SEMA SIMAI Springer Series 9

Inmaculada Higueras
Teo Roldán
Juan José Torrens *Editors*

# Numerical Simulation in Physics and Engineering

## Lecture Notes of the XVI 'Jacques-Louis Lions' Spanish-French School

SEMA SIMAI Springer
Springer

# SEMA SIMAI Springer Series



**Volume 9**

More information about this series at http://www.springer.com/series/10532

Inmaculada Higueras • Teo Roldán •
Juan José Torrens
Editors

# Numerical Simulation in Physics and Engineering

## Lecture Notes of the XVI 'Jacques-Louis Lions' Spanish-French School

Springer

*Editors*
Inmaculada Higueras
Ingeniería Matemática e Informática
Universidad Pública de Navarra
Pamplona, Spain

Teo Roldán
Ingeniería Matemática e Informática
Universidad Pública de Navarra
Pamplona, Spain

Juan José Torrens
Ingeniería Matemática e Informática
Universidad Pública de Navarra
Pamplona, Spain

ISSN 2199-3041 ISSN 2199-305X (electronic)
SEMA SIMAI Springer Series
ISBN 978-3-319-32145-5 ISBN 978-3-319-32146-2 (eBook)
DOI 10.1007/978-3-319-32146-2

Library of Congress Control Number: 2016943645



Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG Switzerland

# Preface

This contributed book contains lecture notes of the XVIth 'Jacques-Louis Lions' Spanish-French School on Numerical Simulation in Physics and Engineering, which took place in Pamplona (Navarra, Spain) in September 2014 hosted by the Public University of Navarre.

This series of Schools has been organized every 2 years since 1984 at different locations in Spain and is intended for professionals, researchers and students interested in numerical methods. The 15 previous editions were held in Santiago de Compostela (1984), Benalmádena (1986), Madrid (1988), Santiago de Compostela (1990), Benicàssim (1992), Sevilla (1994), Oviedo (1996), Córdoba (1998), Laredo (2000), Jaca (2002), Cádiz (2004), Castro Urdiales (2006), Valladolid (2008), A Coruña (2010) and Torremolinos (2012). The next edition will take place in Gijón in June 2016.

Since its foundation in 1991, the Sociedad Española de Matemática Aplicada (SEMA) has been actively involved in the organization of these Schools which, together with the Congreso de Ecuaciones Diferenciales y Aplicaciones/Congreso de Matemática Aplicada, represent the two series of scientific meetings sponsored by the Society. In 2004, the Spanish-French School honoured the French mathematician Jacques-Louis Lions by giving his name to the School. Since 2008, the Société de Mathématiques Appliquées et Industrielles (SMAI) has co-organized the School. The main goals of the Schools are the following:

- To initiate people interested in Applied Mathematics into research topics, in particular the mathematical modeling and numerical simulation arising in research areas being developed in France and Spain.
- To become a meeting point for young/senior researchers, professors, industrial technicians and graduate students from both countries.
- To showcase current applications of numerical simulation in industry, with an emphasis on French and Spanish companies.

The Schools are aimed at graduate students in Engineering or Science who are seeking an introduction to numerical simulation, either as a research topic or in the field of industrial applications. They are also oriented to technicians working

in industry who are interested in the use of numerical techniques for particular applications or want to know about research taking place in both French and Spanish universities and scientific institutions. Finally, these Schools may also be of interest for academics in general, since they permit the exchange of knowledge and experience concerning research topics being developed in different laboratories.

Each edition is organized around several main courses delivered by renowned French and Spanish scientists. On this last occasion there were four 6-h courses, for which the lecturers were Isabelle Faille, Francisco-Javier Sayas, Benjamin Stamm and Rafael Vázquez, together with three 1-h talks by Antonio Baeza, Eva Balsa-Canto and Florence Hubert and a 4-h workshop led by José Miguel Mantas. Furthermore, the participants in the School had the opportunity to present their research work in a poster session.

The Editors warmly thank all the speakers and participants for their contributions to the success of the School. In particular, we would like to acknowledge the efforts of all the lecturers and speakers who have contributed to this volume. In addition, we are indebted to the anonymous referees for their thorough reviews of the papers, which have contributed in improving the quality of this book.

We are also grateful to the Organizing and Scientific Committees for their efforts in the preparation of the School. We extend our thanks and gratitude to all sponsors and supporting institutions for their valuable contributions: SEMA, SMAI, the French Embassy in Spain, the Public University of Navarre and its Department of Mathematical Engineering and Computer Science.

Pamplona, Spain
December 2015

Inmaculada Higueras
Teo Roldán
Juan José Torrens

# Contents

# List of Contributors

**Antonio Baeza** Facultad de Matemáticas, Departamento de Matemática Aplicada, Universidad de Valencia, Valencia, Spain

**Eva Balsa-Canto** (Bio)Process Engineering Group, IIM-CSIC, Vigo, Spain

**José Miguel Mantas** Departamento de Lenguajes y Sistemas informáticos, Universidad de Granada, Granada, Spain

**Francisco-Javier Sayas** Department of Mathematical Sciences, University of Delaware, Newark, DE, USA

**Rafael Vázquez** Istituto di Matematica Applicata e Tecnologie Informatiche 'E. Magenes', Pavia, Italy

# Part I
# Advances in Numerical Analysis

# An Introduction to the Numerical Analysis of Isogeometric Methods

**Lourenço Beirão da Veiga, Annalisa Buffa, Giancarlo Sangalli, and Rafael Vázquez**

**Abstract** This paper gives an introduction to isogeometric methods from a mathematical point of view, with special focus on some theoretical results that are part of the mathematical foundation of the method. The aim of this work is to serve as a complement to other existing references in the field, that are more engineering oriented, and to provide a reference that can be used for didactic purposes. We analyse variational techniques for the numerical resolutions of PDEs using isogeometric methods, that is, based on splines or NURBS, and we provide optimal approximation and error estimates for scalar elliptic problems. The theoretical results are demonstrated by some numerical examples. We also present the definition of structure-preserving discretizations with splines, a generalization of edge and face finite elements, also with approximation estimates and some numerical tests for time harmonic Maxwell equations in a cavity.

**Keywords** Isogeometric methods • NURBS • Finite elements • De Rham complex

L. Beirão da Veiga
Dipartimento di Matematica e Applicazioni, Università degli Studi di Milano-Bicocca, Via Cozzi 55, 20125 Milano, Italy

Istituto di Matematica Applicata e Tecnologie Informatiche 'E. Magenes' del CNR, via Ferrata 1, 27100, Pavia, Italy
e-mail: lourenco.beirao@unimib.it

A. Buffa • R. Vázquez (✉)
Istituto di Matematica Applicata e Tecnologie Informatiche 'E. Magenes' del CNR, via Ferrata 1, 27100, Pavia, Italy
e-mail: annalisa@imati.cnr.it; vazquez@imati.cnr.it

G. Sangalli
Dipartimento di Matematica, Università di Pavia, via Ferrata 1, 27100, Pavia, Italy

Istituto di Matematica Applicata e Tecnologie Informatiche 'E. Magenes' del CNR, via Ferrata 1, 27100, Pavia, Italy
e-mail: Giancarlo.sangalli@unipv.it



I. Higueras et al. (eds.), *Numerical Simulation in Physics and Engineering*,
SEMA SIMAI Springer Series 9, DOI 10.1007/978-3-319-32146-2_1

## 1 Introduction

Isogeometric analysis refers to a collection of methods (called from now isogeometric methods), introduced by T.J.R. Hughes and coauthors in the seminal paper [28], that use splines, or some of their generalizations such as NURBS (non-uniform rational B-splines) and T-splines, as functions to build approximation spaces which are then used to numerically solve partial differential equations (PDEs).

The use of splines for the solution of variational problems started more than 20 years ago (see for instance [45] and references therein) but they have recently gained popularity in the computational mechanics community after the publication of [28], where the objective was to use splines (or NURBS) to improve the interoperability between computer aided design (CAD) software and PDE solvers. It is already clear that full interoperability is a challenging target, mainly because standard CAD software does not provide geometry representations that can be automatically used for numerical simulation. However, isogeometric methods have attracted the interest of the scientific community, as it is clearly documented by the number of publications on the subject appeared in the fields of geometric modelling, mechanical engineering and numerical analysis of PDEs.

One of the most interesting features of isogeometric methods is the high continuity of splines and NURBS with respect to standard finite elements. Isogeometric methods have been tested and applied on a variety of problems of engineering interest, and there is indeed a large engineering literature showing the beneficial effects of higher regularity in several practical problems, see for instance the references given in [21] and in the introduction of [8]. Moreover, the high continuity of splines and NURBS, and their use as building blocks for the construction of discrete spaces, pave the way to new numerical schemes for the discretization of high-order PDEs that would be extremely hard to achieve within a standard finite element framework.

The aim of this work is to provide an introduction to isogeometric methods from a mathematical viewpoint, with special focus on the numerical analysis of the method. This should serve as a complement to other existing references about isogeometric methods which are more engineering oriented, in particular [28] and the book [21] written by the same authors. The present work is largely inspired by our recent overview on isogeometric methods [8], but with respect to that paper we have tried to lighten the presentation by removing the most technical sections and proofs, and also to add some details for didactic purposes.

The contents of the paper are organized as follows. In Sect. 2 we present an introduction to splines and NURBS, and introduce some concepts and properties from classical splines theory that are needed for the rest of the paper. In Sect. 3, we introduce the main ideas behind isogeometric methods, together with the isoparametric-isogeometric concept, and apply the method for the solution of a simple scalar elliptic problem. In Sect. 4 we analyse the approximation properties of isogeometric spaces defined from splines or NURBS. The analysis is based on appropriate quasi-interpolant operators that generalize those from classical spline

theory. Finally, in Sect. 5 we introduce a De Rham complex of spline spaces, that can be seen as a generalization of edge and face finite elements in the context of isogeometric methods. We study the approximation properties of these spaces and perform some numerical tests.

## 2 Splines and NURBS: Basics

In this section we give a brief overview on B-splines and NURBS, to introduce the main definitions and results that will be used throughout the paper. The section is divided in three parts. In the first one we present some basic definitions and properties about univariate B-splines, and their use for the definition of spline curves. In the second part we give the generalization to the multivariate case, which is of interest for geometric modelling of surfaces and volumes. In the last part of the section we present the construction and the approximation properties of a quasi-interpolant for splines.

Reference books where the contents of this section can be found together with many more results are [46] and [12], whereas the basic ingredients for the practical use of splines in the context of isogeometric methods are summarized in [21]. In view of our focus, some of the properties that have made B-splines and NURBS the most successful technology for geometric modelling and computer aided design are left aside from this introduction, or simply mentioned without further discussion. For readers interested in getting a better understanding of this field, we recommend the books [18, 41] and [43].

### *2.1 Univariate B-Splines: Splines and NURBS Curves*

We start presenting the definition and properties of B-splines and NURBS in the univariate case.

#### 2.1.1 Definition and Properties of Univariate B-Splines

Let an ordered sequence of knots be given by the *knot vector*

$$\Xi = \{\xi_1 \leq \xi_2 \leq \ldots \leq \xi_{n+p+1}\},$$

where repeated knots are allowed. Without loss of generality, we assume in the following that $\xi_1 = 0$ and $\xi_{n+p+1} = 1$.

We say that the knot vector $\Xi$ is a $p$-open knot vector if the first and last knot are repeated $p + 1$ times, that is, $\xi_1 = \ldots = \xi_{p+1}$ and $\xi_{n+1} = \ldots = \xi_{n+p+1}$.

From the knot vector $\Xi$, B-spline functions of degree $p$ are defined following the well-known Cox-DeBoor recursive formula: we start with piecewise constants ($p = 0$):

$$\hat{B}_{i,0}(\zeta) = \begin{cases} 1 \text{ if } \xi_i \le \zeta < \xi_{i+1}, \\ 0 \text{ otherwise,} \end{cases} \tag{1}$$

and for $p \ge 1$ the *B-spline* functions are defined by the recursion

$$\hat{B}_{i,p}(\zeta) = \frac{\zeta - \xi_i}{\xi_{i+p} - \xi_i}\hat{B}_{i,p-1}(\zeta) + \frac{\xi_{i+p+1} - \zeta}{\xi_{i+p+1} - \xi_{i+1}}\hat{B}_{i+1,p-1}(\zeta), \tag{2}$$

where it is here formally assumed that 0/0 = 0. This gives a set of $n$ B-splines, that have the following properties

- Non-negativity: $\hat{B}_{i,p}(\zeta) \ge 0$.
- Partition of unity: $\sum_{i=1}^{n} \hat{B}_{i,p}(\zeta) = 1$.
- Local support:

$$\hat{B}_{i,p}(\zeta) = 0, \text{ for } \zeta \notin [\xi_i, \xi_{i+p+1}],$$
$$\hat{B}_{i,p}(\zeta) = 0, \text{ for } \zeta \in (\xi_r, \xi_{r+1}) \text{ and } i \notin \{r, r-1, \ldots, r-p\}.$$

Note that the definition of each B-spline $\hat{B}_{i,p}$ depends only on $p + 2$ knots, which are collected in the *local knot vector*

$$\Xi_{i,p} := \{\xi_i, \ldots, \xi_{i+p+1}\}.$$

An example of B-splines is given in Fig. 1. We denote the *univariate spline space* spanned by the B-splines by

$$S_p(\Xi) = \text{span}\{\hat{B}_{i,p},\ i = 1, \ldots, n\}. \tag{3}$$

We introduce now the vector $Z = \{\zeta_1, \ldots, \zeta_N\}$ of knots without repetitions, also called breakpoints, and denote by $m_j$ the multiplicity of the breakpoint $\zeta_j$, such that

$$\Xi = \{\underbrace{\zeta_1, \ldots, \zeta_1}_{m_1 \text{ times}}, \underbrace{\zeta_2, \ldots, \zeta_2}_{m_2 \text{ times}}, \ldots, \underbrace{\zeta_N, \ldots, \zeta_N}_{m_N \text{ times}}\}, \tag{4}$$

with $\sum_{i=1}^{N} m_i = n + p + 1$. We assume $1 \le m_j \le p + 1$ for all internal knots.

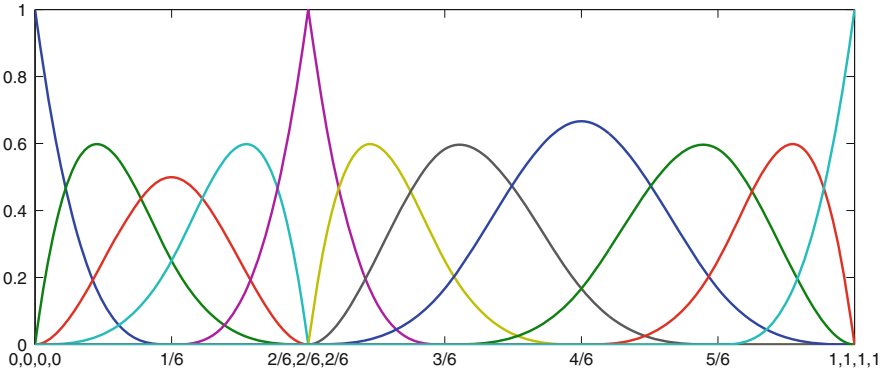


**Fig. 1** Cubic B-splines and the corresponding knot vector with repetitions

Let $k_j = p - m_j$, and define the *regularity vector* $\mathbf{k} = \{k_1, \ldots, k_N\}$. We define the space of piecewise polynomials of degree $p$ with $k_j = p - m_j$ continuous derivatives at the breakpoints $\zeta_j$,

$$\mathcal{P}_{p,Z,\mathbf{k}} = \{u : u_{|(\zeta_i,\zeta_{i+1})} \in \mathbb{P}_p,\ D^j_- u(\zeta_i) = D^j_+ u(\zeta_i),\ j = 0, \ldots, k_i,\ i = 1 \ldots, N-1\},$$

where $\mathbb{P}_p$ are the polynomials of degree $p$, and $D^j_\pm$ denote the $j$-order left and right derivative (or left and right limit for $j = 0$). Note that the maximum value of $m_j = p + 1$ stands for a discontinuity at $\zeta_j$.

We have the following results:

- Characterization of B-splines: the B-spline functions $\hat{B}_{i,p}$ form a basis of the space of piecewise polynomials $\mathcal{P}_{p,Z,\mathbf{k}}$. As a trivial consequence, we have $S_p(\Xi) = \mathcal{P}_{p,Z,\mathbf{k}}$.
- Local linear independence: in a knot interval $(\xi_r, \xi_{r+1})$ the $p + 1$ B-spline basis functions $\{\hat{B}_{r-p,p}, \ldots, \hat{B}_{r,p}\}$ are linearly independent.

Assuming the maximum multiplicity of the internal knots is less than or equal to the degree $p$, i.e., the B-spline functions are at least continuous, the derivative of each B-spline $\hat{B}_{i,p}$ is given by the following expression:

- Derivative of a B-spline:

$$\frac{d\hat{B}_{i,p}}{d\zeta}(\zeta) = \frac{p}{\xi_{i+p} - \xi_i}\hat{B}_{i,p-1}(\zeta) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}}\hat{B}_{i+1,p-1}(\zeta). \tag{5}$$

where we have assumed that $\hat{B}_{1,p-1}(\zeta) = \hat{B}_{n+1,p-1}(\zeta) = 0$. In fact, the derivative belongs to the spline space $S_{p-1}(\Xi')$, where $\Xi' = \{\xi_2, \ldots, \xi_{n+p}\}$ is a $(p-1)$-open knot vector. Moreover, it is easy to see that $\dfrac{d}{d\zeta} : S_p(\Xi) \to S_{p-1}(\Xi')$ is a surjective

application. For later use, we define the so called *Curry-Schoenberg spline basis* (see e.g., [12, Chap. IX]), as follows

$$\hat{D}_{i,p-1}(\zeta) = \frac{p}{\xi_{i+p+1} - \xi_{i+1}} \hat{B}_{i+1,p-1}(\zeta), \quad \text{for } i = 1, \ldots, n-1.$$

The indices for the new basis have been shifted in order to start numbering from 1. Then formula (5) becomes

$$\frac{d\hat{B}_{i,p}}{d\zeta}(\zeta) = \hat{D}_{i-1,p-1}(\zeta) - \hat{D}_{i,p-1}(\zeta), \tag{6}$$

where, again, we adopt the convention $\hat{D}_{0,p-1} = \hat{D}_{n,p-1} = 0$.

Finally, we note that the points in $Z$ form a partition of the unit interval $I = (0, 1)$, i.e., a mesh, and the local mesh size of the element $I_i = (\zeta_i, \zeta_{i+1})$ is called $h_i = \zeta_{i+1} - \zeta_i$, for $i = 1, \ldots, N-1$. Moreover, given an interval $I_j = (\zeta_j, \zeta_{j+1})$ of the partition, which can also be written as $(\xi_r, \xi_{r+1})$ for a certain (unique) $r$, we associate the *support extension* $\tilde{I}_j$ defined as

$$\tilde{I}_j := (\xi_{r-p}, \xi_{r+p+1}), \tag{7}$$

that is the interior of the union of the supports of basis functions whose support intersects $I_j$.

*Remark 2.1* In the particular case of a $p$-open knot vector without internal knots, that is, $\Xi = \{0, \ldots, 0, 1, \ldots, 1\}$, B-splines reduce to *Bernstein polynomials*, a particular basis of the space of polynomials of degree $p$. Bernstein polynomials in the unit interval are given by the equation

$$b_{i+1,p}(\zeta) := \binom{p}{i} \zeta^i (1-\zeta)^{p-i}, \quad i = 0, \ldots p.$$

### 2.1.2 Splines and NURBS Curves

A *spline curve* in $\mathbb{R}^d$, $d = 2, 3$ is a linear combination of B-splines and control points as follows:

$$\mathbf{C}(\zeta) = \sum_{i=1}^{n} \mathbf{c}_i \hat{B}_{i,p}(\zeta) \qquad \mathbf{c}_i \in \mathbb{R}^d, \tag{8}$$

where $\{\mathbf{c}_i\}_{i=1}^n$ are called control points. Given a spline curve $\mathbf{C}(\zeta)$, we call *control polygon* the piecewise linear curve obtained by joining the control points $\{\mathbf{c}_i\}_{i=1}^n$ (see Fig. 2). The control polygon has several nice properties and some of them are

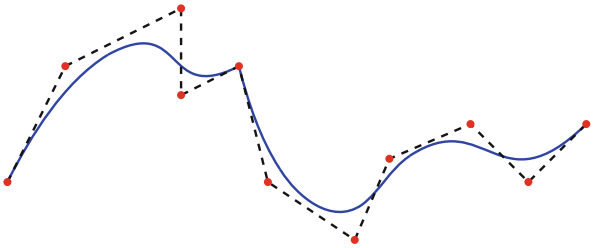

**Fig. 2** Spline curve (*solid line*), control polygon (*dashed line*) and control points (*red dots*)

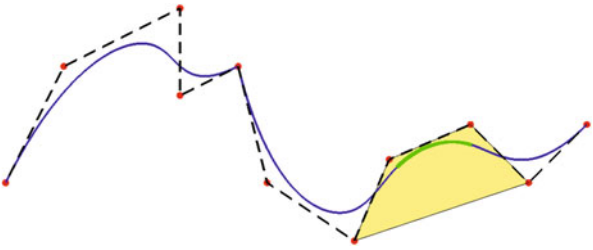

**Fig. 3** Local convex hull property for a cubic spline curve

at the very core of their use in geometric modelling. We list here the most important ones, and refer to [18] for details.

- Affine invariance: an affine transformation of the spline curve is obtained by applying it to the control points.
- Variation diminishing property: a straight line cannot intersect the curve more times than it intersects the control polygon.
- Local convex hull: given $\zeta \in (\xi_i, \xi_{i+1})$, only the values $\mathbf{c}_{i-p}, \ldots, \mathbf{c}_i$ act on $\mathbf{C}(\zeta)$, which belongs to their convex hull (see Fig. 3).

Despite the many advantages of splines, there exist several curves important for design that cannot be represented with piecewise polynomials, and in particular all conic curves except the parabola. It is known that the conic sections can be parametrized by rational polynomials in the form

$$x(\zeta) = \frac{X(\zeta)}{W(\zeta)}, \qquad y(\zeta) = \frac{Y(\zeta)}{W(\zeta)}.$$

For instance, the circumference of radius 1 can be parametrized with $X(\zeta) = 1-\zeta^2$, $Y(\zeta) = 2\zeta$, $W(\zeta) = 1 + \zeta^2$. This has motivated the introduction of *Non-Uniform Rational B-Splines*, better known as NURBS,[1] which are now the most widespread technology for geometric modelling, thanks to their versatility to represent both free-form and sculptured surfaces and conic sections. A discussion about the need for rational curves and surfaces can be found in [41, Sect. 1.4].

[1] Non-Uniform refers to the distance between the knots.

In order to define NURBS, we set the *weight* $W(\zeta) = \sum_{\ell=1}^{n} w_\ell \hat{B}_{\ell,p}(\zeta)$ where the positive coefficients $w_\ell > 0$ for $\ell = 1, \ldots, n$ are usually called *weights*. After setting the weight, we define the NURBS basis functions

$$\hat{N}_{i,p}(\zeta) = \frac{w_i \hat{B}_{i,p}(\zeta)}{\sum_{\ell=1}^{n} w_\ell \hat{B}_{\ell,p}(\zeta)} = \frac{w_i \hat{B}_{i,p}(\zeta)}{W(\zeta)}, \qquad i = 1, \ldots, n, \tag{9}$$

which are clearly rational B-splines. Since the basis functions depend on the choice of the weight $W$, we denote the NURBS space they span by

$$N_p(\Xi, W) = \text{span}\{\hat{N}_{i,p},\ i = 1, \ldots, n\}. \tag{10}$$

Similarly to splines, a NURBS curve is defined by associating one control point to each basis function, in the form:

$$\mathbf{C}(\zeta) = \sum_{i=1}^{n} \mathbf{c}_i \hat{N}_{i,p}(\zeta) \qquad \mathbf{c}_i \in \mathbb{R}^d. \tag{11}$$

Actually, the NURBS curve is a projection into $\mathbb{R}^d$ of a non-rational B-spline curve in the space $\mathbb{R}^{d+1}$, which is defined by

$$\mathbf{C}^w(\zeta) = \sum_{i=1}^{n} \mathbf{c}_i^w \hat{B}_{i,p}(\zeta),$$

where $\mathbf{c}_i^w = [w_i \mathbf{c}_i, w_i] \in \mathbb{R}^{d+1}$ (see Fig. 4). For more details about NURBS we refer to [41].

### 2.1.3 Knot Insertion and Degree Elevation

In the context of geometric modelling, refinement refers to the possibility of adding new control points into a spline or NURBS curve without changing its

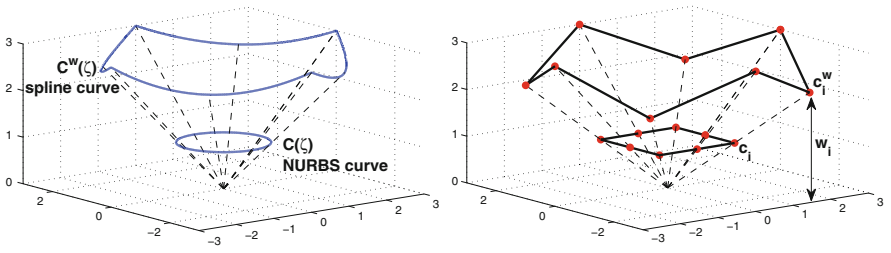


**Fig. 4** Representation of the circumference as the projection of a non-rational spline curve

parametrization $\mathbf{C}(\zeta)$. In order to do so, a refined space must be defined which contains the space of the parametrization. Given the spline space $S_{p^0}(\Xi^0)$, we say that $S_p(\Xi)$ is a *refinement* of $S_{p^0}(\Xi^0)$ if

$$S_{p^0}(\Xi^0) \subset S_p(\Xi). \tag{12}$$

In the case of NURBS, since the parametrization must not change during refinement, also the weight $W(\zeta)$ must remain the same. Thus, given a NURBS space $N_{p^0}(\Xi^0, W)$ we say that $N_p(\Xi, W)$ is a refinement if

$$N_{p^0}(\Xi^0, W) \subset N_p(\Xi, W).$$

In fact, since $W$ does not change, the previous condition reduces to (12) on the corresponding spline spaces.

For splines and NURBS curves, refinement is performed by knot insertion and degree elevation algorithms, which allow to recompute the control points and the weights of the parametrization in the refined space. It can be proved that applying these two algorithms the control polygon converges to the curve, see for instance [18, Chap. 16] for details. By combining knot insertion and degree elevation, three kinds of refinement are possible in isogeometric methods, as explained in [28]:

1. *h-refinement* which corresponds to mesh refinement and is obtained by knot insertion. Let $\bar{\Xi} := \Xi \cup \{\bar{\xi}\}$ be the knot vector after inserting the knot $\bar{\xi}$ into $\Xi$, and assume that $\xi_j \le \bar{\xi} \le \xi_{j+1}$. Denoting the knots in $\bar{\Xi}$ by $\bar{\xi}_\ell$, and the corresponding B-spline functions by $\bar{B}_{i,p}$, the original B-spline functions can be expressed in terms of the refined ones by

$$\hat{B}_{i,p}(\zeta) = \alpha_i \bar{B}_{i,p}(\zeta) + (1 - \alpha_{i+1})\bar{B}_{i+1,p}(\zeta) \tag{13}$$

   with the coefficients

$$\alpha_i = \begin{cases} 1, & i = 1, \dots, j-p, \\ \dfrac{\bar{\xi} - \bar{\xi}_i}{\bar{\xi}_{i+p+1} - \bar{\xi}_i}, & i = j-p+1, \dots, j, \\ 0, & i = j+1, \dots, n+1. \end{cases} \tag{14}$$

   When $\bar{\xi}$ is equal to $\xi_j$ or $\xi_{j+1}$ or to both, the knot insertion corresponds to reduction of the inter-element regularity at $\bar{\xi}$. Substituting the expression for the refined basis functions (13) in the parametrization (8) gives the expression for the control points after knot insertion. An example of knot insertion, with the insertion of several knots, is shown in Fig. 5.
2. *p-refinement* which corresponds to degree raising with fixed interelement regularity. Successive application of $p$-refinement generates a sequence of nested spaces. An example of degree elevation is given in Fig. 6.

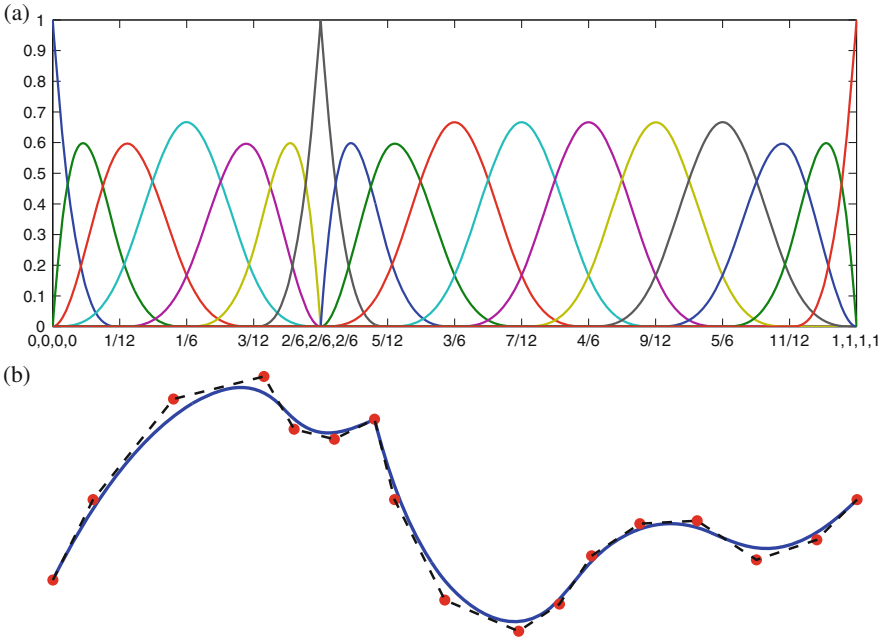


**Fig. 5** Knot insertion algorithm applied to the curve in Fig. 2, with basis functions as in Fig. 1. (**a**) Quartic basis functions and knot vector after applying *h*-refinement. (**b**) Control polygon computed with the knot insertion algorithm

3. *k-refinement* which corresponds to apply degree elevation to the curve in the space $N_{p^0}(\Xi^0, W)$, and then knot insertion. By doing so, regularity is maintained at the knots of $\Xi^0$, but it is increased along with the degree in all the other knots. Therefore, successive application of *k*-refinement generates a sequence of spaces that are not nested. The name *k*-refinement has been coined in [28].

## 2.2 *Multivariate Splines and NURBS: Tensorization*

Multivariate B-splines are defined by simple tensor product, starting from univariate B-splines on each spatial direction. By applying standard tensorization arguments, most of the properties of the univariate case can be extended to the multivariate case. Since the argument is quite standard, we proceed without many details and refer the reader to [46] and [12], or to the book [21] for further details.

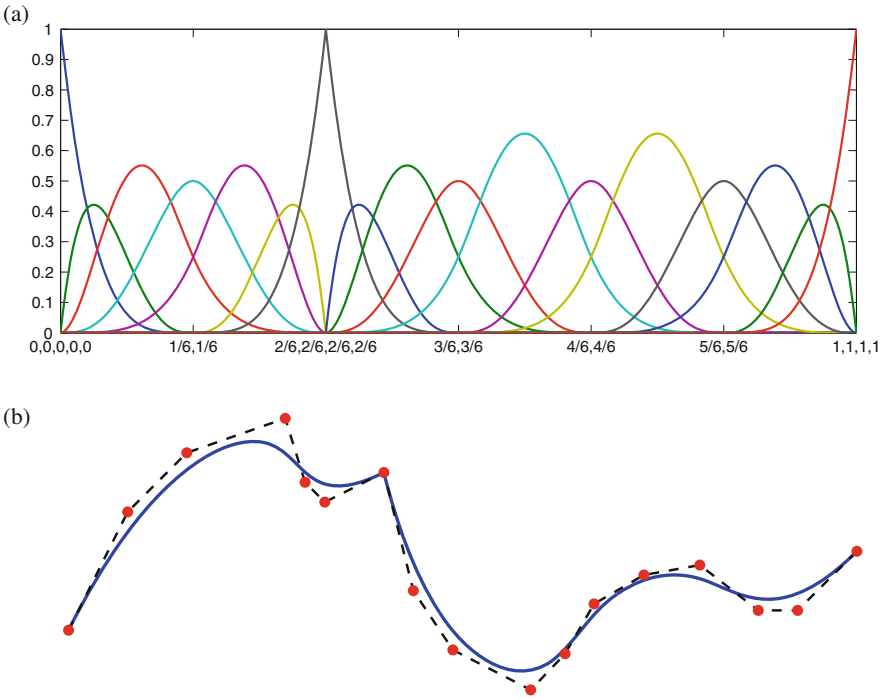


**Fig. 6** Degree elevation algorithm applied to the curve in Fig. 2 with basis functions as in Fig. 1. (**a**) Basis functions and knot vector after applying $p$-refinement. (**b**) Control polygon computed with the degree elevation algorithm

### 2.2.1 Multivariate Splines and NURBS Basis Functions

Let $d$ be the space dimensions (in practical cases, $d = 2, 3$). Assume $n_\ell \in \mathbb{N}$, the degree $p_\ell \in \mathbb{N}$ and the $p_\ell$-open knot vector $\Xi_\ell = \{\xi_{\ell,1}, \ldots, \xi_{\ell,n_\ell+p_\ell+1}\}$ are given, for $\ell = 1, \ldots, d$. We set the polynomial degree vector $\mathbf{p} = (p_1, \ldots, p_d)$ and $\boldsymbol{\Xi} = \Xi_1 \times \ldots \times \Xi_d$. The corresponding knot values without repetitions are given for each direction $\ell$ by $Z_\ell = \{\zeta_{\ell,1}, \ldots, \zeta_{\ell,N_\ell}\}$.

The knots $Z_\ell$ form a Cartesian grid in the *parametric domain* $\hat{\Omega} = (0, 1)^d$, giving the *parametric Bézier mesh*, which is denoted by $\widehat{\mathcal{M}}$:

$$\widehat{\mathcal{M}} = \{Q_{\mathbf{j}} = I_{1,j_1} \times \ldots \times I_{d,j_d} \text{ such that } I_{\ell,j_\ell} = (\zeta_{\ell,j_\ell}, \zeta_{\ell,j_\ell+1}) \text{ for } 1 \leq j_\ell \leq N_\ell - 1\}. \tag{15}$$

For a generic Bézier element $Q_{\mathbf{j}} \in \widehat{\mathcal{M}}$, we also define its *support extension* $\tilde{Q}_{\mathbf{j}} = \tilde{I}_{1,j_1} \times \ldots \times \tilde{I}_{d,j_d}$, with $\tilde{I}_{\ell,j_\ell}$ the univariate support extension given by (7).

B-spline spaces are defined by tensor product. We first introduce the set of multi-indices $\mathbf{I} = \{\mathbf{i} = (i_1, \ldots, i_d) : 1 \leq i_\ell \leq n_\ell\}$, and for each multi-index

$\mathbf{i} = (i_1, \ldots, i_d)$, we define the local knot vector $\boldsymbol{\Xi}_{\mathbf{i},\mathbf{p}} = \Xi_{i_1,p_1} \times \ldots \times \Xi_{i_d,p_d}$. Then we introduce the set of multivariate B-splines

$$\left\{ \hat{B}_{\mathbf{i},\mathbf{p}}(\boldsymbol{\zeta}) = \hat{B}_{i_1,p_1}(\zeta_1) \ldots \hat{B}_{i_d,p_d}(\zeta_d),\ \forall \mathbf{i} \in \mathbf{I} \right\}. \tag{16}$$

The spline space in the parametric domain $\hat{\Omega}$ is then

$$S_{\mathbf{p}}(\boldsymbol{\Xi}) = \text{span}\{\hat{B}_{\mathbf{i},\mathbf{p}}(\boldsymbol{\zeta}),\ \mathbf{i} \in \mathbf{I}\},$$

which is the space of piecewise polynomials of degree $\mathbf{p}$ with the regularity across Bézier elements given by the knots multiplicity.

Multivariate NURBS are defined as rational tensor product B-splines. Given a set of *weights* $\{w_\mathbf{i},\ \mathbf{i} \in \mathbf{I}\}$, and the weight function $W(\boldsymbol{\zeta}) = \sum_{\mathbf{j}\in\mathbf{I}} w_\mathbf{j} \hat{B}_{\mathbf{j},\mathbf{p}}(\boldsymbol{\zeta})$, we define the NURBS basis functions

$$\hat{N}_{\mathbf{i},\mathbf{p}}(\boldsymbol{\zeta}) = \frac{w_\mathbf{i} \hat{B}_{\mathbf{i},\mathbf{p}}(\boldsymbol{\zeta})}{\sum_{\mathbf{j}\in\mathbf{I}} w_\mathbf{j} \hat{B}_{\mathbf{j},\mathbf{p}}(\boldsymbol{\zeta})} = \frac{w_\mathbf{i} \hat{B}_{\mathbf{i},\mathbf{p}}(\boldsymbol{\zeta})}{W(\boldsymbol{\zeta})}.$$

The NURBS space in the parametric domain $\hat{\Omega}$ is then

$$N_{\mathbf{p}}(\boldsymbol{\Xi}, W) = \text{span}\{\hat{N}_{\mathbf{i},\mathbf{p}}(\boldsymbol{\zeta}),\ \mathbf{i} \in \mathbf{I}\}.$$

As in the case of NURBS curves, the choice of the weight depends on the geometry to parametrize, and should remain fixed after refinement.

*Remark 2.2* Note that $S_{\mathbf{p}}(\boldsymbol{\Xi}) = \otimes_{\ell=1}^{d} S_{p_\ell}(\Xi_\ell)$. The same is not true for NURBS, since the weight $W$ is not defined from the tensor product of univariate weights.

### 2.2.2 Multivariate Splines and NURBS Geometries

As we did for curves, we can now define parametrizations of multivariate geometries in $\mathbb{R}^m$, $m = 2, 3$, using splines or NURBS. A spline parametrization is any linear combination of B-splines basis functions via control points $\mathbf{c}_\mathbf{i} \in \mathbb{R}^m$

$$\mathbf{F}(\boldsymbol{\zeta}) = \sum_{\mathbf{i}\in\mathbf{I}} \mathbf{c}_\mathbf{i} \hat{B}_{\mathbf{i},\mathbf{p}}(\boldsymbol{\zeta}), \qquad \text{with } \boldsymbol{\zeta} \in \hat{\Omega}, \tag{17}$$

and a NURBS parametrization is just a linear combination of NURBS instead of B-splines. Depending on the values of $d$ and $m$, the map (17) can define a planar surface in $\mathbb{R}^2$ ($d = 2, m = 2$), a manifold in $\mathbb{R}^3$ ($d = 2, m = 3$), or a volume in $\mathbb{R}^3$ ($d = 3, m = 3$).

The refinement algorithms of knot insertion and degree elevation, to recompute the control points in finer spaces, can be generalized to multivariate splines and

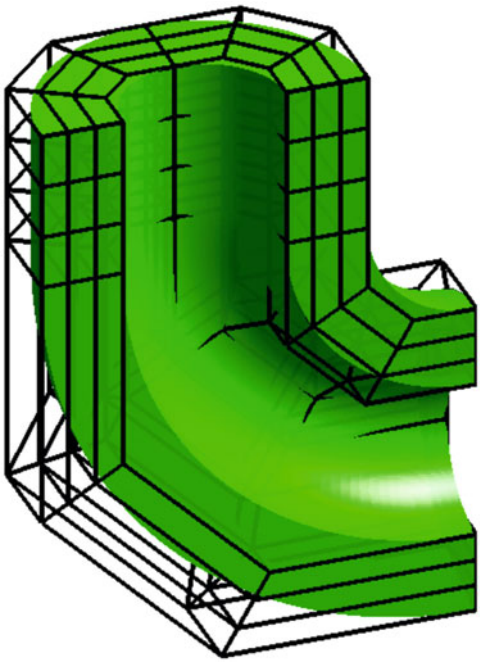

**Fig. 7** The control mesh for the *green pipe* is represented

NURBS [41]. As in the univariate case, given a spline space $S_{\mathbf{p}^0}(\boldsymbol{\Xi}^0)$ we say that $S_{\mathbf{p}}(\boldsymbol{\Xi})$ is a refinement if $S_{\mathbf{p}^0}(\boldsymbol{\Xi}^0) \subset S_{\mathbf{p}}(\boldsymbol{\Xi})$. For NURBS spaces, the weight function $W$ must remain fixed after refinement, and the refinement algorithms also recompute the set of weights. Thus, given a NURBS space $N_{\mathbf{p}^0}(\boldsymbol{\Xi}^0, W)$ we say that $N_{\mathbf{p}}(\boldsymbol{\Xi}, W)$ is a refinement if $N_{\mathbf{p}^0}(\boldsymbol{\Xi}^0, W) \subset N_{\mathbf{p}}(\boldsymbol{\Xi}, W)$. The three possibilities of refinement introduced in Sect. 2.1.3, that is *h*-, *p*- and *k*-refinement, also apply to multivariate splines and NURBS spaces.

The definition of the control polygon is generalized for multivariate splines and NURBS to a control mesh, which is given by the control points $\mathbf{c_i}$. An example of a NURBS volume with its control mesh is given in Fig. 7. Notice that, since B-splines and NURBS functions are not interpolatory, the control mesh is not a mesh on the domain.

Finally, as in the univariate case, it can be proved that when applying knot insertion and degree elevation, the control mesh converges to the parametrized geometry.

## 2.3 Projections and Quasi-interpolation Operators

In this section we introduce quasi-interpolation and projection operators onto the spaces of splines, as described in [46]. These operators will be used later for the

numerical analysis of isogeometric methods. We start introducing the projectors in the univariate case, and then generalize them to the multivariate case by tensor product.

### 2.3.1 Univariate Quasi-interpolants

We first introduce interpolation and projection operators onto the space of univariate splines $S_p(\Xi)$. There are several ways to define projections for splines, but here we only describe the one that will be used in the sequel of the paper.

In the present contribution we will often make use of the following local quasi-uniformity condition on the knot vector, that is a classical assumption in the mathematical isogeometric literature.

**Assumption 2.3** *The partition defined by the knots $\zeta_1, \zeta_2, \ldots, \zeta_N$ is locally quasi-uniform, that is, there exists a constant $\theta \geq 1$ such that the mesh sizes $h_i = \zeta_{i+1} - \zeta_i$ satisfy the relation $\theta^{-1} \leq h_i/h_{i+1} \leq \theta$, for $i = 1, \ldots, N-2$.*

Since splines are not in general interpolatory, a common way to define projections is by giving a dual basis, i.e.,

$$\Pi_{p,\Xi} : L^2([0,1]) \to S_p(\Xi), \qquad \Pi_{p,\Xi}(u) = \sum_{j=1}^{n} \lambda_{j,p}(u)\hat{B}_{j,p}, \tag{18}$$

where $\lambda_{j,p}$ are a set of dual functionals verifying

$$\lambda_{j,p}(\hat{B}_{k,p}) = \delta_{jk}, \tag{19}$$

$\delta_{jk}$ being the standard Kronecker symbol. It is trivial to prove that, thanks to this property, the quasi-interpolant $\Pi_{p,\Xi}$ preserves splines, that is

$$\Pi_{p,\Xi}(u) = u, \quad \forall u \in S_p(\Xi). \tag{20}$$

In the spline theory, projections defined by dual basis are commonly called *quasi-interpolant operators*. There are several possible choices for the dual basis $\{\lambda_{j,p}\}_{j=1}^{n}$, and we refer the reader to [33] for a very general construction and theory about quasi-interpolant operators. However, the most common choice in the theoretical study of isogeometric methods, and the one that we will adopt from now on, is the one detailed in [46, Sect. 4.6]. In this paper we do not give any details about the construction of this dual basis, which the interested reader can find in the aforementioned book. Instead, we focus on its mathematical properties. It is proved

that these functionals are dual to B-splines in the sense of (19), and also stable (see [46, Thm. 4.41]), that is

$$|\lambda_{j,p}(u)| \leq C(\xi_{j+p+1} - \xi_j)^{-1/2} \|u\|_{L^2(\xi_j, \xi_{j+p+1})}, \tag{21}$$

where the constant $C$ depends on the degree $p$.

The reason for this choice of the dual functionals is mainly historical (it is the same dual basis used in [6], the first paper on the numerical analysis of isogeometric methods), but also because it satisfies the following important stability property.

**Proposition 2.4** *For any non empty knot span $I_i = (\zeta_i, \zeta_{i+1})$ it holds*

$$\|\Pi_{p,\Xi}(u)\|_{L^2(I_i)} \leq C\|u\|_{L^2(\tilde{I}_i)}, \tag{22}$$

*where the constant $C$ depends only upon the degree $p$, and $\tilde{I}_i$ is the support extension defined in (7). Moreover, if Assumption 2.3 holds, we also have*

$$|\Pi_{p,\Xi}(u)|_{H^1(I_i)} \leq C|u|_{H^1(\tilde{I}_i)}, \tag{23}$$

*with the constant $C$ depending only on $p$ and $\theta$, and where $H^1$ denotes the Sobolev space of order one, endowed with the standard norm and seminorm.*

*Proof* We first show (22). There exists a unique index $j$ such that $I_i = (\zeta_i, \zeta_{i+1}) = (\xi_j, \xi_{j+1})$, and using the properties of B-splines at the beginning of Sect. 2.1.1, and in particular their local support, it immediately follows that

$$\left\{\ell \in \{1, 2, \ldots, n\} \,:\, \mathrm{supp}(\hat{B}_{\ell,p}) \cap I_i \neq \emptyset\right\} = \{j-p, j-p+1, \ldots, j\}. \tag{24}$$

Let $h_i$ denote the length of $I_i$ and $\tilde{h}_i$ indicate the length of $\tilde{I}_i$. First by definition (18), then recalling that the B-spline basis is positive and a partition of unity, we get

$$\begin{aligned}\|\Pi_{p,\Xi}(u)\|_{L^2(I_i)} &= \Big\| \sum_{\ell=j-p}^{j} \lambda_{\ell,p}(u)\hat{B}_{\ell,p} \Big\|_{L^2(I_i)} \leq \max_{j-p\leq\ell\leq j} |\lambda_{\ell,p}(u)| \Big\| \sum_{\ell=j-p}^{j} \hat{B}_{\ell,p} \Big\|_{L^2(I_i)} \\ &= h_i^{1/2} \max_{j-p\leq\ell\leq j} |\lambda_{\ell,p}(u)|.\end{aligned}$$

We now apply bound (21) and obtain

$$\begin{aligned}\|\Pi_{p,\Xi}(u)\|_{L^2(I_i)} &\leq C h_i^{1/2} \max_{j-p\leq\ell\leq j} (\xi_{\ell+p+1} - \xi_\ell)^{-1/2} \|u\|_{L^2(\xi_\ell, \xi_{\ell+p+1})} \\ &\leq C h_i^{1/2} \max_{j-p\leq\ell\leq j} (\xi_{\ell+p+1} - \xi_\ell)^{-1/2} \|u\|_{L^2(\tilde{I}_i)},\end{aligned}$$

that yields (22) since clearly $h_i \leq (\xi_{\ell+p+1} - \xi_\ell)$ for all $\ell$ in $\{j-p, \ldots, j\}$.

We now show (23). For any real constant $c$, since the operator $\Pi_{p,\Xi}$ preserves constant functions and using a standard inverse estimate for polynomials on $I_i$, we get

$$|\Pi_{p,\Xi}(u)|_{H^1(I_i)} = |\Pi_{p,\Xi}(u)-c|_{H^1(I_i)} = |\Pi_{p,\Xi}(u-c)|_{H^1(I_i)} \leq Ch_i^{-1}\|\Pi_{p,\Xi}(u-c)\|_{L^2(I_i)}.$$

We now apply (22) and a standard approximation estimate for constant functions, yielding

$$|\Pi_{p,\Xi}(u)|_{H^1(I_i)} \leq Ch_i^{-1}\|u-c\|_{L^2(\tilde{I}_i)} \leq Ch_i^{-1}\tilde{h}_i|u|_{H^1(\tilde{I}_i)}.$$

Using Assumption 2.3, it is immediate to check that $\tilde{h}_i \leq Ch_i$ with $C = C(p,\theta)$ so that (23) follows. □

*Remark 2.5* The operator $\Pi_{p,\Xi}$ can be modified in order to match boundary conditions, and stability results similar to the ones in the previous proposition also hold. We refer to [8] for details.

### 2.3.2 Multivariate Quasi-interpolants

The univariate interpolation and quasi-interpolation operators introduced in Sect. 2.3 can be also extended to the multi-dimensional case by a tensor product construction. Let, for $i = 1,\ldots,d$, the symbol $\Pi_{p_i,\Xi_i}$ denote the univariate operators onto the space $S_{p_i}(\Xi_i)$. We define

$$\Pi_{\mathbf{p},\boldsymbol{\Xi}}(u) = (\Pi_{p_1,\Xi_1} \otimes \ldots \otimes \Pi_{p_d,\Xi_d})(u). \tag{25}$$

For a rigorous explanation on the tensorization of the projectors, we refer the reader to [8] and [12, Chap. XVII].

The definition above is given for any smooth $u$ and then extended by continuity to the correct functional space. For the quasi-interpolant of [46] the $L^2$ regularity requirement stays unchanged and the operator $\Pi_{\mathbf{p},\boldsymbol{\Xi}}$ is well defined for all $u \in L^2([0,1]^d)$.

Finally, it is important to note that, since the univariate quasi-interpolants are defined from a dual basis as in (18), then the multivariate quasi-interpolant is also defined from a dual basis. Indeed, we have

$$\Pi_{\mathbf{p},\boldsymbol{\Xi}}(u) = \sum_{\mathbf{i}\in\mathbf{I}} \lambda_{\mathbf{i},\mathbf{p}}(u)\hat{B}_{\mathbf{i},\mathbf{p}},$$

where each dual functional is defined from the univariate dual bases by the expression

$$\lambda_{\mathbf{i},\mathbf{p}} = \lambda_{i_1,p_1} \otimes \ldots \otimes \lambda_{i_d,p_d}. \tag{26}$$

Tensor product of dual functionals are defined exactly as the tensor product of the projectors, and all details can be found in [12, Chap. XVII].

# 3 Discrete Spaces for Isogeometric Methods: Application to Elliptic Problems

In this section we present the basic concepts of isogeometric methods, as they were introduced in [28]. We will assume that our computational geometry is given as a NURBS geometry, and the finite-dimensional space for the discretization will be defined directly on the NURBS geometry and on its mesh, possibly refined applying the algorithms of knot insertion and degree elevation.

We start this section by introducing the discrete spaces and basis functions in a domain defined as a NURBS geometry. These spaces are then used for the discretization of a simple elliptic scalar problem. We explain how to perform the numerical analysis of the discrete problem, and also give details on the implementation. We end the section by describing how the method can be generalized to more complicated geometries, that are defined with several NURBS patches.

## *3.1 Isogeometric Spaces in a Single Patch Domain*

We start defining the spaces in the case that the physical domain is given as the image of the unit square, or the unit cube, by a single NURBS parametrization.

### 3.1.1 The Parametrization and the Mesh

In isogeometric methods, the computational domain $\Omega$, which we will also call *physical domain*, is often supposed to be given through a NURBS transformation of the parametric domain $\hat{\Omega}$, defined by a set of control points as in (17). More precisely, for a given degree vector $\mathbf{p}^0$, the knot vectors $\boldsymbol{\Xi}^0$ and a weight function $W \in S_{\mathbf{p}^0}(\boldsymbol{\Xi}^0)$, we assume that there exists a map $\mathbf{F} \in (N_{\mathbf{p}^0}(\boldsymbol{\Xi}^0, W))^d$ such that $\Omega = \mathbf{F}(\hat{\Omega})$, as in Fig. 8.

We have seen in Sect. 2.2.2 that the parametrization $\mathbf{F}$ is related to the control mesh, given by the control points. Apart from defining the geometry, the control grid also plays a role in the definition of the space in multi-patch domains, as we will see below. However, the control grid is not a real mesh in $\Omega$, because in general the control points do not lie in the geometry.

In order to define a mesh in $\Omega$, we consider the image through $\mathbf{F}$ of the partition given by the knot vectors, as shown in Fig. 8. This mesh, which is commonly

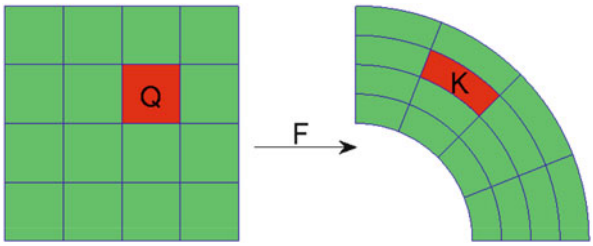


**Fig. 8** Mesh $\widehat{\mathcal{M}}$ in the parametric domain, and its image $\mathcal{M}$ in the physical domain

referred to as the *Bézier mesh*, plays a similar role to the mesh in finite elements, and is the mesh used for the numerical computations, as we will see in Sect. 3.2.2.

Let us now define this mesh in a precise way. Assume that $N_{\mathbf{p}}(\boldsymbol{\Xi}, W)$ is a refinement of $N_{\mathbf{p}}(\boldsymbol{\Xi}^0, W)$, in the sense explained in Sect. 2.2. We have introduced in (15) the parametric Bézier mesh $\widehat{\mathcal{M}}$ as the Cartesian grid associated to the knot vectors $\boldsymbol{\Xi}$. We define now the physical Bézier mesh, or simply Bézier mesh, as the image of the (open) elements in $\widehat{\mathcal{M}}$ through $\mathbf{F}$:

$$\mathcal{M} := \{K \subset \Omega : K = \mathbf{F}(Q), Q \in \widehat{\mathcal{M}}\}, \tag{27}$$

see Fig. 8. The elements of $\mathcal{M}$ are called Bézier elements. The meshes for the coarsest knot vector $\boldsymbol{\Xi}^0$ will be denoted by $\widehat{\mathcal{M}}_0$ and $\mathcal{M}_0$.

For any element $K = \mathbf{F}(Q) \in \mathcal{M}$, we define its support extension as $\tilde{K} = \mathbf{F}(\tilde{Q})$, with $\tilde{Q}$ the support extension of $Q$, defined in Sect. 2.2.1. Moreover, we denote the element size of any element $Q \in \widehat{\mathcal{M}}$ by $h_Q = \text{diam}(Q)$, and the global mesh size is $h = \max\{h_Q : Q \in \widehat{\mathcal{M}}\}$. Analogously, we define the element sizes $h_K = \text{diam}(K)$ and $h_{\tilde{K}} = \text{diam}(\tilde{K})$. Assumption 3.1 below will ensure that $h_Q \simeq h_K$.

In the following, we will make use of a regularity assumption on $\mathbf{F}$.

**Assumption 3.1 (Regularity of F)** *The parametrization $\mathbf{F} : \hat{\Omega} \to \Omega$ is a bi-Lipschitz homeomorphism. Moreover, $\mathbf{F}_{|\bar{Q}}$ is in $C^\infty(\bar{Q})$ for all $Q \in \widehat{\mathcal{M}}_0$, where $\bar{Q}$ denotes the closure of $Q$, and $\mathbf{F}^{-1}_{|\bar{K}}$ is in $C^\infty(\bar{K})$ for all $K \in \mathcal{M}_0$.*

The assumption prevents the existence of singularities and self-intersections in the parametrization $\mathbf{F}$. In the two-dimensional case, for instance, the most common singularities occur when a rectangular element in $\widehat{\mathcal{M}}$ is mapped through $\mathbf{F}$ to a curvilinear triangular element (see Fig. 9), which is not allowed under Assumption 3.1. We remark that the study of isogeometric methods in the case of a parametrization with singularities has been already started in [49] and [50].

### 3.1.2 Isogeometric Discrete Spaces and Basis Functions

The discrete approximation spaces for isogeometric methods, as they were introduced in [28], are constructed by applying the *isoparametric* paradigm. That is,

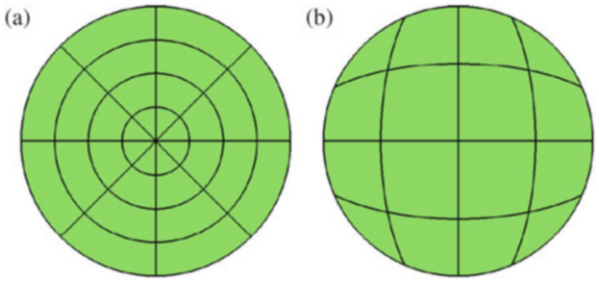


**Fig. 9** Two possible singular parametrizations of the circle. (**a**) One singularity at the origin. (**b**) Four singularities on the boundary

they are constructed using the same NURBS space that defines the geometry, and mapped to the physical domain through the NURBS parametrization $\mathbf{F}$.

In detail, let $\hat{V}_h = N_{\mathbf{p}}(\boldsymbol{\Xi}, W)$ be a refinement of $N_{\mathbf{p}^0}(\boldsymbol{\Xi}^0, W)$, we define the *isogeometric discrete space*:

$$V_h = \{\hat{u}_h \circ \mathbf{F}^{-1} : \hat{u}_h \in \hat{V}_h\}, \tag{28}$$

where $h$ parametrizes the family of spaces and stands, as usual, for the mesh size. It is also important to provide a basis for the discrete space $V_h$. Under Assumption 3.1, it is clear that

$$V_h = \text{span}\{N_{\mathbf{i},\mathbf{p}}(\mathbf{x}) := \hat{N}_{\mathbf{i},\mathbf{p}} \circ \mathbf{F}^{-1}(\mathbf{x}),\ \mathbf{i} \in \mathbf{I}\}, \tag{29}$$

and the functions $N_{\mathbf{i},\mathbf{p}}$ form a basis of the space $V_h$. We show in Fig. 10 an example of two basis functions defined in the parametric domain, and how they are mapped to the physical domain.

In some problems it will be necessary to make use of spaces with boundary conditions. Let $\Gamma_D \subset \partial\Omega$ be a non-empty part of the boundary. We denote by

$$V_{h,\Gamma_D} = \{u_h \in V_h \ :\ u_h|_{\Gamma_D} = 0\}, \tag{30}$$

the space with homogeneous Dirichlet boundary conditions on $\Gamma_D$. For simplicity, in the following we will always consider the following assumption.

