Logiweb source text has been verified by Logiweb. In addition, in 2005, ten graduate students have written eight reports that have been formally verified by a test version of Logiweb (c.f. `http://logiweb.eu/`), and the present version is being used in the spring of 2006 for another group of students. 800 kilobyte of formal proofs [11] await verification.

References

1. Kohlhase, M.: OMDoc: An open markup format for mathematical documents (version 1.1) (2003) `http://www.mathweb.org/omdoc/index.html`.
2. Miner, R., Schaeffer, J.: A gentle introduction to MathML (2001) http://www.dessci.com/en/support/tutorials/mathml/default.htm.
3. Trybulec, A., Blair, H.: Computer assisted reasoning with MIZAR. In Joshi, A., ed.: Proceedings of the 9th International Joint Conference on Artificial Intelligence, Los Angeles, CA, Morgan Kaufmann (1985) 26–28 `http://www.mizar.org/`.

4. Muzalewski, M.: An Outline of PC Mizar. Foundation of Logic, Mathematics and Informatics, Mizar User Group, Brussels. (1993)
5. Berline, C., Grue, K.: A κ-denotational semantics for Map Theory in ZFC+SI. Theoretical Computer Science **179**(1–2) (1997) 137–202
6. Grue, K.: Map theory. Theoretical Computer Science **102**(1) (1992) 1–133
7. Skalberg, S.C.: An Interactive Proof System for Map Theory. PhD thesis, University of Copenhagen (2002) `http://www.mangust.dk/skalberg/phd/`.
8. Vallée, T.: "Map Theory" et Antifondation. Volume 79 of Electronic Notes in Theoretical Computer Science. Elsevier (2003)
9. Constable, R.L., Allen, S., Bromly, H., Cleaveland, W., Cremer, J., Harper, R., Howe, D., Knoblock, T., Mendler, N., Panangaden, P., Sasaki, J., Smith, S.: Implementing Mathematics with the Nuprl Proof Development System. Prentice-Hall, Englewood Cliffs, New Jersey (1986)
10. Grue, K.: Mathematics and Computation. 7 edn. Volume 1–3. DIKU, Universitetsparken 1, DK-2100 Copenhagen (2001)
11. Grue, K.: Map theory with classical maps. Technical Report 02/21, DIKU (2002) `http://www.diku.dk/publikationer/tekniske.rapporter/2002/`.
12. Paulson, L.C.: Introduction to Isabelle. Technical report, University of Cambridge, Computer Laboratory (1998)
13. Paulson, L.C.: The Isabelle reference manual. Technical report, University of Cambridge, Computer Laboratory (1998)
14. Berners-Lee, T., Masinter, L., McCahill, M.: Uniform Resource Locators (URL). RFC 1738, Internet Engineering Task Force (1994) `http://rfc.net/rfc1738.html`.
15. Dobbertin, H., Bosselaers, A., Preneel, B.: RIPEMD-160: A strengthened version of RIPEMD. In: Fast Software Encryption. (1996) 71–82 `http://citeseer.ist.psu.edu/dobbertin96ripemd.html`.
16. Grue, K.: Logiweb - a system for web publication of mathematics - appendix. Technical report, Logiweb (2006) http://logiweb.eu:8080/logiweb/server/relay/64/BokG7o6Za5JH5cFpE6nduncQ-rtRhuYtj-e_iigBB/2/body/tex/appendix.pdf.
17. Grue, K.: A logiweb base page. Technical report, Logiweb (2006) http://logiweb.eu:8080/logiweb/server/relay/64/BAGTyrgl5Qmkb-DUmGqAWUda0vx9aHftYSe_iigBB/2/.
18. Grue, K.: The logiweb sequent calculus. Technical report, Logiweb (2006) http://logiweb.eu:8080/logiweb/server/relay/64/BsPa2QtKsY6_LULMdLl-THnDv58AVac-Mbe_iigBB/2/.
19. Mendelson, E.: Introduction to Mathematical Logic. 3. edn. Wadsworth and Brooks (1987)
20. Knuth, D.: The TeXbook. Addison Wesley (1983)
21. Grue, K.: Logiweb. In Kamareddine, F., ed.: Mathematical Knowledge Management Symposium 2003. Volume 93 of Electronic Notes in Theoretical Computer Science., Elsevier (2004) 70–101

Interfacing with the Numerical Homotopy Algorithms in *PHCpack*[⋆]

Anton Leykin[1] and Jan Verschelde[2]

Department of Mathematics, Statistics, and Computer Science
University of Illinois at Chicago, 851 South Morgan (M/C 249)
Chicago, IL 60607-7045, USA
[1]`leykin@math.uic.edu`
`http://www.math.uic.edu/~leykin`
[2]`jan@math.uic.edu`
`http://www.math.uic.edu/~jan`

Abstract. *PHCpack* implements numerical algorithms for solving polynomial systems using homotopy continuation methods. In this paper we describe two types of interfaces to *PHCpack*. The first interface *PHCmaple* originally follows OpenXM, in the sense that the program (in our case Maple) that uses *PHCpack* needs only the executable version `phc` built by the package *PHCpack*. Following the recent development of *PHCpack*, *PHCmaple* has been extended with functions that deal with singular polynomial systems, in particular, the deflation procedures that guarantee the ability to refine approximations to an isolated solution even if it is multiple. The second interface to *PHCpack* was developed in conjunction with MPI (Message Passing Interface), needed to run the path trackers on parallel machines. This interface gives access to the functionality of *PHCpack* as a conventional software library.

1 Introduction

In various fields of science and engineering one must solve polynomial systems, see for example [14], or the case studies in [22, Chapter 9], or [24]. Computer algebra packages like Maple have a convenient worksheet interface which allows to document very precisely the derivation of the polynomial systems using the language of the application area. We designed *PHCmaple* [5] (based on a small Maple procedure in [20], applying our experience with OpenXM [11]) to give a Maple user access to the functionality of a numerical polynomial system solver `phc` (**p**olynomial **h**omotopy **c**ontinuation), a program built by *PHCpack* [25].

Besides a carefully documented problem formulation, the user will need to analyze and interpret the results returned by the solver. The visualization capabilities of a computer algebra system, combined with high level facilities to manipulate and export data in various formats, extend the usefulness of the solver. So the interaction with computer algebra is not only seen as a natural, but as a vital part of the solving process.

[⋆] This material is based upon work supported by the National Science Foundation under Grant No. 0134611 and Grant No. 0410036. *Date:* 22 June 2006.

© Springer-Verlag Berlin Heidelberg 2006

The interaction between computer algebra systems and numerical solvers is one practical side of symbolic-numeric computation, concerned with the implementation of hybrid methods [3]. Examples of hybrid methods are the algorithms developed in the fields of numerical polynomial algebra [23] and numerical algebraic geometry [21] [22]. The algorithms at the heart of numerical algebraic geometry are homotopy continuation methods (see e.g. [1], [10], or [14]), which can be seen as the combination of two methods. Homotopy methods embed the system to be solved in a suitable family of systems, connecting the given problem with a system which is trivial or at least easier to solve. Once this family (called the homotopy) is created, numerical path following methods (also known as continuation methods) track solution paths starting at solutions of the easier problem leading to the solutions of the given problem. As the number crunching is usually all done in hardware floating point arithmetic, the performance of the numerical solver depends critically on the ability of the symbolic homotopy method to capture the structure of the given problem.

Originally designed [25] to offer a wide variety of homotopy algorithms, *PHCpack* has grown into a platform for numerical algebraic geometry, most notably extended with features [20] to deal with positive dimensional solution sets of polynomial systems, implementing a so-called numerical irreducible decomposition [19]. While the *pack* in its name suggests *PHCpack* to be in the glorious tradition of highly successful software like LAPACK [2], its main target is the standalone executable program `phc` which currently is available on a wide range of platforms: PCs running Linux and Windows, computers running MacOS X, SUN workstations running Solaris and IBM machines running AIX. The recent "parallel PHCpack" project is described in [7].

The Maple interface *PHCmaple* was used in [13]. In [16], Maple and *PHCpack* were combined to develop a prototype implementation of new algorithms in numerical jet geometry. In addition to problem formulation and result analysis, this algorithm prototyping is our third motivation for interfaces like *PHCmaple*.

In this paper we present an overview of *PHCmaple*, we refer to [5] for details on its original design and to [9] for the added interface to the new deflation algorithms [8] to recondition multiple isolated roots of polynomial systems. While *PHCmaple* is a user oriented interface, in this paper we document an alternative interface, developed for programming purposes, in particular for use with MPI [17], for tracking solution paths on parallel machines.

The programmer interface to *PHCpack* is characterized by two important features inspired by the *PHCmaple* user interface. Like *PHCmaple* relies only on the executable `phc`, the programmer interface is concentrated in one single routine. Moreover, as the user of *PHCmaple* enjoys the existing data manipulation facilities of Maple, the internal data structures of *PHCpack* for representing polynomials and solutions to systems, are available to the user of the programmer interface so the programmer calling on *PHCpack* should not define similar data structures. In this sense, the programmer interface to *PHCpack* resembles very much a conventional software library.

2 The Evolution of the Interfaces to *PHCpack*

In this section we briefly describe the chronological evolution of the interfaces to the capabilities in *PHCpack*.

1. **OpenXM calls the blackbox solver.** The first interface is still available via OpenXM (Open message eXchange protocol for Mathematics [11], see also [12] and [15]) and only needs an executable file of the program. Just as one calls the blackbox solver of *PHCpack* as `phc -b input output`, a simple system call achieves the same effect.
2. **A simple Maple procedure calls the blackbox solver.** On [20, page 114], a simple Maple 7 procedure (less than 20 lines long) applies the experience of the first interface, calling `phc -b`.
3. **A functional C interface to the Ada routines in *PHCpack*.** To process the output of the Pieri homotopies in *PHCpack* to compute feedback laws to control a linear system, a dedicated C interface was written, used in [26] and described in an online appendix (available at the second author's web site). The main C program calls the Ada routines in *PHCpack* which then call another C function to process the output.
4. ***PHCmaple* gives access to the tools of `phc`.** The main executable program of *PHCpack* can be used as a blackbox or as a toolbox, calling the program with the appropriate options and selecting the desired actions from the menu. Via input redirections, it also just takes the executable version to gain access to the tools offered by *PHCpack*. In [5], we presented *PHCmaple*, a Maple interface to *PHCpack*.
5. **Using *PHCpack* as a state machine.** The dedicated C interface was not adequate for the parallel implementation of the path tracking routines in *PHCpack*. Therefore, a new interface was developed for use in [27], [4] [6] and [28], which describe the progress by parallel *PHCpack* [7].
   The Ada function `use_c2phc`

   ```
   function use_c2phc ( job : integer;
                        a : C_intarrs.Pointer;
                        b : C_intarrs.Pointer;
                        c : C_dblarrs.Pointer ) return integer;
   ```

   is available to the C programmer as

   ```
   extern void adainit( void );
   extern int _ada_use_c2phc (int job, int *a, int *b, double *c);
   extern void adafinal( void );
   ```

   The first parameter `job` specifies the action requested from `phc`. The meaning of the other parameters `a`, `b`, and `c` depends on the `job`. We will describe this interface in greater detail below.

The chronological evolution pictures two distinct trends in interfacing with *PHCpack*: (1) using the executable originated with OpenXM [11]; and (2) calling the compiled code, motivated by the use of MPI [17].

3 Overview of *PHCmaple*

PHCmaple is available for download at `www.math.uic.edu/~leykin/PHCmaple/` and works with Maple version 8 or higher. At the moment the software is developed and tested only on the Windows distribution of Maple.

The goal of *PHCmaple* is to provide computer algebra users with a convenient interface to the

– blackbox solver of `phc`;
– homotopy path tracking facilities;
– deflation procedure;
– routines that create and manipulate witness sets for positive-dimensional components;
– factorization/decomposition capabilities of `phc`.

Using the following procedures one can deal with *isolated solutions* of square systems (with a finite number of solutions): to run the black-box solver execute **solve**, which returns approximations to all complex isolated roots of a square system; refines the solutions to any specified precision with **refine**, which also provides a way to set certain parameters in order to fine-tune the solver; **track** a subset of the solutions set of the start system to the corresponding solutions of the target system and visualize the results with **drawPaths**; for singular isolated solutions one might consider applying **deflationStep**, the implementation of the first-order *deflation* procedure [8], which given a polynomial system and an approximation to one of its multiple isolated solutions produces a new system of equations that has the same solution, but with lower multiplicity. See [9] for more details about the implementation of the deflation algorithm and examples for its use by *PHCmaple*.

The positive-dimensional solution sets of general polynomial systems can be represented by means of *witness sets*, computing which reduces the problem to the isolated solution case.

The following functions of *PHCmaple* serve this purpose: construct an embedded system with **embed** in assumption that the dimension of its solution set is known; **cascade** runs the so-called *cascade of homotopies* for an embedded system, it computes the list of *witness sets* for the components of the solution set in every dimension; after producing the witness sets **filter** the points in lower-dimensional witness sets belonging to higher-dimensional components; to produce a *numeric irreducible decomposition* of a pure-dimensional solution component **decompose** its witness set; absolute factorization capability for multivariate polynomial is given by **factor**.

The new, recently added **eqnbyeqn** routine launches the equation-by-equation solver [18], which produces the witness sets similarly to **cascade**, though using a different method. These witness sets are returned already filtered.

4 Using *PHCpack* in C Programs

The function `use_c2phc` serves as a gateway to the full functionality of *PHCpack*, concentrated in one single routine. When designing this interface, we viewed

PHCpack as a state machine. Like we input coins into a vending machine, make a selection pushing a button to then collect our selected beverage, the programmer calls `use_c2phc` with the appropriate `job` number to read in a polynomial system, select the type of homotopy and various other options to then finally activate the path trackers which will compute the solutions and write to the output.

The `use_c2phc` was written to get access to the path tracking routines (written in Ada) by a main program in C, which calls the communication primitives of MPI. It was used in a series of papers, starting with [27] (parallel Pieri homotopies), continuing with [6] and [4] (parallel factorization), and most recently used in [28] (parallel polyhedral homotopies). As `use_c2phc` called on various homotopy algorithms in *PHCpack*, the number of different `job` numbers grew and the direct use of the gateway by the main parallel program became too tedious. So we developed an additional layer between the main parallel program and the `use_c2phc` function.

This additional layer forms the programmer interface to *PHCpack*. It consists of a collection of header files (suffix `.h`) offering the programmer meaningful names for the various jobs performed by `use_c2phc`, hiding the precise `job` number in the definition of the function listed in the header files. Details about this evolving interface can be found in the source code distribution of *PHCpack*.

5 Future Developments

In this paper we described a user and a programmer interface to *PHCpack*.

While currently *PHCmaple* still operates by making system calls to the standalone program `phc`, a more efficient interface will apply the `use_c2phc` in a dynamic link library.

Besides adding more functionality to `use_c2phc`, a very useful extension of its argument list will be the addition of two functions to allow the user to define so-called straight line programs to evaluate and differentiate the polynomial systems generated by the homotopy.

The interfaces we described will serve as a model to use *PHCpack* in other computer algebra systems like Axiom or Macaulay 2 for example, as well as in scientific computing systems like Octave or Scilab. As C seems to be the least common denominator language for computer programming, the programmer interface could lead to bindings to other languages or used to build other dedicated interfaces.

References

1. E.L. Allgower and K. Georg. *Introduction to Numerical Continuation Methods*, volume 45 of *Classics in Applied Mathematics*. SIAM, 2003.
2. E. Anderson, Z. Bai, C. Bischof, L.S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammerling, A. McKenny, and D. Sorensen. *LAPACK User's Guide*. SIAM, 3rd edition, 1999. Available online via `http://www.netlig.org/lapack`.

3. R.M. Corless, E. Kaltofen, and S.M. Watt. Hybrid methods. In J. Grabmeier, E. Kaltofen, and V. Weispfenning, editors, *Computer Algebra Handbook*, pages 112–125. Springer–Verlag, 2002.
4. A. Leykin and J. Verschelde. Decomposing solution sets of polynomial systems: a new parallel monodromy breakup algorithm. Accepted for publication in *The International Journal of Computational Science and Engineering.*
5. A. Leykin and J. Verschelde. PHCmaple: A Maple interface to the numerical homotopy algorithms in PHCpack. In Quoc-Nam Tran, editor, *Proceedings of the Tenth International Conference on Applications of Computer Algebra (ACA'2004)*, pages 139–147, 2004.
6. A. Leykin and J. Verschelde. Factoring solution sets of polynomial systems in parallel. In Tor Skeie and Chu-Sing Yang, editors, *Proceedings of the 2005 International Conference on Parallel Processing Workshops. 14-17 June 2005. Oslo, Norway. High Performance Scientific and Engineering Computing*, pages 173–180. IEEE Computer Society, 2005.
7. A. Leykin, J. Verschelde, and Zhuang Y. Parallel homotopy algorithms to solve polynomial systems. Proceedings of ICMS'06, this volume.
8. A. Leykin, J. Verschelde, and A. Zhao. Newton's method with deflation for isolated singularities of polynomial systems. To appear in *Theoretical Computer Science.*
9. A. Leykin, J. Verschelde, and A. Zhao. Evaluation of Jacobian matrices for Newton's method with deflation to approximate isolated singular solutions of polynomial systems. In D. Wang and L. Zhi, editors, *SNC 2005 Proceedings. International Workshop on Symbolic-Numeric Computation. Xi'an, China, July 19-21, 2005*, pages 19–28, 2005.
10. T.Y. Li. Numerical solution of polynomial systems by homotopy continuation methods. In F. Cucker, editor, *Handbook of Numerical Analysis. Volume XI. Special Volume: Foundations of Computational Mathematics*, pages 209–304. North-Holland, 2003.
11. M. Maekawa, M. Noro, K. Ohara, Y. Okutani, N. Takayama, and Tamura Y. Openxm – an open system to integrate mathematical softwares. Available at `http://www.OpenXM.org/`.
12. M. Maekawa, M. Noro, K. Ohara, N. Takayama, and Y. Tamura. The design and implementation of OpenXM-RFC 100 and 101. In K. Shirayanagi and K. Yokoyama, editors, *Computer mathematics. Proceedings of the Fifth Asian Symposium (ASCM 2001) Matsuyama, Japan 26 - 28 September 2001*, volume 9 of *Lecture Notes Series on Computing*, pages 102–111. World Scientific, 2001. Available at `http://www.math.kobe-u.ac.jp/OpenXM/ascm2001/ascm2001/ascm2001.html`.
13. M.M. Maza, G.J. Reid, R. Scott, and W. Wu. On approximate triangular decomposition I. Dimension zero. In D. Wang and L. Zhi, editors, *SNC 2005 Proceedings. International Workshop on Symbolic-Numeric Computation. Xi'an, China, July 19-21, 2005*, pages 19–28, 2005.
14. A. Morgan. *Solving polynomial systems using continuation for engineering and scientific problems.* Prentice-Hall, 1987.
15. M. Noro. A computer algebra system: Risa/Asir. In M. Joswig and N. Takayama, editors, *Algebra, Geometry, and Software Systems*, pages 147–162. Springer–Verlag, 2003.
16. G. Reid, J. Verschelde, A. Wittkopf, and W. Wu. Symbolic-numeric completion of differential systems by homotopy continuation. In M. Kauers, editor, *Proceedings of the 2005 International Symposium on Symbolic and Algebraic Computation (ISSAC'05), July 24-27 2005, Beijing, China*, pages 269–276. ACM, 2005.

17. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI - The Complete Reference Volume 1, The MPI Core.* Massachusetts Institute of Technology, 2nd edition, 1998. Available via `http://www-unix.mcs.anl.gov/mpi/`.
18. A.J. Sommese, J. Verschelde, and C.W. Wampler. Solving polynomial systems equation by equation. Submitted for publication.
19. A.J. Sommese, J. Verschelde, and C.W. Wampler. Numerical decomposition of the solution sets of polynomial systems into irreducible components. *SIAM J. Numer. Anal.*, 38(6):2022–2046, 2001.
20. A.J. Sommese, J. Verschelde, and C.W. Wampler. Numerical irreducible decomposition using PHCpack. In M. Joswig and N. Takayama, editors, *Algebra, Geometry, and Software Systems*, pages 109–130. Springer–Verlag, 2003.
21. A.J. Sommese, J. Verschelde, and C.W. Wampler. Introduction to numerical algebraic geometry. In *Solving Polynomial Equations. Foundations, Algorithms and Applications*, volume 14 of *Algorithms and Computation in Mathematics*, pages 301–337. Springer–Verlag, 2005.
22. A.J. Sommese and C.W. Wampler. *The Numerical solution of systems of polynomials arising in engineering and science.* World Scientific Press, Singapore, 2005.
23. H.J. Stetter. *Numerical Polynomial Algebra.* SIAM, 2004.
24. B. Sturmfels. *Solving Systems of Polynomial Equations.* Number 97 in CBMS Regional Conference Series in Mathematics. AMS, 2002.
25. J. Verschelde. Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Trans. Math. Softw.*, 25(2):251–276, 1999. Software available at `http://www.math.uic.edu/~jan`.
26. J. Verschelde and Y. Wang. Computing dynamic output feedback laws. *IEEE Transactions on Automatic Control*, 49(8):1393–1397, 2004.
27. J. Verschelde and Y. Wang. Computing feedback laws for linear systems with a parallel Pieri homotopy. In Y. Yang, editor, *Proceedings of the 2004 International Conference on Parallel Processing Workshops, 15-18 August 2004, Montreal, Quebec, Canada. High Performance Scientific and Engineering Computing*, pages 222–229. IEEE Computer Society, 2004.
28. J. Verschelde and Y. Zhuang. Parallel implementation of the polyhedral homotopy method. Accepted for publication in the proceedings of *The 8th Workshop on High Performance Scientific and Engineering Computing (HPSEC-06)*, Columbus, Ohio, USA, August 18, 2006.

Computational Construction of a Maximum Equilateral Triangle Inscribed in an Origami

Tetsuo Ida, Hidekazu Takahashi, Mircea Marin,
Fadoua Ghourabi, and Asem Kasem

Department of Computer Science
University of Tsukuba, Tsukuba 305-8573, Japan
`{ida, hidekazu, mmarin, ghourabi, kasem}@score.cs.tsukuba.ac.jp`

Abstract. We present an origami construction of a maximum equilateral triangle inscribed in an origami, and an automated proof of the correctness of the construction. The construction and the correctness proof are achieved by a computational origami system called *Eos* (E-origami system). In the construction we apply the techniques of geometrical constraint solving, and in the automated proof we apply Gröbner bases theory and the cylindrical algebraic decomposition method. The cylindrical algebraic decomposition is indispensable to the automated proof of the maximality since the specification of this property involves the notion of inequalities. The interplay of construction and proof by Gröbner bases method and the cylindrical algebraic decomposition supported by *Eos* is the feature of our work.

1 Introduction

Origami is a traditional Japanese art of paper folding, which is now enjoyed by many people all over the world. Computational origami is a scientific discipline to study computational aspects of origami, and aims to complement and even goes beyond traditional paper folding by hands [4]. We are aiming at overcoming the limitations of traditional origami construction, and, with the use of computers, at providing capabilities that support geometrical reasoning about origami.

We study the problem of computational origami construction of an equilateral triangle that is maximally inscribed in an origami square. Although the problem is easy, we use it as an example to appeal the importance of computational origami and of the effectiveness of our approach, since it reveals the essence of computational origami construction and of the automated correctness proof of the construction thereafter. In the construction we apply the techniques of geometrical constraint solving, and in the automated proof we apply Gröbner bases theory [2] and the cylindrical algebraic decomposition method [1].

We reported earlier some examples of computational origami constructions [9,8], such as the construction of Morley's triangle by origami with correctness proof. What is new in this paper is the application of cylindrical algebraic decomposition (cad) to prove the correctness of our construction. The use of cad has lead to further improvement of our computational origami system called *Eos*.

© Springer-Verlag Berlin Heidelberg 2006

The paper is organized as follows. In section 2, we briefly explain our computational origami environment as we will use it for the study of origami construction of a maximum equilateral triangle. In section 3, we will present a method for the construction using *Eos*. In section 4, we will give the proof of the correctness of the construction. We will show two methods of proving the correctness. In section 5, we draw some conclusion and point out future research.

2 E-Origami System *Eos*

Before we describe the construction of the equilateral triangle, we briefly explain the main features of *Eos*. *Eos* is a collection of *Mathematica* programs, specialized in origami processing, that have capabilities of basic geometrical computing tailored to origami, of constraint solving both by numeric and symbolic methods, of theorem proving, of visualizations of origami and of web interfaces. It performs origami construction that a human would do by hand with a piece of paper. Moreover, it performs automated proving of geometrical properties of the constructed origami, which would require much geometric intuition in addition to complicated and tedious algebraic manipulation.

2.1 Origami Construction by *Eos*

Eos implements Huzita's origami axioms [7], which describe the basic folding operations. Huzita's axiom set is more powerful than the straight-edge and compass method in Euclidean plane geometry [5,6]. Using Huzita's axiom set, by origami we can construct a trisector of an angle, whereas by the straight-edge and compass we can not. Huzita's axiom set is described by the following statements about the fundamental origami folding operations.

- (**O1**) Given two points, we can make a fold along the fold line that passes through them.
- (**O2**) Given two points, we can make a fold to bring one of the points onto the other.
- (**O3**) Given two lines, we can make a fold to superpose the two lines.
- (**O4**) Given a point P and a line m, we can make a fold along the fold line that is perpendicular to m and passes through P.
- (**O5**) Given two points P and Q and a line m, we can make a fold to superpose P and m along the fold line that passes through Q.
- (**O6**) Given two points P and Q and two lines m and n, we can make a fold to superpose P and m, and Q and n, simultaneously.

These axioms are formalized in a properly chosen sub-language of first-order logic, with function symbols for the geometric constructs and predicate symbols for the geometric properties mentioned there. For example, axiom (O5) is formalized as follows: $\exists\{k \in \texttt{Line}\}\ \texttt{OnLine}[Q, k] \wedge \texttt{OnLine}[\texttt{SymmetricPoint}[P, k], m]$. The atomic formula $\texttt{OnLine}[Q, k]$ specifies that point Q is on line k, and the term $\texttt{SymmetricPoint}[P, k]$ denotes the symmetric point of P with respect to line k. The algebraic meaning of each predicate is specified by a set of equations. For example, the meaning of `OnLine`$[Q, k]$ is given by $a\ x_1 + b\ y_1 + c = 0$, where Q is positioned at (x_1, y_1) and k is defined by the equation $a\ x + b\ y + c = 0$.

Origami constructions proceed stepwise, where each step indicates a fold operation that satisfies one of axioms (O1)–(O6). *Eos* provides a function `Fold` to realize the fold steps. In addition, every call of `Fold` has two effects: (1) to visualize the result of the fold step by computing the fold line and performing the fold, and (2) to record the geometric constraints that characterize the fold step. In this way, *Eos* provides interactive access to (1) a view of the origami constructed so far, and (2) the collection of geometric constraints that describe the origami constructed so far.

2.2 Theorem Proving of Origami Construction

After origami construction with *Eos*, we can proceed to prove geometrical properties of the construction in the following steps.

Premise generation: Extract the geometrical properties of the construction and transform them into polynomial equalities and inequalities which form the premise of the theorem to be proved.

Conclusion formulation: Represent the conclusion in polynomial equalities and/or inequalities.

Proof generation: Give the polynomial equalities and inequalities to a theorem prover such as *Theorema* when the theorem is described in polynomial equalities, or to cylindrical algebraic decomposition programs when the theorem is described partly in inequalities, and obtain the proof result.

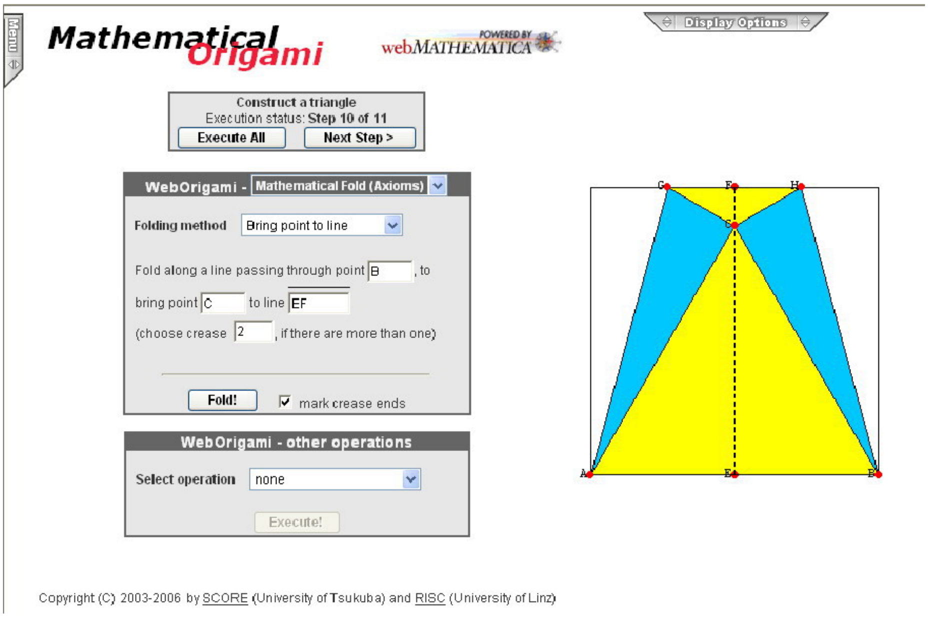
The most important step is the formalization of the geometrical properties to be proved. Note that although the conclusion formulation step is in the third computation step, we have to think about the conclusion at first. Also note that the steps except for the conclusion formulation step are automated.

3 Construction of an Equilateral Triangle

In this section we illustrate origami construction of an equilateral triangle by *Eos*. A straightforward method to construct an equilateral triangle is to use one of the sides of the origami as a side of an equilateral triangle. The following snapshot of the origami construction using the *Eos* website illustrates the construction.

`http://weborigami.score.cs.tsukuba.ac.jp`

However, we immediately recognize that this is not a maximum triangle inscribed in the origami. In the following, we will show another origami construction that creates a maximum equilateral triangle. Since we want to make the side of a triangle as long as possible, it is natural to take one of the corners of the triangle to be also a corner of the origami. We expect that the other two vertices are also on the sides of the origami.


Mathematical Origami

webMATHEMATICA POWERED BY

Display Options

Construct a triangle
Execution status: Step 10 of 11
Execute All | Next Step >

WebOrigami - Mathematical Fold (Axioms)

Folding method: Bring point to line

Fold along a line passing through point B, to bring point C to line EF (choose crease 2, if there are more than one)

Fold! | ☑ mark crease ends

WebOrigami - other operations

Select operation: none

Execute!

Copyright (C) 2003-2006 by SCORE (University of Tsukuba) and RISC (University of Linz)

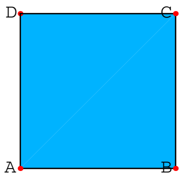
Fig. 1. A webpage of *WebOrigami*

We will illustrate a step-wise construction of a maximum equilateral triangle via a *Mathematica* notebook interface rather than *Eos* web interface, since in the notebook we can use all the capabilities of *Eos* and *Mathematica*.

3.1 Construction

We start our origami construction by declaring the origami to be used. In the following example, we will use a unit square paper with four vertices marked as A, B, C and D.

```
BeginOrigami[{1, MarkPoints → {"A","B","C","D"},
                FigureCaption→"Step "}]
```

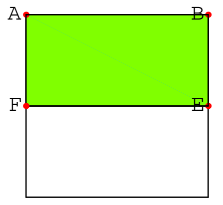


Step 1

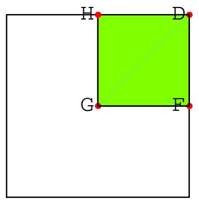
The calls of function `Fold` and others with various parameters will be self-explanatory.