**Assumption 3.2** *The boundary region $\Gamma_D \subset \partial\Omega$ is the union of full faces of the boundary. More precisely, $\Gamma_D = \mathbf{F}(\hat{\Gamma}_D)$, with $\hat{\Gamma}_D$ a collection of full faces of the parametric domain $\hat{\Omega}$.*

*Remark 3.3* Notice that the space $V_h$ is defined from the NURBS space $N_{\mathbf{p}}(\boldsymbol{\Xi}, W)$. Abandoning the isoparametric paradigm, we can analogously construct the approximation space as the image through the NURBS parametrization $\mathbf{F}$ of the spline space

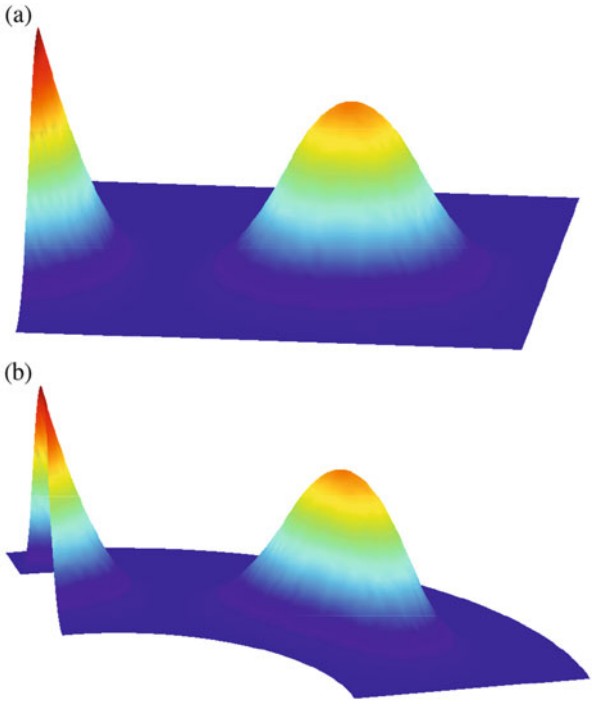


**Fig. 10** Basis functions in the parametric and the physical domain, with the same geometry as in Fig. 8. (**a**) Two basis functions in the parametric domain. (**b**) The basis functions mapped to the physical domain

$\hat{V}_h = S_{\mathbf{p}}(\boldsymbol{\Xi})$, which is a refinement of $S_{\mathbf{p}^0}(\boldsymbol{\Xi}^0)$. In this case the basis functions are $B_{\mathbf{i},\mathbf{p}} := \hat{B}_{\mathbf{i},\mathbf{p}} \circ \mathbf{F}^{-1}$.This non-isoparametric approach will be used in Sect. 5.

## *3.2 Application to Scalar Elliptic Problems*

After having introduced the discrete spaces to be used, we now apply isogeometric methods for the solution of a scalar elliptic problem by Galerkin's method, and present some numerical tests to see the performance of the method. We will discuss about the well posedness of the discrete problem to be solved with isogeometric spaces, and the convergence of the computed numerical solution to the exact one. The theory is the general one of Galerkin's method, and it is indeed very similar to that of the finite element method, therefore we will not give all the details in the proofs. However, the approximation results require the definition of projectors for the isogeometric spaces, that will be presented in Sect. 4.

### 3.2.1 Definition and Analysis of the Problem

As the first example we apply the isogeometric method to the discretization of the scalar elliptic problem (advection-reaction-diffusion equation):

$$\begin{aligned} -\operatorname{div}(A(\mathbf{x})\operatorname{grad} u) + \mathbf{b}(\mathbf{x})\cdot\operatorname{grad} u + c(\mathbf{x})u &= f \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \Gamma_D, \\ \frac{\partial u}{\partial \mathbf{n}} &= g_N \quad \text{on } \Gamma_N, \end{aligned}$$

with $f \in L^2(\Omega)$ and $g_N \in L^2(\Gamma_N)$. For the coefficients, we assume that $\mathbf{b} \in \mathbf{W}^{1,\infty}(\Omega)$, $c \in L^\infty(\Omega)$ and $A(\mathbf{x}) = (a_{ij}(\mathbf{x}))_{1\le i,j\le d}$ is a symmetric tensor with $a_{ij} \in L^\infty(\Omega)$, which satisfies the ellipticity condition

$$\sum_{i,j=1}^{d} a_{ij}(\mathbf{x})\phi_i\phi_j \ge \alpha|\boldsymbol{\phi}|^2 \quad \forall \boldsymbol{\phi} \in \mathbb{R}^d.$$

We introduce the following bilinear form defined on $H^1(\Omega)$

$$a(v,w) = \int_\Omega A(\mathbf{x})\operatorname{grad} v\cdot\operatorname{grad} w\,\mathrm{d}\mathbf{x} + \int_\Omega \mathbf{b}(\mathbf{x})\cdot\operatorname{grad} v\, w\,\mathrm{d}\mathbf{x} + \int_\Omega c(\mathbf{x})v\,w\,\mathrm{d}\mathbf{x},$$

which is continuous and coercive, provided $c(\mathbf{x}) - \frac{1}{2}\operatorname{div}\mathbf{b}(\mathbf{x}) \ge 0$ a.e. in $\Omega$. Introducing the space $V_{0,\Gamma_D} = \{v \in H^1(\Omega) : v|_{\Gamma_D} = 0\}$, the variational formulation of the problem reads: Find $u \in V_{0,\Gamma_D}$ such that

$$a(u,v) = \int_\Omega f\,v\,\mathrm{d}\mathbf{x} + \int_{\Gamma_N} g_N\,v\,\mathrm{d}\Gamma, \quad \forall v \in V_{0,\Gamma_D}. \tag{31}$$

Existence and uniqueness of the solution follows from the continuity and coercivity of $a(\cdot,\cdot)$, using Lax-Milgram lemma. The coercivity of $a(\cdot,\cdot)$ also implies the stability of the problem: there exists a constant $C > 0$ such that

$$\|u\|_{H^1(\Omega)} \le C\left(\|f\|_{L^2(\Omega)} + \|g_N\|_{L^2(\Gamma_N)}\right). \tag{32}$$

We assume now that the domain $\Omega$ is given by a NURBS parametrization $\mathbf{F}$, and that $\Gamma_D \subset \partial\Omega$ is the image through $\mathbf{F}$ of a collection of faces of $\hat{\Omega}$, as in Assumption 3.2. Then, we define the finite-dimensional space $V_{h,\Gamma_D}$ as in (30), and the discrete version of problem (31) is: Find $u_h \in V_{h,\Gamma_D}$ such that

$$a(u_h,v_h) = \int_\Omega f\,v_h\,\mathrm{d}\mathbf{x} + \int_{\Gamma_N} g_N\,v_h\,\mathrm{d}\Gamma, \quad \forall v_h \in V_{h,\Gamma_D}. \tag{33}$$

As in the continuous case, existence and uniqueness of the solution follows from Lax-Milgram lemma, and the stability result (32) is also valid for $u_h$. Finally, we have the following error estimate.

**Theorem 3.4** *Let $u$ the solution to (31) belong to $H^{s+1}(\Omega)$, with $s > 0$. Then there exists a constant $C$ independent of the mesh size $h$ such that the solution to (33) satisfies*

$$\|u - u_h\|_{H^1(\Omega)} \leq Ch^q \|u\|_{H^{q+1}(\Omega)},$$

*with $q = \min\{p, s\}$.*

*Proof* The complete proof requires the use of quasi-interpolants and the approximation results that will be introduced in Sect. 4. We start by applying Céa's lemma:

$$\|u - u_h\|_{H^1(\Omega)} \leq C \inf_{v_h \in V_{h,\Gamma_D}} \|u - v_h\|_{H^1(\Omega)}.$$

We now use the quasi-interpolant from Sect. 4.3.1, in particular $\Pi_{V_{h,\Gamma_D}}$ from Remark 4.17. This has the same approximation properties as $\Pi_{V_h}$, for which is valid the estimate in Corollary 4.16 with $r = 1$. Therefore, we have

$$\inf_{v_h \in V_{h,\Gamma_D}} \|u - v_h\|_{H^1(\Omega)} \leq C\|u - \Pi_{V_{h,\Gamma_D}} u\|_{H^1(\Omega)} \leq Ch^q \|u\|_{H^{q+1}(\Omega)},$$

which completes the proof. □

### 3.2.2 Some Notes About the Implementation

We now present a brief explanation on how to solve the discrete problem (33) with the isogeometric method. Being a Galerkin's method, the implementation is very similar to that of finite elements. For more detailed explanations we refer the reader to [21, Chap. 3] and [25].

First of all we define our trial function $u_h$ as a linear combination of the basis functions in (29), that is

$$u_h = \sum_{\mathbf{j} \in \mathbf{I}} \alpha_{\mathbf{j}} N_{\mathbf{j},\mathbf{p}}.$$

We substitute this expression into (33), and test again the basis functions of $V_h$ to obtain

$$\sum_{\mathbf{j} \in \mathbf{I}} a(N_{\mathbf{j},\mathbf{p}}, N_{\mathbf{i},\mathbf{p}})\, \alpha_{\mathbf{j}} = \int_\Omega f N_{\mathbf{i},\mathbf{p}}\, \mathrm{d}\mathbf{x} + \int_{\Gamma_N} g_N N_{\mathbf{i},\mathbf{p}}\, \mathrm{d}\Gamma, \quad \forall \mathbf{i} \in \mathbf{I}.$$

Thus, the problem becomes to find the coefficients $\alpha_{\mathbf{j}}$ such that the previous equation holds. As in finite elements, we can rewrite the previous problem as a linear system, with the entries of the matrix and the right-hand side given by

$$K_{\mathbf{ij}} = a(N_{\mathbf{j},\mathbf{p}}, N_{\mathbf{i},\mathbf{p}}), \qquad b_{\mathbf{i}} = \int_{\Omega} f N_{\mathbf{i},\mathbf{p}} \, \mathrm{d}\mathbf{x} + \int_{\Gamma_N} g_N N_{\mathbf{i},\mathbf{p}} \, \mathrm{d}\Gamma.$$

Notice that it is necessary to give a numbering for the multi-indices $\mathbf{i} \in \mathbf{I}$. Since the spaces are tensor product, the simplest and most usual numbering is generated by lexicographical ordering.

Now the main difficulty is the computation of the integrals to obtain the entries of the matrix and the right-hand side. As in finite elements, these integrals can be expressed as the sum of the integrals on the elements of the mesh, that in our case is the Bézier mesh, and by a simple change of variable we can write them in the parametric domain. For instance, the first term of the bilinear form $a(N_{\mathbf{j},\mathbf{p}}, N_{\mathbf{i},\mathbf{p}})$ is written as follows:

$$\int_{\Omega} A(\mathbf{x}) \operatorname{grad} N_{\mathbf{i},\mathbf{p}}(\mathbf{x}) \cdot \operatorname{grad} N_{\mathbf{j},\mathbf{p}}(\mathbf{x}) \, \mathrm{d}\mathbf{x} = \sum_{K \in \mathcal{M}} \int_{K} A(\mathbf{x}) \operatorname{grad} N_{\mathbf{i},\mathbf{p}}(\mathbf{x}) \cdot \operatorname{grad} N_{\mathbf{j},\mathbf{p}}(\mathbf{x}) \, \mathrm{d}\mathbf{x}$$

$$= \sum_{Q \in \widehat{\mathcal{M}}} \int_{Q} A(\mathbf{F}(\hat{\mathbf{x}})) \, D\mathbf{F}^{-\top} \widehat{\operatorname{grad}} \hat{N}_{\mathbf{i},\mathbf{p}}(\hat{\mathbf{x}}) \cdot D\mathbf{F}^{-\top} \widehat{\operatorname{grad}} \hat{N}_{\mathbf{i},\mathbf{p}}(\hat{\mathbf{x}}) \, |D\mathbf{F}(\hat{\mathbf{x}})| \, \mathrm{d}\hat{\mathbf{x}}.$$

These integrals can be computed numerically by applying standard Gaussian quadrature rules. Notice that $D\mathbf{F}(\hat{\mathbf{x}})$ is not constant on each element, hence it must be computed for each quadrature point.

For the implementation of boundary conditions, homogeneous Dirichlet conditions are imposed setting to zero the degrees of freedom associated to boundary control points. To impose non-homogeneous Dirichlet conditions, a lifting $\tilde{u}_h \in H^1(\Omega)$, such that $\tilde{u}_h|_{\Gamma_D} = g_D$ must be constructed. Since B-splines and NURBS are not interpolatory at the knots, the lifting can be computed using some surface fitting technique, such as the least squares approximation or a quasi-interpolant, see for instance [21, Chap. 3], [26] and [51]. Neumann boundary conditions, instead, do not require any special treatment, and it is enough to compute the integrals $\int_{\Gamma_N} g_N N_{\mathbf{i},\mathbf{p}} \, \mathrm{d}\Gamma$, using numerical quadrature analogously to the volumetric integrals.

Finally, it is worth to mention that there exist several open-source codes available where isogeometric methods have been implemented. GeoPDEs [25] is an Octave/Matlab code that can serve for a first approach to understand the basics of isogeometric methods, and the one we have used for the numerical experiments in this paper. Another Matlab code, focused on solid mechanics, is MIGFEM [37]. For those readers aiming at high performance computing and large applications, we recommend the general purpose C++ library igatools [39], or the PETSc based library PetIGA [22].

*Remark 3.5* For the evaluation of the basis functions and the parametrization $\mathbf{F}$, one can use standard algorithms for splines and NURBS, see for instance [41]. An alternative is to use the so-called Bézier extraction operators, and to write the B-splines as linear combination of Bernstein polynomials on each knot span [13]. The advantage of this approach is that, since Bernstein polynomials are the same on every knot span (up to scaling), it is possible to do most of the computations in a reference element, and thus it becomes easier to re-utilize an existing finite element code. We refer to [13] for details.

### 3.2.3 Numerical Tests

Test 1

The first numerical test consists on a simple geometry: a quarter of a ring with inner and outer radius equal to 1 and 2, respectively, and described through a quadratic NURBS parametrization, as the one in Fig. 8. We solve the advection-reaction-diffusion problem with $A(\mathbf{x})$ the identity matrix, $\mathbf{b}(\mathbf{x}) = (-x_2, x_1)$, and $c(\mathbf{x}) = 1$. The right-hand side $f$ is imposed to obtain the exact solution $u = \mathrm{e}^{x_1}\sin(x_2)$, which is infinitely smooth, and non-homogenous Dirichlet boundary conditions are imposed on the boundary.

We solve the problem in a set of successively refined meshes, the coarsest three meshes are plotted in Fig. 11, for degree $p$ varying from 2 to 4, and in NURBS spaces of maximum ($C^{p-1}$) and minimum ($C^0$) continuity. In Fig. 12 we present the error in $H^1$-norm with respect to the mesh size $h$, and with respect to the number of degrees of freedom $N_{\text{dof}}$. The results in terms of the mesh size confirm the estimate of Theorem 3.4, and show that in the same mesh lower continuity gives better accuracy, since the $C^{p-1}$ space is contained in the $C^0$ one. In terms of the degrees of freedom, the result always converges like $O(N_{\text{dof}}^{-p/2})$, and in this case the $C^{p-1}$ spaces give better results.

It has been observed by several authors (see [21] and references therein) that high continuity splines improve the accuracy per degree of freedom with respect to finite elements. However, as has been observed in [19] the computational cost per degree

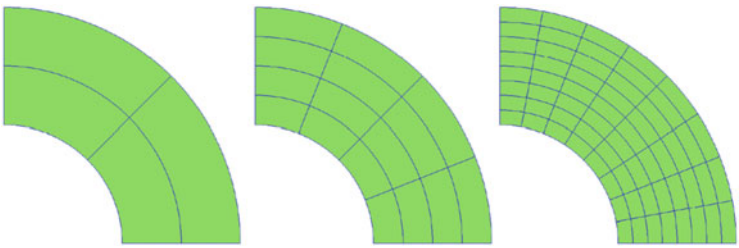

**Fig. 11** The first three meshes on which we solve Test 1

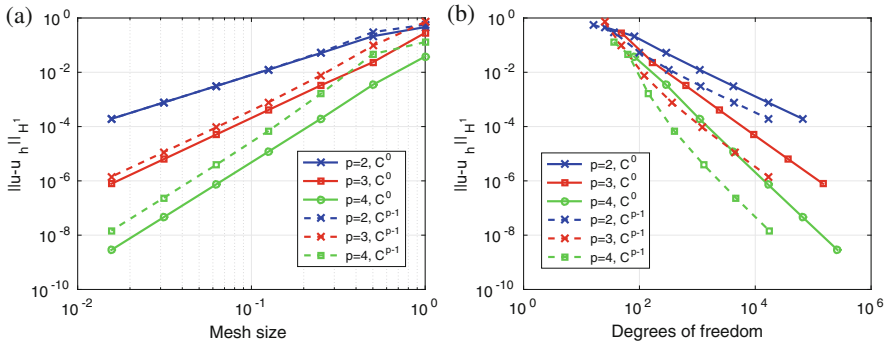


**Fig. 12** Absolute error in $H^1$-norm in the quarter of a ring. (**a**) Error in terms of the mesh size. (**b**) Error in terms of the degrees of freedom

of freedom is also higher for high continuity splines/NURBS. The development of efficient techniques for the implementation of isogeometric methods with high continuity functions is an important topic of research.

Test 2

For the second numerical test we choose the same problem with discontinuous coefficients given in [35] and [40], and derived from the results in [30]. The domain is the square $\Omega = (-1, 1)^2$, with the coefficient $A = k_i I$ in the $i^{th}$ quadrant, with $I$ the identity tensor, and now with $\mathbf{b} = \mathbf{0}$ and $c = 0$. An exact weak solution for $f = 0$, and for non-homogeneous Dirichlet boundary conditions, is given in polar coordinates by $u = r^\gamma \mu(\theta)$, with

$$\mu(\theta) = \begin{cases} \cos((\pi/2-\sigma)\gamma)\,\cos((\theta-\pi/2+\rho)\gamma) & \text{if } 0 \le \theta \le \pi/2, \\ \cos(\rho\gamma)\,\cos((\theta-\pi+\sigma)\gamma) & \text{if } \pi/2 \le \theta \le \pi, \\ \cos(\sigma\gamma)\,\cos((\theta-\pi-\rho)\gamma) & \text{if } \pi \le \theta \le 3\pi/2, \\ \cos((\pi/2-\rho)\gamma)\,\cos((\theta-3\pi/2-\sigma)\gamma) & \text{if } 3\pi/2 \le \theta \le 2\pi. \end{cases}$$

The scalars $\gamma$, $\rho$ and $\sigma$, and the coefficients $k_i$ must be chosen in such a way that the function $\mu$ satisfies the condition along the interfaces

$$k_{i-1}\mu'(\theta_i^-) = k_i \mu'(\theta_i^+), \quad i = 1, \ldots, 4,$$

with $\theta_i = (i-1)\pi/2$, and assuming for convenience $k_0 = k_4$. In the most common examples, the materials are configured in a checkerboard pattern, like in Fig. 13a. In this case, one can choose $\gamma \in (0, 1]$ and then set $\rho = \pi/4$, $\sigma = (1+\frac{1}{\gamma})\frac{\pi}{2} - \rho$, and the interface conditions are satisfied with $k_1 = k_3 = -\tan(\gamma\sigma)$ and $k_2 = k_4 = \tan(\gamma\rho)$. This produces a singular solution $u \notin H^{1+\gamma}(\Omega)$ and $u \in H^{1+\gamma-\varepsilon}(\Omega)$ for any $\varepsilon > 0$.

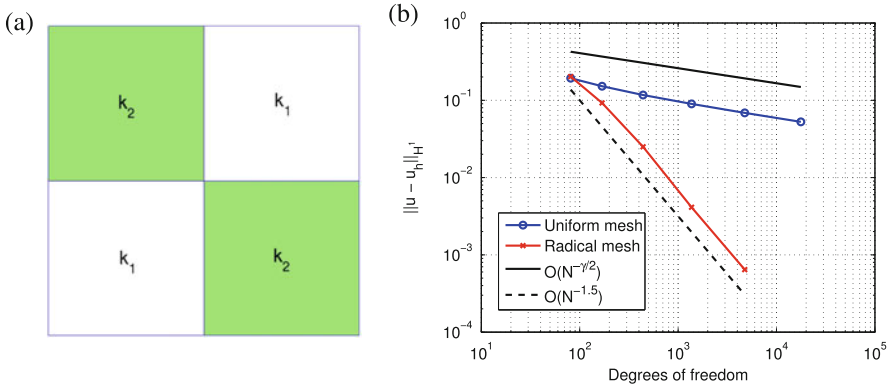


**Fig. 13** Checkerboard domain: distribution of the coefficients and error in $H^1$ norm. (**a**) Coefficients in the checkerboard pattern. (**b**) Absolute error in $H^1$-norm

We have solved the problem in the checkerboard domain for the ratio $k_1/k_2 = 10$, which corresponds to $\gamma \approx 0.39$. The square is first divided into four elements, setting $C^0$ continuity along the interfaces, and then refined using cubic splines and $C^2$ continuity along the new mesh lines. The results in Fig. 13b show that, in a uniform mesh, the absolute error in $H^1$-norm converges like $O(N_{\text{dof}}^{-\gamma/2})$, with $N_{\text{dof}}$ the number of degrees of freedom. Using a tensor product radical graded mesh like in [9] (see also [5, Sect. 3.4]), with the univariate knots on each patch defined as $\zeta_j = \left(\frac{j-1}{N-1}\right)^{\alpha}$ and $\alpha = 7$, we can recover the optimal convergence $O(N_{\text{dof}}^{-p/2})$. However, for stronger singularities and high degree $p$, the optimal convergence may not be recovered unless higher precision is employed in the numerical computation, as already noticed for the finite element case in [5, Sect. 3.4] and references therein.

*Remark 3.6* In all the previous tests we have considered a diffusion-dominated or pure diffusion problem. Design and numerical benchmarking of isogeometric methods for advection dominated or reaction dominated problems can be found in [14, 24, 28] and [48].

## *3.3 Isogeometric Spaces in a Multi-patch Domain*

In the previous sections the domain $\Omega$ was defined as the image through $\mathbf{F}$ of the unit square or the unit cube. In order to enhance flexibility and allow for more complex geometries, we generalize the definition of tensor-product spline and NURBS parametrized domains to domains that are union of several images of squares or cubes, and that we call multi-patch domains.

Let $\Omega$ be an open, bounded and connected set, which is defined as the union of $M_p$ subdomains, in the form

$$\overline{\Omega} = \bigcup_{j=1}^{M_p} \overline{\Omega^{(j)}}, \tag{34}$$

where the subdomains $\Omega^{(j)} = \mathbf{F}^{(j)}(\hat{\Omega})$ are referred to as *patches*, and are assumed to be disjoint. Each patch has its own parametrization $\mathbf{F}^{(j)}$, defined using the NURBS space $N_{\mathbf{p}^{(j)}}(\boldsymbol{\Xi}^{(j)}, W^{(j)})$, which differs from patch to patch. The whole $\Omega$ is then referred to as a *multi-patch domain*. In the following, the superindex $(j)$ will identify the mathematical entities that are different on each patch $\Omega^{(j)}$, such as the basis functions and the control points.

Let us assume for simplicity that the degree vector $\mathbf{p}^{(j)} = \mathbf{p}$ is the same for all the patches, and all the components of $\mathbf{p}$ are equal to $p$. Noting that the knot vectors may be different from patch to patch, we define in $\Omega^{(j)}$ the discrete space

$$V_h^{(j)} = \text{span}\{N_{\mathbf{i},\mathbf{p}}^{(j)}(\mathbf{x}) := \hat{N}_{\mathbf{i},\mathbf{p}}^{(j)} \circ \mathbf{F}^{(j)^{-1}}(\mathbf{x}),\ \mathbf{i} \in \mathbf{I}^{(j)}\}, \quad j = 1, \ldots, M_p.$$

For the definition of the discrete space in the whole domain $\Omega$, we take the functions that restricted to each patch belong to $V_h^{(j)}$, and possibly impose the continuity of the functions at the interfaces between patches, that is

$$V_h = \{u \in C^0(\Omega) : u|_{\Omega^{(j)}} \in V_h^{(j)} \text{ for } j = 1, \ldots, M_p\}. \tag{35}$$

In order to construct a basis for the discrete space $V_h$, we introduce a suitable conformity assumption, which follows the one given in [32].

**Assumption 3.7** *Let $\Gamma_{ij} = \partial\Omega^{(i)} \cap \partial\Omega^{(j)}$ be the interface between the patches $\Omega^{(i)}$ and $\Omega^{(j)}$, with $i \neq j$. We say that the two patches are* fully matching *if the two following conditions hold.*

*(i) $\Gamma_{ij}$ is either a vertex, or the image of a full edge, or the image of a full face for both parametric domains.*

*(ii) For each $N_{\mathbf{k},\mathbf{p}}^{(i)} \in V_h^{(i)}$ such that $\text{supp}(N_{\mathbf{k},\mathbf{p}}^{(i)}) \cap \Gamma_{ij} \neq \emptyset$, there exists a function $N_{\mathbf{l},\mathbf{p}}^{(j)} \in V_h^{(j)}$ such that $N_{\mathbf{k},\mathbf{p}}^{(i)}|_{\Gamma_{ij}} = N_{\mathbf{l},\mathbf{p}}^{(j)}|_{\Gamma_{ij}}$ (and vice versa).*

Assumption 3.7 means that the physical Bézier meshes $\mathcal{M}^{(i)}$ and $\mathcal{M}^{(j)}$ coincide on the interface $\Gamma_{ij}$, and the coincident knot vectors are affinely related, including knot repetitions. Thus the partition

$$\mathcal{M} = \cup_{j=1}^{M_p} \mathcal{M}^{(j)} \tag{36}$$

is a *conforming*, globally unstructured, locally (to each patch) structured mesh of the computational domain $\Omega$.

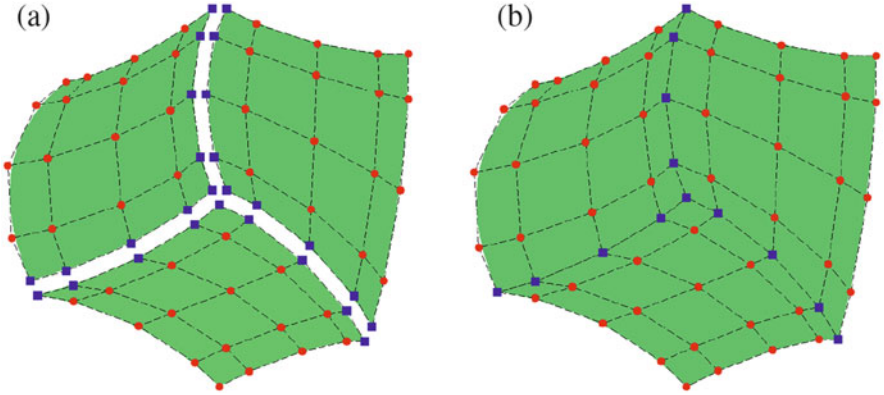


**Fig. 14** Generation of a multi-patch domain with conforming meshes. The *square* control points are associated to basis functions that match on the interface. (**a**) Control mesh of the three separate patches. (**b**) Control mesh of the multi-patch domain

Moreover, the control points and weights associated to the matching basis functions $N^{(i)}_{\mathbf{k},\mathbf{p}}$ and $N^{(j)}_{\mathbf{l},\mathbf{p}}$ must also coincide, i.e., $\mathbf{c}^{(i)}_{\mathbf{k}} = \mathbf{c}^{(j)}_{\mathbf{l}}$ and $w^{(i)}_{\mathbf{k}} = w^{(j)}_{\mathbf{l}}$, and as a consequence the control meshes must also match conformally (see Fig. 14).

Having conformity of the control meshes, the continuity condition is implemented very easily by generating a global numbering, in a process that resembles the generation of the connectivity array in finite element meshes. For each non-empty interface $\Gamma_{ij}$, we collect the pairs of coincident basis functions $N^{(i)}_{\mathbf{k},\mathbf{p}}$ and $N^{(j)}_{\mathbf{l},\mathbf{p}}$, and identify them as one single function, constraining their associated degrees of freedom to coincide. Note that for corners and edges (in the three-dimensional case), the new function may be generated from the contribution of functions coming from more than two patches.

Once the global numbering of the degrees of freedom has been generated, the implementation of isogeometric methods in multi-patch domains, in the case of conforming meshes, is almost identical to the one in a single-patch geometry. The numerical analysis, however, requires the definition of a different quasi-interpolant. We refer to [8, Sect. 4.4] and [17] for details.

*Remark 3.8* In this paper we only address the multi-patch case with conforming meshes. The study of multi-patch domains with non-conforming meshes was started in [20] and [32], allowing the case where one of the meshes on the interface is a refinement of the other one, and imposing some constraints to ensure $C^0$ continuity between patches. The general case of non-conforming meshes has been recently considered in [2, 44] and [38], imposing the continuity between patches in a weak form.

*Remark 3.9* We are also restricting ourselves to the case of $C^0$ continuity between patches. The construction of a basis and the properties of the spaces when considering higher continuity among patches are not well understood, and are

related to the questions of *extraordinary points*. Indeed, this is one of the most active research areas in isogeometric methods and in computer aided design. We refer the reader to [47, Chap. 6] and the references therein for further information.

*Remark 3.10* It should be noted that classical hexahedral finite elements can be seen as a special case of multi-patch spline spaces. Indeed, we recover finite elements by having each patch composed only by one element, which are sometimes called Bézier patches. In this case, the basis selected by our choice are mapped Bernstein polynomials, while control points and the corresponding control mesh provide a localization of degrees of freedom similar to the one associated with Lagrangian bases.

## 4 Approximation Properties of Isogeometric Spaces

For the numerical analysis of isogeometric methods, a fundamental issue is the study of the approximation properties of the mapped NURBS space $V_h$ defined in (28), and on which we seek the solution of our PDE. Although a large range of results already exist for splines, which are the starting point for the construction of isogeometric methods, the presence of the map $\mathbf{F}$ (representing the geometry) and the weight $W$ (NURBS are *rational*) adds a further degree of complexity.

As it is usual when deriving approximation estimates, we will make use of standard Sobolev spaces on a domain $D$, that can be either the parametric domain $\Omega$, the physical domain $\hat{\Omega}$, or any of their relevant subsets such as $Q$, $\tilde{Q}$, $K$ or $\tilde{K}$. We denote by $H^s(D)$, $s \in \mathbb{N}$ the space of square integrable functions $u \in L^2(D)$ such that its derivatives up to order $s$ are square integrable. As is well known (see e.g., [1]), the definition of Sobolev spaces extends to real regularity exponent $s$, but we will use this extension very rarely.

However, conventional Sobolev spaces are not suitable for studying the approximation estimates of the space $V_h$. Indeed, since the mapping $\mathbf{F}$ is not arbitrarily regular across mesh lines, even if a scalar function $u$ in physical space satisfies $u \in H^s(\Omega)$, its pull-back to the parametric domain $\hat{u} = u \circ \mathbf{F}$ is not guaranteed to be in $H^s(\hat{\Omega})$. As a consequence, the natural functional space in parameter space, in order to study the approximation properties of mapped NURBS, is not the standard Sobolev space $H^s(\hat{\Omega})$ but rather a "bent" version that allows for less regularity across mesh lines, and that we will define below.

The approximation estimates of this section are given with respect to the mesh size $h$, that is, we only consider $h$-refinement. Although the mathematical study of $h$-refinement was initiated in [6], we follow here the analysis in [9], that is more general and allows for a weaker assumption on the quasi-uniformity of the mesh. The procedure is similar to that already seen in Sect. 2.3: first we derive the approximation estimates in the univariate case in terms of the bent Sobolev spaces, and then we generalize them to the multivariate case in the parametric domain by

tensor product. Finally we obtain the approximation result in the physical domain by applying the parametrization $\mathbf{F}$.

In the following, $C$ will denote a constant, possibly different at each occurrence, which is independent of the knot vector's characteristic size $h$, but may depend on the polynomial degree $p$.

## 4.1 Univariate Approximation Estimates in Parametric Domain

As already noted, we start by approximation estimates for univariate B-spline spaces.

### 4.1.1 Bent Sobolev Spaces

Let $d = 1$, $\Xi$ be a knot vector, and $p$ be the polynomial degree. Recalling from Sect. 2.1.1 that $I = (0, 1)$, and that $I_i = (\zeta_i, \zeta_{i+1})$ are the intervals of the partition given by the knot vector, we define for any $q \in \mathbb{N}$ the piecewise polynomial space

$$\mathcal{P}_q(\Xi) = \{v \in L^2(I) \text{ such that } v|_{I_i} \text{ is a } q\text{-degree polynomial}, \forall i = 1, \dots, N-1\}.$$

We recall that, given $s \in \mathbb{N}$ and any sub-interval $E \subset I$, we indicate by $H^s(E)$ the usual Sobolev space endowed with norm $\|\cdot\|_{H^s(E)}$ and semi-norm $|\cdot|_{H^s(E)}$. We are now ready to introduce the bent Sobolev spaces, starting by the one-dimensional case. We define the *bent Sobolev space* (see [6]) on $I$ as

$$\mathcal{H}^s(I) = \left\{ \begin{array}{l} u \in L^2(I) \ \text{ such that } u|_{I_i} \in H^s(I_i) \ \forall\ i = 1, \dots, N-1, \text{ and} \\ D^k_- u(\zeta_i) = D^k_+ u(\zeta_i), \ \forall k = 0, \dots, \min\{s-1, k_i\}, \forall i = 2, \dots, N-1, \end{array} \right\} \tag{37}$$

where $D^k_\pm$ denote the $k$-order left and right derivative (or left and right limit for $k = 0$), and $k_i$ is the number of continuous derivatives at the breakpoint $\zeta_i$, as defined in Sect. 2.1. Note that, although we leave this implicit in the notation, the space $\mathcal{H}^s(I)$ depends on the knot vector $\Xi$. We endow the above space with the broken norm and semi-norms

$$\|u\|^2_{\mathcal{H}^s(I)} = \sum_{j=0}^{s} |u|^2_{\mathcal{H}^j(I)}, \quad |u|^2_{\mathcal{H}^j(I)} = \sum_{i=1}^{N-1} |u|^2_{H^j(I_i)} \quad \forall j = 0, 1, \dots, s,$$

where $|\cdot|_{H^0(I_i)} = \|\cdot\|_{L^2(I_i)}$. Moreover, we indicate with $\mathcal{H}^s(E)$, for any sub-interval $E \subset I$, the restriction of $\mathcal{H}^s(I)$ to $E$, with the obvious norm and semi-norms as above.

Given an integer $s$ such that $0 \leq s \leq p$, we define the space

$$\tilde{S}_{s,p}(\Xi) = \mathcal{P}_s(\Xi) \cap S_p(\Xi), \tag{38}$$

which is the space of piecewise polynomials of degree $s$ and the same regularity as the spline space $S_p(\Xi)$. Clearly, $S_p(\Xi) = \tilde{S}_{p,p}(\Xi)$, and for any $s < p$, the space $\tilde{S}_{s,p}(\Xi)$ is still a spline space but associated to a knot vector different from $\Xi$. For all $s \leq s' \leq p$ we also have the inclusion

$$\tilde{S}_{s,p}(\Xi) \subseteq \tilde{S}_{s',p}(\Xi). \tag{39}$$

### 4.1.2 Approximation Estimates in $h$

In the present section we prove one-dimensional approximation estimates for B-splines, that are more general than the classical ones in [46]. This more general estimates are needed in order to derive the approximation estimates for mapped NURBS spaces that follow. The following results were already proved in [9].

We need the following preliminary results in order to prove the approximation estimates for functions in bent Sobolev spaces (37), stated in Proposition 4.3.

**Lemma 4.1** *Let $s \in \mathbb{N}$, $s \leq p+1$. There exists a projector $\Gamma : \mathcal{H}^s(I) \to \tilde{S}_{s-1,p}(\Xi)$ such that for all $u \in \mathcal{H}^s(I)$,*

$$u - \Gamma(u) \in H^s(I). \tag{40}$$

*Proof* The proof can be found in [9, Lemma 3.1]. □

We recall that the length of the element $I_i = (\zeta_i, \zeta_{i+1})$ is denoted by $h_i$, and the global mesh size is represented by $h = \max\{h_i, 1 \leq i \leq N-1\}$. In the following we will also denote by $\tilde{h}_i$ the length of the support extensions $\tilde{I}_i$, defined in (7), for $i = 1, \ldots, N-1$.

Let $\Pi_{p,\Xi}$ be the quasi-interpolant on the space $S_p(\Xi)$ introduced in Sect. 2.3. We have the following approximation estimate.

**Proposition 4.2** *There exists a positive constant $C$, only dependent on $p$, such that for all $s \in \mathbb{N}$, $s \leq p+1$, and all $u \in H^s(I)$*

$$\|u - \Pi_{p,\Xi}(u)\|_{L^2(I_i)} \leq C(\tilde{h}_i)^s |u|_{H^s(\tilde{I}_i)} \quad \forall i = 1, \ldots, N-1. \tag{41}$$

*Furthermore let the knot vector $\Xi$ satisfy Assumption 2.3. Then, there exists a constant $C$ depending only on $p$ and $\theta$ such that for all $r, s \in \mathbb{N}$, $0 < r \leq s \leq p+1$, and all $u \in H^s(I)$*

$$|u - \Pi_{p,\Xi}(u)|_{H^r(I_i)} \leq C(\tilde{h}_i)^{s-r} |u|_{H^s(\tilde{I}_i)} \quad \forall i = 1, \ldots, N-1. \tag{42}$$

*Proof* Let any non empty knot span $I_i = (\zeta_i, \zeta_{i+1})$ and let $q$ be any polynomial of degree at most $p$ living on $[0, 1]$. Noting that, since $q \in S_p(\Xi)$, it holds $\Pi_{p,\Xi}(q) = q$ and using Proposition 2.4 it follows

$$\|u - \Pi_{p,\Xi}(u)\|_{L^2(I_i)} \leq \|u - q\|_{L^2(I_i)} + \|\Pi_{p,\Xi}(q - u)\|_{L^2(I_i)} \leq C\|u - q\|_{L^2(\tilde{I}_i)}.$$

The term above is bounded by standard polynomial approximation estimates in one dimension, leading immediately to (41). We note that, since the restriction of $S_p(\Xi)$ to any element $I_i$ is a polynomial of fixed degree $p$, we can apply inverse estimates. Therefore, regarding (42), we have

$$\begin{aligned} |u - \Pi_{p,\Xi}(u)|_{H^r(I_i)} &\leq |u - q|_{H^r(I_i)} + |\Pi_{p,\Xi}(q - u)|_{H^r(I_i)} \\ &\leq |u - q|_{H^r(I_i)} + Ch_i^{-r}\|\Pi_{p,\Xi}(q - u)\|_{L^2(I_i)} \\ &\leq |u - q|_{H^r(I_i)} + Ch_i^{-r}\|u - q\|_{L^2(\tilde{I}_i)}. \end{aligned}$$

Due to Assumption 2.3, it is easy to check that it holds $h_i \leq \tilde{h}_i \leq C'h_i$ for some fixed positive constant $C'$. Therefore, applying again standard polynomial approximation estimates, we get

$$|u - \Pi_{p,\Xi}(u)|_{H^r(I_i)} \leq C\Big(h_i^{s-r} + (\tilde{h}_i)^s h_i^{-r}\Big)|u|_{H^s(\tilde{I}_i)} \leq C(\tilde{h}_i)^{s-r}|u|_{H^s(\tilde{I}_i)},$$

which completes the proof. □

Lemma 4.1 and the Proposition above yield the following improved approximation result.

**Proposition 4.3** *There exists a positive constant $C = C(p)$ such that for all $s \in \mathbb{N}$, $s \leq p + 1$, and all $u \in \mathcal{H}^s(I)$*

$$\|u - \Pi_{p,\Xi}(u)\|_{L^2(I_i)} \leq C(\tilde{h}_i)^s |u|_{\mathcal{H}^s(\tilde{I}_i)} \quad \forall i = 1, \ldots, N - 1. \tag{43}$$

*Moreover, under Assumption 2.3, there exists a constant $C = C(p, \theta)$ such that for all $r, s \in \mathbb{N}$, $0 < r \leq s \leq p + 1$, and all $u \in \mathcal{H}^s(I)$*

$$|u - \Pi_{p,\Xi}(u)|_{H^r(I_i)} \leq C(\tilde{h}_i)^{s-r} |u|_{\mathcal{H}^s(\tilde{I}_i)} \quad \forall i = 1, \ldots, N - 1. \tag{44}$$

*Proof* Let $i$ be in $\{1, \ldots, N - 1\}$. We apply Lemma 4.1 and, since $\Gamma(u) \in \tilde{S}_{s-1,p}(\Xi) \subset S_p(\Xi)$, we have

$$u - \Pi_{p,\Xi}(u) = (u - \Gamma(u)) - \Pi_{p,\Xi}(u - \Gamma(u)). \tag{45}$$

Again noting that $\Gamma(u) \in \tilde{S}_{s-1,p}(\Xi)$, it follows that $|\Gamma(u)|_{H^s(I_j)} = 0$ for all $1 \leq j \leq N-1$. Moreover, since $(u - \Gamma(u)) \in H^s(I)$ due to (40), we can use Proposition 4.2 and get

$$\begin{aligned} |u - \Pi_{p,\Xi}(u)|_{H^r(I_i)} &= |u - \Gamma(u) - \Pi_{p,\Xi}(u - \Gamma(u))|_{H^r(I_i)} \leq C(\tilde{h}_i)^{s-r}|u - \Gamma(u)|_{H^s(\tilde{I}_i)} \\ &= C(\tilde{h}_i)^{s-r} \sum_{I_j \subset \tilde{I}_i} |u - \Gamma(u)|_{H^s(I_j)} = C(\tilde{h}_i)^{s-r} \sum_{I_j \subset \tilde{I}_i} |u|_{H^s(I_j)} \\ &= C(\tilde{h}_i)^{s-r}|u|_{\mathcal{H}^s(\tilde{I}_i)}. \end{aligned}$$

This gives (43)–(44), for $r = 0$ and $r > 0$ respectively. □

*Remark 4.4* Due to Assumption 2.3, we can replace $\tilde{h}_i$ by $h_i$ in (42) and in (44).

*Remark 4.5* Setting $0 \leq r = s \leq p + 1$, (43)–(44) guarantees also the stability of $\Pi_{p,\Xi}$ in Sobolev semi-norms:

$$|\Pi_{p,\Xi}(u)|_{H^r(I_i)} \leq C|u|_{\mathcal{H}^r(\tilde{I}_i)} \qquad \forall i = 1, \ldots, N-1, \quad \forall u \in \mathcal{H}^r(\tilde{I}_i).$$

*Remark 4.6* When $p \geq 1$, it is also possible to deal with spaces with boundary conditions, using a modified version of the operator $\Gamma$. The approximation result is analogous to the one in Proposition 4.3, and the proof follows the same arguments. We refer to [9] for details.

## 4.2 Multivariate Approximation Estimates in Parametric Domain

We now address the multi-dimensional tensor-product case for B-splines in the parametric domain. The multivariate analysis takes the steps from the one-dimensional estimates of the previous section, extended to the multi-dimensional case by a tensor product argument.

### 4.2.1 Tensor Product Bent Sobolev Spaces

In more dimensions, the tensor product bent Sobolev spaces are defined as follows. Let $\mathbf{s} = (s_1, s_2, \ldots, s_d)$ in $\mathbb{N}^d$. By a tensor product construction starting from (37), we define the tensor product bent Sobolev spaces in the parametric domain $\hat{\Omega} := (0,1)^d$

$$\mathcal{H}^{\mathbf{s}}(\hat{\Omega}) := \mathcal{H}^{s_1}(0,1) \otimes \mathcal{H}^{s_2}(0,1) \otimes \ldots \otimes \mathcal{H}^{s_d}(0,1)$$

iteratively as follows. For all $j = 2, \dots, d$

$$\begin{aligned}\mathcal{H}^{(s_1,\dots,s_j)}((0,1)^j) &:= \mathcal{H}^{(s_1,\dots,s_{j-1})}((0,1)^{j-1}) \otimes \mathcal{H}^{s_j}(0,1) \\ &\equiv \mathcal{H}^{(s_1,\dots,s_{j-1})}((0,1)^{j-1}; \mathcal{H}^{s_j}(0,1)).\end{aligned}$$

We endow the above spaces with the norm and seminorms

$$\|u\|^2_{\mathcal{H}^{\mathbf{s}}(\hat{\Omega})} = \sum_{r_1=0}^{s_1} \cdots \sum_{r_d=0}^{s_d} |u|^2_{\mathcal{H}^{(r_1,\dots,r_d)}(\hat{\Omega})}, \qquad |u|^2_{\mathcal{H}^{(r_1,\dots,r_d)}(\hat{\Omega})} = \sum_{Q \in \widehat{\mathcal{M}}} |u|^2_{\mathcal{H}^{(r_1,\dots,r_d)}(Q)}, \tag{46}$$

where for the elements $Q \in \widehat{\mathcal{M}}$ we have used the local semi-norm

$$|u|_{\mathcal{H}^{(r_1,\dots,r_d)}(Q)} = \left\| \frac{\partial^{r_1} \dots \partial^{r_d} u}{\partial \zeta_1^{r_1} \dots \partial \zeta_d^{r_d}} \right\|_{L^2(Q)}.$$

The above definition clearly extends immediately to the case of any hyper-rectangle $E \subset \hat{\Omega}$ that is a union of elements in $\widehat{\mathcal{M}}$.

### 4.2.2 Approximation Estimates in *h*

We restrict, for simplicity of exposition, the detailed analysis to the bi-dimensional case and present the general dimensional case without proof at the end of the section. As in the one-dimensional case, we introduce the following local quasi-uniformity assumption.

**Assumption 4.7** *Assumption 2.3 holds for each univariate partition* $Z_j = \{\zeta_{j,1}, \dots, \zeta_{j,N_j}\}$.

Let $\Pi_{p_i,\Xi_i} : L^2(I) \to S_{p_i}(\Xi_i)$, for $i = 1, 2$, indicate the univariate quasi-interpolant associated to the knot vector $\Xi_i$ and polynomial degree $p_i$ introduced in Sect. 2.3. Let moreover $\Pi_{\mathbf{p},\boldsymbol{\Xi}} = \Pi_{p_1,\Xi_1} \otimes \Pi_{p_2,\Xi_2}$ from $L^2(\hat{\Omega})$ to $S_{\mathbf{p}}(\boldsymbol{\Xi})$ denote the tensor product quasi-interpolant built using the $\Pi_{p_i,\Xi_i}$ above according to the construction in (25) for $d = 2$.

In the sequel, given any sufficiently regular function $u : \hat{\Omega} \to \mathbb{R}$, we will indicate the partial derivative operators with the symbol

$$\hat{D}^{\mathbf{r}} u = \frac{\partial^{r_1} \partial^{r_2} u}{\partial \zeta_1^{r_1} \partial \zeta_2^{r_2}} \qquad \mathbf{r} = (r_1, r_2) \in \mathbb{N}^2. \tag{47}$$

Let $E \subset \hat{\Omega}$ be any union of elements $Q \in \widehat{\mathcal{M}}$ of the spline mesh. Then, in the following we will adopt the notation

$$\|u\|^2_{L^2_h(E)} := \sum_{\substack{Q \in \widehat{\mathcal{M}} \\ Q \subset E}} \|u\|^2_{L^2(Q)},$$

which will be useful for distributions $u$ which are not in $L^2$ of the whole $E$.

Finally, the element size of a generic element $Q_{\mathbf{i}} = I_{1,i_1} \times \ldots \times I_{d,i_d} \in \widehat{\mathcal{M}}$ (see Sect. 2.2.1) will be denoted by $h_{Q_{\mathbf{i}}} = \operatorname{diam}(Q_{\mathbf{i}})$. Moreover, we will indicate the length of the edges of $Q_{\mathbf{i}}$ by $h_{1,i_1}, h_{2,i_2}, \ldots, h_{d,i_d}$, and the length of the edges of its extended patch $\tilde{Q}_{\mathbf{i}}$ by $\tilde{h}_{1,i_1}, \tilde{h}_{2,i_2}, \ldots, \tilde{h}_{d,i_d}$.

We are now able to show the following result.

**Proposition 4.8** *Let Assumption 4.7 hold. Let the integers $0 \le r_1 \le s_1 \le p_1 + 1$ and $0 \le r_2 \le s_2 \le p_2 + 1$. Then, there exists a constant $C$ depending only on $\mathbf{p}, \theta$ such that for all elements $Q_{\mathbf{i}} \in \widehat{\mathcal{M}}$, it holds*

$$\|\hat{D}^{(r_1,r_2)}(u - \Pi_{\mathbf{p},\boldsymbol{\Xi}} u)\|_{L^2(Q_{\mathbf{i}})} \le C\Big((\tilde{h}_{1,i_1})^{s_1-r_1} \|\hat{D}^{(s_1,r_2)} u\|_{L^2_h(\tilde{Q}_{\mathbf{i}})} + (\tilde{h}_{2,i_2})^{s_2-r_2} \|\hat{D}^{(r_1,s_2)} u\|_{L^2_h(\tilde{Q}_{\mathbf{i}})}\Big)$$

*for all $u$ in $\mathcal{H}^{(s_1,r_2)}(\hat{\Omega}) \cap \mathcal{H}^{(r_1,s_2)}(\hat{\Omega})$. When $(r_1, r_2) = (0, 0)$, Assumption 4.7 is not needed and $C$ depends only on $\mathbf{p}$.*

*Proof* Let $u$ be as above and $Q_{\mathbf{i}} = I_{1,i_1} \times I_{2,i_2}$ be any element of the mesh. From the definition of $\Pi_{\mathbf{p},\boldsymbol{\Xi}}$ as in (25), and the triangle inequality it immediately follows that

$$\|\hat{D}^{(r_1,r_2)}(u - \Pi_{\mathbf{p},\boldsymbol{\Xi}} u)\|_{L^2(Q_{\mathbf{i}})} \le T_1 + T_2, \tag{48}$$

with the terms

$$T_1 = \|\hat{D}^{(r_1,r_2)}(u - \Pi_{p_1,\Xi_1} u)\|_{L^2(Q_{\mathbf{i}})}, \quad T_2 = \|\hat{D}^{(r_1,r_2)}(\Pi_{p_1,\Xi_1} u - \Pi_{\mathbf{p},\boldsymbol{\Xi}} u)\|_{L^2(Q_{\mathbf{i}})}.$$

It is easy to check that the derivative with respect to $\zeta_2$ and the operator $\Pi_{p_1,\Xi_1}$ commute. Then we have $\hat{D}^{(r_1,r_2)}\Pi_{p_1,\Xi_1} = \hat{D}^{(r_1,0)}\Pi_{p_1,\Xi_1}\hat{D}^{(0,r_2)}$. Indicating with $w = \hat{D}^{(0,r_2)}u$, then using Proposition 4.3, we get

$$\begin{aligned}(T_1)^2 &= \|\hat{D}^{(r_1,0)}(w - \Pi_{p_1,\Xi_1} w)\|^2_{L^2(Q_{\mathbf{i}})} = \int_{I_{2,i_2}} \int_{I_{1,i_1}} \Big(\hat{D}^{(r_1,0)}(w - \Pi_{p_1,\Xi_1} w)\Big)^2 \mathrm{d}\zeta_1 \mathrm{d}\zeta_2 \\ &= \int_{I_{2,i_2}} |w - \Pi_{p_1,\Xi_1} w|^2_{H^{r_1}(I_{1,i_1})} \mathrm{d}\zeta_2 \le C(\tilde{h}_{1,i_1})^{2(s_1-r_1)} \int_{I_{2,i_2}} |w|^2_{\mathcal{H}^{s_1}(\tilde{I}_{1,i_1})} \mathrm{d}\zeta_2 \\ &\le C(\tilde{h}_{1,i_1})^{2(s_1-r_1)} \|\hat{D}^{(s_1,r_2)} u\|^2_{L^2_h(\tilde{Q}_{\mathbf{i}})}.\end{aligned} \tag{49}$$

Let now $v = \hat{D}^{(r_1,0)}u$. Using again the property $\hat{D}^{(r_1,r_2)}\Pi_{p_1,\Xi_1} = \hat{D}^{(r_1,0)}\Pi_{p_1,\Xi_1}\hat{D}^{(0,r_2)}$, the similar property $\hat{D}^{(r_1,r_2)}\Pi_{p_2,\Xi_2} = \hat{D}^{(0,r_2)}\Pi_{p_2,\Xi_2}\hat{D}^{(r_1,0)}$ and recalling Remark 4.5, yields

$$\begin{aligned}
(T_2)^2 &= \|\hat{D}^{(r_1,0)}\Pi_{p_1,\Xi_1}\hat{D}^{(0,r_2)}(u - \Pi_{p_2,\Xi_2}u)\|^2_{L^2(Q_{\mathbf{i}})} \\
&= \int_{I_{2,i_2}} |\Pi_{p_1,\Xi_1}\hat{D}^{(0,r_2)}(u - \Pi_{p_2,\Xi_2}u)|^2_{H^{r_1}(I_{1,i_1})}\mathrm{d}\zeta_2 \\
&\leq C\int_{I_{2,i_2}} |\hat{D}^{(0,r_2)}(u - \Pi_{p_2,\Xi_2}u)|^2_{\mathcal{H}^{r_1}(\tilde{I}_{1,i_1})}\mathrm{d}\zeta_2 \qquad (50) \\
&= C\|\hat{D}^{(r_1,r_2)}(u - \Pi_{p_2,\Xi_2}u)\|^2_{L^2_h(\tilde{I}_{1,i_1}\times I_{2,i_2})} \\
&= C\|\hat{D}^{(0,r_2)}(v - \Pi_{p_2,\Xi_2}v)\|^2_{L^2_h(\tilde{I}_{1,i_1}\times I_{2,i_2})}.
\end{aligned}$$

The last term in (50) is bounded with the same steps used in (49), simply exchanging the role of $\zeta_1$ and $\zeta_2$. One gets

$$(T_2)^2 \leq C(\tilde{h}_{2,i_2})^{2(s_2-r_2)}\|\hat{D}^{(r_1,s_2)}u\|^2_{L^2_h(\tilde{Q}_{\mathbf{i}})}. \qquad (51)$$

The result follows combining (48) with (49) and (51). □

The three, or more, dimensional case follows obviously with the same arguments used above. We have the following result in general dimension $d$, where now $\Pi_{\mathbf{p},\boldsymbol{\Xi}}$ is the quasi-interpolant built following the tensor product construction (25) and using the one-dimensional operators $\Pi_{p_i,\Xi_i}$, $i = 1, 2, \ldots, d$.

**Proposition 4.9** *Let Assumption 4.7 hold. Let the integers $0 \leq r_\ell \leq s_\ell \leq p_\ell + 1$ for all $\ell = 1, \ldots, d$. Then, there exists a constant $C$ depending only on $\mathbf{p}, \theta$ such that for all elements $Q_{\mathbf{i}} \in \widehat{\mathcal{M}}$, it holds*

$$\|\hat{D}^{(r_1,\ldots,r_d)}(u - \Pi_{\mathbf{p},\boldsymbol{\Xi}}u)\|_{L^2(Q_{\mathbf{i}})} \leq C\Big(\sum_{\ell=1,\ldots,d}(\tilde{h}_{\ell,i_\ell})^{s_\ell - r_\ell}\|\hat{D}^{(r_1,\ldots,r_{\ell-1},s_\ell,r_{\ell+1},\ldots,r_d)}u\|_{L^2_h(\tilde{Q}_{\mathbf{i}})}\Big)$$

*for all $u$ in $\mathcal{H}^{(s_1,r_2,\ldots,r_d)}(\hat{\Omega}) \cap \mathcal{H}^{(r_1,s_2,r_3,\ldots,r_d)}(\hat{\Omega}) \cap \ldots \cap \mathcal{H}^{(r_1,\ldots,r_{d-1},s_d)}(\hat{\Omega})$. When $r_\ell = 0$ for all $\ell = 1, \ldots, d$, Assumption 4.7 is not needed and $C$ depends only on $\mathbf{p}$.*

*Remark 4.10* Due to Remark 4.6, it is immediate to extend the results of this section to spaces with boundary conditions on the whole $\partial\hat{\Omega}$ or on some faces of $\hat{\Omega}$.

## 4.3 NURBS Approximation Estimates in the Physical Domain

We here finally present error estimates for the mapped NURBS spaces introduced in (28), by taking the steps from the B-spline approximation results in the parametric

domain of the previous section. We start by defining the projector for the discrete space $V_h$ of mapped NURBS, and then we derive the approximation estimates with respect to the mesh size $h$. In the whole present section we are requiring that the mapping $\mathbf{F}$ satisfies Assumption 3.1.

### 4.3.1 Definition of the Projector in the Physical Domain

Before defining the projector we recall the definition of the discrete space $V_h$ in (28): assuming that the parametrization $\mathbf{F}$ is constructed from the NURBS space $\hat{V}_h = N_{\mathbf{p}}(\boldsymbol{\Xi}, W)$, we define

$$V_h = \{u \circ \mathbf{F}^{-1} : u \in \hat{V}_h\}.$$

That is, the space $V_h$ is defined mapping through $\mathbf{F}$ the NURBS space $N_{\mathbf{p}}(\boldsymbol{\Xi}, W)$, which is constructed from $S_{\mathbf{p}}(\boldsymbol{\Xi})$ using the weight $W$, as we have seen in Sect. 2.

From the definition of the space, the most natural definition of the projector is the one introduced in [6]. Let $\Pi_{\mathbf{p},\boldsymbol{\Xi}}$ be the spline projector (25), defined in the parametric domain by tensor product of the univariate projectors. In the physical domain we define the projector $\Pi_{V_h} : L^2(\Omega) \to V_h$ as

$$\Pi_{V_h} u := \frac{\Pi_{\mathbf{p},\boldsymbol{\Xi}}(W(u \circ \mathbf{F}))}{W} \circ \mathbf{F}^{-1}. \tag{52}$$

The following result proves that $\Pi_{V_h}$ is actually a projector on $V_h$.

**Proposition 4.11** *It holds that $\Pi_{V_h} v_h = v_h$ for all $v_h \in V_h$. That is, $\Pi_{V_h}$ is a projector.*

*Proof* Let $v_h \in V_h$, which can be written as $v_h = \hat{v}_h \circ \mathbf{F}^{-1}$ for a unique $\hat{v}_h \in \hat{V}_h = N_{\mathbf{p}}(\boldsymbol{\Xi}, W)$. Moreover, $\hat{v}_h = \dfrac{\hat{u}_h}{W}$ for a unique $\hat{u}_h \in S_{\mathbf{p}}(\boldsymbol{\Xi})$. Using definition (52) and these two previous expressions, and recalling that $\Pi_{\mathbf{p},\boldsymbol{\Xi}}$ is a projector onto the space $S_{\mathbf{p}}(\boldsymbol{\Xi})$, the result is proved. □

### 4.3.2 Approximation Estimates in $h$

Before proving the approximation estimates for the projector, we need to introduce some notation and intermediate results. As before, for simplicity of exposition we will show the details and proofs for the bivariate case and present the multi-dimensional case briefly at the end of the section (the proofs for $d > 2$ being a simple extension of the $d = 2$ argument).

In the physical domain $\Omega = \mathbf{F}(\hat{\Omega})$, besides from the Cartesian coordinates $x_1$ and $x_2$, we also introduce the coordinate system naturally induced by the geometrical map $\mathbf{F}$, referred as $\mathbf{F}$-coordinate system, that associates to a point $\mathbf{x} \in \Omega$ the

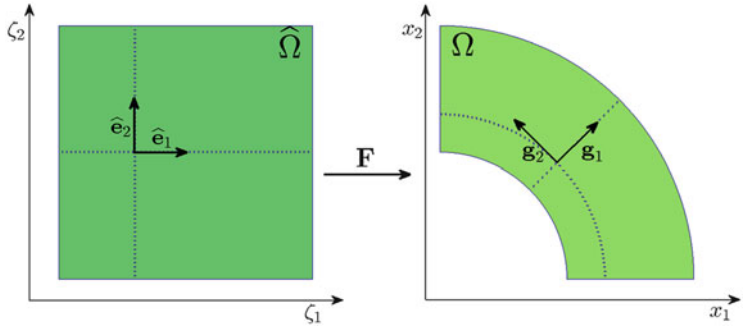


**Fig. 15** Definition of the **F**-coordinate system in the physical domain

Cartesian coordinates in $\hat{\Omega}$ of its counter-image $\mathbf{F}^{-1}(\mathbf{x})$. At each $\mathbf{x} \in K \in \mathcal{M}_0$ (more generally, at each $\mathbf{x}$ where $\mathbf{F}$ is differentiable) the tangent *base vectors* $\mathbf{g}_1$ and $\mathbf{g}_2$ of the **F**-coordinate system can be defined as

$$\mathbf{g}_i = \mathbf{g}_i(\mathbf{x}) = \frac{\partial \mathbf{F}}{\partial \zeta_i}(\mathbf{F}^{-1}(\mathbf{x})), \quad i = 1, 2; \tag{53}$$

these are the image of the canonical base vectors $\hat{\mathbf{e}}_i$ in $\hat{\Omega}$, and represent the axis directions of the **F**-coordinate system (see Fig. 15).

Analogously to the derivatives in the parametric domain (47), the derivatives of $u : \Omega \to \mathbb{R}$ in Cartesian coordinates are denoted by

$$D^{\mathbf{r}} u = \frac{\partial^{r_1} \partial^{r_2} u}{\partial x_1^{r_1} \partial x_2^{r_2}} \qquad \mathbf{r} = (r_1, r_2) \in \mathbb{N}^2.$$

We also consider the derivatives of $u : \Omega \to \mathbb{R}$ with respect to the **F**-coordinates. These are just the directional derivatives: for the first order we have

$$\frac{\partial u}{\partial \mathbf{g}_i}(\mathbf{x}) = \nabla u(\mathbf{x}) \cdot \mathbf{g}_i(\mathbf{x}) = \lim_{t \to 0} \frac{u(\mathbf{x} + t\mathbf{g}_i(\mathbf{x})) - u(\mathbf{x})}{t}, \tag{54}$$

which is well defined for any $\mathbf{x}$ in the (open) elements of the coarse triangulation $\mathcal{M}_0$, as already noted. Higher order derivatives are defined in the similar way

$$\frac{\partial^{r_i} u}{\partial \mathbf{g}_i^{r_i}} = \frac{\partial}{\partial \mathbf{g}_i}\left(\frac{\partial^{r_i - 1} u}{\partial \mathbf{g}_i^{r_i - 1}}\right) = \left(\frac{\partial}{\partial \mathbf{g}_i}\left(\cdots\left(\frac{\partial}{\partial \mathbf{g}_i}\left(\frac{\partial u}{\partial \mathbf{g}_i}\right)\right)\right)\right);$$

more generally, we adopt the notation

$$D_{\mathbf{F}}^{\mathbf{r}} u = \frac{\partial^{r_1}}{\partial \mathbf{g}_1^{r_1}} \frac{\partial^{r_2} u}{\partial \mathbf{g}_2^{r_2}} \qquad \mathbf{r} = (r_1, r_2) \in \mathbb{N}^2. \tag{55}$$

Derivatives with respect to the **F**-coordinates are directly related to derivatives in the parametric domain, as stated in the following proposition.

**Proposition 4.12** *Let $u : \Omega \to \mathbb{R}$. For all $K \in \mathcal{M}$, $\mathbf{r} \in \mathbb{N}^2$, we have*

$$D^{\mathbf{r}}_{\mathbf{F}} u = \left(\widehat{D}^{\mathbf{r}} (u \circ \mathbf{F})\right) \circ \mathbf{F}^{-1}. \tag{56}$$

*Proof* The proof can be obtained by a simple application of the chain rule, plus an induction argument. We refer to [9, Prop. 5.1] for the details. □

Let $E$ be a union of elements $K \in \mathcal{M}$, we introduce the norms and seminorms

$$\|u\|^2_{\mathcal{H}^{(s_1,s_2)}_{\mathbf{F}}(E)} = \sum_{r_1=0}^{s_1} \sum_{r_2=0}^{s_2} |u|^2_{\mathcal{H}^{(r_1,r_2)}_{\mathbf{F}}(E)}, \qquad |u|^2_{\mathcal{H}^{(s_1,s_2)}_{\mathbf{F}}(E)} = \sum_{\substack{K \in \mathcal{M} \\ K \subset E}} |u|^2_{H^{(s_1,s_2)}_{\mathbf{F}}(K)}, \tag{57}$$

where

$$|u|_{H^{(s_1,s_2)}_{\mathbf{F}}(K)} = \left\| D^{(s_1,s_2)}_{\mathbf{F}} u \right\|_{L^2(K)}.$$

We also introduce the following space

$$H^{(s_1,s_2)}_{\mathbf{F}}(\Omega) = \text{Closure of } C^\infty(\Omega) \text{ with respect to the norm } \|\cdot\|_{\mathcal{H}^{(s_1,s_2)}_{\mathbf{F}}(\Omega)}, \tag{58}$$

endowed with the norm $\|\cdot\|_{H^{(s_1,s_2)}_{\mathbf{F}}(\Omega)} = \|\cdot\|_{\mathcal{H}^{(s_1,s_2)}_{\mathbf{F}}(\Omega)}$. Note that the space $H^{(s_1,s_2)}_{\mathbf{F}}(\Omega)$ is a kind of tensor-product Sobolev space with respect to the physical coordinates. For instance, it holds

$$H^{s_1+s_2}(\Omega) \subseteq H^{(s_1,s_2)}_{\mathbf{F}}(\Omega) \subseteq H^{\min(s_1,s_2)}(\Omega),$$

where as usual $H^t(\Omega)$ represents, for $t \in \mathbb{N}$, the standard Sobolev space of order $t$ on $\Omega$.

We then have the following proposition, which is a consequence of Assumption 3.1 and the inter-element continuity of the parametrization **F**. The simple proof can be found in [9].

**Proposition 4.13** *Let $u : \Omega \to \mathbb{R}$ and $\mathbf{s} \in \mathbb{N}^2$. If $u \in H^{(s_1,s_2)}_{\mathbf{F}}(\Omega)$ then its pull-back $\hat{u} = u \circ \mathbf{F}$ is in the bent Sobolev space $\mathcal{H}^{(s_1,s_2)}(\hat{\Omega})$. Moreover there exists a positive constant $C = C(\mathbf{F})$ such that for all $K = \mathbf{F}(Q) \in \mathcal{M}$, $\mathbf{s} \in \mathbb{N}^2$, it holds*

$$C^{-1} \|u\|_{H^{(s_1,s_2)}_{\mathbf{F}}(K)} \leq \|\hat{u}\|_{H^{(s_1,s_2)}(Q)} \leq C \|u\|_{H^{(s_1,s_2)}_{\mathbf{F}}(K)}. \tag{59}$$

The following theorem from [9] states the main estimate for the approximation error of $\Pi_{V_h} u$ and, making use of derivatives in the **F**-coordinate system, it is suitable

for anisotropic meshes. We recall that, for a generic element $K_{\mathbf{i}} = \mathbf{F}(Q_{\mathbf{i}}) \in \mathcal{M}$, the notation $\tilde{K}_{\mathbf{i}} = \mathbf{F}(\tilde{Q}_{\mathbf{i}})$ indicates its support extension. Moreover, $\tilde{h}_{1,i_1}$ and $\tilde{h}_{2,i_2}$ indicate the edge lengths of $\tilde{Q}_{\mathbf{i}}$, as in Proposition 4.8.

**Theorem 4.14** *Let Assumption 4.7 hold. Let the integers $r_i, s_i$ be such that $0 \le r_i \le s_i \le p_i + 1$, $i = 1, 2$. Then, there exists a constant $C$ depending only on $\mathbf{p}, \theta, \mathbf{F}, W$ such that for all elements $K_{\mathbf{i}} = \mathbf{F}(Q_{\mathbf{i}}) \in \mathcal{M}$, it holds*

$$|u - \Pi_{V_h} u|_{\mathcal{H}_{\mathbf{F}}^{(r_1,r_2)}(K_{\mathbf{i}})} \le C\Big( (\tilde{h}_{1,i_1})^{s_1 - r_1} \|u\|_{\mathcal{H}_{\mathbf{F}}^{(s_1,r_2)}(\tilde{K}_{\mathbf{i}})} + (\tilde{h}_{2,i_2})^{s_2 - r_2} \|u\|_{\mathcal{H}_{\mathbf{F}}^{(r_1,s_2)}(\tilde{K}_{\mathbf{i}})} \Big) \tag{60}$$

*for all $u$ in $H_{\mathbf{F}}^{(s_1,r_2)}(\Omega) \cap H_{\mathbf{F}}^{(r_1,s_2)}(\Omega)$. When $(r_1, r_2) = (0, 0)$, Assumption 4.7 is not needed.*

*Proof* The argument is similar to that in [6]. We first use (56) and perform a change of variable

$$\begin{aligned} \left\| D_{\mathbf{F}}^{(r_1,r_2)} (u - \Pi_{V_h} u) \right\|_{L^2(K_{\mathbf{i}})} &\le C \left\| \widehat{D}^{(r_1,r_2)} \left( u \circ \mathbf{F} - (\Pi_{V_h} u) \circ \mathbf{F} \right) \right\|_{L^2(Q_{\mathbf{i}})} \\ &= C \left\| \widehat{D}^{(r_1,r_2)} \left( \frac{W\,(u \circ \mathbf{F}) - \Pi_{\mathbf{p},\boldsymbol{\Xi}} \left( W (u \circ \mathbf{F}) \right)}{W} \right) \right\|_{L^2(Q_{\mathbf{i}})} \end{aligned} \tag{61}$$

where $K_{\mathbf{i}} = \mathbf{F}(Q_{\mathbf{i}})$, and the constant $C$ takes into account the factor $\det(\nabla \mathbf{F})$ in the change of variable. We now note that, due to the regularity of $u$ and using Proposition 4.13, it follows $W\,(u \circ \mathbf{F}) \in \mathcal{H}^{(s_1,r_2)}(\hat{\Omega}) \cap \mathcal{H}^{(r_1,s_2)}(\hat{\Omega})$. Then we use the Leibniz formula

$$\left\| \widehat{D}^{(r_1,r_2)} \left( \frac{\phi}{W} \right) \right\|_{L^2(Q_{\mathbf{i}})}^2 \le C \sum_{q_i = 0,\dots,r_i} \left\| \widehat{D}^{(q_1,q_2)} \phi \right\|_{L^2(Q_{\mathbf{i}})}^2,$$

where $C$ depends on the derivatives (inside the elements) of $W^{-1}$, and then we use Proposition 4.8, yielding

$$\begin{aligned} &\left\| \widehat{D}^{(r_1,r_2)} \left( \frac{W\,(u \circ \mathbf{F}) - \Pi_{\mathbf{p},\boldsymbol{\Xi}} \left( W\,(u \circ \mathbf{F}) \right)}{W} \right) \right\|_{L^2(Q_{\mathbf{i}})}^2 \\ &\le C \sum_{q_i = 0,\dots,r_i} \left\| \widehat{D}^{(q_1,q_2)} \left( W\,(u \circ \mathbf{F}) - \Pi_{\mathbf{p},\boldsymbol{\Xi}} \left( W\,(u \circ \mathbf{F}) \right) \right) \right\|_{L^2(Q_{\mathbf{i}})}^2 \\ &\le C (\tilde{h}_{1,i_1})^{2(s_1 - r_1)} \sum_{q_2 = 0,\dots,r_2} \left\| \widehat{D}^{(s_1,q_2)} \left( W\,(u \circ \mathbf{F}) \right) \right\|_{L_h^2(\tilde{Q}_{\mathbf{i}})}^2 \\ &\quad + C (\tilde{h}_{2,i_2})^{2(s_2 - r_2)} \sum_{q_1 = 0,\dots,r_1} \left\| \widehat{D}^{(q_1,s_2)} \left( W\,(u \circ \mathbf{F}) \right) \right\|_{L_h^2(\tilde{Q}_{\mathbf{i}})}^2. \end{aligned} \tag{62}$$

The last step is to use the Leibniz formula again (now with $C$ depending on the derivatives of $W$), identity (56) and the change of variable again. We obtain

$$\sum_{q_2=0,\dots,r_2} \left\| \widehat{D}^{(s_1,q_2)} \left(W \left(u \circ \mathbf{F}\right)\right)\right\|^2_{L^2_h(\tilde{Q}_\mathbf{i})} \leq C \sum_{q_1=0,\dots,s_1} \sum_{q_2=0,\dots,r_2} \left\| \widehat{D}^{(q_1,q_2)} \left(u \circ \mathbf{F}\right)\right\|^2_{L^2_h(\tilde{Q}_\mathbf{i})}$$
$$\leq C \sum_{q_1=0,\dots,s_1} \sum_{q_2=0,\dots,r_2} \left\| D_\mathbf{F}^{(q_1,q_2)} u \right\|^2_{L^2_h(\tilde{K}_\mathbf{i})}, \tag{63}$$

and

$$\sum_{q_1=0,\dots,r_1} \left\| \widehat{D}^{(q_1,s_2)} \left(W \left(u \circ \mathbf{F}\right)\right)\right\|^2_{L^2_h(\tilde{Q}_\mathbf{i})} \leq C \sum_{q_1=0,\dots,r_1} \sum_{q_2=0,\dots,s_2} \left\| \widehat{D}^{(q_1,q_2)} \left(u \circ \mathbf{F}\right)\right\|^2_{L^2_h(\tilde{Q}_\mathbf{i})}$$
$$\leq C \sum_{q_1=0,\dots,r_1} \sum_{q_2=0,\dots,s_2} \left\| D_\mathbf{F}^{(q_1,q_2)} u \right\|^2_{L^2_h(\tilde{K}_\mathbf{i})}. \tag{64}$$

Collecting (61), (62), (63), and (64) finally gives (60). □

Under Assumption 4.7, it clearly holds $\tilde{h}_{\ell,i_\ell} \simeq h_{\ell,i_\ell}$, $\ell = 1, 2$. Therefore, the parametric mesh sizes $\tilde{h}_{\ell,i_\ell}$ appearing in (60) can be also substituted by $h_{\ell,i_\ell}$, $\ell = 1, 2$. Note that, since the mapping $\mathbf{F}$ is fixed at the coarse level of discretization (and thus it is element-wise uniformly regular), and thanks to Assumption 3.1, the lengths $h_{\ell,i_\ell}$ are equivalent to the lengths of the corresponding edges of the physical element $K_\mathbf{i} = \mathbf{F}(Q_\mathbf{i})$, as shown in Fig. 16.

We have the following corollary of Theorem 4.14, which bounds the local approximation error in standard $H^r$ Sobolev norms. We show this result directly in the general case of dimension $d$.

**Corollary 4.15** *Let Assumption 4.7 hold. Let the integers $r, s_\ell$ be such that $0 \leq r \leq s_\ell \leq p_\ell + 1$, $\ell = 1, \dots, d$. Then, there exists a constant $C$ depending only on $\mathbf{p}, \theta, \mathbf{F}, W$ such that for all elements $K_\mathbf{i} = \mathbf{F}(Q_\mathbf{i}) \in \mathcal{M}$, it holds*

$$\|u - \Pi_{V_h} u\|_{H^r(K_\mathbf{i})} \leq C \Big( \sum_{\ell=1}^{d} (\tilde{h}_{\ell,i_\ell})^{s_\ell - r} \sum_{j=0,\dots,s_\ell - r} \|D_\mathbf{F}^{j\mathbf{e}_\ell} u\|_{H^r(\tilde{K}_\mathbf{i})} \Big), \tag{65}$$

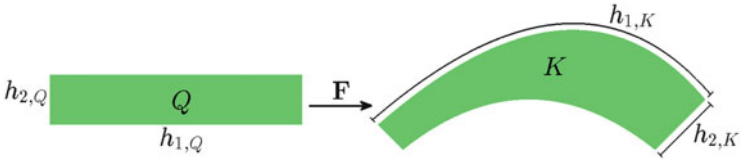


**Fig. 16** $Q$ is mapped by the geometrical map $\mathbf{F}$ to $K$

*where* $\mathbf{e}_\ell$ *are the vectors of the canonical basis of* $\mathbb{R}^d$*, and the result holds for all* $u \in H^r(\Omega)$ *such that* $D_{\mathbf{F}}^{(s_\ell - r)\mathbf{e}_\ell} u$ *are in* $H^r(\Omega)$ *for* $\ell = 1, 2, \ldots, d$.

*Proof* It is enough to remark that

$$\|u - \Pi_{V_h} u\|_{H^r(K_\mathbf{i})} \leq C \sum_{\mathbf{r}:\ 0 \leq r_1 + \ldots + r_d \leq r} \|u - \Pi_{V_h} u\|_{\mathcal{H}^{\mathbf{r}}_{\mathbf{F}}(K_\mathbf{i})}$$

and the proof follows by applying Theorem 4.14. □

Finally, it is easy to see that, for $r$ and $s$ as in the corollary below, we have

$$\sum_{j=0,\ldots,s-r} \|D_{\mathbf{F}}^{j\mathbf{e}_\ell} u\|_{H^r(\tilde{K}_\mathbf{i})} \leq \|u\|_{H^s(\tilde{K}_\mathbf{i})},$$

thus the following is true:

**Corollary 4.16** *Let Assumption 4.7 hold. Let the integers* $r, s$ *be such that* $0 \leq r \leq s \leq \min(p_1, \ldots, p_d) + 1$*. Then, there exists a constant* $C$ *depending only on* $\mathbf{p}, \theta, \mathbf{F}, W$ *such that*

$$\begin{aligned} &\|u - \Pi_{V_h} u\|_{H^r(K_\mathbf{i})} \leq C(h_{\tilde{K}_\mathbf{i}})^{s-r} \|u\|_{H^s(\tilde{K}_\mathbf{i})} \quad \forall K_\mathbf{i} \in \mathcal{M}, \\ &\|u - \Pi_{V_h} u\|_{H^r(\Omega)} \leq C h^{s-r} \|u\|_{H^s(\Omega)}, \end{aligned} \tag{66}$$

*for all* $u$ *in* $H^s(\Omega)$.

Note that the result here above is very similar to that in [6, Thm. 3.1], but without the uniformity assumption on the mesh.

*Remark 4.17* As already commented in the previous sections, the cases with boundary conditions are handled similarly and thus are not detailed here. We can therefore set homogeneous boundary conditions on $\Gamma_D \subset \partial\Omega$ that is the image through $\mathbf{F}$ of a collection of faces of $\hat{\Omega}$, as in Assumption 3.2, and define a projector $\Pi_{V_{h,\Gamma_D}}$ into the space with boundary conditions (30), with the same approximation properties of $\Pi_{V_h}$.

*Remark 4.18* It is worth to note that deriving approximation estimates which explicitly take into account the degree $p$ and the continuity $k$ is one of the most challenging open problems concerning the numerical analysis of isogeometric methods. Preliminary results in the case of low continuity, roughly $k \leq p/2$, have been given in [7].

# 5 Construction and Analysis of Isogeometric Spaces for Vector Fields

This section is devoted to the definition and the analysis of isogeometric methods for the approximation of vector fields, for which we will mainly follow the two papers [15] and [16]. We focus our attention on the construction of the so-called spline complex, i.e., spline approximation spaces for the De Rham diagram. In the finite element setting, the construction of discrete De Rham complexes has been object of intense study, and its various mathematical aspects have been object of three review papers: for computational electromagnetics [27], for discrete exterior calculus [3] and for eigenvalue problems [10].

Throughout this section, we will work in three space dimensions, however the constructions and results apply to any space dimensions with obvious changes. The reason for this choice is to avoid the use of the language of differential forms and to improve clarity. Moreover, we will often use a "verbose" notation. For example, the space $S_{\mathbf{p}}(\boldsymbol{\Xi})$ will also be denoted $S_{p_1,p_2,p_3}(\Xi_1, \Xi_2, \Xi_3)$. The reason for this choice will be clear when we introduce the vector spline spaces.

In Sect. 5.1 we briefly recall the De Rham diagram for three-dimensional domains, and define the continuous spaces in the parametric domain $\hat{\Omega} = (0,1)^3$, and in the physical domain $\Omega = \mathbf{F}(\hat{\Omega})$. In Sect. 5.2 we construct the De Rham complex with discrete spline spaces, called here *the spline complex*, which is obtained by taking a combination of spline spaces of suitable mixed degree. The construction is first done in the parametric domain, and then mapped to the physical domain using suitable mappings defined from $\mathbf{F}$. Then, in Sect. 5.3 we define a set of projectors in such a way that the continuous and the discrete spaces form a commutative De Rham diagram. Using these projectors we prove approximation estimates in Sect. 5.4. Finally, the spaces of the spline complex are applied to the discretization of time harmonic Maxwell equations in Sect. 5.5, and some numerical tests are presented.

## 5.1 The De Rham Complex

In this section, we briefly introduce the concept of the De Rham complex and introduce our notation for the related spaces. We adopt the language of functional analysis and proxy fields and not the one of differential forms, and we mainly concentrate on the definition of pull-backs that will be needed in the next sections.

We first recall that in $\mathbb{R}^3$, for a vector $\mathbf{u}$ the curl and the divergence are given by

$$\operatorname{curl} \mathbf{u} = \left( \frac{\partial u_3}{\partial x_2} - \frac{\partial u_2}{\partial x_3}, \frac{\partial u_1}{\partial x_3} - \frac{\partial u_3}{\partial x_1}, \frac{\partial u_2}{\partial x_1} - \frac{\partial u_1}{\partial x_2} \right),$$

$$\operatorname{div} \mathbf{u} = \frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2} + \frac{\partial u_3}{\partial x_3}.$$

For a general domain $D$, $L^2(D)$ is the space of real valued, square integrable functions, and

$$\begin{aligned}\mathbf{H}(\mathrm{curl};D) &:= \{\mathbf{u}\in L^2(D)^3 \ : \ \mathrm{curl}\ \mathbf{u}\in L^2(D)^3\},\\ \mathbf{H}(\mathrm{div};D) &:= \{\mathbf{u}\in L^2(D)^3 \ : \ \mathrm{div}\ \mathbf{u}\in L^2(D)\}.\end{aligned}$$

As in the previous sections, we will assume that our physical domain is given by a NURBS parametrization. Let $\hat{\Omega}=(0,1)^3$ and $\Omega$ be a single-patch physical domain, i.e., there exists a map $\mathbf{F}:\hat{\Omega}\to\mathbb{R}^3$ satisfying Assumption 3.1 such that $\Omega=\mathbf{F}(\hat{\Omega})$. Due to Assumption 3.1, the domain $\Omega$ is *simply connected with connected boundary*. We define the spaces

$$\begin{aligned}\hat{X}^0 &:= H^1(\hat{\Omega})\,,\ \hat{X}^1 := \mathbf{H}(\mathrm{curl};\hat{\Omega})\,,\ \hat{X}^2 := \mathbf{H}(\mathrm{div};\hat{\Omega})\,,\ \hat{X}^3 := L^2(\hat{\Omega})\,,\\ X^0 &:= H^1(\Omega)\,,\ X^1 := \mathbf{H}(\mathrm{curl};\Omega)\,,\ X^2 := \mathbf{H}(\mathrm{div};\Omega)\,,\ X^3 := L^2(\Omega)\,.\end{aligned}$$

Notice that $\hat{X}^j$ and $X^j$ are spaces of scalar fields for $j=0,3$, while they are spaces of vector fields for the indices $j=1,2$. The spaces with homogeneous boundary conditions will be indicated with the subindex 0. The boundary conditions for $\mathbf{H}_0(\mathrm{curl};\Omega)$ and $\mathbf{H}_0(\mathrm{div};\Omega)$ refer to the tangential and the normal components of the vectors, respectively.

Thanks to Assumption 3.1, both $\mathbf{F}$ and its inverse are smooth. We can then define the pull-backs that relate the spaces in the parametric and in the physical domain as (see [27, Sect. 2.2]):

$$\begin{aligned}&\iota^0(\phi) := \phi\circ\mathbf{F}, && \phi\in X^0,\\ &\iota^1(\mathbf{u}) := (D\mathbf{F})^\top(\mathbf{u}\circ\mathbf{F}), && \mathbf{u}\in X^1,\\ &\iota^2(\mathbf{v}) := \det(D\mathbf{F})(D\mathbf{F})^{-1}(\mathbf{v}\circ\mathbf{F}), && \mathbf{v}\in X^2,\\ &\iota^3(\psi) := \det(D\mathbf{F})(\psi\circ\mathbf{F}), && \psi\in X^3,\end{aligned} \tag{67}$$

where $D\mathbf{F}$ is the Jacobian matrix of the mapping $\mathbf{F}$. Note that these are the same pull-backs used in the definition of finite elements, with the only difference that in finite elements the parametrization $\mathbf{F}$ maps the reference element to the element in the mesh of the physical domain.

Due to the curl and divergence conserving properties of $\iota^1$ and $\iota^2$, respectively, the following commutative De Rham diagram is automatically satisfied (see [27, Sect. 2.2]):

$$\begin{array}{ccccccccccc}
\mathbb{R} & \longrightarrow & \hat{X}^0 & \xrightarrow{\widehat{\mathrm{grad}}} & \hat{X}^1 & \xrightarrow{\widehat{\mathrm{curl}}} & \hat{X}^2 & \xrightarrow{\widehat{\mathrm{div}}} & \hat{X}^3 & \longrightarrow & 0\\
 & & \iota^0\uparrow & & \iota^1\uparrow & & \iota^2\uparrow & & \iota^3\uparrow & & \\
\mathbb{R} & \longrightarrow & X^0 & \xrightarrow{\mathrm{grad}} & X^1 & \xrightarrow{\mathrm{curl}} & X^2 & \xrightarrow{\mathrm{div}} & X^3 & \longrightarrow & 0.
\end{array} \tag{68}$$

Moreover, since $\Omega$ is simply connected with connected boundary, the sequence is exact. This means that the kernel of each operator is exactly the image of the preceding one, and the last operator has full range, that is

$$\ker(\text{grad}) = \mathbb{R}, \quad \ker(\text{curl}) = \text{Im}(\text{grad}), \quad \ker(\text{div}) = \text{Im}(\text{curl}), \quad X^3 = \text{Im}(\text{div}).$$

## 5.2 Definition of the Spaces of the Spline Complex

We now define a sequence of spline spaces, that will form a De Rham spline complex. We start by introducing the spaces in the parametric domain, and then map them to the physical domain.

### 5.2.1 The Spline Complex on the Parametric Domain

First of all, we recall that in the one-dimensional case, the derivative is a surjective operator from $S_p(\Xi)$ to $S_{p-1}(\Xi')$, where the spaces of the derivatives is defined using the knot vector $\Xi' = \{\xi_2, \dots, \xi_{n+p}\}$. Moreover, the derivative of a B-spline function in the one-dimensional case is given in (6).

Using the expression for the derivative in three dimensions, it is clear that, for instance,

$$\frac{\partial}{\partial \zeta_1} : S_{p_1,p_2,p_3}(\Xi_1, \Xi_2, \Xi_3) \to S_{p_1-1,p_2,p_3}(\Xi_1', \Xi_2, \Xi_3),$$

where the knot vector $\Xi_1'$ is defined from $\Xi_1$ as in the one-dimensional case above.

Following the same rationale, we define the spaces on the parametric domain $\hat{\Omega}$:

$$\begin{aligned}
\hat{X}_h^0 &:= S_{p_1,p_2,p_3}(\Xi_1, \Xi_2, \Xi_3), \\
\hat{X}_h^1 &:= S_{p_1-1,p_2,p_3}(\Xi_1', \Xi_2, \Xi_3) \times S_{p_1,p_2-1,p_3}(\Xi_1, \Xi_2', \Xi_3) \times S_{p_1,p_2,p_3-1}(\Xi_1, \Xi_2, \Xi_3'), \\
\hat{X}_h^2 &:= S_{p_1,p_2-1,p_3-1}(\Xi_1, \Xi_2', \Xi_3') \times S_{p_1-1,p_2,p_3-1}(\Xi_1', \Xi_2, \Xi_3') \times S_{p_1-1,p_2-1,p_3}(\Xi_1', \Xi_2', \Xi_3), \\
\hat{X}_h^3 &:= S_{p_1-1,p_2-1,p_3-1}(\Xi_1', \Xi_2', \Xi_3').
\end{aligned} \tag{69}$$

In order for $\hat{X}_h^1$, $\hat{X}_h^2$ and $\hat{X}_h^3$ to be meaningful, we require $0 \leq m_{\ell,i} \leq p_\ell$, for $i = 2, \dots, N_\ell - 1$ and $\ell = 1, 2, 3$. This means that the functions in $\hat{X}_h^0$ are at least continuous. Then, thanks to (5) it is easily seen that $\widehat{\text{grad}}(\hat{X}_h^0) \subset \hat{X}_h^1$, and analogously, from the definition of the curl and the divergence operators we get $\widehat{\text{curl}}(\hat{X}_h^1) \subset \hat{X}_h^2$, and $\widehat{\text{div}}(\hat{X}_h^2) \subset \hat{X}_h^3$. Moreover, it is proved in [15] that the kernel of

each operator is exactly the image of the preceding one. In other words, these spaces form an exact sequence:

$$\mathbb{R} \longrightarrow \hat{X}_h^0 \xrightarrow{\widehat{\text{grad}}} \hat{X}_h^1 \xrightarrow{\widehat{\text{curl}}} \hat{X}_h^2 \xrightarrow{\widehat{\text{div}}} \hat{X}_h^3 \longrightarrow 0, \tag{70}$$

that is, a discrete version of the first line of (68).

### 5.2.2 The Spline Complex in the Physical Domain

As in Sect. 3, we suppose $\Omega$ is obtained from $\hat{\Omega}$ through a NURBS (or spline) single patch mapping $\mathbf{F} \in (N_{\mathbf{p}^0}(\boldsymbol{\Xi}^0, W))^3$, verifying Assumption 3.1. To construct the spaces (69) in the parametric domain, first we choose the space $\hat{X}_h^0 = S_{p_1,p_2,p_3}(\Xi_1, \Xi_2, \Xi_3) = S_{\mathbf{p}}(\boldsymbol{\Xi})$, in such a way that $S_{\mathbf{p}^0}(\boldsymbol{\Xi}^0) \subset S_{\mathbf{p}}(\boldsymbol{\Xi})$. That is, we first consider the spline space used to define the NURBS geometry, in practice neglecting the weight, and the space $\hat{X}_h^0$ is a refinement of this spline space. Once the first space of the spline complex is chosen, the other spaces in the parametric domain are defined from (69).

The discrete spaces $X_h^0, \ldots, X_h^3$ in the physical domain $\Omega$ can be defined from the spaces (69) on the parametric domain $\hat{\Omega}$ by *push-forward*, that is, the inverse of the transformations defined in (67), that commute with the differential operators (as given by the diagram (68)):

$$\begin{array}{ccccccccc}
\mathbb{R} & \longrightarrow & \hat{X}_h^0 & \xrightarrow{\widehat{\text{grad}}} & \hat{X}_h^1 & \xrightarrow{\widehat{\text{curl}}} & \hat{X}_h^2 & \xrightarrow{\widehat{\text{div}}} & \hat{X}_h^3 \longrightarrow 0 \\
 & & \iota^0 \uparrow & & \iota^1 \uparrow & & \iota^2 \uparrow & & \iota^3 \uparrow \\
\mathbb{R} & \longrightarrow & X_h^0 & \xrightarrow{\text{grad}} & X_h^1 & \xrightarrow{\text{curl}} & X_h^2 & \xrightarrow{\text{div}} & X_h^3 \longrightarrow 0,
\end{array} \tag{71}$$

that is, the discrete spaces in the physical domain are defined as

$$\begin{aligned}
X_h^0 &:= \{\phi_h : \iota^0(\phi_h) \in \hat{X}_h^0\}, \\
X_h^1 &:= \{\mathbf{u}_h : \iota^1(\mathbf{u}_h) \in \hat{X}_h^1\}, \\
X_h^2 &:= \{\mathbf{v}_h : \iota^2(\mathbf{v}_h) \in \hat{X}_h^2\}, \\
X_h^3 &:= \{\psi_h : \iota^3(\psi_h) \in \hat{X}_h^3\}.
\end{aligned} \tag{72}$$

We remark that the space $X_h^1$, which is a discretization of $\mathbf{H}(\text{curl}; \Omega)$, is defined through the curl conserving transformation $\iota^1$, and that the space $X_h^2$, which is a discretization of $\mathbf{H}(\text{div}; \Omega)$, is defined through the divergence conforming transformation $\iota^2$. These are equivalent to the curl and divergence preserving

transformations that are used to define edge and face elements, respectively (see [34, Sect. 3.9]).

Thanks to the properties of the operators (67) the spaces $X_h^0, \dots, X_h^3$ inherit the same fundamental properties of $\hat{X}_h^0, \dots, \hat{X}_h^3$, that we have discussed in the previous section, and in particular they form an exact De Rham complex.

### 5.2.3 Choice of Bases and Topological Structure

Let us first remind the formula of the derivative of a B-spline function, already given in (6):

$$\frac{d\hat{B}_{i,p}}{d\zeta}(\zeta) = \hat{D}_{i-1,p-1}(\zeta) - \hat{D}_{i,p-1}(\zeta),$$

where $\hat{D}_{i,p-1} = \dfrac{p}{\xi_{i+p+1} - \xi_{i+1}} \hat{B}_{i+1,p-1}$. As in [42] and [16], we make use of the univariate functions $\hat{D}_{i,p-1}$ to choose different bases for the spaces in the spline complex. It is immediate to see that

$$\hat{X}_h^i = \text{span}(\hat{\mathcal{B}}^i), \text{ for } i = 0, \dots, 3,$$

where the bases $\hat{\mathcal{B}}^i$ are defined as follows:

$$\hat{\mathcal{B}}^0 = \left\{ \hat{B}_{i_1,p_1}(\zeta_1) \hat{B}_{i_2,p_2}(\zeta_2) \hat{B}_{i_3,p_3}(\zeta_3),\ 1 \le i_\ell \le n_\ell,\ \ell = 1,2,3 \right\}, \tag{73}$$

$$\begin{aligned}
\hat{\mathcal{B}}^1 &= (I \cup II \cup III) \text{ with} \\
I &= \left\{ \hat{D}_{i_1,p_1-1}(\zeta_1) \hat{B}_{i_2,p_2}(\zeta_2) \hat{B}_{i_3,p_3}(\zeta_3) \hat{\mathbf{e}}_1,\ 1 \le i_1 \le n_1 - 1,\ 1 \le i_\ell \le n_\ell,\ \ell = 2,3 \right\}, \\
II &= \left\{ \hat{B}_{i_1,p_1}(\zeta_1) \hat{D}_{i_2,p_2-1}(\zeta_2) \hat{B}_{i_3,p_3}(\zeta_3) \hat{\mathbf{e}}_2,\ 1 \le i_2 \le n_2 - 1,\ 1 \le i_\ell \le n_\ell,\ \ell = 1,3 \right\}, \\
III &= \left\{ \hat{B}_{i_1,p_1}(\zeta_1) \hat{B}_{i_2,p_2}(\zeta_2) \hat{D}_{i_3,p_3-1}(\zeta_3) \hat{\mathbf{e}}_3,\ 1 \le i_3 \le n_3 - 1,\ 1 \le i_\ell \le n_\ell,\ \ell = 1,2 \right\},
\end{aligned} \tag{74}$$

$$\begin{aligned}
\hat{\mathcal{B}}^2 &= (I \cup II \cup III) \text{ with} \\
I &= \left\{ \hat{B}_{i_1,p_1}(\zeta_1) \hat{D}_{i_2,p_2-1}(\zeta_2) \hat{D}_{i_3,p_3-1}(\zeta_3) \hat{\mathbf{e}}_1,\ 1 \le i_1 \le n_1,\ 1 \le i_\ell \le n_\ell - 1,\ \ell = 2,3 \right\}, \\
II &= \left\{ \hat{D}_{i_1,p_1-1}(\zeta_1) \hat{B}_{i_2,p_2}(\zeta_2) \hat{D}_{i_3,p_3-1}(\zeta_3) \hat{\mathbf{e}}_2,\ 1 \le i_2 \le n_2,\ 1 \le i_\ell \le n_\ell - 1,\ \ell = 1,3 \right\}, \\
III &= \left\{ \hat{D}_{i_1,p_1-1}(\zeta_1) \hat{D}_{i_2,p_2-1}(\zeta_2) \hat{B}_{i_3,p_3}(\zeta_3) \hat{\mathbf{e}}_3,\ 1 \le i_3 \le n_3,\ 1 \le i_\ell \le n_\ell - 1,\ \ell = 1,2 \right\},
\end{aligned} \tag{75}$$

$$\hat{\mathcal{B}}^3 = \left\{ \hat{D}_{i_1,p_1-1}(\zeta_1)\hat{D}_{i_2,p_2-1}(\zeta_2)\hat{D}_{i_3,p_3-1}(\zeta_3),\ 1 \le i_\ell \le n_\ell - 1,\ \ell = 1,2,3 \right\}, \tag{76}$$

where $\{\hat{\mathbf{e}}_\ell\}_{\ell=1,2,3}$ denote the canonical basis of $\mathbb{R}^3$. We remark that all basis functions defined in (73), (74), (75), and (76) are non-negative, and vectorial basis functions have the same orientation as the vectors of the canonical basis.

Analogously to (72), the basis functions in the physical domain are defined by applying the pull-backs (67), namely

$$\mathcal{B}^i = \{\iota^i(\phi), \phi \in \hat{\mathcal{B}}^i\}, \quad \text{for } i = 0, \dots, 3.$$

When constructing discrete spaces using the isoparametric approach, as in Sect. 3, we can associate one basis function to each control point in the control mesh (possibly refined). For the basis functions of the spline complex the situation is very similar, and we can associate our basis functions to the geometrical entities of the control mesh. In particular, the basis functions in $\mathcal{B}^0$, $\mathcal{B}^1$, $\mathcal{B}^2$ and $\mathcal{B}^3$ are associated to the vertices, the edges, the faces and the volumes of the control mesh, respectively. In Fig. 17, the degrees of freedom location for $X_h^0$ and for $X_h^1$ are shown.

The main reason for the use of the modified basis functions $\hat{D}_{i,p-1}$ is given in the following proposition from [16]:

**Proposition 5.1** *With the choices* (73)*,* (74)*,* (75)*, and* (76)*, the matrices representing differential operators* grad*,* curl*, and* div *are the incidence matrices of the control mesh. Thus, the spline complex* $(X_h^0, X_h^1, X_h^2, X_h^3)$ *is isomorphic to the cochain complex associated with the control mesh.*

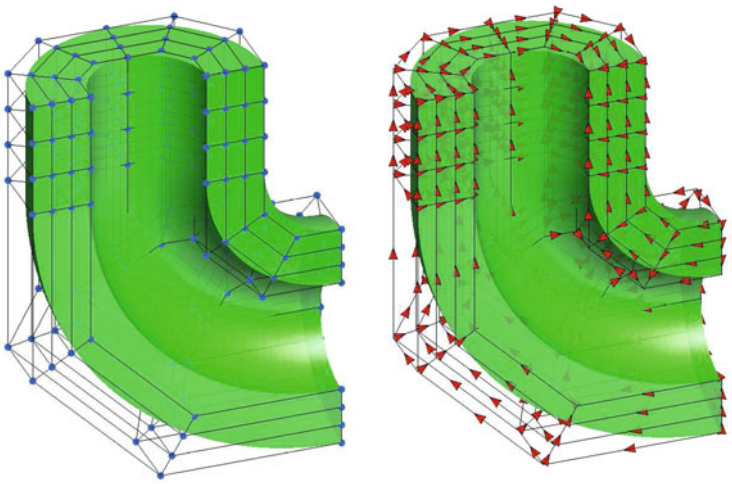

**Fig. 17** *Left*: Representation of the degrees-of-freedom location (*blue dots*) for the space $X_h^0$. *Right*: Representation of the degrees-of-freedom location (*red arrows*) for the space $X_h^1$

The previous proposition states that the spline complex has exactly the same structure of the well known Whitney forms when defined on the control mesh, see, e.g. Section 3 in [27].

*Remark 5.2* In the paper [16], the authors have introduced the concept of control field, which are Whitney forms defined on the control mesh. In view of the properties listed above, of Proposition 5.1, and the properties of the pull-back operators, the Whitney vector fields can be interpreted as control fields, i.e., exactly in the spirit of control points, fields which "carry" the degrees of freedom for the spline complex. It should be noted that, on the same setting of degrees of freedom, the spline complex enjoys approximation properties of order $h^p$, where $p = \min\{p_1, p_2, p_3\}$, whereas the underlying Whitney forms provide only the low order approximation rates.

## 5.3 Commutative Projections

It is now necessary to define suitable projectors into the discrete spaces of the spline complex in order to prove the approximation estimates in the corresponding norms of the space. In the context of discrete differential forms, what we need to prove the approximation estimates is a set of projectors $\Pi^i$ that render the following diagram commutative:

$$\begin{array}{ccccccccccc} \mathbb{R} & \longrightarrow & X^0 & \xrightarrow{\text{grad}} & X^1 & \xrightarrow{\text{curl}} & X^2 & \xrightarrow{\text{div}} & X^3 & \longrightarrow & 0 \\ & & \Pi^0 \downarrow & & \Pi^1 \downarrow & & \Pi^2 \downarrow & & \Pi^3 \downarrow & & \\ \mathbb{R} & \longrightarrow & X_h^0 & \xrightarrow{\text{grad}} & X_h^1 & \xrightarrow{\text{curl}} & X_h^2 & \xrightarrow{\text{div}} & X_h^3 & \longrightarrow & 0. \end{array} \tag{77}$$

To define these kind of projectors for our spline spaces, we will proceed as in previous sections. We will start by defining suitable projectors in the one-dimensional case. These will be then generalized to the tensor product case, defining the set of projector in the parametric domain, and at the last step they will be mapped to the physical domain using the appropriate pull-backs.

### 5.3.1 Commutative Projectors in the One-Dimensional Case

Our starting point is the projector for the univariate case, $\Pi_{p,\Xi}$, defined in (18). We want to define a new projector $\Pi^c_{p-1,\Xi'}$ such that the following diagram commutes

$$\begin{array}{ccccccc} \mathbb{R} & \longrightarrow & H^1(0,1) & \xrightarrow{\frac{d}{d\zeta}} & L^2(0,1) & \longrightarrow & 0 \\ & & \Pi_{p,\Xi} \downarrow & & \Pi^c_{p-1,\Xi'} \downarrow & & \\ \mathbb{R} & \longrightarrow & S_p(\Xi) & \xrightarrow{\frac{d}{d\zeta}} & S_{p-1}(\Xi') & \longrightarrow & 0. \end{array} \tag{78}$$

Therefore we define

$$\Pi^c_{p-1,\Xi'} g\,(\zeta) := \frac{d}{d\zeta} \Pi_{p,\Xi} \int_0^\zeta g(s)\,\mathrm{d}s, \tag{79}$$

for all functions $g$ such that $f(\zeta) = \int_0^\zeta g(s)\,\mathrm{d}s$ is in $L^2(0,1)$, which is the domain of definition of $\Pi_{p,\Xi}$. The index $c$ stands for commutative and it is indeed trivial to see that the projectors commute with the derivative, i.e.,

$$\Pi^c_{p-1,\Xi'} \frac{d}{d\zeta} f = \frac{d}{d\zeta} \Pi_{p,\Xi} f, \ \text{for} f \in L^2(0,1). \tag{80}$$

Moreover, and as a consequence of the spline preserving property (20), it is also immediate to prove that $\Pi^c_{p-1,\Xi'}$ preserves B-splines, that is

$$\Pi^c_{p-1,\Xi'} g = g \qquad \forall g \in S_{p-1}(\Xi'), \tag{81}$$

and from the two previous results we get that the diagram (78) commutes.

Furthermore, it is also possible to prove that, analogously to $\Pi_{p,\Xi}$, the new projector can be defined from a dual basis, and that it is stable, as stated in the following proposition.

**Proposition 5.3** *Let $g \in L^2(0,1)$, and let the projector $\Pi_{p,\Xi}$ be defined as in (18), that is, $\Pi_{p,\Xi} f(\zeta) = \sum_{i=1}^n \lambda_{i,p}(f)\hat{B}_{i,p}(\zeta)$ for any $f \in L^2(0,1)$. Then it holds:*

$$\Pi^c_{p-1,\Xi'} g(\zeta) = \sum_{j=1}^{n-1} \lambda^c_{j,p-1}(g)\hat{D}_{j,p-1}(\zeta),$$

*with*

$$\lambda^c_{j,p-1}(g) = \lambda_{j+1,p}\left(\int_{\xi_j}^\zeta g(s)\mathrm{d}s\right) - \lambda_{j,p}\left(\int_{\xi_j}^\zeta g(s)\mathrm{d}s\right). \tag{82}$$

*Moreover, if Assumption 2.3 is satisfied, then for all $I_i = (\zeta_i, \zeta_{i+1})$, it holds:*

$$\|\Pi^c_{p-1,\Xi'} g\|_{L^2(I_i)} \le C\|g\|_{L^2(\tilde{I}_i)}, \tag{83}$$

*where $\tilde{I}_i$ is the support extension of $I_i$ defined in (7).*

*Proof* Let $f(\zeta) := \int_0^\zeta g(s)\mathrm{d}s$. By definition of $\Pi^c_{p-1,\Xi'}$ and $\Pi_{p,\Xi}$, and then using the expression for the derivative (6), we have

$$\Pi^c_{p-1,\Xi'} g(\zeta) = \frac{d}{d\zeta}\Pi_{p,\Xi} f(\zeta) = \frac{d}{d\zeta}\sum_{i=1}^{n} \lambda_{i,p}(f)\hat{B}_{i,p}(\zeta) = \sum_{i=1}^{n} \lambda_{i,p}(f)(\hat{D}_{i-1,p}(\zeta) - \hat{D}_{i,p}(\zeta)),$$

and recalling the convention $\hat{D}_{0,p}(\zeta) = \hat{D}_{n,p}(\zeta) = 0$, we obtain

$$\Pi^c_{p-1,\Xi'} g(\zeta) = \sum_{j=1}^{n-1} \left(\lambda_{j+1,p}(f) - \lambda_{j,p}(f)\right) \hat{D}_{j,p-1}(\zeta).$$

Due to the linearity of the functionals $\lambda_{j,p}$, we have, for any given $\zeta^* \in \mathbb{R}$

$$\lambda_{j,p}(f) = \lambda_{j,p}\left(\int_0^{\zeta^*} g(s)\mathrm{d}s\right) + \lambda_{j,p}\left(\int_{\zeta^*}^{\zeta} g(s)\mathrm{d}s\right),$$

and noting that the term $\int_0^{\xi_j} g(s)\mathrm{d}s$ is a constant, thanks to the partition of unity of the B-spline functions $\hat{B}_{i,p}$ it holds

$$\lambda_{j+1,p}\left(\int_0^{\xi_j} g(s)\mathrm{d}s\right) = \lambda_{j,p}\left(\int_0^{\xi_j} g(s)\mathrm{d}s\right).$$

Combining the last three equations, we obtain (82).

To prove (83), we use again the definition of $\Pi^c_{p-1,\Xi'}$, and then the stability of the projector $\Pi_{p,\Xi}$ from (23), to get

$$\|\Pi^c_{p-1,\Xi'} g\|_{L^2(I_i)} = |\Pi_{p,\Xi} f|_{H^1(I_i)} \leq C|f|_{H^1(\tilde{I}_i)} = C\|g\|_{L^2(\tilde{I}_i)},$$

and the result is proved. □

*Remark 5.4* Notice that the definition of the dual functional $\lambda^c_{j,p-1}$ depends on the local knot vectors $\Xi_{j,p}$ and $\Xi_{j+1,p}$, and therefore it goes beyond the support of $\hat{D}_{j,p-1}$. Moreover, in the estimate (83), the support extension $\tilde{I}_i$ is defined for degree $p$, not $p-1$. This means that the projector $\Pi^c_{p-1,\Xi'}$ loses some locality with respect to $\Pi_{p-1,\Xi'}$, which would be the quasi-interpolant defined in [46, Sect. 4.6]. This is the "price to pay" in order to obtain the commutative diagram.

### 5.3.2 Commutative Projectors in the Parametric Domain

We now define appropriate projectors into the discrete spaces in the parametric domain, by tensor product of the univariate projectors of the previous section.

To alleviate notation, from this point we will not detail the knot vector in the interpolant, that is, we will denote $\Pi_p \equiv \Pi_{p,\Xi}$ and $\Pi^c_{p-1} \equiv \Pi^c_{p-1,\Xi'}$. The choice of the interpolants follows from the definition of the spaces $\hat{X}^0_h, \ldots, \hat{X}^3_h$ in (69), and precisely we set:

$$\hat{\Pi}^0 := \Pi_{p_1} \otimes \Pi_{p_2} \otimes \Pi_{p_3}, \tag{84}$$

$$\begin{aligned}\hat{\Pi}^1 := &(\Pi^c_{p_1-1} \otimes \Pi_{p_2} \otimes \Pi_{p_3}) \times (\Pi_{p_1} \otimes \Pi^c_{p_2-1} \otimes \Pi_{p_3}) \\ &\times(\Pi_{p_1} \otimes \Pi_{p_2} \otimes \Pi^c_{p_3-1}),\end{aligned} \tag{85}$$

$$\begin{aligned}\hat{\Pi}^2 := &(\Pi_{p_1} \otimes \Pi^c_{p_2-1} \otimes \Pi^c_{p_3-1}) \times (\Pi^c_{p_1-1} \otimes \Pi_{p_2} \otimes \Pi^c_{p_3-1}) \\ &\times(\Pi^c_{p_1-1} \otimes \Pi^c_{p_2-1} \otimes \Pi_{p_3}),\end{aligned} \tag{86}$$

$$\hat{\Pi}^3 := \Pi^c_{p_1-1} \otimes \Pi^c_{p_2-1} \otimes \Pi^c_{p_3-1}. \tag{87}$$

The next lemma shows that the interpolants are projectors onto the corresponding spline spaces.

**Lemma 5.5** *The interpolants* (84), (85), (86), *and* (87) *satisfy the spline preserving property, that is*

$$\begin{aligned}\hat{\Pi}^0\hat{\phi}_h = \hat{\phi}_h, &\quad \forall \hat{\phi}_h \in \hat{X}^0_h,\\ \hat{\Pi}^1\hat{\mathbf{u}}_h = \hat{\mathbf{u}}_h, &\quad \forall \hat{\mathbf{u}}_h \in \hat{X}^1_h,\\ \hat{\Pi}^2\hat{\mathbf{v}}_h = \hat{\mathbf{v}}_h, &\quad \forall \hat{\mathbf{v}}_h \in \hat{X}^2_h,\\ \hat{\Pi}^3\hat{\psi}_h = \hat{\psi}_h, &\quad \forall \hat{\psi}_h \in \hat{X}^3_h.\end{aligned}$$

*Proof* The result is an immediate consequence of the splines preserving property of the interpolants $\Pi_{p_\ell}$ and $\Pi^c_{p_\ell-1}$, $\ell = 1, 2, 3$, given in (20) and in (81), respectively. □

Moreover, the projectors are also stable in the $L^2(\Omega)$ norm.

**Lemma 5.6** *Under Assumption 4.7, the following inequalities hold for any* $Q \in \widehat{\mathcal{M}}$*:*

$$\begin{aligned}\|\hat{\Pi}^0\hat{\phi}\|_{L^2(Q)} &\le C\|\hat{\phi}\|_{L^2(\tilde{Q})} \quad \forall \hat{\phi} \in L^2(\hat{\Omega}),\\ \|\hat{\Pi}^1\hat{\mathbf{u}}\|_{L^2(Q)^3} &\le C\|\hat{\mathbf{u}}\|_{\mathbf{L}^2(\tilde{Q})} \quad \forall \hat{\mathbf{u}} \in L^2(\hat{\Omega})^3,\\ \|\hat{\Pi}^2\hat{\mathbf{v}}\|_{L^2(Q)^3} &\le C\|\hat{\mathbf{v}}\|_{\mathbf{L}^2(\tilde{Q})} \quad \forall \hat{\mathbf{v}} \in L^2(\hat{\Omega})^3,\\ \|\hat{\Pi}^3\hat{\psi}\|_{L^2(Q)} &\le C\|\hat{\psi}\|_{L^2(\tilde{Q})} \quad \forall \hat{\psi} \in L^2(\hat{\Omega}).\end{aligned}$$

*Proof* The result follows immediately from (22) and (83), which state that the involved one-dimensional operators $\Pi_{p_\ell}$ and $\Pi^c_{p_\ell-1}$, $\ell = 1, 2, 3$ are $L^2$ stable. □

The commutativity of the interpolants with the differential operators is stated in the following lemma.

**Lemma 5.7** *It holds*

$$\widehat{\mathrm{grad}}(\hat{\Pi}^0\hat{\phi}) = \hat{\Pi}^1(\widehat{\mathrm{grad}}\hat{\phi}) \qquad \forall \hat{\phi} \in \hat{X}^0, \tag{88}$$

$$\widehat{\mathrm{curl}}(\hat{\Pi}^1\hat{\mathbf{u}}) = \hat{\Pi}^2(\widehat{\mathrm{curl}}\hat{\mathbf{u}}) \qquad \forall \hat{\mathbf{u}} \in \hat{X}^1, \tag{89}$$

$$\widehat{\mathrm{div}}(\hat{\Pi}^2\hat{\mathbf{v}}) = \hat{\Pi}^3(\widehat{\mathrm{div}}\hat{\mathbf{v}}) \qquad \forall \hat{\mathbf{v}} \in \hat{X}^2. \tag{90}$$

*Proof* The proof is based on the commutativity property (80) and the tensor product structure of the spaces and interpolants. Consider first (88): let $\hat{f}$ be a smooth scalar field with compact support in $\hat{\Omega}$. With some abuse of notation, the first component of $\widehat{\mathrm{grad}}(\hat{\Pi}^0\hat{f})$ is given by

$$\begin{aligned}\partial_{\hat{x}}(\hat{\Pi}^0\hat{f}) &= \partial_{\hat{x}}((\Pi_{p_1} \otimes \Pi_{p_2} \otimes \Pi_{p_3})\hat{f}) = \partial_{\hat{x}}(\Pi_{p_1}(\Pi_{p_2}(\Pi_{p_3}\hat{f}))) \\ &= \Pi^c_{p_1-1}\partial_{\hat{x}}(\Pi_{p_2}(\Pi_{p_3}\hat{f})) = (\Pi^c_{p_1-1} \otimes \Pi_{p_2} \otimes \Pi_{p_3})\,\partial_{\hat{x}}\hat{f},\end{aligned}$$

which is the first component of $\hat{\Pi}^1(\widehat{\mathrm{grad}}\hat{f})$. A similar reasoning, using the commutativity of the univariate interpolants, yields

$$\begin{aligned}\partial_{\hat{y}}(\hat{\Pi}^0\hat{f}) &= (\Pi_{p_1} \otimes \Pi^c_{p_2-1} \otimes \Pi_{p_3})\,\partial_{\hat{y}}\hat{f}, \\ \partial_{\hat{z}}(\hat{\Pi}^0\hat{f}) &= (\Pi_{p_1} \otimes \Pi_{p_2} \otimes \Pi^c_{p_3-1})\,\partial_{\hat{z}}\hat{f},\end{aligned}$$

which proves that $\widehat{\mathrm{grad}}(\hat{\Pi}^0\hat{f}) = \hat{\Pi}^1(\widehat{\mathrm{grad}}\hat{f})$. By a density argument (88) follows, thanks to Lemma 5.6. The proof of (89)–(90) is similar, from the definition of the interpolants and the expression of the curl and divergence operators. □

### 5.3.3 Commutative Projectors in the Physical Domain

Finally, we define the projectors into the spaces $X^i_h$ of the complex in the physical domain. These operators are defined from the ones in the parametric domain (84), (85), (86), and (87), by applying the corresponding pull-backs from (67). Hence, the

projectors in the physical domain are uniquely determined by the equations

$$\begin{aligned} \iota^0(\Pi^0\phi) &= \hat{\Pi}^0(\iota^0(\phi)), \\ \iota^1(\Pi^1\mathbf{u}) &= \hat{\Pi}^1(\iota^1(\mathbf{u})), \\ \iota^3(\Pi^2\mathbf{v}) &= \hat{\Pi}^2(\iota^2(\mathbf{v})), \\ \iota^2(\Pi^3\psi) &= \hat{\Pi}^3(\iota^3(\psi)). \end{aligned} \tag{91}$$

As an immediate consequence of the previous definitions we have that the operators $\Pi^i$ are projectors, that is, they preserve the functions of the spaces $X_h^i$. Moreover, from these definitions together with the commutative properties of Lemma 5.7, it is immediate to prove the following result.

**Proposition 5.8** *The following diagram commutes.*

$$\begin{array}{ccccccccccc} \mathbb{R} & \longrightarrow & X^0 & \xrightarrow{\text{grad}} & X^1 & \xrightarrow{\text{curl}} & X^2 & \xrightarrow{\text{div}} & X^3 & \longrightarrow & 0 \\ & & \Pi^0\downarrow & & \Pi^1\downarrow & & \Pi^2\downarrow & & \Pi^3\downarrow & & \\ \mathbb{R} & \longrightarrow & X_h^0 & \xrightarrow{\text{grad}} & X_h^1 & \xrightarrow{\text{curl}} & X_h^2 & \xrightarrow{\text{div}} & X_h^3 & \longrightarrow & 0. \end{array} \tag{92}$$

## 5.4 Approximation Estimates

This section is devoted to the study of the approximation estimates of the complex $(X_h^0, \ldots, X_h^3)$. The content of this section is mainly based on the papers [15] and [8]. First we need to define the bent Sobolev spaces for our spline spaces. Then we will give the approximation results for the spline complex in the parametric domain, and in the physical domain. We will only present the main results, and refer to [8] for the proofs.