We make a fold to bring point A onto D (application of (O2)), and then make a fold to bring point A onto B (application of (O2)). Function `UnfoldAll` completely unfolds the folded origami into the original origami. These steps are illustrated below.

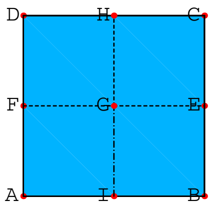
```
Fold[A,D]; Fold[A,B];UnfoldAll[];
```



Step 2



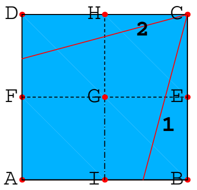
Step 3



Step 4

After these preparatory steps, we make a fold to superpose B and line $\overline{\mathrm{HI}}$ along the fold line that passes through C (application of (O5))[1]. In this case we have two possible folds. It can be shown that to find the fold line in the use of axiom (O5) is a constraint solving problem formulated with a set of algebraic equalities of the second order. Hence, we have at most two choices. *Eos* will interact with the user to ask for the choice, as seen below.

$\texttt{Fold}[\texttt{B},\texttt{HI},\texttt{Through}\to\texttt{C},\texttt{MarkCrease}\to\{\texttt{AB}\}];$

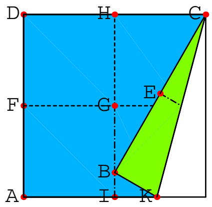


Step 5

There are two ways to proceed. One is to specify the line number shown in the origami, and the other is to give the constraint that uniquely determines the choice. In the above case, both responses will do. However, there is a difference when it comes to proving later. By specifying the constraint, *Eos* not only selects one of the numeric solutions using the constraint but also saves the constraint symbolically, which can be used to prove geometrical properties later. Since we are interested in geometric theorem proving we specify the constraint as follows.

$\texttt{Fold}[\texttt{B},\texttt{HI},\texttt{Through}\to\texttt{C},\texttt{MarkCrease}\to\{\texttt{AB}\},\texttt{Constraint}\to\texttt{OnSegment}[\texttt{B},\texttt{HI}]];$

[1] XY denotes a segment between point X and point Y and $\overline{\mathrm{XY}}$ denotes the line that extends segment XY, i.e the line that pass through X and Y.

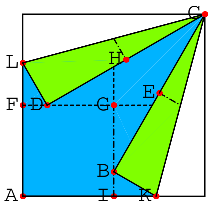


Step 6

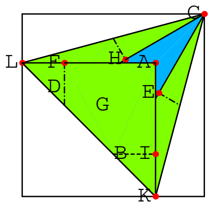
In the above we specify the constraint that point B is on the segment HI. This constraint determines the choice of the fold line 1.

Similarly we will construct point L and , finally, apply (O1) to fold the origami along line $\overline{\text{KL}}$.

$\texttt{Fold}[\text{D}, \text{FG}, \texttt{Through} \rightarrow \text{C}, \texttt{Constraint} \rightarrow \texttt{OnSegment}[\text{D}, \text{FG}]];$
$\texttt{Fold}[\text{A}, \texttt{Along} \rightarrow \text{KL}];$

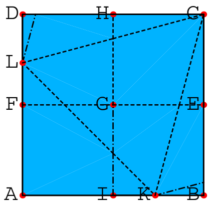


Step 8



Step 9

The triangle ΔCLK appears to be equilateral, and at this point we conjecture that this is the case. The following complete development of the origami, which is obtained by the call of `UnfoldAll`, will be helpful for further study of the geometrical properties.



Step 10

Fig. 2. Complete development of the folded origami

The figure is called *ori-zu*[2]. It shows the creases made by both valley folds and mountain folds, and the points, created in the whole construction.

[2] *ori* means "fold" and *zu* means "figure".

4 Proof of the Correctness of the Construction

We will now prove the following theorem:

Theorem 1 (Maximum Equilateral Triangle Origami Theorem). *Given an origami $\square ABCD$, the origami construction in subsection 3.1 has the following properties:*

(a) ΔCLK is equilateral, and furthermore
(b) ΔCLK is a maximum equilateral triangle inscribed in the origami.

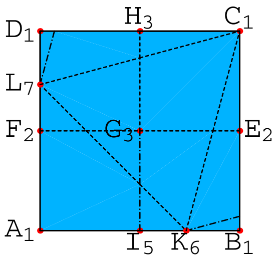
The automated proof of this theorem is given in the following two subsections. We will follow the proof procedure as outlined in subsection 2.2, interacting with *Eos*.

4.1 Proof that ΔCLK Is Equilateral

Automated Proof by *Eos* Interacting with *Theorema*. For proving, we switch the context from construction to proof by calling function `BeginProof`.

```
BeginProof[];
```

The ori-zu shown below contains more information than the complete development of the folded origami of Fig. 2. In the figure, X_n indicates that point X is on the origami whose id is n. Origami id is not shown in the figures, but can be displayed by setting the switch on.



We first consider part (a) of the theorem. The proof is based on the Gröbner bases method [2].

Premise Generation. For theorem proving, we transform the symbolic constraints accumulated during the construction of origami into algebraic form. This is achieved by deciding the coordinate system and then translating the symbolic representation of the geometrical properties into polynomials.

The generated premise sometimes needs to be strengthened in order to eliminate degenerate cases and other unwanted geometrical configurations. In our case, we need the condition that K_6, C_1 and L_7 are not collinear. This is related to the fact that at steps 6 and 8 we have two folding possibilities. Axiom (O5) describes a geometric constraint that is expressed in second degree polynomials.

This gives rise to 2 sets of solutions. Although at steps 6 and 8, we have chosen one of the two possibilities, in this premise generation phase we do not specify these committed choices in order to allow possibly more general statements about the intended geometrical properties. In our case, we would expect that in four possible cases we could construct equilateral triangles. However, this is not the case as shown in the following figure.

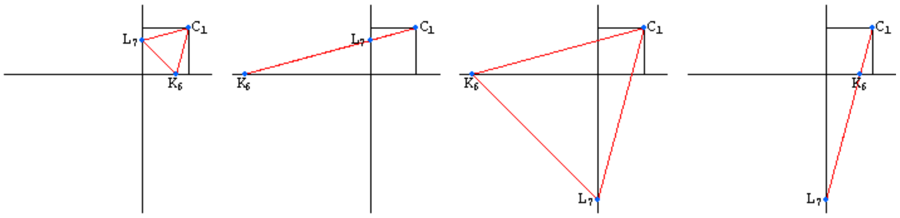


Fig. 3. Possible cases of the construction

As seen in the figure, only the first and third cases make the equilateral triangles. The second and the fourth cases can be eliminated by specifying that $\neg$ Collinear[K_6, C_1, L_7].

We can translate the constraints related to points K_6, C_1 and L_7 from symbolic to algebraic form as follows.

$$\texttt{premisePoly} = \texttt{ToAlgebraic}[\{K_6, C_1, L_7\}, \texttt{InitialShape} \to \{\texttt{Point}[0,0], 1\}, \texttt{Constraints} \to \{\neg\, \texttt{Collinear}[K_6, C_1, L_7]\}]$$

$-1 + a2^2 + b2^2, c2 + a2x2 + b2y2, -1 + a3^2 + b3^2, a3 + b3 + c3, -1 + a4^2 + b4^2, a4 + b4 + c4, -1 + a1^2 + b1^2, c1 + a1x3 + b1y3, y4, c3 + a3x4 + b3y4, x5, c4 + a4x5 + b4y5, a1, \frac{b1}{2} + c1, b1(-1 + x1) + (-1)a1y1, c1 + \frac{1}{2}a1(1 + x1) + \frac{b1y1}{2}, b3(-1 + x2) + (-1)a3y2, c3 + \frac{1}{2}a3(1 + x2) + \frac{b3y2}{2}, -b2x1 + a2(-1 + y1), c2 + \frac{a2x1}{2} + \frac{1}{2}b2(1 + y1), b4x3 + a4(1 + (-1)y3), c4 + \frac{a4x3}{2} + \frac{1}{2}b4(1 + y3), -1 + (x5 + (-1)y5 + x4(-1 + y5) + (1 + (-1)x5)y4)\xi 1$

In our representation of a line $a\ x + b\ y + c = 0$, we need to make sure that both a and b are not equal to zero simultaneously, namely $a^2 + b^2 \neq 0$. And without loss of generality, we can add the constraint that $a^2 + b^2 = 1$.

Note also that $\neg$ Collinear[K_6, $C_1 L_7$] is translated to equalities using Rabinowich trick. This can be further transformed to the equivalent logic form (output not shown).

```
premise = ToLogic[premisePoly];
```

Conclusion Formation. The next step is to transform the conclusion that ΔCLK is an equilateral triangle into algebraic form. An equilateral triangle is characterized by the property that all the three sides are equal in length. Hence we have the following specification.

```
conclusion = ToLogic[{ Distance [K6, L7]^2 − Distance [C1, L7]^2 ,
                       Distance [K6, L7]^2 − Distance [C1, K6]^2}]
```

$-(1-\mathrm{x4})^2+(\mathrm{x4}-\mathrm{x5})^2-(1-\mathrm{y4})^2+(\mathrm{y4}-\mathrm{y5})^2 == 0 \wedge$
$(\mathrm{x4}-\mathrm{x5})^2+(\mathrm{y4}-\mathrm{y5})^2-(1-\mathrm{x5})^2-(1-\mathrm{y5})^2 == 0$

Note that we use two kinds of equality symbols; `==` as the output of Mathematica, and = as in the ordinary mathematics.

Proof Generation. *Eos* can prove geometric properties of origamis by accessing to the theorem prover *Theorema* [3]. Since both *Eos* and *Theorema* are implemented in *Mathematica*, accessing to *Theorema* is straightforward. First we call

```
TheoremaFormula["Equilateral Triangle Th",
    Variables[premisePoly], premise ⇒ conclusion, "(1)"];
```

This method call encodes the proposition `premise ⇒ conclusion` into a *Theorema* formula, labeled `"Equilateral Triangle Th"`. The variables occurring in the formula are universally quantified. Next, we invoke *Theorema* to prove it:

```
Prove[Formula["Equilateral Triangle Th"], using → {},
              by → GroebnerBasesProver]//Timing
```

```
{3.032 Second, −ProofObject−}
```

This call will yield a human-readable proof with proof text structured as nested cells of *Mathematica*.

Proof by *Eos* Using the Cylindrical Algebraic Decomposition. The success of our proof by Gröbner bases method relies on the crucial observation that we must add the condition ¬ `Collenear`$[\mathrm{K}_6, \mathrm{C}_1, \mathrm{L}_7]$. In the following we will show an alternative proof based on the cylindrical algebraic decomposition. We recall that in order to specify one of the two choices of the fold lines we specified the constraint that `OnSegment`[B, HI] at step 6 and `OnSegment`[D, FG] at step 8. These additional constraints are not included in the previous proof since they give rise to inequalities, which are not handled by Gröbner bases method.

We now add these constraints by setting the option `AddConstraint` to `True` in the call of function `ToAlgebraic`.

```
premise = ToLogic[ToAlgebraic[{K6, C1, L7},
InitialSahpe → {Point[0, 0], 1}, AddConstraint → True ]];
```

The outcome is that `premise` contains the following additional inequalities:

$(-\mathrm{x3}+\mathrm{x4})(-\mathrm{x3}+\mathrm{x5}) \leq 0 \wedge (\mathrm{y4}+(-1)\mathrm{y3})(\mathrm{y5}+(-1)\mathrm{y3}) \leq 0 \wedge$
$(\mathrm{x6}+(-1)\mathrm{x2})(\mathrm{x7}+(-1)\mathrm{x2}) \leq 0 \wedge (\mathrm{y6}+(-1)\mathrm{y2})(\mathrm{y7}+(-1)\mathrm{y2}) \leq 0$

We next perform the cylindrical algebraic decomposition [1] of the above formula by calling the *Mathematica* function `CylindricalDecomposition`. This will transform the above formula to a fully solved form.

```
sPremise = CylindricalDecomposition[premise,Variables[premise]];
```

Now it is very easy to prove the conclusion by substituting those values in the conclusion formula. We can also use the previous code and call *Theorema* to prove the theorem as follows.

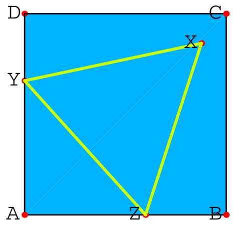
```
proposition = sPremise ⇒ conclusion
TheoremaFormula["Maximum Equilateral Triangle Th",
vars, proposition, "(2)"]; 3
Prove[Formula["Equilateral Triangle Th"], using → {},
                                  by → GroebnerBasesProver]//Timing
{3.094 Second, −ProofObject−}
```

We should note in passing the following properties. Let |AB| = |BC| = |CD| = |AD| = 1. Let $h = |\mathrm{BH}|$ and $x = |\mathrm{AK}|$ at step 7. Then we have the set of equations eqns $= \left\{\frac{1}{4} + h^2 == 1, (1-h)^2 + \left(x - \frac{1}{2}\right)^2 == (1-x)^2\right\}$, whose solutions are given by $\left\{\left\{x \to -1 - \sqrt{3}, h \to -\frac{\sqrt{3}}{2}\right\}, \left\{x \to -1 + \sqrt{3}, h \to \frac{\sqrt{3}}{2}\right\}\right\}$.

The first solution is irrelevant because $x = |\mathrm{AK}|$ has to be non-negative. From the second solution we learn that point K is at ($\sqrt{3}$ - 1, 0). Therefore, the area of the triangle is $-3 + 2\sqrt{3}$. This property is used to establish the proof of the part (b) of Theorem 1.

4.2 Proof That *Δ*CLK Is the Maximum Equilateral Triangle Inscribed in the Origami

To prove part (b) of Theorem 1, we proceed as follows. We consider an equilateral triangle *Δ*XYZ as depicted below, where we have A(0,0), C(1,1), X(x, y), Y($0, b$), Z($c, 0$) with $b, c \in (0, 1)$.



*Δ*XYZ is equilateral, and therefore $|\mathrm{YZ}|^2 = |\mathrm{YX}|^2 = |\mathrm{XZ}|^2$. This geometric constraint is expressed algebraically as: $(y-b)^2 + x^2 = b^2 + c^2 = (x-c)^2 + y^2$. We can solve these algebraic constraints by calling

Solve $\left[\left\{\left((y-b)^2 + x^2\right) == b^2 + c^2 == (x-c)^2 + y^2\right\}, \{x, y\}\right]$
$\left\{\left\{y \to \frac{b-\sqrt{3}c}{2}, x \to \frac{-\sqrt{3}b+c}{2}\right\}, \left\{y \to \frac{b+\sqrt{3}c}{2}, x \to \frac{\sqrt{3}b+c}{2}\right\}\right\}$

[3] `vars` is the set of variables occurring in `proposition`.

This call computes x and y as functions of b and c. We know that the area of an equilateral triangle with edge of length l is $\sqrt{3}l^2/4$. Since $|\mathrm{YZ}| = \sqrt{a^2+b^2}$, we learn that the area of ΔXYZ is $S = \sqrt{3}(b^2+c^2)/4$. Our problem is to compute the values of b and c for which S is maximal.

Let S_0 be the area of the equilateral triangle ΔCLK constructed with *Eos*. Since $S_0 = -3 + 2\sqrt{3}$, we know that the maximum value of S is greater than or equal to S_0. We can compute the values of b, c for which $S \geq S_0$ by using CAD:

1. If $y = (b+\sqrt{3}c)/2$ and $x = (\sqrt{3}b+c)/2$ then the call

   `CylindricalDecomposition`$[\{S \geq S_0, 0 \leq x \leq 1, 0 \leq y \leq 1, 0 < b < 1,$
   $0 < c < 1,\ y{=}{=}\frac{1}{2}\left(b+\sqrt{3}c\right), x{=}{=}\frac{1}{2}\left(\sqrt{3}b+c\right)\}, \{x, y, b, c\}]$

   yields the cylindrical decomposition

   $y = 1 \wedge x = 1 \wedge c = -1+\sqrt{3} \wedge b = -1+\sqrt{3}$.
2. If $y = (b-\sqrt{3}c)/2$ and $x = (-\sqrt{3}b+c)/2$ then the call

   `CylindricalDecomposition`$[\{S \geq S_0, 0 \leq x \leq 1, 0 \leq y \leq 1, 0 < b < 1,$
   $0 < c < 1,\ y{=}{=}\frac{1}{2}\left(b-\sqrt{3}c\right), x{=}{=}\frac{1}{2}\left(-\sqrt{3}b+c\right)\}, \{x, y, b, c\}]$

   yields `False`.

We conclude that the maximum area of ΔXYZ is reached only when $x = 1$, $c = -1+\sqrt{3}$ and $b = -1+\sqrt{3}$. In this case, we have X=C, Y=L and Z=K. Therefore, we can conclude that Theorem 1.(b) holds.

5 Conclusion

We have shown the origami construction of an equilateral triangle and the automated proof of the property that the constructed triangle is equilateral, maximally inscribed in the origami by the computational origami environment *Eos*. It not only simulates origami folds, but also computes and proves geometric properties of the construction. *Eos* keeps track of the geometrical properties of all points and lines during the construction. From those properties, polynomial algebraic constraints are generated, which then are supplied to theorem provers.

Recently, we added several improvements in the proof part. By specifying additional constraints using equalities and inequalities during the construction, the system can perform more automated deduction. Inequality constraints require more powerful solving methods such as cylindrical algebraic decomposition.

Our experience with *Eos* shows that the number of polynomials grows very rapidly as the number of origami construction steps grows. Even a small number of fold steps (for example, 30 steps) may generate over 100 polynomials. Handling constraints of this size is a challenging task for most of the geometric solvers and provers. We can easily generate sets of constraints which make a Gröbner bases prover run out of computing resources. These limitations can be overcome by adopting a computational framework with access to networked resources.

Acknowledgements. This research is supported in part by the JSPS Grants-in-Aid for Scientific Research No. 17300004 and No. 17700025, and by MEXT Grant-in-Aid for Exploratory Research NO. 17650003.

References

1. D. S. Arnon, G. E. Collins, and S. McCallum. Cylindrical algebraic decomposition I: The basic algorithm. *SIAM Journal on Computing*, 13(4):865–877, 1984.
2. B. Buchberger. Ein algorithmisches Kriterium für die Lösbarkeit eines algebraischen Gleichungssystems. *Aequationes mathematicae*, 4(3):374–383, 1970.
3. B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Văsaru, and W. Windsteiger. The Theorema Project: A Progress Report. In M. Kerber and M. Kohlhase, editors, *Symbolic Computation and Automated Reasoning: The Calculemus-2000 Symposium*, pages 98–113, St. Andrews, Scotland, August 6-7 2000.
4. E. D. Demaine and M. L.Demaine. Recent results in computational origami. In Thomas Hull, editor, *Origami^3: Third International Meeting of Origami Science, Mathematics and Education*, pages 3–16, Natick, Massachusetts, 2002. A K Peters, Ltd.
5. R. Geretschläger. *Geometric Constructions in Origami.* Morikita Publishing Co., 2002. In Japanese, translation by Hidetoshi Fukagawa.
6. T. Hull. Origami and geometric constructions. 2005. http://www.merrimack.edu/˜thull/omfiles/geoconst.html.
7. H. Huzita. Axiomatic Development of Origami Geometry. In H. Huzita ed, editor, *Proceedings of the First International Meeting of Origami Science and Technology*, pages 143–158.
8. T. Ida, D. Ţepeneu, B. Buchberger, and J. Robu. Proving and Constraint Solving in Computational Origami. In *Proceeding of the 7th International Symposium on Artificial Intelligence and Symbolic Computation (AISC 2004)*, volume 3249 of *Lecture Notes in Artificial Intelligence*, pages 132 – 142, 2004.
9. T. Ida, H. Takahashi, D. Ţepeneu, and M. Marin. Morley's Theorem Revisited through Computational Origami. In *Proceedings of the 7th International Mathematica Symposium (IMS 2005)*, 2005.

A System for Interfacing MATLAB with External Software Geared Toward Automatic Differentiation

H. Martin Bücker[1], Atya Elsheikh[2], and Andre Vehreschild[1]

[1] RWTH Aachen University, Institute for Scientific Computing,
Seffenter Weg 23, D-52074 Aachen, Germany
[2] University of Siegen, Department of Simulation,
Faculty 11/12, 57068 Siegen, Germany

Abstract. MATLAB is commonly considered to be an attractive, high-productivity programming environment by many computational scientists and engineers. So-called MEX-files are dynamically linked subroutines produced from, say, C or Fortran source code that, when compiled, can be run directly from within MATLAB as if they were MATLAB built-in functions. When applying automatic differentiation to a MATLAB program that calls external software via MEX-files, code is mechanically generated for the MATLAB part and for the external part in two separate phases. These resulting code fragments need to be put together via new MEX-files. This work introduces a novel software tool called automatic differentiation mexfunction generator that automatically generates MEX interface functions for gluing these automatically generated code fragments.

1 Introduction

The field of computational science and engineering is rapidly increasing, bringing together applied mathematics and computer science with a large number of different scientific and engineering disciplines. The goal is to better understand phenomena arising from real-world problems by using advanced simulation techniques. The role of mathematical software is to implement these techniques in a reliable, robust, and portable way. MATrix LABoratory (MATLAB)[1] [1] is an extremely successful interactive environment for scientific and technical computing. Its high-level programming language is heavily based on concepts from linear algebra with vectors and matrices constituting the most fundamental data structures. This language design as well as the rich set of powerful functions provided by so-called MATLAB toolboxes facilitate rapid prototyping for tackling complex computational problems with modest human effort.

Rather than excessively concentrating on the efficiency of the interpreted programs, the focus of MATLAB has been on the accuracy and stability of the underlying numerical methods. That is, rather than getting a *fast* answer, the focus is on getting a *correct* answer. Accuracy is also a crucial factor when derivatives

[1] MATLAB is a registered trademark of The Mathworks, Inc.

© Springer-Verlag Berlin Heidelberg 2006

are to be evaluated in areas such as design optimization, data assimilation, or inverse problems. Since numerical differentiation by divided differencing involves truncation error, a set of truncation error-free techniques known as automatic differentiation (AD) [2,3] is of particular interest in this context. Using the AD technology, a given program for evaluating some function is transformed into a new program capable of evaluating the derivatives of a subset of the input variables of the original program with respect to some of its output variables. Here, the subsets of input and output variables are chosen by the user, and this specification enters the AD transformation process as input. Software tools implementing AD are available for various languages. In this work, the AD tools ADIC [4] for C programs, ADIFOR [5,6] for Fortran programs, and ADiMat [7] for programs written in MATLAB are used; see `www.autodiff.org` for a list of current AD tools.

In MATLAB, it is not uncommon to call some external software, for instance, C or Fortran source code that, when compiled, can be run directly from within MATLAB as if it were a MATLAB built-in function. This mechanism is implemented using so-called MEX-files. In this note, we assume that AD is applied to a MATLAB program that calls an external software written in, say, C. Then, currently, two different AD tools need to be applied in two separate phases to each of the two parts, written in MATLAB and C. The resulting AD-generated codes written in MATLAB and C are required to be put together using MEX-files. The purpose of this note is to introduce a new software tool called Automatic differentiation Mexfunction Generator (AMG) that automatically generates the MEX interface functions for AD-generated code. To the best of our knowledge, there is currently no other software tool aiming at gluing AD-generated MATLAB code with AD-generated Fortran or C code.

The structure of this note is as follows. In Sect. 2, a short introduction to MEX-files is given. The program transformation of automatic differentiation is sketched in Sect. 3. The AMG tool and its application to a problem in multisensorics are reported in Sect. 4 and Sect. 5, respectively.

2 Calling C or Fortran from Within MATLAB

A MATLAB program usually consists of a collection of functions and scripts written in the MATLAB language itself, which is sufficient for a wide range of computational problems. More complicated tasks like database queries, interprocess communication, device drivers, or the use of library routines can be accomplished by calling functions, that are written in languages different from MATLAB. MATLAB provides a way to call external functions, that may be written in C/C++, Fortran, or any other language which is able to produce a compatible object file. As a simple example, consider the function `foo()` implemented in C which is presented in Fig. 1. The function accepts two scalar inputs `x` and `y` and maps them to `u` and `v`. Because C passes arguments via "call by value," the memory addresses storing the results have to be supplied to the function. This implies the (unreadable) dereferencing of the pointers of u and v whenever their content is accessed.

```
#include <math.h>
void foo(double x, double y, double *u, double *v){
   *u = 2 * y;
   *v = *u * sin(x);
}
```

Fig. 1. A simple function computing u and v from given values of x and y

The interface for calling a non-MATLAB function like `foo()` is called MEX which stands for Matlab EXternal interface [8]. The performance overhead present in MATLAB is significantly reduced using C or Fortran via MEX-files. However, a drawback is that a MEX-file is no longer platform-independent as are pure MATLAB scripts.

A function using the MEX interface is called a mexFunction which is, at the same time, the symbol that the object file has to export for MATLAB to call the implemented routine. A mexFunction is called using two arrays. One of them is used for the input arguments passed to the function. Within the other array, the mexFunction has to store references to its results. MATLAB distinguishes different mexFunctions by the filename given to the function. This filename is the mexFunction's name used by MATLAB to call the function.

Besides the definition of the interface to call an external function, several application programmer's interface (API) functions are provided. These allow the manipulation of matrices, creation and manipulation of structures, memory management, and some more sophisticated functionality.

Figure 2 shows an example of a mexFunction to call the function `foo()` as defined in Fig. 1. The mexFunction expects at least two inputs in `prhs` and returns two results in `plhs`. The checks, that these requirements are true and that both inputs have the same size, are omitted in this code for simplicity. The number of rows and columns of the first input is extracted by `mxGetM(prhs[0])` and `mxGetN(prhs[0])`, respectively. Next, the start addresses of the input data are fetched by `mxGetPr()` and stored in `x` and `y`. After that, two matrices whose size is identical to the size of the input matrices are allocated and are stored in the first and second entry of the output array `plhs`. The following two statements get the references to the memory, where the data of the results has to be stored. The `for`-loop at the end of the function body calls the function `foo()` computing two scalar outputs for two given scalar inputs. The input and output matrices are traversed in a column first order.

A MEX-file called `foo` with a platform-dependent extension is generated from the code given in Fig. 2. This process involving compiling and linking is carried out by a script called mex which is distributed with MATLAB. From within MATLAB, the function `foo()` may now be called with scalars, vectors or matrices. Given two scalar input variables `a=1.0` and `b=2.0`, a simple call

```
[r,s]=foo(a,b);
```

```
#include <mex.h>
void mexFunction(int nlhs,       mxArray *plhs[],
                 int nrhs, const mxArray *prhs[]) {
   double *x, *y, *u, *v;
   int c;
   /* Get input arguments */
   int m = mxGetM(prhs[0]);
   int n = mxGetN(prhs[0]);
   x = mxGetPr(prhs[0]);
   y = mxGetPr(prhs[1]);
   /* Allocate storage for output arguments */
   plhs[0] = mxCreateDoubleMatrix(m, n, mxREAL);
   plhs[1] = mxCreateDoubleMatrix(m, n, mxREAL);
   u = mxGetPr(plhs[0]);
   v = mxGetPr(plhs[1]);
   for(c=0; c< m*n; ++c)
      foo(x[c], y[c], &u[c], &v[c]);
}
```

Fig. 2. The MEX interface to call the function `foo()` given in Fig. 1 supporting arbitrary matrices as inputs. Consistency checks of the inputs are omitted for simplicity.

returns `r=4.0` and `s=3.3659`. If matrices rather than scalars are used as inputs for `a` and `b`, then these matrices have to be of the same size.

3 Automatic Differentiation by Source Transformation

Automatic differentiation refers to a set of techniques for transforming a given computer program P into a new program P' capable of computing derivatives in addition to the original output. More precisely, if P implements some function

$$f : \mathbb{R}^n \to \mathbb{R}^m,$$

mapping an input x to an output $y = f(x)$, the transformed program P' computes not only the function value y but also the $m \times n$ Jacobian matrix

$$J = \frac{\partial y}{\partial x}$$

at the same point x. The AD-generated program P' is called the differentiated program.

The basic idea behind AD is the fact that the execution of P is nothing but a—potentially very long—sequence of elementary arithmetic operations such as binary addition, multiplication, or intrinsic functions like $\sin(\cdot)$ or $\cos(\cdot)$. The partial derivatives of all these elementary functions are known and, following the chain rule of differential calculus, can be combined in a step-wise manner to yield the overall derivative of the entire program. The user of an AD tool specifies

(i) the input variables with respect to which derivatives are to be computed and (ii) the output variables whose derivatives are sought. These variables are called independent and dependent variables, respectively.

To illustrate AD, consider the simple code fragment given in Fig. 1 and keep in mind that the technique is also applicable to larger codes. For the sake of simplicity, we sketch a straightforward AD strategy called the forward mode by studying the transformation of the sample code fragment. In the forward mode, a gradient object ∇w is associated to every variable w appearing in the original code. In the differentiated code, this gradient object ∇w is used to store the partial derivatives of the variable w with respect to the independent variables, and additional statements for updating ∇w are executed whenever the value of w changes. Suppose that we would like to obtain derivatives of u and v with respect to x and y. Note that the forward mode follows the control flow of the original program, i.e., the differentiated code computes u and v together with ∇u and ∇v from given values of x, y, ∇x, and ∇y. Thus, in mathematical notation the above code segment would be augmented to

$$\nabla u = 2 \cdot \nabla y \; ; \; u = 2 \cdot y \tag{1}$$

$$\nabla v = \sin(x) \cdot \nabla u + u \cdot \cos(x) \cdot \nabla x \; ; \; v = u \cdot \sin(x) \quad . \tag{2}$$

The corresponding code implementing these equations is depicted in Fig. 3 as generated by the AD tool ADIC [4]. Here, all double-precision variables and

```
#include "ad_deriv.h"
#include <math.h>
#include "adintrinsics.h"
void ad_foo(DERIV_TYPE x, DERIV_TYPE y, DERIV_TYPE *u, DERIV_TYPE *v) {
   DERIV_TYPE  ad_var_0;
   double  ad_adji_0;
   double  ad_loc_0;

   ad_loc_0 = 2 * DERIV_val(y);
   ad_grad_axpy_1(&(*u), 2, &(y));
   DERIV_val(*u) = ad_loc_0;

   DERIV_val(ad_var_0) = sin( DERIV_val(x)); /*sin*/
   ad_adji_0 = cos( DERIV_val(x));
   ad_grad_axpy_1(&(ad_var_0), ad_adji_0, &(x));

   ad_loc_0 = DERIV_val(*u) * DERIV_val(ad_var_0);
   ad_grad_axpy_2(&(*v), DERIV_val(ad_var_0), &(*u),
                         DERIV_val(*u), &(ad_var_0));
   DERIV_val(*v) = ad_loc_0;
}
```

Fig. 3. Result of transforming the function `foo()` given in Fig. 1 by the forward mode of AD using ADIC [4]

type declarations that were present in Fig. 1 have been redeclared to be of type `DERIV_TYPE`. The original double value for each variable of type `DERIV_TYPE` is denoted by `DERIV_val`. The vector of derivatives with respect to the independent variables associated with an original double variable is represented by `DERIV_grad`. The functions `ad_grad_axpy_1()` and `ad_grad_axpy_2()` implement the updates of the gradients represented by the `DERIV_grad` objects according to (1) and (2).

Finally, we stress that AD is different from symbolic differentiation. In symbolic differentiation as implemented by computer algebra systems, derivative computations are based on formulae given in the form of expressions. AD, in contrast, does not generate formulae but code to evaluate the derivatives at certain points of interests. An example illustrating the differences between symbolic differentiation and AD is given in [9]. Though today's AD tools are quite remarkable in transforming large programs [10], there are still several challenges and difficulties with current implementations of AD as illustrated in [11]. The reader is referred to [2,3] and the proceedings [12,13,14,15] for more details on theory, implementations, and applications of AD.