For the sake of simplicity, in the following we will assume that $p_1 = p_2 = p_3 = p$.

### 5.4.1 Bent Sobolev Spaces

We start from the definition of the bent Sobolev spaces that we need. Since the interelement regularity changes from space to space (and from component to component), we need here to make the notation more explicit, starting from the one-dimensional definition: we denote by $\mathcal{H}^s_{\mathbf{k}}(I)$ the space defined in (37), where $\mathbf{k} = (k_2, \ldots, k_{N-1})$ and $k_i$ is the number of continuous derivatives at the point $\zeta_i \in Z$.

In three dimensions, given $\mathbf{s} = (s_1, s_2, s_3) \in \mathbb{N}^3$ and the three vectors $\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3$ constructed from $\boldsymbol{\Xi}$, we set:

$$\begin{aligned}
\mathcal{H}^{0,\mathbf{s}} &= \mathcal{H}^{\mathbf{s}}_{\mathbf{k}_1,\mathbf{k}_2,\mathbf{k}_3}, \\
\mathcal{H}^{1,\mathbf{s}} &= \mathcal{H}^{\mathbf{s}}_{\mathbf{k}_1-1,\mathbf{k}_2,\mathbf{k}_3} \times \mathcal{H}^{\mathbf{s}}_{\mathbf{k}_1,\mathbf{k}_2-1,\mathbf{k}_3} \times \mathcal{H}^{\mathbf{s}}_{\mathbf{k}_1,\mathbf{k}_2,\mathbf{k}_3-1}, \\
\mathcal{H}^{2,\mathbf{s}} &= \mathcal{H}^{\mathbf{s}}_{\mathbf{k}_1,\mathbf{k}_2-1,\mathbf{k}_3-1} \times \mathcal{H}^{\mathbf{s}}_{\mathbf{k}_1-1,\mathbf{k}_2,\mathbf{k}_3-1} \times \mathcal{H}^{\mathbf{s}}_{\mathbf{k}-1,\mathbf{k}_2-1,\mathbf{k}_3}, \\
\mathcal{H}^{3,\mathbf{s}} &= \mathcal{H}^{\mathbf{s}}_{\mathbf{k}-1,\mathbf{k}_2-1,\mathbf{k}_3-1}.
\end{aligned} \tag{93}$$

This choice is made in order to ensure that $\hat{X}^j_h \subset \mathcal{H}^{j,\mathbf{s}}$, for all $\mathbf{s} \in \mathbb{N}^3$, $j = 1, 2, 3$, i.e., the interelement regularity of $\mathcal{H}^{j,\mathbf{s}}$ is *not higher* than the one of $\hat{X}^j_h$. The corresponding norms and semi-norms are defined as in (46), component by component for the spaces $\mathcal{H}^{1,\mathbf{s}}$ and $\mathcal{H}^{2,\mathbf{s}}$. Moreover, for the scalar $|\mathbf{s}| = s_1 + s_2 + s_3$ we define the norm

$$|\phi|^2_{\mathcal{H}^{0,|\mathbf{s}|}} = \sup_{\mathbf{r}:|\mathbf{r}|\leq|\mathbf{s}|} |\phi|^2_{\mathcal{H}^{0,\mathbf{r}}},$$

and similar semi-norms can be defined for $|\cdot|_{\mathcal{H}^{j,|\mathbf{s}|}}$ for $j = 1, 2, 3$, component by component for vectorial spaces.

The necessity of these bent Sobolev spaces becomes clear with the following lemma, which generalizes the result from Proposition 4.13.

**Proposition 5.9** *Let $\mathbf{s} = (s_1, s_2, s_3) \in \mathbb{N}^3$ such that $|\mathbf{s}| = s$, and let $\phi, \psi \in H^s(\Omega)$, and $\mathbf{u}, \mathbf{v} \in H^s(\Omega)^3$. Then*

$$\begin{aligned}
\iota^0(\phi) &\in \mathcal{H}^{0,\mathbf{s}}, \\
\iota^1(\mathbf{u}) &\in \mathcal{H}^{1,\mathbf{s}}, \\
\iota^2(\mathbf{v}) &\in \mathcal{H}^{2,\mathbf{s}}, \\
\iota^3(\psi) &\in \mathcal{H}^{3,\mathbf{s}}.
\end{aligned} \tag{94}$$

*Moreover, there exists a constant $C$ such that for all elements $K = \mathbf{F}(Q) \in \mathcal{M}$, with $Q \in \widehat{\mathcal{M}}$, it holds:*

$$\begin{aligned}
C^{-1} \|\phi\|_{H^s(K)} &\leq \left\|\iota^0(\phi)\right\|_{H^s(Q)} \leq C \|\phi\|_{H^s(K)}, \\
C^{-1} \|\mathbf{u}\|_{H^s(K)^3} &\leq \left\|\iota^1(\mathbf{u})\right\|_{H^s(Q)^3} \leq C \|\mathbf{u}\|_{H^s(K)^3}, \\
C^{-1} \|\mathbf{v}\|_{H^s(K)^3} &\leq \left\|\iota^2(\mathbf{v})\right\|_{H^s(Q)^3} \leq C \|\mathbf{v}\|_{H^s(K)^3}, \\
C^{-1} \|\psi\|_{H^s(K)} &\leq \left\|\iota^3(\psi)\right\|_{H^s(Q)} \leq C \|\psi\|_{H^s(K)}.
\end{aligned}$$

*Proof* We focus on the proof for $\iota^1$ and $\mathbf{u} \in H^s(\Omega)^3$, since all the other cases are similar. We start proving (94). Let $\hat{\mathbf{u}} = \iota^1(\mathbf{u}) = (D\mathbf{F})^\top(\mathbf{u} \circ \mathbf{F})$. Since $\mathbf{F}$ is regular inside each element, we just need to check that the inter-element regularity is the one we expect. It is easy to see that, for instance,

$$\frac{\partial \mathbf{F}}{\partial \zeta_1} \in \mathcal{H}^{\mathbf{s}'}_{\mathbf{k}_1-1,\mathbf{k}_2,\mathbf{k}_3}, \ \forall \mathbf{s}' \in \mathbb{N}^3,$$

and a similar result for the other partial derivatives implies that $\iota^1(\mathbf{f}) \in \mathcal{H}^{1,\mathbf{s}}$. We refer to [15] for details.

The inequalities follow from the definition of the pull-backs $\iota^j$, by a simple application of the chain rule. □

### 5.4.2 Approximation Estimates in the Parametric Domain

Once we have defined the right bent Sobolev spaces, we present the approximation estimates for functions belonging to these spaces when approximated with the interpolants of the previous section. The result is given in the following proposition, that we present without detailing the proof.

**Proposition 5.10** *Let $\mathbf{s} \in \mathbb{N}^3$ with $|\mathbf{s}| = s_1 + s_2 + s_3$. Let Assumption 4.7 hold, and let moreover $Q$ be an element of $\widehat{\mathcal{M}}$, with $\tilde{Q}$ its extension. Then it holds,*

$$\left\| \hat{\phi} - \hat{\Pi}^0 \hat{\phi} \right\|_{L^2(Q)} \leq C h_{\tilde{Q}}^{|\mathbf{s}|} \, |\hat{\phi}|_{\mathcal{H}^{0,|\mathbf{s}|}(\tilde{Q})}, \quad 0 \leq |\mathbf{s}| \leq p+1, \ \text{for } \hat{\phi} \in \mathcal{H}^{0,\mathbf{t}} \text{ for all } \mathbf{t} \text{ with } |\mathbf{t}| \leq |\mathbf{s}|,$$

$$\left\| \hat{\mathbf{u}} - \hat{\Pi}^1 \hat{\mathbf{u}} \right\|_{L^2(Q)^3} \leq C h_{\tilde{Q}}^{|\mathbf{s}|} \, |\hat{\mathbf{u}}|_{\mathcal{H}^{1,|\mathbf{s}|}(\tilde{Q})^3}, \ 0 \leq |\mathbf{s}| \leq p, \ \text{for } \hat{\mathbf{u}} \in \mathcal{H}^{1,\mathbf{t}} \text{ for all } \mathbf{t} \text{ with } |\mathbf{t}| \leq |\mathbf{s}|,$$

$$\left\| \hat{\mathbf{v}} - \hat{\Pi}^2 \hat{\mathbf{v}} \right\|_{L^2(Q)^3} \leq C h_{\tilde{Q}}^{|\mathbf{s}|} \, |\hat{\mathbf{v}}|_{\mathcal{H}^{2,|\mathbf{s}|}(\tilde{Q})^3}, \ 0 \leq |\mathbf{s}| \leq p, \ \text{for } \hat{\mathbf{v}} \in \mathcal{H}^{2,\mathbf{t}} \text{ for all } \mathbf{t} \text{ with } |\mathbf{t}| \leq |\mathbf{s}|,$$

$$\left\| \hat{\psi} - \hat{\Pi}^3 \hat{\psi} \right\|_{L^2(Q)} \leq C h_{\tilde{Q}}^{|\mathbf{s}|} \left| \hat{\psi} \right|_{\mathcal{H}^{3,|\mathbf{s}|}(\tilde{Q})}, \ 0 \leq |\mathbf{s}| \leq p, \ \text{for } \hat{\psi} \in \mathcal{H}^{3,\mathbf{t}} \text{ for all } \mathbf{t} \text{ with } |\mathbf{t}| \leq |\mathbf{s}|.$$

*Proof* The result can be found in [8, Prop. 5.6], where a more general estimate is proposed. □

### 5.4.3 Approximation Estimates in the Physical Domain

Finally, we are ready to write the approximation estimates for the projectors defined in the physical domain, that is, for the projectors $\Pi^j, j = 0, 1, 2, 3$ defined in (91).

**Theorem 5.11** *Let Assumptions 3.1 and 4.7 hold. Then, there exists a constant $C$ depending only on the degree $\mathbf{p}$, the parametrization $\mathbf{F}$, and the constant $\theta$ from*

*Assumption 4.7, such that for all elements $K = \mathbf{F}(Q) \in \mathcal{M}$, with $\tilde{K} = \mathbf{F}(\tilde{Q})$, it holds*

$$\begin{aligned}
&\|\phi - \Pi^0\phi\|_{L^2(K)} \le Ch_{\tilde{K}}^s \|\phi\|_{H^s(\tilde{K})}, && 0 \le s \le p+1, && \phi \in H^s(\Omega),\\
&\|\mathbf{u} - \Pi^1\mathbf{u}\|_{L^2(K)^3} \le Ch_{\tilde{K}}^s \|\mathbf{u}\|_{H^s(\tilde{K})^3}, && 0 \le s \le p, && \mathbf{u} \in H^s(\Omega)^3,\\
&\|\mathbf{v} - \Pi^2\mathbf{v}\|_{L^2(K)^3} \le Ch_{\tilde{K}}^s \|\mathbf{v}\|_{H^s(\tilde{K})^3}, && 0 \le s \le p, && \mathbf{v} \in H^s(\Omega)^3,\\
&\|\psi - \Pi^3\psi\|_{L^2(K)} \le Ch_{\tilde{K}}^s \|\psi\|_{H^s(\tilde{K})}, && 0 \le s \le p, && \psi \in H^s(\Omega).
\end{aligned} \tag{95}$$

*Proof* We prove the result for $\Pi^1$, but the reasoning is analogous for all the other cases. Let $Q \in \widehat{\mathcal{M}}$ and $K = \mathbf{F}(Q) \in \mathcal{M}$, and let $\mathbf{u} \in H^s(\Omega)^3$. From Proposition 5.9 and using the definition of the projector $\Pi^1$ in (91), we know that for $\hat{\mathbf{u}} = \iota^1\mathbf{u}$ we have

$$\|\mathbf{u} - \Pi^1\mathbf{u}\|_{L^2(K)^3} \le C\|\hat{\mathbf{u}} - \hat{\Pi}^1\hat{\mathbf{u}}\|_{L^2(Q)^3}.$$

Moreover, since $\mathbf{u} \in H^s(\Omega)^3$ from Proposition 5.9 we also have $\hat{\mathbf{u}} \in \mathcal{H}^{1,\mathbf{t}}$ for any $\mathbf{t} \in \mathbb{N}^3$ such that $|\mathbf{t}| \le s$. Thus we can apply the estimate of Proposition 5.10 with any $\mathbf{s}$ such that $|\mathbf{s}| = s$, and then Proposition 5.9 again, to obtain

$$\|\hat{\mathbf{u}} - \hat{\Pi}^1\hat{\mathbf{u}}\|_{L^2(Q)^3} \le Ch_{\tilde{K}}^s \|\mathbf{u}\|_{H^s(\tilde{K})^3},$$

which ends the proof. □

We finish this section with the estimates in the graph norm of the spaces $X^j$ of the De Rham complex (68). First we need to define the spaces

$$\begin{aligned}
\mathbf{H}^s(\mathrm{curl};\Omega) &:= \{\mathbf{u} \in H^s(\Omega)^3 : \mathrm{curl}\ \mathbf{u} \in H^s(\Omega)^3\},\\
\mathbf{H}^s(\mathrm{div};\Omega) &:= \{\mathbf{v} \in H^s(\Omega)^3 : \mathrm{div}\,\mathbf{v} \in H^s(\Omega)\},
\end{aligned}$$

equipped with the norms

$$\begin{aligned}
\|\mathbf{u}\|^2_{\mathbf{H}^s(\mathrm{curl};\Omega)} &:= \|\mathbf{u}\|^2_{H^s(\Omega)^3} + \|\mathrm{curl}\ \mathbf{u}\|^2_{H^s(\Omega)^3},\\
\|\mathbf{v}\|^2_{\mathbf{H}^s(\mathrm{div};\Omega)} &:= \|\mathbf{v}\|^2_{H^s(\Omega)^3} + \|\mathrm{div}\,\mathbf{v}\|^2_{H^s(\Omega)}.
\end{aligned}$$

Now we can prove the following results.

**Corollary 5.12** *Let Assumptions 3.1 and 4.7 hold. Then, there exists a constant $C$ depending only on $\mathbf{p}$, $\mathbf{F}$, and $\theta$ such that for $0 \le s \le p$ it holds*

$$\begin{aligned}
&\|\phi - \Pi^0\phi\|_{H^1(\Omega)} \le Ch^s \|\phi\|_{H^{s+1}(\Omega)}, && \phi \in H^{s+1}(\Omega),\\
&\|\mathbf{u} - \Pi^1\mathbf{u}\|_{\mathbf{H}(\mathrm{curl};\Omega)} \le Ch^s \|\mathbf{u}\|_{\mathbf{H}^s(\mathrm{curl};\Omega)}, && \mathbf{u} \in \mathbf{H}^s(\mathrm{curl};\Omega),
\end{aligned}$$

$$\|\mathbf{v} - \Pi^2 \mathbf{v}\|_{\mathbf{H}(\mathrm{div};\Omega)} \leq C h^s \|\mathbf{v}\|_{\mathbf{H}^s(\mathrm{div};\Omega)}, \qquad \mathbf{v} \in \mathbf{H}^s(\mathrm{div};\Omega),$$
$$\|\psi - \Pi^3 \psi\|_{L^2(\Omega)} \leq C h^s \|\psi\|_{H^s(\Omega)}, \qquad \psi \in H^s(\Omega).$$

*Proof* As before, we only prove the result for $\Pi^1$, since all the other cases are analogous. The result is a simple consequence of the commutative diagram (92) and the estimates from Theorem 5.11. Indeed, for any element $K \in \mathcal{M}$ we get

$$\begin{aligned}
\|\mathbf{u} - \Pi^1 \mathbf{u}\|_{\mathbf{H}(\mathrm{curl};K)} &= \left( \|\mathbf{u} - \Pi^1 \mathbf{u}\|^2_{L^2(K)^3} + \|\operatorname{curl} \mathbf{u} - \operatorname{curl}(\Pi^1 \mathbf{u})\|^2_{L^2(K)^3} \right)^{1/2} \\
&= \left( \|\mathbf{u} - \Pi^1 \mathbf{u}\|^2_{L^2(K)^3} + \|\operatorname{curl} \mathbf{u} - \Pi^2(\operatorname{curl} \mathbf{u})\|^2_{L^2(K)^3} \right)^{1/2} \\
&\leq C \left( h^{2s}_{\tilde{K}} \|\mathbf{u}\|^2_{H^s(\tilde{K})^3} + h^{2s}_{\tilde{K}} \|\operatorname{curl} \mathbf{u}\|^2_{H^s(\tilde{K})^3} \right)^{1/2} = C h^s_{\tilde{K}} \|\mathbf{u}\|_{\mathbf{H}^s(\mathrm{curl};\tilde{K})}.
\end{aligned}$$

The result follows from a summation on the elements, noting that every element of the mesh belongs to a bounded number of extended supports $\tilde{K}$. □

*Remark 5.13* For simplicity we have limited ourselves to the case of a single patch domain and without boundary conditions. The estimates in a more general case can be found in [8].

## 5.5 *Application to Time Harmonic Maxwell Equations*

In this section we use the spline spaces for vector fields for the discretization of Maxwell equations. For the sake of simplicity, the theoretical discussion is restricted to the single patch isogeometric method, but the numerical tests are performed in a more general setting.

### 5.5.1 Eigenvalue Problem: Cavity Resonator

Given a bounded and simply connected domain $\Omega \subset \mathbb{R}^3$ with connected boundary $\partial\Omega$, we look for a scalar wavenumber $\kappa$ and a non-zero electric field $\mathbf{E}$ such that

$$\begin{aligned}
\operatorname{curl}(\mu_r^{-1} \operatorname{curl} \mathbf{E}) - \kappa^2 \epsilon_r \mathbf{E} &= \mathbf{0} \quad \text{in } \Omega, \\
\mathbf{E} \times \mathbf{n} &= \mathbf{0} \quad \text{on } \partial\Omega,
\end{aligned}$$

where $\mu_r$ and $\epsilon_r$ are the relative magnetic permeability and the relative electric permittivity, respectively. We assume that they are real valued, strictly positive, and piecewise constant functions.

The variational formulation of the problem reads: Find $\mathbf{E} \in \mathbf{H}_0(\mathrm{curl};\Omega)$ and $\kappa \in \mathbb{R}$ such that

$$\int_\Omega \mu_r^{-1} \operatorname{curl} \mathbf{E} \cdot \operatorname{curl} \mathbf{v} \, \mathrm{d}\mathbf{x} = \kappa^2 \int_\Omega \epsilon_r \mathbf{E} \cdot \mathbf{v} \, \mathrm{d}\mathbf{x}, \quad \forall \mathbf{v} \in \mathbf{H}_0(\mathrm{curl};\Omega). \tag{96}$$

The solutions of the eigenvalue problem are closely related to the Helmholtz decomposition:

$$\mathbf{H}_0(\mathrm{curl};\Omega) = \mathbf{Z}_0(\epsilon_r,\Omega) \oplus \operatorname{grad}(H_0^1(\Omega)), \tag{97}$$

with

$$\mathbf{Z}_0(\epsilon_r,\Omega) := \left\{ \mathbf{u} \in \mathbf{H}_0(\mathrm{curl};\Omega) : \int_\Omega \epsilon_r \mathbf{u} \cdot \operatorname{grad} \phi \, \mathrm{d}\mathbf{x} = 0 \quad \forall \phi \in H_0^1(\Omega) \right\}.$$

We have the following result [34, Thm. 4.18].

**Theorem 5.14** *The solutions of the eigenvalue problem (96) have the following properties:*

- *The eigenvalue $\kappa = 0$ is associated to an infinite family of eigenfunctions $\mathbf{E} = \operatorname{grad} \phi$ for any $\phi \in H_0^1(\Omega)$.*
- *There is an infinite set of eigenvalues $0 < \kappa_1 \le \kappa_2 \le \cdots$, with $\lim_{j\to\infty} \kappa_j = \infty$, and corresponding eigenfunctions $\mathbf{0} \neq \mathbf{E}_j \in \mathbf{Z}_0(\epsilon_r,\Omega)$.*

Since the publication of [11], it is known that the approximation of problem (96) with nodal finite elements produces spurious eigenmodes, due to inexact approximations of the infinite family of zero eigenvalues. By contrast, Nédélec edge elements of the first class [36], which are $\mathbf{H}(\mathrm{curl};\Omega)$-conforming, give good approximations of the same problem. The proof relies on the existence of a De Rham diagram with commutative projectors. With similar arguments, in [15] it is proved that a discretization of the eigenvalue problem based on the spaces of the spline complex introduced in Sect. 5.2 is also spurious-free. We summarize here the main results.

Assuming that the domain $\Omega$ is defined by a single patch NURBS parametrization, we consider the discrete space $X_{0,h}^1 = X_h^1 \cap \mathbf{H}_0(\mathrm{curl};\Omega)$, with $X_h^1$ defined as in (72). Then, the variational formulation of the discrete problem is: Find $\mathbf{E}_h \in X_{0,h}^1$ and $\kappa_h \in \mathbb{R}$ such that

$$\int_\Omega \mu_r^{-1} \operatorname{curl} \mathbf{E}_h \cdot \operatorname{curl} \mathbf{v}_h \, \mathrm{d}\mathbf{x} = \kappa_h^2 \int_\Omega \epsilon_r \mathbf{E}_h \cdot \mathbf{v}_h \, \mathrm{d}\mathbf{x}, \quad \forall \mathbf{v}_h \in X_{0,h}^1. \tag{98}$$

Thanks to the commutative diagram (92) (in fact its analogue with boundary conditions), we have for the space $X_{0,h}^1$ a discrete version of the Helmholtz decomposition (97):

$$X_{0,h}^1 = \mathbf{Z}_{0,h}(\epsilon_r,\Omega) \oplus \operatorname{grad}(X_{0,h}^0),$$

with

$$\mathbf{Z}_{0,h}(\epsilon_r, \Omega) := \left\{ \mathbf{u}_h \in X^1_{0,h} : \int_\Omega \epsilon_r \mathbf{u}_h \cdot \operatorname{grad} \phi_h \, \mathrm{d}\mathbf{x} = 0 \quad \forall \phi_h \in X^0_{0,h} \right\},$$

and $X^0_{0,h} = X^0_h \cap H^1_0(\Omega)$, the discrete scalar space with boundary conditions. Moreover, and analogously to the continuous case, the eigenvalue $\kappa_h = 0$ is associated to the finite dimensional space $\operatorname{grad}(X^0_{0,h})$. The rest of the eigenvalues are strictly positive, and their associated eigenfunctions belong to $\mathbf{Z}_{0,h}(\epsilon_r, \Omega)$.

The spectral correctness and the convergence of the solutions of the discrete eigenvalue problem (98) to those of (96) follow from the theory of finite element exterior calculus [4]. We refer to Section 6 in [15] for a detailed proof of the spectral correctness of the method.

### 5.5.2 Numerical Test

We solve the eigenvalue problem in Fichera's corner, with the domain defined as $\Omega = (-1, 1)^3 \setminus [0, 1]^3$ and $\mu_r = \epsilon_r = 1$. Using linear splines, we define the domain as a multi-patch geometry given by seven patches. On each patch we set a tensor product mesh with 12 elements in each direction, refined towards the corner and the reentrant edges with a radical refinement, as in [9], in order to catch the corner and edge singularities (see Fig. 18). We notice that, since the mesh is tensor product, the refinement propagates also away from the singularities. We solve the problem with splines of degree 3, 4 and 5.

Up to our knowledge, reliable benchmark results are not available yet. We compare in Table 1 our results for isogeometric methods with the most accurate ones

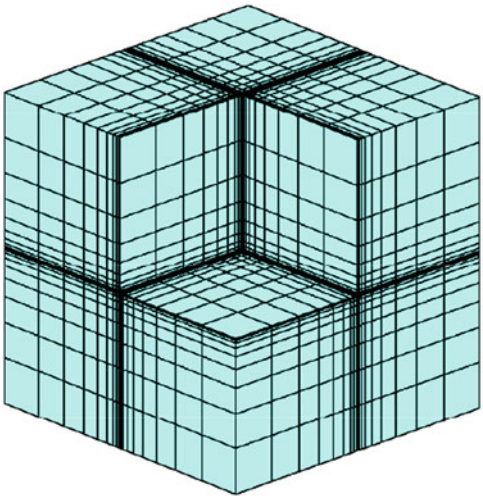

**Fig. 18** Mesh for Fichera's corner

**Table 1** Eigenvalues in Fichera's corner

| Eig. | Duruflé, $p = 5$ | IGA, $p = 3$ | IGA, $p = 4$ | IGA, $p = 5$ |
|---|---|---|---|---|
| 1st | 3.21987401386 | 3.21987496658 | 3.21988066916 | 3.21988316848 |
| 2nd | 5.88041891178 | 5.88041920051 | 5.88041871891 | 5.88041854438 |
| 3rd | 5.88041891780 | 5.88041920051 | 5.88041871897 | 5.88041854605 |
| 4th | 10.6854921311 | 10.6854887693 | 10.6854782775 | 10.6854756735 |
| 5th | 10.6937829409 | 10.6937804726 | 10.6937680695 | 10.6937640480 |
| 6th | 10.6937829737 | 10.6937804726 | 10.6937680701 | 10.6937640484 |
| 7th | 12.3165204656 | 12.3165080267 | 12.3164992881 | 12.3164998491 |
| 8th | 12.3165204669 | 12.3165080267 | 12.3164992882 | 12.3164998498 |
| d.o.f. | 177,720 | 62,412 | 76,365 | 92,256 |

provided by M. Duruflé in the webpage [23], which correspond to a finite element discretization of degree 5. We can observe that the results we obtain with degree 3 are very similar to those obtained with FEM, with around one third of the degrees of freedom. The results obtained with degrees 4 and 5 are even more accurate.

### 5.5.3 Maxwell Source Problem

We now consider the Maxwell source problem. We look for a time-harmonic complex-valued electric field $\mathbf{E}$ corresponding to a given solenoidal current density $\mathbf{J} \in L^2(\Omega)^3$, subject to perfect conductor boundary conditions on $\Gamma_D \subset \partial\Omega$, and an impedance boundary condition on $\Sigma \subset \partial\Omega$. The equations of the problem are

$$\begin{aligned} \operatorname{curl}(\mu_r^{-1} \operatorname{curl} \mathbf{E}) - \kappa^2 \epsilon_r \mathbf{E} &= \mathrm{i}\kappa \mu_0^{1/2} \mathbf{J} && \text{in } \Omega, \\ \mu_r^{-1} (\operatorname{curl} \mathbf{E}) \times \mathbf{n} - \mathrm{i}\kappa\lambda \mathbf{E}_T &= \mathbf{g} && \text{on } \Sigma, \\ \mathbf{E} \times \mathbf{n} &= \mathbf{0} && \text{on } \Gamma_D, \end{aligned}$$

where $\kappa$, $\mu_r$ and $\epsilon_r$ are as in Sect. 5.5.1, $\mu_0$ is the magnetic permeability of free space, $\mathbf{E}_T = (\mathbf{n} \times \mathbf{E}) \times \mathbf{n}$, and the impedance $\lambda$ is a positive function. Following [34, Chap. 4], we define the space

$$X = \left\{ \mathbf{u} \in \mathbf{H}(\operatorname{curl}; \Omega) : \mathbf{u} \times \mathbf{n} = \mathbf{0} \text{ on } \Gamma_D \text{ and } \mathbf{u}_T = (\mathbf{n} \times \mathbf{u}) \times \mathbf{n} \in L^2(\Sigma)^3 \text{ on } \Sigma \right\},$$

equipped with the norm

$$\|\mathbf{u}\|_X^2 = \|\mathbf{u}\|_{\mathbf{H}(\operatorname{curl};\Omega)}^2 + \|\mathbf{u}_T\|_{L^2(\Sigma)^3}^2.$$

Using this space, the variational formulation reads: Find $\mathbf{E} \in X$ such that

$$\int_\Omega \mu_r^{-1} \operatorname{curl} \mathbf{E} \cdot \operatorname{curl} \bar{\mathbf{v}} \, \mathrm{d}\mathbf{x} - \int_\Omega \kappa^2 \epsilon_r \mathbf{E} \cdot \bar{\mathbf{v}} \, \mathrm{d}\mathbf{x} - \int_\Sigma \mathrm{i}\kappa\lambda \mathbf{E}_T \cdot \bar{\mathbf{v}}_T \, \mathrm{d}\Gamma = \int_\Omega \mathrm{i}\kappa\mu_0^{1/2} \mathbf{J} \cdot \bar{\mathbf{v}} \, \mathrm{d}\mathbf{x} + \int_\Sigma \mathbf{g} \cdot \bar{\mathbf{v}} \, \mathrm{d}\Gamma, \quad \forall \mathbf{v} \in X. \tag{99}$$

The well-posedness of the problem follows using the Helmholtz decomposition and Fredholm alternative, see for instance Theorem 4.17 and Corollary 4.19 in [34], and Theorem 5.2. in [27]:

**Theorem 5.15** *Assume either that $\kappa \neq 0$ is such that $\kappa^2$ is not an eigenvalue of (96), or that $\Sigma \neq \emptyset$. Then there exists one unique solution $\mathbf{E}$ to (99) such that*

$$\|\mathbf{E}\|_X \leq C(\|\mathbf{J}\|_{L^2(\Omega)^3} + \|\mathbf{g}\|_{L^2(\Sigma)^3}),$$

*with $C$ independent on the data $\mathbf{J}$ and $\mathbf{g}$.*

In order to discretize the problem with mixed boundary conditions, we first assume that both $\Sigma$ and $\Gamma_D$ are the image through $\mathbf{F}$ of full faces of $\hat{\Omega}$, as in Assumption 3.2. Then we introduce the discrete space

$$X^1_{h,\Gamma_D} = \{\mathbf{u} \in X^1_h : \mathbf{u} \times \mathbf{n} = \mathbf{0} \text{ on } \Gamma_D\},$$

with $X^1_h$ the space defined in (72), and the variational formulation of the discrete problem reads: Find $\mathbf{E}_h \in X^1_{h,\Gamma_D}$ such that

$$\int_\Omega \mu_r^{-1} \operatorname{curl} \mathbf{E}_h \cdot \operatorname{curl} \bar{\mathbf{v}}_h \, \mathrm{d}\mathbf{x} - \int_\Omega \kappa^2 \epsilon_r \mathbf{E}_h \cdot \bar{\mathbf{v}}_h \, \mathrm{d}\mathbf{x} - \int_\Sigma \mathrm{i}\kappa\lambda (\mathbf{E}_h)_T \cdot (\bar{\mathbf{v}}_h)_T \, \mathrm{d}\Gamma = \int_\Omega \mathrm{i}\kappa\mu_0^{1/2} \mathbf{J} \cdot \bar{\mathbf{v}}_h \, \mathrm{d}\mathbf{x} + \int_\Sigma \mathbf{g} \cdot \bar{\mathbf{v}}_h \, \mathrm{d}\Gamma, \quad \forall \mathbf{v}_h \in X^1_{h,\Gamma_D}. \tag{100}$$

Using the commutative diagram and the approximation results of Sect. 5, the well-posedness of the discrete problem and error estimates are given in the following theorem. The proof can be found in [15] and follows the same arguments used for the convergence of finite elements in [27, Sect. 5].

**Theorem 5.16** *Assume either that $\kappa \neq 0$ is such that $\kappa^2$ is not an eigenvalue of (96), or that $\Sigma \neq \emptyset$, then there exists $\bar{h} > 0$ such that for all $h \leq \bar{h}$ problem (100) is well-posed and produces quasi-optimal approximation to (100), that is*

$$\|\mathbf{E} - \mathbf{E}_h\|_X \leq C \inf_{\boldsymbol{w}_h \in X^1_{h,\Gamma_D}} \|\mathbf{E} - \boldsymbol{w}_h\|_X. \tag{101}$$

*Moreover, if* $\mathbf{E}$ *and* curl $\mathbf{E}$ *are in* $(H^s(\Omega))^3$ *for some* $s \leq p$*, then*

$$\|\mathbf{E} - \mathbf{E}_h\|_X \leq Ch^s(\|\mathbf{E}\|_{H^s(\Omega)^3} + \|\operatorname{curl} \mathbf{E}\|_{H^s(\Omega)^3}). \tag{102}$$

### 5.5.4 Numerical Tests

Test 1

The first numerical test consists on the propagation of a plane wave in free space, that is, $\mu_r = \epsilon_r = 1$. Written in polar coordinates, the domain is given by $\Omega = \{\mathbf{x} = (\rho, \theta) : \rho < 10\}$. We assume that $\mathbf{J} = \mathbf{0}$, and on $\Gamma_D = \partial\Omega$ we impose a Dirichlet boundary condition of the form $\mathbf{E} \times \mathbf{n} = \mathbf{E}^i \times \mathbf{n}$, where $\mathbf{E}^i$ is a plane wave given by $\mathbf{E}^i = \mathbf{p}\exp(\mathrm{i}\kappa\mathbf{x} \cdot \mathbf{d})$, with $\kappa = \sqrt{10}$ and the polarization $\mathbf{p} = (0, 1, 0)$ and direction of propagation $\mathbf{d} = (1, 0, 0)$.

We test a NURBS multi-patch parametrization of $\Omega$, represented in Fig. 20a. The problem is discretized using the space $X^1_{h,\Gamma_D}$ in a sequence of successively refined meshes. In Fig. 19 we plot the error in $\mathbf{H}(\mathbf{curl})$-norm for the discretization with degree $p = 3$, and compare the cases of maximum continuity ($C^2$ tangential continuity) and minimum continuity ($C^0$ tangential continuity) within each patch. As in Sect. 3.2.1, the results show a better convergence in terms of the degrees of freedom with higher continuity. Moreover, high continuity splines are faster in reaching the asymptotic regime. This is indeed consistent with the known good behaviour of smooth splines in the approximation of the full spectrum of discrete differential operators (see, e.g., [29]).

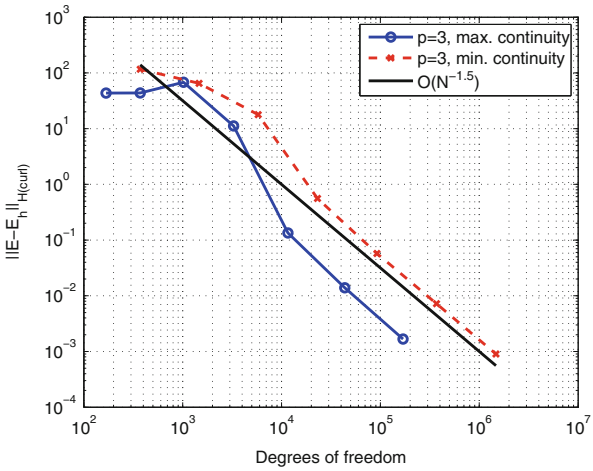


**Fig. 19** Comparison of the error in $\mathbf{H}(\mathbf{curl})$-norm for the approximation of the plane wave

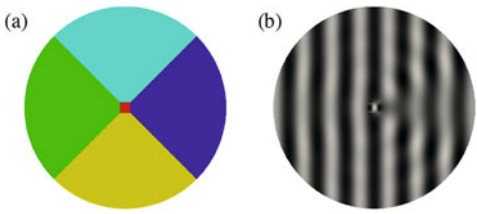


**Fig. 20** Geometry and computed solution of the scattering problem. (**a**) Geometry of the scattering problem using five patches. (**b**) Real part of the $y$-component of $\mathbf{E}_h$

Test 2

For the second numerical test we consider the same domain $\Omega$, and assume that the square region $(-1,1)^2$ is occupied by a dielectric material with $\epsilon_r = 10$ and $\mu_r = 1$. As before, we assume that $\mathbf{J} = \mathbf{0}$, and now we impose an absorbing boundary condition on $\Sigma = \partial\Omega$, which can be written as the impedance condition with $\lambda = 1$ and $\mathbf{g} = (\operatorname{curl}\mathbf{E}^i) \times \mathbf{n} - \mathrm{i}\kappa\mathbf{E}^i_T$, with the incidence field $\mathbf{E}^i$ the same plane wave as in the previous test.

The problem is discretized using the space $X^1_h$ defined with degree $p = 3$ and maximum continuity within each patch, in a mesh formed by 1600 elements on each patch. In Fig. 20b we show the real part of the second component of the computed electric field. We note that only tangential continuity is imposed across the patches, and in particular between the dielectric region and the free space. This allows a more accurate computation at interface regions, since the exact solution is only tangentially continuous due to the jump of the coefficient $\epsilon_r$.

## 6 Conclusions

In this paper we have presented an introduction to isogeometric methods from a mathematical viewpoint, with the focus on the numerical analysis of the method. One of our goals when writing this paper was to provide a didactic reference for those who want to study isogeometric methods for the first time. As a consequence, the paper only contains results that are now well-established, and somehow "classical". Since isogeometric analysis is a relatively new research field, many active and interesting research topics have been left aside of this manuscript. Some of them are: the development of locally refined spaces (using hierarchical splines, T-splines or LR-splines) and adaptive methods; spline spaces defined on triangulations (Powell-Sabin splines); high continuity and approximation properties in geometries with extraordinary points; preconditioners and domain decomposition methods for isogeometric methods; isogeometric boundary element methods.

As it was mentioned in the introduction, the goal of full interoperability between CAD software and an isogeometric PDE solver has not been attained yet, and it is probably one of the most challenging problems in isogeometric methods. Some of the obstacles to reach this goal are related to boolean operations and trimming,

that are widely used in CAD modellers, but without any doubt the main difficulty is that volumetric objects are described in CAD only through a representation of their boundary, while the isogeometric methods require a volumetric parametrization that is not readily available. This is limiting the use of the method to academic problems and applications where the modelled object has a relatively "simple" geometry, which can be represented as the image of several cubes. We remark that the demand for volumetric CAD models is not exclusive to isogeometric methods, and is also shared by the emerging technologies of 3D printing and additive manufacturing, which makes it a strategic research topic.

Despite not being ready for their application at industrial level, isogeometric methods are having a great impact in computational engineering, and are being successfully applied in many different research fields, in particular in computational mechanics, computational fluids and contact problems. Actually, one of the most successful applications of isogeometric methods is the discretization of shell models that require $C^1$ continuity of the trial functions [31], which is not easily attained with classical finite elements, but can be obtained almost for free with splines or NURBS. Since for shell models the variational problem is formulated in the middle surface of the modelled object, the problem of generating a volumetric parametrization is circumvented, and in this case isogeometric methods can work directly on a CAD description of the object.

Rather than giving here a list of references of the topics mentioned above, that is likely to remain incomplete and early outdated, we encourage the interested readers to search for the most recent developments of isogeometric methods in their area of interest.

## References

1. Adams, R.A.: Sobolev Spaces. Pure and Applied Mathematics, vol. 65. Academic, New York/London (1975)
2. Apostolatos, A., Schmidt, R., Wüchner, R., Bletzinger, K.U.: A Nitsche-type formulation and comparison of the most common domain decomposition methods in isogeometric analysis. Int. J. Numer. Methods Eng. **97**(7), 473–504 (2014)
3. Arnold, D.N., Falk, R.S., Winther, R.: Finite element exterior calculus, homological techniques, and applications. Acta Numer. **15**, 1–155 (2006)
4. Arnold, D.N., Falk, R.S., Winther, R.: Finite element exterior calculus: from Hodge theory to numerical stability. Bull. Am. Math. Soc. (N.S.) **47**(2), 281–354 (2010)
5. Babuška, I., Strouboulis, T.: The Finite Element Method and Its Reliability. Numerical Mathematics and Scientific Computation. The Clarendon Press/Oxford University Press, New York (2001)
6. Bazilevs, Y., Beirão da Veiga, L., Cottrell, J.A., Hughes, T.J.R., Sangalli, G.: Isogeometric analysis: approximation, stability and error estimates for *h*-refined meshes. Math. Models Methods Appl. Sci. **16**(7), 1031–1090 (2006)
7. Beirão da Veiga, L., Buffa, A., Rivas, J., Sangalli, G.: Some estimates for *h*-*p*-*k*-refinement in isogeometric analysis. Numer. Math. **118**(2), 271–305 (2011)
8. Beirão da Veiga, L., Buffa, A., Sangalli, G., Vázquez, R.: Mathematical analysis of variational isogeometric methods. Acta Numer. **23**, 157–287 (2014)

9. Beirão da Veiga, L., Cho, D., Sangalli, G.: Anisotropic NURBS approximation in isogeometric analysis. Comput. Methods Appl. Mech. Eng. **209–212**, 1–11 (2012)
10. Boffi, D.: Finite element approximation of eigenvalue problems. Acta Numer. **19**, 1–120 (2010)
11. Boffi, D., Fernandes, P., Gastaldi, L., Perugia, I.: Computational models of electromagnetic resonators: analysis of edge element approximation. SIAM J. Numer. Anal. **36**(4), 1264–1290 (electronic) (1999)
12. de Boor, C.: A Practical Guide to Splines. Applied Mathematical Sciences, vol. 27, revised edn. Springer, New York (2001)
13. Borden, M.J., Scott, M.A., Evans, J.A., Hughes, T.J.R.: Isogeometric finite element data structures based on Bézier extraction of NURBS. Int. J. Numer. Methods Eng. **87**(1–5), 15–47 (2011)
14. Buffa, A., Cho, D., Kumar, M.: Characterization of T-splines with reduced continuity order on T-meshes. Comput. Methods Appl. Mech. Eng. **201–204**, 112–126 (2012)
15. Buffa, A., Rivas, J., Sangalli, G., Vázquez, R.: Isogeometric discrete differential forms in three dimensions. SIAM J. Numer. Anal. **49**(2), 818–844 (2011)
16. Buffa, A., Sangalli, G., Vázquez, R.: Isogeometric methods for computational electromagnetics: B-spline and T-spline discretizations. J. Comput. Phys. **257, Part B**, 1291–1320 (2014)
17. Buffa, A., Vázquez, R., Sangalli, G., Beirão da Veiga, L.: Approximation estimates for isogeometric spaces in multipatch geometries. Numer. Methods Partial Differ. Equ. **31**(2), 422–438 (2015)
18. Cohen, E., Riesenfeld, R., Elber, G.: Geometric Modeling with Splines: An Introduction. AK Peters, Wellesley (2001)
19. Collier, N., Dalcin, L., Pardo, D., Calo, V.M.: The cost of continuity: performance of iterative solvers on isogeometric finite elements. SIAM J. Sci. Comput. **35**(2), A767–A784 (2013)
20. Cottrell, J.A., Hughes, T., Reali, A.: Studies of refinement and continuity in isogeometric structural analysis. Comput. Methods Appl. Mech. Eng. **196**, 4160–4183 (2007)
21. Cottrell, J.A., Hughes, T.J.R., Bazilevs, Y.: Isogeometric Analysis: Toward Integration of CAD and FEA. Wiley, Chichester/Hoboken (2009)
22. Dalcin, L., Collier, N., Vignal, P., Cortes, A., Calo, V.: PetIGA: a framework for high-performance isogeometric analysis (2015). arXiv:1305.4452v3 [cs.MS]
23. Dauge, M.: Benchmark computations for Maxwell equations for the approximation of highly singular solutions. http://perso.univ-rennes1.fr/monique.dauge/benchmax.html (2014)
24. Dörfel, M., Jüttler, B., Simeon, B.: Adaptive isogeometric analysis by local *h*-refinement with T-splines. Comput. Methods Appl. Mech. Eng. **199**(5–8), 264–275 (2010)
25. de Falco, C., Reali, A., Vázquez, R.: GeoPDEs: a research tool for isogeometric analysis of PDEs. Adv. Eng. Softw. **42**(12), 1020–1034 (2011)
26. Govindjee, S., Strain, J., Mitchell, T.J., Taylor, R.L.: Convergence of an efficient local least-squares fitting method for bases with compact support. Comput. Methods Appl. Mech. Eng. **213–216**, 84–92 (2012)
27. Hiptmair, R.: Finite elements in computational electromagnetism. Acta Numer. **11**, 237–339 (2002)
28. Hughes, T.J.R., Cottrell, J.A., Bazilevs, Y.: Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Comput. Methods Appl. Mech. Eng. **194**(39–41), 4135–4195 (2005)
29. Hughes, T.J.R., Reali, A., Sangalli, G.: Duality and unified analysis of discrete approximations in structural dynamics and wave propagation: comparison of *p*-method finite elements with *k*-method NURBS. Comput. Methods Appl. Mech. Eng. **197**(49–50), 4104–4124 (2008)
30. Kellogg, R.B.: On the Poisson equation with intersecting interfaces. Appl. Anal. **4**(2), 101–129 (1974/75)
31. Kiendl, J., Bazilevs, Y., Hsu, M.C., Wüchner, R., Bletzinger, K.U.: The bending strip method for isogeometric analysis of Kirchhoff–Love shell structures comprised of multiple patches. Comput. Methods Appl. Mech. Eng. **199**(37–40), 2403–2416 (2010)
32. Kleiss, S.K., Pechstein, C., Jüttler, B., Tomar, S.: IETI-isogeometric tearing and interconnecting. Comput. Methods Appl. Mech. Eng. **247–248**, 201–215 (2012)

33. Lee, B.G., Lyche, T., Mørken, K.: Some examples of quasi-interpolants constructed from local spline projectors. In: Mathematical Methods for Curves and Surfaces, Oslo, 2000. Innovations in Applied Mathematics, pp. 243–252. Vanderbilt University Press, Nashville (2001)
34. Monk, P.: Finite Element Methods for Maxwell's Equations. Oxford University Press, Oxford (2003)
35. Morin, P., Nochetto, R.H., Siebert, K.G.: Convergence of adaptive finite element methods. SIAM Rev. **44**(4), 631–658 (2002/2003)
36. Nédélec, J.C.: Mixed finite elements in $\mathbb{R}^3$. Numer. Math. **35**, 315–341 (1980)
37. Nguyen, V.P., Bordas, S.P.A., Rabczuk, T.: Isogeometric analysis: an overview and computer implementation aspects (2013). arXiv:1205.2129v2 [cs.NA]
38. Nguyen, V.P., Kerfriden, P., Brino, M., Bordas, S.P.A., Bonisoli, E.: Nitsche's method for two and three dimensional NURBS patch coupling. Comput. Mech. **53**(6), 1163–1182 (2014)
39. Pauletti, M.S., Martinelli, M., Cavallini, N., Antolín, P.: Igatools: an isogeometric analysis library. SIAM J. Sci. Comput. **37**(4), C465–C496 (2015)
40. Petzoldt, M.: Regularity and error estimators for elliptic problems with discontinuous coefficients. Ph.D. thesis, Freie Universität Berlin (2001)
41. Piegl, L., Tiller, W.: The Nurbs Book. Springer, New York (1997)
42. Ratnani, A., Sonnendrücker, E.: An arbitrary high-order spline finite element solver for the time domain Maxwell equations. J. Sci. Comput. **51**, 87–106 (2012)
43. Rogers, D.F.: An Introduction to NURBS: With Historical Perspective. Morgan Kaufmann Publishers Inc., San Francisco (2001)
44. Ruess, M., Schillinger, D., Özcan, A.I., Rank, E.: Weak coupling for isogeometric analysis of non-matching and trimmed multi-patch geometries. Comput. Methods Appl. Mech. Eng. **269**, 46–71 (2014)
45. Sabin, M.A.: Spline finite elements. Ph.D. thesis, Cambridge University (1997)
46. Schumaker, L.L.: Spline Functions: Basic Theory. Cambridge Mathematical Library, 3rd edn. Cambridge University Press, Cambridge (2007)
47. Scott, M.: T-splines as a design-through-analysis technology. Ph.D. thesis, The University of Texas at Austin (2011)
48. Speleers, H., Manni, C., Pelosi, F., Sampoli, M.L.: Isogeometric analysis with Powell-Sabin splines for advection-diffusion-reaction problems. Comput. Methods Appl. Mech. Eng. **221–222**, 132–148 (2012)
49. Takacs, T., Jüttler, B.: Existence of stiffness matrix integrals for singularly parameterized domains in isogeometric analysis. Comput. Methods Appl. Mech. Eng. **200**(49–52), 3568–3582 (2011)
50. Takacs, T., Jüttler, B.: Regularity properties of singular parameterizations in isogeometric analysis. Graph. Models **74**(6), 361–372 (2012)
51. Thomas, D., Scott, M., Evans, J., Tew, K., Evans, E.: Bézier projection: a unified approach for local projection and quadrature-free refinement and coarsening of NURBS and T-splines with particular application to isogeometric design and analysis. Comput. Methods Appl. Mech. Eng. **284**, 55–105 (2015)

# Convolution Quadrature for Wave Simulations

**Matthew Hassell and Francisco-Javier Sayas**

**Abstract** These notes develop the algorithmic aspects of convolution equations and their discretization by Convolution Quadrature, with an emphasis on the convolution equations that occur in the boundary integral equation formulation of wave scattering problems. The authors explore the development of CQ from a number of different perspectives. Clear algorithms for implementation of CQ are presented. A final example brings together the entire course to demonstrate the full discretization of a time domain boundary integral equation using Convolution Quadrature in time and a simple to program Nyström flavored method in space.

**Keywords** Convolution Quadrature • Acoustic waves • Time domain boundary integral equations • Overresolving in the Laplace domain for Convolution Quadrature methods

## 1 Introduction

The following document contains the notes prepared for a course to be delivered by the second author at the *XVI Jacques-Louis Lions Spanish-French School on Numerical Simulation in Physics and Engineering*, in Pamplona (Spain), September 2014. We will not spend much time with the introduction. Let it be said that this is a course on how to approximate causal convolutions and convolution equations, that is, expressions

$$\int_0^t f(t-\tau)g(\tau)\mathrm{d}\tau = h(t),$$

where either $g$ or $h$ is unknown. This seems like a very small problem to be working on when it is presented in this flippant form. The truth is hidden in what the convolution integral means. We will be dealing with operator valued distributions $f$ in convolution with a function valued distribution $g$. A large set of problems

M. Hassell (✉) • F.-J. Sayas
Department of Mathematical Sciences, University of Delaware, Newark, DE, USA
e-mail: mhassell@udel.edu; fjsayas@udel.edu



I. Higueras et al. (eds.), *Numerical Simulation in Physics and Engineering*,
SEMA SIMAI Springer Series 9, DOI 10.1007/978-3-319-32146-2_2

related to the scattering of linear waves (acoustic, elastic, electromagnetic) can be written in this form after being moved to the boundary of the scatterer. We will focus on acoustic waves (we will actually restrict all our attention to a single model problem) and on a particular class of discretization methods, the so-called Convolution Quadrature method, introduced by Christian Lubich in the late 1980s. These notes will emphasize the introduction of concepts and algorithms in a rigorous language, while avoiding proofs. As a matter of fact, we will not state a single theorem explicitly. (This is not due to us not liking theorems, but with the goal of keeping a more narrative tone.) However, clear results will be stated as part of the text. The mathematics of the field of time domain boundary integral equations (which include many important and highly non-trivial examples of convolution equations) involve many interesting and deep analytic concepts, as the reader will be able to ascertain from these notes. The deeper mathematical structure of this field is explored in the lecture notes [29], in a step-by-step traditional mathematical fashion, with not much time for computation. From that point of view, these notes represent the algorithmic counterpart to [29]. Convolution Quadrature is not the only method to approximate convolution equations that appear in wave propagation phenomena. Galerkin and collocation methods compete with CQ in interest, applicability, and good properties. We will not discuss or compare methods, especially because much is still to be explored both in theory and practice. We will not comment on existing literature on CQ for scattering problems on elastic and electromagnetic waves either.
Before we start, let us take some time for acknowledgements. Our research is partially funded by the National Science Foundation (grant DMS 1216356). We now become I. I (FJS) want to thank the organizers of the EHF2014 for the invitation to participate in the school. It is actually my second time in this series (the first one was in Laredo, so many years ago). Since then, the school has made its name even longer by honoring the extraordinary Jacques-Louis Lions, who happens to be my academic great-grandfather. Much of what I know on time domain integral equations and CQ has been a consequence of readings and discussions with Lehel Banjai and Christian Lubich. Both of them are an inspiration for practitioners of serious numerical analysis. Matthew Hassell and I have been working on these notes for several months, trying to give a wide perspective of the mathematical and computational aspects of the problem. We hope the readers will enjoy them as much as we did writing them.

## 2 Causal Convolutions and Laplace Transforms

In this section we introduce what will be the scope of this course: convolution of causal functions and distributions. Causal convolution operators will often be recognized through their Laplace transforms (which will be called their transfer functions). As a first step towards a precise determination of the kind of functions we will be dealing with, let us define the term **causal**. A function $f : \mathbb{R} \to X$ (where $X$ is any vector space) is said to be an $X$-valued causal function when $f(t) = 0$ for all

$t < 0$. The reader might already wonder what the point is to have functions defined in $\mathbb{R}$ when all we want from them is their restriction to $[0, \infty)$. However, this will be a fundamental distinction in our way of presenting wave propagation operators, where the vanishing past of the function will be often used, and where a non-zero value at time $t = 0$ will be considered a jump discontinuity. The attentive reader will have already seen that the independent variable $t$ will often be called time.

We are going to spend this entire section giving a very general justification to causal convolutions and their Laplace transforms:

$$(f * g)(t) = \int_0^t f(t-\tau)g(\tau)\mathrm{d}\tau,$$
$$\mathrm{F}(s) = \int_0^\infty e^{-st} f(t)\mathrm{d}t.$$

## 2.1 Causal Functions and Convolutions

The **causal convolution** of two causal functions $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ is defined as

$$(f * g)(t) := \int_0^t f(t-\tau)g(\tau)\mathrm{d}\tau. \tag{1}$$

This definition makes sense, for instance, if both functions are continuous. It also makes sense if one of the functions is continuous and the other one is integrable. (It can be extended to many other cases, but we will wait for this.) Note that this definition coincides with the more traditional form of the convolution of functions

$$(f * g)(t) = \int_{-\infty}^\infty f(t-\tau)g(\tau)\mathrm{d}\tau,$$

when $f$ and $g$ are causal. The first extension we will need to consider is when $f : \mathbb{R} \to \mathbb{R}^{n\times m}$ and $g : \mathbb{R} \to \mathbb{R}^m$. In this case, (1) defines a causal function $f * g : \mathbb{R} \to \mathbb{R}^n$. In this more general definition (where the convolution integrals are easily defined component by component), it is clear that we cannot even discuss commutativity of the convolution operator (1). The second big generalization involves two Hilbert spaces $X$ and $Y$ and the space $\mathcal{B}(X, Y) := \{A : X \to Y : A \text{ linear and bounded}\}$. We can then start with a causal continuous function $f : \mathbb{R} \to \mathcal{B}(X, Y)$ and a causal function $g : \mathbb{R} \to X$ and obtain through convolution (1) a causal function $f * g : \mathbb{R} \to Y$. Because all functions involved have been assumed to be continuous, the integration in (1) can be easily understood to be defined in

the sense of a Riemann integral for each value of $t$. Finally, a **causal convolution equation** is an equation of the form

$$(f * g)(t) = h(t) \qquad \forall t, \tag{2}$$

where $h : \mathbb{R} \to Y$ is causal, $f : \mathbb{R} \to \mathcal{B}(X, Y)$ is causal, and we look for a causal $X$-valued function $g$.

### 2.1.1 Some Preliminary Examples

The simplest possible example of convolution is the causal antiderivative

$$\int_0^t g(\tau)\mathrm{d}\tau,$$

corresponding to the convolution with the Heaviside function:

$$H(t) := \begin{cases} 1, & t \geq 0, \\ 0, & t < 0. \end{cases}$$

A slightly more general operator is given by the expression

$$\int_0^t e^{\lambda(t-\tau)} g(\tau)\mathrm{d}\tau. \tag{3}$$

Note that if we define

$$y(t) = \int_0^t e^{\lambda(t-\tau)} g(\tau)\mathrm{d}\tau, \tag{4}$$

then $y$ is the only causal solution to the equation $\dot{y} - \lambda\, y = g$, or, in the more traditional language of ordinary differential equations, $y$ satisfies

$$\dot{y} - \lambda\, y = g, \qquad \text{in } [0, \infty), \qquad y(0) = 0, \tag{5}$$

and has been extended by zero to the negative real axis. The formula (4) is the variation of parameters formula (or Duhamel principle) for the initial value problem (5). Equation (5) shows our first use of the dot as the symbol for time-differentiation. Similarly

$$y(t) = \lambda^{-1} \int_0^t \sin(\lambda(t-\tau)) g(\tau)\mathrm{d}\tau$$

is the operator that yields the unique causal solution to $\ddot{y} + \lambda^2 y = g$. The latter example can be easily extended to cover some sort of discrete wave equations. We start with a discrete version of the second derivative in one (implicitly given) space dimension:

$$\mathbb{R}^{N\times N} \ni \Delta_N := \frac{1}{(N+1)^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}.$$

Since $\Delta_N$ is symmetric and it is not terribly complicated to see that it is negative definite, we can find a unitary matrix $D_N$ such that

$$\Delta_N = D_N \begin{bmatrix} -\lambda_1^2 & & & \\ & -\lambda_2^2 & & \\ & & \ddots & \\ & & & -\lambda_N^2 \end{bmatrix} D_N^\top, \qquad \lambda_j > 0 \quad \forall j.$$

We can then define

$$(-\Delta_N)^{1/2} = D_N \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_N \end{bmatrix} D_N^\top,$$

and

$$\sin(t(-\Delta_N)^{1/2}) = D_N \begin{bmatrix} \sin(\lambda_1 t) & & & \\ & \sin(\lambda_2 t) & & \\ & & \ddots & \\ & & & \sin(\lambda_N t) \end{bmatrix} D_N^\top,$$

and finally the vector-valued operator

$$\mathbf{y}(t) := (-\Delta_N)^{-1/2} \int_0^t \sin((t-\tau)(-\Delta_N)^{1/2})\mathbf{g}(\tau)\mathrm{d}\tau,$$

which yields the solution of

$$\ddot{\mathbf{y}} - \Delta_N \mathbf{y} = \mathbf{g} \quad \text{in } [0,\infty), \qquad \mathbf{y}(0) = \dot{\mathbf{y}}(0) = \mathbf{0},$$

extended by zero to negative $t$. This is a discrete version (after using low order finite differences) of the solution operator to the one-dimensional wave equation

$$\begin{aligned} \partial_t^2 y - \partial_x^2 y &= g && \text{in } [0,\infty)\times[0,1],\\ y(\cdot,0) = \partial_t y(\cdot,0) &\equiv 0 && \text{in } [0,1],\\ y(0,\cdot) = y(1,\cdot) &= 0 && \text{in } [0,\infty). \end{aligned}$$

### 2.1.2 A Convolution Equation

One example of causal convolution equation is the Abel integral equation

$$\int_0^t \frac{g(\tau)}{\sqrt{t-\tau}}\mathrm{d}\tau = h(t) \qquad t \geq 0. \tag{6}$$

The weakly singular operator in the left-hand-side of (6) is actually related to the anti-differentiation operator. If we define

$$y(t) = \int_0^t \left( \int_0^r \frac{g(\tau)}{\sqrt{r-\tau}\sqrt{t-r}}\mathrm{d}\tau \right) \mathrm{d}r,$$

then it is easy to see (it requires some patience and the use of the Euler Beta function) that

$$y(t) = \pi \int_0^t g(\tau)\mathrm{d}\tau,$$

that is, the Abel integral operator can be consider as a square root of the antiderivative and, therefore,

$$\frac{1}{\sqrt{\pi}} \int_0^t \frac{\dot{g}(\tau)}{\sqrt{t-\tau}}\mathrm{d}\tau \tag{7}$$

is a square root of the differentiation operator. This is one of the Caputo fractional derivatives. Note that at this time we cannot yet understand the operator in (7) as a convolution operator.

### 2.1.3 A Much More Complicated Example

Just to give a better flavor of operators to come, let us show one related to propagation of linear waves in the plane. Let $X = Y = L^2(\Gamma)$, where $\Gamma$ is a simple

closed curve in the plane. Consider now the operator

$$(\mathcal{V}(t)\xi)(\mathbf{x}) := \frac{1}{2\pi}\int_\Gamma \frac{H(t-|\mathbf{x}-\mathbf{y}|)}{\sqrt{t^2-|\mathbf{x}-\mathbf{y}|^2}}\xi(\mathbf{y})\mathrm{d}\Gamma(\mathbf{y}),$$

where $H$ is the Heaviside function. For given $t$, this is a well defined operator $X \to Y$. We can then define the convolution of $\mathcal{V}$ with a function $g : \mathbb{R} \to L^2(\Gamma)$ (which can better be understood as a function $g(\mathbf{y}, t)$ with $\mathbf{y} \in \Gamma$ such that $g(\cdot, t) \equiv 0$ for $t < 0$), leading to the expression

$$\frac{1}{2\pi}\int_\Gamma \left(\int_0^{t-|\mathbf{x}-\mathbf{y}|} \frac{g(\mathbf{y},\tau)}{\sqrt{(t-\tau)^2-|\mathbf{x}-\mathbf{y}|^2}}\mathrm{d}\tau\right)\mathrm{d}\Gamma(\mathbf{y}).$$

## 2.2 Causal Distributions

It is customary to keep on extending the definitions of the previous subsection to 'functions' $f$ that include Dirac deltas or their derivatives. These extensions can be carried out with better or worse justified limiting processes. Instead of doing that, we will go the whole nine yards and deal with some *very elementary concepts of function- and operator-valued distributions.*

**On notation** We will always keep in mind some Hilbert spaces $X, Y, Z, \ldots$ and the spaces of bounded linear operators between pairs of them $\mathcal{B}(X, Y)$, ... When we want to refer at the same time to the Hilbert spaces or to the spaces of operators, we will just refer to a general Banach space $\mathbb{X}$.

### 2.2.1 The Test Space

We consider the space of smooth compactly supported functions

$$\mathcal{D}(\mathbb{R}) := \{\psi \in C^\infty(\mathbb{R}) \,:\, \psi \equiv 0 \text{ outside } [-M, M] \text{ for some } M\}.$$

We will not need a precise definition of support, but here it is just in case

$$\operatorname{supp}\psi = \text{closure of } \{t \,:\, \psi(t) \neq 0\}.$$

A sequence $\{\psi_n\} \subset \mathcal{D}(\mathbb{R})$ is said to converge to $\psi \in \mathcal{D}(\mathbb{R})$ when the support of all the elements of the sequence and of the limiting function is contained in a bounded interval $[-M, M]$, and for all $m \geq 0$, $\psi_n^{(m)} \to \psi^{(m)}$ uniformly in $\mathbb{R}$.

### 2.2.2 Causal Vector-Valued Distributions

A distribution with values in a Banach space $\mathbb{X}$ is a functional $f : \mathcal{D}(\mathbb{R}) \to \mathbb{X}$ that is sequentially continuous, that is, such that it transforms convergent sequences to convergent sequences. An $\mathbb{X}$-valued distribution $f$ such that

$$\langle f, \psi \rangle = 0 \qquad \text{whenever supp } \psi \subset (-\infty, 0)$$

is called a causal distribution. Note how we have used the angled bracket for the action of the distribution $f$ on the test function $\psi$. The simplest example of causal distributions are causal functions. If $f : \mathbb{R} \to \mathbb{X}$ is a continuous function, then we can define

$$\langle f, \psi \rangle := \int_0^\infty f(t)\psi(t)\mathrm{d}t,$$

with the integral defined in the sense of a Riemann integral, or, even simpler, as a limit

$$\int_0^\infty f(t)\psi(t)\mathrm{d}t = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{\infty} f(n/N)\psi(n/N).$$

Then $f$ defines a causal distribution. If $x \in \mathbb{X}$, we can define

$$\langle x \otimes \delta_0, \psi \rangle := \psi(0)\, x,$$

which is a causal $\mathbb{X}$-valued distribution. More generally, if $t_0 \geq 0$, then

$$\langle x \otimes \delta_{t_0}, \psi \rangle := \psi(t_0)x$$

is a causal distribution. If we take $t_0 < 0$, we still have a distribution, but it is not causal anymore.

### 2.2.3 Steady State Operators

Once we have a causal distribution $f$ with values in the space $X$, any bounded linear operator $A : X \to Y$ allows us to define the distribution

$$\langle Af, \psi \rangle := A\langle f, \psi \rangle,$$

with values in $Y$. When we make a linear operator $A : X \to Y$ act on $X$-valued causal distributions, we will say that we have used a **steady state** operator. In particular, if $X \subset Y$ with bounded inclusion, then every $X$-valued distribution can be understood as a $Y$-valued distribution.

### 2.2.4 Differentiation

It is very simple to see why if $\psi_n$ converges to $\psi$ in $\mathcal{D}(\mathbb{R})$, then $\dot{\psi}_n$ converges to $\dot{\psi}$ in $\mathcal{D}(\mathbb{R})$. Therefore, if $f$ is a causal $X$-valued distribution, then

$$\langle \dot{f}, \psi \rangle := -\langle f, \dot{\psi} \rangle$$

is a causal $X$-valued distribution too. For instance, consider $x \in X$ and let $H$ be the Heaviside function. The derivative of

$$\langle x \otimes H, \psi \rangle = \left( \int_0^\infty \psi(t) \mathrm{d}t \right) x$$

is

$$-\langle x \otimes H, \dot{\psi} \rangle = \psi(0)\, x = \langle x \otimes \delta_0, \psi \rangle.$$

The derivative of $x \otimes \delta_0$ is $\langle x \otimes \dot{\delta}_0, \psi \rangle = -\dot{\psi}(0)x$.

## 2.3 Laplace Transforms

### 2.3.1 Why?

Let us first go back to Sect. 2.1. Let us admit that all the following formal manipulations actually make sense:

$$\begin{aligned}
\int_0^\infty e^{-st}(f * g)(t) \mathrm{d}t &= \int_0^\infty e^{-st} \left( \int_0^t f(t-\tau) g(\tau) \mathrm{d}\tau \right) \mathrm{d}t \\
&= \int_0^\infty \left( \int_\tau^\infty e^{-st} f(t-\tau) g(\tau) \mathrm{d}t \right) \mathrm{d}\tau \\
&= \int_0^\infty e^{-s\tau} \left( \int_\tau^\infty e^{-s(t-\tau)} f(t-\tau) \mathrm{d}t \right) g(\tau) \mathrm{d}\tau \\
&= \left( \int_0^\infty e^{-st} f(t) \mathrm{d}t \right) \left( \int_0^\infty e^{-st} g(t) \mathrm{d}t \right).
\end{aligned}$$

This computation shows how the **Laplace transform**

$$h \mapsto \int_0^\infty e^{-st} h(t) \mathrm{d}t$$

maps convolutions to 'products'. Let us try to first give a precise meaning to the Laplace transform.

### 2.3.2 The Schwartz Class

The expression

$$\mathcal{S}(\mathbb{R}) := \{\psi \in C^\infty(\mathbb{R}) \,:\, p_m \tfrac{\mathrm{d}}{\mathrm{d}t^m}\psi \in L^\infty(\mathbb{R}) \quad \forall m \geq 0\}, \qquad p_m(t) := 1 + t^{2m},$$

gives an abbreviated definition of the set of all smooth functions whose derivatives of all orders can be bounded by non-vanishing rational functions of all possible decays at infinity. This set obviously contains $\mathcal{D}(\mathbb{R})$. Convergence in $\mathcal{S}(\mathbb{R})$ is defined as follows: the sequence $\{\psi_n\} \subset \mathcal{S}(\mathbb{R})$ converges to $\psi \in \mathcal{S}(\mathbb{R})$, when for all $m \geq 0$, $p_m\, \psi_n^{(m)} \to p_m\, \psi^{(m)}$ uniformly in $\mathbb{R}$. Using some simple cut-off arguments, it is possible to show that every element of $\mathcal{S}(\mathbb{R})$ is the limit of a sequence of elements of $\mathcal{D}(\mathbb{R})$, that is, $\mathcal{D}(\mathbb{R})$ is dense in the Schwartz class.

### 2.3.3 A Careful Construction of the Laplace Transform

We start with a smooth version of the Heaviside function $H$. Consider a function $h : \mathbb{R} \to \mathbb{R}$ with the following properties:

$$h \in C^\infty(\mathbb{R}), \qquad 0 \leq h \leq 1, \qquad h \equiv 1 \text{ in } [-\tfrac{1}{2}, \infty), \qquad h \equiv 0 \text{ in } (-\infty, -1]. \tag{8}$$

This function can be easily constructed using the antiderivative of a positive smooth compactly supported function, conveniently displaced and scaled. Let now

$$s \in \mathbb{C}_+ := \{s \in \mathbb{C} \,:\, \operatorname{Re} s > 0\}.$$

Then, the function

$$\psi_s(t) := h(t)e^{-st} \tag{9}$$

is an element of the Schwartz class. Therefore, there exists a sequence $\{\psi_{n,s}\} \subset \mathcal{D}(\mathbb{R})$ converging to $\psi_s$ in $\mathcal{S}(\mathbb{R})$. We will say that the causal $\mathbb{X}$-valued distribution $f$ has a Laplace transform when $\lim_{n\to\infty}\langle f, \psi_{n,s}\rangle$ exists (in $\mathbb{X}$) for all $s \in \mathbb{C}_+$. We then define the function $\mathrm{F} : \mathbb{C}_+ \to \mathbb{X}$ given by the limit

$$\mathrm{F}(s) := \lim_{n\to\infty} \langle f, \psi_{n,s}\rangle. \tag{10}$$

It is common to use all the following expressions:

$$\mathrm{F}(s) = \mathcal{L}\{f\}(s) = \langle f, \psi_s\rangle = \langle f, e^{-s\cdot}\rangle.$$

In the last one we act on the understanding that because $f$ is causal, the function $h$ is actually invisible. Actually, if we fix $s$ but define any other $h$ with the same

properties, the difference between the corresponding $\psi_s$ is an element of $\mathcal{D}(\mathbb{R})$ with support in $[-1, -\frac{1}{2}]$ and thus completely invisible for every causal distribution. In other words, the definition (10) is independent of the particular choice of $h$, as long as this smooth Heaviside function satisfies (8) or a similar expression with $[-1, -\frac{1}{2}]$ substituted by any closed interval contained in the negative real axis.

### 2.3.4 Some Examples

The Laplace transform of $x \otimes \delta_{t_0}$ is

$$\langle x \otimes \delta_{t_0}, e^{-s\cdot} \rangle = \langle \delta_{t_0}, e^{-s\cdot} \rangle\, x = e^{-st_0} x.$$

The Laplace transform of $x \otimes H$ is

$$\langle x \otimes H, e^{-s\cdot} \rangle = \left( \int_0^\infty e^{-st} \mathrm{d}t \right) x = \frac{1}{s} x.$$

If the Laplace transform of an $X$-valued distribution $f$ exists and $A : X \to Y$ is a steady-state operator (bounded linear), then the Laplace transform of $Af$ is

$$\mathcal{L}\{Af\}(s) = \langle Af, e^{-s\cdot} \rangle = A\langle f, e^{-s\cdot} \rangle = A\mathrm{F}(s).$$

### 2.3.5 Laplace Transform and Differentiation

A simple computation shows that $\dot{\psi}_s = -s\psi_s + \varphi_s$, where $\varphi_s \in \mathcal{D}(\mathbb{R})$ is supported in $(-\infty, 0)$. Therefore, if $f$ has a Laplace transform, then $\dot{f}$ has a Laplace transform and

$$\mathcal{L}\{\dot{f}\} = \langle \dot{f}, \psi_s \rangle = -\langle f, -s\psi_s + \varphi_s \rangle = \langle f, s\psi_s \rangle = s\mathrm{F}(s),$$

since $\langle f, \varphi_s \rangle = 0$, due to the causality of $f$. This is the differentiation theorem.

### 2.3.6 A Remark

Readers used to classical Laplace transforms might be missing the value of $f$ at $t = 0$ subtracted from the right-hand-side. It is missing in our formulation, because the derivative $\dot{f}$ includes also a Dirac delta at $t = 0$ caused by a non-zero $f(0)$. More precisely: assume that $f$ is a fastly decaying smooth function $f : [0, \infty) \to \mathbb{X}$, extended by zero to the negative real axis. Let $f' : [0, \infty) \to \mathbb{X}$ be its classical derivative, which we also assume to be fastly decaying. Then

$$\dot{f} = f(0) \otimes \delta_0 + f',$$

that is, the distributional derivative contains a functional part $f'$ and a Dirac delta term related to the initial value of $f$. Then

$$\mathcal{L}\{f'\}(s) = \mathcal{L}\{\dot{f}\}(s) - \mathcal{L}\{f(0) \otimes \delta_0\} = s\mathrm{F}(s) - f(0),$$

which is the formula we typically learn in an introduction to ODE class. The lesson to learn here is to be exactly aware of what we understand by the derivative. In our case, the derivative will always be defined for a causal function, and will include anything happening at $t = 0$.

### 2.3.7 Warning

The reader might think that we are developing quite a general theory. This is however not the case. The Laplace transform can sometimes be defined in a half plane of the complex plane $\mathbb{C}$, larger or smaller than $\mathbb{C}_+$. It is sometimes possible to extend it to the entire complex plane or to a larger region of the complex plane. The kind of operators that we want to focus on (arising from hyperbolic evolutionary equations) does not require to go any further than what we have done here. Parabolic equations produce transforms that are defined in the complementary set of a sector in the negative complex half plane. The theory of Laplace transforms is often defined for non-causal distributions, leading to double-sided Laplace transforms, which end up being analytic extensions of Fourier transforms. As already mentioned, we will try to keep the theory quite close to our interests.

### 2.3.8 Analiticity

One of the nicest surprises of advanced analysis is the fact that many results in the theory of analytic functions of a complex variable can be extended almost word by word to analytic functions of a complex variable with values in a Banach space $\mathbb{X}$. In particular, it is quite easy to prove that if $f$ is causal and Laplace transformable, with values in $\mathbb{X}$, then the function $\mathbb{C}_+ \ni s \longmapsto \mathrm{F}(s) \in \mathbb{X}$ is differentiable in $s$ and therefore it is an analytic $\mathbb{X}$-valued function.

### 2.3.9 Real or Complex Spaces?

There is some fine detail missing in our treatment of the definition of the Laplace transform. We have implicitly assumed that $\mathcal{D}(\mathbb{R})$ and $\mathcal{S}(\mathbb{R})$ are spaces of real-valued functions. In this case, $\psi_s = h\exp(-s\cdot)$ is complex-valued, and we should be careful in defining

$$\mathrm{F}(s) = \langle f, \psi_s \rangle := \langle f, \mathrm{Re}\,\psi_s \rangle + \mathrm{i}\langle f, \mathrm{Im}\psi_s \rangle.$$

For the kind of applications we have in mind $\mathbb{X}$ is a real Banach space and $\mathrm{F}(s)$ takes values in the complexification $\mathbb{X} + \mathrm{i}\mathbb{X}$. (There is no problem when $\mathbb{X}$ is a complex space though.) A reader who is not comfortable with the idea of complexification should think of

$$L^2(\Omega) := \left\{ f : \Omega \to \mathbb{R} \,:\, \int_\Omega |f|^2 < \infty \right\} \quad \text{becoming} \quad \left\{ f : \Omega \to \mathbb{C} \,:\, \int_\Omega |f|^2 < \infty \right\}.$$

Here is an important consequence that we will exploit in Sect. 4: if $\mathbb{X}$ is a real space, then its complexification is a complex vector space that admits a conjugation operator; in this case

$$\mathrm{F}(\overline{s}) = \overline{\mathrm{F}(s)}. \tag{11}$$

## 2.4 Transfer Functions and Convolution Operators

### 2.4.1 A Very Weak Definition of Convolution

Let $f$ be a $\mathcal{B}(X, Y)$-valued causal Laplace transformable distribution and let $g$ be an $X$-valued causal Laplace transformable distribution. The convolution $f * g$ is defined to be the $Y$-valued causal distribution whose Laplace transform satisfies

$$\mathcal{L}\{f * g\}(s) = \mathrm{F}(s)\mathrm{G}(s). \tag{12}$$

The right-hand-side of (12) is the action of the operator $\mathrm{F}(s) \in \mathcal{B}(X, Y)$ on $\mathrm{G}(s) \in X$. The justification of how this formula makes sense follows from a theorem for the inversion of the Laplace transform given at the end of this section. For the moment being, we will accept this as a definition. A similar argument can be invoked to substitute the $X$-valued distribution $g$ by a $\mathcal{B}(Z, X)$-valued distribution. In this case, the convolution defines a new distribution which is again operator-valued, this time in $\mathcal{B}(Z, Y)$.

### 2.4.2 Examples

The first new case of a convolution operator is differentiation:

$$\dot{\delta}_0 * f = \dot{f} \qquad \longleftrightarrow \qquad s\mathrm{F}(s) = \mathcal{L}\{\dot{f}\}.$$

This is also valid when $f$ takes values on $X$, by convoluting with

$$\tfrac{\mathrm{d}}{\mathrm{d}t}(I_X \otimes \delta_0) = I_X \otimes \dot{\delta}_0,$$

where $I_X$ is the identity operator in $X$. This is one of the few cases where we can talk about commutativity: if $f$ takes values in $B(X, Y)$, we can write

$$(I_Y \otimes \dot{\delta}_0) * f = \dot{f} = f * (I_X \otimes \dot{\delta}_0),$$

which is the time domain form of $(sI_Y)\mathrm{F}(s) = s\mathrm{F}(s) = \mathrm{F}(s)(sI_X)$. Delays are also causal convolution operators

$$(I \otimes \delta_{t_0}) * f = f(\cdot - t_0) \qquad \longleftrightarrow \qquad e^{-st_0}\mathrm{F}(s).$$

In this case either $f$ is $X$-valued and $I$ is the identity in $X$, or $f$ is $\mathcal{B}(X, Y)$-valued and $I$ is the identity in $Y$. We can then combine derivatives and delays in the form of an equation: given a causal function $g$, we look for a causal function $y$ satisfying

$$\dot{y} = g - g(\cdot - t_1), \qquad \text{where } t_1 > 0.$$

This is then equivalent to

$$\mathrm{Y}(s) = \frac{1 - e^{-st_1}}{s}\mathrm{G}(s).$$

The inversion theory will allow us to see how the solution operator $g \mapsto y$ is also a convolution operator.

### 2.4.3 Terminology

If $f$ is a $\mathcal{B}(X, Y)$-valued causal Laplace transformable distribution, its Laplace transform $\mathrm{F} : \mathbb{C}_+ \to \mathcal{B}(X, Y)$ is often referred to as its **transfer function**. In more analytic contexts, the transfer function is also called the **symbol** of the operator.

### 2.4.4 Inversion Theory

There is an unfortunate fact in the theory of Laplace transforms, stemming from the fact that few classical smoothness properties of a distribution (the distribution equals a function which is continuous or smoother) can be directly seen in their Laplace transforms. However, by restricting the set of distributions and their Laplace transforms, we can find two sets that correspond one to one. At the distributional sense, the set is:

All causal continuous functions $f : \mathbb{R} \to \mathbb{X}$ with polynomial growth as $t \to \infty$ and their distributional derivatives.

Polynomial growth of $f$ means that $\|f(t)\|$ can be bounded by a polynomial in the variable $t$. The set of all Laplace transforms of these distribution can be characterized as follows:

All analytic functions $\mathrm{F} : \mathbb{C}_+ \to \mathbb{X}$ such that

$$\|\mathrm{F}(s)\| \leq C(\operatorname{Re} s)|s|^{\mu} \qquad \forall s \in \mathbb{C}_+, \tag{13}$$

where $\mu \in \mathbb{R}$ and $C : (0, \infty) \to (0, \infty)$ is a non-increasing function such that

$$\sigma^m C(\sigma) \leq C_0, \qquad \forall \sigma \in (0, 1]. \tag{14}$$

### 2.4.5 Convolution Justified

If $f$ is a $\mathcal{B}(X, Y)$-valued distribution whose Laplace transform satisfies properties (13)–(14), and $g$ is an $X$-valued distribution whose Laplace transform satisfies the same type of bound (with different $\mu$ and $C$), from where it is clear that $\mathrm{F}(s)\mathrm{G}(s)$ is also the Laplace transform of a causal distribution, which is what we call $f * g$.

### 2.4.6 A Final Complicated Example

For this example we need the Sobolev space $H_0^1(\Omega)$, which can be defined as the closure of the set of $C^\infty(\Omega)$ functions that vanish in a neighborhood of $\partial\Omega$, with respect to the Sobolev norm

$$\|u\|_{1,\Omega}^2 := \|\nabla u\|_\Omega^2 + \|u\|_\Omega^2 = \int_\Omega |\nabla u|^2 + \int_\Omega |u|^2.$$

We now start with $g \in L^2(\Omega)$ and look for

$$u \in H_0^1(\Omega) \qquad \text{s.t.} \qquad (\nabla u, \nabla v)_\Omega + s^2(u, v)_\Omega = (g, v)_\Omega \quad \forall v \in H_0^1(\Omega). \tag{15}$$

With some help of the Lax-Milgram lemma, it is not complicated to see that the solution operator $L^2(\Omega) \ni g \mapsto u \in H_0^1(\Omega)$ is well defined and satisfies

$$\|u\|_{1,\Omega} \leq C(\operatorname{Re} s)\|g\|_\Omega, \qquad C(\sigma) := \frac{1}{\sigma \min\{1, \sigma\}}.$$

It is also easy to see how the solution operator is an analytic function of the parameter $s \in \mathbb{C}_+$. Therefore, there exists a $\mathcal{B}(L^2(\Omega), H_0^1(\Omega))$-valued causal distribution $f$ whose Laplace transform is the operator $\mathrm{F}(s)$ that solves problem (15). We keep the same letter $g$ for a causal function $g : \mathbb{R} \to L^2(\Omega)$ with polynomial growth. Then $\mathrm{F}(s)\mathrm{G}(s)$ is analytic in $\mathbb{C}_+$, with values in $H_0^1(\Omega)$. It is also the Laplace transform of a causal $H_0^1(\Omega)$-valued distribution $u = f * g$. This is the very weak form of the problem looking for a causal $H_0^1(\Omega)$-valued distribution $u$ such that

$$(\nabla u, \nabla v)_\Omega + (\ddot{u}, v)_\Omega = (g, v)_\Omega \qquad \forall v \in H_0^1(\Omega).$$

If $u$ were a smooth function of time (it might not be depending on how smooth $g$ is), then we could characterize $u : [0, \infty) \to H^1_0(\Omega)$ in a stronger form

$$(\nabla u(t), \nabla v)_\Omega + (\ddot{u}(t), v)_\Omega = (g(t), v)_\Omega \qquad \forall v \in H^1_0(\Omega), \qquad t \geq 0,$$

with vanishing initial conditions $u(0) = 0$ and $\dot{u}(0) = 0$. Readers acquainted with basic Sobolev space theory (the one used for the most elementary elliptic equations) will recognize a weak form of the wave equation $\partial_{tt} u = \Delta u + g$ with homogeneous Dirichlet boundary conditions and homogeneous initial condition. The moral of the story is that the solution operator for this problem is a causal convolutional operator.

### *2.5 Credits*

Laplace transforms of (scalar) distributions are an important part of the original theory designed by Laurent Schwartz. A readable, while quite general, introduction is given in [32, Chapter 6]. Note that the general case allows for the Laplace transform to be defined in any right semiplane of $\mathbb{C}$ (we only pay attention here to the one with positive real part). The problem of defining convolutions of distributions is also a classic in modern analysis [32, Chapter 3], especially because many different pairs of distributions can be convoluted, but not every pair. However, the causal case is much simpler, since any pair of causal distributions can be put in convolution. There are no easy references to learn vector-valued distributions. A very comprehensive treatment is given in [33], and a much more concise introduction can be found in [13]. For an introduction tailored to our needs, the reader is referred to [29]. The inversion theorem for the Laplace transform is part of the general theory, but the class of Laplace transforms that we present here, as well as their time-domain representatives, is inherited from [29], as a further refinement of a class introduced in [22]. This class of symbols is the one that appears systematically in the treatment of exterior problems for the wave equation [22].

## 3 Multistep Convolution Quadrature

The goal of this section is the presentation and justification of some discrete quadrature approximations of convolutions and convolution equations

$$y(t) = \int_0^t f(\tau) g(t-\tau) \mathrm{d}\tau, \qquad \int_0^t f(\tau) g(t-\tau) \mathrm{d}\tau = h(t).$$

The discretization will be developed on a uniform grid of time-step $\kappa > 0$

$$t_n := n\kappa \quad n \geq 0.$$

Data will always be dealt with in the time domain. This will lead to discrete convolutions

$$y(t_n) \approx y_n := \sum_{m=0}^{n} \omega_m^{\mathrm{F}}(\kappa) g(t_{n-m}) \qquad n \geq 0,$$

and discrete convolution equations

$$\sum_{m=0}^{n} \omega_m^{\mathrm{F}}(\kappa) g_{n-m} = h(t_n) \qquad n \geq 0.$$

## *3.1 A Backward Differentiation Approach*

### 3.1.1 A Very Simple Model Problem

Given a causal function $g : \mathbb{R} \to \mathbb{R}$ and $c > 0$, we look for causal $y$ such that

$$\dot{y} - cy = g. \tag{16}$$

The causality of $y$ means exactly that $y(t) = 0$ for $t < 0$. If we want to impose an initial condition $y(0) = y_0$, the way to go is to incorporate it to the right-hand-side

$$\dot{y} - cy = g + y_0 \delta_0.$$

We will not deal with this case, since we will insist on the right-hand-side of (16). The reader might think it is quite a stupid notion to deal only with homogeneous initial conditions. (It is also true that (16) is a linear equation for with the exact solution can be given a closed formula.) The emphasis of this exposition is in picking up a very simple example to try to understand what we will do for more complicated problems where time-stepping strategies are far from obvious.

### 3.1.2 Backward Euler Differentiation

The starting point for applying the Backward Euler (BE) method to (16) is the simple discrete differentiation formula:

$$\dot{y}(t_n) \approx \tfrac{1}{\kappa}(y(t_n) - y(t_{n-1})). \tag{17}$$

We can then define the BE approximation to (16) by

$$\tfrac{1}{\kappa}(y_n - y_{n-1}) - cy_n = g(t_n). \tag{18}$$

We want the sequence $\{y_n\}$ to be causal, which means that $y_n = 0$ for $n < 0$ (but not for $n = 0$). Since the discrete differentiation formula (17) only uses one point in the past, we will only need to set $y_{-1} = 0$. In this case we will have that $g(0) = g(t_0) = 0$ implies $y_0 = 0$. If $g(0) \neq 0$, then the derivative of the solution of (16) needs to jump at $t = 0$ and therefore (17) is not a very good approximation of a quantity that just does not exist at the point $t_0$. Once again, this is not our concern, because the kind of problems we will be dealing with start smoothly from zero. We can write (18) in a more explicit form

$$y_n = \frac{1}{1-\kappa c} y_{n-1} + \frac{\kappa}{1-\kappa c} g(t_n),$$

and work backwards by induction until we reach $n = 0$ to get

$$y_n = \kappa \sum_{m=0}^{n} \frac{1}{(1-\kappa c)^{m+1}} g(t_{n-m}). \tag{19}$$

The proper BE method would stop the sum at $m = 1$ and never use the value $g(0)$, i.e., (19) is the proper BE method only when $g(0) = 0$.

### 3.1.3 Remark

The expression (19) is somewhat worrying for $1 - \kappa c$ might be zero. In principle one could always think of taking $\kappa$ small enough to ensure that (19) makes sense. (Actually it only fails if $\kappa c = 1$.) But you might keep on wondering, were we not using an implicit method? How come this might not work? The reason is actually that when we think of implicit methods and stability issues, we automatically assume $c < 0$ (we look for stable approximations of stable problems!), so there is nothing to worry about. We need this form for some future computations.

### 3.1.4 A More Convolutional Point of View

Let us remind you that the causal solution of (16) is given by the convolution expression

$$y(t) = \int_0^t e^{c\tau} g(t-\tau) \mathrm{d}\tau.$$

We focus on the point $t_n$ and write

$$y(t_n) = \sum_{m=0}^{n} \int_{t_{m-1}}^{t_m} e^{c\tau} g(t_n - \tau) \mathrm{d}\tau. \tag{20}$$

(The reader should not be too concerned with us including the interval $(t_{-1}, t_0)$, where everything should be zero. This is part of the method and we have already warned that when $g(0) \neq 0$ the method should be modified.) The BE method makes the slightly strange approximation

$$\int_{t_{m-1}}^{t_m} e^{c\tau} g(t_n - \tau) \mathrm{d}\tau \approx \frac{\kappa}{(1 - \kappa c)^{m+1}} g(t_n - t_m),$$

which we will not try to justify.

### 3.1.5 Challenges We Will Not Accept

We can also think of our very primitive wave equation $\ddot{y} + c^2 y = g$, when we approximate

$$\ddot{y}(t_n) = \frac{1}{\kappa^2}(y(t_n) - 2y(t_{n-1}) + y(t_{n-2})),$$

which is the first order approximation of the second derivative corresponding to (17). The associated discrete method is

$$\frac{1}{\kappa^2}(y_n - 2y_{n-1} + y_{n-2}) + c^2 y_n = g(t_n),$$

starting with causal conditions $y_{-2} = y_{-1} = 0$. While it is possible to find a formula

$$y_n = \kappa^2 \sum_{m=0}^{n} \alpha_m(c\kappa) g(t_{n-m}), \tag{21}$$

we will not waste our time trying to figure out these coefficients. What is important is the fact that (21) approximates

$$y(t_n) = c^{-1} \int_0^{t_n} \sin(c\tau) g(t_n - \tau) \mathrm{d}\tau.$$

If we go back to (16) but now approximate the derivative using a double backward differentiation formula

$$\dot{y}(t_n) \approx \tfrac{1}{\kappa}(\tfrac{3}{2} y(t_n) - 2y(t_{n-1}) + \tfrac{1}{2} y(t_{n-2})),$$

we end up with the BDF2 method

$$\tfrac{1}{\kappa}(\tfrac{3}{2} y_n - 2y_{n-1} + \tfrac{1}{2} y_{n-2}) + c y_n = g(t_n).$$

This recurrence can also be solved to obtain a formula resembling (19). These formulas become really cumbersome to obtain (especially when it is unclear why we need them), so instead of hitting our heads against the wall repeatedly, we are going to move on to understand the language of finite difference equations.

## *3.2 The Language of $\zeta$ Transforms*

### 3.2.1 In Two Words

The $\zeta$ transform is to initial value problems for difference equations (recurrences) what the Laplace transform is to initial value problems for ODEs. However, it is mainly a formal transform, involving series whose convergence will not concern us at all.

### 3.2.2 The $\zeta$ Transform

Our input is a sequence $\{y_n\}$, which we can consider to start at $n = 0$ or to be causal ($y_n = 0$ for $n \leq -1$). We then associate the formal series

$$\mathrm{Y}(\zeta) := \sum_{n=0}^{\infty} y_n \zeta^n.$$

This series is to be understood in a purely algebraic way, as a sort of polynomial with infinitely many coefficients. A simple operation to be performed with causal sequences is the displacement to the right

$$(y_0, y_1, y_2, \ldots) \quad \longmapsto \quad (0, y_0, y_1, \ldots)$$

which we can understand with the full causal sequence as $\{y_n\} \mapsto \{y_{n-1}\}$. This operation is very easy to describe with $\zeta$ series:

$$\mathrm{Y}(\zeta) \quad \longmapsto \quad \zeta \mathrm{Y}(\zeta) = \sum_{n=1}^{\infty} y_{n-1} \zeta^n.$$

The second important operation that $\zeta$ transforms describe in a simple way is discrete convolution. The convolution

$$\sum_{m=0}^{n} a_m b_{n-m}$$

corresponds to the product of the $\zeta$ transforms

$$\mathrm{A}(\zeta)\mathrm{B}(\zeta) = \sum_{n=0}^{\infty} \left( \sum_{m=0}^{n} a_m b_{n-m} \right) \zeta^n.$$

In these expressions we might be thinking of scalar quantities (sequences taking values in $\mathbb{R}$ or $\mathbb{C}$) or more complicated situations, where, for instance, $\{y_n\}$ is a sequence in a vector space $X$ and the sequences $\{a_n\}$ and $\{b_n\}$ take values on spaces of operators where the multiplications $a_m b_{n-m}$ are meaningful.

### 3.2.3 BE Again

The discrete recurrence

$$\tfrac{1}{\kappa}(y_n - y_{n-1}) - c y_n = g_n, \qquad g_n := g(t_n),$$

is transformed into

$$\left( \frac{1-\zeta}{\kappa} - c \right) \mathrm{Y}(\zeta) = \mathrm{G}(\zeta)$$

and thus yields

$$\mathrm{Y}(\zeta) = \frac{1}{\frac{1-\zeta}{\kappa} - c} \mathrm{G}(\zeta). \tag{22}$$

Working through the algebra, we can easily write

$$\frac{1}{\frac{1-\zeta}{\kappa} - c} = \frac{\kappa}{(1-\kappa c) - \zeta} = \frac{\kappa}{1-\kappa c} \left( 1 - \frac{\zeta}{1-\kappa c} \right)^{-1} = \sum_{n=0}^{\infty} \frac{\kappa}{(1-\kappa c)^{n+1}} \zeta^n, \tag{23}$$

which means that (22) and (23) are just encoding (19).

### 3.2.4 BDF2

The $\zeta$ transform of the BDF2 equations

$$\tfrac{1}{\kappa}(\tfrac{3}{2} y_n - 2 y_{n-1} + \tfrac{1}{2} y_{n-2}) + c y_n = g_n, \qquad g_n := g(t_n)$$

is

$$\left(\frac{1}{\kappa}\left(\frac{3}{2}-2\zeta+\frac{1}{2}\zeta^2\right)-c\right)\mathrm{Y}(\zeta)=\mathrm{G}(\zeta),$$

or in explicit form

$$\mathrm{Y}(\zeta)=\frac{1}{\frac{1}{\kappa}\left(\frac{3}{2}-2\zeta+\frac{1}{2}\zeta^2\right)-c}\mathrm{G}(\zeta). \tag{24}$$

### 3.2.5 Implicit Differentiation by the Trapezoidal Rule

The equation $\dot{y}-cy=g$ can also be approximated using the trapezoidal rule

$$\frac{1}{\kappa}(y_n-y_{n-1})-\frac{c}{2}(y_n+y_{n-1})=\frac{1}{2}(g_n+g_{n-1}). \tag{25}$$

In this case we are not starting from a backward discretization of the derivative, but have found the method working directly on the differential equation. With $\zeta$ transforms, (25) is written as

$$\left(\frac{1-\zeta}{\kappa}-c\frac{1+\zeta}{2}\right)\mathrm{Y}(\zeta)=\frac{1+\zeta}{2}\mathrm{G}(\zeta),$$

which can be given explicitly as

$$\mathrm{Y}(\zeta)=\frac{1}{\frac{1}{\kappa}2\frac{1-\zeta}{1+\zeta}-c}\mathrm{G}(\zeta) \tag{26}$$

### 3.2.6 Summary

It is instructive to pay attention to the expressions of the recurrences (22), (24), and (26). All of them share the form

$$\mathrm{Y}(\zeta)=\frac{1}{\frac{1}{\kappa}\delta(\zeta)-c}\mathrm{G}(\zeta),$$

with

$$\delta(\zeta):=\begin{cases}1-\zeta, & \text{(BE)},\\ \frac{3}{2}-2\zeta+\frac{1}{2}\zeta^2, & \text{(BDF2)},\\ 2\frac{1-\zeta}{1+\zeta}, & \text{(TR)}.\end{cases}$$

We already understood $\delta(\zeta)$ as a backward differentiation formula for BE and BDF2. We could also think of higher order BDF methods

$$\delta(\zeta) := \sum_{\ell=1}^{p} \frac{1}{\ell}(1-\zeta)^{\ell}.$$

While these methods are useful in the parabolic world, we will not be able to use them in wave propagation problems, so we will quietly put them aside. The trapezoidal approximation deserves some special attention. The function

$$\delta(\zeta) = 2\frac{1-\zeta}{1+\zeta} = 2(1-\zeta)\sum_{n=0}^{\infty}(-\zeta)^n = 2 + 4\sum_{n=1}^{\infty}(-1)^n\zeta^n$$

does not represent a simple difference formula for approximation of the derivative of a function. Instead, it can be seen as a long memory approximation of the derivative of a causal function

$$\dot{y}(t_n) \approx \frac{1}{\kappa}\big(2y(t_n) - 4y(t_{n-1}) + 4y(t_{n-2}) - 4y(t_{n-3}) + \dots\big)$$

(the sum is finite because $y$ is causal), or as an implicit method to approximate the derivative. If we know $y$, then $h = \dot{y}$ is approximated by solving recurrently

$$\frac{1}{2}(h_n + h_{n-1}) = \frac{1}{\kappa}(y_n - y_{n-1}), \qquad y_n = y(t_n).$$

This formula might look surprising to the reader. Just think that differentiating the function is a sort of opposite process to solving a differential equation, so now the data are being differentiated and appear in the right-hand-side.

## *3.3 Moving from s to ζ*

### 3.3.1 Back to the Laplace Transform

The causal solution to $\dot{y} - cy = g$ is given through its Laplace transform by

$$\mathrm{Y}(s) = \frac{1}{s-c}\mathrm{G}(s).$$

The discrete versions we have obtained in the previous subsection fit in the general frame

$$\mathrm{Y}(\zeta) = \frac{1}{\frac{1}{\kappa}\delta(\zeta) - c}\mathrm{G}(\zeta),$$

or also in implicit form

$$\frac{1}{\kappa}\delta(\zeta)\mathrm{Y}(\zeta) - c\mathrm{Y}(\zeta) = \mathrm{G}(\zeta), \tag{27}$$

where we see the approximation of the differentiation operator by a discrete derivative operator, given in $\zeta$ transformed style by $\frac{1}{\kappa}\delta(\zeta)$. Our transfer function was $(s-c)^{-1}$. The discrete transfer function is

$$\left(\frac{1}{\kappa}\delta(\zeta) - c\right)^{-1}.$$

We do not need the expansion of this function into coefficients, because the method itself is built from a recurrence we can just apply. However, just for the sake of the argument, let us formally expand:

$$\left(\frac{1}{\kappa}\delta(\zeta) - c\right)^{-1} = \sum_{n=0}^{\infty} \omega_n^c(\kappa)\zeta^n.$$

Then, the recurrence hidden in (27) can be given an explicit form

$$y_n = \sum_{m=0}^{n} \omega_n^c(\kappa) g(t_{n-m}).$$

### 3.3.2 A First Generalization

Let us think again of our primitive wave equation $\ddot{y} + c^2 y = g$. In the Laplace domain this is

$$\mathrm{Y}(s) = \frac{1}{s^2 + c^2}\mathrm{G}(s),$$

which corresponds to

$$y(t) = c^{-1}\int_0^t \sin(c\tau) g(t-\tau)\mathrm{d}\tau.$$

If we use one of our time-stepping methods, we are essentially applying the recurrence

$$\left(\frac{1}{\kappa}\delta(\zeta)\right)^2 \mathrm{Y}(\zeta) + c^2\mathrm{Y}(\zeta) = \mathrm{G}(\zeta),$$

or in explicit form

$$\mathrm{Y}(\zeta) = \frac{1}{\left(\frac{1}{\kappa}\delta(\zeta)\right)^2 + c^2}\mathrm{G}(\zeta).$$

If we are able to expand

$$\frac{1}{\left(\frac{1}{\kappa}\delta(\zeta)\right)^2 + c^2} = \sum_{n=0}^{\infty} \omega_n^{c^2}(\kappa)\zeta^n,$$

then the recurrence becomes a discrete convolution

$$y_n = \sum_{m=0}^{n} \omega_n^{c^2}(\kappa) g(t_{n-m}).$$

Even if we are just rewriting very simple solvers for linear differential equations, we can see how the final discrete convolution formula uses the Laplace transform of the solution operator (in this case $(s^2 + c^2)^{-1}$), while data are sampled in the time domain.

### 3.3.3 Convolutions Become Discrete

We next move to a general convolution

$$y = f * g. \tag{28}$$

Here $f$ and $g$ are known. We need $g$ to be a causal function $\mathbb{R} \to X$. On the other hand $f$ can be any causal Laplace transformable distribution with values in $\mathcal{B}(X, Y)$. We actually do not need to know $f$, but are happy enough with its Laplace transform $\mathrm{F}(s)$. The Laplace transform of (28) is

$$\mathrm{Y}(s) = \mathrm{F}(s)\mathrm{G}(s).$$

The discrete equations are then

$$\mathrm{Y}(\zeta) = \mathrm{F}(\tfrac{1}{\kappa}\delta(\zeta))\mathrm{G}(\zeta), \qquad \mathrm{G}(\zeta) := \sum_{n=0}^{\infty} g(t_n)\zeta^n. \tag{29}$$

If we are able to expand

$$\mathrm{F}(\tfrac{1}{\kappa}\delta(\zeta)) = \sum_{n=0}^{\infty} \omega_n^{\mathrm{F}}(\kappa)\zeta^n, \tag{30}$$

then (29) becomes a discrete convolution

$$y_n = \sum_{m=0}^{n} \omega_m^{\mathrm{F}}(\kappa) g(t_{n-m}), \qquad n \geq 0 \tag{31}$$

which uses time-domain readings of the data function $g$ combined with a discrete transfer function that arises from the original transfer function and the approximation of the differentiation operator. The discrete convolution (31), with coefficients computed following (30), is the **Convolution Quadrature** method to approximate the convolution (28) at discrete time steps. Recall that our hypotheses for F included F to be analytic in $\mathbb{C}_+$. If $\delta(0) \in \mathbb{C}_+$, then the function $\zeta \mapsto \mathrm{F}(\frac{1}{\kappa}\delta(\zeta))$ is analytic in a neighborhood of $\zeta = 0$ and the expansion (30) is just a Taylor expansion.

### 3.3.4 Convolution Equations

A convolution equation can be seen (in the Laplace domain) as

$$\mathrm{F}(s)\mathrm{G}(s) = \mathrm{H}(s) \qquad \text{or} \qquad \mathrm{G}(s) = \mathrm{F}(s)^{-1}\mathrm{H}(s). \tag{32}$$

The version in the right-hand-side is an explicit convolution, with the caveat that $\mathrm{F}(s)^{-1}$ might not be known. The Convolution Quadrature method is then given by the recurrence that is equivalent to the $\zeta$ transform equation:

$$\mathrm{F}(\tfrac{1}{\kappa}\delta(\zeta))\mathrm{G}(\zeta) = \mathrm{H}(\zeta), \qquad \mathrm{H}(\zeta) := \sum_{n=0}^{\infty} h(t_n)\zeta^n.$$

In discrete times, this is equivalent to

$$\sum_{m=0}^{n} \omega_m^{\mathrm{F}}(\kappa) g_{n-m} = h(t_n), \qquad n \geq 0, \tag{33}$$

or, in a slightly more explicit form, to

$$\omega_0^{\mathrm{F}}(\kappa) g_n = h(t_n) - \sum_{m=1}^{n} \omega_m^{\mathrm{F}}(\kappa) g_{n-m} = h(t_n) - \sum_{m=0}^{n-1} \omega_{n-m}^{\mathrm{F}}(\kappa) g_m.$$

The coefficients $\{\omega_n^{\mathrm{F}}(\kappa)\} \subset \mathcal{B}(X, Y)$ are computed using (30). The method is well defined when $\omega_0^{\mathrm{F}}(\kappa)$ is invertible.

### 3.3.5 Just in Case

The way we have written the convolution equation (32) raises an interesting issue. Assume that we happen to know both $\mathrm{F}(s)$ and $\mathrm{F}(s)^{-1}$, and that we can expand

$$\mathrm{F}(\tfrac{1}{\kappa}\delta(\zeta)) = \sum_{n=0}^{\infty} \omega_n^{\mathrm{F}}(\kappa)\zeta^n, \qquad \mathrm{F}(\tfrac{1}{\kappa}\delta(\zeta))^{-1} = \sum_{n=0}^{\infty} \omega_n^{\mathrm{F}^{-1}}(\kappa)\zeta^n. \tag{34}$$

We wonder whether (33) (the discretization of the convolution equation) is the same method as

$$g_n = \sum_{m=0}^{n} \omega_m^{\mathrm{F}^{-1}}(\kappa) h(t_{n-m}).$$

The answer is yes. The key to understanding why lies in the fact that both expansions (34) are Taylor expansions. In fact, there is an underlying assumption in the background: we assume that F is a transfer function in the conditions of Sect. 2.4, and so is $\mathrm{F}^{-1}$. This means that for all $s \in \mathbb{C}_+$, we need $\mathrm{F}(s)$ to be invertible and we want the function $\mathbb{C}_+ \ni s \mapsto F(s)^{-1} \in \mathcal{B}(Y, X)$ to be analytic (we get this for free) and to admit a bound like (13). In particular, these hypotheses imply (see Sect. 2.4) that the convolution equation $f * g = h$ is solved with another convolution equation $g = f^{-1} * g$, where $\mathcal{L}\{f^{-1}\}(s) = \mathrm{F}(s)^{-1}$. In this case

$$\mathrm{F}(\tfrac{1}{\kappa}\delta(\zeta))\mathrm{F}(\tfrac{1}{\kappa}\delta(\zeta))^{-1} = \mathrm{I}$$

for $\zeta$ small enough, which implies that

$$\omega_0^{\mathrm{F}}(\kappa)^{-1} = \omega_0^{\mathrm{F}^{-1}}(\kappa), \qquad \sum_{m=0}^{n} \omega_m^{\mathrm{F}}(\kappa)\omega_{n-m}^{\mathrm{F}^{-1}}(\kappa) = 0 \quad n \geq 1.$$

This proves that

$$\mathrm{F}(\tfrac{1}{\kappa}\delta(\zeta))\mathrm{G}(\zeta) = \mathrm{H}(\zeta) \qquad \text{and} \qquad \mathrm{G}(\zeta) = \mathrm{F}(\tfrac{1}{\kappa}\delta(\zeta))^{-1}\mathrm{H}(\zeta)$$

deliver the same sequence $\{g_n\}$.

### 3.3.6 Why Implicit

It is tempting to think whether an explicit approximation of the derivative would work. While in principle there might not be any problem for it (you just need to compute what $\delta(\zeta)$ is, plug it in (30), and expand), it is not very clear what we would gain from it. As already emphasized, we are not dealing with linear differential equations. We are merely using them to motivate the methods. The actual transfer

functions are analytic functions of the variable $s$ and using one $\delta(\zeta)$ or another might not make much of a difference at the point of implementing the method. It has to be said, nevertheless, that for applications in wave propagation, using the discrete differentiation operators associated to explicit approximations of the derivative does not work. We can also just wash our hands and claim that we are dealing with causal problems (causal convolutions), and we want to keep everything causal. Explicit (forward) differentiation breaks causality by looking into the future.

## *3.4 A Bold Approach in the Laplace Domain*

### 3.4.1 A New and Fast Introduction of BE-CQ

Let us just focus on causal convolutions

$$y = f * g \qquad \text{i.e.} \qquad \mathrm{Y}(s) = \mathrm{F}(s)\mathrm{G}(s).$$

A particular case is $\mathrm{F}(s) = s$, corresponding to differentiation $y = \dot{g}$. We fix the time step $\kappa$ but do not pay special attention to the discrete times. The Euler discrete backward derivative approximation to $y = \dot{g}$ is

$$y \approx \tfrac{1}{\kappa}(g - g(\cdot - \kappa)) =: y_\kappa.$$

The Laplace domain form of this equation is

$$\mathrm{Y}_\kappa(s) = \tfrac{1}{\kappa}(1 - e^{-s\kappa})\mathrm{G}(s).$$

This can be written as

$$\mathrm{Y}_\kappa(s) = s_\kappa \mathrm{G}(s), \tag{35}$$

where

$$s_\kappa = \tfrac{1}{\kappa}(1 - e^{-s\kappa}) = \tfrac{1}{\kappa}\delta(e^{-s\kappa}), \qquad \delta(\zeta) = 1 - \zeta.$$

### 3.4.2 BDF2 and TR

We can try the same ideas with our other two particular methods introduced in Sects. 3.2 and 3.3. The BDF2 approximation of the derivative applied to $y = \dot{g}$ is

$$y \approx y_\kappa := \tfrac{1}{\kappa}(\tfrac{3}{2}g - 2g(\cdot - \kappa) + \tfrac{1}{2}g(\cdot - 2\kappa)),$$

which can be written as

$$\mathrm{Y}_\kappa(s) = \underbrace{\tfrac{1}{\kappa}(\tfrac{3}{2} - 2e^{-s\kappa} + \tfrac{1}{2}e^{-2s\kappa})}_{s_\kappa}\mathrm{G}(s).$$

In the trapezoidal rule $y_\kappa$ is approximated implicitly

$$\tfrac{1}{2}(y_\kappa + y_\kappa(\cdot - \kappa)) = \tfrac{1}{\kappa}(g - g(\cdot - \kappa))$$

and we similarly obtain (35) with

$$s_\kappa := \frac{2}{\kappa}\frac{1 - e^{-s\kappa}}{1 + e^{-s\kappa}} = \frac{1}{\kappa}\delta(e^{-s\kappa}).$$

### 3.4.3 Convolution Quadrature Again

We then extend the idea of approximating

$$\mathrm{Y}(s) = s\mathrm{G}(s) \qquad \text{by} \qquad \mathrm{Y}_\kappa(s) = s_\kappa\mathrm{G}(s),$$

for more general transfer functions, approximating then

$$\mathrm{Y}(s) = \mathrm{F}(s)\mathrm{G}(s) \qquad \text{by} \qquad \mathrm{Y}_\kappa(s) = \mathrm{F}(s_\kappa)\mathrm{G}(s).$$

We next have to figure out what this method is. Recall that we are using

$$\mathrm{F}(s_\kappa), \qquad \text{with} \qquad s_\kappa = \tfrac{1}{\kappa}\delta(e^{-s\kappa}).$$

We start by Taylor expanding

$$\mathrm{F}(\tfrac{1}{\kappa}\delta(\zeta)) = \sum_{n=0}^{\infty}\omega_n^{\mathrm{F}}(\kappa)\zeta^n,$$

and then substituting

$$\mathrm{F}(s_\kappa) = \sum_{n=0}^{\infty}\omega_n^{\mathrm{F}}(\kappa)e^{-n\kappa s} = \sum_{n=0}^{\infty}\omega_n^{\mathrm{F}}(\kappa)e^{-t_n s}. \tag{36}$$

We finally recognize that $\mathrm{F}(s_\kappa)$ is the Laplace transform of the distribution

$$\sum_{n=0}^{\infty}\omega_n^{\mathrm{F}}(\kappa) \otimes \delta_{t_n}$$

and therefore $Y_\kappa(s) = \mathrm{F}(s_\kappa)\mathrm{G}(s)$ is the Laplace transform of

$$y_\kappa = \sum_{m=0}^{\infty} (\omega_m^{\mathrm{F}}(\kappa) \otimes \delta_{t_m}) * g = \sum_{m=0}^{\infty} \omega_m^{\mathrm{F}}(\kappa) g(\cdot - t_m). \tag{37}$$

In this form $g$ is allowed to be discontinuous in time and $y_\kappa$ is obtained as a function of continuous time. The sum in (37) is finite for any given value of the time variable. For instance

$$y_\kappa(t) = \sum_{m=0}^{n} \omega_m^{\mathrm{F}}(\kappa) g(t - t_m), \qquad t_n \le t < t_{n+1}.$$

In particular, with $t = t_n$ we obtain the discrete Convolution Quadrature method of Sect. 3.3

$$y_n := y_\kappa(t_n) = \sum_{m=0}^{n} \omega_m^{\mathrm{F}}(\kappa) g(t_{n-m}).$$

This means that the discrete sequence $\{y_n\}$ obtained in Sect. 3.3 is actually composed of time samples of a continuous-in-time function $y_\kappa$. Of course, if we want to evaluate $y_\kappa$ at a point that is not on the time grid, we have to use $g$ at different points, instead of only at the points $t_m$. The case of convolution equations follows similar lines: when we approximate $\mathrm{F}(s)\mathrm{G}(s) = \mathrm{H}(s)$ by $\mathrm{F}(s_\kappa)\mathrm{G}_\kappa(s) = \mathrm{H}(s)$, we are equivalently writing the equation

$$\sum_{m=0}^{\infty} \omega_m^{\mathrm{F}}(\kappa) g_\kappa(\cdot - t_m) = h. \tag{38}$$

Ideally we could solve (38) progressively in intervals:

$$\omega_0^{\mathrm{F}}(\kappa) g(t) = h(t) - \sum_{m=1}^{n} \omega_m^{\mathrm{F}}(\kappa) g(t - t_m), \qquad t \in [t_{n-1}, t_n), \quad n \ge 0.$$

This would give $g_\kappa$ one interval after another. What we will focus on is the values $g_n := g_\kappa(t_n)$, which satisfy

$$\omega_0^{\mathrm{F}}(\kappa) g_n = h(t_n) - \sum_{m=1}^{n} \omega_m^{\mathrm{F}}(\kappa) g(t_{n-m}), \qquad n \ge 0,$$

sending us back to the CQ method of Sect. 3.3.

### 3.4.4 Lubich's Notation

We have favored some sort of convolutional notation in these notes. In his expositions of the method, Christian Lubich has adopted a clever operational notation that is often useful. The idea is simple. Differentiation $y = \partial g = \dot{g}$ corresponds to the operator 'multiply by $s$'. We then admit the notation

$$y = \mathrm{F}(\partial)g = f * g, \qquad \mathrm{F} = \mathcal{L}\{f\},$$

that makes the transfer function apparent even in the time domain. In CQ, there is a discrete differentiation operator

$$\partial_\kappa g \approx \dot{g}, \qquad \mathcal{L}\{\partial_\kappa g\}(s) = s_\kappa \mathrm{G}(s), \qquad s_\kappa = \tfrac{1}{\kappa}\delta(e^{-s\kappa}),$$

and the CQ method is denoted

$$\mathrm{F}(\partial_\kappa)g = \sum_{m=0}^{\infty} \omega_m^{\mathrm{F}}(\kappa) g_\kappa(\cdot - t_m) = f_\kappa * g, \qquad f_\kappa := \sum_{m=0}^{\infty} \omega_m^{\mathrm{F}}(\kappa) \otimes \delta_{t_m},$$

which makes $\mathcal{L}\{f_\kappa\}(s) = \mathrm{F}(s_\kappa)$.

## 3.5 *Another Point of View*

### 3.5.1 Aims and Tools

In this section we are going to give Lubich's original introduction to the BE-CQ method. We will restrict to some particular transfer functions $\mathrm{F} : \mathbb{C}_+ \to \mathcal{B}(X, Y)$ satisfying

$$\|\mathrm{F}(s)\| \le C(\mathrm{Re}\, s)|s|^\mu \qquad \forall s \in \mathbb{C}_+, \quad \mu < -1.$$

This condition is enough to show that the Laplace inversion formula

$$f(t) = \frac{1}{2\pi \mathrm{i}} \int_{\sigma - \mathrm{i}\infty}^{\sigma + \mathrm{i}\infty} e^{st} \mathrm{F}(s) \mathrm{d}s = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{(\sigma + \mathrm{i}\omega)t} \mathrm{F}(\sigma + \mathrm{i}\omega) \mathrm{d}\omega \tag{39}$$

defines a causal continuous function whose Laplace transform is F, independently of the value of $\sigma$ taken for the inversion path. The hypotheses also ensure that if $a > \sigma$, then

$$\frac{1}{n!}\mathrm{F}^{(n)}(a) = -\frac{1}{2\pi \mathrm{i}} \int_{\sigma - \mathrm{i}\infty}^{\sigma + \mathrm{i}\infty} \frac{1}{(s-a)^{n+1}} \mathrm{F}(s) \mathrm{d}s. \tag{40}$$

Equation (40) is a somewhat uncommon expression deriving from the Cauchy integral formulas. It can be proved using the Cauchy formulas: integrating along a rectangular contour with the left edge in the line $\sigma + \mathrm{i}\mathbb{R}$, the right edge in $R + \mathrm{i}\mathbb{R}$ and the upper and bottom edges in $\mathbb{R} \pm \mathrm{i}R$, and then sending $R \to \infty$.

### 3.5.2 A Mixed Formulation of Causal Convolution

All the following steps can be justified, just stay with the flow:

$$(f * g)(t) = \int_0^t f(\tau) g(t-\tau) \mathrm{d}\tau = \int_0^t \left( \frac{1}{2\pi \mathrm{i}} \int_{\sigma-\mathrm{i}\infty}^{\sigma+\mathrm{i}\infty} e^{s\tau} \mathrm{F}(s) \mathrm{d}s \right) g(t-\tau) \mathrm{d}\tau$$
$$= \frac{1}{2\pi \mathrm{i}} \int_{\sigma-\mathrm{i}\infty}^{\sigma+\mathrm{i}\infty} \mathrm{F}(s) \underbrace{\left( \int_0^t e^{s\tau} g(t-\tau) \mathrm{d}\tau \right)}_{y(t)=y(t;s)} \mathrm{d}s.$$

We have now reached a point where $g$ appears in a scalar convolution equation, corresponding to the solution of $\dot{y} = sy+g$, with the implicit assumption of causality that can also be phrased as $y(0) = 0$.