4 Automatic Differentiation Mexfunction Generator

The Automatic differentiation Mexfunction Generator (AMG) takes up the idea of supporting the generation of MEX-files like the tools genmex [16,17] and H2MEX [18] do. Both tools support the generation of MEX interface functions either by using macro expansion or by giving various command line arguments to specify the desired interface. Both tools do not support the integration into a sequence of codes written in different languages to handle automatic differentiation.

Figure 4 shows a sample high-level specification called AMG-file, which generates the C implementation of a mexFunction to call the function `foo()`. The first line defines the mexFunction to be generated. AMG will generate a file `foo.c` which, when compiled and linked by mex, makes the function `foo()` available to MATLAB. The function takes exactly two input arguments denoted by `x` and `y` and returns two outputs `u` and `v`. The sizes of the outputs are identical to the sizes of the inputs. This is indicated by the three lines starting with `-size`. The first two `-size`-lines set the sizes of the output matrices to be identical to the size of `x`. They allocate a sufficient amount of memory for the output matrices. The third `-size`-line inserts a code fragment which checks that the sizes of `x` and `y` are identical. Finally, the kernel segment is given. The kernel segment may contain arbitrary C code. The C code generated by AMG which is similar to the one sketched in Fig. 2 will check for the correct number of actual arguments and results when the mexFunction is called. If these numbers are not as expected, then a MATLAB error message is issued.

The tool AMG is a macro processor, which supports not only generation of MEX interfaces, but has also support for converting ADiMat's [7] derivative objects to derivative formats used in ADIC [4] and ADIFOR [5,6]. The three

```
[u, v] = foo(x, y)
-size(u)=x
-size(v)=x
-size(y)=x
-kernel {
   int c;
   for(c=0;c<numel(x);++c)
      foo(x[c], y[c], &u[c], &v[c]);
}
```

Fig. 4. AMG-file to generate a MEX interface for `foo()` given in Fig. 1

AD-tools ADIC, ADIFOR and ADiMat have different formats for storing derivatives. They differ in the way derivative information is associated with its original object and in the order the derivative information is stored. An association can be done by name or by reference. An association by name is done by prefixing the original variable's name by, for example, `g_` to denote a gradient. An association by reference is done by changing the original variable's data type to `DERIV_TYPE` containing fields to hold the original data and the derivative data. ADIFOR and ADiMat use association by name, while ADIC uses association by reference. In Fig. 5 the storage schemes are illustrated. The figure presents a vector `v` in the original code and how derivative information is associated with it in the differentiated code. The derivative information is colored gray.

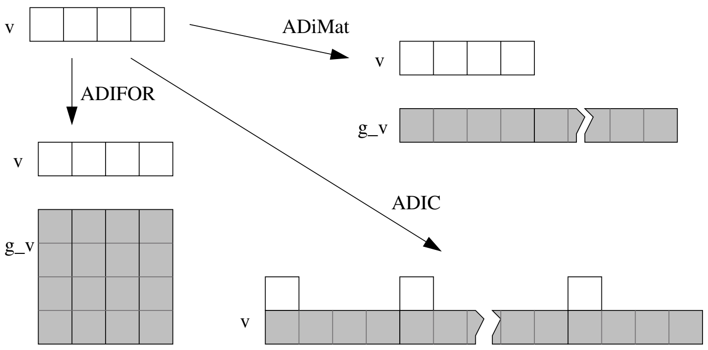


Fig. 5. Association and storage schemes of the three AD-tools ADIC, ADIFOR and ADiMat

Besides dissimilar association schemes, the storage schemes are different in the three AD-tools. ADIFOR and ADIC associate a vector of derivative information with each scalar entry of the original object, which is depicted in Fig. 5. With an $m \times n$ matrix in the original code, ADiMat associates a vector of $m \times n$ matrices storing the derivative information. Note, that the difference between the ADIFOR derivative and the ADiMat derivative is subtle. In ADiMat the derivative of the vector is a horizontal concatenation of row vectors, while in ADIFOR it is a horizontal concatenation of column vectors. Furthermore, ADIFOR and ADiMat use a column order storage scheme for matrices, i.e., entries in the same column are stored successively in the memory. ADIC uses a row order storage scheme, where entries in the same row are stored successively. The storage scheme does not depend on the AD-tool, but is motivated by the programming languages Fortran, C and MATLAB. Fortran's and MATLAB's storage schemes are identical, because MATLAB was originally implemented in Fortran.

The AMG tool offers switches to generate conversion code to interface from ADiMat-generated MATLAB code to differentiated code produced by either ADIC or ADIFOR.

The example in Fig. 6 shows the AMG-file to generate an interface for calling the ADIC differentiated code shown in Fig. 3. The first line has changed in comparison to Fig. 4. Additional variables `g_` are present for transferring derivative data from and to MATLAB. The switches `-adic` and `-active(x,y,u,v)` have been added. The first one defines, that ADIC was used to differentiate the function in the kernel. The second one declares the variables `x,y,u,v` to be active, that is, derivatives objects are associated with these variables. AMG generates code which associates the input derivative data `g_x` with the variable `x` and creates an object needed by ADIC. The variables `x` and `g_x` are not selected by their name, but by their order in the variable list. That is, the variable `g_x` occurring directly in front of the variable `x` is used as its derivative data. The ADIC derivative object produced by combining the two variables `g_x` and `x` can be referenced by specifying `adic(x)`. This returns the `DERIV_TYPE` object which is needed by the ADIC differentiated code.

```
[g_u, u, g_v, v] = g_foo(g_x, x, g_y, y)
-size(u)=x
-size(v)=x
-size(y)=x
-adic
-active(x,y,u,v)
-kernel {
   int c;
   for(c=0;c<numel(x);++c)
      ad_foo(adic(x)[c], adic(y)[c], &(adic(u)[c]), &(adic(v)[c]));
}
```

Fig. 6. AMG-file to generate a MEX interface for the ADIC-generated function `ad_foo()` given in Fig. 3.

The specification given in Fig. 6 is processed by AMG to produce a MEX interface which is then compiled and linked by mex. In MATLAB one can now use the initialization

`g_a = [1.0;0.0]` and `g_b = [0.0;1.0]`

to indicate that the first and second entries of a derivative object are used to store the derivatives with respect to `a` and `b`, respectively. Calling

```
[g_r,r,g_s,s]= g_foo(g_a,a,g_b,b);
```

at `a=1.0` and `b=2.0` then gives the results

`g_r=[0.0;2.0]`, `r=4.0`, `g_s=[2.1612;1.6829]`, and `s=3.3659` .

The values of `g_r`, for instance, show that `r` does not depend on `a` and that the derivative of `r` with respect to `b` is 2 which is easily verified from inspection of the code given in Fig. 1. The AMG tool generates 138 lines of code from the 11 lines of the specification given in Fig. 6. The benefits of this automatic interface generation are ease of use, automatic insertion of code checking for correct sizes, and reduced probability of errors or omissions that may often occur in manually written code.

5 An Application in Multisensorics

In this section, we report on the use of AMG in an application arising from multisensorics. At the University of Siegen, one is interested in the simulation of satellite constellations requiring a high accuracy of the underlying flight trajectory model [19]. Therefore, various different effects have to be considered in the mathematical model including relativity, earth tides, solar radiation pressure as well as gravitation of earth, sun and, moon. Atmospheric effects are taken into account by relying on a thermospheric model called MSIS–86 [20,21] developed at NASA's Goddard Space Flight Center. The overall simulation is formulated in MATLAB while MSIS–86 is written in Fortran77.

In a Kalman-filter for data fusion of simulated trajectory and GPS-data, derivatives are needed that are computed by means of automatic differentiation. More precisely, ADiMat is applied to the overall MATLAB code and ADIFOR is used to obtain the differentiated code for the MSIS–86 code. These two differentiated codes, written in MATLAB and Fortran77, are put together by a MEX interface generated by AMG. In Fig. 7, the AMG specification to produce the interface between the ADIFOR-generated code `g_msis_` and MATLAB is given. Note that the AMG code is written in C and that no switch is needed to tell AMG that it is interfacing to ADIFOR-generated code. The latter is not necessary because all inputs of MSIS–86 are scalars and only vectors are returned. The resulting vectors and derivative objects do not need any modification because the storage schemes of ADIFOR and MATLAB are identical for this data structure.

AMG generated a total of 130 lines of codes from the 19 lines of specification shown in Fig. 7. The ratio of lines of the generated C code and the lines of the AMG specification is $130/19 \approx 6.8$ for the `g_msis_f()` interface and $138/11 \approx 12.5$ for the `g_foo()` interface. The reason that more coded is produced for the simple `g_foo()` interface than for the more complicated `g_msis_f()` interface is as follows. More code is needed to handle the more complicated `DERIV_TYPE` objects of ADIC than the simple data structures used by ADIFOR.

```
[g_D,D,g_T,T] = g_msis_f(iday, ut, g_alt, alt, g_xlat, xlat,
                    g_xlong, xlong, xlst, f107a, f107, ap, mass)
-size(D)={1,8}
-size(g_D)={numel(g_alt),8}
-size(T)={1,2}
-size(g_T)={numel(g_alt),2}
--header{
extern void g_msis_(int*, int*, double*, double*, double*, int*,
                    double*, double*, int*, double*, double*, int*,
                    double*, double*, double*, double*, int*,
                    double*, double*, int*, double*, double*, int*);
}
-kernel {
   int tmass = (int) mass[0];
   int tiday = (int) iday[0];
   int g_p = numel(g_alt);
   g_msis_(&g_p, &tiday, ut, alt, g_alt, xlat, g_xlat, xlong, g_xlong,
           xlst, f107a, f107, ap, &tmass, D, g_D, T, g_T);
}
```

Fig. 7. AMG-file to generate a MEX interface for the ADIFOR-generated subroutine `g_msis_`.

6 Concluding Remarks

When automatic differentiation is applied to some MATLAB function that calls an external code `foo()` written in, say, C or Fortran via a MEX-file, the differentiated code consists of the corresponding differentiated MATLAB function in which the differentiated code `ad_foo()` is called via another MEX-file. The manual generation of this MEX-file is time-consuming and error-prone. To this end, a new software tool called AMG is introduced that, given a suitable high-level specification of the differentiated code `ad_foo()`, generates the corresponding MEX-file automatically. AMG is primarily designed to be used together with the AD tool ADiMat for transforming code written in MATLAB and either the AD tool ADIC for C or the AD tool ADIFOR for Fortran. The feasibility of this approach is demonstrated by a nontrivial application arising from multisensorics in which code produced by the AD tools ADiMat and ADIFOR is put together via an AMG-generated MEX-file.

Acknowledgments

This work was stimulated in part by Marc Kalkuhl and Wolfgang Wiechert of the Department of Simulation, University of Siegen, Germany, by their interest in high precision satellite orbit simulations.

References

1. Moler, C.B.: Numerical Computing with MATLAB. SIAM, Philadelphia (2004)
2. Rall, L.B.: Automatic Differentiation: Techniques and Applications. Volume 120 of Lecture Notes in Computer Science. Springer-Verlag, Berlin (1981)
3. Griewank, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. SIAM, Philadelphia (2000)
4. Bischof, C.H., Roh, L., Mauer, A.: ADIC — An extensible automatic differentiation tool for ANSI-C. Software: Practice and Experience **27**(12) (1997) 1427–1456
5. Bischof, C., Carle, A., Corliss, G., Griewank, A., Hovland, P.: ADIFOR: Generating derivative codes from Fortran programs. Scientific Programming **1**(1) (1992) 11–29
6. Bischof, C., Carle, A., Khademi, P., Mauer, A.: ADIFOR 2.0: Automatic Differentiation of Fortran 77 Programs. IEEE Computational Science & Engineering **3**(3) (1996) 18–32
7. Bischof, C.H., Bücker, H.M., Lang, B., Rasch, A., Vehreschild, A.: Combining source transformation and operator overloading techniques to compute derivatives for MATLAB programs. In: Proceedings of the Second IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2002), Los Alamitos, CA, USA, IEEE Computer Society (2002) 65–72
8. The MathWorks, Inc.: MATLAB, The External Interface Guide, Natick, Mass. (2005)
9. Bischof, C.H., Bücker, H.M., Lang, B.: Automatic Differentiation for Computational Finance. In Kontoghiorghes, E.J., Rustem, B., Siokos, S., eds.: Computational Methods in Decision-Making, Economics and Finance. Volume 74 of Applied Optimization. Kluwer Academic Publishers, Dordrecht (2002) 297–310
10. Bischof, C.H., Bücker, H.M., Lang, B., Rasch, A., Risch, J.W.: Extending the functionality of the general-purpose finite element package SEPRAN by automatic differentiation. International Journal for Numerical Methods in Engineering **58**(14) (2003) 2225–2238
11. Slusanschi, E., Bücker, H.M.: On the Limits of Current Implementations of Algorithmic Differentiation. In Petcu, D., Negru, V., Zaharie, D., Jebelean, T., eds.: Proceedings of the 6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC04, Timisoara, Romania, September 26–30, 2004, Timisoara, MIRTON (2004) 295–306
12. Griewank, A., Corliss, G.: Automatic Differentiation of Algorithms. SIAM, Philadelphia (1991)
13. Berz, M., Bischof, C., Corliss, G., Griewank, A., eds.: Computational Differentiation: Techniques, Applications, and Tools. SIAM, Philadelphia (1996)
14. Corliss, G., Faure, C., Griewank, A., Hascoët, L., Naumann, U., eds.: Automatic Differentiation of Algorithms: From Simulation to Optimization. Springer, New York (2002)
15. Bücker, H.M., Corliss, G.F., Hovland, P.D., Naumann, U., Norris, B., eds.: Automatic Differentiation: Applications, Theory, and Implementations. Volume 50 of Lecture Notes in Computational Science and Engineering. Springer, New York (2005)
16. Keady, G., Laine, M.: Using genmex – examples of calls to NAG (1995) Unpublished manuscript.
17. Hawkes, E., Keady, G.: Two more links to NAG numerics involving CA systems. In Wester, M., Steinberg, S., Jahn, M., eds.: Electronic Proceedings of the 1st International IMACS Conference on Applications of Computer Algebra. Albuquerque, New Mexico, USA, May 16–19, 1995. (1995)

18. Makdissi, A.: H2MEX gateway generator. A program available on the internet at `http://yazdiet.club.fr/h2mex.html` (2006)
19. Kalkuhl, M., Wiechert, W., Bücker, H.M., Vehreschild, A.: High Precision Satellite Orbit Simulation: A Test Bench for Automatic Differentiation in MATLAB. In Hülsemann, F., Kowarschik, M., Rüde, U., eds.: Proceedings of the Eighteenth Symposium on Simulation Techniques, ASIM 2005, Erlangen, September 12–15. Number 15 in Frontiers in Simulation, Erlangen, SCS Publishing House (2005) 428–433
20. Hedin, A.E.: MSIS–86 thermospheric model. Journal of Geophysical Research **92**(A5) (1987) 4649–4662
21. Hedin, A.E., Thuillier, G.: Comparison of OGO 6 measured thermospheric temperatures with the MSIS–86 empirical model. Journal of Geophysical Research **93**(A6) (1988) 5965–5971

KNOPPIX/Math: Portable and Distributable Collection of Mathematical Software and Free Documents

Tatsuyoshi Hamada[1], Kuniyasu Suzaki[2], Kengo Iijima[2], and Arimitsu Shikoda[3]

[1] Fukuoka University, Fukuoka 814-0180, Japan
[2] AIST, Tokyo 101-0021, Japan
[3] Tohoku Gakuin University, Tagajo 985-8537, Japan

Abstract. We propose a new computer environment for mathematicians that can be set up easily and quickly.

1 Introduction

In the 1960s, using computers was synonymous with writing programs. When the first generation of microcomputers appeared in the 1970s, computer users shared their source code and programs. During the 1980s, desktop personal computers were becoming common, and software packages were commercialized. During the 1990s, the power of personal computers increased remarkably, and hard disk storage began to be used for operating systems and software applications.

Typically, before using a software package, a user had to install it onto the hard disk. Increasingly, however, ordinary computer users have become accustomed to using pre-installed systems and applications, and laptop computers have become ubiquitous. Recently, the form and function of computers has changed radically.

A particular set of computer users, mathematicians, require special mathematical software such as the TEX system for writing papers, as well as calculating and visualizing mathematical objects. These systems are not typically pre-installed on computers; therefore, the user must configure the desktop environment.

In order to carry out investigations and get new ideas, it is a prerequisite for mathematicians to communicate each other. An important part of this communication is in the form of participation in seminars and workshops, which are often held in foreign countries. Laptop computers are a ubiquitous piece of equipment for such events. Of course, universities and research facilities always have several computers that can be rented or borrowed, and the user may not be accustomed to the desktop environment of available systems. Another disadvantage of this system is that the borrowed system may not provide support for reading and writing e-mail messages in the user's native language. As a result, before traveling to international conferences, one has to set up and configure the desktop environment on the laptop, such that they may be carried along: however, this is not a pleasant task, and hence, there is a need for a portable desktop environment that can be set up easily and quickly.

© Springer-Verlag Berlin Heidelberg 2006

2 Desktop Teleportation

We propose a new computer environment that would eliminate the need to carry laptop computers on business trips or to set up a computer for travel.

We call the proposed computer environment, “Desktop Teleportation.” This system allows users to work from their native desktop environment irrespective of their location. This is a next-generation computer environment that does not require installation, and satisfies the needs of not only for mathematicians, but also of any computer user.

3 CD Bootable System

Recently, computer operating systems that are bootable from a single CD, such as KNOPPIX, are gaining popularity. KNOPPIX is a bootable CD that contains a collection of GNU/Linux software. This project was started in Germany by Klaus Knopper. KNOPPIX can be used for Linux demos, educational presentations, and system recovery. It does not require prior installation on the hard disk, offers automatic hardware detection, and provides support for a variety of graphics cards, sound cards, SCSI and USB devices, and other peripherals. Owing to on-the-fly decompression, an optical disc can hold up to 2 GB of executable software.

Starting in the autumn of 2002, AIST created a Japanese version of KNOPPIX, as a part of the network-transferable computer project (cf. [1]). The KNOPPIX/Math project began in February 2003. This project resulted in a program that can be stored on a single KNOPPIX/Math CD. Using the CD and a USB flash memory device, and a rented computer, the user can set up his or her desktop environment on any computer.

4 KNOPPIX/Math Project

KNOPPIX/Math is the first implementation of the system that supports mathematical software and documents. The 2nd OpenXM committers meeting (February 12–14, 2003), saw the introduction of the beta version of the system, which contained OpenXM, Maxima, Gnuplot, Geomview, surf, and TEX.

The first downloadable version of the package was KNOPPIX/Math/2003. For licensing reasons, this version was distributed without the OpenXM package. Over 400 CDs were distributed at the MSJ[1] meeting (March 23–26,2003) at the University of Tokyo. At this meeting, a growing need for mathematical software suitable for mathematical research activities was felt. After discussions at this meeting, we found some core members for the “KNOPPIX/Math Project” among mathematicians who are interested in mathematical software that could be used in their research and for teaching.

[1] Mathematical Society of Japan, http://wwwsoc.nii.ac.jp/msj6/math/

The next version KNOPPIX/Math/2004 was released at the MSJ (March 27–30, 2004) at the University of Tsukuba. During that time over twenty mathematical software applications were installed on the CD. Starting from the release of this version, the OpenXM package has been network-installable. In addition, we released the first English version of KNOPPIX/Math.

KNOPPIX/Math/2005 was distributed at the 67th National Convention of IPSJ[2] (March 2–4, 2005) at the University of Electro Communications, and at the MSJ meeting (March 27–30, 2005) at Nihon University, and at other meetings. Japanese documents regarding mathematical software were also included during the distribution. On realizing the importance of lightweight languages, a few Ruby libraries for algebraic computations and NZMATH, a Python-based number-theory oriented calculation system, were also included. The first Korean version of KNOPPIX/Math was released at ASCM2005 (December 8–11, 2005) at KIAS in Korea. It supports both Korean input methods and a Hangul LaTeX package.

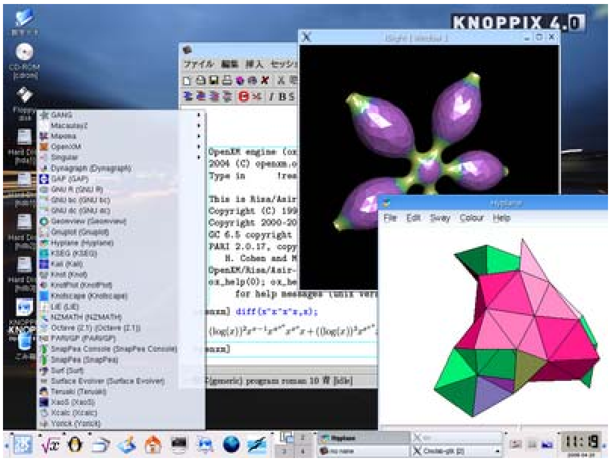
The current version, KNOPPIX/Math/2006, was presented at the 68th National Convention of IPSJ (March 8–10, 2006) at the Kogakuin University, and at the MSJ meeting (March 26–30, 2006) at Chuo University.

We changed the development system, KNOPPIX/Math/2006 is the first result of this new research project: KNOPPIX/Math Project. The members of this project include many mathematicians, mathematical software designers, and writers of free documents.

In this CD, we provide new and more versatile environments for TeX editing. For example, we included Kile, a KDE based GUI TeX editor, WhizzyTeX and Active-DVI, a pair of WYSIWYG utilities for TeX documents, and GNU TeXmacs, an office suite for TeX writing. The CD includes many additional mathematical software programs, such as BLAS, Dynagraph, EGGX/ProCALL, GAP, Geomview, Gnuplot, Hyplane, Kali, Kan, KSEG, LAPACK, Macaulay2, Maxima, NZMATH, Octave, OpenXM, PARI/GP, Qhull, R, Singular, SnapPea, surf, Surface Evolver, Teruaki, XaoS, and Yorick. For licensing reasons, there were some desirable applications that have not been included in the CD, but we have developed network installers and menus for the same, and since the user can connect to the Internet, he or she can easily use Knot, KnotPlot, Knotscape, LiE, and Risa/Asir.

We have distributed over five thousand CDs during this development period. There are many people interested in our project and have downloaded CD images from our web site (`http://www.knoppix-math.org/`). Further, some professors are currently using KNOPPIX/Math to teach mathematics at their universities. This was expected as KNOPPIX/Math provides a very convenient and economical computer environment for teaching mathematics as well as performing numerical calculations and programming. Moreover, some educators are using the program in campus information sessions for preparatory students, where the suite of programs helps generate fun and excitement related to mathematics.

[2] Information Processing Society of Japan, http://www.ipsj.or.jp/english/



```
OpenXM engine (ox
2004 (C) openxm.o
Type in      !rea

This is Risa/Asir
Copyright (C) 199
Copyright 2000-20
GC 6.5 copyright
PARI 2.0.17, copy
    H. Cohen and M
OpenXM/Risa/Asir-
ox_help(0); ox_he
     for help messages (unix vers
maxm] diff(x^x^x^x,x);
maxm]
```

Fig. 1. KNOPPIX/Math/2006 Desktop

5 The Project Policy

We initially developed KNOPPIX/Math as a desktop environment for Japanese mathematicians, and the current selection of software assumes that mathematicians are the primary users. The distribution policy of KNOPPIX/Math is that only free software and free documents are included. Because this is an experimental project, and several thousand CDs have to be distributed, the KNOPPIX/Math package had to be economical for production and easy to copy and redistribute. We are pleased to distribute this software without any license fee. Anyone is allowed to copy and distribute KNOPPIX/Math. If KNOPPIX/Math included anything besides free software, redistribution would have been difficult. But by including only free software and free documents, we are able to simultaneously distribute both software and documents. Because it is not necessary to install the program onto the hard disk, we expect to maintain low-cost distribution and high-level portability. This means that users can compare different mathematical systems from various places, and further, we have developed a method of introducing better mathematical software into Japan.

KNOPPIX/Math has succeeded in attracting the interest of many Japanese mathematicians. During the first three days of the MSJ Spring meeting we distributed 1,000 KNOPPIX/Math packages. Approximately 1,500 people are believed to have participated in the programs at this meeting.

However, most of the mathematicians who received the KNOPPIX/Math software package at the MSJ meeting are likely not use it for desktop teleportation but more as a means of trying out the intriguing mathematical software it provides – programs that are otherwise difficult to install, because our package provides a "Desktop Cloning" of diverse computer expertise. The popularity of KNOPPIX/Math lies in the facility with which it allows a user to access the suite of specialized mathematical software. Previously, even professional mathematicians have had difficulty in using unfamiliar mathematical software. Therefore, it is important to understand the mathematical basis for such software. The KNOPPIX/Math project helps introduce a variety of mathematical software and writing free documents, and provides a guide book for KNOPPIX/Math.

6 Other Remastered Versions of KNOPPIX

We consider other remastered versions of KNOPPIX, such as Quantian[3] and Freeduc[4]. Quantian is a scientific computing environment for numerical and quantitative analysis that supports openMosix, a Linux kernel extension for single-system image clustering. This kernel extension converts a network of computers into a supercomputer for Linux applications. Freeduc is intended for use in primary schools. There are many educational software and free documents. These projects are being pursued actively and support many mathematical software programs; however their main targets are not mathematics, whereas KNOPPIX/Math only supports teaching, research activities, and facilitates mathematical presentations.

Two of the authors of this paper are investigating network boot systems, such as SFS-KNOPPIX (cf. [2]) and HTTP-FUSE KNOPPIX, which enable one to select the root file system from the server. In particular, HTTP-FUSE KNOPPIX may become the next generation system of thin clients. Despite its economical design, this program is used for general office work in some municipalities. T. Kosuge's team at Japan Electronic College is investigating the CF-KNOPPIX and USB-KNOPPIX projects. They have produced systems that boot from compact flash memory and USB flash memory. The team has also fabricated computer education rooms using CF-KNOPPIX, thereby developing economical and maintenance-free educational environments. Another author of this paper is working on the ongoing KNOPPIX Edu project (cf. [3])(supported by AIST and Alpha Systems Inc).

The desktop teleportation system is an effective tool in education. If students have personal computers, they can study at home using the same desktop environment that they use at school. Last year, we began a collaborative project, in which we distributed both KNOPPIX/Math/2005 and KNOPPIX Edu4 in a single package. Now, we are continuing our collaboration, and our projects are spreading to various educational and research areas. Recently, Alpha Systems Inc. has released the Live CD Acceleration Toolkit (LCAT) and Accelerated-KNOPPIX[5]. Accelerated-KNOPPIX is a fast boot KNOPPIX that by optimizes the start-up part of the live CD. We have applied this toolkit to KNOPPIX/Math/2006 and KNOPPIX Edu5 as well.

[3] http://dirk.eddelbuettel.com/quantian/
[4] http://www.ofset.org/freeduc-cd/
[5] http://www.alpha.co.jp/ac-knoppix/index_en.html

Finally, we would like to introduce a new version of the computer environment that uses a virtual machine. N. Takayama and M. Noro have released VMware/knoppix/math, which is the virtual machine image of KNOPPIX/Math. The VMware Player[6] is a freely downloadable application developed by VMware Inc; it enables users to easily run any virtual machine on a Windows or Linux operating system. Using this product renders KNOPPIX/Math user friendly even on a PC even without a CD-ROM drive.

References

1. Suzaki, K.: Network Transferable Computer, `http://staff.aist.go.jp/k.suzaki/English/NTC/`.
2. Suzaki, K., Iijima, K., Yagi, T., Tan, H. and Goto, K.: SFS-KNOPPIX, The 4th IEEE International Symposium on Network Computing and Applications(IEEE NCA05), July 27–29, 2005, `http://unit.aist.go.jp/itri/knoppix/IEEE-NCA05-paper.pdf`.
3. Shikoda, A., Ono, T., Kumagai, M., Ishikawa, M., Chiba, D., Suzaki, K., Kato, K. and Hamada, T.: A practicable educational contents sharing system utilizing Knoppix Edu series the 1 CD bootable Linux teaching tools, ITHET 6th Annual International Conference, T1A-1, July 7–9, 2005, `http://fie.engrng.pitt.edu/ithet2005/papers/2125.pdf`.

[6] http://www.vmware.com/products/player/

Stability of Parametric Decomposition

Kazuhiro Yokoyama

Department of Mathematics, Rikkyo University
3-34-1 Nishi Ikebukuro Toshima-ku, Tokyo, 171-8501, Japan
yokoyama@rkmath.rikkyo.ac.jp

Abstract. We deal with ideals generated by polynomials with parametric coefficients, and introduce "stabilities on ideal structures" based on stability of forms of Gröbner bases. Then, we extend those stabilities to radicals and irreducible decompositions and show the computational tractability on those computations by integrating existing techniques.

1 Introduction

Computations of Gröbner bases and triangular systems play very important roles in analyzing the structure of polynomial ideals and their varieties. Lots of efforts have been devoted to developing efficient such computations, and these developed efficient methods contribute to solving engineering problems. However, ideal generators derived from engineering or pure mathematics often contain parameters in coefficients or exponents, and depending on the values of these parameters, the structures of ideals/varieties change. For a systematic study of these ideals, analyzing conditions on parameters for desired zeros and analyzing conditions on parameters for a "stable" ideal/variety structure are very important. Several works were done in this direction; as to "uniformity" of the shape of Gröbner basis, analyzing specialization for such uniformity [1,5], computing methods for comprehensive Gröbner basis [13,7,10,14,11], and computing methods based on computation of triangular systems [3,12]. Recently, ideals generated by polynomials with parametric exponents are dealt with in [15,16,17].

Here we introduce the term "parametric ideal", as a general term, which represents polynomial ideals generated by a number of polynomials with parameters in coefficients or exponents. Our first target is to analyze/classify the values of parameters for "stable structures" of ideals/varieties. After classification of parameters on such stabilities, we can go further to consider how decomposition, like radical computation, prime/primary decomposition, of such ideals can be described uniformly and how one can analyze/classify the values of parameters for having such uniformity. We may call this "parametric decomposition".

In this paper, concentrating on parametric coefficient case, we give a systematic approach for computation of "parametric decomposition" of a parametric ideal, and we show that one can realize radical computation and irreducible decomposition by integrating existing algorithms. Making these computational realizations *efficient and practical* is not dealt with here, but it is very *challenging* for Computer Algebra not only in theory, but also in application. It will certainly help to develop the abilities of Computer Algebra and widen its application.

© Springer-Verlag Berlin Heidelberg 2006

2 Stability of Parametric Ideals

Here, for ideals generated by polynomials with parametric coefficients, we explain "stabilities" of structures of ideals and give brief discussion on tractability of classification of the values of parameters for such "stable structures".

We consider polynomial rings $K[A,X]$ and $\bar{K}[A,X]$ in two kind of variable set $A = \{a_1,\ldots,a_m\}$ and $X = \{x_1,\ldots,x_n\}$ over a computable field K of characteristic 0 and its algebraic closure $\bar{K}$, where A represents the set of parameters and X the set of (ordinary) variables. For each value $\alpha = (\alpha_1,\ldots,\alpha_m)$ of A in $\bar{K}^m$, we denote by φ_A the ring homomorphism from $\bar{K}[A,X]$ to $\bar{K}[X]$ obtained by substitution of A with α:

$$\varphi_\alpha : \bar{K}[A,X] \ni f(a_1,\ldots,a_m,x_1,\ldots,x_n) \to f(\alpha_1,\ldots,\alpha_m,x_1,\ldots,x_n) \in \bar{K}[X].$$

For a finite subset F of $K[A,X]$, we call the family $\{\langle\varphi_\alpha(F)\rangle \mid \alpha \in \bar{K}^m\}$ of ideals of $\bar{K}[X]$ the *parametric ideal* generated by F. Letting $\mathcal{I} = \langle F\rangle$ (the ideal generated by F) in $\bar{K}[A,X]$, we have $\langle\varphi_\alpha(F)\rangle = \varphi_\alpha(\mathcal{I})$ for any $\alpha \in \bar{K}^m$. Thus, we also call $\mathcal{I}$ *the parametric ideal* generated by F. (If we restrict all parameter values in K^m, we consider ideals of $K[A,X]$ and the restriction of φ_α on $K[A,X]$.) Let $T(X), T(A), T(X,A)$ denote the set of all terms of X, A, and $X \cup A$, respectively.

Definition 1 (Stability of Parametric Ideal)

1. Forms of Generators: *A subset G of $K[A,X]$ is called a* stable Gröbner basis *of a parametric ideal $\mathcal{I}$ in a region $\mathcal{C}$ of $\bar{K}^m$ with respect to an order $\prec$ on $T(X)$, if $\varphi_\alpha(G)$ is a Gröbner basis of the ideal $\varphi_\alpha(\mathcal{I})$ in $\bar{K}[X]$ with respect to $\prec$ and $\varphi_\alpha(HC_\prec(g)) \neq 0$ for all $g \in G$ for any α in $\mathcal{C}$, where $HC_\prec(g)$ denotes the head coefficient of g considered as a polynomial in X over $K(A)$.*
2. Other Structural Invariants: *We can deal with* non-triviality, dimension, *and also* linear-dimension for 0-dimensional case. *For example, the dimension of $\mathcal{I}$ is said to be* stable *in a region $\mathcal{C}$ of $\bar{K}^m$, if the dimension of $\varphi_\alpha(\mathcal{I})$ coincides for any $\alpha \in \mathcal{C}$. Related to dimension, we can also deal with a* maximal strongly independent set (MSIS) with maximal cardinality modulo a parametric ideal. *That is, a subset Y is called a* stable MSIS with maximal cardinality modulo $\mathcal{I}$ *in a region $\mathcal{C}$, if Y is a MSIS with maximal cardinality modulo $\varphi_\alpha(\mathcal{I})$ for any α in $\mathcal{C}$. (For details on MSIS, see [2].)*

Here each region $\mathcal{C}$ is supposed to be algebraically constructible, *that is, two finite subsets P_1 and P_2 of $K[A]$ are explicitly given and $\mathcal{C}$ is defined as $\mathcal{C} = V(P_1) \setminus V(P_2)$, where $V(P_i)$ denotes the set of all zero of P_i in $\bar{K}^m$. We may assume that $V(P_2) \subset V(P_1)$ by replacing P_2 with $P_2 \cup P_1$ if necessary.*

What we want to do is to *classify parameter values*, that is, for a specified *structural invariant*, say Q, and a given subset $\mathcal{S}$ of $\bar{K}^m$, we compute a finite number of algebraically constructible subsets $\mathcal{C}$ called *cells* such that

$$\mathcal{S} \subset \cup\, \mathcal{C},$$

and in each cell $\mathcal{C}$, the target Q is stable for any value $\alpha \in \mathcal{C}$.

Definition 2 (Cell Decomposition). *We call the above decomposition a* cell decomposition *of $\mathcal{I}$ among $\mathcal{S}$ for the target structural invariant Q. For simplicity, by saying that we compute a* stable Q *of $\mathcal{I}$ among $\mathcal{S}$, we mean that we compute this cell decomposition for Q. (So, computing a stable Gröbner basis means computing a cell decomposition for a stable Gröbner basis.) When $\mathcal{S} = \bar{K}^m$, we simply say that we compute a* stable Q *of $\mathcal{I}$.*

By several existing studies [7,11], one can compute stable Gröbner bases with respect to any term oder $\prec$ on $T(X)$. (For computation of examples in the paper, the author utilized an implementation of [11] by Akira Suzuki.) Main tool for this computation is Gröbner basis with respect to a block order $\prec_{X,A}$ on $T(X, A)$ with $A \prec\prec X$ such that its restriction $\prec_X$ on $T(X)$ coincides with the given $\prec$.

Lemma 1. [5,11] *Let F, G_0 be subsets of $K[A, X]$, and assume that $G_0 \subset K[A, X]$ is a Gröbner basis of the ideal $\mathcal{I} = \langle F \rangle$ of $\bar{K}[A, X]$ with respect to $\prec_{X,A}$. If $\varphi_\alpha(HC_\prec(g)) \neq 0$ for each $g \in G_0 \setminus (G_0 \cap K[A])$, where $HC_\prec(g)$ denotes the head coefficient of g considered as a polynomial in X over $K(A)$, then $\varphi_\alpha(G_0 \setminus (G_0 \cap K[A]))$ is a Gröbner basis of $\varphi_\alpha(\mathcal{I})$ with respect to $\prec$ for any $\alpha \in V(G_0 \cap K[A])$. In other word, $G = G_0 \setminus (G_0 \cap K[A])$ is a stable Gröbner basis of $\mathcal{I}$ with respect to $\prec$ in $\mathcal{C} = V(G_0 \cap K[A]) \setminus V(\{\prod_{g \in G} HC_\prec(g)\} \cup (G_0 \cap K[A]))$. Of course, if $G_0 \cap K[A] \neq \emptyset$, $\{1\}$ is a stable Gröbner basis in $\bar{K}^m \setminus V(G_0 \cap K[A])$.*

By Lemma 1, the region $\mathcal{C}' = V(\{\prod_{g \in G} HC_\prec(g)\} \cup (G_0 \cap K[A]))$ remains. Then, we compute a Gröbner basis of $\langle F \cup \{HC_\prec(g)\} \rangle$ for each g in G, and applying Lemma 1, we have a stable Gröbner basis in new region $\mathcal{C}'' \subset \mathcal{C}'$. Repeating this process, we finally obtain a stable Gröbner basis $\{G_\mathcal{C}\}$. We note that $G_\mathcal{C} \subset K[A, X]$ for all $\mathcal{C}$. The termination of the above procedures in finitely many step can be shown by the noetherian property of a polynomial ring, because at each step, newly generated ideal becomes strictly larger. See the detail in [11], where stable Gröbner basis is called *Comprehensive Gröbner system.*

Remark 1. In the process of cell decomposition for ideal structure, each cell will be decomposed into new smaller cells repeatedly. Also, sometimes the intersection of two different cells becomes necessary. For this case, the intersection of $\mathcal{C}_1 = V(P_{1,1}) \setminus V(P_{1,2})$ and $\mathcal{C}_2 = V(P_{2,1}) \setminus V(P_{2,2})$ can be computed by $V(P_{1,1} \cup P_{2,1}) \setminus V(P' \cup P_{1,1} \cup P_{2.1})$, where $\langle P' \rangle = \langle P_{1,2} \rangle \cap \langle P_{2,2} \rangle$. Also we can compute "exclusion" $\mathcal{C}_1 \setminus \mathcal{C}_2$ as follows: Letting $\langle P'' \rangle = \langle P_{1,2} \rangle \cap \langle P_{2,1} \rangle$, $\mathcal{C}_1 \setminus \mathcal{C}_2 = V(P_{1,1}) \setminus V(P'' \cup P_{1,1})$, if $V(P_{2,2}) \cap V(P_{1,1}) \subset V(P_{1,2})$, and $\mathcal{C}_1 \setminus \mathcal{C}_2 = (V(P_{1,1}) \setminus V(P'' \cup P_{1,1})) \cup (V(P_{2,2} \cup P_{1,1}) \setminus V(P_{1,2} \cup P_{2,2}))$, otherwise.

Once we have such a stable Gröbner basis, its stable dimension, and stable linear-dimension can be efficiently computed, because those are computed from only head terms of the computed Gröbner basis. In more details, for each pair of a stable Gröbner basis $G_\mathcal{C}$ with respect to a total degree order and a region $\mathcal{C}$, we can compute a stable MSIS Y with maximal cardinality, by seeing all head terms of $G_\mathcal{C}$ which are considered as terms in X. Then the size of Y is the dimension and it is stable in $\mathcal{C}$. (See [2].)

Lemma 2. *For a parametric ideal, one can compute its stable Gröbner basis and its stable MSIS with maximal cardinality, which gives its stable dimension. Moreover, when a parametric ideal has a stable dimension 0 in a region $\mathcal{C}$, one can compute a stable linear dimension among $\mathcal{C}$.*

Also, one can compute the followings as those can be reduced to a stable Gröbner basis of some modified ideal with respect to an elimination order.

Lemma 3. *For a parametric ideal $\mathcal{I}$ and a polynomial h in $K[A, X]$, one can compute a* stable saturation $\mathcal{I} : h^\infty$. *Also for parametric ideals $\mathcal{I}_1, \ldots, \mathcal{I}_s$, one can compute their* stable intersection $\cap_{i=1}^s \mathcal{I}_i$.

3 Stability of Parametric Decomposition

The main target is *parametric decomposition*, which means to classify the values of parameters for "stable" structural invariants related to "ideal decomposition". Here we deal with two decompositions, radical computation and absolutely irreducible decomposition, and give the computational tractability of those.

From now on, we assume that for a parametric ideal $\mathcal{I} = \langle F \rangle$, where $F \subset K[A, X]$, its stable Gröbner basis and its stable dimension (with stable MSIS) are already computed, that is, we already know $\bar{K}^m = \cup\, \mathcal{C}$, and for each $\mathcal{C}$, its stable Gröbner basis $G_\mathcal{C}$ is computed. And for simplicity, the phrase "computing a parametric ideal" always means "computing its stable Gröbner basis". For parametric decompositions, corresponding decomposition of $\mathcal{I}$ in $\bar{K}[A, X]$ is very useful. Here, for an ideal $\mathcal{J}$, $\sqrt{\mathcal{J}}$ denotes its radical.

Lemma 4 (Correspondence of Decomposition). [17]

1. *Assume that $\mathcal{I} = \cap_{i=1}^u \mathcal{I}_i$ holds for ideals $\mathcal{I}, \mathcal{I}_i$ in $K[A, X]$. If $\mathcal{I}_i$'s are co-maximal, then $\varphi_\alpha(\mathcal{I}) = \cap_{i=1}^u \varphi_\alpha(\mathcal{I}_i)$ for any α.*
2. *Assume that $\sqrt{\mathcal{I}} = \cap_{i=1}^v \sqrt{\mathcal{J}_i}$ for ideals $\mathcal{I}, \mathcal{J}_i$ in $K[A, X]$. Then $\sqrt{\varphi_\alpha(\sqrt{\mathcal{I}})} = \cap_{i=1}^v \sqrt{\varphi_\alpha(\sqrt{\mathcal{J}_i})}$ for any α.*

Lemma 4 can be shown by the property $\cap_{i=1}^u \mathcal{I}_u = \prod_{i=1}^u \mathcal{I}_u$ for co-maximal ideals $\mathcal{I}_i$ and $\cap_{i=1}^v \sqrt{\mathcal{J}_u} = \sqrt{\prod_{i=1}^v \mathcal{J}_u}$ for ideals $\mathcal{J}_i$.

3.1 Stability of Radical

First we consider the stability of the radical of $\mathcal{I}$. As $\sqrt{\varphi_\alpha(\mathcal{I})} = \sqrt{\varphi_\alpha(\sqrt{\mathcal{I}})}$, we may assume that $\mathcal{I}$ is already radical in $K[A, X]$. Naturally we can introduce the notion of stability of radicals.

Definition 3 (Stable Radical)
A parametric ideal $\mathcal{H} = \langle H \rangle$ of $\bar{K}[A, X]$, where $H \subset K[A, X]$, is a stable radical of $\mathcal{I}$ in a region $\mathcal{C}$, if $\varphi_\alpha(\mathcal{H})$ is the radical of $\varphi_\alpha(\mathcal{I})$ for any $\alpha \in \mathcal{C}$.

Now we outline the tractability of stable radical computation. We begin with 0-dimensional case.

0-DIMENSIONAL CASE:
We assume that a given parametric ideal $\mathcal{I}$ is 0-dimensional in a give region $\mathcal{C}_0$.

Definition 4 (Minimal Polynomial and its Square-free Part)

1. Minimal polynomial: *For a polynomial f in $K[A,X]$, a univariate polynomial $m(y)$ in new variable y over $K[A]$ is a* stable minimal polynomial *of f with respect to $\mathcal{I}$ in a region $\mathcal{C} \subset \mathcal{C}_0$, if $\varphi_\alpha(m_f(y))$ is a minimal polynomial (not necessarily monic) of $\varphi_\alpha(f)$ with respect to $\varphi_\alpha(\mathcal{I})$ for any $\alpha \in \mathcal{C}$.*
2. Square-free part of a polynomial: *For a univariate polynomial $g(y)$ in y over $K[A]$, $h(y)$ is a* stable square-free part *of $g(y)$ in a region $\mathcal{C}$, if $\varphi_\alpha(h(y))$ is the square-free part of $\varphi_\alpha(g(y))$ for any $\alpha \in \mathcal{C}$.*

As to minimal polynomial, it can be computed by using linear algebra over $K[A,X]$ with parameter classification. Or it may be considered as computation of a stable Gröbner basis of $\langle G \cup y - f \rangle$ in $K[A,X,y]$ with respect to an elimination order $y \prec\prec X$. As to square-free part of a polynomial $h(y)$, it can be computed by computing a parametric ideal generated by $h(y)$ and $\frac{dh}{dy}$ in $K[A,X,y]$, or parametric Sturm-Habich sequence computation. Then "stable" radical can be computed by simply adding all "stable" square-free parts of stable minimal polynomials of variables x_i to $\mathcal{I}$.

In the whole procedure, we add new structure for testing its stability successively; minimal polynomials, those square-free parts and ideals generated by $\mathcal{I}$ and those stable square-free parts. So, at each step of adding new stable structure, we execute an additional cell decomposition for each already computed cell, which can be terminated in finitely many steps and produces finitely many newly computed cells. Thus, finally we obtain the following cell decomposition within finitely many steps: $\mathcal{C}_0 = \cup_{i \geq 1} \mathcal{C}_i$ with finitely many cells $\mathcal{C}_i$ such that, in each $\mathcal{C}_i$, $\mathcal{I}$ has a stable linear dimension, stable minimal polynomials $m_j(x_j)$ $(1 \leq j \leq n)$ and those stable square-free parts, and $G_i \subset K[A,X]$ is a stable Gröbner basis of a stable radical of $\mathcal{I}$.

Lemma 5 *For a parametric ideal $\mathcal{I}$ and a region $\mathcal{C}_0$, where $\mathcal{I}$ has stable dimension 0, one can compute its stable radical, that is, one can compute a cell decomposition $\mathcal{C}_0 = \cup \mathcal{C}_i$ with a set of stable Gröbner bases $G_i \subset K[A,X]$ such that $\varphi_\alpha(G_i)$ is a Gröbner basis of $\sqrt{\varphi_\alpha(\mathcal{I})}$ for any α in $\mathcal{C}_i$.*

Example 1 Consider $F = \{x^3 + ax, yx + ay - 1\}$, where x, y are variables and a is a parameter. Then, $G_0 = \{x^3 + ax, (a^3 + a^2)y - x^2 + ax - a^2 - a, y(x+a) - 1\}$ is a Gröbner basis of $\langle F \rangle$ and it has a stable dimension 0 in a region $\mathcal{C} = \mathbf{C}^2 \setminus \{0, -1\}$. In $\mathcal{C}$, the minimal polynomial $m_x(x)$ of x is $x^3 + a$ and that $m_y(y)$ of y is $(-a^3 - a^2)y^3 + (3a^2 + a)y^2 - 3ay + 1$. Then, $m_x(x)$ is square-free in $\mathcal{C}$ and $m_y(y)$ is also square-free in $\mathcal{C}$. Because $\langle m_x(x), dm_x(x)/dx \rangle = \langle a^2, ax, 3x^2 + a \rangle$ and $\langle m_y(y), dm_y(y)/dy \rangle$ is trivial as ideals of $\mathbf{C}[x,y,a]$. Thus, $\mathcal{I}$ is a stable radical of itself in $\mathcal{C}$. For $a = 0$, $\varphi_0(F) = \{x^3, yx - 1\}$ and the ideal becomes trivial. For $a = -1$, $\langle \varphi_{-1}(F) \rangle = \langle x^2 + x, x + 2y + 2 \rangle$ is radical.

Non 0-dimensional case:
Next we consider a parametric ideal $\mathcal{I}$ which has a stable Gröbner basis $G_0 \subset K[A, X]$ with stable dimension greater than 0 in a region C_0. In this case, there might be prime divisors of various dimensions. So, tracing "radical computation" of ordinary polynomial ideals, we may modify the stability as follows:

Definition 5 (Set Representation of Stable Radical)
A set of parametric ideals $\{\mathcal{H}_1 = \langle H_1\rangle, \ldots, \mathcal{H}_s = \langle H_s\rangle\}$, where $H_i \subset K[A, X]$, of $\bar{K}[A, X]$ is a set representation of stable radical of $\mathcal{I}$ in a region $\mathcal{C}$, if $\cap_{i=1}^{s}\varphi_\alpha(\mathcal{H}_i)$ is the radical of $\varphi_\alpha(\mathcal{I})$ for any $\alpha \in \mathcal{C}$.

To obtain a set representation of stable radical, we trace all steps of radical computation for ordinary polynomial ideals. As to ideal extension and contraction, we can show the following (see [4,2] for its details on ordinary computation).

Lemma 6 *For a parametric ideal $\mathcal{I}$ and a region $\mathcal{C}$, assume that $\mathcal{I}$ has a stable MSIS Y with respect to a total degree order. Let $\mathcal{J}$ be a parametric ideal of $\bar{K}(Y)[A, X \setminus Y]$. Then, with respect to Y, one can compute a stable extension $\mathcal{I}^e$ among $\mathcal{C}$, a stable contraction of $\mathcal{J}$ among $\mathcal{C}$ and a stable extension-contraction $\mathcal{I}^{ec}$ among $\mathcal{C}$.*

Computation of a stable extension of $\mathcal{I}$ can be done by computation of a stable Gröbner basis of $\mathcal{I}$ with respect to a block order $Y \prec\prec X \setminus Y$. Also, for a stable contraction of $\mathcal{J}$ can be done by computation of the saturation $\mathcal{J} : h^\infty$, where $G \subset K[A, X]$ is a stable Gröbner basis of $\mathcal{J}$ and $h = \prod_{g\in G} HC(g)$. Here $HC(g) \in K[A, Y]$ denotes the leading coefficient of g considered as a polynomial in $X \setminus Y$ over $K[A, Y]$.

Applying Lemma 2, we have a cell decomposition $\mathcal{C}_0 = \cup_{i\geq 1} \mathcal{C}_i$ of finitely many cells $\mathcal{C}_i$ with a set of stable Gröbner basis $G_i \subset K[A, X]$ such that $\varphi_\alpha(G_i)$ is a Gröbner basis of the extension $\varphi_\alpha(\mathcal{I})^e$ generated by $\varphi_\alpha(\mathcal{I})$ in $\bar{K}(Y)[X \setminus Y]$ for any α in $\mathcal{C}_i$. Moreover, we have $\mathcal{I}^{ec} = \mathcal{I}^e \cap \bar{K}[A, X] = \mathcal{I} : h^\infty$ and $\varphi_\alpha(\mathcal{I})^{ec} = \varphi_\alpha(\mathcal{I})^e \cap \bar{K}[X] = \varphi_\alpha(\mathcal{I}) : \varphi_\alpha(h)^\infty$. Now we give the notion of minimal polynomial for non-zero dimensional case:

Definition 6 (Minimal Polynomial modulo Extension). *Let Y be a stable MSIS with maximal cardinality modulo $\mathcal{I}$ in a region $\mathcal{C}$. For a polynomial f in $K[A, X]$, a univariate polynomial $m(z)$ in new variable z over $K[A, Y]$ is a* stable minimal polynomial *of f with respect to the extension $\mathcal{I}^e \subset \bar{K}(Y)[A, X \setminus Y]$ in a region $\mathcal{C}$, if $\varphi_\alpha(m_f(z))$ is a minimal polynomial of $\varphi_\alpha(f)$ with respect to the extension $\varphi_\alpha(\mathcal{I})^e \subset \bar{K}(Y)[X \setminus Y]$ for any $\alpha \in \mathcal{C}$.*

As the extension ideal $\varphi_\alpha(\mathcal{I})^e$ is 0-dimensional in $\bar{K}(Y)[A, X \setminus Y]$, we compute its stable radical by using all stable square-free parts of stable minimal polynomials of variables $X \setminus Y$, all of which can be computed by the same manner as in 0-dimensional case. (We omit the details here.) Thus, we have a cell decomposition $\mathcal{C}_i = \cup_j \mathcal{C}_j^*$ of finitely many cells $\mathcal{C}_j^*$ with a set of stable Gröbner bases $G_j^* \subset K[A, X]$ such that $\varphi_\alpha(G_j^*)$ is a Gröbner basis of $\sqrt{\varphi_\alpha(\mathcal{I})^e}$ for any α in $\mathcal{C}_j^*$.

Then, by computing stable contractions, we obtain a further cell decomposition $\mathcal{C}_i = \cup_j \mathcal{C}_{i,j}$ of finitely many cells $\mathcal{C}_{i,j}$ with a set of stable Gröbner bases $G_{i,j} \subset K[A, X]$ such that $\varphi_\alpha(G_{i,j})$ is a Gröbner basis of $\sqrt{\varphi_\alpha(\mathcal{I})^e} \cap \bar{K}[X]$ for any α in $\mathcal{C}_{i,j}$. Since $\varphi_\alpha(\mathcal{I}) = (\varphi_\alpha(\mathcal{I}^e) \cap \bar{K}[X]) \cap (\varphi_\alpha(\mathcal{I}) + \langle \varphi_\alpha(h)^t \rangle)$, we have the following by Lemma 4.

Lemma 7. *For any α in $\mathcal{C}_{i,j}$,*

$$\sqrt{\varphi_\alpha(\mathcal{I})} = \varphi_\alpha(\langle G_{i,j} \rangle) \cap \sqrt{\varphi_\alpha(\langle G_{i,j} \cup \{h\} \rangle)}$$

By Lemma 7, we can repeat the whole computation described above for the parametric ideal $\langle G_{i,j} \cup \{h\} \rangle$, and so on. By the noetherian property, we finally obtain a set representation of stable radical of $\mathcal{I}$ in finitely many repetition. As each cell decomposition for $\{G_{i.j}\}$ can be executed in a finitely many steps, the whole computation is terminated in a finitely many steps. Also by Lemma 3, the intersection of parametric ideals can be computable. Thus, we have

Theorem 1. *For a parametric ideal, one can compute a set representation of its stable radical, and thus one can also compute its stable radical.*

Using Lemma 4, we may use any decomposition $\mathcal{I} = \cap \mathcal{J}_i$. For each $\mathcal{J}_i$, we compute its stable radical and then, combining those stable radicals, we obtain a set representation of stable radical of $\mathcal{I}$. This might improve the computational efficiency.

Example 2. We consider the *spectral decomposition* of an even polynomial which is very important operation in control theory [6]. Let $f_1(x) = x^6 + s^2x^4 - t^2$, where x, s, t are variables. Then, we want to computer a polynomial $g_1(x) = x^3 + b_2x^2 + b_1x + b_0$ such that $f_1(x) = -g_1(x)g_1(-x)$. Considering b_0, b_1, b_2 as variables, we have an ideal $\mathcal{I}_1 = \langle b_0^2 - t^2, b_1^2 - 2b_0b_2, b_2^2 - 2b_1 + s^2 \rangle$ of $\mathbf{C}[b0, b1, b2, s, t]$. Now we consider b_0, b_1, b_2, s as variables and t as a parameter, and compute a stable radical of $\mathcal{I}$. Here, the ring-homomorphism φ_a is set as $\varphi_a : \mathbf{C}[b_0, b_1, b_2, s, t] \ni f(b_0, b_1, b_2, s, t) \to f(b_0, b_1, b_2, s, a) \in \mathbf{C}[b_0, b_1, b_2, s]$.

With respect to the reverse lexicographic order, the generators form a Gröbner basis for any parameter value, and $\{s\}$ is a stable MSIS. Computing the primary decomposition of $\mathcal{I}_1 \cap \mathbf{Q}[b_0, b_1, b_2, s, t]$, we have $\mathcal{I}_1 = \mathcal{J}_1 \cap \mathcal{J}_2$, where $\mathcal{J}_1 = \langle b_0 - t, b_1^2 - 2tb_2, b_2^2 - 2b_1 + s^2 \rangle$ and $\mathcal{J}_2 = \langle b_0 + t, b_1^2 + 2tb_2, b_2^2 - 2b_1 + s^2 \rangle$. Then, for $\mathcal{J}_1$, we compute its stable radical.

For any value of t, b_2 has its stable minimal polynomial $m_2(x) = x^4 + 2s^2x^2 - 8tx + s^4$ with respect to $\mathcal{J}_1^e$ and its stable square-free part is described as follows: If $t \neq 0$, $m_2(x)$ is square-free and if $t = 0$, its square-free part is $x^2 + s^2$. Similarly, b_1 has its stable minimal polynomial $m_1(x) = x^4 - 8t^2x + 4t^2s^2$ for any parameter value, and its stable square-free part is described as follows: If $t \neq 0$, $m_1(x)$ is square-free and if $t = 0$, its square-free part is x. And $x - t$ is the stable minimal polynomial b_0 for any parameter values and it is also the stable square-free part. As a result, for any $a \neq 0$, $\varphi_a(\mathcal{J}_1)$ is radical, and $\sqrt{\varphi_0(\mathcal{J}_1)} = \langle b_0, b_1, b_2^2 + s^2 \rangle$.

By the same manner, it follows that for any $a \neq 0$, $\varphi_a(\mathcal{J}_2)$ is radical, and $\sqrt{\varphi_0(\mathcal{J}_2)} = \langle b_0, b_1, b_2^2 + s^2 \rangle$, which coincides with $\sqrt{\varphi_0(\mathcal{J}_1)}$.

3.2 Stability of Irreducible Decomposition

Next we consider the stability of absolutely irreducible decomposition. For simplicity, we omit the word "absolutely". Different from *radical computation*, where all coefficients of stable Gröbner bases are still belonging to the original coefficient field K, irreducible decomposition requires some algebraic extensions depending on the value α. Thus, we modify the stability as follows:

Definition 7 (Stable Irreducible Decomposition)
A set of parametric ideals $\{\mathcal{P}_1 = \langle P_1\rangle, \ldots, \mathcal{P}_t = \langle P_t\rangle\}$, *where* $P_i \subset L[X]$ *for each* i *and* L *is some residue class ring* $K[A,Z]/\mathcal{J}$ *factored by an ideal* $\mathcal{J}$ *of* $K[A,Z]$, *gives a* stable irreducible decomposition *of* $\mathcal{I}$ *in a region* $\mathcal{C}$, *if* $\cap_{i=1}^t \varphi_{\alpha,\beta}(\mathcal{P}_i)$ *gives the irreducible decomposition* $\sqrt{\varphi_\alpha(\mathcal{I})}$ *for any* $\alpha \in \mathcal{C}$ *and any zero* β *of* $\varphi_\alpha(\mathcal{J})$. *Here* $\varphi_{\alpha,\beta}$ *is a homomorphism from* $\bar{K}[A,Z,X]$ *to* $\bar{K}[X]$, *which is a natural extension of* φ_α *and defined as follows: For an element* $h(A,Z)$ *in* $\bar{K}[A,Z]$, $\varphi_{\alpha,\beta}(h(A,Z)) = h(\alpha,\beta)$.

Now we outline the *theoretical tractability* of irreducible decomposition. For stable irreducible decomposition, we first compute a set representation of stable radical. So, we consider the pair of a set $\{\mathcal{H}_1, \ldots, \mathcal{H}_s\}$ and a region $\mathcal{C}_0$ such that $\{\mathcal{H}_1, \ldots, \mathcal{H}_s\}$ is a set representation of stable radical of $\mathcal{I}$ in $\mathcal{C}_0$. Then, computation of stable irreducible decomposition of $\mathcal{I}$ can be reduced to those of $\mathcal{H}_i$, where $\varphi_\alpha(\mathcal{H}_i)$ consists of the same dimensional prime divisors. Thus, we only consider a parametric ideal $\mathcal{H}$ which has stable MSIS Y with maximal cardinality (stable dimension) in a region $\mathcal{C}$ and $\varphi_\alpha(\mathcal{H})^{ec} = \varphi_\alpha(\mathcal{H})$ for any α in $\mathcal{C}$.

Samely as stable radical computation, once one can compute a stable irreducible decomposition for 0-dimensional parametric ideal $\mathcal{H}^e$ in $K(Y)[A, X\setminus Y]$, one can compute a stable irreducible decomposition by using stable extension-contraction like Lemma 6 and Lemma 7.

Definition 8 (Generic Element and Decomposition of Polynomial)

1. Generic Element (Element in Generic Position): *A polynomial* h *in* $K[A,X]$ *is a* stable generic element *with respect to* $\mathcal{H}$ *and its stable MSIS* Y *in a region* $\mathcal{C}$, *if* $\varphi_\alpha(h)$ *is a generic element with respect to* $\varphi_\alpha(\mathcal{I})^e$ *for any* $\alpha \in \mathcal{C}$. *We note that* $\varphi_\alpha(h)$ *is a generic element with respect to* $\varphi_\alpha(\mathcal{H})^e$ *if and only if the degree of its minimal polynomial coincides with the linear dimension of* $\varphi_\alpha(\mathcal{H})^e$.
2. Decomposition of a polynomial: *For a univariate polynomial* $h(z)$ *over* $K[A,Y]$, $\{h_1(y), \ldots, h_s(y)\} \subset L[Y,z]$, *where* L *is some residue class ring* $K[A,Z]/\mathcal{J}$ *factored by an ideal* $\mathcal{J}$ *of* $K[A,Z]$, *gives a* stable irreducible decomposition *of* $h(y)$ *in a region* $\mathcal{C}$, *if* $\varphi_{\alpha,\beta}(h_1(z)), \ldots, \varphi_{\alpha,\beta}(h_s(z))$ *are all distinct absolutely irreducible factors of* $\varphi_\alpha(h(z))$ *for any* $\alpha \in \mathcal{C}$ *and any zero* β *of the ideal* $\varphi_\alpha(\mathcal{J})$ *of* $\bar{K}[Z]$. *Here* $\varphi_{\alpha,\beta}$ *is a homomorphism from* $\bar{K}[A,Z,X]$ *to* $\bar{K}[X]$, *which is a natural extension of* φ_α *and defined as follows: For an element* $h(A,Z)$ *in* $\bar{K}[A,Z]$, $\varphi_{\alpha,\beta}(h(A,Z)) = h(\alpha,\beta)$.

We note that for any $\varphi_\alpha(\mathcal{H})$, there exists a generic element in a set $\mathcal{E}$ consisting of fixed linear sums of variables $x_i \in X \setminus Y$ whose cardinality is bounded by a function on n and the linear dimension (see [18]). Using this finite set $\mathcal{E}$ for finding generic elements gives the termination of our irreducible decomposition within finitely many steps.

Since a stable minimal polynomial of a polynomial h can be computed, we can compute a stable generic element h with respect to $\mathcal{I}^e$ by examining whether its stable minimal polynomial has the same degree as its stable linear dimension. Here, beforehand, we assume that a stable linear dimension of $\mathcal{I}^e$ is already computed. Also, as $\mathcal{H}$ is a stable radical of itself, $\varphi_\alpha(h)$ is always square-free. Thus, we have a cell decomposition $\mathcal{C} = \cup_i \mathcal{C}_i$ of finitely many cells $\mathcal{C}_i$ with a set of stable generic elements g_i such that $\varphi_\alpha(g_i)$ is a generic element of $\varphi_\alpha(\mathcal{H})$. Then, for each stable generic element g_i, we consider its stable minimal polynomial m_i which was already computed for the test of g_i.