### 3.5.3 Introducing the ODE Solver

We restrict our attention to $t = t_n$ and approximate

$$\int_0^{t_n} e^{s\tau} g(t_n - \tau) \mathrm{d}\tau \approx \sum_{m=0}^{n} \frac{\kappa}{(1-\kappa s)^{m+1}} g(t_{n-m}),$$

that is, we approximate $y(t_n)$ using the BE method. This leads to

$$(f * g)(t_n) \approx \sum_{m=0}^{n} \underbrace{\left( \frac{1}{2\pi \mathrm{i}} \int_{\sigma-\mathrm{i}\infty}^{\sigma+\mathrm{i}\infty} \frac{\kappa}{(1-\kappa s)^{m+1}} \mathrm{F}(s) \mathrm{d}s \right)}_{\omega_m^{\mathrm{F}}(\kappa)} g(t_{n-m}).$$

The final step consists of the figuring out what the coefficients $\omega_n^{\mathrm{F}}(\kappa)$ are. From (40) and some elementary arguments, we obtain

$$\omega_n^{\mathrm{F}}(\kappa) = \frac{(-1)^n}{\kappa^n} \left( -\frac{1}{2\pi \mathrm{i}} \int_{\sigma-\mathrm{i}\infty}^{\sigma+\mathrm{i}\infty} \frac{1}{(s-1/\kappa)^{n+1}} \mathrm{F}(s) \mathrm{d}s \right) = \frac{(-1)^n}{\kappa^n} \frac{1}{n!} \mathrm{F}^{(n)}(1/\kappa)$$
$$= \frac{1}{n!} \frac{\mathrm{d}^n}{\mathrm{d}\zeta^n} \left( \mathrm{F}\Big( \frac{1-\zeta}{\kappa} \Big) \right) \Bigg|_{\zeta=0}.$$

Therefore

$$\mathrm{F}\left(\frac{1-\zeta}{\kappa}\right) = \sum_{n=0}^{\infty} \omega_n^{\mathrm{F}}(\kappa)\zeta^n,$$

and we have found the BE-CQ method in a slightly different way.

## 3.6 A Discrete Operational Calculus

### 3.6.1 Rephrasing

What we have done so far can be understood in many different ways, but it can be synthesized with several simple statements. Given a causal $\mathcal{B}(X, Y)$-valued distribution with Laplace transform $\mathrm{F} : \mathbb{C}_+ \to \mathcal{B}(X, Y)$, we

$$\text{approximate } f \approx \sum_{n=0}^{\infty} \omega_n^{\mathrm{F}}(\kappa) \otimes \delta_{t_n}, \text{ where } \mathrm{F}(\tfrac{1}{\kappa}\delta(\zeta)) = \sum_{n=0}^{\infty} \omega_n^{\mathrm{F}}(\kappa)\zeta^n. \tag{41}$$

The function $\delta(\zeta)$ can be understood as a generator for an approximation of the differentiation operator, so that

$$\tfrac{1}{\kappa}\delta(\zeta) = \tfrac{1}{\kappa}\sum_{n=0}^{\infty} \delta_n \zeta^n \tag{42}$$

is the symbolic form of the operator

$$\partial_\kappa g := \tfrac{1}{\kappa}\sum_{n=0}^{\infty} \delta_n g(\cdot - n\kappa).$$

Equations (41) and (42) can also be written in the Laplace domain, via the definition of

$$s_\kappa := \tfrac{1}{\kappa}\delta(e^{-s\kappa}) = \tfrac{1}{\kappa}\sum_{n=0}^{\infty} \delta_n e^{-s t_n} \approx s,$$

to give approximations

$$\mathrm{F}(s_\kappa) = \sum_{n=0}^{\infty} \omega_n^{\mathrm{F}}(\kappa) e^{-s t_n} \approx \mathrm{F}(s). \tag{43}$$

As already mentioned, there are different ways of notating the convolution operator defined by (41). One way is to name

$$f_\kappa := \sum_{n=0}^{\infty} \omega_n^{\mathrm{F}}(\kappa) \otimes \delta_{t_n}$$

and then note that $f_\kappa * g \approx f * g$. The other way is to name the operator directly $\mathrm{F}(\partial_\kappa)g$. *There is a caveat to this condensed introduction.* Even if the CQ expressions $\mathrm{F}(\partial_\kappa)g = f_\kappa * g$ exist in continuous time, they are only computed in discrete times, so knowledge of the output of these operators is purely given at discrete time steps.

### 3.6.2 Associativity

Let now $\mathrm{F}_1 : \mathbb{C}_+ \to \mathcal{B}(X, Y)$ and $\mathrm{F}_2 : \mathbb{C}_+ \to \mathcal{B}(Z, X)$. It is clear from the Laplace transform of the convolution quadrature operators that

$$\mathrm{F}_1(s_\kappa)\big(\mathrm{F}_2(s_\kappa)\mathrm{G}(s)\big) = \big(\mathrm{F}_1(s_\kappa)\mathrm{F}_2(s_\kappa)\big)\mathrm{G}(s),$$

which is another way of writing

$$\mathrm{F}_1(\partial_\kappa)\big(\mathrm{F}_2(\partial_\kappa)g\big) = \big(\mathrm{F}_1(\partial_\kappa)\mathrm{F}_2(\partial_\kappa)\big)g, \tag{44}$$

or equivalently $f_{1,\kappa} * (f_{2,\kappa} * g) = (f_{1,\kappa} * f_{2,\kappa}) * g$. The latter expression emphasizes the convolutional form of all operators, from where associativity is clear. The interest of (44) might not be apparent at first sight. In practice, we are going to apply the left-hand-side of (44), that is, first one discrete convolution process, and then the second one. However, this is equivalent to applying a simple CQ process, which we will never compute (it would mean composing sequences of operators), but we can use for analysis.

### 3.6.3 Forward Convolutions and Equations

The associativity idea can be further exploited to see why analyzing CQ for convolution equations is equivalent to analyzing CQ for the inverse operator, even if this is only known at the theoretical level. If $\mathrm{F} : \mathbb{C}_+ \to \mathcal{B}(X, Y)$ and $\mathrm{F}^{-1} : \mathbb{C}_+ \to \mathcal{B}(Y, X)$ are operators such that

$$\mathrm{F}(s)\mathrm{F}^{-1}(s) = \mathrm{I}_Y, \qquad \mathrm{F}^{-1}(s)\mathrm{F}(s) = \mathrm{I}_X, \qquad \forall s \in \mathbb{C}_+,$$

(or, in other words, $\mathrm{F}^{-1}(s) = (\mathrm{F}(s))^{-1}$), then

$$\mathrm{F}(s_\kappa)\mathrm{F}^{-1}(s_\kappa) = \mathrm{I}_Y, \qquad \mathrm{F}^{-1}(s_\kappa)\mathrm{F}(s_\kappa) = \mathrm{I}_X,$$

which means that the discrete operators associated to $\mathrm{F}(s_\kappa)$ and $\mathrm{F}^{-1}(s_\kappa)$ are inverse of each other. Therefore, the analysis of the equation $f_\kappa * g = h$ (here $g$ is unknown) is equivalent to the analysis of the inverse operator $g = f_\kappa^{-1} * h$.

## 3.7 Discrete Transfer Functions

### 3.7.1 Motivation

Just for the sake of it, let us explore the problem of inverting discrete convolution equations. Let $\{f_n\} \subset \mathcal{B}(X, Y)$ and $\{h_n\} \subset Y$ be given sequences. We look for a sequence $\{g_n\} \subset X$ satisfying

$$\sum_{m=0}^{n} f_m g_{n-m} = h_n. \tag{45}$$

It is clear that if $f_0$ is invertible, then the sequence $\{g_n\}$ can be computed with the semi-implicit recurrence

$$g_n = f_0^{-1}(h_n - \sum_{m=1}^{n} f_m g_{n-m}). \tag{46}$$

With $\zeta$ transforms the convolution equation (45) can be written as

$$\mathrm{F}(\zeta)\mathrm{G}(\zeta) = \mathrm{H}(\zeta). \tag{47}$$

### 3.7.2 The Discrete Transfer Function

At least formally we can try to invert $\mathrm{F}(\zeta)$ and send it to the right-hand-side:

$$\mathrm{G}(\zeta) = \mathrm{R}(\zeta)\mathrm{H}(\zeta), \tag{48}$$

hoping to find a sequence $\{r_n\}$ that rewrites (46) in the explicit form

$$g_n = \sum_{m=0}^{n} r_m h_{n-m}.$$

The simplest way to find $\{r_n\}$ is to use a discrete unit impulse in the right-hand-side of (45). This unit impulse at discrete time zero is the sequence $\{\delta_{0,n}\}$ (written in terms of Kronecker deltas) and its $\zeta$ transform is $\mathrm{D}_0(\zeta) \equiv 1$. Clearly this is a way of formally obtaining $\mathrm{R}(\zeta)$ by plugging $\mathrm{H}(\zeta) \equiv 1$ as data in (48) or as right-hand-side of (47). Before we do it, let us just try to be more precise: in this case the

impulse is $\{\delta_{0,n}I\}$ where $I$ is the identity operator in the space $Y$ and $\{r_n\}$ will be a sequence in $\mathcal{B}(Y,X)$. The sequence $\{r_n\}$ (or $\mathrm{R}(\zeta) = \mathrm{F}(\zeta)^{-1}$) can be found with a simple recurrence

$$r_0 = f_0^{-1}, \qquad r_n = -f_0^{-1}\Big(\sum_{m=1}^{n} f_m r_{n-m}\Big), \quad n \geq 1.$$

This is the discrete transfer function for the solution operator associated to the convolution equation (45).

### 3.7.3 Throwing Analyticity into the Mix

Let us go further. In principle

$$\mathrm{F}(\zeta) = \sum_{n=0}^{\infty} f_n \zeta^n \tag{49}$$

is just a formal series. If $\|f_n\| \leq R^n$ for all $n$, then the power series (49) converges in the space $\mathcal{B}(X,Y)$ for $|\zeta| < R^{-1}$. It thus defines an analytic function from the disk centered at the origin with radius $1/R$ to $\mathcal{B}(X,Y)$. Let us write (49) in a slightly different way:

$$\mathrm{F}(\zeta) = f_0\Big(\sum_{n=0}^{\infty} f_0^{-1} f_n \zeta^n\Big),$$

so that we can focus on operators $\{f_0^{-1}f_n\} \subset \mathcal{B}(X,X)$. We then worry about invertibility of the operator

$$I_X + \mathrm{Q}(\zeta), \qquad \mathrm{Q}(\zeta) := \sum_{n=1}^{\infty} f_0^{-1} f_n \zeta^n. \tag{50}$$

A back of the envelope calculation yields

$$\Big\| \sum_{n=1}^{\infty} f_0^{-1} f_n \zeta^n \Big\| \leq \|f_0\|^{-1} \frac{1}{1-|\zeta|R}, \qquad |\zeta| < 1/R.$$

If we are lucky enough to have $\|f_0^{-1}\| < 1$, then, values of $\zeta$ satisfying

$$|\zeta| < \frac{1 - \|f_0^{-1}\|}{R},$$

provide $\|\mathrm{Q}(\zeta)\| < 1$ and therefore $I_X + \mathrm{Q}(\zeta)$ is invertible. In this case, the analytic function $\mathrm{F}(\zeta)$ is invertible in a disk around $\zeta = 0$ and

$$\mathrm{F}(\zeta)^{-1} = \sum_{n=0}^{\infty} r_n \zeta^n$$

is also a convergent power series corresponding to the discrete transfer function. Note that $\mathrm{F}(\zeta)^{-1}$ means exactly what it is written: it is the inverse operator for $\mathrm{F}(\zeta)$ for a given value of $\zeta$.

### *3.8 Credits*

The Convolution Quadrature method and its associated discretized operational calculus were created by Christian Lubich in the late 1980s [23], as a surprising computational extension of some classical ideas by Liouville. Lubich's introduction is the one shown in Sect. 3.5. A key paper in what respects to CQ applied to the wave equation is [25], which is the first reference to the use of CQ for a wave propagation problem. Any introduction to the numerical analysis of discretization methods for Ordinary Differential Equations contains basic material on multistep methods, and more specifically on the BDF formulas. Even if we have not seen it yet, A-stability will play an important role in how multistep methods become CQ solvers for hyperbolic problems. To learn about A-stability, see [17]. The language of $\zeta$-series is common to the analysis of ODEs, even if sometimes is just implicitly used. A highly readable presentation is given in Henrici's classic introduction to applied complex analysis [19]. The point of view of substituting a continuous symbol $\mathrm{F}(s)$ by a discrete symbol $\mathrm{F}(s_\kappa)$ is more or less implicit to [25]. It was made more apparent in Antonio Laliena's thesis [20] and it is the center of the theory for multistep CQ in [29].

## 4 Implementation

In this section we are going to show how to compute discrete convolutions and convolution equations with CQ.

## 4.1 The Discrete Fourier Transform

### 4.1.1 DFT and IDFT

Given a vector $\mathbf{x} := (x_0, \dots, x_M) \in \mathbb{C}^{M+1}$, we consider the vector $\hat{\mathbf{x}} \in \mathbb{C}^{M+1}$ given by

$$\hat{x}_\ell := \sum_{n=0}^{M} x_n \zeta_{M+1}^{-\ell n}, \quad \ell = 0, \dots, M, \qquad \text{where} \qquad \zeta_{M+1} := e^{\frac{2\pi \mathrm{i}}{M+1}}. \tag{51}$$

The expression (51) is the well known definition of the Discrete Fourier Transform (DFT). The Fast Fourier Transform (FFT) is a very optimized algorithm to compute exactly (51) when $M + 1 = 2^p$ for an integer $p$. The Inverse Discrete Fourier Transform (IDFT) is given by

$$x_n := \frac{1}{M+1} \sum_{\ell=0}^{M} \hat{x}_\ell \zeta_{M+1}^{\ell n}, \qquad n = 0, \dots, M. \tag{52}$$

The DFT and the IDFT are inverse operators. Moreover, the IDFT can be computed exactly as the DFT: first conjugate the vector $\hat{\mathbf{x}}$, then apply the DFT, finally conjugate the result again and divide by $M + 1$. This implies that any fast algorithm for the DFT can also be applied to the IDFT.

### 4.1.2 The DFT and Periodic Discrete Convolutions

Let $\mathbf{x}, \mathbf{y} \in \mathbb{C}^{M+1}$. We define their periodic discrete convolution $\mathbf{x} *_{\text{per}} \mathbf{y} \in \mathbb{C}^{M+1}$ with the formula

$$(\mathbf{x} *_{\text{per}} \mathbf{y})_n := \sum_{m=0}^{n} x_m y_{n-m} + \sum_{m=n+1}^{M} x_m y_{M+1+n-m}, \qquad n = 0, \dots, M. \tag{53}$$

The formula in the definition (53) is not very inspiring. It is much easier to think that $\mathbf{x}, \mathbf{y} \in \ell^0(\mathbb{Z})$ are sequences indexed by $n \in \mathbb{Z}$ that are $(M + 1)$-periodic. Then

$$(\mathbf{x} *_{\text{per}} \mathbf{y})_n = \sum_{m=0}^{M} x_m y_{n-m} \qquad n \in \mathbb{Z} \tag{54}$$

is also an $(M+1)$-periodic sequence coinciding with (53) in its main entries, indexed from $n = 0$ to $n = M$. The DFT diagonalizes periodic convolutions in the following sense:

$$\widehat{(\mathbf{x} *_{\text{per}} \mathbf{y})}_\ell = \hat{x}_\ell \hat{y}_\ell. \tag{55}$$

Therefore, a fast way of computing periodic convolutions is to take the DFT of the two vectors, multiply them component by component, and then take the IDFT of the result.

### 4.1.3 Discrete Convolutions Made Periodic

Let now $\mathbf{x}, \mathbf{y}$ be causal sequences. We are interested in computing the first $N$ components of their discrete convolution

$$(\mathbf{x} * \mathbf{y})_n = \sum_{m=0}^{n} x_m y_{n-m}, \qquad n = 0, \ldots, N. \tag{56}$$

This formula obviously involves only the vectors $(x_0, \ldots, x_N), (y_0, \ldots, y_N) \in \mathbb{C}^{N+1}$. Let then

$$\mathbf{x}^{\text{ext}} := (x_0, \ldots, x_N, \underbrace{0, \ldots, 0}_{N+1}), \quad \mathbf{y}^{\text{ext}} := (y_0, \ldots, y_N, \underbrace{0, \ldots, 0}_{N+1}) \in \mathbb{C}^{2N+2} \tag{57}$$

be the result of cutting the vectors to the components that are needed for (56) and then extending them with zeros, thus doubling the number of components of the vectors. A simple argument shows that

$$(\mathbf{x} * \mathbf{y})_n = (\mathbf{x}^{\text{ext}} *_{\text{per}} \mathbf{y}^{\text{ext}})_n, \qquad n = 0, \ldots, N. \tag{58}$$

Formulas (57) and (58) give an algorithm to compute the beginning of a discrete convolution of sequences (56).

**Algorithm 1 (discrete convolutions by the DFT)**

We aim to compute

$$(\mathbf{x} * \mathbf{y})_n = \sum_{m=0}^{n} x_m y_{n-m}, \qquad n = 0, \ldots, N.$$

(a) Keep $N + 1$ components of the sequences $\mathbf{x}$ and $\mathbf{y}$ and extend them with $N + 1$ zeros at the end (57).
(b) Take the DFT of the extended vectors.
(c) Multiply the resulting vectors component by component.
(d) Take the IDFT of the result of (c).
(e) Keep only the first $N + 1$ components of the result of (d).

### 4.1.4 Symmetry Arguments

It will be common in our computations that roughly half of the coefficients of a vector will be conjugated from the other half. Before we meet this, let us introduce some notation. Assume that $\mathbb{X}$ is a Banach space where we can conjugate. Let $\mathbf{x} = (x_0, \dots, x_N) \in \mathbb{X}^{N+1}$ be a vector of elements of $\mathbb{X}$. If

$$x_{N+1-\ell} = \overline{x}_\ell \qquad \ell = 1, \dots, N,$$

we will say that the vector $\mathbf{x}$ is Hermitian (note that this is not a standard definition). This is equivalent to the following construction: extend first $\mathbf{x}$ to a sequence $x_n$ with $n \in \mathbb{Z}$ which is $(N+1)$-periodic; then $\mathbf{x}$ is Hermitian whenever $x_\ell = \overline{x}_\ell$ for all $\ell \geq 1$. In later examples we will also have that $\overline{x}_0 = x_0$, which would yield a better definition of Hermiticity. A clear example of a Hermitian vector is the DFT of a vector such that $\overline{x}_n = x_n$ for all $n$. In forthcoming algorithms to compute components $x_\ell$ for $\ell = 0, \dots, N$, we will say that the algorithm can be symmetrized when we know in advance that $\mathbf{x}$ is Hermitian. Then we will:

- Compute $x_\ell$ for $\ell = 0, \dots, \lfloor \frac{N+1}{2} \rfloor$.
- Copy the missing components $x_{N+1-\ell} = \overline{x}_\ell$ for $\ell = 1, \dots \lfloor \frac{N}{2} \rfloor$.

## *4.2 Computation of CQ Weights*

### 4.1.1 Our Next Goal

We now prepare the way to compute some of the weights of the CQ process:

$$\omega_n^{\mathrm{F}}(\kappa) := \frac{1}{n!} \frac{\mathrm{d}^n}{\mathrm{d}\zeta^n} \left( \mathrm{F}\left( \frac{1}{\kappa} \delta(\zeta) \right) \right) \Big|_{\zeta=0}, \qquad n = 0, \dots, N. \tag{59}$$

One of the coefficients is straightforward:

$$\omega_0^{\mathrm{F}}(\kappa) = \mathrm{F}(\tfrac{1}{\kappa}\delta(0)). \tag{60}$$

For the other ones, we are going to use Cauchy's Formula for the computation of derivatives of an analytic function, using a contour $C_R := \{\zeta \in \mathbb{C} : |\zeta| = R\}$ for some $R < 1$ that we will later determine. Since the resulting integral can be written as the integral of a periodic function, the best way to approximate it will be with the trapezoidal rule. We will use as many points in the trapezoidal rule as coefficients we want to compute in (59). The process is synthesized in the following formulas:

$$\begin{aligned} \omega_n^{\mathrm{F}}(\kappa) &= \frac{1}{2\pi \mathrm{i}} \oint_{C_R} \zeta^{-n-1} \mathrm{F}(\tfrac{1}{\kappa}\delta(\zeta)) \mathrm{d}\zeta && \text{(Cauchy formula)} \\ &= R^{-n} \int_0^1 e^{-2\pi \mathrm{i} n\theta} \mathrm{F}(\tfrac{1}{\kappa}\delta(Re^{2\pi \mathrm{i}\theta})) \mathrm{d}\zeta && \text{(parametrization } \zeta = Re^{2\pi \mathrm{i}\theta}) \end{aligned}$$

$$\approx \frac{R^{-n}}{N+1}\sum_{\ell=0}^{N}\zeta_{N+1}^{-n\ell}\mathrm{F}(\tfrac{1}{\kappa}\delta(R\zeta_{N+1}^{\ell})) \qquad \text{(trapezoidal rule: } \zeta_{N+1}^{\ell}=e^{\frac{2\pi i\ell}{N+1}}\text{)}$$

$$= \frac{R^{-n}}{N+1}\sum_{\ell=0}^{N}\zeta_{N+1}^{n\ell}\mathrm{F}(\tfrac{1}{\kappa}\delta(R\zeta_{N+1}^{-\ell})). \qquad \text{(reindexing)}$$

This formula gives us an algorithm to compute (59).

The formula

$$\omega_n^{\mathrm{F}}(\kappa) \approx \frac{R^{-n}}{N+1}\sum_{\ell=0}^{N}\zeta_{N+1}^{n\ell}\hat{\mathrm{F}}_\ell, \quad \text{where} \quad \hat{\mathrm{F}}_\ell := \mathrm{F}(\tfrac{1}{\kappa}\delta(R\zeta_{N+1}^{-\ell})),$$

will be chosen as an approximation of the CQ coefficients. The chosen value of the radius of the integration path is

$$R = \epsilon^{\frac{1}{2(N+1)}}, \tag{61}$$

where $\epsilon$ is proportional to the machine epsilon. If $\mathrm{F}(\overline{s}) = \overline{\mathrm{F}(s)}$ (see the end of Sect. 2.3) and $\delta(\overline{\zeta}) = \overline{\delta(\zeta)}$, then the vector of evaluations $\hat{\mathrm{F}}_\ell$ is Hermitian and its computation can be reduced by symmetry.

**Algorithm 2 (computation of CQ coefficients)**

In some of the algorithms to follow we will write **(Par+sym)** when the computation in that step can be reduced by symmetry and parallelized.

(a) **(Par+sym)** Evaluate

$$\hat{\mathrm{F}}_\ell := \mathrm{F}(\tfrac{1}{\kappa}\delta(R\zeta_{N+1}^{-\ell})), \qquad \ell = 0,\ldots,N.$$

(b) Apply the IDFT

$$\mathrm{F}_n := \frac{1}{N+1}\sum_{\ell=0}^{N}\hat{\mathrm{F}}_\ell\zeta_{N+1}^{n\ell}, \qquad n = 0,\ldots,N.$$

When F is matrix valued, this has to be done component by component.

(c) Scale the result

$$\omega_n^{\mathrm{F}}(\kappa) = R^{-n}\mathrm{F}_n, \qquad n = 0,\ldots,N.$$

At this step it is also possible to substitute the approximation of $\omega_0^{\mathrm{F}}(\kappa)$ by its exact value $\omega_0^{\mathrm{F}}(\kappa) = \mathrm{F}(\frac{1}{\kappa}\delta(0))$.

### 4.2.2 Forward Convolutions and Convolution Equations

To compute now

$$\sum_{m=0}^{n} \omega_m^{\mathrm{F}}(\kappa) g_{n-m} \qquad n = 0, \dots, N, \tag{62}$$

we can either apply a naive implementation formula (62) or use the zero padding strategy of Algorithm 1. Note that one of the first things we will do in Algorithm 1 will be to take the DFT of the sequence of coefficients $\omega_n^{\mathrm{F}}(\kappa)$. In Sect. 4.3, we will explore a more direct way of doing this. A similar problem arises in the solution of convolution equations

$$\omega_0^{\mathrm{F}}(\kappa) g_n = h_n - \sum_{m=1}^{n} \omega_m^{\mathrm{F}}(\kappa) g_{n-m}. \tag{63}$$

The right-hand-side of (63) contains a discrete convolution that can be evaluated using Algorithm 1 by artificially including a term $\omega_0^{\mathrm{F}}(\kappa) = 0$. Solving the CQ equations by the forward substitution scheme (63) is a particular case of what people in the know call **marching-on-in-time** (MoT) schemes.

## *4.3 All-Steps-At-Once CQ Computation*

### 4.3.1 Development of the Algorithm

Our next goal is the computation of

$$\sum_{m=0}^{n} \omega_{n-m}^{\mathrm{F}}(\kappa) g_m, \qquad n = 0, \dots, N,$$

assuming that we have approximated the CQ coefficients by

$$\omega_n^{\mathrm{F}}(\kappa) \approx \frac{R^{-n}}{N+1} \sum_{\ell=0}^{N} \hat{\mathrm{F}}_\ell \zeta_{N+1}^{\ell n}, \qquad \hat{\mathrm{F}}_\ell := \mathrm{F}(\tfrac{1}{\kappa}\delta(R\zeta_{N+1}^{-\ell})).$$

We are actually going to compute something slightly different. Because of Cauchy's Theorem

$$\omega_n^{\mathrm{F}}(\kappa) = \oint_{C_R} \zeta^{-n+1} \mathrm{F}(\zeta) \mathrm{d}\zeta = 0, \qquad \forall n \leq -1,$$

we are going to define (using the same idea as in Sect. 4.2)

$$\tilde{\omega}_n^{\mathrm{F}}(\kappa) := \frac{R^{-n}}{N+1} \sum_{\ell=0}^{N} \hat{\mathrm{F}}_\ell \zeta_{N+1}^{\ell n} \qquad n \in \mathbb{Z},$$

knowing that the approximated coefficients for $n \leq -1$ converge to zero. Then

$$\begin{aligned}
u_n &:= \sum_{m=0}^{n} \omega_{n-m}^{\mathrm{F}}(\kappa) g_m && \text{(exact convolution)} \\
&= \sum_{m=0}^{N} \omega_{n-m}^{\mathrm{F}}(\kappa) g_m && (\omega_n^{\mathrm{F}}(\kappa) = 0 \text{ for } n \leq -1) \\
&\approx \sum_{m=0}^{N} \left( \frac{R^{m-n}}{N+1} \sum_{\ell=0}^{N} \hat{\mathrm{F}}_\ell \zeta_{N+1}^{\ell(n-m)} \right) g_m && (\omega_n^{\mathrm{F}}(\kappa) \approx \tilde{\omega}_n^{\mathrm{F}}(\kappa)) \\
&= R^{-n} \left( \frac{1}{N+1} \sum_{\ell=0}^{N} \hat{\mathrm{F}}_\ell \left( \sum_{m=0}^{N} R^m g_m \zeta_{N+1}^{-m\ell} \right) \zeta_{N+1}^{\ell n} \right).
\end{aligned}$$

These formulas give Algorithm 3.

### 4.3.2 Modification for Convolution Equations

When the goal is to solve a convolution equation $f * g = h$, we can use exactly the same ideas applied to the operator $\mathrm{F}^{-1}$. To compute

$$g_n = \sum_{m=0}^{n} \omega_m^{\mathrm{F}^{-1}}(\kappa) h(t_{n-m}), \qquad n = 0, \ldots, N,$$

we approximate the coefficients

$$\omega_n^{\mathrm{F}^{-1}}(\kappa) \approx \frac{R^{-n}}{N+1} \sum_{\ell=0}^{N} \hat{\mathrm{F}}_\ell^{-1} \zeta_{N+1}^{\ell n}, \qquad \hat{\mathrm{F}}_\ell^{-1} = \mathrm{F}(\tfrac{1}{\kappa}\delta(R\zeta_{N+1}^{-\ell}))^{-1}.$$

The final expression is

$$g_n \approx R^{-n} \left( \frac{1}{N+1} \sum_{\ell=0}^{N} \hat{\mathrm{F}}_\ell^{-1} \left( \sum_{m=0}^{N} R^m h_m \zeta_{N+1}^{-m\ell} \right) \zeta_{N+1}^{\ell n} \right),$$

**Algorithm 3 (all-steps-at-once forward convolution)**

We sample the input at given discrete times

$$g_n := g(t_n), \qquad n = 0, \dots, N$$

and aim to compute an approximation of

$$u_n = \sum_{m=0}^{n} \omega^{\mathrm{F}}_{n-m}(\kappa) g_m, \qquad n = 0, \dots, N.$$

(a) Scale the data

$$h_m := R^m g_m, \qquad m = 0, \dots, N.$$

(b) Compute the DFT of the previous sequence

$$\hat{h}_\ell := \sum_{m=0}^{N} h_m \zeta_{N+1}^{-m\ell}, \qquad \ell = 0, \dots, N.$$

(c) **(Par+sym)** Apply the transfer functions in the transformed domain

$$\hat{v}_\ell := \hat{\mathrm{F}}_\ell \hat{h}_\ell, \qquad \hat{\mathrm{F}}_\ell := \mathrm{F}(\tfrac{1}{\kappa}\delta(R\zeta_{N+1}^{-\ell})).$$

(d) Compute the IDFT of the result of (c)

$$v_n := \frac{1}{N+1} \sum_{\ell=0}^{N} \hat{v}_\ell \zeta_{N+1}^{\ell n}, \qquad n = 0, \dots, N.$$

(e) Scale the result back

$$u_n = R^{-n} v_n.$$

from where it is clear that the inverses $\hat{\mathrm{F}}_\ell^{-1}$ do not need to be computed. Instead, some linear systems will be solved. The process is given in Algorithm 4.

## *4.4 Computing Pieces of a Discrete Convolution*

### 4.4.1 Pieces of a Discrete Convolution

Our next goal is the computation of quantities

$$g_n := \sum_{m=0}^{Q} \tilde{\omega}^{\mathrm{F}}_{n-m}(\kappa) u_m, \qquad n = Q+1, \dots, M, \tag{64}$$

**Algorithm 4 (all-steps-at-once convolution equation)**

We sample the input at given discrete times

$$h_n := h(t_n), \qquad n = 0, \dots, N$$

and aim to compute an approximation of the solution of

$$\sum_{m=0}^{n} \omega^{\mathrm{F}}_{n-m}(\kappa) g_m = h_n, \qquad n = 0, \dots, N.$$

(a) Scale the data

$$v_m := R^m h_m, \qquad m = 0, \dots, N.$$

(b) Compute the DFT of the previous sequence $\hat{v}_\ell$, $\ell = 0, \dots, N$.
(c) **(Par+sym)** Solve equations in the transformed domain

$$\hat{\mathrm{F}}_\ell \hat{w}_\ell := \hat{v}_\ell, \qquad \hat{\mathrm{F}}_\ell := \mathrm{F}(\tfrac{1}{\kappa}\delta(R\zeta_{N+1}^{-\ell})).$$

(d) Compute the IDFT of the result of (c) $w_\ell$, $\ell = 0, \dots, N$.
(e) Scale the result back

$$g_n = R^{-n} w_n, \qquad n = 0, \dots, N.$$

where

$$\tilde{\omega}^{\mathrm{F}}_n(\kappa) = \frac{R^{-n}}{N+1} \sum_{\ell=0}^{N} \hat{\mathrm{F}}_\ell \zeta_{N+1}^{n\ell}, \qquad \hat{\mathrm{F}}_\ell := \mathrm{F}(\tfrac{1}{\kappa}\delta(R\zeta_{N+1}^{-\ell})), \tag{65}$$

for a given $N \geq M$, which can be chosen as $N = 2M$ for example. Note that (64) uses $u_0, \dots, u_Q$ and $\tilde{\omega}^{\mathrm{F}}_n(\kappa)$ for $n = 1, \dots, M$. We then proceed as follows:

$$\begin{aligned}
\tilde{g}_k &:= g_{k+Q+1} && (k = 0, \dots, M-Q-1) \\
&= \sum_{m=0}^{Q} \left( \frac{R^{m-k-Q-1}}{N+1} \sum_{\ell=0}^{N} \hat{\mathrm{F}}_\ell \zeta_{N+1}^{\ell(k+Q+1-m)} \right) u_m \\
&= R^{-k-Q-1} \left( \frac{1}{N+1} \sum_{\ell=0}^{N} \zeta_{N+1}^{\ell(Q+1)} \hat{\mathrm{F}}_\ell \left( \sum_{m=0}^{Q} \zeta_{N+1}^{-\ell m} R^m u_m \right) \zeta_{N+1}^{\ell k} \right) \\
&= R^{-k-Q-1} \left( \frac{1}{N+1} \sum_{\ell=0}^{N} \left( \zeta_{N+1}^{\ell(Q+1)} \hat{\mathrm{F}}_\ell \left( \sum_{m=0}^{N} \zeta_{N+1}^{-\ell m} w_m \right) \right) \zeta_{N+1}^{\ell k} \right),
\end{aligned}$$

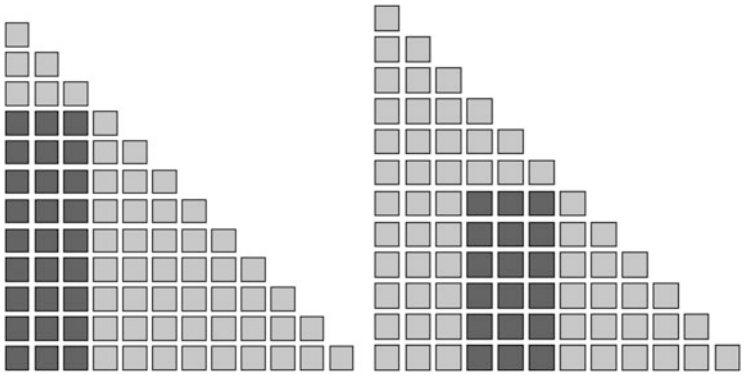

**Fig. 1** The *leftmost* figure shows a typical convolution piece as computed by Algorithm 5. Three elements of the sequence of data are input and nine are output. The figure on the *right* is a simple variation. By a shifting of indices in the input data a different block of the Toeplitz matrix can be accessed. Note that each of the elements is a matrix (or an operator if we have not discretized the original operator)

where

$$w_m := \begin{cases} R^m u_m, & 0 \le m \le Q, \\ 0, & Q+1 \le m \le N. \end{cases} \tag{66}$$

This computation is depicted in Fig. 1.

**Algorithm 5 (computation of a piece of a convolution)**

Our goal is to compute (64). The value $N$ in (65) is given as a parameter. The data $u_m, m = 0, \dots, Q$ has already been sampled.

(a) Scale the data and add zeros following (66).
(b) Compute the DFT of the vector in (a) $\hat{w}_\ell$, $\ell = 0, \dots, N$.
(c) **(Par+sym)** Apply the operators

$$\hat{h}_\ell := \zeta_{N+1}^{\ell(Q+1)} \hat{\mathrm{F}}_\ell \hat{w}_\ell, \qquad \ell = 0, \dots, N.$$

(d) Compute the IDFT of the sequence in (c) $h_k$, $k = 0, \dots, N$.
(e) Scale and chop the resulting sequence

$$\tilde{g}_k := R^{-k-Q-1} h_k, \qquad k = 0, \dots, M-Q-1.$$

(f) Change indices

$$g_n := \tilde{g}_{n-Q-1}, \qquad n = Q+1, \dots, M.$$

If we skip step (f), what we have computed is

$$\tilde{g}_k := \sum_{m=0}^{Q} \tilde{\omega}^{\mathrm{F}}_{k+Q+1-m}(\kappa) u_m, \qquad k = 0, \ldots, M-Q-1,$$

which can be understood as the formal product with a piece of a Toeplitz matrix

$$\begin{bmatrix} \tilde{g}_0 \\ \tilde{g}_1 \\ \vdots \\ \tilde{g}_{M-Q-1} \end{bmatrix} = \begin{bmatrix} \tilde{\omega}^{\mathrm{F}}_{Q+1}(\kappa) & \tilde{\omega}^{\mathrm{F}}_{Q}(\kappa) & \ldots & \tilde{\omega}^{\mathrm{F}}_{1}(\kappa) \\ \tilde{\omega}^{\mathrm{F}}_{Q+2}(\kappa) & \tilde{\omega}^{\mathrm{F}}_{Q+1}(\kappa) & \ldots & \tilde{\omega}^{\mathrm{F}}_{2}(\kappa) \\ \tilde{\omega}^{\mathrm{F}}_{Q+3}(\kappa) & \tilde{\omega}^{\mathrm{F}}_{Q+2}(\kappa) & \ldots & \tilde{\omega}^{\mathrm{F}}_{3}(\kappa) \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{\omega}^{\mathrm{F}}_{M}(\kappa) & \tilde{\omega}^{\mathrm{F}}_{M-1}(\kappa) & \ldots & \tilde{\omega}^{\mathrm{F}}_{M-Q}(\kappa) \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_Q \end{bmatrix}.$$

## 4.5 Recursive Strategies

### 4.5.1 Small Systems

We are going to next include a new parameter $\underline{n}$ corresponding to sizes of lower triangular systems (convolution equations) that we want to solve directly:

$$\sum_{m=0}^{n} \omega_{n-m} g_m = u_n, \qquad n = 0, \ldots, \underline{n}. \tag{67}$$

From now on, we are going to ignore where the coefficients $\omega_n$ are coming from. They are operators, $\omega_0$ being invertible, or simply square matrices. We can write in matrix form with help of the matrix

$$\Omega_{\underline{n}} := \begin{bmatrix} \omega_0 & & & \\ \omega_1 & \omega_0 & & \\ \vdots & \ddots & \ddots & \\ \omega_{\underline{n}-1} & \ldots & \omega_1 & \omega_0 \end{bmatrix}.$$

### 4.5.2 Look-Ahead Method

With this strategy, we solve groups of $\underline{n}$ equations and then subtract the contribution of the newly computed unknowns from the right-hand-side of the system. The method can then be understood as block recursive forward substitution.

**Algorithm 6 (look-ahead solution of CQ equations)**

The goal is the solution of

$$\sum_{m=0}^{n} \omega_{n-m} g_m = u_n, \qquad n = 0, \ldots, N.$$

We give a parameter $\underline{n}$ for the size of the small systems and break the list of indices in the form

$$\underbrace{0, \ldots, \underline{n}-1}_{b(1)}, \underbrace{\underline{n}, \ldots, 2\underline{n}-1}_{b(2)}, \ldots, \underbrace{(k-1)\underline{n}, \ldots, k\underline{n}-1}_{b(k)}, \underbrace{k\underline{n}, \ldots, N}_{\text{remainder}},$$

so that

$$b(i) = \{(i-1)\underline{n}, \ldots, i\underline{n}-1\} \qquad i = 1, \ldots, k, \quad \text{where } k = \lfloor N/\underline{n} \rfloor,$$

is a generic block. The algorithm loops in $i = 1, \ldots, k$ in the following form:

(a) Make a copy of a block of data $\boldsymbol{v} = \boldsymbol{u}_{b(i)}$ Solve

$$\Omega_{\underline{n}} \boldsymbol{h} = \boldsymbol{v},$$

and copy the result $\boldsymbol{q}_{b(i)} = \boldsymbol{h}$.

(b) Compute the following piece of convolution

$$r_n = \sum_{m=0}^{\underline{n}-1} \omega_{n-m} h_m, \qquad n = \underline{n}, \ldots, N - (i-1)\underline{n}.$$

(c) Correct the right-hand-side

$$u_n = u_n - r_{n-(i-1)\underline{n}} \qquad n = i\underline{n}, \ldots, N.$$

Finally, after the loop is finished, we have to solve for the tail (because of $N+1$ not being a multiple of $\underline{n}$):

$$\sum_{m=0}^{n} \omega_{n-m} g_{k\underline{n}+m} = u_{k\underline{n}+m}, \qquad m = 0, \ldots, N - k\underline{n}.$$

### 4.5.3 Recursive Methods

Another option to deal with the CQ equations is the development of a recursive algorithm, based on the break down of a triangular system into two pieces of the same size and the square block that interconnects them. We will not develop this algorithm any further. A pictorial representation, side by side with the look-ahead algorithm, is given in Fig. 2.

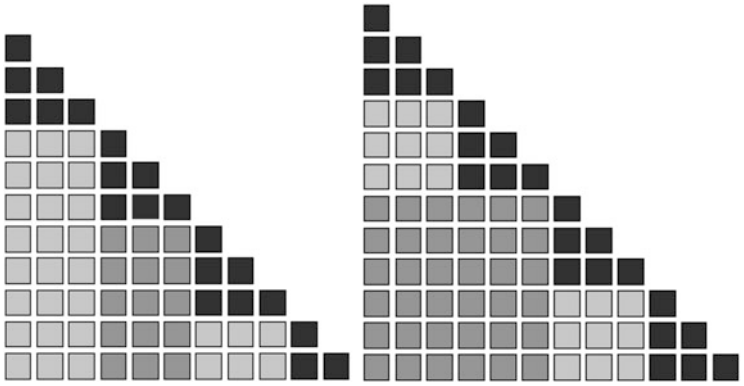

**Fig. 2** This cartoon shows the basic ideas of a look-ahead and a recursive strategy side-by-side. The *black blocks* of equations are inverted directly by forward substitution. In the look-ahead strategy, once a group of unknowns has been computed, its influence on the right-hand-side is immediately taken into account. The recursive idea breaks down the triangular system into smaller triangular systems joined with square blocks

### *4.6 Credits*

An excellent introduction to the algorithms for CQ is given in [6]. The idea of using the trapezoidal rule on a Cauchy integral representation in order to compute the CQ coefficients appears already in the original papers by Lubich [23, 25]. The simultaneous computation of all time steps is introduced in a paper by Lehel Banjai and Stefan Sauter [5] for the specific problem of the wave equation (see Sect. 5), based on algorithms developed in [18]. The choice of $R$ in the integration contour (61) is justified in [24]. A very thorough study of the dependence of the radius $R$ and of the number of points $N$ in the trapezoidal rule used to approximate the CQ weights is developed in [10].

## 5 Integral Equations for Waves

In this section we are going to explain how the ideas on Laplace transforms, convolution operators and equations, and CQ discretizations can be used for a problem of scattering of acoustic waves by an obstacle. This section is considerably heavier with respects to Sobolev spaces. The reader should have at least some working knowledge of the Sobolev space $H^1(O)$ for an open set $O$ and of the spaces $H^{\pm 1/2}(\partial O)$ for an open set $O$ which lies on one side of its Lipschitz boundary.

The space

$$H^1_\Delta(O) := \{u \in H^1(O) \,:\, \Delta u \in L^2(O)\},$$

endowed with its natural norm, will play a key role in this section. Note that from this space we can define three bounded operators:

$$\gamma : H^1_\Delta(\partial O) \to H^{1/2}(\partial O), \qquad \partial_\nu : H^1_\Delta(\partial O) \to H^{-1/2}(\partial O), \qquad \Delta : H^1_\Delta(\partial O) \to L^2(\partial O),$$

the first two of which correspond to the trace and the normal derivative.

## *5.1 A Scattering Problem*

### 5.1.1 Incident Waves

Let $\Omega_- \subset \mathbb{R}^3$ be a connected domain, lying on one side of its connected Lipschitz boundary $\Gamma$. Let $\Omega_+ := \mathbb{R}^3 \setminus \overline{\Omega_-}$. At time $t = 0$ an incident wave approaches the obstacle $\Gamma$. The incident wave can be described as a solution of the wave equation in free space. For instance, a plane wave

$$u^{\text{inc}}(\mathbf{z}, t) := \psi(c(t - t_{\text{lag}}) - \mathbf{z} \cdot \mathbf{d}), \qquad |\mathbf{d}| = 1, \quad c > 0, \tag{68}$$

solves

$$c^{-2}\ddot{u}^{\text{inc}} = \Delta u^{\text{inc}} \qquad \text{in } \mathbb{R}^3 \times (-\infty, \infty). \tag{69}$$

Equation (69) is satisfied in a classical way when the signal is smooth enough, for instance, when $\psi \in C^2(\mathbb{R})$. When $\psi$ is just continuous, and even when it is only locally integrable, (69) can be understood in a weak sense of distributions in four variables (an approach we have decided not to take), or even more surprisingly in a sense of distributions of the time variable, with values in some functions spaces. We will discuss this later. We need to make some assumptions on the plane wave (68) in order for the scattering process to be physically meaningful. For instance, we can assume that $\psi$ is causal, $\Omega_- \subset B(\mathbf{0}; R) := \{\mathbf{z} \in \mathbb{R}^3 : |\mathbf{z}| < R\}$ and $t_{\text{lag}} > R/c$. This ensures that at time $t = 0$, the wave, which is moving in the direction $\mathbf{d}$, has not reached the area of the space where the obstacle $\Omega_-$ is placed. A spherical incident wave

$$u^{\text{inc}}(\mathbf{z}, t) := \frac{\psi(ct - |\mathbf{z} - \mathbf{z}_{\text{sc}}|)}{4\pi|\mathbf{z} - \mathbf{z}_{\text{sc}}|}, \tag{70}$$

can also be defined for any causal $\psi$ and source point $\mathbf{z}_{\text{sc}} \notin \overline{\Omega}_-$. Instead of (69), this $u^{\text{inc}}$ is a causal solution of an equation

$$c^{-2}\ddot{u}^{\text{inc}} = \Delta u^{\text{inc}} + f, \tag{71}$$

where $f$ is formally a distribution that is supported in the source point. Let us warn the reader that, even if it is possible to give a completely rigorous presentation of the meaning of the equation satisfied by the incident wave, we will only need its values on $\Gamma$ and we can make much simpler while rigorous arguments for those values.

### 5.1.2 Transient Scattering by a Sound-Soft Obstacle

Let us then assume that we have an incident wave satisfying (71) for all times and for a given $f$. We do not need to know $f$. Knowing that $f$ exists is enough. The total wave is a function $u^{\text{tot}}$ satisfying

$$c^{-2}\ddot{u}^{\text{tot}} = \Delta u^{\text{tot}} + f \qquad \text{in } \Omega_+ \times (-\infty, \infty), \tag{72a}$$

with a boundary condition

$$u^{\text{tot}} = 0 \qquad \text{on } \Gamma \times (-\infty, \infty), \tag{72b}$$

and the assumption

$$u^{\text{tot}} - u^{\text{inc}} \quad \text{is causal.} \tag{72c}$$

Instead of working with the total wave field, we will think in terms of the **scattered wave field**

$$u = u^{\text{tot}} - u^{\text{inc}},$$

which satisfies a homogeneous wave equation

$$c^{-2}\ddot{u} = \Delta u \qquad \text{in } \Omega_+ \times (-\infty, \infty), \tag{73a}$$

a non-homogeneous boundary condition

$$u + u^{\text{inc}} = 0 \qquad \text{on } \Gamma \times (-\infty, \infty), \tag{73b}$$

and the assumption

$$u \quad \text{is causal.} \tag{73c}$$

It is interesting to note that our assumptions for the incident wave (causality of the signal in (70), and more complicated geometric assumptions on (68)) were destined to ensure that the restriction of $u^{\text{inc}}$ to $\Gamma$ is a causal function. Therefore, instead of (73), we can think of looking for a causal distribution with values in

$$H^1_\Delta(\Omega_+) := \{v \in H^1(\Omega_+) \,:\, \Delta v \in L^2(\Omega_+)\}.$$

satisfying

$$c^{-2}\ddot{u} = \Delta u \quad \text{in } H^1_\Delta(\Omega_+), \qquad \gamma u = \beta \quad \text{in } H^{1/2}(\Gamma). \tag{74}$$

Let us first clarify the meaning of equations (74). The steady state operator $\Delta : H^1_\Delta(\Omega_+) \to L^2(\Omega_+)$ can be applied to any $H^1_\Delta(\Omega_+)$-valued distribution $u$, producing an $L^2(\Omega_+)$-valued distribution $\Delta u$. Similarly, since $u$ is $H^1_\Delta(\Omega_+)$-valued, so is $\ddot{u}$, and therefore, we can understand $\ddot{u}$ as an $L^2(\Omega_+)$-valued distribution, by using the steady-state embedding operator $H^1_\Delta(\Omega_+) \to L^2(\Omega_+)$. The second part of (74) arises from taking the trace operator $\gamma : H^1_\Delta(\Omega_+) \to H^{1/2}(\Gamma)$ and comparing the $H^{1/2}(\Gamma)$-valued distribution $\gamma u$ with a given causal $H^{1/2}(\Gamma)$-valued distribution $\beta$. At this time, it is clear what we need from the incident wave: as long as the boundary values of $u^{\text{inc}}$ define a causal distribution with values in $H^{1/2}(\Gamma)$, we can give meaning to equations (74).

## 5.2 *The Acoustic Single Layer Potential*

### 5.2.1 A What-It-Does Definition

We are first going to formally introduce the $d$-dimensional acoustic single layer potential and two related boundary integral operators not through mathematical expressions, but through their properties. To do this we need two jump operators

$$[\![\gamma u]\!] := \gamma^- u - \gamma^+ u, \qquad [\![\partial_\nu u]\!] := \partial_\nu^- u - \partial_\nu^+ u.$$

Both of them are well defined in $H^1_\Delta(\mathbb{R}^d \setminus \Gamma)$. Let now $\eta$ be a causal $H^{-1/2}(\Gamma)$-valued distribution, which will be referred to as a **density**. We then consider the following wave propagation problem in free space

$$\ddot{u} = \Delta u \qquad \text{in } L^2(\mathbb{R}^d \setminus \Gamma), \tag{75a}$$

$$[\![\gamma u]\!] = 0 \qquad \text{in } H^{1/2}(\Gamma), \tag{75b}$$

$$[\![\partial_\nu u]\!] = \eta \qquad \text{in } H^{-1/2}(\Gamma). \tag{75c}$$

The mathematically savvy reader will undoubtedly find silly that we have written $L^2(\mathbb{R}^d \setminus \Gamma)$ instead of $L^2(\mathbb{R}^d)$ in (75a). We do it just to emphasize that we are applying the Laplacian as an operator $\Delta : H^1_\Delta(\mathbb{R}^d \setminus \Gamma) \to L^2(\mathbb{R}^d \setminus \Gamma)$. Note also that in (75a) we consider the wave equation on both sides of the boundary of the scatterer, at speed $c = 1$. For the moment being, we are going to admit that problem (75) is solvable, and we then (formally) denote

$$u =: \mathcal{S} * \eta, \qquad \gamma u =: \mathcal{V} * \eta = \gamma^{\pm}(\mathcal{S} * \eta), \qquad \tfrac{1}{2}(\partial_\nu^- u + \partial_\nu^+ u) =: \mathcal{J} * \eta. \tag{76}$$

The convolution symbol in the definitions (76) will be justified below. By definition

$$\partial_\nu^+ u = -[\![\partial_\nu u]\!] + \tfrac{1}{2}(\partial_\nu^- u + \partial_\nu^+ u) = -\tfrac{1}{2}\eta + \mathcal{J} * \eta. \tag{77}$$

### 5.2.2 The Laplace Transform Way

The simplest way to justify problem (75), and the three definitions (76) that follow from it, is to take the Laplace transform. Let then $\mathrm{E} = \mathcal{L}\{\eta\}$ and consider the problem of finding $\mathrm{U}(s) \in H^1_\Delta(\mathbb{R}^d \setminus \Gamma)$ satisfying

$$s^2\mathrm{U}(s) = \Delta\mathrm{U}(s) \qquad \text{in } \mathbb{R}^d \setminus \Gamma, \text{ (i.e., as functions of } L^2(\mathbb{R}^d \setminus \Gamma)) \tag{78a}$$

$$[\![\gamma\mathrm{U}(s)]\!] = 0, \qquad \text{(as functions in } H^{1/2}(\Gamma)) \tag{78b}$$

$$[\![\partial_\nu\mathrm{U}(s)]\!] = \mathrm{E}(s). \qquad \text{(as functions in } H^{-1/2}(\Gamma)) \tag{78c}$$

Equations (78) are uniquely solvable for values $s \in \mathbb{C}_+$. Actually the theory of time-harmonic acoustic layer potentials gives an explicit solution to (78). We first introduce the fundamental solution of the Laplace resolvent operator $\Delta - s^2$,

$$\Phi(s;\mathbf{r}) := \begin{cases} \frac{\mathrm{i}}{4}H_0^{(1)}(-\mathrm{i}s|\mathbf{r}|), & \text{when } d = 2, \\ \dfrac{e^{-s|\mathbf{r}|}}{4\pi|\mathbf{r}|}, & \text{when } d = 3. \end{cases}$$

With it, and for a given density $\xi \in H^{-1/2}(\Gamma)$, we define the potential

$$\mathrm{S}(s)\xi := \int_\Gamma \Phi(s;|\cdot - \mathbf{y}|)\xi(\mathbf{y})\mathrm{d}\Gamma(\mathbf{y}) \;:\; \mathbb{R}^d \setminus \Gamma \to \mathbb{C} \tag{79a}$$

and two integral operators

$$\mathrm{V}(s)\xi := \int_\Gamma \Phi(s;|\cdot - \mathbf{y}|)\xi(\mathbf{y})\mathrm{d}\Gamma(\mathbf{y}) \qquad : \Gamma \to \mathbb{C}, \tag{79b}$$

$$\mathrm{J}(s)\xi := \boldsymbol{\nu} \cdot \int_\Gamma \nabla\Phi(s;|\cdot - \mathbf{y}|)\xi(\mathbf{y})\mathrm{d}\Gamma(\mathbf{y}) : \Gamma \to \mathbb{C}. \tag{79c}$$

Given the overall lack of regularity, the definitions (79) have to be taken with a grain of salt. For instance, for a given $\mathbf{z} \in \mathbb{R}^d \setminus \Gamma$, the definition (79a) of $(\mathrm{S}(s)\xi)(\mathbf{z})$ can always be understood as a duality product of $\xi \in H^{-1/2}(\Gamma)$ with $\Phi(s;|\mathbf{z} - \cdot|) \in H^{1/2}(\Gamma)$, which admits an integral expression like (79a) when $\xi \in L^2(\Gamma)$. The definitions (79b) and (79c) are completely reasonable for smooth enough $\xi$ and $\Gamma$.

Otherwise they need to be extended using density arguments. What really matters now is that

$$\mathrm{U}(s) = \mathrm{S}(s)\mathrm{E}(s), \qquad \gamma\mathrm{U}(s) = \mathrm{V}(s)\mathrm{E}(s), \qquad \tfrac{1}{2}(\partial_\nu^- \mathrm{U}(s) + \partial_\nu^+ \mathrm{U}(s)) = \mathrm{J}(s)\mathrm{E}(s) \tag{80}$$

is the solution to (78). For future reference (and mimicking (77)) we are going to care about the operator

$$\mathrm{E}(s) \longmapsto -\tfrac{1}{2}\mathrm{E}(s) + \mathrm{J}(s)\mathrm{E}(s). \tag{81}$$

### 5.2.3 Some Sobolev Space Notation

Given an open set $O$, we will write

$$\|u\|_O^2 := \int_O |u|^2, \qquad \|u\|_{1,O}^2 := \|u\|_O^2 + \|\nabla u\|_O^2.$$

The norm in the trace space $H^{1/2}(\Gamma)$ will be denoted $\|\cdot\|_{1/2,\Gamma}$, and its dual norm is

$$\|\eta\|_{-1/2,\Gamma} := \sup_{0\neq\phi\in H^{1/2}(\Gamma)} \frac{|\langle \eta, \phi\rangle_\Gamma|}{\|\phi\|_{1/2,\Gamma}},$$

where $\langle \eta, \phi\rangle_\Gamma$ is the duality bracket.

### 5.2.4 From Laplace to Time

It does not take very long (the Lax-Milgram theorem with some carefully crafted bounds taking care of the parameter $s$) to see that for every $s \in \mathbb{C}_+$, the solution of (78) exists, and we can bound

$$\|\mathrm{U}(s)\|_{1,\mathbb{R}^d\setminus\Gamma} \leq C\frac{|s|}{\sigma\underline{\sigma}^2}\|\mathrm{E}(s)\|_{-1/2,\Gamma}, \tag{82a}$$

$$\|\partial_\nu^\pm \mathrm{U}(s)\|_{-1/2,\Gamma} \leq C\frac{|s|^{3/2}}{\sigma\underline{\sigma}^{3/2}}\|\mathrm{E}(s)\|_{-1/2,\Gamma}, \tag{82b}$$

where

$$\sigma := \operatorname{Re} s, \qquad \underline{\sigma} := \min\{\sigma, 1\}.$$

From (80) and (82) we can derive the following bounds (depending on $s$)

$$\|\mathrm{S}(s)\|_{H^{-1/2}(\Gamma)\to H^1(\mathbb{R}^d)} \leq C\frac{|s|}{\sigma\underline{\sigma}^2}, \tag{83a}$$

$$\|\mathrm{V}(s)\|_{H^{-1/2}(\Gamma)\to H^{1/2}(\Gamma)} \leq C\frac{|s|}{\sigma\underline{\sigma}^2}, \tag{83b}$$

$$\|\mathrm{J}(s)\|_{H^{-1/2}(\Gamma)\to H^{-1/2}(\Gamma)} \leq C\frac{|s|^{3/2}}{\sigma\underline{\sigma}^{3/2}}. \tag{83c}$$

Using the results of Sect. 2.4 (see specifically conditions (13) and (14)) we can show that there exist causal operator valued distributions $\mathcal{S}$, $\mathcal{V}$, and $\mathcal{J}$, whose Laplace transforms are S, V, and J respectively. This long process through the Laplace domain gives a precise meaning to the time-domain convolution operators (76) and (77).

### 5.2.5 Time Domain Expressions

Explicit expressions for the operators $\mathcal{S} * \eta$, $\mathcal{V} * \eta$, and $\mathcal{J} * \eta$ are available in two and three dimensions. Note that with the CQ approach, there is no need for them, since only their Laplace transform is ever used. For instance in three dimensions

$$(\mathcal{S} * \eta)(t) := \int_\Gamma \frac{\eta(\mathbf{y}; t - |\cdot - \mathbf{y}|)}{4\pi|\cdot - \mathbf{y}|}\mathrm{d}\Gamma(\mathbf{y}) : \mathbb{R}^3 \setminus \Gamma \to \mathbb{R}, \tag{84}$$

is the retarded (or Huygens) potential. It is interesting to remark that while we know an integral expression for $\mathcal{S} * \eta$, it is not that easy to understand what $\mathcal{S}$ is. Just for the sake of illustration, let us give an idea how $\mathcal{S}$ looks. Consider the operator

$$(\mathcal{H}(t)\xi)(\mathbf{z}) := \int_{\Gamma\cap B(\mathbf{z},t)} \frac{\xi(\mathbf{y})}{4\pi|\mathbf{z}-\mathbf{y}|}\mathrm{d}\Gamma(\mathbf{y}),$$

where $B(\mathbf{z}, t) := \{\mathbf{y} \in \mathbb{R}^3 : |\mathbf{y} - \mathbf{z}| < t\}$. Using arguments in Fourier analysis it is possible to show that $\mathcal{H}$ is a continuous causal function of $t$ with values in the space of bounded linear operators from $H^{-1/2}(\Gamma)$ to $H^1(\mathbb{R}^3)$. Its distributional time derivative is $\mathcal{S}$.