Once we have a stable irreducible decomposition of m_i among each $\mathcal{C}_i$, that is, a cell decomposition $\mathcal{C}_i = \cup_j \mathcal{C}_{i,j}$ of finitely many cells $\mathcal{C}_{i,j}$ with a set of $\{h_1^{(i,j)}(y), \ldots, h_{s(i,j)}^{(i,j)}(y)\} \subset (K[A,Z]/\mathcal{J}_{i,j})[y]$, then we have the following irreducible decomposition of $\varphi_\alpha(\mathcal{H})$ for each α in $\mathcal{C}_{i,j}$ and each zero β of $\mathcal{J}_{i,j}$, corresponding to ordinary prime decomposition (see [2]).

$$\varphi_\alpha(\mathcal{H}) = \varphi_\alpha(\mathcal{H})^{ec} = \cap_{k=1}^{s(i,j)} (\varphi_{\alpha,\beta}(\mathcal{H} + \langle h_k^{(i,j)}(g_i)\rangle))^{ec}.$$

Thus, by additional stable extension-contraction computation which can be also executed in finitely many steps, we finally obtain a stable irreducible decomposition. For stable Gröbner bases of $(\mathcal{H}+\langle h_k^{(i,j)}(g_i)\rangle)^{ec}$, where each $\mathcal{H}+\langle h_k^{(i,j)}(g_i)\rangle$ is an ideal of $\bar{K}[A,Z,X]$, we can consider an ideal $\mathcal{H}+\langle h_k^{(i,j)}(g_i)\rangle+\mathcal{J}$ of $K[A,Z,X]$. We show the tractability of irreducible decomposition of a parametric polynomial $m_i(y)$ in the next section.

Theorem 2 *One can compute a stable irreducible decomposition.*

3.3 Decomposition of Parametric Polynomial

We show, in brief, that irreducible decomposition of a "parametric polynomial" can be done by *successive construction of parametric ideals.* (This is based on an idea in [8].) Here we call a polynomial in $K[A,X]$ a *parametric polynomial.*

To simplify our arguments, we assume that a parametric polynomial $f(z)$ is monic. We can show non-monic case by the same arguments, but we have a cell decomposition for "stable degree" of $f(z)$ beforehand. (For the minimal polynomial of a generic element, its cell decomposition is already done.)

Univariate case:
Consider a polynomial $f(z)$ over $K[A]$ and assume that $\varphi_\alpha(f(z))$ is square-free and its degree is n for any α in a given region $\mathcal{C}$. Then, for any α, $\varphi_\alpha(f(z))$ should be decomposed into n linear factors. Thus, we consider its decomposition $f(z) = g_1(z)\cdots g_n(z)$ by introducing new variables B for coefficients of $g_1(z), \ldots, g_n(z)$, where $\deg(g_i) = 1$. In more detail, let $f(z) = z^n + f_{n-1}(A)z^{n-1} + \cdots + f_0(A)$. We consider the following decomposition:

$$f = (z + b_1)(z + b_2)\cdots(z + b_n),$$

where $b_1, \ldots, b_n$ are newly introduced variables. Then, we have a system of algebraic equations $\{b_1 + \cdots + b_n = f_{n-1}(A), \ldots, b_1 \cdots b_n = f_0(A)\}$ which gives new ideal $\mathcal{J}$ of $K[A, B]$, where $B = \{b_1, \ldots, b_n\}$. Then, $h_1(z) = z + b_1, \ldots, h_n(z) = z + b_n$ gives a stable irreducible decomposition of $f(z)$.

General case:
Consider a polynomial $f(z)$ over $K[A, Y]$ for some $Y \subset X$. We also assume that $\varphi_\alpha(f(z))$ is square-free and its degree is n for any α in a given region $\mathcal{C}$. Let $f(z) = z^n + f_{n-1}(A)z^{n-1} + \cdots + f_0(A, Y)$. Then, we consider *all possible decomposition patterns:*

$$f(z) = g_1(z) \cdots g_s(z), \tag{1}$$

by introducing new variables $B = \{b_{1,e_{1,1}}, \ldots, b_{1,0}, \ldots, b_{s,e_{s,1}}, \ldots, b_{s,0}\}$ determined from the shape of f such that

$$\begin{aligned}
g_1(z) &= z^{d_1} + b_{1,e_{1,1}} y^{e_{1,1}} z^{d_1 - 1} + \cdots + b_{1,0}, \\
g_2(z) &= z^{d_2} + b_{2,e_{2,1}} y^{e_{2,1}} z^{d_2 - 1} + \cdots + b_{2,0}, \\
&\vdots \\
g_s(z) &= z^{d_s} + b_{s,e_{s,1}} y^{e_{s,1}} z^{d_s - 1} + \cdots + b_{s,0},
\end{aligned}$$

where $\deg(g_1) = d_1, \ldots, \deg(g_s) = d_s$ and $d_1 + \cdots + d_s = n$. We note that y^e denotes $y_1^{e_1} \cdots y_t^{e_t}$ for $Y = \{y_1, \ldots, y_t\}$ and $e = (e_1, \ldots, e_s)$. Then, we have a system of algebraic equations derived from Equation (1), which gives new ideal $\mathcal{J}_{d_1,\ldots,d_s}$ of $K[A, B]$. The possibility of the above decomposition can be reduced the non-triviality of $\mathcal{J}_{d_1,\ldots,d_s}$, and the solution for B can be expressed over some extension $L = K[A]/\mathcal{J}_{d_1,\ldots,d_s}$ of $K[A]$.

As each factorization pattern corresponds to a partition $(d_1, \ldots, d_s)$ of n, we introduce a complete order on the set of all partitions of n as follows: For two partition $D = (d_1, d_2, \ldots, d_s), E = (e_1, e_2, \ldots, e_t)$, $D \prec E$ if $s > t$. For the case $s = t$, any order is feasible. (For example, we may apply a lexicographic order.)

Beginning from the partition of the smallest order, that is, $(1, 1, \ldots, 1)$, and increasing its order, we search a region $\mathcal{C}_{d_1,\ldots,d_s}$ where $\mathcal{J}_{d_1,\ldots,d_s}$ has *stable non-triviality.* In this repetition, we have to exclude all regions already computed at each step. By this operation, for any α in a newly computed region $\mathcal{C}$, each zero β of $\varphi_\alpha(\mathcal{J}_{d_1,\ldots,d_s})$ gives irreducible factors. When the union of all computed regions covers $\mathcal{C}$, we can stop our search. As the number of partitions is bounded and stable non-triviality can be executed within finitely many steps, the termination of stable irreducible decomposition of f is guaranteed.

Theorem 3 *One can compute a stable irreducible decomposition of a parametric polynomial.*

Example 3 For $F(z, y) = z^2 - y^2 - ay - a$, where z, y are variables and a is a parameter. Adding new parameters $b_{1,1}, b_{1,0}, b_{2,1}, b_{2,0}$, we consider the factorization $F(z, y) = (z + b_{1,1}y + b_{1,0})(z + b_{2,1}y + b_{2,0})$, by which we derived a new parametric ideal $\mathcal{J}_{1,1}$ generated by $\{b_{1,1}+b_{2,1}, b_{1,0}+b_{2,0}, b_{1,1}b_{2,1}+1, b_{1,1}b_{2,0}+b_{1,0}b_{2,1}+a, b_{1,0}b_{2,0}+a\}$. Using an elimination order $a \prec\prec \{b_{1,1}, b_{1,0}, b_{2,1}, b_{2,0}\}$, we compute a Gröbner basis G of $\mathcal{J}_{1,1}$. Then we have $\{a^2-4a\} = G \cap \bar{K}[a]$ and conclude that $\varphi_\alpha(F(z,y))$ is decomposed to linear factors if and only if $\alpha = 0$ or $\alpha = 4$ by Lemma 1. Otherwise, $\varphi_\alpha(F(z,y))$ is absolutely irreducible.

Example 4 We consider another parametric ideal $\mathcal{I}_2$ derived from the spectral decomposition of $f_2(x) = x^6 + sx^2 + t$. (See Example 2.) Then, $\mathcal{I}_2 = \langle b_0^2 + t, b_1^2 - 2b_0b_2 - s, b_2^2 - 2b_1\rangle$ is radical in $\mathbf{C}[b_0, b_1, b_2, s, t]$ and has a stable MSIS $\{s\}$ for any $t \neq 0$. With respect to its extension $\mathcal{I}_2^e$, b_0, b_1, b_2 have stable minimal polynomials for any $t \neq 0$. Seeing stable square-free parts, it follows that for any $a \neq 0$, $\varphi_a(\mathcal{I}_2)$ is radical and $\sqrt{\varphi_0(\mathcal{I}_2)} = \langle b_0, b_1^2 - s, b_2^4 - 4s\rangle$. Also, with respect to $\mathcal{I}_2^e$, b_2 is a stable generic element for any $t \neq 0$ and its stable minimal polynomial is $m_2(x) = x^8 - 8sx^4 + 64tx^2 + 16s^2$. As m_2 is a quadratic polynomial in s, we can compute its stable irreducible decomposition by similar manner as in Example 3: For any $a \neq 0$, $\varphi_a(m_2(x)) = (x^4 - 8x\sqrt{-a} - 4s)(x^4 + 8x\sqrt{-a} - 4s)$ gives the irreducible decomposition. In this case, considering a residue class ring $L = \mathbf{Q}[t,z]/\langle z^2 + 64t\rangle$, we can compute the parametric decomposition of $m_2(x)$.

Finally, we obtain a stable irreducible decomposition of $\mathcal{I}_2$ as follows: For any $a \neq 0$, $\varphi_a(\mathcal{I}_2) = \langle b_0 - \sqrt{-a}, 2b_1 - b_2^2, b_2^4 - 8\sqrt{-a}b_2 - 4s\rangle \cap \langle b_0 + \sqrt{-a}, 2b_1 - b_2^2, b_2^4 + 8\sqrt{-a}b_2 - 4s\rangle$ gives the irreducible decomposition, and for $a = 0$, $\sqrt{\varphi_0(\mathcal{I}_2)} = \langle b0, 2b_1 - b_2^2, b_2^4 - 4s\rangle$ gives the irreducible decomposition.

Remark 2 Although we show certain theoretical tractability of stable irreducible decomposition over $\bar{K}$, "stable" primary/prime decomposition over K seems much harder *even in theory*. When K is the rational number field, this corresponds to finding rational points over algebraic surfaces or curves. Thus, it seems impossible to give a general method for this subject. However, when $\mathcal{J}$ or $\mathcal{J}_{d_1,\ldots,d_s}$ is 0-dimensional like as F in Example 3, all rational zero can be computed exactly. In this case, we have a chance to compute a stable prime decomposition over $\mathbf{Q}$.

4 Concluding Remarks

In this paper, we show that a stable radical and a stable irreducible decomposition are computationally tractable by repeating stable Gröbner bases computations for various ideals. However, as such repetition of stable Gröbner bases computations shall make huge computational trees of cell decompositions and combinatorial explosion may occur, it seems very hard for its actual computation without practical improvements. Thus, in the next step, we will give more precise and efficient procedures on these parametric decompositions and examine those efficiency/ability by complexity analysis and experiments on real computer. For controlling growing trees of cell decompositions, we may refer to studies [7,11]. Also for practical implementation of decomposition of parametric polynomials, much works shall be done to find efficient strategies on factorization patterns and careful choices of B.

Moreover, as to mulitiplicies of isolated prime divisors, we may apply "effective localization" [9]. However, stable primary/prime decomposition over $K[X]$ is still hard problem in general settings. For this problem, a promising approach may be to find wider classes where certain stabilities over K can be assumed.

References

1. Becker, T. (1994). On Gröbner bases under specialization. Applicable Algebra in Engineering, Communication and Computing **5**, 1 – 8.
2. Becker T., Weispfenning V. (1993). Gröbner Bases. GTM **141**, Springer-Verlag.
3. Gao X., Chou S. (1992). Solving parametric algebraic systems. In Proceedings of ISSAC 1992, pages 335-341, ACM Press.
4. Gianni P., Trager B., Zacharias G. (1988). Gröbner base and primary decomposition of polynomial ideals. J. Symb. Comp. **6**, 149-167.
5. Kalkbrener, M. (1997). On the stability of Gröbner bases under specializations. J. Symb. Comp. **24**, 51 – 58.
6. Kanno M., Anai H., Yokoyama K. (2006) On the relationship between the sum of roots with positive real parts and polynomial spectral factorization. to appear in the Proceedings of Sixth International Conference on Numerical Methods and Applications NM & A'06.
7. Montes, A. (2002). A new algorithm for discussing Gröbner bases with parameters. J. Symb. Comp. **33**, 183 – 208.
8. Sen, H., Wang, D. (1986). Fast factorization of polynomials over rational number field or its extension fields. Kexue Tongbao **31**, 150-156.
9. Shimoyama T., Yokoyama K. (1996). Localization and primary decomposition of polynomial ideals. J. Symb. Comp. **22**, 247-277.
10. Suzuki, A., Sato, Y. (2003). An alternative approach to comprehensive Gröbner bases. J. Symb. Comp. **36**, 649 – 667.
11. Suzuki, A., Sato, Y. (2006). A simple algorithm to compute comprehensive Gröbner bases using Gröbner bases. to appear in the Proceedings of ISSAC 2006. (Programs can be down-loadable from http://kurt.scitech.kobe-u.ac.jp/ sakira/CGBusingGB/)
12. Wang, D. (2005). The projection property of regular systems and its application to solving parametric polynomial systems. In Algorithmic Algebra and Logic — Proceedings of the A3L 2005, Herstellung und Verlag, Norderstedt, pp. 269–274.
13. Weispfenning, V. (1992). Comprehensive Gröbner bases, J. Symb. Comp. **14**, 1 – 29.
14. Weispfenning, V. (2003). Canonical Comprehensive Gröbner bases. J. Symb. Comp. **36**, 669 – 683.
15. Weispfenning, V. (2004) Gröbner bases for binomials with parametric exponents. In Proceedings of CASC 2004, TUM, pp. 467 – 478.
16. Yokoyama, K. (2004). On systems of algebraic equations with parametric exponents. In Proceedings of ISSAC 2004, ACM Press, New York, pp. 312–317.
17. Yokoyama, K. (2005). On systems of algebraic equations with parametric exponents II. Presented at the 2005 Conference on Applications of Computer Algebra (Nara, Japan, July 31 – August 3, 2005) and submitted for publication.
18. Yokoyama, K., Noro, M., Takeshima, T. (1992). Solution of systems of algebraic equations and linear maps on residue class rings. J. Symb. Comp. **14**, 399-417.

On the GAP Package *sgpviz*

Manuel Delgado[1] and José Morais[2,⋆]

[1]Departamento de Matemática e CMUP, Universidade do Porto
Rua do Campo Alegre, 687, 4169-007 Porto, Portugal
mdelgado@fc.up.pt,
[2]jjoao@netcabo.pt

Abstract. This is a brief description of the GAP package *sgpviz*, a package designed to visualize finite semigroups through their $\mathcal{D}$-classes or Cayley graphs, as well as to make friendlier the usage of GAP when dealing with finite semigroups.

1 Introduction

GAP [4] is a system for computation in discrete abstract algebra. The name stands for Groups, Algorithms and Programming and was chosen to reflect the original aim of the system. Presently the system has grown in several other directions, in part due to the possibility of integrating additional packages. GAP provides a programming language, thousands of functions implementing algorithms written in GAP and also large data libraries of algebraic objects. It is used in research as well as in teaching. The GAP system is free of charge and it is open, that is, one may examine the code. A user can write his own programs in GAP, and use them in just the same way as the functions which form part of the system. In this sense, the system is extensible. The GAP packages are self-contained extensions having, in particular, its own documentation.

The aim of this note is to illustrate the usage of the GAP package *sgpviz* [3]. The interested reader is encouraged to consult also the manual, which contains much more information and is part of the package. Besides being part of the package, its `html` version can also be consulted directly on the web: `http://www.gap-system.org/Manuals/pkg/sgpviz/doc/manual.html`.

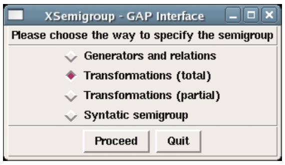
Since interaction with GAP is made through a text terminal, in order to ease some of this interaction when one works with finite semigroups and with finite automata, the GAP package *sgpviz* provides a Tcl/Tk (`http://www.tcl.tk/`) graphical interface to specify finite semigroups and automata. The next example of a GAP session illustrates how to launch the graphical interfaces.

```
gap> XAutomaton();
gap> XSemigroup();
```

⋆ Both authors gratefully acknowledge support of FCT and the POCTI program through CMUP.

© Springer-Verlag Berlin Heidelberg 2006

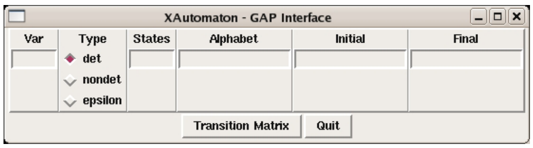
New windows like those in the next picture are then opened.



XSemigroup - GAP Interface

Please choose the way to specify the semigroup

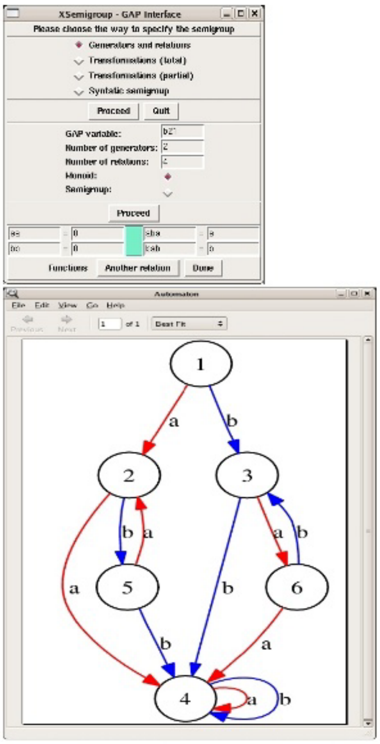
- Generators and relations
- [✓ Transformations (total)]
- Transformations (partial)
- Syntatic semigroup



XAutomaton - GAP Interface

Var	Type	States	Alphabet	Initial	Final
	[✓ det] nondet epsilon				

These windows may then be used to specify the objects and, by using the "Functions" button, some functions can be applied:

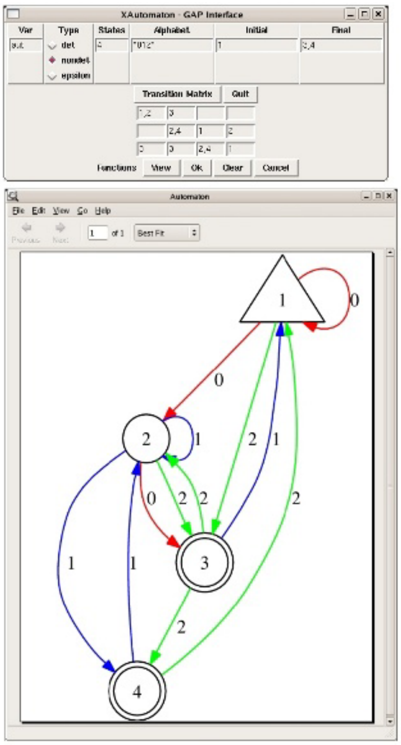


XSemigroup - GAP Interface

Please choose the way to specify the semigroup

- [✓ Generators and relations]
- Transformations (total)
- Transformations (partial)
- Syntatic semigroup

GAP variable: b21
Number of generators: 2
Number of relations: 4
Monoid: [✓]
Semigroup:



XAutomaton - GAP Interface

Var	Type	States	Alphabet	Initial	Final
aut	det [✓ nondet] epsilon	4	"012"	1	3,4

1,2	3		
	2,4	1	2
3	3	2,4	1

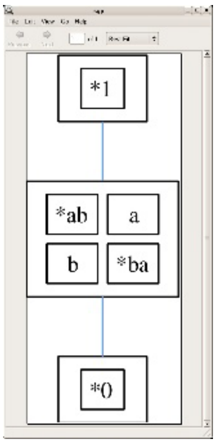
In the GAP session, after specifying the semigroup `b21` above, one gets the following, by pressing the button "Done":

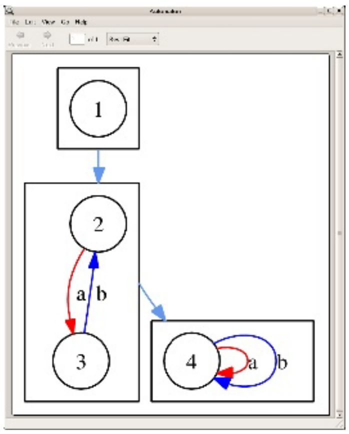
```
<free monoid on the generators [ a, b ]> gap>
a:=GeneratorsOfMonoid( fxsgp )[ 1 ];; a gap>
b:=GeneratorsOfMonoid( fxsgp )[ 2 ];; b gap>
rxsgp:=[[a*a*a,a*a],[a*a*a,a*a],[a*a*b,a*a],[b*a*a,a*a],[a*b*a,a],
[b*b*a,b*b],[a*b*b,b*b],[b*b*b,b*b],[b*b*b,b*b],[b*a*b,b]];; [[
a^3, a^2 ],[ a^3, a^2 ],[ a^2*b, a^2 ],[ b*a^2, a^2 ],[ a*b*a, a
],
  [ b^2*a, b^2 ],[ a*b^2, b^2 ],[ b^3, b^2 ],[ b^3, b^2 ],[ b*a*b, b
    ]]
gap> b21:=fxsgp/rxsgp;
<fp monoid on the generators [ a, b ]>
```

As should already be clear, the package under consideration in this note allows to draw Cayley graphs of finite semigroups. But the $\mathcal{D}$-Classes of finite semigroups, as well as Schützenberger graphs of finite inverse semigroups may also be drawn. For these drawings it is used Graphviz, a Graph Visualization Software (`http://www.graphviz.org/`). It should be remarked here that the *sgpviz* package needs the GAP package *automata* [2]: Cayley graphs are treated as automata, without distinguished states. The drawings can be achieved by choosing the appropriate function from the Tcl/Tk window used to specify the object, as happened in the above picture, or by using the GAP command line. Continuing our GAP session:

```
gap> b21;
<fp monoid on the generators [ a, b ]>
gap> DrawDClasses(b21);
I will work with an isomorphic transformation semigroup instead
Displaying file: /tmp/tmp.KsAcFt/semigroup.dot.ps
gap> DrawSchutzenbergerGraphs(b21);
I will work with an isomorphic transformation semigroup instead
Displaying file: /tmp/tmp.F4E1N2/schutzenbergergraphs.dot.ps
```

The following windows are popped up:





Although it does not become evident from the pictures shown, we have to remark that the package contains implementations of some useful algorithms. For instance, an algorithm due to Graham [1] to normalize the structure matrix of a Rees matrix semigroup is used and as a consequence the idempotents in the $\mathcal{D}$-Classes can be displayed in Graham blocks.

References

1. R.L. Graham, *On Finite 0-Simple Semigroups and Graph Theory*, Math. Syst. Theory, **2** (1968) 325–339.
2. M. Delgado, S. Linton and J. Morais, *Automata: a refereed* GAP *[4] package on finite state automata.* (`http://www.gap-system.org/Packages/automata.html`)

3. M. Delgado e J. Morais, *SgpViz: a* GAP *[4] package for finite semigroup visualization.* (`http://www.gap-system.org/Packages/sgpviz.html`)
4. The GAP Group. *GAP– Groups, Algorithms, and Programming, Version 4.4*, 2004. (`http://www.gap-system.org`)

Making Research on Symmetric Functions with MuPAD-Combinat

Francois Descouens

Laboratoire d'Informatique de l'Institut Gaspard Monge,
Université de Marne-la-Vallée, 77454 Marne-la-Vallée Cedex 2, France
`francois.descouens@univ-mlv.fr`
`http://www-igm.univ-mlv.fr/~descouen`

Abstract. We report on the 2005 AIM workshop "Generalized Kostka Polynomials", which gathered 20 researchers in the active area of q,t-analogues of symmetric functions. Our goal is to present a typical use-case of the open source package MuPAD-Combinat in a research environment.

1 Introduction

The project `MuPAD-Combinat` (see `http://mupad-combinat.sf.net/` and [4]), born in spring 2001 under the leadership of F. Hivert and N. Thiéry, is an open source package for making algebraic combinatorics using the computer algebra system MuPAD (see `www.mupad.de` and [3] for more details). The main goal of this package is to bring an open source flexible and easily extensible toolbox for checking conjectures in a short programming time. This package contains a large collection of tools as implementations of classical combinatorial objects (partitions, Young tableaux, trees, ...), computations in combinatorial Hopf algebras (in particular symmetric functions), manipulations of graphs, automata, ...

The study of symmetric functions is a major historical field in algebraic combinatorics ([8]) and some people are mainly interested in generalizations of Kostka polynomials. In section 2, we give briefly mathematical definitions about different objects we work on. We illustrate how a computer algebra system which contains classical combinatorial objects, evolutive implementation of symmetric functions and easy way to incorporate program written in C++ can be used for our research.

In section 3, we explain the design of our implementation of symmetric functions and some of our technical choices. We also give some examples of computations and show some advanced functionalities as adding new bases on the fly (using different characterizations) or implementing new operators on symmetric functions. Finally, in section 4 we describe how we incorporate programs into a coherent design using MAPITL library. The technical concepts used in section 3 and 4 are essentially classical, but we want to stress on their integration into the package in order to have a powerfull tool for making efficient. The difficulty is to find the appropriate combination that yields an intuitive yet flexible and

© Springer-Verlag Berlin Heidelberg 2006

powerful research tool. We show that our choices are promising in a real situation of collaboration at a workshop on generalized Kostka polynomials in Palo Alto, California organized by the American Institute of Mathematics in July 2005 (see `http://www.aimath.org/WWN/kostka/` for more informations on this event).

2 Symmetric Functions and Kostka Polynomials

Basic Definitions. A symmetric polynomial in variables $X = \{x_1, \dots, x_n\}$ is a polynomial in X invariant under permutations of variables. When is infinite, we call symmetric function such a polynomial. The set of symmetric functions with coefficients in $\mathbb{C}(t)$, denoted Λ_t, is a graded algebra with respect to the degree of polynomials, i.e

$$\Lambda_t = \oplus_{n \geq 0} \Lambda_t^n .$$

For all $n \geq 0$, the dimension of Λ_t^n is the number of partitions of n (a partition of a positive integer n, written $\lambda \vdash n$, is a decreasing sequence of positive integers with sum n). One main basis of this algebra is constituted with monomial functions defined for all partitions by $\lambda = (\lambda_1, \dots, \lambda_n)$, by

$$m_\lambda = \sum_{v \in O(\lambda)} x_1^{v_1} \dots x_n^{v_n},$$

where $O(\lambda)$ represents the set of all the permutations of λ. Another interesting basis consists of the symmetric powersums, defined for all partitions λ by

$$p_\lambda(X) = p_{\lambda_1} \dots p_{\lambda_n} \quad \text{where} \quad p_{\lambda_i} = \sum_{j=1}^{n} x_j^{\lambda_i}.$$

A scalar product on Λ_t can be uniquely defined by

$$(p_\lambda, p_\mu) = \delta_{\lambda,\mu} \prod_{i \geq 1} (m_i)! \; i^{m_i(\lambda)},$$

where $m_i(\lambda)$ represents the multiplicity of part i in partition λ. Applying Gram-Schmidt process of orthonormalization on the monomial basis with respect to this scalar product yields us the basis of Schur functions $(s_\lambda)_\lambda$. These functions are intensively studied (from an algebraic and combinatorial point of view) and one of the beautiful results is the Littlewood-Richardson rule, a combinatorial interpretation of the product of two Schur functions.

Kostka Polynomials. One can introduce a t-deformation of the previous scalar product by

$$(p_\lambda, p_\mu)_t = \delta_{\lambda,\mu} \prod_{i \geq 1} (m_i)! \; i^{m_i(\lambda)} \prod_{i=1}^{l(\lambda)} \frac{1}{1 - t^{\lambda_i}}.$$

The orthogonalization of the monomial basis with respect to this scalar product defines the Hall-Littlewood functions $P_\lambda(X;t)$. There exist two other families of Hall-Littlewood functions: on the one hand, the $Q_\lambda(X;t)$ which are the dual elements of $P_\lambda(X;t)$ with respect to the scalar product $(\ ,\ )_t$ and on the other hand $Q'_\lambda(X;t)$ which are the dual elements of $P_\lambda(X;t)$ with respect to $(\ ,\ )$. The expansion of the Q'_λ on Schur functions is an algebraic way to define the Kostka polynomials $K_{\lambda,\mu}(t)$

$$Q'_\lambda(X;t) = \sum_{\mu \vdash |\lambda|} K_{\lambda,\mu}(t) s_\mu.$$

These polynomials have also a combinatorial interpretation, using the charge on semi-standard Young tableaux in [7], or the rigged configurations introduced by Kerov, Kirillov and Reshetikhin in [5].

Generalizations of Kostka Polynomials. Using k-ribbon tableaux introduced by Lascoux, Leclerc and Thibon in [6], we define for each positive integer k and each partition a particular symmetric functions $H_\lambda^{(k)}(X;t)$. Their expansion on Schur functions gives us a way to define an increasing filtration of Kostka polynomials $K_{\lambda,\mu}^{(k)}(t)$

$$H_\lambda^{(k)}(X;t) = \sum_{\mu \vdash |\lambda|} K_{\lambda,\mu}^{(k)}(t) s_\mu.$$

The expansion of $K_{\lambda,\mu}^{(k)}(t)$ on the monomial basis is a way to define unrestricted generalized Kostka polynomials. In order to generalize Kostka polynomials, there exist other combinatorial ways (with unrestricted rigged configurations recently introduced by L. Deka and A. Schilling in [1]) and algebraic ways (using the theory of crystal bases for quantum groups of type A_n) to generalize Kostka polynomials. Other generalizations are given by M. Zabrocki using creation operators in [16] and by L. Lapointe and J. Morse introducing a t-deformation of k-Schur functions in [9]. An interesting problem, explained in [10], is to show that these generalizations coincide in some particular cases.

3 Implementation of Symmetric Functions

3.1 Design Goals

Symmetric functions can be represented in many different ways, and in particular in different basis (powersum,elementary, Schur, monomials,...). As usual in computer science, it is essential at each point to use the appropriate representation, both for efficiency and interpretation of the results. What make the situation specific is the number of those representations. In particular, it is neither practical nor sometimes possible to implement explicitely all conversions. Instead we want to be able to only implement a few and deduce the others by compositions or linear tranformations such as inversion or transposition. Furthermore to thratend, the user and also programmers should not need to know which conversion are implemented because this information is too volatile. For example, the addition of symmetric function not given in the same basis is possible

```
>> S::s([2,1]) + S::QP([2,1]) + S::p([2,1]);

            (t + 4) m[1, 1, 1] + (t + 1) m[3] + (t + 3) m[2, 1]
```

The monomial basis has been choosen by the system because it minimizes the number conversions (the cost of the conversion is not taken into account). The same process accurs for making conversion between two bases as the expansion of the Hall-Littlewood function Q'_{311} on the monomial basis

```
  >> S::m(S::QP([2,1]));

            (t + 2) m[1, 1, 1] + (t + 1) m[2, 1] + t m[3]
```

We also consider operators on symmetric functions. In most cases, they are easier to define (and consequently to implement) on one of the bases but not on all of them. Consequently, the system uses implicit conversions between bases in order to apply operators on any given basis. The main technologies used are linear algebra and overloading mechanisms which are not essentially new, but a great effort is made in order to make manipulation intuitive.

3.2 General Overview of the Implementation

In our design, for each basis of symmetric functions there is a domain (in the category `Cat::GradedHopfAlgebraWithBasis`) which represents the space of symmetric functions expanded on this basis. For example, here is the example of implementation of the complete basis

```
domain SymT::complete(R: DOM_DOMAIN)
    inherits SymT::common(R);
    category Cat::GradedHopfAlgebraWithBasis(R), Cat::CommutativeRing;
    info_str := "Domain for symmetric functions expanded on complete basis";

    basisName := hold(h);

    // Implementation of multiplication (complete basis is a multiplicative basis)
    mult2Basis := dom::term@revert@sort@_concat;
    ...
end_domain:
```

The domain `Sym` of symmetric functions, representing symmetric functions in whatever representation, is in category `Cat::HopfAlgebraWith SeveralBases`. This category helps us in defining implicit conversions between bases

```
domain Sym(R=Dom::ExpressionField())
    inherits Dom::BaseDomain;
    category Cat::HopfAlgebraWithSeveralBases(R), Cat::CommutativeRing;
    ...

    // Each domain corresponding to a basis is declared
    h := SymT::complete(dom::coeffRing, Options);

    // Implementation of different conversions between bases (stored in a table)

    basisChangesBasis :=
    table(
          ...
          // Explicit combinatorial conversions
          (dom::QP, dom::s)= (part ->(_plus(combinat::tableaux::kostkaPol(mu, part, dom::vHL)
                                            * dom::s(mu) $ mu in
                                              combinat::partitions::list(_plus(op(part))))),
          ...
          (dom::McdP, dom::m) = (part ->((dom::GramSchmidt(dom::m, _plus(op(part)),
                                          dom::scalartq))[op(part)])),
          ...
          // Dual conversions and inverse conversions
          (dom::s, dom::QP) = dom::invertBasisChange(dom::QP, dom::s),
          (dom::s, dom::m) = dom::transposeBasisChange(dom::h, dom::s, dom::s, dom::m),
         )
```

Note that only some conversions are implemented in this table; the overloading mechanism is in charge of finding the shortest number of intermediate conversions needed (it doesn't take into account the cost of each conversion). In order to use (q, t)-deformation of symmetric function, we can declare

```
>> S:=examples::SymmetricFunctions(Dom::ExpressionFieldWithDegreeOneElements([t,q]),
                                                                     vHL=t, vMcd=q);
```

Adding new bases on the fly. In order to add new basis on the fly, we define a generic domain.

```
  domain SymT::NewBasis(R: DOM_DOMAIN, DomName: DOM_STRING)
      inherits SymT::common(R);
      category Cat::GradedHopfAlgebraWithBasis(R), Cat::CommutativeRing;
      info_str := "Domain for symmetric functions expanded on a new added basis";

      basisName := text2expr(DomName);

  end_domain:
```

In order to add a new basis named E for example, we can use

```
  >> B := S::newBasis(S::coeffRing, ‘‘E’’);
```

and we define the change of bases we want. Let suppose that the change of basis between B and monomial basis corresponds to the function `testChange`

```
  >> S::declareBasisChangeBasis((B, dom::m, testChange);

  >> S::declareBasisChangeBasis(dom::m, B, dom::invertBasisChange(B, dom::m));
```

Adding New Operators. If we want to define new operators on symmetric functions, we only have to define their action on a particular basis, of course for efficiency. First, we declare our operators as an overloaded operator

```
   >> newOp := operators::overloaded(
          (x,y)->error("Don't know how to compute the newOp on ".expr2text(domtype(x))),
                       Name="newOp");
```

and by assuming that the action on the Schur basis is given by a function `f`, we declare

```
 >>  operators::overloaded::declareSignature(
        S::newOp, [S::s, DOM_INT],
        S::s::moduleMorphism(f, dom::s));
```

and you can apply this operator on any basis due to the overloading mechanism.

4 Integration of Others Programs Written in C++

In July 2005, the American Institute of Mathematics organized in Palo Alto, California, a workshop on the generalized Kostka polynomials under the leadership of A. Schilling and M. Vazirani. During problem sessions, MuPAD-Combinat was put to use for testing conjectures. As usual in algebraic combinatorics, computations required the combination of preexisting combinatorial and algebraic functionnalities (as provided by MuPAD-Combinat) with new combinatorial features (namely a highly technical bijection between k-tuples of Young Tableaux and unrestricted rigged configurations). Thanks to a dynamic module we could reuse a pre-existing robust C++ implementation of this bijection written by L. Deka; this allowed us to start manipulating large examples quickly. In general, dynamic modules permit us to reuse C++ code with two goals in mind: to avoid reimplementing nontrivial and tested code, and to get quicker computations than in pure MuPAD language. In section 4.1, we describe the implementation of combinatorial objects in MuPAD, and in section 4.2 we present our technical choices for a seamless integration of a combinatorial bijection implemented in C++.

4.1 Implementation of Combinatorial Objects in MuPAD-Combinat

We implement each combinatorial class as a domain in the MuPAD category `Cat::CombinatorialClassWith2DBoxedRepresentation`.

This category provides, among other things, a pretty printing method using ASCII characters. Let us illustrate this design in the case of skew riggings implemented in the domain `combinat::skewRiggings`.

```
domain combinat::skewRiggings
    inherits Dom::BaseDomain;
    category Cat::CombinatorialClassWith2DBoxedRepresentation;
    axiom    Ax::canonicalRep;

    info_str := "Combinatorial class for rigged skew partitions";
...
end_domain:
```

We implement the constructor `combinat::skewRiggings::new` which builds an object from the list of its operands:

```
>> a:=combinat::skewRiggings([[], [[[0], [0, 1]], [[""], [0, 1]]]]);

                              +---+
                              | 0 | 0   1
                              +---+
                              |   | 0   1
                              +---+
```

The first element of the internal representation is the type

```
>> a[0];

                         combinat::skewRiggings
```

Ribbon rigged configurations are particular sequences of skew riggings, implemented as plain lists of typed MuPAD objects. We implement them in the following domain

```
domain combinat::riggedConfigurations::RcRibbonsTableaux
    inherits Dom::BaseDomain;
    category Cat::CombinatorialClassWith2DBoxedRepresentation,
    // Elements of this domain are represented using a plain MuPAD data structure
             Cat::FacadeDomain(DOM_LIST);

    info_str := "Combinatorial class for rigged configurations";
...
end_domain:
```

Here is an example of the bijection applied on the set of all 3-ribbon tableaux of shape (432) and evaluation (111)

```
>> rc := map(combinat::ribbonsTableaux::list([4,3,2],[1,1,1],3),
       combinat::riggedConfigurations::RcRibbonsTableaux::fromRibbonTableau);

-- --                                   +---+     --
|  |                                    |   |      |
|  |  +---+          +---+---+          +---+---+  |
|  |  | 1 | 0   1  , | 0 | 0 | 0   0  , |   |   |  |,
|  -- +---+          +---+---+          +---+---+ --
--

 --                                   +---+     --
 |                                    |   |      |
 |  +---+          +---+---+          +---+---+  |
 |  | 0 | 0   1  , | 0 | 0 | 0   0  , |   |   |  |,
 -- +---+          +---+---+          +---+---+ --

 --                               +---+     -- --
 |                 +---+          |   |      |  |
 |  +---+          | 0 | 0   0    +---+---+  |  |
 |  | 0 | 0   0  , +---+        , |   |   |  |  |
 |  +---+          |   | 0   1    +---+---+  |  |
 --                +---+                    -- --

>> a:= rc[1];

           --                                   +---+     --
           |                                    |   |      |
           |  +---+          +---+---+          +---+---+  |
           |  | 1 | 0   1  , | 0 | 0 | 0   0  , |   |   |  |
           -- +---+          +---+---+          +---+---+ --
>> op(a)[2][0];

                        combinat::skewRiggings
```

L. Deka and A. Schilling introduced in [1] a new kind of rigged configurations, namely the unrestricted ones. They were implemented as an independent C++ program (file: `FromOneCrystalPath.cc`, headers: `FromOneCrystalPath.h`). On each rigged configurations (ribbons one and unrestricted one) we can compute a statistic. The interesting question is to find, in a special case, a bijection which preserves the statistic between these two kinds of rigged configurations. In order to manipulate these two objects in a single program, we decided in collaboration with A. Schilling and L. Deka, to also integrate this C++ program into MuPAD-Combinat. In order to make the integration of this version of rigged configurations in a transparent way for the user, we kept the same design we used for ribbon rigged configurations.

4.2 Implementation of Combinatorial Objects in a Dynamic Module Using the MAPITL Library

We describe now the integration of the previous C++ program using MAPITL library (MuPAD Application Programming Interface Template Library (included in `MuPAD-Combinat`). This library provides

- Wrappers to use MuPAD lists as standard containers
- Easy C++ $\longleftrightarrow$ MuPAD conversions with one single overloaded template for each directions:
  - C++ object $\longrightarrow$ MuPAD object: `CtoM(c)`
  - MuPAD object $\longrightarrow$ C++ object: `MtoC(c)`

This includes conversions to/from containers and recursive containers.

The C++ program computes the rigged configurations corresponding to a list of Young tableaux also called path which is denoted by the class `path_class` . We want to call from MuPAD `void path_class::build_rigged_for_path` using the procedure `combinat::RiggedConfigurations::newRiggedConfigurations`. The building of the interaction is divided into three steps which are realized in the file `RiggedConfigurationsPaths.mcc`

- convert a list of Young Tableaux in MuPAD to a C++ object of the class path_class
- call the function `void path_class::build_rigged_for_path`
- convert the result into a MuPAD object of type `DOM_LIST`

Original files are `FromOneCrystalPath.cc` containing declarations of different classes. We now explain some parts of the file `RiggedConfigurationsPaths.mcc` which is entirely given in appendix. The beginning of the file is the inclusion of the header file of the independent program and MAPITL

```
#include<vector>
#include "FromOneCrystalPath.h"
#include "MAPITL.h"
#include "MAPITL_tmpl.h"
```

Conversion MuPAD to C++

```
  MFUNC( newRiggedConfigurations, MCnop )
  {
    MFnargsCheck(3);
    MFargCheck(1, DOM_INT);
    MFargCheck(2, DOM_INT);
    MFargCheck(3, DOM_LIST);

    n = MtoC<int>(MFarg(1));
    int path_len  = MtoC<int>(MFarg(2));
    Cell input(MFarg(3));
    ...
    Cell *toto = input.toArray<Cell>();
    ...
}
```

`MFargCheck` is the type checking of MuPAD arguments and `MtoC` permits to convert MuPAD objects into C++ ones. After we initialize an object of class `path_class` with values contained in `toto`

```
  path_class*  input_path = new path_class(path_len)
  for(long a=0; a < input.size(); a++)
    {
      ...
    }
```

Using the original function in C++. We can call now the original function in order to compute the bijection

```
  input_path->build_rigged_for_path()
```

Conversion C++ to MuPAD. Next we create a C++ vector which contained operands of the corresponding into MuPAD object

```
  for(i=0; i<n; i++)
   {
      vector<int> yyy;
      j = 0;
      while(input_path->rigged[i][0][j] != UNUSED)
      {
          yyy.push_back(input_path->rigged[i][0][j]);
          yyy.push_back(input_path->rigged[i][2][j]);
          yyy.push_back(input_path->rigged[i][1][j]);
          j++;
      }
      res[i] = yyy;
   }
```

and we return the MuPAD list

```
MFreturn(Cell(res));
```

4.3 Compilation of the Dynamic Module

The next step is the compilation of the `mcc` file giving us the dynamic module `RiggedConfigurationsPaths.mdm`. We have to load this module from MuPAD using the function

```
  combinat::RiggedConfigurationsPaths :=
  proc()
    save RiggedConfigurationsPaths;
  begin
    if module::which("RiggedConfigurationsPaths") = FAIL then
      userinfo(1, "Dynamic module RiggedConfigurationsPaths not available"):
      RiggedConfigurationsPaths := FAIL;
    else
      if traperror(module("RiggedConfigurationsPaths")) <> 0 then
         warning("Error loading the dynamic module RiggedConfigurationsPaths:");
         lasterror();
      end_if;
    end_if;
    RiggedConfigurationsPaths;
end_proc():
```

We create the domain `combinat::riggedConfigurations::RcPathsEnergy` and procedure `combinat::riggedConfigurations::RcPathsEnergy::fromOne Path` computes the bijection by calling the function implemented in the dynamic module `combinat::RiggedConfigurations::newRiggedConfigurations`

```
 >> a:=[combinat::tableaux([[3,3]]),combinat::tableaux([[2,2]]),combinat::tableaux([[1,1]])];

                    -- +---+---+  +---+---+  +---+---+ --
                    |  | 3 | 3 |, | 2 | 2 |, | 1 | 1 |  |
                    -- +---+---+  +---+---+  +---+---+ --
```

We compute the bijection

```
>> rc := combinat::riggedConfigurations::RcPathsEnergy::fromOnePath(a);

                 -- +---+---+                        --
                 |  |   | 0 |    0    +---+---+        |
                 |  +---+---+       , |   | 0 |    0   |
                 |  |   | 0 |    0    +---+---+        |
                 -- +---+---+                        --

>> op(rc[1])[0]

                 combinat::skewRiggings
```

The object computed using the C++ program is interfacing in a transparent way for the user who can use other MuPAD functionalities on the previous result.

Acknowledgment. The author wants to thank A. Schilling and L. Deka for their collaboration on rigged configurations, M. Zabroki for its code on creating operators, and F. Hivert and N. Thiery for their support in the project MuPAD-Combinat.

References

1. L. Deka and A. Schilling, *New fermionic formula for unrestricted Kostka polynomials*, Journal of Combinatorial Theory, Series A, to appear (math.CO/0509194).
2. F. Descouens, *A generating algorithm for ribbon tableaux*, Journal of Automata, Languages and Combinatorics, to appear.
3. B. Fuchssteiner and al, *MuPAD User's Manual - MuPAD Version 1.2.2.*, John Wiley and sons, Chichester, New York, first edition, march 1996. includes a CD for Apple Macintosh and UNIX.
4. F. Hivert and N. Thiéry, *MuPAD-Combinat, an open-source package for research in algebraic combinatorics*, Sém. Lothar. Combin. **51** (2004), 70p. (electronic). `http://mupad-combinat.sourceforge.net/`
5. S. Kerov, A. Kirillov and N. Yu. Reshetikhin, *Combinatorics, the Bethe ansatz and representations of Symmetric group*, J. Soviet Math **41** (1988) no.2, 916-924.
6. A. Lascoux, B. Leclerc and J.-Y. Thibon, *Ribbon tableaux, Hall-Littlewood functions, quantum affine algebras and unipotent varieties*, Journal of Mathematical Physics **38** (1997), 1041-1068.
7. A. Lascoux and M.P. Schützenberger, *Sur une conjecture de H.O. Foulkes*, C.R Acad. Sci. Paris **288**(1979), 95-98.
8. I.G. Macdonald, *Symmetric functions and Hall polynomials*, 2nd ed., Oxford University Press, 1995.
9. L. Lapointe and J. Morse, *Schur function identities, their t-analogs, and k-Schur irreducibility*, Advances in Math (2003).
10. A. Schilling, *X=M Theorem: Fermionic formulas and rigged configurations under review* preprint (math.QA/0512161).
11. A. Schilling, *Crystal structure on rigged configurations* IMRN, to appear (math.QA/0508107).

12. A. Schilling, *q-Supernomial coefficients: From riggings to ribbons*, MathPhys Odyssey 2001, M. Kashiwara and T. Miwa (eds.), Birkhaeuser Boston, Cambridge, MA, 2002, pp. 437-454 (math.CO/0107214).
13. J. Stembridge, *Package SF, Posets, Coxeter/Weyl*, `http://www.math.lsa.umich.edu/~jrs/maple.html`.
14. Symmetrica , `http://www.mathe2.uni-bayreuth.de/axel/symneu_engl.html`.
15. S. Veigneau, *ACE, an Algebraic Combinatorics Environment for the computer algebra system MAPLE: User's Reference Manual, Version 3.0*, Report 98-11, IGM, 1998. `http://www-igm.univ-mlv.fr/~ace/ACE/3.0/ACE.html`
16. M. Zabrocki, *Vertex operators for standard bases of the symmetric functions*, Journal of Algebraic Combinatorics, 13, No. 1 (2000), pp. 83-101.

Appendix: RiggedConfigurationsPaths.mcc

```
#include<vector>
#include "FromOneCrystalPath.h"
#include "MAPITL.h"
#include "MAPITL_tmpl.h"
using MAPITL::MtoC;
using MAPITL::CtoM;
using MAPITL::Container;
using MAPITL::SimpleCell;
using MAPITL::Cell;
using MAPITL::ObjectInCell;
using MAPITL::ArrayInCell;

using namespace std;

MFUNC( newRiggedConfigurations, MCnop )
{
  MFnargsCheck(3);
  MFargCheck(1, DOM_INT);
  MFargCheck(2, DOM_INT);
  MFargCheck(3, DOM_LIST);
  n = MtoC<int>(MFarg(1));
  int path_len  = MtoC<int>(MFarg(2));
  Cell input(MFarg(3));
  int i, j, k, tmp, path_index, tblu_index, col;

  tmp = UNUSED;
  path_index = 0;
  tblu_index = 0;
  i = 0; j = 0; k = 0; col = 0; l = 0;
  initialize_lambda();
  path_class*  input_path = new path_class(path_len);
  reset_tableau();
  Cell *toto = input.toArray<Cell>();
  vector< vector<int> > res(n+1);

  for(long a=0; a < input.size(); a++)
  {
      Cell* alpha=(toto[a]).toArray<Cell>();
      int s = (alpha[0]).size();
      tblu_class *my_tblu = new tblu_class((toto[a]).size() ,s);
      my_tblu->tblu_id = tblu_index;
      tblu_index += 1;

      for(long d=0; d<toto[a].size(); d++)
      {
          int* beta=(alpha[d]).toArray<int>();
          for(long b=0; b < s ;b++)
          {
              my_tblu->tb[d][b] = beta[b];
              my_tblu->tab_lambda[beta[b]-1] =
                  my_tblu->tab_lambda[beta[b]-1] + 1;
          }
      }
      input_path->path[path_index] = my_tblu;
      reset_tableau ();
      path_index += 1;
  }
  input_path->build_rigged_for_path();
  for(i=0; i<n; i++)
  {
      vector<int> yyy;
      j = 0;
      while(input_path->rigged[i][0][j] != UNUSED)
      {
          yyy.push_back(input_path->rigged[i][0][j]);
          yyy.push_back(input_path->rigged[i][2][j]);
          yyy.push_back(input_path->rigged[i][1][j]);
          j++;
      }
      res[i] = yyy;
  }
  input_path->calculate_cocharge();
  vector<int> stat;
  stat.push_back(input_path->cocharge);
  res[n] = stat;

  MFreturn(Cell(res));
} MFEND
```

Calculating Cocyclic Hadamard Matrices in *Mathematica*: Exhaustive and Heuristic Searches

V. Álvarez, J.A. Armario, M.D. Frau, and P. Real[⋆]

Dpto. Matemática Aplicada I, Universidad de Sevilla, Avda. Reina Mercedes s/n
41012 Sevilla, Spain
{valvarez, armario, mdfrau, real}@us.es

Abstract. We describe a notebook in *Mathematica* which, taking as input data a homological model for a finite group G of order $|G| = 4t$, performs an exhaustive search for constructing the whole set of cocyclic Hadamard matrices over G. Since such an exhaustive search is not practical for orders $4t \geq 28$, the program also provides an alternate method, in which an heuristic search (in terms of a genetic algorithm) is performed. We include some executions and examples.

Nowadays, most of Computer Algebra Systems (CAS) begin to be concerned with the computation of homological information. This is the case of Gap [13], MAGMA [17]. In a parallel way, some specific softwares for achieving calculations in homological algebra are being developed, as it is the case of Kenzo [7] and Hap [15]. However none of them are concerned with the explicit calculation of Hadamard cocyclic matrices. In spite of that, some researchers have developed their own computations, working on different systems [6,11,12,14,2].

In this paper we present a notebook [1] in *Mathematica 4.0*, which implements the homological reduction method described in [4]. When exhaustive search is not feasible, the notebook provides a second routine for performing an heuristic search, which corresponds to the implementation of the genetic algorithm in [3]. The interested reader is referred to these papers for the theoretical background. Another non exhaustive method for finding out Hadamard cocyclic matrices is described in [5], in terms of image restorations.

The notebook takes as input data a homological model $\phi\colon \bar{B}(\mathbb{Z}[G]) \underset{g}{\overset{f}{\rightleftharpoons}} (hG, d)$ for a finite group G of order $|G| = 4t$. The term *homological model* refers to a contraction $\phi\colon \bar{B}(\mathbb{Z}[G]) \underset{g}{\overset{f}{\rightleftharpoons}} hG$ from the *reduced bar construction* of the group G (i.e. the reduced complex associated to the standard bar resolution [18]) to a differential graded module of finite type hG, so that $H_*(G) = H_*(hG)$ and the homology of hG may be effectively computed by means of Veblen's algorithm [19] (involving the Smith's normal forms of the matrices representing the differential operator).

⋆ All authors are partially supported by the PAICYT research project FQM–296 from Junta de Andalucía (Spain).

© Springer-Verlag Berlin Heidelberg 2006

In order to construct a basis for 2-coboundaries, the program needs to know the group law on G. To this end, the user must fix an ordering on the elements of G, say $G = \{g_1 = 1, \ldots, g_{4t}\}$. Now, a matrix P representing the group law in G may be constructed at once, so that $P(i,j) = k$ if and only if $g_i g_j = g_k$.

The only additional information which is needed is that of the diagrams

$$\begin{tikzcd} \bar{B}_1(\mathbb{Z}[G]) \arrow[r, "f"] & \mathcal{B}_1 \arrow[d, "Q"] \\ & \bar{\mathcal{B}}_1 \end{tikzcd} \qquad \begin{tikzcd} \bar{B}_2(\mathbb{Z}[G]) \arrow[r, "f"] & \mathcal{B}_2 \arrow[d, "Q"] \\ & \bar{\mathcal{B}}_2 \end{tikzcd}$$

That is, the matrices M_2 and M_3 representing the differentials operators d_2 and d_3 of hG, as well as the matrices F_1 and F_2 representing the maps $f_1 : \bar{B}_1(\mathbb{Z}[G]) \to \mathcal{B}_1$ and $f_2 : \bar{B}_2(\mathbb{Z}[G]) \to \mathcal{B}_2$.

In these circumstances, we may now determine exactly what the input and output data are.

Input data

- The matrices P, M_{i+1}, F_i representing the group law, d_{i+1} and f_i, $1 \leq i \leq 2$.
- The searching method to use: introduce 1 for an exhaustive search, anything else for a heuristic search.

Output data

- A full basis for normalized 2-cocycles over G.
- In case that an exhaustive search was chosen, the whole set of Hadamard cocyclic matrices over G. Otherwise, a few Hadamard cocyclic matrices, obtained from the heuristic search.

All the executions and examples included below have been worked out with aid of the *Mathematica 4.0* notebook which we have described here, running on a *Pentium IV 2.400 Mhz DIMM DDR266 512 MB*. Since some calculations obtained from the heuristic search are provided in [3], we focus on calculations obtained from an exhaustive search.

In the sequel, the elements of a product $A \times B$ are ordered as the rows of a matrix indexed in $|A| \times |B|$. For instance, if $|A| = r$ and $|B| = c$, the ordering is

$$< a_1b_1, a_1b_2, \ldots, a_1, b_c, a_2b_1, a_2b_2, \ldots, a_2b_c, \ldots, a_rb_1, \ldots, a_rb_c >$$

We assume $\mathbb{Z}_k = \{0, 1, \ldots, k-1\}$ with additive law. Next we include a table with the number of cocyclic Hadamard matrices that we have found in each case.

t	$\mathbb{Z}_{4t}$	$\mathbb{Z}_2 \times \mathbb{Z}_{2t}$	$\mathbb{Z}_4 \times \mathbb{Z}_t$	$\mathbb{Z}_2^2 \times \mathbb{Z}_t$	D_{4t}	$\mathbb{Z}_{2t} \ltimes_f \mathbb{Z}_2$	$(\mathbb{Z}_t \ltimes_f \mathbb{Z}_2) \rtimes_\chi \mathbb{Z}_2$
1	2	6	2	6	6	6	6
2	0	**16**	**16**	168	32	**16**	168
3	0	**24**	**0**	24	72	**0**	72
4	0	**96**	**192**	**1984**	768	**0**	**272**
5	0	120	0	120	**2200**	120	**2200**

Here f denotes the 2-cocycle $f(g_i, g_j) = \begin{cases} \lceil \frac{t}{2} \rceil + 1 \text{ if } g_i = g_j = 1 \\ 0 \qquad\qquad \text{otherwise} \end{cases}$ and χ denotes the dihedral action $\chi(a, b) = \begin{cases} -b \text{ if } a = 1 \\ \ \ b \text{ if } a = 0 \end{cases}$

The computing time required in all cases was practically of the same order: less than 3 seconds for $1 \leq t \leq 3$, a couple of minutes for $t = 4$ and about 30 minutes for $t = 5$. The black entries correspond to new results, as far as we know (the case of $G_5^5 = D_{4 \cdot 5}$ is included, since the computation of Flannery in [16], 2380, differs from ours, 2200).

References

1. V. Álvarez. http://mathworld.wolfram.com/HadamardSearch.html (2006). To appear.
2. V. Álvarez, J.A. Armario, M.D. Frau and P. Real. An algorithm for computing cocyclic matrices developed over some semidirect products. *AAECC-14 Proceedings*, LNCS **2227**, 287–296 (2001).
3. V. Álvarez, J.A. Armario, M.D. Frau and P. Real. A genetic algorithm for cocyclic Hadamard matrices. *AAECC-16 Proceedings*, LNCS **3857**, 144–153, (2006).
4. V. Álvarez, J.A. Armario, M.D. Frau and P. Real. Homological reduction method for constructing Hadamard cocyclic matrices. Communication submitted to the EACA-06 conference, Sevilla (2006).
5. A. Baliga and J. Chua. Self-dual codes using image resoration techniques. *Proceedings AAECC'14*. Eds. S. Boztas, I.E. Shparlinski. *Springer Lecture Notes in Computer Science*, Springer-Verlag, Heidelberg, (2001).
6. A. Baliga and K.J. Horadam. Cocyclic Hadamard matrices over $\mathbb{Z}_t \times \mathbb{Z}_2^2$. *Australas. J. Combin.*, **11**, 123–134, (1995).
7. X. Dousson, J. Rubio, F. Sergeraert and Y. Siret. The Kenzo program. Institute Fourier, Grenoble (19998). http://www-fourier.ujf-grenoble.fr/~sergeraert/Kenzo/
8. W. de Launey and K.J. Horadam. Cocyclic development of designs. *J. Algebraic Combin.*, **2** (3), 267–290, 1993. Erratum: *J. Algebraic Combin.*, (1), pp. 129, 1994.
9. W. de Launey and K.J. Horadam. Generation of cocyclic Hadamard matrices. *Computational algebra and number theory* (Sydney, 1992), volume **325** of *Math. Appl.*, 279–290. Kluwer Acad. Publ., Dordrecht, (1995).
10. D.L. Flannery. Calculation of cocyclic matrices. *J. of Pure and Applied Algebra*, **112**, 181–190, (1996).
11. D.L. Flannery. Cocyclic Hadamard matrices and Hadamard groups are equivalent. *J. Algebra*, **192**, 749–779, (1997).
12. D.L. Flannery and E.A. O'Brien. Computing 2-cocycles for central extensions and relative difference sets. *Comm. Algebra*, **28(4)**, 1939–1955, (2000).
13. The GAP group, 'GAP- Group, Algorithms and programming', School of Mathematical and Computational Sciences, University of St. Andrews, Scotland (1998).
14. J. Grabmeier, L.A. Lambe. Computing Resolutions Over Finite p-Groups. *Proceedings ALCOMA'99*. Eds. A. Betten, A. Kohnert, R. Lave, A. Wassermann. *Springer Lecture Notes in Computational Science and Engineering*, Springer-Verlag, Heidelberg, (2000).
15. G. Ellis. GAP package HAP, 'Homological Algebra Programming', http:// hamilton.nuigalway.ie/Hap/www/

16. K. Horadam. Progress in cocyclic matrices. *Congressus numerantium*, **118**, 161–171 (1996).
17. The MAGMA computational algebra system, http://magma.maths.usyd.edu.au
18. S. Mac Lane. Homology. *Classics in Mathematics*, Springer-Verlag, Berlin, (1995). Reprint of the 1975 edition.
19. O. Veblen. Analisis situs. A.M.S. Publications, **5** (1931).

An Interactive User Interface for Division Algorithms and the Buchberger Algorithm

Hiromasa Nakayama

Graduate School of Science and Technology, Kobe University
nakayama@math.kobe-u.ac.jp

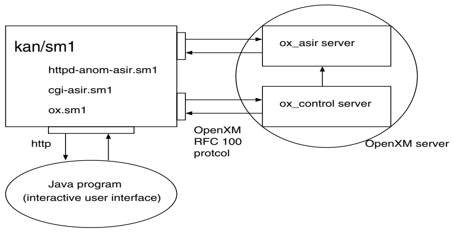
1 Introduction

Our objectives of building the interactive user interface are as follows:

(1) To select reducers in division algorithms and S-pairs in the Buchberger algorithm interactively.
(2) To visualize division algorithms and the Buchberger algorithm and to understand the algorithms intuitively.
(3) To create a user interface of division algorithms and the Buchberger algorithm without using computer algebra system languages.

Objective (1) has a mathematical background. We have studied and implemented division algorithms and the Buchberger algorithm in the ring of differential operators with rational function coefficients whose denominators do not vanish at the origin, $\mathcal{D}_{alg}$ ([5], [3]). In the ring of polynomials and the local ring of that, methods of efficiently computing a remainder and a Gröbner basis have been studied in detail ([1], [2], [4]). However, in the ring $\mathcal{D}_{alg}$, methods of those have not been studied in detail. As far as we have known, no system has satisfied our objectives. Therefore, we have designed an interactive user interface as a tool to understand and improve these algorithms. This system is a tool for us to study algorithms, however it may be useful for educational purposes.

2 System Architecture



The proposed system consists of two parts: polynomial computation and user interface. The Polynomial computation part is performed by Risa/Asir. Risa/Asir is an open source computer algebra system, and it is efficient at factorization, Gröbner basis computation ([8]). The user interface part is written in Java. These two parts are connected by OpenXM over HTTP ([7]). We can concentrate on the user interface with this architecture.

© Springer-Verlag Berlin Heidelberg 2006

3 Interactive User Interface for Division Algorithms

As shown in the screenshot (Fig.1, Fig.2), each ball stands for a monomial. Each row stands for a polynomial. The first row is a divident. The rest are divisors. In this case, $x^{10} + 1$ is divided by $x^2 + xy, xy + y^2$ with respect to the lex order such that $x > y$. The Monomial balls are sorted by the lex order. The possible reducers are emphasized in blue. In the manual mode, when we click a blue ball, the system executes a reduction by the selected reducer. In the automatic mode, the system automatically executes reductions with the current strategy on the clicking of the start button. We can easily switch between these two modes.