## 5.3 A Boundary Integral Equation for Scattering

### 5.3.1 The Potential Ansatz

We want to find causal distributional solutions to

$$c^{-2}\ddot{u} = \Delta u \qquad \text{in } L^2(\Omega_+) \tag{85a}$$

$$\gamma u = \beta \qquad \text{in } H^{1/2}(\Gamma). \tag{85b}$$

In order to write an integral representation we need to modify the speed of the wave operators of Sect. 5.2. We thus define

$$\begin{aligned}\mathcal{S}_c &:= \mathcal{L}^{-1}\{\mathrm{S}(s/c)\},\\ \mathcal{V}_c &:= \mathcal{L}^{-1}\{\mathrm{V}(s/c)\} = \gamma\mathcal{S}_c,\\ \mathcal{J}_c &:= \mathcal{L}^{-1}\{\mathrm{J}(s/c)\} = \tfrac{1}{2}\left(\partial_\nu^+\mathcal{S}_c + \partial_\nu^-\mathcal{S}_c\right).\end{aligned}$$

We represent the solution of (85) by means of a single layer potential

$$u = \mathcal{S}_c * \eta \tag{86a}$$

for a causal density $\eta$ (an $H^{-1/2}(\Gamma)$-valued distribution) to be determined. We then impose the boundary condition to obtain an integral equation for the density

$$\mathcal{V}_c * \eta = \beta. \tag{86b}$$

The exterior normal derivative can then be computed as a postprocessing of $\eta$:

$$\lambda = -\tfrac{1}{2}\eta + \mathcal{J}_c * \eta = \partial_\nu^+ u = (\partial_\nu^+\mathcal{S}_c) * \eta. \tag{86c}$$

### 5.3.2 The Transfer Functions

From the point of view of data, equations (86) involve three transfer functions

$$\mathrm{V}(s/c)^{-1}, \qquad \text{(convolution equation)} \tag{87a}$$

$$\mathrm{S}(s/c)\mathrm{V}(s/c)^{-1}, \qquad \text{(conv eqn followed by a forward conv)} \tag{87b}$$

$$(-\tfrac{1}{2}\mathrm{I} + \mathrm{J}(s/c))\mathrm{V}(s/c)^{-1}. \qquad \text{(same)} \tag{87c}$$

We will come back to how these transfer functions behave as functions of $s \in \mathbb{C}_+$. For the moment, let us just understand what they do in the Laplace domain. If we focus on the problem

$$\left(\tfrac{s}{c}\right)^2 \mathrm{U} = \Delta U, \qquad \gamma\mathrm{U} = \mathrm{B},$$

and think of the integral representation $\mathrm{U} = \mathrm{S}(s/c)\mathrm{E}$, the associated operators are:

$$\mathrm{V}(s/c)^{-1}, \qquad \text{(solution of integral equation)}$$

$$\mathrm{S}(s/c)\mathrm{V}(s/c)^{-1}, \qquad \text{(solution of exterior Dirichlet problem)}$$

$$(-\tfrac{1}{2}\mathrm{I} + \mathrm{J}(s/c))\mathrm{V}(s/c)^{-1}. \qquad \text{(Dirichlet to Neumann operator)}$$

### 5.3.3 Galerkin Semidiscretization in Space

Let us now fix a finite dimensional subspace $X_h \subset H^{-1/2}(\Gamma)$. For instance, if $\Gamma$ is a polygon, we can admit any triangulation of $\Gamma$ and the space of piecewise constant functions with respect to this triangulation. The semidiscrete version of (86) is the search for a causal $X_h$-valued distribution $\eta^h$ satisfying

$$\langle \mu^h, \mathcal{V}_c * \eta^h - \beta \rangle_\Gamma = 0 \qquad \forall \mu^h \in X_h, \tag{88a}$$

postprocessed to obtain a potential

$$u^h := \mathcal{S}_c * \eta^h \tag{88b}$$

and the associated Neumann data

$$\lambda^h := -\tfrac{1}{2}\eta^h + \mathcal{J}_c * \eta^h. \tag{88c}$$

It is inherent to the fact that we are discretizing using a boundary integral representation to see that $u^h$ is exactly a causal solution of the wave equation $c^{-2}\ddot{u}^h = \Delta u^h$. What has changed is the level of satisfaction of the boundary condition.

### 5.3.4 The Semidiscrete System in the Time Domain

It might help the reader understand the difficulties of dealing with equations like (88) to see what happens in the simplest case. Assume that $\Gamma$ has been partitioned into $J$ non-overlapping elements $\{\Gamma_1, \ldots, \Gamma_J\}$ and that we take

$$X_h = \{\mu^h : \Gamma \to \mathbb{R} \,:\, \mu^h|_{\Gamma_j} \in \mathcal{P}_0 \quad \forall j\} = \operatorname{span}\{\chi_{\Gamma_1}, \ldots, \chi_{\Gamma_J}\},$$

where $\mathcal{P}_0$ is the set of constant functions and we have used the symbol $\chi_{\Gamma_j}$ for the characteristic function of the element $\Gamma_j$. Let us assume that $\eta^h$ is actually a function $\mathbb{R} \to X_h$. We can then write

$$\eta^h = \sum_{j=1}^{J} \eta_j(t)\chi_{\Gamma_j}(\mathbf{y})$$

for unknown causal scalar functions $\eta_j$. The equations (88a) are equivalent to

$$\sum_{j=1}^{J} \int_{\Gamma_i}\int_{\Gamma_j} \frac{\eta_j(t - c^{-1}|\mathbf{x}-\mathbf{y}|)}{4\pi|\mathbf{x}-\mathbf{y}|} \mathrm{d}\Gamma(\mathbf{y})\mathrm{d}\Gamma(\mathbf{x}) = \int_{\Gamma_i} \beta(t,\mathbf{x})\mathrm{d}\Gamma(\mathbf{x}) \qquad i = 1, \ldots, J. \tag{89}$$

This system is quite difficult to deal with in the time domain. The unknowns are subject to a continuum of delays which are then integrated against a weakly singular kernel. However, the Laplace transform of (89) reveals much more about the problem. If $\mathrm{E}_j = \mathcal{L}\{\eta_j\}$ and $\mathrm{B} = \mathcal{L}\{\beta\}$, equations (89) are transformed into

$$\sum_{j=1}^{J} \left( \int_{\Gamma_i} \int_{\Gamma_j} \frac{e^{-\frac{s}{c}|\mathbf{x}-\mathbf{y}|}}{4\pi|\mathbf{x}-\mathbf{y}|} \mathrm{d}\Gamma(\mathbf{y}) \mathrm{d}\Gamma(\mathbf{x}) \right) \mathrm{E}_j(s) = \int_{\Gamma_i} \mathrm{B}(s,\mathbf{x}) \mathrm{d}\Gamma(\mathbf{x}). \tag{90}$$

It is interesting to note how the unknowns have left the integrals after the Laplace transform has been taken. The system (90) corresponds to the $X_h$ Galerkin discretization of

$$\int_{\Gamma} \frac{e^{-\frac{s}{c}|\cdot-\mathbf{y}|}}{4\pi|\cdot-\mathbf{y}|} \mathrm{E}(s,\mathbf{y}) \mathrm{d}\Gamma(\mathbf{y}) = \mathrm{B}(s,\cdot) \qquad \text{in } \Gamma,$$

which is an integral form of the equation $\mathrm{V}(s/c)\mathrm{E}(s) = \mathrm{B}(s)$.

## 5.4 *Full Discretization*

### 5.4.1 First Space, Then Time

In Sect. 5.3 we have written a semidiscrete form of an integral formulation for the exterior scattering problem (by a sound-soft obstacle). These are the equations (88) in the Laplace domain: we look for $\mathrm{E}^h : \mathbb{C}_+ \to X_h$ satisfying

$$\langle \mu^h, \mathrm{V}(s/c)\mathrm{E}^h(s) - \mathrm{B}(s) \rangle_\Gamma = 0 \qquad \forall \mu^h \in X_h, \quad \forall s \in \mathbb{C}_+. \tag{91a}$$

The solution of this equation is then postprocessed to compute a potential

$$\mathrm{U}^h(s) = \mathrm{S}(s/c)\mathrm{E}^h(s) \tag{91b}$$

and the exterior normal derivative

$$\Lambda^h(s) = -\tfrac{1}{2}\mathrm{E}^h(s) + \mathrm{J}(s/c)\mathrm{E}^h(s). \tag{91c}$$

It is very easy to describe a fully discrete method for (91) (that is, (88)) using this form. Note that there are three convolutions involved: one of them is a convolution equation and two other are needed for postprocessing of the solution. We just need to substitute the continuous differentiation parameter $s$ by $s_\kappa = \frac{1}{\kappa}\delta(e^{-s\kappa})$ within all the operators above: we now look for $\mathrm{E}^h_\kappa : \mathbb{C}_+ \to X_h$ satisfying

$$\langle \mu^h, \mathrm{V}(s_\kappa/c)\mathrm{E}^h_\kappa(s) - \mathrm{B}(s) \rangle_\Gamma = 0 \qquad \forall \mu^h \in X_h, \quad \forall s \in \mathbb{C}_+, \tag{92a}$$

and postprocess twice

$$\mathrm{U}_\kappa^h(s) = \mathrm{S}(s_\kappa/c)\mathrm{E}_\kappa^h(s), \tag{92b}$$

$$\Lambda_\kappa^h(s) = -\tfrac{1}{2}\mathrm{E}_\kappa^h(s) + \mathrm{J}(s_\kappa/c)\mathrm{E}_\kappa^h(s). \tag{92c}$$

### 5.4.2 A Look at the Associated Integral Operators

We are now going to pay attention to the integral equation in (92). Let us forget for a while about the space discretization (we will see later on how this is a legitimate move). We have an integral operator

$$\eta \mapsto \mathrm{V}(s/c)\eta = \int_\Gamma \frac{e^{-\frac{s}{c}|\cdot-\mathbf{y}|}}{4\pi|\cdot-\mathbf{y}|}\eta(\mathbf{y})\mathrm{d}\Gamma(\mathbf{y}).$$

When we substitute

$$s \mapsto \tfrac{1}{\kappa}(1-e^{-s\kappa}),$$

that is, when we deal with the backward Euler CQ method, we need to expand

$$\mathrm{V}(\tfrac{1}{c\kappa}(1-\zeta)) = \sum_{n=0}^{\infty}\omega_n^{\mathrm{V}}(c\kappa)\zeta^n,$$

with operators given by the expressions

$$\omega_n^{\mathrm{V}}(c\kappa)\eta = \int_\Gamma \frac{e^{-\frac{1}{c\kappa}|\cdot-\mathbf{y}|}}{4\pi|\cdot-\mathbf{y}|}\frac{1}{n!}\left(\frac{|\cdot-\mathbf{y}|}{c\kappa}\right)^n\eta(\mathbf{y})\mathrm{d}\Gamma(\mathbf{y}). \tag{93}$$

Note that the only operator of the sequence (93) which has a weakly singular kernel is the one we need to invert in each time-step, that is, the one for $n = 0$. This is the single layer operator for the very diffusive elliptic operator $-\Delta + (c\kappa)^{-2}$, which seems to take the role of a transport operator. If instead of the backward Euler discretization we apply BDF2, the operators have a somewhat more complicated expression

$$\omega_n^{\mathrm{V}}(c\kappa)\eta = \int_\Gamma \frac{e^{-\frac{3}{2c\kappa}|\cdot-\mathbf{y}|}}{4\pi|\cdot-\mathbf{y}|}\frac{1}{n!}\left(\frac{|\cdot-\mathbf{y}|}{2c\kappa}\right)^{n/2}H_n\left(\sqrt{\frac{2|\cdot-\mathbf{y}|}{c\kappa}}\right)\eta(\mathbf{y})\mathrm{d}\Gamma(\mathbf{y}), \tag{94}$$

where $H_n$ is the $n$-th Hermite polynomial. Let us now go back to the fully discrete equation (92a). We will only pay attention to samples of this equation at the time

steps. We then produce vectors $\boldsymbol{\eta}_n \in \mathbb{R}^J$ satisfying equations

$$\mathbf{V}_0 \boldsymbol{\eta}_n = \boldsymbol{\beta}_n - \sum_{m=1}^{n} \mathbf{V}_m \boldsymbol{\eta}_{n-m} \qquad n = 0, 1, \dots, \tag{95}$$

where $\mathbf{V}_n$ are the matrices with elements

$$\int_{\Gamma_i} \int_{\Gamma_j} \frac{e^{-\frac{\delta(0)}{c\kappa}|\mathbf{x}-\mathbf{y}|}}{4\pi|\mathbf{x}-\mathbf{y}|} P_n\left(\frac{|\mathbf{x}-\mathbf{y}|}{c\kappa}\right) \mathrm{d}\Gamma(\mathbf{y}) \mathrm{d}\Gamma(\mathbf{x})$$

for adequate $P_n$ (see (93) and (94)), and where $\boldsymbol{\beta}_n$ is the vector with entries

$$\int_{\Gamma_i} \beta(t_n, \mathbf{x}) \mathrm{d}\Gamma(\mathbf{x}).$$

It is a good moment to reiterate how CQ uses the Laplace transform of the operator but how data are sampled in time and the discrete solution is obtained by time-stepping.

### 5.4.3 First Time, Then Space

Let us go back again to the continuous convolutional system (86) before Galerkin semidiscretization. Since the only term that has undergone spatial discretization is the equation $\mathcal{V}_c * \eta = \beta$, let us have a look at what happens if we first discretize in time using CQ and then in space using a Galerkin scheme. When we substitute $\mathrm{V}(s/c)\mathrm{E}(s) = \mathrm{B}(s)$ by $\mathrm{V}(s_\kappa/c)\mathrm{E}_\kappa(s) = \mathrm{B}(s)$ we are just moving from $\mathcal{V}_c * \eta = \beta$ to the sequence of problems

$$\omega_0^{\mathrm{V}}(c\kappa)\eta_n = \beta(t_n) - \sum_{m=1}^{n} \omega_m^{\mathrm{V}}(c\kappa)\eta_{n-m}. \tag{96}$$

If we discretize all the equations (96) with the same $X_h$-based Galerkin scheme, we obtain the sequence of linear system described in (95). Exactly the same. This means that for this family of problems, Convolution Quadrature and Galerkin discretization in space commute.

### 5.4.4 A Note on the All-Steps-At-Once Method

It is interesting to note that if we use Algorithm 4 to solve the convolution equations (95) we are solving a collection of time-harmonic damped wave equations. In fact, if we want to compute $N$ steps of the process, we end up solving equations of

the form

$$\mathrm{V}\big(\tfrac{1}{c\kappa}\delta(R\zeta_{N+1}^{-\ell})\big)\hat{w}_\ell = \hat{v}_\ell, \qquad \ell = 0,\dots,N.$$

These are the single layer operator equations associated to the operators

$$\Delta - \omega_\ell^2, \quad \text{with} \quad \omega_\ell = \tfrac{1}{c\kappa}\delta(R\zeta_{N+1}^{-\ell})).$$

## 5.5 *Well Posedness After Discretization*

### 5.5.1 Why Bother?

Following the previous pages, the reader can be led to believe that the application of CQ before or after space discretization is just a given. It is not however clear at all why we can even apply CQ after a Galerkin semidiscretization process and whether the semidiscrete operator inherits the properties of the continuous operator. In order to clarify concepts, let us compare the operators with what we get after semidiscretization in space. The first piece of good news is a coercivity estimate

$$\mathrm{Re}\left(e^{\mathrm{i}\,\mathrm{Arg}s}\langle\overline{\eta},\mathrm{V}(s)\eta\rangle_\Gamma\right) \geq C\frac{\sigma\underline{\sigma}}{|s|^2}\|\eta\|^2_{-1/2,\Gamma} \qquad \forall\eta\in H^{-1/2}(\Gamma) \qquad \forall s\in\mathbb{C}_+, \tag{97}$$

where as in Sect. 5.2, $\sigma := \mathrm{Re}\,s$, $\underline{\sigma} := \min\{1,\sigma\}$. The estimate (97) is also often written as

$$\mathrm{Re}\langle\overline{\eta},s\mathrm{V}(s)\eta\rangle_\Gamma \geq C\frac{\sigma\underline{\sigma}}{|s|}\|\eta\|^2_{-1/2,\Gamma} \qquad \forall\eta\in H^{-1/2}(\Gamma) \qquad \forall s\in\mathbb{C}_+,$$

which shows how the operator that is actually coercive is $s\mathrm{V}(s)$, that is, application of $\mathrm{V}(s)$ followed (or preceded) by 'differentiation.' Coercivity estimates are inherited by Galerkin discretizations. This means that if we compare $\mathrm{A}(s) := \mathrm{V}(s/c)^{-1}$ with the operator $\mathrm{A}_h(s) : H^{1/2}(\Gamma) \to X_h$ that corresponds to solving

$$\eta^h = \mathrm{A}_h(s)\beta \in X^h \qquad \langle\mu^h,\mathrm{V}(s)\eta^h - \beta\rangle_\Gamma = 0 \quad \forall\mu^h\in X_h,$$

we have

$$\|\mathrm{A}(s)\|_{H^{1/2}(\Gamma)\to H^{-1/2}(\Gamma)} + \|\mathrm{A}_h(s)\|_{H^{1/2}(\Gamma)\to H^{-1/2}(\Gamma)} \leq C\frac{|s|^2}{\sigma\underline{\sigma}} \qquad \forall s\in\mathbb{C}_+. \tag{98}$$

(The constant $C$ is different in (97) and (98), but we will adopt the bad habits of numerical analysts of calling all constants $C$.) The quality of the behavior with respect to $s$ and its real part in the right-hand-side of (98) is highly relevant for the analysis of CQ as we will discuss in Sect. 8.

### 5.5.2 The Postprocessed Solutions

It would look like we were done if we had not postprocessed the solution in two different ways using the operators

$$\mathrm{S}(s)\mathrm{A}_h(s) \qquad (-\tfrac{1}{2}\mathrm{I} + \mathrm{J}(s))\mathrm{A}_h(s).$$

In principle we can just go ahead and combine the estimate (98) (due to coercivity of $s\mathrm{V}(s)$) with the estimates in (83). This yields the bounds

$$\|\mathrm{S}(s)\mathrm{A}_h(s)\|_{H^{1/2}(\Gamma)\to H^1(\Omega_+)} \leq C\frac{|s|^3}{\sigma^2\underline{\sigma}^2} \qquad \forall s \in \mathbb{C}_+,$$

$$\|(-\tfrac{1}{2}\mathrm{I} + \mathrm{J}(s))\mathrm{A}_h(s)\|_{H^{1/2}(\Gamma)\to H^{-1/2}(\Gamma)} \leq C\frac{|s|^{7/2}}{\sigma^2\underline{\sigma}^{5/2}} \qquad \forall s \in \mathbb{C}_+.$$

These estimates are quite pessimistic though. With a different approach it can actually be proved that

$$\|\mathrm{S}(s)\mathrm{A}_h(s)\|_{H^{1/2}(\Gamma)\to H^1(\Omega_+)} \leq C\frac{|s|^{3/2}}{\sigma\underline{\sigma}^{3/2}} \qquad \forall s \in \mathbb{C}_+, \tag{99a}$$

$$\|(-\tfrac{1}{2}\mathrm{I} + \mathrm{J}(s))\mathrm{A}_h(s)\|_{H^{1/2}(\Gamma)\to H^{-1/2}(\Gamma)} \leq C\frac{|s|^2}{\sigma\underline{\sigma}} \qquad \forall s \in \mathbb{C}_+. \tag{99b}$$

In view of (98) and (83), the bound (99a) is slightly shocking and should make the reader consider carefully the role of $s$ in the operators. In principle $s$ is differentiation. Therefore the bound for $\mathrm{S}(s)$ seems to say that application of the single layer potential comes with loss of time regularity in one unit, while the solution of the integral equation $\mathrm{V}(s)\eta = \beta$ (or of its Galerkin discretization) comes with a loss of two indices of smoothness in time. Their composition, however, loses only $3/2$ smoothness indices. (Take this idea of loss of regularity with multiples of $s$ with a grain of salt. Moving from bounds in the Laplace domain to time domain mapping properties is not optimal and yields some additional losses of smoothness in time.)

## *5.6 Credits*

The presentation and analysis of the acoustic layer potentials in the time domain using their Laplace transforms can be traced back to the seminal work of Alain

Bamberger and Tuong Ha-Duong [1, 2]. They were the first to prove the coercivity estimate (97), which is the origin for the modern theory of acoustic layer potentials in the time domain. Here we have adopted the more systematic approach of [22, 30]. The explicit distributional form of the three dimensional Huygens potentials, briefly mentioned at the end of Sect. 5.2, first appears in [21]. The first use of CQ for a boundary integral equation, related to the heat equation, was given by Christian Lubich and Reinhold Schneider in [27]. Two years later, Lubich himself made the first incursion of CQ applied to the boundary integral equations for the wave equation. The use of CQ for several kinds of elastic wave propagation phenomena using integral equations was quite extended in the engineering literature (see the monograph [31] by Martin Schanz, one of the pioneers in the field) by the time the mathematical community went back to the topic. The effect of Galerkin semidiscretization in space was studied in [25], but the postprocessing part (once the boundary integral equation is solved, input the result in a potential) was only studied in [22]. Other approaches for the study of layer potentials in the time domain are given in [9, 14], using very different techniques that avoid the Laplace transform. The expansions for the operators related to CQ for the single layer operator equation were given in [16]. For more precise information on the Sobolev space theory of boundary integral equations for steady state problems (which is required to understand the Laplace domain estimates of this chapter), the reader is recommended to explore William McLean's monograph [28].

## 6 Multistage Convolution Quadrature

In this section we are going to introduce a new family of discretization methods for causal convolutions and convolution equations. The main difference with the multistep method will be in the fact that we will work simultaneously with several points in time (stages). The way we will develop the method, the Runge-Kutta (RK) steps will barely make an appearance.

The general look of an RK-based discrete convolution for $y = f * g$ is

$$\boldsymbol{y}_n = \sum_{m=0}^{n} W_m^{\mathrm{F}}(\kappa)\boldsymbol{g}_{n-m}$$

where $\boldsymbol{g}_n := (g(t_n+\kappa c_1), \ldots, g(t_n+\kappa c_p))^\top$ and $W_m^{\mathrm{F}}(\kappa)$ is a $p\times p$ matrix of operators, with values in the same space as F.

## 6.1 Some Runge-Kutta Methods

### 6.1.1 Vectorized Notation

We will accept the following (shorthand) vectorized form for evaluation of a function:

$$\boldsymbol{c} = \begin{bmatrix} c_1 \\ \vdots \\ c_p \end{bmatrix} \quad \longmapsto \quad g(t+\kappa \boldsymbol{c}) = \begin{bmatrix} g(t+\kappa c_1) \\ \vdots \\ g(t+\kappa c_p) \end{bmatrix} \in \mathbb{R}^p.$$

Similarly, if $f = f(t,y)$, then

$$\boldsymbol{c}, \boldsymbol{y} \in \mathbb{R}^p \quad \longmapsto \quad f(t+\kappa \boldsymbol{c}, \boldsymbol{y}) = \begin{bmatrix} f(t+\kappa c_1, y_1) \\ \vdots \\ f(t+\kappa c_p, y_p) \end{bmatrix} \in \mathbb{R}^p.$$

### 6.1.2 Implicit RK Methods

An implicit RK scheme is often presented through its Butcher tableau

$$\begin{array}{c|c} \boldsymbol{c} & \boldsymbol{A} \\ \hline & \boldsymbol{b}^\top \end{array} \qquad \boldsymbol{b}, \boldsymbol{c} \in \mathbb{R}^p, \qquad \boldsymbol{A} \in \mathbb{R}^{p \times p},$$

with the conditions

$$\boldsymbol{A}\mathbf{1} = \boldsymbol{c}, \qquad \boldsymbol{b}^\top \mathbf{1} = 1, \qquad \mathbf{1} = (1, \dots, 1)^\top. \tag{100}$$

The second condition in (100) is necessary for convergence. The first condition in (100) is related to the possibility of understanding time as an independent variable. All RK methods satisfy these conditions. Unfortunately the letter $s$ will be reserved for the variable in the Laplace transform, so we will use $p$ for the number of stages. The application of a step of an RK method for

$$\dot{y} = f(t,y) \tag{101}$$

is based on the solution of a system of non-linear equations to compute the internal stages

$$\boldsymbol{y}_n = y_n \mathbf{1} + \kappa \boldsymbol{A} f(t_n + \kappa \boldsymbol{c}, \boldsymbol{y}_n) \tag{102}$$

followed by the computation of the next step

$$y_{n+1} = y_n + \kappa \boldsymbol{b}^\top f(t_n + \kappa \boldsymbol{c}, \boldsymbol{y}_n). \tag{103}$$

The internal stages and the steps approximate

$$\boldsymbol{y}_n \approx y(t_n + \kappa \boldsymbol{c}) \qquad y_n \approx y(t_n).$$

Starting RK methods requires the knowledge of $y_0$. Unlike multistep methods, bringing vanishing information from the past will not change the initial step, which for us will always be $y_0 = 0$.

### 6.1.3 Some Multistage Differentiation Formulas

We next apply the above RK method to the search of causal solutions to the antiderivative problem

$$\dot{y} = g \qquad \longleftrightarrow \qquad s\mathrm{Y}(s) = \mathrm{G}(s).$$

The stages and steps for the method (102)–(103) translate into

$$\boldsymbol{y}_n = y_n \mathbf{1} + \kappa \boldsymbol{A} \boldsymbol{g}_n, \qquad y_{n+1} = y_n + \kappa \boldsymbol{b}^\top \boldsymbol{g}_n, \qquad \boldsymbol{g}_n := g(t_n + \kappa \boldsymbol{c}). \tag{104}$$

It is clear from (104) that causality of $g$ and the imposition of a causal discrete solution ($y_n = 0$ and $\boldsymbol{y}_n = \mathbf{0}$ for $n < 0$) imposes $y_0 = 0$. It is also clear that for this particular equation (a quadrature), the internal stages do not play any role. We will still pay attention to them, since they are the quantities of our interest. We next write (104) using the $\zeta$ transform. To do it, we introduce

$$\mathrm{Y}(\zeta) := \sum_{n=0}^{\infty} y_n \zeta^n, \qquad \boldsymbol{Y}(\zeta) := \sum_{n=0}^{\infty} \boldsymbol{y}_n \zeta^n, \qquad \boldsymbol{G}(\zeta) := \sum_{n=0}^{\infty} \boldsymbol{g}_n \zeta^n,$$

and rewrite (104) as

$$\boldsymbol{Y}(\zeta) = \mathrm{Y}(\zeta)\mathbf{1} + \kappa \boldsymbol{A} \boldsymbol{G}(\zeta), \qquad \zeta^{-1}\mathrm{Y}(\zeta) = \mathrm{Y}(\zeta) + \kappa \boldsymbol{b}^\top \boldsymbol{G}(\zeta).$$

Therefore

$$\mathrm{Y}(\zeta) = \kappa \frac{\zeta}{1-\zeta} \boldsymbol{b}^\top \boldsymbol{G}(\zeta),$$

and from this

$$\boldsymbol{Y}(\zeta) = \kappa \left( \frac{\zeta}{1-\zeta} \mathbf{1}\boldsymbol{b}^\top + \boldsymbol{A} \right) \boldsymbol{G}(\zeta). \tag{105}$$

Equation (105) is the discrete version of antidifferentiation $\mathrm{Y}(s) = s^{-1}\mathrm{G}(s)$, performed at the level of the stages. Therefore the $p \times p$ matrix

$$\Delta(\zeta) := \left( \frac{\zeta}{1-\zeta} \mathbf{1}\boldsymbol{b}^\top + \boldsymbol{A} \right)^{-1}$$

will play the part of $\delta(\zeta)$ for the multistage case:

$$s\mathrm{Y}(s) \text{ is discretized as } \tfrac{1}{\kappa}\Delta(\zeta)\boldsymbol{Y}(\zeta).$$

In other words, $\frac{1}{\kappa}\Delta(\zeta)$ is the discrete transfer function for multistage differentiation. Note that the possibility of defining a discrete multistage differentiation operator by inverting the RK recurrence requires (105) to be an invertible recurrence, which only happens when $\boldsymbol{A}$ is invertible.

### 6.1.4 A Subclass of RK Methods

We consider the subclass of **stiffly accurate** RK methods where the last row of $\boldsymbol{A}$ is $\boldsymbol{b}^\top$, that is,

$$\boldsymbol{e}_p^\top \boldsymbol{A} = \boldsymbol{b}^\top, \qquad \boldsymbol{e}_p^\top = (0, \dots, 0, 1),$$

and therefore $c_m = 1$ (multiply both sides by $\boldsymbol{c}$). Therefore, multiplying (102) by $\boldsymbol{e}_p^\top$ and using (103) it follows that

$$\boldsymbol{e}_p^\top \boldsymbol{y}_n = y_n \boldsymbol{e}_p^\top \mathbf{1} + \kappa \boldsymbol{e}_p^\top \boldsymbol{A} f(t_n + \kappa \boldsymbol{c}, \boldsymbol{y}_n) = y_n + \kappa \boldsymbol{b}^\top f(t_n + \kappa \boldsymbol{c}, \boldsymbol{y}_n) = y_{n+1}, \qquad (106)$$

which means that the last component of $\boldsymbol{y}_n$ is $y_{n+1}$ and we do not need to worry about the steps any more. For this subclass of methods we can recompute the discrete differentiation operator. Using (106), we can write

$$\boldsymbol{y}_n = y_n \mathbf{1} + \kappa \boldsymbol{A}\boldsymbol{g}_n = \mathbf{1}\boldsymbol{e}_p^\top \boldsymbol{y}_{n-1} + \kappa \boldsymbol{A}\boldsymbol{g}_n,$$

or, in the $\zeta$ domain,

$$(\boldsymbol{I} - \zeta \mathbf{1}\boldsymbol{e}_p^\top)\boldsymbol{Y}(\zeta) = \kappa \boldsymbol{A}\boldsymbol{G}(\zeta).$$

In this case, differentiation is given by

$$\tfrac{1}{\kappa}\Delta(\zeta), \qquad \text{where} \qquad \Delta(\zeta) := \boldsymbol{A}^{-1}(\boldsymbol{I} - \zeta \mathbf{1}\boldsymbol{e}_p^\top).$$

Note that this is just an alternative formula for the matrix $\Delta(\zeta)$ defined in the more general case.

### 6.1.5 Two Examples

The order three Radau IIa method and the order four Lobatto IIIc are respectively given by the tables

$$\begin{array}{c|cc} 1/3 & 5/12 & -1/12 \\ 1 & 3/4 & 1/4 \\ \hline & 3/4 & 1/4 \end{array} \qquad \begin{array}{c|ccc} 0 & 1/6 & -1/3 & 1/6 \\ 1/2 & 1/6 & 5/12 & -1/12 \\ 1 & 1/6 & 2/3 & 1/6 \\ \hline & 1/6 & 2/3 & 1/6 \end{array}$$

## *6.2 Elementary Dunford Calculus*

### 6.2.1 Our Next Goal

Now that we have a new approximation of the derivative (of $s$), we might want to define a new approximation of $\mathrm{F}(s)$. If F is scalar valued and entire (analytic in $\mathbb{C}$), it is not entirely difficult to define $\mathrm{F}(\boldsymbol{B})$ for any $p \times p$ matrix $\boldsymbol{B}$ using the power series expansion for F. When F is analytic only in part of $\mathbb{C}$ this is still doable, but not for every matrix: essentially we need all the eigenvalues of $\boldsymbol{B}$ to be in the domain of analyticity of F. The process is however somewhat more complicated when we deal with operator-valued F.

### 6.2.2 Scalar Functions of Matrices

(While this theory can be made much more general, we will keep it close to our assumptions on transfer functions.) Let $\mathrm{F} : \mathbb{C}_+ \to \mathbb{C}$ be analytic and let $\lambda \in \mathbb{C}_+$. Then

$$\mathrm{F}(\lambda) = \frac{1}{2\pi \mathrm{i}} \oint_C (z - \lambda)^{-1} \mathrm{F}(z) \mathrm{d}z,$$

where $C$ is a simple positively oriented closed path around $\lambda$. It does not take much imagination to figure out a definition for $\mathrm{F}(\boldsymbol{\Lambda})$ where $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_p)$ is a diagonal matrix:

$$\begin{aligned} \mathrm{F}(\boldsymbol{\Lambda}) &:= \frac{1}{2\pi \mathrm{i}} \oint_C (z\boldsymbol{I} - \boldsymbol{\Lambda})^{-1} \mathrm{F}(z) \mathrm{d}z \\ &= \mathrm{diag}\left( \frac{1}{2\pi \mathrm{i}} \oint_C (z - \lambda_1)^{-1} \mathrm{F}(z) \mathrm{d}z, \ldots, \frac{1}{2\pi \mathrm{i}} \oint_C (z - \lambda_p)^{-1} \mathrm{F}(z) \mathrm{d}z \right). \end{aligned}$$

The integral is done component by component, and the path $C$ has to enclose the values $\{\lambda_1, \ldots, \lambda_p\}$, that is, the spectrum of $\boldsymbol{\Lambda}$. If $\boldsymbol{B} = \boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^{-1}$, where $\boldsymbol{\Lambda}$ is diagonal,

then a simple computations yields

$$\boldsymbol{P}\mathrm{F}(\boldsymbol{\Lambda})\boldsymbol{P}^{-1} = \frac{1}{2\pi \mathrm{i}} \oint_C (z\boldsymbol{I} - \boldsymbol{B})^{-1} \mathrm{F}(z) \mathrm{d}z,$$

so it is just logical that we adopt the latter expression

$$\mathrm{F}(\boldsymbol{B}) := \frac{1}{2\pi \mathrm{i}} \oint_C (z\boldsymbol{I} - \boldsymbol{B})^{-1} \mathrm{F}(z) \mathrm{d}z, \tag{107}$$

as a definition, even for non-diagonalizable matrices. As in previous cases, $C$ is a simple closed path in $\mathbb{C}_+$ surrounding the spectrum of $\boldsymbol{B}$.

### 6.2.3 Kronecker Products

Let $\boldsymbol{B} \in \mathbb{C}^{p\times q}$ and $F \in \mathcal{B}(X, Y)$. We then define

$$\boldsymbol{B} \otimes F := \begin{bmatrix} b_{11}F & \dots & b_{1q}F \\ \vdots & \ddots & \vdots \\ b_{p1}F & \dots & b_{pq}F \end{bmatrix} \in \mathcal{B}(X, Y)^{p\times q} \equiv \mathcal{B}(X^q, Y^p).$$

This formula gives us the proper definition of an operator-valued function $\mathrm{F} : \mathbb{C}_+ \to \mathcal{B}(X, Y)$ acting on a matrix:

$$\mathrm{F}(\boldsymbol{B}) := \frac{1}{2\pi \mathrm{i}} \oint_C (z\boldsymbol{I} - \boldsymbol{B})^{-1} \otimes \mathrm{F}(z) \mathrm{d}z. \tag{108}$$

Once again $C$ is a closed path surrounding the spectrum of $\boldsymbol{B}$. The resulting $p \times p$ integrals take place in the Banach space $\mathcal{B}(X, Y)$, where F takes values. Equivalently, we can think of the integral as being computed in the Banach space $\mathcal{B}(X, Y)^{p\times p} \equiv \mathcal{B}(X^p, Y^p)$.

### 6.2.4 Key Properties

The fact that the Dunford calculus is given that name (calculus, not Dunford) is due to the fact that it interacts nicely with the algebra of operators. For instance, for every $\boldsymbol{B}$

$$\frac{1}{2\pi \mathrm{i}} \oint_C (z\boldsymbol{I} - \boldsymbol{B})^{-1} \otimes I_X \mathrm{d}z = \left( \frac{1}{2\pi \mathrm{i}} \oint_C (z\boldsymbol{I} - \boldsymbol{B})^{-1} \mathrm{d}z \right) \otimes I_X = \boldsymbol{I} \otimes I_X = I_{X^p}, \tag{109}$$

as long as $C$ surrounds the spectrum of $\boldsymbol{B}$. Also, if $\mathrm{F} : \mathbb{C}_+ \to \mathcal{B}(X, Y)$ and $\mathrm{G} : \mathbb{C}_+ \to \mathcal{B}(Z, X)$, then

$$(\mathrm{FG})(\boldsymbol{B}) = \mathrm{F}(\boldsymbol{B})\mathrm{G}(\boldsymbol{B}) \tag{110}$$

for every matrix with spectrum contained in $\mathbb{C}_+$. In particular, if $\mathrm{F}(s)$ is invertible for all $s \in \mathbb{C}_+$ and $\sigma(\boldsymbol{B}) \subset \mathbb{C}_+$, then by (109) and (110), it is clear that $\mathrm{F}(\boldsymbol{B})$ is invertible and

$$\mathrm{F}(\boldsymbol{B})^{-1} = \mathrm{F}^{-1}(\boldsymbol{B}). \tag{111}$$

### 6.2.5 Some Properties of Kronecker Products

We are next going to explore how to compute (108) in the diagonalizable case by using only evaluations of F on the spectrum of $\boldsymbol{B}$. We need some preliminary work. It is a simple exercise to prove that if $\boldsymbol{B} \in \mathbb{C}^{p\times q}$ and $\boldsymbol{C} \in \mathbb{C}^{q\times r}$, then

$$(\boldsymbol{BC}) \otimes F = (\boldsymbol{B} \otimes I_Y)(\boldsymbol{C} \otimes F). \tag{112}$$

The product in the right-hand-side of (112) is the product of a matrix of operators in $\mathcal{B}(Y, Y)$ with a matrix of operators in $\mathcal{B}(X, Y)$. It can also be understood as a composition of an operator in $\mathcal{B}(Y^q, Y^p)$ with an operator in $\mathcal{B}(X^r, Y^q)$. Let now $\boldsymbol{\Lambda}$ be a $p \times p$ diagonal matrix. Then

$$(\boldsymbol{\Lambda C}) \otimes F = (\boldsymbol{\Lambda} \otimes I_Y)(\boldsymbol{C} \otimes F) = \begin{bmatrix} \lambda_1 \mathrm{row}(\boldsymbol{C}, 1) \otimes F \\ \vdots \\ \lambda_p \mathrm{row}(\boldsymbol{C}, p) \otimes F \end{bmatrix}, \tag{113}$$

where if $\boldsymbol{C} \in \mathbb{C}^{p,q}$, $\mathrm{row}(\boldsymbol{C}, i)$ is the $1 \times q$ matrix containing the $i$-th row of $\boldsymbol{C}$ and therefore $\mathrm{row}(\boldsymbol{C}, i) \otimes F \in \mathcal{B}(X^q, Y) \equiv \mathcal{B}(X, Y)^{1\times q}$. Using (112) and (113) we can compute

$$\begin{aligned}(\boldsymbol{B\Lambda C}) \otimes F &= (\boldsymbol{B} \otimes I_Y)((\boldsymbol{\Lambda C}) \otimes F) \\ &= \mathrm{col}(\boldsymbol{B}, 1) \otimes \big(\lambda_1 \mathrm{row}(\boldsymbol{C}, 1) \otimes F\big) + \ldots + \mathrm{col}(\boldsymbol{B}, p) \otimes \big(\lambda_p \mathrm{row}(\boldsymbol{C}, p) \otimes F\big),\end{aligned}$$

or in short

$$(\boldsymbol{B\Lambda C}) \otimes F = \sum_{i=1}^{p} \mathrm{col}(\boldsymbol{B}, i) \otimes \big(\lambda_i \mathrm{row}(\boldsymbol{C}, i) \otimes F\big). \tag{114}$$

The outermost Kronecker product in (114) corresponds to a column matrix $r \times 1$ with an operator in $\mathcal{B}(X^q, Y)$, outputting an operator in $\mathcal{B}(X^q, Y^r)$.

### 6.2.6 Operator-Valued Functions of a Diagonalizable Matrix

Assume that $\boldsymbol{B} = \boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^{-1}$ with $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \dots, \lambda_p)$, with $\lambda_i \in \mathbb{C}_+$ for all $i$, and that $\mathrm{F} : \mathbb{C}_+ \to \mathcal{B}(X, Y)$ is analytic. Then, (114) says that for $z \notin \sigma(\boldsymbol{B}) = \{\lambda_i\}$

$$
\begin{aligned}
(z\boldsymbol{I} - \boldsymbol{B})^{-1} \otimes \mathrm{F}(z) &= \sum_{i=1}^{p} \operatorname{col}(\boldsymbol{P}, i) \otimes \big((z - \lambda_i)^{-1} \operatorname{row}(\boldsymbol{P}^{-1}, i) \otimes \mathrm{F}(z)\big) \\
&= \sum_{i=1}^{p} \operatorname{col}(\boldsymbol{P}, i) \otimes \big(\operatorname{row}(\boldsymbol{P}^{-1}, i) \otimes ((z - \lambda_i)^{-1} \mathrm{F}(z))\big).
\end{aligned}
$$

Integrating on a path that surrounds the spectrum of $\boldsymbol{B}$ gives the following computable version of (108):

$$
\mathrm{F}(\boldsymbol{B}) = \sum_{i=1}^{p} \operatorname{col}(\boldsymbol{P}, i) \otimes \big(\operatorname{row}(\boldsymbol{P}^{-1}, i) \otimes \mathrm{F}(\lambda_i)\big).
$$

## 6.3 RKCQ

### 6.3.1 From Discrete Differentiation to Discrete Calculus

In Sect. 6.1 we have defined a discrete differentiation symbol

$$
\tfrac{1}{\kappa}\Delta(\zeta), \qquad \Delta(\zeta) := \left(\frac{\zeta}{1-\zeta}\mathbf{1}\boldsymbol{b}^{\top} + \boldsymbol{A}\right)^{-1}.
$$

In Sect. 6.2 we have shown how to define an operator-valued function of a matrix variable using Dunford calculus. In particular, whenever this makes sense, we define

$$
\mathrm{F}(\tfrac{1}{\kappa}\Delta(\zeta)) = \frac{1}{2\pi \mathrm{i}} \oint_C (z\boldsymbol{I} - \tfrac{1}{\kappa}\Delta(\zeta))^{-1} \otimes \mathrm{F}(z)\mathrm{d}z,
$$

where $C$ is a path around the spectrum of $\frac{1}{\kappa}\Delta(\zeta)$, which is assumed to be included in $\mathbb{C}_+$ for $|\zeta| < 1$ (more about this at the end of this section). Then we use a Taylor expansion to obtain the coefficients:

$$
\mathrm{F}(\tfrac{1}{\kappa}\Delta(\zeta)) = \sum_{n=0}^{\infty} W_n^{\mathrm{F}}(\kappa)\zeta^n. \tag{115}
$$

### 6.3.2 Multistage Discrete Convolutions

To discretize $y = f * g$, we apply (115) to obtain

$$y(t_n + \kappa \boldsymbol{c}) \approx \boldsymbol{y}_n = \sum_{m=0}^{n} W_n^{\mathrm{F}}(\kappa)\boldsymbol{g}_{n-m}, \qquad \boldsymbol{g}_n := g(t_n + \kappa \boldsymbol{c}).$$

This same expression can be written with help of the $\zeta$ transform

$$\boldsymbol{Y}(\zeta) = \mathrm{F}(\tfrac{1}{\kappa}\Delta(\zeta))\boldsymbol{G}(\zeta).$$

In the case of convolution equations $f * g = h$, we apply the same idea to obtain a lower triangular system of operator equations

$$\sum_{m=0}^{n} W_m^{\mathrm{F}}(\kappa)\boldsymbol{g}_{n-m} = \boldsymbol{h}_n := h(t_n + \kappa \boldsymbol{c}), \qquad n \geq 0,$$

or, in more explicit form

$$W_0^{\mathrm{F}}(\kappa)\boldsymbol{g}_n = \boldsymbol{h}_n - \sum_{m=1}^{n} W_m^{\mathrm{F}}(\kappa)\boldsymbol{g}_{n-m}. \tag{116}$$

Equation (116) shows how we are inverting an operator equation associated to $\mathcal{B}(X^p; Y^p)$. Note that

$$W_0^{\mathrm{F}}(\kappa) = \mathrm{F}(\tfrac{1}{\kappa}\Delta(0)) = \mathrm{F}(\tfrac{1}{\kappa}\boldsymbol{A}^{-1})$$

is invertible as shown in (111).

### 6.3.3 A Note on the Spectrum of $\boldsymbol{A}$

A requirement for the correct definition of the RKCQ process is the possibility of producing the CQ coefficients $W_n^{\mathrm{F}}(\kappa)$. We recall that one of our prerequisites to define a multistage discrete derivative $\frac{1}{\kappa}\Delta(\zeta)$ was the existence of $\boldsymbol{A}^{-1}$. We will further assume the following property

$$\sigma(\boldsymbol{A}) \subset \mathbb{C}_+. \tag{117}$$

Readers acquainted with the theory of $A$-stable RK methods will recognize that the invertibility of $\boldsymbol{I} - z\boldsymbol{A}$ for $\mathrm{Re}\, z \leq 0$ is one of the hypotheses of $A$-stable methods. Actually this hypothesis and invertibility of $\boldsymbol{A}$ are equivalent to (117). Since $\Delta(\zeta)^{-1} = \frac{\zeta}{1-\zeta}\mathbf{1}\boldsymbol{b} + \boldsymbol{A}$ is a small perturbation of $\boldsymbol{A}$ for small $\zeta$, hypothesis (117)

implies that for small $\zeta$

$$\sigma(\tfrac{1}{\kappa}\Delta(\zeta)) \subset \mathbb{C}_+$$

and therefore the CQ coefficients are well defined.

## 6.4 Stages, Steps, and More

### 6.4.1 Stages or Steps

As defined in Sect. 6.3 the RKCQ process works purely at the stage level. It samples data in the stages and then produces vectors of approximations in the internal stages. This is especially important when thinking of solving convolution equations, where we need to have as many data as unknowns.

### 6.4.2 The Associated Convolution in Continuous Time

Let us start by reviewing a simple fact of multistep CQ. Given the operator valued distribution $f$ and its Laplace transform F, we had

$$\mathrm{F}(\tfrac{1}{\kappa}\delta(\zeta)) = \sum_{n=0}^{\infty} \omega_n^{\mathrm{F}}(\kappa)\zeta^n, \qquad \mathrm{F}(s_\kappa) = \sum_{n=0}^{\infty} \omega_n^{\mathrm{F}} e^{-st_n} \qquad s_\kappa := \tfrac{1}{\kappa}\delta(e^{-s\kappa}),$$

and, in the time domain

$$f_\kappa := \sum_{n=0}^{\infty} \omega_n^{\mathrm{F}} \otimes \delta_{t_n} \qquad \mathcal{L}\{f_\kappa\}(s) = \mathrm{F}(s_\kappa).$$

This means that even if we only computed convolution at discrete times, there is a continuous convolutional operator in the background. The case of RKCQ is more complicated. We first try to replicate the previous formulas by defining

$$\boldsymbol{S}_\kappa = \boldsymbol{S}_\kappa(s) := \tfrac{1}{\kappa}\Delta(e^{-s\kappa}), \qquad \mathrm{F}(\boldsymbol{S}_\kappa) = \sum_{n=0}^{\infty} W_n^{\mathrm{F}}(\kappa) e^{-st_n},$$

the latter $\mathcal{B}(X^p; Y^p)$-valued function of $s$ being the Laplace transform of the causal distribution $\boldsymbol{F}_\kappa := W_n^{\mathrm{F}}(\kappa) \otimes \delta_{t_n}$. This operator-valued distribution cannot be put in convolution with an $X$-valued distribution $g$. Instead, the distribution $g$ is first

modified to the $X^p$-valued distribution

$$\begin{bmatrix} g(\cdot + c_1\kappa) \\ \vdots \\ g(\cdot + c_p\kappa) \end{bmatrix} =: g(\cdot + \kappa \boldsymbol{c}).$$

Note that this can be understood as a convolution process, *but it is not causal*, because it pushes back the origin to be at the level of the different stages. Then the RKCQ process can be understood as the evaluation at the time steps $t_n$ of the continuous convolution

$$\boldsymbol{F}_\kappa * g(\cdot + \kappa \boldsymbol{c}) = \sum_{n=0}^{\infty} W_n^{\mathrm{F}}(\kappa) g(\cdot - t_n + \boldsymbol{c}\kappa).$$

### 6.4.3 Computation of Steps in the Simplest Case

While the emphasis in the multistage CQ process is on the stages, we might want to compute only values at the time steps $t_n$. In the case when $\boldsymbol{e}_p^\top \boldsymbol{A} = \boldsymbol{b}^\top$, we can compute

$$(\boldsymbol{e}_p^\top \otimes I)\boldsymbol{y}_n =: y_{n+1} \approx y(t_{n+1}).$$

This means that we have never computed an approximation $y_0$ (it is zero), unlike in the multistep case, where $y_0$ was computed from $g(0)$.

### 6.4.4 The General Case

In order to give a definition of a multistage CQ method where the input are the vectors $\boldsymbol{g}_n$ and the output are quantities $y_{n+1} \approx (f * g)(t_{n+1})$, we need to go back to some computations in Sect. 6.1. In particular we have shown that for the differential equation $y' = g$ (that is, for the operator $\mathrm{F}(s) = s^{-1}$), we could compute the steps as a postprocessing of the stages

$$\mathrm{Y}(\zeta) = \frac{\zeta}{1-\zeta} \boldsymbol{b}^\top \Delta(\zeta) \boldsymbol{Y}(\zeta), \qquad \boldsymbol{Y}(\zeta) = \kappa \Delta(\zeta)^{-1} \boldsymbol{G}(\zeta).$$

We can use this formula to extend the computation of steps for a general convolution $y = f * g$, by writing

$$\mathrm{Y}(\zeta) = \frac{\zeta}{1-\zeta} \big(\boldsymbol{b}^\top \Delta(\zeta) \otimes I\big) \mathrm{F}\big(\tfrac{1}{\kappa}\Delta(\zeta)\big) \boldsymbol{G}(\zeta). \tag{118}$$

This would suggest that we need to figure out a way of computing the discrete process described by (118). There is however a simpler formula to compute the steps from the previous step and the most recently computed stages. We start by computing

$$(1-\zeta)\boldsymbol{b}^\top \boldsymbol{A}^{-1}\Delta(\zeta)^{-1} = (1-\zeta)\boldsymbol{b}^\top\big(\boldsymbol{I} + \frac{\zeta}{1-\zeta}\boldsymbol{A}^{-1}\mathbf{1}\boldsymbol{b}^T\big)$$
$$= (1-\zeta)\boldsymbol{b}^\top + \zeta(\boldsymbol{b}^\top\boldsymbol{A}^{-1}\mathbf{1})\boldsymbol{b}^\top = (1-\zeta\mu)\boldsymbol{b}^\top,$$

where

$$\mu := 1 - \boldsymbol{b}^\top\boldsymbol{A}^{-1}\mathbf{1}. \tag{119}$$

(More about this quantity later.) Therefore

$$\frac{\zeta}{1-\zeta}\boldsymbol{b}^\top\Delta(\zeta) = \frac{\zeta}{1-\mu\zeta}\boldsymbol{b}^\top\boldsymbol{A}^{-1},$$

which means that the discrete convolution

$$\mathrm{Y}(\zeta) = \frac{\zeta}{1-\zeta}(\boldsymbol{b}^\top\Delta(\zeta)\otimes I)\boldsymbol{Y}(\zeta)$$

is equivalent to

$$(\zeta^{-1}-\mu)\mathrm{Y}(\zeta) = (\boldsymbol{d}^\top\otimes I)\boldsymbol{Y}(\zeta), \qquad \boldsymbol{d}^\top := \boldsymbol{b}^\top\boldsymbol{A}^{-1}, \tag{120}$$

which allows us to write

$$y_{n+1} = \mu y_n + (\boldsymbol{d}^\top\otimes I)\boldsymbol{y}_n = \mu y_n + d_1 y_{n,1} + \ldots + d_p y_{n,p}. \tag{121}$$

The simple case is easily recovered from this formula by noticing that $\boldsymbol{b}^\top\boldsymbol{A}^{-1} = \boldsymbol{e}_p^\top$ and $\mu = 0$.

## 6.5 Implementation of RKCQ

### 6.5.1 General Idea

Much of what we are going to sketch in this section follows closely what was explained in Sect. 4 for the multistage CQ scheme. We will not repeat many of the arguments there, and will just show some important steps. The key formula to keep

in mind is the practical computation

$$\mathrm{F}(\boldsymbol{B}) = \sum_{i=1}^{p} \mathrm{col}(\boldsymbol{P}, i) \otimes \big(\mathrm{row}(\boldsymbol{P}^{-1}, i) \otimes \mathrm{F}(\lambda_i)\big), \quad \boldsymbol{B} = \boldsymbol{P}\mathrm{diag}(\lambda_1, \ldots, \lambda_p)\boldsymbol{P}^{-1}, \tag{122}$$

that was derived in Sect. 6.2.

### 6.5.2 Computation of RKCQ Coefficients

Using an integration contour $C_R := \{\zeta \in \mathbb{C} : |\zeta| = R\}$, with $R = \epsilon^{1/(2N+2)}$ and a trapezoidal rule of $N+1$ points, we can compute

$$\begin{aligned} W_n^{\mathrm{F}}(\kappa) &= \frac{1}{n!}\frac{\mathrm{d}^n}{\mathrm{d}\zeta^n}\mathrm{F}\left(\frac{1}{\kappa}\Delta(\zeta)\right)\Big|_{\zeta=0} = \frac{1}{2\pi \mathrm{i}}\oint_{C_R} \zeta^{-n-1}\mathrm{F}(\tfrac{1}{\kappa}\Delta(\zeta))\mathrm{d}\zeta \\ &\approx \frac{R^{-n}}{N+1}\sum_{\ell=0}^{N} \zeta_{N+1}^{n\ell}\mathrm{F}(\tfrac{1}{\kappa}\Delta(\zeta_{N+1}^{-\ell})), \qquad n = 0, \ldots, N. \end{aligned}$$

As usual $\zeta_{N+1} = e^{\frac{2\pi \mathrm{i}}{N+1}}$. The corresponding algorithm is to be compared with Algorithm 2.

**Algorithm 7 (computation of RKCQ coefficients)**

Note that in the particular case of stiffly accurate methods, we can write

$$\Delta(\zeta) = \boldsymbol{A}^{-1} - \zeta \boldsymbol{C}, \qquad \boldsymbol{C} := \boldsymbol{A}^{-1}\boldsymbol{1}\boldsymbol{e}_p^{\top}.$$

(a) For $\ell = 0, \ldots, N$, find the spectral decomposition

$$\boldsymbol{P}_\ell \boldsymbol{\Lambda}_\ell \boldsymbol{P}_\ell^{-1} = \tfrac{1}{\kappa}\Delta(R\zeta_{N+1}^{-\ell})$$

and use (122) (looping over stages) to compute

$$\hat{\mathrm{F}}_\ell := \mathrm{F}(\tfrac{1}{\kappa}\Delta(R\zeta_{N+1}^{-\ell})).$$

Note that $\hat{\mathrm{F}}_\ell \in \mathcal{B}(X^p, Y^p)$.

(b) Apply the IDFT and scale

$$W_n^{\mathrm{F}}(\kappa) := R^{-n}\left(\frac{1}{N+1}\sum_{\ell=0}^{N} \hat{\mathrm{F}}_\ell \zeta_{N+1}^{n\ell}\right).$$

This method works under the assumption that $\Delta(\zeta)$ is diagonalizable on the path $C_R$.

### 6.5.3 All-Steps-At-Once Forward Convolution

Let us first sample the data

$$\boldsymbol{g}_n := g(t_n + \kappa \boldsymbol{c}), \qquad n = 0, \dots, N.$$

We want to compute

$$\boldsymbol{u}_n = \sum_{m=0}^{n} W^{\mathrm{F}}_{n-m}(\kappa)\boldsymbol{g}_m = \sum_{m=0}^{N} W^{\mathrm{F}}_{n-m}(\kappa)\boldsymbol{g}_m, \qquad n = 0, \dots, N,$$

(note that $W^{\mathrm{F}}_n(\kappa) = 0$ for negative $n$), using approximations

$$W^{\mathrm{F}}_n(\kappa) \approx \frac{R^{-n}}{N+1} \sum_{\ell=0}^{N} \hat{\mathrm{F}}_\ell \zeta_{N+1}^{n\ell}, \qquad \hat{\mathrm{F}}_\ell := \mathrm{F}(\tfrac{1}{\kappa}\Delta(R\zeta_{N+1}^{-\ell})).$$

Proceeding as in Sect. 4.3, we approximate

$$\boldsymbol{u}_n \approx R^{-n} \left( \frac{1}{N+1} \sum_{\ell=0}^{N} \hat{\mathrm{F}}_\ell \left( \sum_{m=0}^{N} R^m \boldsymbol{g}_m \zeta_{N+1}^{-m\ell} \right) \zeta_{N+1}^{\ell} \right).$$

### 6.5.4 The Finite Dimensional Case

Let us briefly detail what is to be done when $g : \mathbb{R} \to \mathbb{R}^{d_2}$ and $f : \mathbb{R} \to \mathbb{R}^{d_1 \times d_2}$. In this case, it is advantageous to deal with samples at stage points as vectors $\boldsymbol{g}_n \in \mathbb{R}^{p\, d_2}$ organized in $p$ blocks of $d_2$ components. The key step is the multiplication

$$\hat{\mathrm{F}}_\ell \hat{\boldsymbol{h}}_\ell, \qquad \hat{\mathrm{F}}_\ell = \mathrm{F}(\tfrac{1}{\kappa}\Delta(R\zeta_{N+1}^{-\ell})),$$

for a given vector $\boldsymbol{h} \in \mathbb{C}^{p\, d_2}$. It is not difficult to see that when $\frac{1}{\kappa}\Delta(R\zeta_{N+1}^{-\ell}) = \boldsymbol{P}_\ell \boldsymbol{\Lambda}_\ell \boldsymbol{P}_\ell^{-1}$, then

$$\hat{\mathrm{F}}_\ell \hat{\boldsymbol{h}}_\ell = (\boldsymbol{P}_\ell \otimes \boldsymbol{I}_{d_1})\mathrm{diag}(\mathrm{F}(\lambda_1), \dots, \mathrm{F}(\lambda_p))(\boldsymbol{P}_\ell^{-1} \otimes \boldsymbol{I}_{d_2})\hat{\boldsymbol{h}}_\ell. \tag{123}$$

If $g$ takes values in $X$ and $F$ in $\mathcal{B}(X; Y)$, then we have to deal with samples $\boldsymbol{g}_n \in X^p$, with some modified vectors $\hat{\boldsymbol{h}}_\ell \in X^p$ (see Algorithm 8) and that (123) still applies if we substitute $\boldsymbol{I}_{d_1}$ and $\boldsymbol{I}_{d_2}$ by $I_Y$ and $I_X$ respectively.