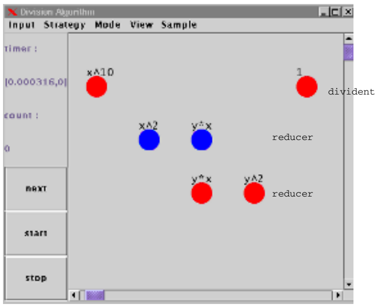


Fig. 1. Initial state

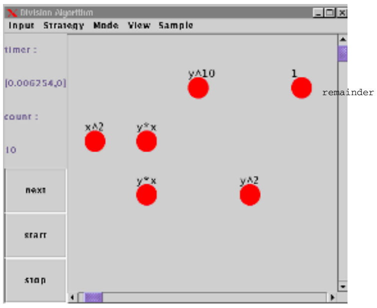


Fig. 2. Result

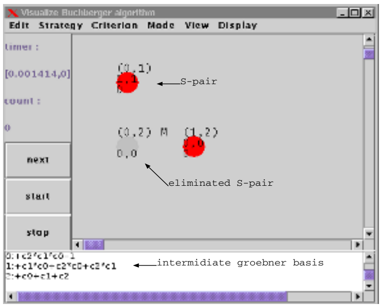
4 Interactive User Interface for the Buchberger Algorithm

As shown in the screenshots (Fig.3, Fig.4), each ball stands for an S-pair. A red ball has not yet been divided. A sky blue ball has been divided. The polynomials in the bottom window are intermidiate Gröbner basis, and will finally become a Gröbner basis.

When we click a ball, the system executes the division of the ball by the intermidiate Gröbner basis. If the remainder is 0, then the ball is eliminated. Else, then the set of S-pairs and the intermidiate Gröbner basis are updated. When we click the start button, the system switches to the automatic mode and an S-pair is automatically chosen with the current strategy, either the normal strategy or the sugar strategy. The pairs eliminated by the Buchberger criterion or the Gebauer-Möller criterion are visualized. These pairs are colored white or gray.

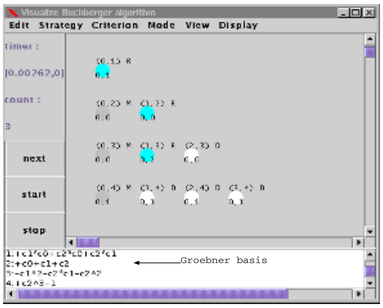
5 Application and Conclusion

We show an example of computing a Gröbner basis in the ring $\mathcal{D}_{alg}[s]$. We consider the computation of the Gröbner basis of the left ideal in $\mathcal{D}_{alg}[s]$ generated


Visualize Buchberger algorithm
Edit Strategy Criterion Mode View Display
timer :
[0.001414,0]
count :
0
next
start
stop
(0,1)
S-pair
(0,2) M
0,0
(1,2)
eliminated S-pair
intermidiate groebner basis

```
0:+c2*c1*c0-1
1:+c1*c0-c2*c0+c2*c1
2:+c0+c1+c2
```

Fig. 3. Initial state



```
1:+c1*c0-c2*c0+c2*c1
2:+c0+c1+c2
3:-c1^2-c2*c1-c2^2
4:+c2^3-1
```

Fig. 4. Result

by $xy + x^3 + y^3, x\partial_x - y\partial_y + 3y^2\partial_x - 3x^2\partial_y, sx + 3sy^2 - xy\partial_y - x^3\partial_y - y^3\partial_y, -3s + 2x\partial_x + y\partial_y + 27sxy - 3x^2\partial_y - 9x^2y\partial_x - 9xy^2\partial_y, 3sy + 9sx^2 - 2xy\partial_x - y^2\partial_y - 3x^3\partial_x - 3x^2y\partial_y$. This example appears in the computation of the local b function of $x^3 + xy + y^3$. We cannot compute the Gröbner basis with the normal strategy or the sugar strategy.

Using the interactive user interface, we can easily try all orders of S-pair selections. However, in any case, the computation stops in the Mora division of an S-pair. We conclude that S-pair selection strategies do not solve the problem, and the implementation of the Mora division algorithm in $\mathcal{D}_{alg}[s]$ should be improved.

This negative discovery was made with our interactive user interface. Nevertheless, our interface can be a useful tool to test new methods of computation such as for playing video games.

References

1. Gebauer, R., Möller, H.M., On an installation of Buchberger's algorithm, Journal of Symbolic Computation, 3, 275-286, (1989)
2. Giovini, A., Mora, T., Nielse, G., Robbiano, L., Traverso, C., "One sugar cube, please" OR Selection strategies in the Buchberger algorithm, Proceeding of ISSAC '91, 49-54
3. Granger, M., Oaku, T., Takayama, N.: Tangent cone algorithm for homogenized differential operators, Journal of Symbolic Computation, 39, 417-431, (2005)
4. Greuel, G.-M., Pfister, G. : Advances and improvements in the theory of standard bases and syzygies, Archiv der Mathematik, 66, 163-176, (1996)
5. Hiromasa Nakayama: An Algorithm Computing the Local b Funtion by an Approximate Division Algorithm in $\widehat{\mathcal{D}}$, `mathRA/0606437`
6. Hiromasa Nakayama and Kosuke Kuwahara: Interactive and Graphical User Interfaces for Computer Algebra Systems (Software), `http://www.math.kobe-u.ac.jp/~nakayama/bg.html`
7. OpenXM, `http://www.math.kobe-u.ac.jp/OpenXM`
8. Risa/Asir, `http://www.math.kobe-u.ac.jp/Asir/asir.html`

Experiment of Multithreading Symbolic and Algebraic Computations with OpenMP

Hirokazu Murao[⋆]

Department of Computer Science, The University of Electro-Communications

Abstract. This paper describes the current status of a project for multithreading algebraic computations, which aims at the utilization of today's high-spec PCs with hyperthreading or dual-core technologies. Our effort is done by applying OpenXM with minimal cost of development, and includes memory management in multithreaded environment. Our empirical results show that the performance gain can be attained in numeric cases and in some cases of purely symbolic computations.

1 Introduction

This paper is the first report of our on-going project for construction of multithreads environment for computer algebra on a small PC. Needless to say, the project is motivated by the availability, in daily-use computers, of processors with hyperthreading or dual-core technologies, and by the desire for their full utilization in algebraic computations. Concerning to software technology supporting multi-threading, there is a simple programming paradigm, called OpenMP[1]. It is supported by many commercial compilers and has become a de facto standard for multi-threading. Because the project is in the initial stage, major emphasis is laid upon the ease of programming rather than attaining higher performance. We shall develop a new application area of OpenMP. Through our experiments, we study the description ability of OpenMP, and investigate the efficiency.

With respect to mathematical software, we treat such simple and basic mathematical objects as arithmetics of polynomials, vectors and matrices over $\mathbb{Z}_p$, and the key software technologies we use include multithreading, OpenMP, dynamic memory management with garbage collection and its concurrent extension. Keywords and topics treated in this paper are listed below.

Mathematical subjects: efficient methods of arithmetics in $\mathbb{Z}_p$, multiplication of vectors and matrices with elements of $\mathbb{Z}_p$, the Berlekamp algorithm, polynomial factorization, polynomial multiplication.

Software technology: multithreads, OpenMP, shared-memory, data-parallelism, work-queue, dynamic memory management, garbage collection, concurrent GC, mutual exclusion, critical section, a computer algebra system Risa/Asir[2].

[⋆] Supported in part by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (C), 17500005, 2005–2008.

© Springer-Verlag Berlin Heidelberg 2006

OpenMP is an industry-standarad for shared-memory parallel programming in C/C++ and Fortran. The specification of OpenMP API defines a small set of compiler directives, library routines, and environment variables. As described in Section 2, parallel execution of program blocks is specified only by direcitives, which are usually recognized as comment by the underlying programming language. This means that with OpenMP, we can parallelize existing programs quite easily without changing the algorithm structure or the organizations of programs. In order to investigate the efficacy of data-parallelism in algebraic computations, we develop independent programs. Besides this, we extend the existing computer algebra system Risa/Asir, to test more practical computations of applications. For this purpose, we need a multi-threaded memory allocator and garbage collector. Risa/Asir uses the well-known GC by Boehm *et al.*[3], and we must adapt it to an OpneMP execution environment. Section 4 describes a simple method to realize. We simply add some restricting conditions so that multiple threads do not manage a single memory portion simultaneously, which can be implemented by the critical section mechanism of OpenMP.

We describe the details of the algebraic computations used in our experiments in Section 3, and some of the empirical results are reported in Section 5. Final section gives a summary of our experiments.

2 OpenMP

OpenMP defines a specification consisting of compiler directives, library routines, and environment variables for parallel processing. Parallelization can be specified so easily as simply adding a directive "*#pragma omp ...*" to the place where we want to execute the codes in parallel.

– An occurrence of the directive "*#pragma omp parallel*" initiates multi-threads execution, and the block following the directive will be executed by all the initiated threads.

```
#pragma omp parallel
{
    parallel block: program statements to be executed by all threads
    (We call this block omp-parallel block.)
}
```

There are two ways to make threads execute different codes inside a block.

– The directive "*#pragma omp for*" directs a compiler to automatically divide the `for`-loop that follows the directive into multiple portions of the iteration steps and to assign each portion to a single running thread. The whole work of the iteration steps is shared by multiple threads, and its portions are executed in parallel. This implements a work-sharing model. The work assignment can be either static or dynamic.

```
#pragma omp parallel
{
   #pragma omp for
   for (i = ...; i...; i+=...) {
        ...
   }
}
```

⇒

```
#pragma omp parallel for
for (i = ...; i...; i+=...) {
     ...
}
```

The directive "*#pragma omp parallel for ...*" can be used as a synonym for the directive "*#pragma omp for ...*" which solely exists in a *omp-parallel* block. The nested combination of the two directives with *parallel* and *for* can be replaced by a single directive with *parallel for*.

- The directive "*#pragma omp section*" inside a block following the directive "*#pragma omp sections*" provides a mechanism to describe a program code to be executed by an independent thread. Each program portion code$_i$ that follows "*#pragma omp section*" is executed by a single thread.
- Variables are shared by all threads unless they are declared as *private* in the directive.
- The directive "*#pragma omp critical*" used inside an *omp-parallel* block provides a mechanism to specify a program portion to be executed exclusively only by a single thread at any one time. The program portion, *statement* or a { }*-block*, cannot be executed by multiple threads simultaneously.

```
#pragma omp sections
{
     #pragma omp section
     {
          code1
     }
     #pragma omp section
     {
          code2
     }
     ...
}
```

```
#pragma omp critical
statement or { }-block
```

- Some library functions are defined. The function `omp_num_threads()` returns the number of running threads, and `omp_thread_num()` does an identification number (ID) of the current thread. An ID 0 is reserved for master thread.

For more directives and so on, or for more detailed specifications, refer to [1].

3 Typical Algebraic Computations and Parallelization

3.1 Algebraic Computation and Representation of Data

The most efficient way of computation is the direct use of hardware for data representation and calculation, and for higher efficiency in parallel processing, data-parallelism will be most promising. Algebraic computation is done using flexible data in general; symbolic formulas are usually represented by linked list, and even numeric data retaining arbitrary precisions are represented by some structure. Generally speaking, work sharing can hardly be a proper model for parallel algebraic computation. However, there are various semi-numerical algorithms[4] used in computer algebra, which treat uniform numeric data and are suited for data parallelism. Their common strategy is the calculation over a prime finite field $\mathbb{Z}_p$ for some prime p represented by a machine word. In this paper, we treat the following three types of algebraic calculations.

(1) Repeated dot-product calculations,
(2) the Berlekamp algorithm[5] with Gaussian elimination, applied to multiple matrices, and
(3) polynomial multiplications.

With (1) and (2), we treat vectors and matrices with $\mathbb{Z}_p$ elements, and parallelize multiple vector operations such as dot-product and elimination, which enjoy data-parallelism. The Berlekamp algorithm is usually used for polynomial factorization over the integers, and applied several times with different primes. The calculations with different primes are independent and can be executed in parallel. We also investigate this task-parallelism. Each vector of the matrix treated in the Berlekamp algorithm stands for a certain polynomial over $\mathbb{Z}_p$, however, in general-purpose computer algebra systems, polynomials are usually represented by linked list. Multiplication of polynomials in this general form will be also investigated, as an example of work-queueing model.

Calculation in $\mathbb{Z}_p$: Let p be a prime. Representing $\mathbb{Z}_p$ as $\{ 0, ..., p-1 \}$, we use *unsigned int*egers for its elements. We employ some simple techniques in order to eliminate division from mod-p reduction, as in [6, 7]. For additive operations, subtraction or addition of p from the integer sum or difference serves as $\bmod p$ operation. Computing the product of two elements requires double the precision of p. While for $p \leq 2^{16}$, 32-bit *int* suffices to hold the products, we use *unsigned long long* in the case of $2^{16} < p < 2^{32}$. The reduction $A \bmod p$ is done using the precomputed value R of $1/p$ in *double*, as $A \bmod p = A - \lfloor AR \rfloor p$. If $A \geq p \times 2^{32}$ in the case of big p, this reduction must be preceded by the high-bit reduction; $A = A - (\lfloor (A{>>}16)R \rfloor p){<<}16$. Notice that the mod-$p$ reduction is almost as costly as integer arithmetics, and that in the context of data-parallelism, the granularity can be bigger compared with the usual numerical processing.

3.2 Dot-Product over Prime Fields and Its Parallelized Repetition

Let a_i and $b_j \in \mathbb{Z}_p$ $(0 \leq a_i, b_j < p)$, and their (partial) dot-product by $S_k \overset{\text{def}}{=} \sum_{i=0}^{k} a_i b_i \bmod p$. The dot-product S_n can be computed by repeating the multiplication and addition sequence as $S_k = a_k b_k + S_{k-1} \bmod p$. To reduce the number of mod-p reduction, we accumulate the products without performing the reduction as long as possible. Actually, we have only to perform the reduction only once for the final S_n if $S_n > p$. An overflow caused by the addition is guaranteed to be only one bit, and can be treated as a value exactly. We call the datatype used for product as *dprod*-type.

– Let $\bar{X}$ be the value of (the maximum value represented by *dprod*-type modulo p) plus 1; i.e., in the 32-bit case, $\bar{X} = ((2^{32} - 1) \bmod p) + 1$ and in the 64-bit case, $\bar{X} = ((2^{64} - 1) \bmod p) + 1$. These constant values can be computed by repeating the additions mod$p$.

– Let $t = a_k b_k$ computed in the intermediate stage, and let $S_{k+1} = t + S_k$. If an overflow when the addition for S_{k+1} is detected by $t > S_{k+1}$ or $S_k > S_{k+1}$ as unsigned integers, we simply add the above $\bar{X}$ to S_{k+1}.

In order to examine how much efficacy we can attain by parallelization, we experiment with two kinds of highly compute-intensive calculations of dot-products.

Matrix Exponentiation. Our first example is exponentiation. A simple example of repeated dot-product calculation is matrix multiplication. We experiment further with exponentiation, which is a sequence of multiplications. Matrix multiplication itself consists of independent calculations, and is easily parallelized.

```
#pragma omp parallel for private(j,w, k,t)
for (i=0; i < $l$; i++)
    for (j=0; j < $m$; j++) {
        M[i][j] = $\sum_k^n$ A[i][k]*B[k][j] mod$p$;
```

Let A be an $n \times n$ matrix $\in {\mathbb{Z}_p}^{n\times n}$, and consider the exponentiation A^e for $e \geq 2$. Let $e = 2^s + \sum_{i=0}^{s-1} 2^i b_i$ where $b_i \in \{0, 1\}$, and we let $B_i = A^{2^i}$. Then, $A^e = {B_0}^{b_0} \cdots {B_{s-1}}^{b_{s-1}} B_s$ can be calculated by the repetition of the squaring $B_i = {B_{i-1}}^2$ and the multiplication $M_i = M_{i-1}{B_i}^{b_i}$ for $b_i \neq 0$, where $M_i = {B_0}^{b_0} \cdots {B_i}^{b_i}$. We apply the conversion of software-pipelining to combine the two multiplications (when $b_i \neq 0$) in the repetition for higher efficiency.

```
$B = f$;                              for ( ; $(i = i + 1) < s$; $B = X$)
for ($i = 0$; $b_i$ == 0; $i$++)         if ($b_i$ == 1) {
    $B = B * B$;                             $X = B * B$, $M = B * M$;
$M = B$;                                 } else $X = B * B$;
if ($i \geq s$) return $M$;            $M = B * M$;
$B = B * B$;                          return $M$;
```

Notice that the above method requires extra memory space for three $n \times n$ matrix products other than the one for the result.

Vector Sequence of $\boldsymbol{Ab}, \boldsymbol{A^2b}, ..., \boldsymbol{A^eb}$. A more practical example of dot-product calculation will be in Wiedemann's algorithm[8], a version of Krylov subspace method for solving linear systems over $\mathbb{Z}_p$. Given a matrix $A \in {\mathbb{Z}_p}^{n\times n}$ and a vector $\boldsymbol{b} \in {\mathbb{Z}_p}^n$, the algorithm computes a minimal polynomial of the sequence $A^k\boldsymbol{b}$, $k = 0, ..., 2n - 1$, and gives the solution to $A\boldsymbol{x} = \boldsymbol{b}$ as a linear combination of the sequence. Let $\boldsymbol{b}_d$ denote $A^d\boldsymbol{b}$. While the elements of $\boldsymbol{b}_{d+1} = A\boldsymbol{b}_d$ can be computed in parallel, the calculation of the sequence must be sequential.

```
for (d = 0; d < e; d++)  /*     <-- must be sequential */
    #pragma omp parallel for private(k,w)
    for (i=0; i < n; i++)  /*  <-- may better be unrolled */
        $\boldsymbol{b}_{d+1}$[i] = $\sum_{k=0}^{n-1}$A[i][k]*$\boldsymbol{b}_d$[k]  mod $p$;     /* dot-product */
```

In the above program, if the number of operations performed in the loop of `i` is quite limited, loop unrolling will be effective.

3.3 Berlekamp Algorithm for Polynomial Factorization

Let $f(x)$ be a polynomial $\in \mathbb{Z}_p[x]$ of degree n to be factored. The algorithm constructs a matrix $Q = \left(q_{j,k} \right)$ from the coefficients of $x^{kp} \bmod (f,p) = \sum_{j=0}^{n-1} q_{j,k}x^j$ for $0 \leq k < n$, and performs elimination on the matrix $(Q - I_n)$, where I_n is an $n \times n$ identity matrix, to determine the number of factors of $f(x)$ via the rank.

We experiment the following two methods of parallelization.

(B-1) work-sharing of multiple $(\alpha\boldsymbol{x} + \boldsymbol{y})$ vector operations in each elimination
(B-2) multiple invocation and simultaneous execution of the algorithm with different primes from work-queue

The former method (B-1) can be implemented as follows.

```
for (k = 0; k < n; k++) { /* eliminate the k-th column, Q[*][k] */
    /* pivoting */
    #pragma omp parallel for private(a,i)
    for (j = k+1; j < n; j++)
        if ((a = Q[j][k]) != 0)
            Q[j][i] = (p-a)*Q[k][i]+Q[j][i] mod p,  i = k+1,...,n-1;
}
```

When used for factorization over the integers, the algorithm is applied with different primes until the same value of the rank is obtained for sufficiently many primes. With the method (B2), we prepare a queue for primes and employ work-queueing model for parallelization.

```
j = 0, Nfacs = deg(f)+1, i = 1;
#pragma omp parallel private(p)
for ( ; j < Count && Nfacs > 1; ) {
    #pragma omp critical
    { p = i-th candidate for a modulo;  i++; }
    if (p divides lc(f) || deg gcd(f, f') > 0) continue;
    Q =  generate the Q-matrix for p and f;
    r = rank( Q-I );
    #pragma omp critical
    if ( r < Nfacs ) {
        remember p and the related information;
        Nfacs = r, j++;
    }
}
```

One prime is supplied to each thread on request. Each thread first checks if the supplied p is appropriate to the polynomial. The prime p cannot be used if p divides the leading coefficient of $f(x)$ or if $(f \bmod p)$ is not square-free (checked by the non-zero degree of $\gcd(f, f')$). If the check is successful, the algorithm will initiates. Thus, the grain size varies in three ways depending on the choice of p. The above code will choose such valid p that makes the number of factors mod p smallest.

3.4 Polynomial Multiplication

Let P_1 and P_2 be polynomials, and we consider the multiplication $P_1 * P_2$. The product can be computed in S, initially equal to zero, by performing $S \leftarrow S + t * P_2$ for each term t in P_1. Usually, polynomials are represented by a list of terms. In the above iteration for S, the list P_1 can be regarded as a queue of terms. The work for each entry t of the queue is the multiplication with P_2 and the addition of the result to S.

We apply the working-queueing model for parallelization. The following depicts the structure actually used for our implementation in Risa/Asir. `DCP` is a *typedef*-ed struct of a cell of linked-list containing one polynomial term. `NEXT()` gives a pointer to the next cell, and a NULL-pointer terminates lists.

```
/* dequeue function */
DCP GetNextTerm(DCP *pdcp)
{
   DCP a;
   #pragma omp critical
   if ((a = *pdcp) != NULL)
      *pdcp = NEXT(a);
    return a;
}
```

```
$S \leftarrow 0,\ P \leftarrow P_1$;
#pragma omp parallel private(s,t)
{
   $s \leftarrow 0$;
   while (($t$ = GetNextTerm(&$P$))
          != NULL)
      $s \leftarrow s + t * P_2$;
   #pragma omp critical
   $S \leftarrow S + s$
}
```

In our experiment, we use recursive representation of polynomials, and the above parallelization will be done with respect to the toplevel variable. As a queue polynomial, we shall use the polynomial with less terms of P_1 or P_2, to reduce the overhead for critical control. Furthermore, if we assign the terms in P_1 to each thread before performing multiplication, we don't need the critical treatment when dequeueing a term. We might be able to employ the load-balancing by counting the sizes of the coefficients. We call this method as balanced method.

Notice that in this implementation, the product of P_2 and a term in P_1 is collected in a thread-local variable s, and the results in the local variables are gathered in S after completing all the term-wise multiplications. Also notice that, although the polynomial addition here is performed destructively to prevent memory consumption as far as possible, the multiplications generate new data structures of the product and therefore, dynamic memory allocation is required.

4 Memory Management

One of the most important mechanisms in symbolic computation is run-time memory management. Usually, data of symbolic formula is represented by linked list of small memory portions, and those memory portions are dynamically allocated, sometimes deallocated, and discarded as garbage without being explicitly deallocated. Garbage collector collects and reclaims such discarded memory portions, and is indispensable to general-purpose computer algebra systems. Risa/Asir [2], a computer algebra system which we use for our experiment, makes use of the wide-spread garbage collector by Boehm *et al.* [3]. This memory allocator and garbage collector supports threads implementations on some platforms, as well as thread-local memory management on a limited kinds of platforms, and there has been an experimental parallel extension [9]. However, there is no OpenMP support, neither we cannot use threads implementation with OpenMP because the relation between OpenMP and the thread library is not clear. We concentrate on adapting the existing GC program to multi-threads processing environment by OpenMP.

As to the parallel execution of GC, there can be many ways of the adaptation, as is having been studied and published extensively. The following summarizes the principles and decisions we made in our effort for parallelization.

- Minimum effort is our basic principle; use the existing code unchanged as far as possible. For multi-threads control, we use only OpenMP.
- No parallelization in any step in GC itself. Rather, GC is executed concurrently with main computation, so long as the main computation does not call to memory allocator or deallocator (or explicitly to GC).
- One heap area maintained for all running threads, and no thread-local memory management.
- Make allocator and deallocator critical, in order to resolve such conflicts as simultaneous multiple requests for memory allocation and request of memory allocation during the execution of GC. All the related critical sections are named as `risa_gc`, and executed mutually exclusively.
- Only master thread can execute GC, and subthreads are allowed only to expand heap area, to prevent tracing from the stack areas with no known bounds.
- In order to suppress the swelling of heap area by excessive expansion of the heap, relax the condition to initiate GC. GC may be initiated in allocation time even if there are free memory space to allocate.

```
void *Risa_GC_malloc(size_t d)
{
  void *ret;
  #pragma omp critical (risa_gc)
  {
    /* call the original */
    ret = (void *) GC_malloc(d);
    /* register the memory portion
         if in subthread */
  }
  return ret;
}
```

- Prepare a global table per thread to register every allocated memory portion in each subthread, which enables tracing from the portions alive in some subthreads without knowing the bounds of the stack areas used by subthreads. Registration is done by the allocator wrapper, and the mutator (user program) removes the table entries pointed by any live memory portions in order to prevent table overflow.

– Prepare wrapper functions for allocators and deallocators, as `Risa_GC_malloc()`. They replace the original functions via macro definitions, as follows:

```
#define GC_malloc(d) Risa_GC_malloc(d)
```

One may worry that restricting GC to master thread leads to swelling of the heap area due to repeated expansions by subthreads, compared with the usual of a single thread. The situation will vary depending on the average lifetime length of allocated memory portions. Basically, the frequency of GC will decrease, while that of heap area expansion will increase, almost by the ratio of the number of running threads. Therefore, the memory portions with original lifetime shorter than the extended length will be affected, and their collection as garbage will have to be deferred in our multi-threads computation. We could retain the frequency simply by relaxing the GC-invocation condition so that the amount of free memory owned by the memory allocator gets less than the expected amount required for the GC interval. The lifetime of memory portion depends on the type of computation and implementation. In summary, we can expect that the excessive swelling of heap area will not happen for usual computations. Thread-local memory management might improve the behavior of memory consumption, but it is out of our scope because of our basic principle.

5 Empirical Studies

All of the computational examples in Section 3 are implemented, and tested on the following two types of platforms.

(HT) Pentium 4@3.0GHz with 1MB L2-cache, 512MB memory and hyper-threading turned on. OS: Fedora Core 4 with kernel 2.6.13 for SMP

(X2) Athlon 64 X2 4800+ (2.4GHz, 2× 1MB L2-cache) with 2×1GB memory. OS: Fedora Core 5 with stock kernel for SMP

On these platforms, the IA-32 and EM64T versions of Intel C++ compiler for Linux 9.0 are used.[1] The compiler option `-O3` is used for dot-products, and `-O2` for Risa/Asir including the Berlekamp algorithm and polynomial multiplication. While the dual-core processor (X2) consists of two independent processors which uses an external memory in common, (HT) is actually a uniprocessor and behaves like two processors by using the components of a processor, all of which are not fully utilized, in two phases. Timings in the tables are in msec except when noted otherwise, and represent the wall-clock time taken by gettimeofday().

Matrix Exponentiation. A^e, where $A \in {\mathbb{Z}_p}^{n\times n}$. With a matrix A randomly generated, various combinations of the following values for p, n, and e are tested.

[1] The current version of GCC does not support OpenMP, although its development is declared to be in progress.

Table 1. Timings of matrix exponentiation

e	31	63	127	255	511	1023	2047	4095	8191	16383	32767	65535
(HT)-(1)	134.4	169.9	201.8	233.5	267.0	299.3	333.2	365.9	398.9	432.0	468.0	497.7
(HT)-(2)	90.9	115.3	134.8	159.5	188.0	205.8	231.4	251.5	275.9	297.2	320.6	347.3
ratio	1.48	1.47	1.50	1.46	1.42	1.45	1.44	1.45	1.45	1.45	1.46	1.43
(X2)-(1)	76.3	96.1	114.7	138.1	157.7	176.9	197.2	216.2	236.6	255.4	277.3	294.7
(X2)-(2)	38.6	48.6	58.2	69.6	79.6	89.5	99.4	109.2	119.3	129.3	139.7	148.7
ratio	1.97	1.98	1.97	1.98	1.98	1.98	1.98	1.98	1.98	1.98	1.98	1.98

Timings of exponentiation of $A \in {\mathbb{Z}_p}^{128\times 128}$ to the power of e with $p = 65533$.

Table 2. Time of a dot-product

	$p = 65533$					$p = 911$				
n	128	256	512	1024	2048	128	256	512	1024	2048
(HT)-(1)	0.897	1.788	3.738	7.262	14.49	0.303	0.575	1.281	2.677	5.061
(HT)-(2)	0.704	1.197	2.416	4.709	9.456	0.393	0.573	1.126	2.098	4.152
ratio	1.27	1.49	1.55	1.54	1.53	0.77	1.00	1.14	1.28	1.22
(X2)-(1)	0.705	1.442	2.930	5.927	11.83	0.445	0.876	1.798	3.634	7.236
(X2)-(2)	0.378	0.770	1.463	2.994	5.996	0.306	0.482	0.892	1.844	3.693
ratio	1.82	1.87	2.00	1.98	1.97	1.45	1.82	2.02	1.97	1.96

Average times(μsec) for a dot-product over $\mathbb{Z}_p$ in the calculation of a vector sequence.

- $n = 128, 256, 512$.
- $p = 911$ or 65533. Notice that with $p = 911$, no mod-p reduction is required in the intermediate stage of the dot-product because $911^2 \times 512 < 2^{32}$.
- $e = 100, 200, 400, 800, 1000, 1200, 1600, 2000, 4000$; and 2^d-1 for $d = 5, ..., 16$.

Table 1 summarizes the typical timing data, actually taken for exponentiation with $p = 65533$ and $n = 128$. In the table, the row numbers (1) and (2) correspond to the number of threads, and the speed-up ratios are given. Notice that the ratio is almost constant for all e and that the difference between the neighboring columns, which are the times for two matrix multiplications, are also almost constant. These constant values reflect the characteristics of the processors. We observed similar results in other combinations. We simply give their summary without indicating actual timings.

- Complicated timings in the case that the working set of memory exceeds the capacity of cache memory ($n \geq 256$ with (HT) and $n \geq 512$ with (X2)).
- With (HT), if $p = 911$ or $n = 512$, the speed-up ratio decreases to around 1.2, even though CPU's are almost fully utilized.
- With (HT), no such degrade by the magnitude of p.

Vector Sequence. $A^k\boldsymbol{b}$ for $k = 1, ..., e$, where $A \in {\mathbb{Z}_p}^{n\times n}$ and $\boldsymbol{b} \in {\mathbb{Z}_p}^n$. Again, various combinations of the values for n, e and p are tested. Table 2 lists the average computing times (μsec) for a single dot-product of vectors of length n. Our experiments showed that loop unrolling is effective for $p = 911$, and

Table 3. Timings of multiple invocations of the Berlekamp algorithm

	(HT)				**(X2)**			
n	100	200	400	800	100	200	400	800
original:sequential	1.751	4.815	28.48	408.7	1.879	6.539	28.04	338.8
(2)-(B-1)	1.851	4.927	28.60	431.6	4.434	14.18	40.00	347.9
(2)-(B-2)	1.875	5.643	102.0	495.6	1.391	4.450	29.72	286.9

Table 4. Timings of polynomial multiplication

	$(x+1)^n \times (x+1)^m$							$(x+y+1)^n \times (x+y+1)^n$				
n	100	200	400	800	400	800	1600	40	50	60	70	80
m	↑	↑	↑	↑	1600	1600	1600					
(HT)-(1)	23.97	104.30	500.1	3279	3427	9854	30400	1664	3749	7576	14050	24170
(HT)-(1i)	17.58	84.44	422.6	3025	3157	9173	30010	1299	2958	5951	11160	19260
(HT)-(2)	19.50	89.07	462.3	2872	2850	8087	25220	1413	3400	6903	12590	22000
(X2)-(1)	18.51	64.57	314.3	1982	2239	6106	16840	1149	2382	4135	10110	16150
(X2)-(1i)	11.82	46.46	242.1	1739	1865	5508	15870	822	1799	5459	7866	12230
(X2)-(2)	24.20	88.95	422.5	1747	1876	4257	10810	1664	3798	7845	14640	24280
(X2)-(2b)	16.92	86.59	400.2	1708	1823	4156	10540	1652	3736	7739	14670	24200

the computing time decreases by about 10%. Summing up the above results, multi-threading turned out very effective for numeric algebraic calculations.

Multiple invocation of the Berlekamp algorithm. Factorization of $(x^n - 1024) \bmod p$ for multiple p's. Primes actually used in the (B-1) case are 13, 17, 19, 23, 29, and in the (B-2) case, one thread used 13, 19, 29 and another did 17, 23, 31. Table 3 shows the actual timings. The method (B-1) contains data-parallel processing for matrix elimination, but seems almost useless. Currently, the reason of the performance degrade of (B-1) is not known, and will require much more empirical studies. Note the performance improvement by the task-parallel method (B-2) in the X2 case, from which we may expect the performance improvement in non-numeric computations.

Polynomial Multiplication. The final example employs task-parallelism. Table 4 gives timings of multiplications of univariate and bivariate polynomials, where (1) uses the original sequential algorithm, (1i) does the improved sequential algorithm which uses the inplace operation implemented for parallelization, and (2) does the parallel algorithm with dynamic work-queueing executed by two threads. We observe that the parallelization is effective when the term-wise multiplication with a polynomial is sufficiently costly and the queue is sufficiently long, especially if multi-core processor is used. We tested the balanced-method which employs static load-balancing before the actual multiplication on (X2), and (2b) in the table gives the timings. In our experiment, there was slight differences in the actual numbers of distributed terms between the two methods, however we observe no significant difference in the timings. In the bivariate cases, we cannot observe any merit of parallelization so far as the simple example is concerned.

6 Concluding Remarks

We described our effort for multi-threads parallelization by using OpenMP. Although the experiments described in this paper is the first step of our project, our attempts seem very successful, attaining the performance improvement in some cases. The success is due to the ease of programming with OpenMP. Especially, memory management and garbage collection is always a difficult but important problem in parallel programming. Our simple method for adaptation, implemented with minimal modifications to the existing code, works almost fine, and is sufficient for casual use. If we can obtain the address of the thread-local stack area, the array for registration of allocated memory is not required and the collector can be made almost complete.

With respect to performance, there exist some kinds of algebraic computations which indicate clear performance gain by multithreading, and among them, numerical computation, performed efficiently by data-parallel processing, turned out to be most effective, as was imagined. Task-parallelism is difficult to make efficient, but our experimental results may give some prospect. Besides such research subjects, development of well-tuned library programs for basic operations in algebraic computation with vectors, matrices and polynomials will be necessary to making full use of the high-spec processors. In order to realize, we need to clarify the required functionalities in computer algebra, and design a good interface of subroutines with memory management facility taken into account.

References

[1] OpenMP ARB: OpenMP Application Program Interface, Version 2.5. (2005) available online as `http://www.openmp.org/drupal/mp-documents/spec25.pdf`.
[2] Noro, M., Takeshima, T.: Risa/Asir — a computer algebra system. In Wang, P.S., ed.: Proceedings of ISSAC '92, Berkeley, CA (1992) 387–396
[3] Boehm, H.: A garbage collector for C and C++. (`http://www.hpl.hp.com/personal/Hans_Boehm/gc/`)
[4] Knuth, D.E.: Seminumerical Algorithms. 3rd edn. Volume 2 of The Art of Computer Programming. Addison-Wesley (1997)
[5] Berlekamp, E.R.: Algebraic Coding Theory. McGraw-Hill, New York (1968)
[6] Dumas, J.G., et al.: LinBox: A generic library for exact linear algebra. In: Proc. ICMS '02, World Scientific Pub. (2002) 40–50
[7] Kawame, Y., Murao, H.: **MBLAS**: Modular basic linear algebra subprograms for computer algebra. In: SACSIS 2004. (2004) 139–140 in Japanese.
[8] Wiedemann, D.H.: Solving sparse linear equations over finite fields. IEEE Trans. Information Theory **IT-32**(1) (1986) 54–62
[9] Endo, T., Taura, K.: SGC: A parallel conservative garbage collector on shared-memory multiprocessors. (`http://www.yl.is.s.u-tokyo.ac.jp/gc/sgc.shtml`)

Links to Projects. Mathematical Software, icms2006—Developer's Meeting

Abstract. This document is a collection of links and descriptions of projects related to icms2006—developer's meeting.

1. **ActiveMath**, `http://www.activemath.org`
   License: Free and open-source for public educational institution
   ActiveMath is a learning environment for mathematics. It is a web-server based on java servlets which presents semantic OMDoc content to contemporary web-browsers with good quality.
   ActiveMath supports the learning experience by modelling the learners' competencies in a learner-model that is fed by tracking the reading and the interactive exercises. The interactive exercises can be rich and multiple-steps exercises and use computer-algebra-systems to evaluate a learner's input.
   The content items of ActiveMath are annotated with pedagogical and mathematical knowledge which allows the learner to obtain courses prepared on demand on given learning goals.
   Finally, the semantic OMDoc representation allows copy-and-paste of formulae and search for content items by text, annotations, or formulae.
2. **Aldor**, `http://www.aldor.org`
   License: Other (BSD like)
   Aldor is a computer programming language, like Scheme, Java or C$\sharp$.
3. **Algorithms in real algebraic geometry (interactive book)**
   `http://perso.univ-rennes1.fr/marie-francoise.roy/bpr-posted1.html`
   License: GPL, others
   Algorithms in real algebraic geometry, by S. Basu, R. Pollack and M.-F. Roy, is a preliminary test version of an extension of the first edition published by Springer in 2003. This interactive book is based on texmacs, and live examples can be computed within the book through the SARAG maxima library by F. Caruso. The final version is due to be published by Springer.
4. **Arageli**, `http://www.unn.ru/cs/arageli`
   License: Free for academic and noncommercial use
   Arageli is a C++ library for doing symbolic computations. It contains arbitrary precision integer and rational numbers, vectors, matrices, polynomials, modular arithmetic, algorithms for number factorization, linear and integer programming etc. Creating new mathematical structures from existing ones (also in other libraries) is easy. Arageli is used in Skeleton, Integizer, Prelinea projects.
5. **The Atlas of Lie Groups and Representations**
   `http://atlas.math.umd.edu/`
   License:

© Springer-Verlag Berlin Heidelberg 2006

This is a project to make available information about representations of semi-simple Lie groups over real and p-adic fields. The software is in an early stage of development. Given a general connected complex group G, the software will compute among other things: parameters for the irreducible representations of G with regular integral infinitesimal character, Kazhdan-Lusztig polynomials for representations with regular integral infinitesimal character, ... etc. (Alfred G. Noel, University of Massachusetts Boston)

6. **BESSELINT**
`http://www.cs.kuleuven.be/~nines/software/BESSELINT/`
License:
BESSELINT is a Matlab program to compute integrals of the form

$$\int_0^\infty x^m \prod_{i=1}^{k} J_{\nu_i}(a_i x) dx$$

with $J_{\nu_i}(x)$ the Bessel function of the first kind and (real) order ν_i. The parameter m is a real number such that $\sum_i \nu_i + m > -1$ and the coefficients a_i are strictly positive real numbers. In the near future this will be extended to compute the Laplace transform of a product of Bessel functions with possibly an additional factor of $1 + dx^2$ in the denominator (where d is a real number). (Joris Van Deun and Ronald Cools, K.U.Leuven)
7. **CoCoA4**, `http://cocoa.dima.unige.it/`
License: The executables are free
The interactive system CoCoA-4.6 offers facilities for COmputations in COmmutative Algebra: Gröbner bases and related operations on ideals and modules, Hilbert functions, factorization of polynomials, and some exact linear algebra. CoCoA-4.6 is well-suited to teaching with its simple and mathematically natural command language, and an extensive online help facility. It is free and runs on most common platforms.
8. **CoCoALib**, `http://cocoa.dima.unige.it/`
License: GPL, others
The C++ library CoCoALib offers data structures and operations for COmputations in COmmuative Algebra, most particularly Gröbner bases. Ease of use through a clean design is paramount (with some concessions to guarantee good performance). The library comes with full documentation and numerous example programs. A "beta" release is anticipated in late 2006. A server and interactive system are planned.
9. **Computer Algebra Animation**
`http://www.math.kobe-u.ac.jp/caa`
License: GPL
An experimental project for algorithm animations and graphical or interactive user interface in computer algebra with Java and OpenXM.
10. **Coq**, `http://coq.inria.fr`
License: LGPL
Coq is a proof assistant.

11. **DEpthLAUNAY**
    `http://www.dma.fi.upm.es/mabellanas/delonedepth/`
    License: QPL
    DEpthLAUNAY is a C++ software developed with CGAL (`http://www.CGAL.org`) that computes the following geometric structures given a finite set of points in the plane: convex hull, convex layers, convex levels, Delaunay triangulation, Delaunay layers, Delaunay levels, and Voronoi diagram. Input points can be randomly generated, introduced interactively or simply aquired from image files. (Manuel Abellanas, Universidad Politecnica de Madrid)
12. **Epsilon**, `http://www-calfor.lip6.fr/~wang/epsilon`
    License: Free for academic and noncommercial use
    Epsilon is a library of functions implemented in Maple and Java for polynomial elimination and triangular decomposition with (geometric) applications. It has 8 modules and contains more than 70 functions, with documentation, examples, and Maple worksheets. (Dongming Wang, Beihang University, China and UPMC-CNRS, France)
13. **GAP**, `http://www.gap-system.org`
    License: GPL
    GAP is a system for computational discrete algebra, with particular emphasis on Computational Group Theory. GAP provides a programming language, a library of thousands of functions implementing algebraic algorithms written in the GAP language as well as large data libraries of algebraic objects.
14. **GAP package Alnuth**
    `http://www.gap-system.org/Packages/alnuth.html`
    License: GPL
    The Alnuth package provides various methods to compute with number fields which are given by a defining polynomial or by generators. Some of the methods provided in this package are written in GAP code. The other part of the methods is imported from the Computer Algebra System KANT.
15. **GCLC/WinGCLC**
    `http://www.matf.bg.ac.yu/~janicic/gclc/`
    License: Free for academic and noncommercial use
    GCLC/WinGCLC is a tool for visualizing and teaching geometry (and not only geometry), and for producing mathematical illustrations. It has support for a range of geometrical constructions and transformations, for symbolic expressions, parametric curves, program loops, for automated proving of geometrical conjectures, for exporting figures into LaTeX and bitmap format etc.
16. **GEX**, `http://www.mmrc.iss.ac.cn/gex/`
    License:
    Geometry Expert (GEX) is a software for dynamic geometric diagram drawing and automated geometry theorem proving and discovering. As a dynamic geometry software, GEX can be used to build dynamic visual models to assist manipulating and teaching various mathematical concepts. As an automated

reasoning software, GEX can be used to prove and discover hundreds of non-trivial geometry theorems automatically Geometry Expert (GEX) is a software for dynamic geometric diagram drawing and automated geometry theorem proving and discovering. As a dynamic geometry software, GEX can be used to build dynamic visual models to assist manipulating and teaching various mathematical concepts. As an automated reasoning software, GEX can be used to prove and discover hundreds of non-trivial geometry theorems automatically.

17. **Gfan**, `http://home.imf.au.dk/ajensen/software/gfan/gfan.html`
License:
Gfan is a command line tool for enumerating the reduced Gröbner bases of a polynomial ideal in n variables. Hereby the Gröbner fan, an n-dimensional polyhedral complex, is computed. The tropical variety is a certain subcomplex which can also be computed by the software. Gfan uses Gmp and Cddlib for exact arithmetic and polyhedral computations, respectively.
18. **GiNaC**, `http://www.ginac.de/`
License: GPL
GiNaC looks like a computer algebra system and a library, but the name GiNac is an iterated and recurseive abbreviation for GiNaC is Not a Computer Algebra System.
19. **GloptiPoly**, `http://www.laas.fr/~henrion/software/gloptipoly`
License: GPL, requires Matlab
Matlab/SeDuMi add-on to build and solve convex LMI relaxations of the (generally non-convex) global optimization problem of minimizing a multivariable polynomial function subject to polynomial inequality, equality or integer constraints. (Didier Henrion and Jean-Bernard Lasserre, LAAS-CNRS, Toulouse, France)
20. **GiANT: Graphical Algebraic Number Theory**
`http://giantsystem.sourceforge.net`
License: GPL
GiANT is a graphical interface for working with number fields. GiANT offers interactive diagrams, drag-and-drop functionality, and typeset formulas. GiANT is written in Java, but uses KASH to perform its computations.
21. **GMP: GNU Multiple Precision Arithmetic Library**
`http://www.swox.com/gmp`
License: LGPL
GMP is a free library for arbitrary precision arithmetic, operating on signed integers, rational numbers, and floating point numbers. There is no practical limit to the precision except the ones implied by the available memory in the machine GMP runs on. GMP has a rich set of functions, and the functions have a regular interface.
22. **HOL Light**
`http://www.cl.cam.ac.uk/users/jrh13/hol-light/index.html`
License: Custom BSD-like
HOL Light is a theorem prover for classical higher-order logic. It is a rationalized re-implementation in Objective CAML (OCaml) of Mike Gordon's

original HOL system. HOL Light generates proofs using very low-level primitive inference steps, but has numerous higher-level automated rules and a good library of pre-proved mathematics. (John Harrison.)

23. **jReality**, `http://www.jreality.de`
License:
jReality is a library that allows for interactive manipulation of scientific data on a wide variety of platforms. It is written in Java. The output media include software only Java rendering, hardware accelerated OpenGL rendering stereo viewing, immersive virtual reality applications like in the PORTAL as well as file formats like PIXAR's renderman and SVG.
24. **jtem - Java Tools for Experimental Mathematics**
`http://www.jtem.de`
License: GPL
jtem is a collection of mostly mathematical tools and algorithms implemented in Java. The libraries include a wide range from complex numbers to theta-functions, basic linear algebra and adaptive ODE solvers.
25. **KASH/KANT**, `http://www.math.tu-berlin.de/~kant/kash.html`
License: Other
KASH/KANT is a computer algebra system specialized for algebraic number theory and its applications. It offers powerful functions for working with number fields, function fields, and local fields, as well as functions for solving Diophantine equations. The KANT shell KASH provides a programming language and an interactive help system.
26. **KENZO**, `http://www-fourier.ujf-grenoble.fr/~sergeraert/Kenzo`
License:
KENZO program is a Common Lisp Object System (CLOS) whose purpose is to compute various invariants in Algebraic Topology Framework (mainly homology groups). It implements and uses numerous algebraic structures as dg-algebras, simplicial groups, morphisms, reductions, chain complexes, etc. It allows to compute homology groups of sophisticated spaces.
27. **KENZO package for A-infinity structures: ARAIA and CRAIC**
`http://www.ehu.es/aba/investigation.htm`
License:
The package component we add to KENZO allows us to compute A-infinity-(co)algebra structures induced on the small module of a reduction if the big dg-module is a (co)algebra. (Ainhoa Berciano Alcaraz)
28. **KETpic**, `http://www.kisarazu.ac.jp/~masa/math/E.html`
License: Free for academic and noncomercial use
KETpic is a macro package for Maple, which generates TEXsource codes for clear drawings. This supports users to draw every kind of complicated figure easily with the highest accuracy. The current version as well as the latest Maple runs on major OS: Windows, Mac and Linux. One can download the package, various samples, the command reference and a template of TEXsource from the indicated web site. (Masayoshi SEKIGUCHI)
29. **KNOPPIX/math**, `http://www.knoppix-math.org`
License:

KNOPPIX/Math is a project to archive free mathematical software and free mathematical documents and provide them on KNOPPIX.

30. **libMpIeee**, `http://www.mpieee.ua.ac.be`
License:
libMpIeee is a C++ software library for radix 2^n or 10^m mixed precision floating-point arithmetic, that is fully compliant with the IEEE 754/854 standards for floating-point arithmetic.
This implies, among others, that the basic operations as well as the remainder and square root operations are exactly rounded, involving a relative error of at most 0.5 units in the last place (ULP) for round to nearest and 1 ULP for directed roundings. As required by the IEEE 754/854 standards, libMpIeee also provides denormal numbers, signed zeroes, signed infinities and NaN (Not-a-Number).
Moreover, the conversions between decimal and binary are implemented to satisfy the same error bound as the basic operations. The elementary functions are computed with a relative error of at most 1 ULP for round to nearest and 2 ULP for directed roundings. More information on the implementation of these elementary functions can be found in the contribution "Towards reliable software for the evaluation of a class of special functions" by Annie Cuyt and Stefan Becuwe.
31. **Linbox**, `http://www.linalg.org`
License: LGPL
LinBox is a C++ template library for exact, high-performance linear algebra computation.
32. **LiveTeXmacs**, `ftp://math.cgu.edu.tw/pub/KNOPPIX`
License: GPL
LiveTeXmacs is a knoppix-based live distribution for mathematical education. Many mathematical software systems (Axiom, Gnuplot, Maxima, Octave, R etc) are integrated in this distribution. In addition to this, it also includes free documentations for calculus, mathematical (2D/3D) visualization and stochastic processes etc, which are made by TeXmacs.
Web applications, such as PHP, Python, Apache, SQL and Zope etc, are also supported. Especially, the Web Mathematics Interactive is pre-installed as the web application for Maxima, Gnuplot and more. Pre-installed development environment gives a scalability of the distribution even with very few knowledge about linux.
33. **Logiweb**, `http://logiweb.eu/`
License: GPL
Logiweb is an infrastructure for publication and archival of machine verified mathematics. It allows users to publish definitions, theories, lemmas, proofs, and computer programs. The system allows proofs to reference previously published material across the Internet so that users in different sites may build on top of the work of each other. (Klaus Grue, University of Copenhagen)
34. **Macaulay2**, `http://www.math.uiuc.edu/Macaulay2`
License: GPL

Macaulay2 is a software system to support researches in algebraic geometry and commutative algebra.

35. **Magma**, `http://magma.maths.usyd.edu.au`
License: Commercial
Magma is a large, well-supported software package designed to solve computationally hard problems in algebra, number theory, geometry and combinatorics. It provides a mathematically rigorous environment for computing with algebraic, number-theoretic, combinatoric and geometric objects.
36. **Maple**, `http://www.maplesoft.com`
License: Commercial
Maple is a tool for solving mathematical problems and creating interactive technical applications.
37. **Mathematica**, `http://www.wolfram.com`
License: Commercial
Mathematica is a system for doing mathematics. The cite also provides http://functions.wolfram.com (a data base for special functions) and http://library.wolfram.com (a collection of notebooks).
38. **math-polyglot**, `http://www.math.kobe-u.ac.jp/math-polyglot`
License: GFL, BSD
This project is started for editing icms2006 — developer's meeting DVD's. It provides sample codes for mathematical software systems in our DVD's. Samples are grouped with mathematical problems. This project also provides inputs for testing and checking if the installation is properly done to the knoppix-math. The project will be moved to a MediaWiki of knoppix-math after the icms2006. (Nobuki Takayama, Kobe University)
39. **MATLAB**, `http://www.mathworks.com`
License: Commercial
Matlab is a language for technical computing and an interactive environment.
40. **Mizar**, `http://www.mizar.org`
License:
Mizar project provides a database for mathematics (7000 definitions, 40000 theorems) as well as the mizar system.
41. **MPFR library for multiple precision floating point computation**
`http://www.mpfr.org`
License: LGPL
The MPFR library is a C library for multiple-precision floating-point computations with exact rounding (also called correct rounding). It is based on the GMP multiple-precision library. The main goal of MPFR is to provide a library for multiple-precision floating-point computation which is both efficient and has a well-defined semantics.
42. **Multi Parametric Toolbox**, `http://control.ee.ethz.ch/~mpt/`
License: GPL
The Multi-Parametric Toolbox (MPT) is a free Matlab toolbox for design, analysis and deployment of optimal controllers for constrained linear, nonlinear and hybrid systems. Efficiency of the code is guaranteed by the extensive

library of algorithms from the field of computational geometry and multi-parametric optimization.

The toolbox offers a broad spectrum of algorithms compiled in a user friendly and accessible format: starting from different performance objectives (linear, quadratic, minimum time) to the handling of systems with persistent additive and polytopic uncertainties. Users can add custom constraints, such as polytopic, contraction, or collision avoidance constraints, or create custom objective functions. Resulting optimal control laws can either be embedded into your applications in a form of a C code, or deployed to target platforms using Real Time Workshop.

43. **MuPad**, `http://www.mupad.de`
License: Commercial
MuPad is a mathematical expert system mainly symbolic/algebraic computation and numerical calculations with arbitrary accuracy.
44. **MuPAD-Combinat** , `http://mupad-combinat.sourceforge.net`
License:
MuPAD-Combinat is an open-source algebraic combinatorics package for the computer algebra system MuPAD `http://www.mupad.de`. Its main purpose is to provide an extensible toolbox for computer exploration, and foster code sharing between researchers in this area. The core of the package is integrated in the official library of MuPAD since version 2.5.0.
45. **NZMATH**, `http://tnt.math.metro-u.ac.jp/nzmath/`
License: BSD
NZMATH is a number theory oriented calculation system based on the scripting language Python. It is currently providing the Python library package named nzmath, which contains modules for primality testings, factorization, polynomial, matrix, elliptic curves, etc. (MATSUI Tetsushi, Tokyo Metropolitan University)
46. **Oorange**, `http://www.oorange.de`
License: GPL
Oorange is a development environment that was initially designed for experimental mathematics but has become a general Java programming tool. It's main feature is its interactive programming, also known as rapid prototyping. It also fits into component oriented development.
47. **OpenMath**, `http://www.openmath.org`
License:
OpenMath is an emerging standard for representing mathematical objects with their semantics, allowing them to be exchanged between computer programs, stored in databases, or published on the worldwide web.
48. **OpenXM**, `http://www.openxm.org`
License: BSD, GPL, FFL
OpenXM is an infrastructure for communications among mathematical software systems. The project has proposed protocols OpenXM RFC 100, 101, 102, 103, 104. The OpenXM package is a collection of software systems supporting OpenXM protocols. The current distribution contains asir, sm1, Macaulay 2, PHCpack, gnuplot, polymake, cdd, ntl, pari, supports for Maple, Mathematica, OpenMath, and some small sized systems.

49. **webOrigami**
`http://weborigami.score.cs.tsukuba.ac.jp/webOrigami/jsp/`
License:
Origami assistant on a web.
50. **ORMS**, `http://orms.mfo.de`
License:
The Oberwolfach References on Mathematical Software (ORMS) project is a web-interfaced collection of information and links on mathematical software, see `http://orms.mfo.de`.
It presents carefully selected software, including general purpose software systems, teaching software, and more specialized packages up to specific implementations on particular mathematical research problems. Each software package is presented by a short description of its mathematical features, information on basic properties like license type and distribution media are provided, and links to syntax examples, possible demo versions and manuals are given. The ORMS offers a choice of retrieval functions, for example, searching via browsing through a mathematical classification scheme, a structured key word search, and a full text search in the description of the software systems.
The success of this project will rely on the co-operation of experienced users from different areas of mathematical software. We therefore encourage discussion (at orms@mfo.de) and we will be grateful to contributions (at contrib-orms@mfo.de).
51. **Pari-GP**, `http://pari.math.u-bordeaux.fr`
License: GPL
The PARI system is a package which is capable of doing formal computations on recursive types at high speed. It is primarily aimed at number theorists, but is useful to anyone whose primary need is speed. The functionality of PARI is available through a sophisticated programmable calculator, named GP, which also implements many features of high level languages.
52. **PHCpack**, `http://www.math.uic.edu/~jan/download.html`
License: GPL, free software
PHCpack is a package for Polynomial Homotopy Continuation. Version 1 was archived by ACM TOMS as Algorithm 795. Currently it is a platform for “numerical algebraic geometry” providing algorithms developed jointly with Andrew Sommese and Charles Wampler. Contributions made by Anton Leykin, Yusong Wang, Ailing Zhao, and Yan Zhuang. (Jan Verschelde, UIC.)
53. **PHCmaple**, `http://www.math.uic.edu/~leykin/PHCmaple`
License: GPL
This Maple package provides a convenient interface to the functions of PHCpack, a collection of numerical algorithms for solving polynomial systems using polynomial homotopy continuation.
54. **polymake**, `http://www.math.tu-berlin.de/polymake/`
License: GPL
polymake is a modular, object-oriented tool for experimental discrete geometry. The main applications are dealing with polytopes and polyhedra,

polyhedral surfaces, and finite simplicial complexes. It offers an unified interface to a wide variety of free software packages from the field, especially for visualisation. It can be used interactively as well as driven by perl scripts or C++ programs. (Ewgenij Gawrilow, Technische Universität Berlin, and Michael Joswig, Technische Universität Darmstadt)

55. **ProofPower**
`http://www.lemma-one.com/ProofPower/index/index.html`
License: GPL
ProofPower is a tool for specification and proof in Higher-Order Logic (HOL) and in the Z Notation, based on a re-engineering of Mike Gordon's original HOL system. Like other systems in the LCF tradition, it uses a powerful strongly-typed functional programming language (Standard ML) to ensure logical correctness by reducing all proof steps to primitive inferences rules and to provide a rich and extensible repertoire of automated derived inference rules and decision procedures. Its is used commercially for the verification of safety-critical avionics control systems. Libraries of discrete and continuous mathematics have been and continue to be developed to support these applications and for their intrinsic interest.
56. **QaoS - Querying Algebraic Objects System**
`http://www.math.tu-berlin.de/~kant/database.html`
License: BSD like
QaoS (Querying Algebraic Objects System) is a stand-alone solution for access to the KANT databases of algebraic and transcendental field extensions. QaoS transfers information via the hypertext transport protocol (HTTP). It can be accessed through a web interface and various computer algebra systems (GAP 4, KASH 2.56, KASH 3, Maple, and SAGE).
57. **Risa/Asir**, `http://www.math.kobe-u.ac.jp/Asir`
License: Other(FFL, BSD)
Risa/Asir is a computer algebra system. Here is a list of some commands: fctr (factorization), gr, nd_gr_main (Grobner basis), primadec (primary ideal decomposition), af (factorization over algebraic numbers), ifplot (plot of implicit functions), ox_* (OpenXM communication functions), generic_bfct (b-function).
58. **SAGE: Software for Algebra and Geometry Experimentation**
`http://sage.scipy.org/sage`
License: GPL
The Python-based computer algebra system SAGE comes with the systems GAP, PARI, Singular. and provides interfaces KASH/KANT, Gnuplot, Octave, Magma, Mathematica, and Maple. It includes functions for basic algebraic geometry, elliptic curves over the rational numbers, modular forms, linear algebra and Z-modules, and noncommutative algebra.
59. **SARAG**
`http://perso.univ-rennes1.fr/marie-francoise.roy/bpr-posted1.html`
License: GPL

SARAG(Some Algorithms in Real Algebraic Geometry) is a software library that implements some algorithms in real algebraic geometry written in the free computer algebra system Maxima. SARAG has two main applications: extending the capabilities of Maxima and being part of the interactive version of the book “Algorithms in Real Algebraic Geometry” by S. Basu, R. Pollack, M.-F. Roy, which can be now freely downloaded.

60. **sgpviz**, `http://www.gap-system.org/Packages/sgpviz.html`
License:
The GAP package *sgpviz* is a package designed to visualize finite semigroups through their $\mathcal{D}$-classes or Cayley graphs, as well as to make friendlier the usage of GAP when dealing with finite semigroups.
61. **Singular**, `http://www.singular.uni-kl.de`
License: GPL, others
SINGULAR is a Computer Algebra System for polynomial computations with emphasis on the special needs of commutative algebra, algebraic geometry, and singularity theory. SINGULAR’s main computational objects are ideals and modules over a large variety of rings, including local rings and non-commutative G-algebras (in the subsystem PLURAL). Large variety of algorithms, including those based on Gröbner and standard bases, have powerful implementations in SINGULAR.
62. **Strong Noether Position**
`http://www-calfor.lip6.fr/~hashemi/Noether`
License:
We introduce the notion of a homogeneous ideal in Strong Noether Position (SNP); a new definition for the notion of generic coordinates for some problems. This definition is simple to check, because one can test it for the initial ideal of the ideal with respect to the degree reverse lexicographic ordering. It is explicit, because we can provide an algorithm to decide whether a monomial ideal is in SNP or not. We propose some methods to compute the Castelnuovo-Mumford regularity of an ideal which one of them is more efficient than that of [Bermejo-Gimenez, 2005]. (Amir Hashemi)
63. **Surface Evolver**
`http://www.susqu.edu/facstaff/b/brakke/evolver/`
License:
The Surface Evolver is an interactive program for the study of surfaces shaped by surface tension and other energies, and subject to various constraints. A surface is implemented as a simplicial complex (a union of triangles) and is evolved toward minimal energy by a gradient descent method. The Evolver can handle arbitrary topology. Graphical output is available in several file formats.
64. **surfex**, `http://www.surfex.algebraicsurface.net`
License: Freeware
surfex is a tool for interactive high quality real algebraic surface visualization. It is mainly an intuitive interface which combines the strenghts of several visualization tools; at the moment, these include surf, javaview, and convert.

One of the main features of surfex is the interactive visualization of families of algebraic varieties: The user can introduce parameters into some implicit equations and then play with these parameters via sliders. E.g., this allows one to visualize deformations of surface singularities.

All the pictures can easily be exported, either as an animation which shows the whole deformation process, or as a snapshot of a single variety. The user can specify the quality of the output which can range from very high quality for printed publications to lower quality for websites.

We also implemented a library for the computer algebra system Singular, called surfex.lib, which enhances the quality of the visualization of surfex using pre-computation of the singular locus etc.

65. **SYNAPS**, `http://www-sop.inria.fr/galaad/logiciels/synaps`
License: GPL
SYNAPS (SYmbolic Numeric APplications) is a C++ library devoted to symbolic and numeric computations. It provides data-structures for the manipulation of basic algebraic objects, such as vectors, matrices (dense, sparse, structured), univariate and multivariate polynomials. It contains solvers for univariate and multivariate polynomials, including generalized normal form or subdivision solvers, tools for the manipulatiion of algebraic numbers, for the construction of resultants, ...
66. **TC**, `ftp://tnt.math.metro-u.ac.jp/pub/math-packs/tc/`
License: GPL
TC (Tiny C) is an interpreter of multi-precision C language suitable for floating-point calculations of several thousands digits which often appear in computational algebraic number theory. Furthermore TC ver.4, which is equipped with PARI library and turned TCP ver.1, provides frequently used PARI library functions by C-like functions which are easy to call from TC. (Takashi Fukuda)
67. **TeXmacs**, `http://www.texmacs.org`
License: GPL
GNU TeXmacs is a free wysiwyw editing platform for scientists.
68. **Virtual Math Laboratories**, `http://www.jreality.de`
License: GPL
Virtual Math Labs is a collection of educational and scientific java web start applications. The library of mathematical experiments shows topics from varies areas, e.g. dynamical systems, polyhedral surfaces, calculus, and curves and surfaces.

About this document

This document, edited by Masayuki Noro and Nobuki Takayama, is distributed under the GNU Free Documentation License Version 1.2. June 27, 2006.

List of contributors of this document: Ainhoa Berciano Alcaraz, Alfred Gerard Noel, Amir Hashemi, Anders Nedergaard Jensen, Anton Leykin, Bernard Mourrain, Chu-ching Huang, Colin Jones, Dongming Wang, Ewgenij Gawrilow, Fabrizio Caruso, Francois Descouens, Fukuda

Takashi, Gert Martin Greuel, Jan Verschelde, Jean B. Lasserre, John Abbott, John Harrison, Joris Van Deun, Klaus Ebbe Grue, MATSUI Tetsushi, Manuel Abellanas, Manuel Delgado, Marie-Francoise Roy, Markus Schmies, Masayoshi SEKIGUCHI, Masayuki Noro, Nobuki Takayama, Oliver Labs, Paul Libbrecht, Predrag Janicic, Rob Arthan, Sebastian Pauli, Sergey Lyalin, Stefan Becuwe, Viktor Levandovskyy, Xiaoshan Gao, Xin Li.

Author Index