If this is the last step of a sequence of convolutions, that is, if we are not going to apply any other convolution operator to this result, we can keep the last component of $\boldsymbol{u}_n$ as approximation in the point $t_{n+1}$. Note that because the RKCQ

**Algorithm 8 (all-steps-at-once convolution)**

We begin by sampling data at the discrete times

$$\boldsymbol{g}_n := g(t_n + \kappa \boldsymbol{c}) \in X^p, \qquad n = 0, \dots, N.$$

(a) Scale data:

$$\boldsymbol{h}_m := R^m \boldsymbol{g}_m, \qquad m = 0, \dots, N.$$

(b) Compute the DFT:

$$\hat{\boldsymbol{h}}_\ell := \sum_{m=0}^{N} \boldsymbol{h}_m \zeta_{N+1}^{-m\ell}, \qquad \ell = 0, \dots, N.$$

These are formal DFTs, componentwise in $X^p$. When $X = \mathbb{R}^d$, these can be broken to $p\,d$ separate scalar DFTs.

(c) For every $\ell = 0, \dots, N$, find the spectral decomposition $\frac{1}{\kappa}\Delta(R\zeta_{N+1}^{-\ell}) = \boldsymbol{P}_\ell \boldsymbol{\Lambda}_\ell \boldsymbol{P}_\ell^{-1}$ and compute

$$\hat{\boldsymbol{v}}_\ell := \hat{\mathrm{F}}_\ell \hat{\boldsymbol{h}}_\ell = (\boldsymbol{P}_\ell \otimes I_Y)\mathrm{diag}(\mathrm{F}(\lambda_1), \dots, \mathrm{F}(\lambda_p))(\boldsymbol{P}_\ell^{-1} \otimes I_X)\hat{\boldsymbol{h}}_\ell.$$

Note that the product by the block-diagonal matrix in the center can be done componentwise in $X$.

(d) Compute the IDFT:

$$\boldsymbol{v}_n := \frac{1}{N+1} \sum_{\ell=0}^{N} \hat{\boldsymbol{v}}_\ell \zeta_{N+1}^{\ell n}, \qquad n = 0, \dots, N.$$

(See the comments on step (b).)

(e) Scale back

$$\boldsymbol{u}_n := R^{-1} \boldsymbol{v}_n \in Y^p, \qquad n = 0, \dots, N.$$

method counts intervals (groups of stages) and not steps, we are not computing an approximation at $t_0$, and the final time-step takes us to $t_{N+1}$ and not to $t_N$.

### 6.5.5 Convolution Equations

We will not repeat the argument for convolution equations. Algorithm 8 can be easily modified to handle this new situation. The only step to be changed is (c), where we need a multiplication

$$\hat{\mathrm{F}}_\ell^{-1} \hat{\boldsymbol{v}}_m = \hat{\mathrm{F}}_\ell \hat{\boldsymbol{h}}_\ell = (\boldsymbol{P}_\ell \otimes I_Y)\mathrm{diag}(\mathrm{F}(\lambda_1)^{-1}, \dots, \mathrm{F}(\lambda_p)^{-1})(\boldsymbol{P}_\ell^{-1} \otimes I_X)\hat{\boldsymbol{v}}_\ell,$$

that is, we need to solve $p$ equations associated to the operators $\mathrm{F}(\lambda_j)$. For a comparison, see how Algorithm 3 is modified to Algorithm 4.

**Algorithm 9 (computation of a piece of a convolution)**

The algorithm to compute

$$\boldsymbol{g}_n := \sum_{m=0}^{Q} W_n^{\mathrm{F}}(\kappa)\boldsymbol{u}_m, \qquad n = Q+1, \dots, M,$$

renumbered in the form

$$\tilde{\boldsymbol{g}}_k := \sum_{m=0}^{Q} W_{k+Q+1-m}^{\mathrm{F}}(\kappa)\boldsymbol{u}_m, \qquad k = 0, \dots, M-Q-1,$$

starting from vectors $\boldsymbol{u}_m \in X^p$ and outputting values in $Y^p$, and using approximations

$$W_n^{\mathrm{F}}(\kappa) \approx \frac{R^{-n}}{N+1} \sum_{\ell=0}^{N} \hat{\mathrm{F}}_\ell \zeta_{N+1}^{n\ell}, \qquad \hat{\mathrm{F}}_\ell := \mathrm{F}(\tfrac{1}{\kappa}\Delta(R\zeta_{N+1}^{-\ell})) \tag{124}$$

(for positive and negative $n$) is derived in an entirely similar way to what we did in Sect. 4.4. The parameter $N \geq M$ is a design parameter that influences the size of the computation, but also the precision to which the approximations (124) are carried out.

(a) Scale and augment data

$$\boldsymbol{w}_m := \begin{cases} R^m \boldsymbol{u}_m, & 0 \leq m \leq Q, \\ \boldsymbol{0}, & Q+1 \leq m \leq N. \end{cases}$$

(b) Compute the DFT $\hat{\boldsymbol{w}}_\ell$ ($\ell = 0, \dots, N$) of the vectors in (a). See Algorithm 8(b) for a comment on this step.

(c) For every $\ell = 0, \dots, N$, find the spectral decomposition $\frac{1}{\kappa}\Delta(R\zeta_{N+1}^{-\ell}) = \boldsymbol{P}_\ell \boldsymbol{\Lambda}_\ell \boldsymbol{P}_\ell^{-1}$ and compute

$$\hat{\boldsymbol{h}}_\ell := \zeta_{N+1}^{\ell(Q+1)} \hat{\mathrm{F}}_\ell \hat{\boldsymbol{w}}_\ell = \zeta_{N+1}^{\ell(Q+1)} (\boldsymbol{P}_\ell \otimes I_Y) \mathrm{diag}(\mathrm{F}(\lambda_1), \dots, \mathrm{F}(\lambda_p)) (\boldsymbol{P}_\ell^{-1} \otimes I_X) \hat{\boldsymbol{w}}_\ell.$$

(d) Compute the IDFT of the sequence in (c), $\boldsymbol{h}_\ell$ ($\ell = 0, \dots, N$).

(e) Scale and chop the resulting sequence

$$\tilde{\boldsymbol{g}}_k := R^{-k-Q-1} \boldsymbol{h}_\ell, \qquad k = 0, \dots, M-Q-1.$$

## 6.6 Credits

For a deeper introduction to the Dunford calculus, the reader is referred to [12]. Runge-Kutta convolution quadrature first appeared in a paper by Christian Lubich

and Alexander Ostermann [26]. Some further theoretical developments can be found in [11]. The interest in RKCQ in the area of time-domain boundary integral equations is more recent [3, 8]. The algorithms shown in Sect. 6.5 can be found in [6]. The analysis of RKCQ applied to problems whose transfer function has the structure (13)–(14) was developed in [4] and [7].

# 7 A Toy Application

In this section we will show a simple fully discrete (and very easy to code) example for the scattering of an acoustic wave by a smooth obstacle in the plane. All the operators will be given directly through their Laplace domain representations. The numerical method that we will present here can be understood as a fully discrete version of a Galerkin method. While the theory for the Galerkin method follows from existing arguments in the literature, the full discretization of the equations is not entirely justified.

## *7.1 Scattering by a Smooth Closed Obstacle*

### 7.1.1 The Geometry

Consider a simple smooth closed curve in the plane parametrized by a 1-periodic function $\mathbf{x} : \mathbb{R} \to \Gamma \subset \mathbb{R}^2$ satisfying:

$$\mathbf{x}(r) = \mathbf{x}(r+1) \quad \forall r, \qquad \mathbf{x}(r) \neq \mathbf{x}(\rho) \quad \text{if } r - \rho \notin \mathbb{Z}, \qquad |\mathbf{x}'(r)| \neq 0 \quad \forall r.$$

We assume that the parametrization gives a positive orientation to the curve, so that $\mathbf{n}(r) := (x_2'(r), -x_1'(r))$ is a normal outward pointing vector at $\mathbf{x}(r)$.

### 7.1.2 One Potential and Two Operators

Given a 1-periodic density $\eta : \mathbb{R} \to \mathbb{C}$, directly defined in parametric space, the associated single layer potential at speed one (given in the Laplace domain) is

$$(\mathrm{S}(s)\eta)(\mathbf{z}) := \frac{\mathrm{i}}{4} \int_0^1 H_0^{(1)}(\mathrm{i}s|\mathbf{z} - \mathbf{x}(\rho)|)\eta(\rho)\mathrm{d}\rho. \tag{125}$$

Two operators can be used to represent the boundary values of the single layer potential, the single layer operator

$$(\mathrm{V}(s)\eta)(r) := \frac{\mathrm{i}}{4} \int_0^1 H_0^{(1)}(\mathrm{i}s|\mathbf{x}(r) - \mathbf{x}(\rho)|)\eta(\rho)\mathrm{d}\rho, \tag{126}$$

and the transposed double-layer operator

$$(\mathrm{J}(s)\eta)(r) := \frac{s}{4}\int_0^1 H_1^{(1)}(\mathrm{i}s|\mathbf{x}(r)-\mathbf{x}(\rho)|)\frac{(\mathbf{x}(r)-\mathbf{x}(\rho))\cdot\mathbf{n}(r)}{|\mathbf{x}(r)-\mathbf{x}(\rho)|}\eta(\rho)\mathrm{d}\rho. \tag{127}$$

The functions $H_0^{(1)}$ and $H_1^{(1)}$ in (125), (126) and (127) are the Hankel functions of the first kind and respective orders zero and one. The exterior boundary values for $U = \mathrm{S}(s)\eta$ are given by the expressions:

$$(U^+\circ\mathbf{x})(r) = (\mathrm{V}(s)\eta)(r), \qquad (\nabla U^+\circ\mathbf{x})(r)\cdot\mathbf{n}(r) = -\tfrac{1}{2}\eta(r)+(\mathrm{J}(s)\eta)(r). \tag{128}$$

Note how instead of using a unit normal vector, we are employing the non-normalized vector field $\mathbf{n}$ in (127) and (128). This simplifies some expressions like the exterior normal derivative in (128).

### 7.1.3 The Transient Problem

We are then going to bring an incident plane wave to the game (see Sect. 5.1)

$$u^{\mathrm{inc}}(\mathbf{z},t) = \psi(c(t-t_{\mathrm{lag}})-\mathbf{z}\cdot\mathbf{d}), \qquad |\mathbf{d}| = 1.$$

This incident wave is read on points of the boundary to create

$$\beta(t)(r) = \beta(r,t) := \psi(c(t-t_{\mathrm{lag}})-\mathbf{x}(r)\cdot\mathbf{d}), \tag{129}$$

which is periodic in $r$ and assumed causal in $t$. The scattering problem then looks for $\eta : \mathbb{R}^2 \to \mathbb{R}$, 1-periodic in its first variable and causal in the second such that

$$(\mathcal{V}_c * \eta)(t) + \beta(t) = 0 \qquad \forall t, \qquad \text{where} \quad \mathcal{L}\{\mathcal{V}_c\} = \mathrm{V}(\cdot/c).$$

(Note how it has been convenient to think of functions as being only functions of the time variable with output in a certain non-specified space of 1-periodic functions.) The density is then input in a potential expression

$$U(t) = (\mathcal{S}_c * \eta)(t), \qquad \text{where} \quad \mathcal{L}\{\mathcal{S}_c\} = \mathrm{S}(\cdot/c),$$

and is used to generate the exterior normal derivative

$$\lambda(t) = -\tfrac{1}{2}\eta(t) + (\mathcal{J}_c * \eta)(t), \qquad \text{where} \quad \mathcal{L}\{\mathcal{J}_c\} = \mathrm{J}(\cdot/c).$$

## 7.2 Fully Discrete Equations

### 7.2.1 The Source Geometry

Let us choose a positive integer $N$ and consider $h := 1/N$ as the discrete mesh-size. The geometry is sampled in the following simple way:

$$\mathbf{m}_j := \mathbf{x}(j\,h), \qquad \mathbf{n}_j := h\mathbf{n}(j\,h), \qquad j \in \mathbb{Z}_N, \tag{130}$$

where $\mathbb{Z}_N$ is the set of integers counted modulo $N$. (It is clear that (130) defines only $N$ different points, since both $\mathbf{x}$ and $\mathbf{n}$ are 1-periodic functions.) A discrete density is now a vector $\boldsymbol{\eta} \in \mathbb{C}^N$, or, more properly (and pedantically) speaking, a function $\boldsymbol{\eta} : \mathbb{Z}_N \to \mathbb{C}$. The associated single layer potential is then given by a sum of sources

$$(\mathbf{S}(s)\boldsymbol{\eta})(\mathbf{z}) := \frac{\mathrm{i}}{4}\sum_{j=1}^{N} H_0^{(1)}(\mathrm{i}s|\mathbf{z}-\mathbf{m}_j|)\eta_j. \tag{131}$$

### 7.2.2 The Observation Geometries

Because of the logarithmic singularity of the Hankel function $H_0^{(1)}$ at the origin, we are not allowed to use a simple evaluation of (126) at the same points were we have concentrated the density. To overcome this difficulty, and for reasons that will be discussed at the end of this section, we are going to choose two observation grids:

$$\mathbf{m}_i^{\pm} := \mathbf{x}((i \pm \tfrac{1}{6})h), \qquad \mathbf{n}_i^{\pm} := h\mathbf{n}((i \pm \tfrac{1}{6})h), \qquad i \in \mathbb{Z}_N. \tag{132}$$

For averaging any pair of discrete functions defined on the observation grids we will use the following notation:

$$\sum_{\pm} a_i^{\pm} := \tfrac{1}{2}(a_i^{+} + a_i^{-}).$$

The discrete version of (126) is given by

$$(\mathbf{V}(s)\boldsymbol{\eta})_i := \sum_j \sum_{\pm} \frac{\mathrm{i}}{4} H_0^{(1)}(\mathrm{i}s|\mathbf{m}_i^{\pm} - \mathbf{m}_j|)\eta_j, \tag{133}$$

while for (127) we use a first identical format

$$(\mathbf{J}^{\circ}(s)\boldsymbol{\eta})_i := \sum_j \sum_{\pm} \frac{s}{4} H_1^{(1)}(\mathrm{i}s|\mathbf{m}_i^{\pm} - \mathbf{m}_j|)\frac{(\mathbf{m}_i^{\pm} - \mathbf{m}_j)\cdot\mathbf{n}_i^{\pm}}{|\mathbf{m}_i^{\pm} - \mathbf{m}_j|}\eta_j, \tag{134}$$

that we next correct in the following form

$$\mathbf{J}(s) = \mathbf{Q}\,\mathbf{J}^{\circ}(s), \qquad \text{where} \quad \mathbf{Q}_{ij} := \begin{cases} \frac{11}{12} & i = j, \\ \frac{1}{24} & i = j \pm 1 \,(\mathrm{mod}\, N), \\ 0 & \text{otherwise.} \end{cases} \tag{135}$$

There is a final tricky point that comes from observation of the second equation in (128). Since the density has been concentrated on the points $\mathbf{m}_j$, how can we observe it in the points $\mathbf{m}_i^{\pm}$? There is no simple reason for the following answer (see the final of this section for precise references on this), but this is it. A matrix

$$\mathbf{M}_{ij} := \begin{cases} \frac{7}{9} & i = j, \\ \frac{1}{9} & i = j \pm 1 \,(\mathrm{mod}\, N), \\ 0 & \text{otherwise,} \end{cases} \tag{136}$$

will play the role of the identity operator. Corresponding to these transfer functions, there are three time domain distributions

$$\mathcal{L}\{\mathcal{S}_c\} = \mathbf{S}(\cdot/c), \qquad \mathcal{L}\{\mathcal{V}_c\} = \mathbf{V}(\cdot/c), \qquad \mathcal{L}\{\mathcal{J}_c\} = \mathbf{J}(\cdot/c).$$

### 7.2.3 The Semidiscrete Time Domain Problem

The incident wave (129) is observed in the observation points to create a discrete causal function $\boldsymbol{\beta} : \mathbb{R} \to \mathbb{R}^N$

$$\beta_i(t) = \sum_{\pm} \psi(c(t - t_{\mathrm{lag}}) - \mathbf{m}_i^{\pm} \cdot \mathbf{d}), \tag{137a}$$

a causal discrete density $\boldsymbol{\eta} : \mathbb{R} \to \mathbb{R}^N$ is then computed by solving the convolution equations

$$(\mathcal{V}_c * \boldsymbol{\eta})(t) + \boldsymbol{\beta}(t) = 0 \qquad \forall t, \tag{137b}$$

and then a potential is generated

$$U(t) = (\mathcal{S}_c * \boldsymbol{\eta})(t) \qquad \forall t, \tag{137c}$$

and as well as an approximation of the normal derivative, $\boldsymbol{\lambda} : \mathbb{R} \to \mathbb{R}^N$,

$$\mathbf{M}\boldsymbol{\lambda}(t) = -\tfrac{1}{2}\mathbf{M}\boldsymbol{\eta}(t) + (\mathcal{J}_c * \boldsymbol{\eta})(t). \tag{137d}$$

**Fig. 3** Six snapshots of a manatee-shaped sound-soft scatterer being hit by a plane wave, clearly visible in the first picture. The integral equations are solved using the method of Sect. 7 on the boundary of the scatterer. After that the potential is computed on points in the given frame. An order three Radau IIA CQ method was used in the time domain

It is interesting to notice than even after discretization the function $U$ is still a causal solution of the wave equation in $\mathbb{R}^2 \setminus \Gamma$. The final fully discrete method comes from applying CQ to each of the three convolutions in (137). In the case of multistep CQ, the function $\boldsymbol{\beta}$ is sampled at equidistant times (the steps), and then three CQ processes (one convolution equation and two forward convolutions) are launched. In the case of RKCQ, the sampling is done at the stage points, the convolution equation is solved at the stage level, and finally two forward convolutions yield approximations at the step points. A simulation produced by these computations is seen in Fig. 3.

## *7.3 Credits*

This section is based on the fully discrete Calderón Calculus developed in [15]. The somewhat puzzling choices of parameters (the relative distances $\pm 1/6$ to the observation grids, the matrices $\mathbf{M}$ and $\mathbf{Q}$) can be justified using careful Fourier analysis. Some intuitive explanation can be gathered from [15].

# 8 The Theory of Convolution Quadrature

In this section we collect some convergence results for multistep and multistage CQ applied to convolutions whose symbol is defined in $\mathbb{C}_+$. The multistage results are taken from [25] with a slight refinement (on the behavior of constants with respect to time) to be found in [30]. The trapezoidal rule is not covered by that analysis but can be found in [3]. The convergence of multistage CQ for problems relevant to the wave equation was developed in [4] and [7].

## *8.1 Multistep CQ*

In this section we give a fast review of some results on convergence for multistep CQ for general operator valued convolutions. For ease of reference, let us recall some notation:

$$\mathbb{C}_+ := \{s \in \mathbb{C} \,:\, \operatorname{Re} s > 0\}, \qquad \sigma := \operatorname{Re} s, \qquad \underline{\sigma} := \min\{1, \sigma\}.$$

### 8.1.1 Hypotheses on $\delta(\zeta)$ for Multistep Methods

The function $\delta : \mathcal{U} \to \mathbb{C}$ is analytic in $\mathcal{U}$ where $\{\zeta \in \mathbb{C} \,:\, |\zeta| \leq 1\} \subset \mathcal{U}$. We also require that $\delta : B(0,1) \to \mathbb{C}_+$, that is, $\operatorname{Re} \delta(\zeta) > 0$ for all $\zeta$ such that $|\zeta| < 1$. Finally, we require that there exists $q \geq 1$, $C_0 > 0$, and $\kappa_0 > 0$ such that

$$\left|\tfrac{1}{\kappa}\delta(e^{-\kappa}) - 1\right| \leq C_0 \kappa^q \qquad \forall \kappa \leq \kappa_0.$$

These hypotheses are satisfied by the BE method $\delta(\zeta) = 1 - \zeta$ with $q = 1$ and by the BDF2 method $\delta(\zeta) = \frac{3}{2} - 2\zeta + \frac{1}{2}\zeta^2$ with $q = 2$. Note that the trapezoidal rule is not covered by this analysis, and TR has to be analyzed using some different tricks. Some work in the complex plane implies that the map $\mathbb{C}_+ \ni s \mapsto s_\kappa := \frac{1}{\kappa}\delta(e^{-s\kappa})$ satisfies

$$|s_\kappa| \leq C_1 |s|, \qquad |s_\kappa - s| \leq C_2 \kappa^q |s|^{q+1}, \qquad \operatorname{Re} s_\kappa \geq C_3 \underline{\sigma} \qquad \forall s \in \mathbb{C}_+. \tag{138}$$

### 8.1.2 Hypotheses on the Transfer Function

Let now

$$\mathrm{F} : \mathbb{C}_+ \to \mathcal{B}(X; Y) \tag{139a}$$

be analytic and satisfy

$$\|\mathrm{F}(s)\|_{X\to Y} \le C_{\mathrm{F}}(\sigma)|s|^{\mu} \qquad \forall s \in \mathbb{C}_+, \qquad \text{with } \mu \ge 0, \tag{139b}$$

where $C_{\mathrm{F}} : (0,\infty) \to (0,\infty)$ is non-increasing and

$$C_{\mathrm{F}}(\sigma) \le \frac{C_0}{\sigma^m} \qquad \forall \sigma \in (0,1], \quad \text{with } m \ge 0. \tag{139c}$$

Recall that Sect. 2.4 identified F with the Laplace transform of an operator-valued causal distribution which could be written as a distributional derivative of a continuous causal operator-valued function with polynomial growth. Thanks to (138), we can also identify the map $\mathbb{C}_+ \ni s \mapsto \mathrm{F}(s_\kappa)$ with the Laplace transform of a causal distribution with values in $\mathcal{B}(X;Y)$.

### 8.1.3 A Convergence Result

Let $\delta$ and F satisfy the above hypotheses. Let $g : \mathbb{R} \to X$ be causal and $C^k$ with $k > \mu + q + 2$. Then

$$\|(f * g)(t) - (f_\kappa * g)(t)\|_Y \le C\kappa^q h(t) \int_0^t \|g^{(k)}(\tau)\|_X \mathrm{d}\tau, \tag{140}$$

where

$$h(t) = \begin{cases} t^{k-(\mu+q+1)}, & t \le 1, \\ t^{k-\mu+m}, & t \ge 1. \end{cases}$$

In (140), $f = \mathcal{L}^{-1}\{\mathrm{F}\}$ and $f_\kappa = \mathcal{L}^{-1}\{\mathrm{F}_\kappa\}$, with $\mathrm{F}_\kappa(s) := \mathrm{F}(s_\kappa)$.

## *8.2 Multistage CQ*

### 8.2.1 Order of Convergence for an RK Method

Consider an RK method applied to the IVP

$$\dot{y} = f(t,y), \quad 0 \le t \le T, \qquad y(0) = y_0.$$

The internal stages

$$\boldsymbol{y}_n = y_n \mathbf{1} + \kappa \boldsymbol{A} f(t_n + \kappa \boldsymbol{c}, \boldsymbol{y}_n)$$

and the steps

$$y_{n+1} = y_n + \kappa \boldsymbol{b}^\top f(t_n + \kappa \boldsymbol{c}, \boldsymbol{y}_n)$$

create approximations

$$\boldsymbol{y}_n \approx y(t_n + \kappa \boldsymbol{c}) \qquad y_n \approx y(t_n).$$

We say that the **stage order** of the RK method is $q$ when for a smooth enough solution $y$

$$|\boldsymbol{y}_1 - y(\kappa \boldsymbol{c})| \leq C h^{q+1}.$$

We say that the **classical order** of the RK method is $r$ when for a smooth enough solution $y$

$$|y_1 - y(t_1)| \leq C h^{r+1}.$$

Note that the two methods given at the end of Sect. 6.1 have respective classical orders 3 and 4, while both share stage order equal to 2.

### 8.2.2 Hypotheses on the RK Method

Consider an RK method and its stability function

$$R(z) = 1 + z \boldsymbol{b}^\top (\boldsymbol{I} - z\boldsymbol{A})^{-1} \mathbf{1}.$$

We will assume that:

(a) The matrix $\boldsymbol{A}$ is invertible. (This is needed right at the beginning of Sect. 6.1, in order to give a definition to the discrete multistage differentiation operator.)

(b) (A-stability) For all $z$ such that $\operatorname{Re} z \leq 0$, the matrix $\boldsymbol{I} - z\boldsymbol{A}$ is invertible and

$$|R(z)| \leq 1.$$

As mentioned in Sect. 6.3, hypotheses (a) and (b) imply that $\sigma(\boldsymbol{A}) \subset \mathbb{C}_+$.

(c) $R(\infty) = 0$, that is $\boldsymbol{b}^\top \boldsymbol{A}^{-1} \mathbf{1} = 1$. (Note that the quantity $\mu = 1 - \boldsymbol{b}^\top \boldsymbol{A}^{-1} \mathbf{1} = R(\infty)$ had appeared in (119) at the time when we wanted to compute steps for RKCQ.)

(d)

$$|R(\mathrm{i}\omega)| < 1 \qquad \forall \omega \in \mathbb{R} \setminus \{0\}.$$

### 8.2.3 A Word on the RKCQ Output

While in principle the RKCQ produces a sequence of vectors $Y^p \ni \boldsymbol{y}_n \approx y(t_n + \kappa \boldsymbol{c})$, with $y = f * g$, and a sequence of 'scalars' $Y \ni y_n \approx y(t_n)$, like in the multistage case, there is a formula that extends these values to continuous times. We saw in Sect. 6.4 that RKCQ could be understood as outputting

$$\boldsymbol{y}_\kappa(t) = \sum_{m=0}^{\infty} W_m^{\mathrm{F}}(\kappa) g(t - t_m + \kappa \boldsymbol{c}), \tag{141}$$

so that $\boldsymbol{y}_\kappa(t_n) = \boldsymbol{y}_n$. The time steps can be produced using a simple postprocessing of the stages

$$y_{n+1} = (\boldsymbol{b}^\top \boldsymbol{A}^{-1} \otimes I)\boldsymbol{y}_n$$

(recall (120) and (121) and notice that we are assuming that $\mu = 0$). At the continuous level, this corresponds to the values at time $t_{n+1}$ of

$$y_\kappa = (\boldsymbol{b}^\top \boldsymbol{A}^{-1} \otimes I)\boldsymbol{y}_\kappa(\cdot - \kappa). \tag{142}$$

### 8.2.4 Hypotheses on the Transfer Function

The hypotheses on F are slightly different than those given for multistep CQ. We assume that

$$\mathrm{F} : \mathbb{C}_+ \to \mathcal{B}(X; Y) \tag{143a}$$

is analytic and satisfies

$$\|\mathrm{F}(s)\|_{X \to Y} \le C_{\mathrm{F}}(\sigma) \frac{|s|^\mu}{\sigma^\nu} \qquad \forall s \in \mathbb{C}_+, \qquad \text{with } \mu \ge 0, \nu \ge 0, \tag{143b}$$

where $C_{\mathrm{F}} : (0, \infty) \to (0, \infty)$ is non-increasing and

$$C_{\mathrm{F}}(\sigma) \le \frac{C_0}{\sigma^m} \qquad \forall \sigma \in (0, 1], \quad \text{with } m \ge 0. \tag{143c}$$

Note that we have factored out $\sigma^\nu$ from $C_{\mathrm{F}}$ so that a bound with a power of $\sigma$ in the denominator is also valid as $\sigma \to \infty$.

### 8.2.5 A Convergence Result

Let F satisfy hypotheses (139) and the RK method satisfy the previous hypotheses with stage order $q$ and classical order $r \geq q$. Assume that $g : \mathbb{R} \to X$ is causal and $C^k$ with $k > \mu + r + 2$. If $y = f * g$ and $y_\kappa$ is the RKCQ approximation of $y$ using (141) and (142), then

$$\|y(t) - y_\kappa(t)\|_Y \leq C\kappa^{\min\{r,q+1-\mu+\nu\}} h(t) \int_0^t \|g^{(k)}(\tau)\|_X \mathrm{d}\tau.$$

Here $h$ is an increasing function of time, whose behavior is not entirely well understood, although it is unlikely that they behave worse than polynomially in time. For the particular case of operators satisfying

$$\|\mathrm{F}(s)\| \leq Ce^{-c\,\sigma}|s|^\mu \qquad \forall s \in \mathbb{C}_+, \qquad \mu \geq 0, \quad c > 0,$$

the full classical convergence order of the method is attained for smooth enough functions, since we can choose any arbitrarily large $\nu$ in the hypotheses for F.

## References


1. Bamberger, A., Duong, T.H.: Formulation variationnelle espace-temps pour le calcul par potentiel retardé de la diffraction d'une onde acoustique. I. Math. Methods Appl. Sci. **8**(3), 405–435 (1986)
2. Bamberger, A., Duong, T.H.: Formulation variationnelle pour le calcul de la diffraction d'une onde acoustique par une surface rigide. Math. Methods Appl. Sci. **8**(4), 598–608 (1986)
3. Banjai, L.: Multistep and multistage convolution quadrature for the wave equation: algorithms and experiments. SIAM J. Sci. Comput. **32**(5), 2964–2994 (2010)
4. Banjai, L., Lubich, C.: An error analysis of Runge-Kutta convolution quadrature. BIT **51**(3), 483–496 (2011)
5. Banjai, L., Sauter, S.: Rapid solution of the wave equation in unbounded domains. SIAM J. Numer. Anal. **47**(1), 227–249 (2008/2009)
6. Banjai, L., Schanz, M.: Wave propagation problems treated with convolution quadrature and BEM. In: Fast Boundary Element Methods in Engineering and Industrial Applications. Lecture Notes in Applied and Computational Mechanics, vol. 63, pp. 145–184. Springer, Heidelberg (2012)
7. Banjai, L., Lubich, C., Melenk, J.M.: Runge-Kutta convolution quadrature for operators arising in wave propagation. Numer. Math. **119**(1), 1–20 (2011)
8. Banjai, L., Messner, M., Schanz, M.: Runge-Kutta convolution quadrature for the boundary element method. Comput. Methods Appl. Mech. Eng. **245/246**, 90–101 (2012)
9. Banjai, L., Laliena, A.R., Sayas, F.J.: Fully discrete Kirchhoff formulas with CQ-BEM. IMA J. Numer. Anal. **35**(2), 859–884 (2015)
10. Betcke, T., Salles, N., Smigaj, W.: Exponentially accurate evaluation of time-stepping methods for the wave equation via convolution quadrature type schemes (2016). arXiv:1603.01761
11. Calvo, M.P., Cuesta, E., Palencia, C.: Runge-Kutta convolution quadrature methods for well-posed equations with memory. Numer. Math. **107**(4), 589–614 (2007)

12. Dautray, R., Lions, J.L.: Mathematical Analysis and Numerical Methods for Science and Technology, vol. 3. Springer, Berlin (1990)
13. Dautray, R., Lions, J.L.: Mathematical Analysis and Numerical Methods for Science and Technology, vol. 5. Springer, Berlin (1992)
14. Domínguez, V., Sayas, F.J.: Some properties of layer potentials and boundary integral operators for the wave equation. J. Integral Equ. Appl. **25**(2), 253–294 (2013)
15. Domínguez, V., Lu, S., Sayas, F.J.: A fully discrete Calderón calculus for two dimensional time harmonic waves. Int. J. Numer. Anal. Model. **11**(2), 332–345 (2014)
16. Hackbusch, W., Kress, W., Sauter, S.A.: Sparse convolution quadrature for time domain boundary integral formulations of the wave equation. SIAM J. Numer. Anal. **29**(1), 158–179 (2009)
17. Hairer, E., Wanner, G.: Solving Ordinary Differential Equations. II. Springer Series in Computational Mathematics, vol. 14. Springer, Berlin (2010)
18. Hairer, E., Lubich, C., Schlichte, M.: Fast numerical solution of nonlinear Volterra convolution equations. SIAM J. Sci. Stat. Comput. **6**(3), 532–541 (1985)
19. Henrici, P.: Applied and computational complex analysis, vol. 1. Wiley Classics Library. Wiley, New York (1988)
20. Laliena, A.R.: Theoretical and algorithmic aspects of the convolution quadrature method applied to scattering of acoustic waves. Ph.D. thesis, Universidad de Zaragoza (2011)
21. Laliena, A.R., Sayas, F.J.: A distributional version of Kirchhoff's formula. J. Math. Anal. Appl. **359**(1), 197–208 (2009)
22. Laliena, A.R., Sayas, F.J.: Theoretical aspects of the application of convolution quadrature to scattering of acoustic waves. Numer. Math. **112**(4), 637–678 (2009)
23. Lubich, C.: Convolution quadrature and discretized operational calculus. I. Numer. Math. **52**(2), 129–145 (1988)
24. Lubich, C.: Convolution quadrature and discretized operational calculus. II. Numer. Math. **52**(4), 413–425 (1988)
25. Lubich, C.: On the multistep time discretization of linear initial-boundary value problems and their boundary integral equations. Numer. Math. **67**(3), 365–389 (1994)
26. Lubich, C., Ostermann, A.: Runge-Kutta methods for parabolic equations and convolution quadrature. Math. Comput. **60**(201), 105–131 (1993)
27. Lubich, C., Schneider, R.: Time discretization of parabolic boundary integral equations. Numer. Math. **63**(4), 455–481 (1992)
28. McLean, W.: Strongly elliptic systems and boundary integral equations. Cambridge University Press, Cambridge (2000)
29. Sayas, F.: Retarded Potentials and Time Domain Boundary Integral Equations: A Road-Map. Springer Series in Computational Mathematics, vol. 50. Springer, New York (2016). http://math.udel.edu/~fjsayas/TDBIEclassnotes2012.pdf (2013)
30. Sayas, F.J.: Energy estimates for Galerkin semidiscretizations of time domain boundary integral equations. Numer. Math. **124**(1), 121–149 (2013)
31. Schanz, M.: Wave Propagation in Viscoelastic and Poroelastic Continua: A Boundary Element Approach. Springer, Berlin/New York (2001)
32. Schwartz, L.: Méthodes mathématiques pour les sciences physiques. Enseignement des Sciences. Hermann, Paris (1961)
33. Trèves, F.: Topological vector spaces, distributions and kernels. Academic, New York/London (1967)

# Part II
# Modeling and Applications

# Mathematical Methods in Image Processing and Computer Vision

**Antonio Baeza**

**Abstract** Image processing and computer vision are growing research fields that take advantage of the increasing power or modern computers linked with sophisticated techniques coming from many fields of expertise and in particular from mathematics. We present an introduction to some problems in computer vision and image processing and to some mathematical techniques and concepts that are nowadays routinely used to approach them.

**Keywords** Computer vision • Image processing • Variational methods

## 1 Introduction

There is an old debate about how to name the application of different algorithms to images with various goals: image analysis, image processing, computer vision, machine vision and artificial vision are some of them. Albeit there is no agreement about it, in these notes we will assume the division into two categories: on the one hand, image processing, understood as a form of signal processing where the input is an image or a video and the output can be either a modified image or video or a set of low level image characteristics; on the other hand, computer vision is also a form of signal processing with the same input (images or videos) but with an output composed by high level information obtained from the images. The division based on low level image parameters and high level information is sometimes also blurry. Image processing includes areas like image deconvolution and denoising, interpolation or histogram equalization, whilst computer vision comprises techniques like object segmentation and categorization or motion analysis. Roughly speaking, in image processing we consider actions on the image that do not try to understand

A. Baeza (✉)
Facultat de Matemàtiques, Departament de Matemàtica Aplicada, Universitat de València, València, Spain
e-mail: antonio.baeza@uv.es



I. Higueras et al. (eds.), *Numerical Simulation in Physics and Engineering*,
SEMA SIMAI Springer Series 9, DOI 10.1007/978-3-319-32146-2_3

what is happening in the image, while in computer vision one of the goals is precisely to analyze and understand the image contents.

The applications of image processing and computer vision are enormous, and even the non-expert can cite a few. Instagram with its image filters is a clear example of image processing; Facebook uses algorithms for face detection to identify persons in images that its users can label; any camera device – even smartphones – includes a set of image enhancement tools, and any owner of a DSLR camera knows about Photoshop or GIMP. We can then conclude that it is worth working on image processing and computer vision because of the big number of applications inherent to them. But for many people the main reason to work with images is the amount of potential applications that they have, i.e. the set of non-existing or underdeveloped applications that can be imagined. For example, most of the image forensic techniques that appear in TV shows like CSI are unrealistic with the technologies available nowadays, but it is accepted by the community that some of them will become possible sooner or later.

The goal of these notes is to describe some problems that appear in image processing and computer vision applications, as illustrative of the kind of problems that scientists are facing and the mathematical tools that can be used to solve them as a motivation for the reader interested in these research fields. A review of the main mathematical methods that are nowadays used in the field is completely out of the scope of these notes, and therefore we will focus on a class of methods, namely variational methods, that translate the problem into the minimization of a functional. Variational methods are popular and representative of the application of mathematics to engineering. In particular, methods based on total variation regularization will be at the center of these notes because of their successful application to many different problems.

## 2 Image Capture: Demosaicking

In order to be able to manipulate images with mathematical tools and algorithms we start by formalizing the image acquisition process into abstract concepts. An image is a function $I : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^k$, where $\Omega$ is typically a square and $k = 1$ for grayscale images or $k = 3$ for color images. Assume by the moment that the image is a color image in RGB format, so that given a pixel $x \in \mathbb{R}^2$, the value $I(x) \in \mathbb{R}^3$ represents the intensities of the red, green and blue channels of the image. Let us take a look to the image acquisition process by means of a digital camera. The camera has a capturing device, called a sensor, composed by an array of elements that are able to measure the amount of light that arrives to them with an (ideally) linear response. Most cameras incorporate a CCD (charge-coupled device) sensor as a device for image capturing.

Current consumer technology allows for sensor elements that measure the light intensity, essentially by counting the amount of photons that arrive to the sensor, but that are not able to differentiate between color channels. In order to be able to

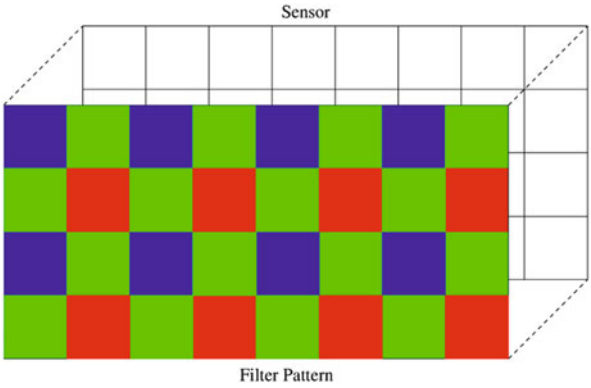


**Fig. 1** Diagram of a Bayer filter

obtain color images the most common approach consists on overlaying the sensor with a color filter array (CFA), so that only light corresponding to a given color channel arrives to the pixel. The most common filters are known as *Bayer* filters and are organized in a way similar to the one depicted in Fig. 1. Note that the filter pattern is 50 % green, 25 % red and 25 % blue, due to the fact that the human eye is more sensitive to green light that to red or blue light. The resulting raw image is therefore composed by an array of pixels measuring the amount or green, blue or red light received by that pixel. To obtain a normal color image in which each pixel is composed by three values (indicating the amount of green, blue and red light received by that pixel) an interpolatory procedure called demosaicking has to be applied. Not any interpolation can be used for demosaicking, as demonstrates the fact that a lot of different methods have been developed for this purpose [9, 11, 18] and many of them are protected by patents [5, 13, 17].

## 3 Image Restoration and Variational Methods

Let us analyze in this section the way how light arrives to the camera sensor. When taking a picture with a camera we do not just put the sensor in front of a light source, but the light crosses a lens or a combination of lenses that allows to control the way how it arrives to the sensor. In a simple but effective way, the overall process performed by the light reflected in an object when traveling from the source to the sensor (including the lens effects but also the action of the atmosphere on the light) can be modeled by means of a convolution operator, so that what is captured by the sensor is a convolved version of the real object. Thus it seems natural to try to restore the image through deconvolution. We will consider this problem as a model problem for introducing variational methods in image processing and computer vision.

The deconvolution problem consists on finding the unknown, "real" image $I$ from the observed image $f$ and possibly some knowledge about the convolution kernel $K$ that models the degradation of the image from the source to the sensor. In practice

the problem amounts to estimate a convolution kernel $K$ so that it holds

$$f = K * I.$$

Convolution is present in all images (due for example to image diffraction caused by finite aperture) but it is especially important in some areas where it cannot be avoided and produces significant image blurring. The areas where deblurring is important range from satellite and astronomical images, where the light that arrives to the sensor necessarily traverses the atmosphere, which is a source of image blur, to microscopical images, in which sometimes the nature of the microscope also introduces blur (for example, caused by out of focus light in confocal microscopy). The problem of image deconvolution depends on how much of the structure of $K$ is known beforehand (blind deconvolution if $K$ is unknown, myopic deconvolution if $K$ is partially known). The amount of incoming light measured by the image sensors is perturbed by parasitic heat photons and electrostatic fluctuations at the time of their loads and discharges. This is a random phenomenon called noise. The simplest model of noise is to assume that the noise $n$ is additive and has known or estimable mean and standard deviation. The problem than includes both convolution and noise is named image restoration and can be stated as: find $I$ such that

$$f = K * I + n. \tag{1}$$

The image restoration problem (1) is ill-posed due to the ill-conditioning of the convolution kernel $K$. Many techniques for image restoration exist in the literature (see [12]) and of course variational methods with regularization are among them. The idea of regularized variational methods is to minimize a functional that includes a term that enforces the difference between $f$ and $K * I$ to be small (data fidelity term) plus a regularization term that selects the solution that is "more regular" in some sense. We discuss some generalities about these methods with particular emphasis in total variation regularization in the next section. For further details about regularization of ill-posed problems in Hilbert spaces we refer to [25].

## *3.1 Regularized Variational Methods for Image Restoration*

The simplest option for a regularization term is probably given by

$$\int_{\Omega} |\nabla I|^2 \, dx, \tag{2}$$

where $|.|$ represents the $\ell_2$ norm. The expression (2) is the semi-norm of the Sobolev space $H^2(\Omega)$, the space of square integrable functions whose first order

weak derivatives are square integrable.[1] Functionals based on (2) fulfill general requirements of regularity, but in practice the solutions obtained from them suffer from excessive edge smearing (see below). As an alternative, total variation is defined for a differentiable function as

$$TV(u) = \int_{\Omega} |\nabla I| \, dx \tag{3}$$

and was introduced in [24] as an improvement over the Sobolev semi-norm for the case of image denoising, which is a particular case of image restoration with $K$ being the identity: find $I$ such that

$$f = I + n.$$

Regularization based on total variation can produce solutions that preserve the image edges to some extent. The total variation of a jump discontinuity does not decrease if it is smoothed, in contrast with the Sobolev semi-norm, hence total variation can be considered, to some extent, as an edge-preserving regularizer. It can be easily explained why this happens by taking the following example: take $\varepsilon > 0$ and consider the function defined by

$$I(x) = \begin{cases} 0, & \text{if } x < -\varepsilon, \\ 1, & \text{if } x > \varepsilon, \\ \frac{x+\varepsilon}{2\varepsilon}, & \text{if } -\varepsilon \leq x \leq \varepsilon, \end{cases} \tag{4}$$

whose graph is plotted in Fig. 2. It can be checked that the Sobolev semi-norm of $I(x)$ is given by $\frac{1}{2\varepsilon}$, while its total variation is equal to 1 regardless the value of $\varepsilon$.

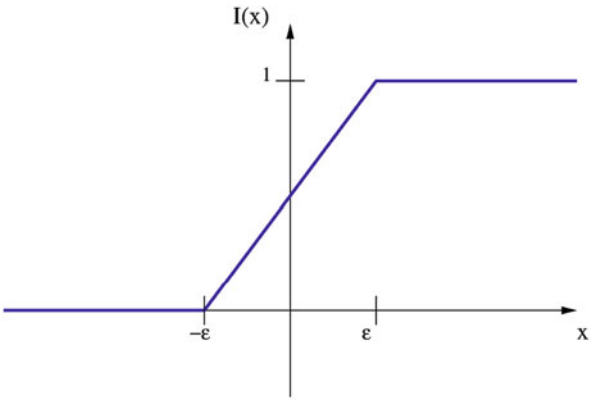


**Fig. 2** Graph of the function $I(x)$ in (4)

[1] All derivatives have to be understood in a weak sense throughout the text.

If one minimizes a functional that includes the Sobolev semi-norm it becomes clear that functions with smaller gradients (higher $\varepsilon$) will be preferred over functions with deeper gradients as the functional takes smaller values. If instead of the Sobolev semi-norm the functional includes the total variation then there is no preference for one solution or the other. This feature is the main reason why total variation is widely used in image processing nowadays. The main drawback of using the total variation instead of the Sobolev semi-norm as a regularizer is the fact that it becomes difficult to devise numerical methods based on the Euler-Lagrange equations, as we analyze below.

In the case of the denoising problem the variational formulation proposed in [24] is

$$\inf_I \left\{ \int_\Omega |\nabla I| \, dx + \frac{\lambda}{2} \int_\Omega |I - f|^2 \, dx \right\} \tag{5}$$

where $\lambda$ is a parameter that represents a trade-off between regularization and data fidelity. The natural extension of (5) to the image restoration problem is the following: minimize with respect to $I$ the functional

$$\int_\Omega |\nabla I| \, dx + \frac{\lambda}{2} \int_\Omega |K * I - f|^2 \, dx. \tag{6}$$

In wider generality, if $I$ is any feature of the image and $f$ is a prior or observation, then (5) can be seen as a functional that requires the solution $I$ to be smooth and close to the prior $f$ at the same time.

The basic strategy to find the infima of (6) consists on finding the solutions of its Euler-Lagrange equation. Let us assume the case $K = I$ for simplicity. The Euler-Lagrange equation of the functional in (5) can be written as:

$$-\nabla \cdot \left( \frac{\nabla I}{|\nabla I|} \right) + \lambda (I - f) = 0. \tag{7}$$

Note that the equation is not well defined when $\nabla I = 0$. This is the main drawback when using total variation as a regularizer with respect to the Sobolev semi-norm, which has a well-defined Euler-Lagrange equation. In fact the Euler-Lagrange equation for the functional

$$\int_\Omega |\nabla I|^2 \, dx + \frac{\lambda}{2} \int_\Omega |I - f|^2 \, dx,$$

is given by $\nabla^2 I + \lambda (I - f) = 0$, with $\nabla^2$ indicating the Laplacian. To overcome the singularity that appears in (7) a simple solution is to add a small perturbation to the definition of the total variation and take

$$TV_\alpha(I) = \int_\Omega \sqrt{|\nabla I|^2 + \alpha} \, dx$$

for small $\alpha > 0$ and solve

$$\inf_I \left\{ TV_\alpha(I) + \frac{\lambda}{2} \int_\Omega |I - f|^2 \, dx. \right\} \tag{8}$$

whose solution converges to the minimizers of (6) as $\alpha \to 0$. Albeit the term $\sqrt{|\nabla I|^2 + \alpha}$ replaces $|\nabla I|$ in (7) the resulting Euler-Lagrange equation

$$-\nabla \cdot \left( \frac{\nabla I}{\sqrt{|\nabla I|^2 + \alpha}} \right) + \lambda (I - f) = 0$$

is still hard to solve because of its high non-linearity. Successful methods to minimize the total variation rely on the dual formulation of the functional. In [3] the authors introduce an algorithm that uses duality for solving the extension of (8) to the image restoration problem. Chambolle [1] proposed a simpler algorithm, also based on duality, for the case of denoising.

The duality argument starts by writing the total variation in variational form as

$$\int_\Omega |\nabla I| \, dx = \sup \left\{ \int_\Omega I \operatorname{div} p \, dx : p \in C_0^1(\Omega, \mathbb{R}^2), \ |p(x)| \leq 1 \ \forall x \in \Omega \right\}, \tag{9}$$

where div stands for the divergence operator and $C_0^1$ is the space of continuously differentiable functions with compact support in $\Omega$. Formula (9) is a more general definition of total variation which is equivalent to (3) for smooth $I$. Problem (5) then reads

$$\inf_I \sup_p \int_\Omega I \operatorname{div} p \, dx + \frac{\lambda}{2} \int_\Omega (I - f)^2 \, dx, \tag{10}$$

which is a primal–dual problem. Switching the infimum and supremum:

$$\sup_p \inf_I \int_\Omega I \operatorname{div} p \, dx + \frac{\lambda}{2} \int_\Omega (I - f)^2 \, dx. \tag{11}$$

For fixed $p$ the optimization of (11) with respect to $I$ is known as primal problem. Its optimum $I^*$ can be found explicitly by deriving (11) with respect to $I$ and equaling to 0 point per point:

$$\operatorname{div} p + \lambda (I^* - f) = 0,$$

which leads to

$$I^* = f - \frac{1}{\lambda} \operatorname{div} p.$$

Now the optimal $I^*$ can be plugged into (11) to get

$$\sup_p \int_\Omega \left( f \operatorname{div} p - \frac{1}{2\lambda} (\operatorname{div} p)^2 \right). \tag{12}$$

Multiplying by $2\lambda$ and adding the term $\lambda^2 f^2$ we get that the optimizers of (12) are the same of equation

$$\sup_p \int (\lambda f - \operatorname{div} p)^2, \tag{13}$$

which is the dual problem. Chambolle's method relies on an iterative algorithm for solving (13). Another option is to alternate the optimization of the primal and dual problems assuming fixed the other variable, as proposed in [28]. Following this latter approach, recall that (10) can be written as

$$\inf_I \sup_p \left\{ - \int_\Omega \nabla I \cdot p \, dx + \frac{\lambda}{2} \int_\Omega (I - f)^2 \, dx \right\},$$

so that the functional is linear on $p$. Therefore a simple gradient descent/ascent iteration for $p$ is not possible and a proximal point type term [23] is added. The algorithm starts with initial guesses $I^0$ and $p^0$ for $I$ and $p$, and iterates the following steps until convergence.

- Primal Step:

$$I^{k+1} = \arg\min_I \left\{ \int I \operatorname{div} p^k + \frac{\lambda}{2} \int (I - f)^2 + \frac{1}{2\tau_I} \int (I - I^k)^2 \right\}. \tag{14}$$

- Dual Step:

$$p^{k+1} = \arg\max_p \left\{ \int I^{k+1} \operatorname{div} p - \frac{1}{2\tau_p} \int (p - p^k)^2 \right\}. \tag{15}$$

The parameters $\tau_I$ and $\tau_p$ are the proximal point step sizes. The solutions of (14) and (15) can be computed explicitly and are given by

$$I^{k+1} = \frac{\tau_I}{1 + \lambda \tau_I} \left( \lambda f + \frac{I^k}{\tau_I} - \operatorname{div} p^k \right)$$
$$p^{k+1} = \mathcal{P}(p^k - \tau_p \nabla I^{k+1}),$$

where $\mathcal{P}$ is the projection on the set $\{p : |p| \leq 1\}$ by

$$\mathcal{P}(p) = \frac{p}{\max\{1, |p|\}}.$$

Equations (5) and (6) represent particular cases of a more general variational model that can be stated as: find a function $u$ which is smooth in some sense and is close to some prior given by a function of the observation $f$. The problem is then the minimization of functionals composed by a regularization term $R(u)$ and a data attachment term $F(u,f)$. The general variational problem is then stated as[2]

$$\min_u R(u) + F(u,f). \tag{16}$$

A global analysis of problem (16) is out of the scope of these notes. Instead, we will present some model problems from computer vision that represent limited but representative test cases and we will analyze some of them in detail. Some other approaches (not necessarily based on variational models) will be presented as well, so as to give a wider view of mathematical methods applied to computer vision problems.

## 4 Some Model Problems in Computer Vision

Let us assume that after some pre-processing we have our restored color images ready to be processed. There are many actions that we might want to perform on images or sets of images. The distinction between single and multiple images is meaningful, since having more than one image usually implies that the knowledge acquired from one image is applied to, or combined with, another image. Typical multi-image setups are: (1) video streams captured with a video camera; and (2) video streams (or snapshots) captured synchronously with many cameras. In what follows we introduce some classical problems in computer vision that represent challenges for the scientific community and are fields of active research. In particular, both monocular and multi-image problems will be presented in the next sections.

### *4.1 Image Segmentation*

Image segmentation aims to split an image into several meaningful parts called segments. Mathematically the problem can be stated as follows: given an image $I : \Omega \to \mathbb{R}^k$ ($k = 1$ or 3) find a partition

$$\Omega = \Omega_1 \cup \Omega_2 \cup \cdots \cup \Omega_n \cup \Gamma,$$

[2]Note that the unknown is now named $u$ and hence $I$ might denote the input image from now on.


**Fig. 3** An example of interactive segmentation. (**a**) Input image with scribbles. (**b**) Segmentation result

where each $\Omega_i$ represents a segment and $\Gamma$ is the boundary set composed by the union of the boundaries of the segments

$$\Gamma = \partial\Omega_1 \cup \cdots \cup \partial\Omega_n.$$

Any partition of the image is a solution of the general segmentation problem and therefore some a priori knowledge about which characteristics define a segment is necessary in order to decide which pixels correspond to each segment. Professional segmentation tools often interact with a human user so as to learn which characteristics define a segment. Typical interactive tools use scribbles to mark whether a region belongs or not to a segment. An example obtained through the interactive segmentation tool[3] described in [20] can be seen in Fig. 3.

For some purposes automatic segmentation, based exclusively on image characteristics like color or sharp edges can be useful as well. One might think that for cases where different objects have different colors it would be easy to segment them just by thresholding in color space. It is really easy for the human eye to do such a task, but it is not the case for a computer, as the reader can verify by experimenting with the *select by color* tools of any image manipulation program. More constraints

[3] http://structuralsegm.sourceforge.net/

have to be added to the model in order to be able to properly segment an image. Typical constraints are based on the following assumptions, among others:

- $I$ varies smoothly in each $\Omega_i$ (pixels with similar colors belong to the same segment).
- $I$ has deep variations across $\Gamma$ (pixels with dissimilar colors belong to different segments).
- The boundary $\Gamma$ is smooth (objects have smooth boundaries).
- The image does not contain very small segments.

The key reference for variational modeling of segmentation is the classical paper of Mumford and Shah [19], where the authors propose to minimize the following functional

$$\int_{\Omega\setminus\Gamma} |\nabla u|^2\, dx + \lambda \int_{\Omega} (u-I)^2 + \mu \mathcal{H}_1(\Gamma) \tag{17}$$

with respect to $u$ and $\Gamma$, where $\mathcal{H}_1$ denotes the Hausdorff measure of $\Gamma$ and $\lambda, \mu$ are positive parameters. Written in this way, a minimizer $u_{\min}$ of (17) is a function that tends to be piecewise smooth. It can be seen as a simplified version of the image with reduced texture. Quantization of the minimizer and an analysis of the boundary $\Gamma$ are routinely performed after minimization so as to produce an output composed by pixel values that can be considered as labels that indicate the segment each pixel belongs to.

Let us analyze the three terms that appear in (17). The first term is a regularization term. Note that the integral is not acting on the image boundaries and hence it does not smooth out image edges. The regularization is complemented by the third term which precisely acts on the image edges, penalizing the total edge length. This prevents the image to be split into many small regions. Additionally, it implicitly imposes smoothness of the boundaries, since objects with irregular boundaries have bigger perimeter. The second term is the data fidelity term, which forces the segmentation to be close to the original image, stating that the segmentation will be based on color or intensity coherence. Note that all three terms are necessary in order to obtain non-trivial solutions. In fact:

- $u = I$, $\Gamma = \emptyset$ is a solution (with zero energy) if the first term is dropped. This solution corresponds to segmentation in just one region which is the entire image.
- $u =$ constant, $\Gamma = \emptyset$ is a solution (with zero energy) if the second term is dropped. It also corresponds to segmentation in a single region.
- A solution with zero energy can be obtained by taking each pixel as a different segment if the third term is dropped.

It is difficult to operate with the Mumford-Shah functional because of the individual treatment of the edge set $\Gamma$. Many techniques have been derived from the original Mumford-Shah work to tackle different problems. As an example, an algorithm for solving a modified version for binary segmentation (segmentation in just two segments, say objects and background) was described by Chan and Vese

for scalar images [2]. The problem is posed under the additional assumption that $u$ is constant on each segment. An extension for color images was presented in [4].

## 4.2 Depth Computation from Multiple Images

In this subsection we introduce the problem of depth computation from multiple images as a model for many techniques commonly used in computer vision, in particular for the so-called labeling problems. In these problems a numeric label is assigned to each pixel in a way such that the label configuration minimizes an energy. Labeling problems can be binary, meaning that just two possible labels can be assigned or multi-label if more than two labels exist. The classification is meaningful as binary problems are much easier to solve than multi-label ones in general (the interested reader can refer to [16] and references therein). Recall that the segmentation problem introduced in Sect. 4.1 is actually an example of labeling problem.

The problem of depth computation aims to answer the following question: given two or more cameras that record a scene from different viewpoints, what are the distances (also called depths) from the objects in the scene to the cameras? It is impossible to infer such a distance from a single image without any other prior (for example, the true size of the objects). That is the reason why most animals have two eyes. The parallax produced by the eye distance and a complex process performed in the brain allows us to deduce relative distances of the objects that we observe. The goal of depth computation is to reproduce such a process on a computer, using the cameras as eyes. The setup that tries to exactly reproduce the natural vision is the so-called disparity problem or stereo correspondence problem. In this scenario two cameras are placed along the same horizontal axis so that they play the role of the two eyes. Then, from the horizontal displacement of the objects from one image to the other the algorithms try to infer the distance from the object to the system. This simplest setup is of utmost importance as it is used for many applications like the creation of 3D movies.

In general, problems where a scene is recorded with multiple cameras from different viewpoints are known under the generic name of multi-view processing and compose a major field of study in computer vision. Basic references for multi-view processing are the handbooks by Hartley and Zisserman [14] and by Faugeras and Luong [8].

The problem of depth computation can be stated as follows: Consider $n$ images $I_1, \ldots, I_n$, obtained with $n$ cameras $C_1, \ldots, C_n$ located in different positions, but recording (at least partially) the same scene. Compute the depth of each pixel $(x, y)$ in each image $I_i$, i.e., the distance $d_i(x, y)$ of the object that is seen at pixel $(x, y)$ in camera $C_i$, to the camera.

The depths compose a depthmap for each camera, which has the same dimensions as the camera image. The depth of a pixel with respect to a camera is therefore a real number. In order to be able to handle the problem with a computer the continuous range of possible depths is discretized into a finite number of target depths, resulting in a multi-label problem.

To be able to solve the depth estimation problem the cameras need to be calibrated. To calibrate a camera means to know its position and orientation with respect to a reference frame as well as its internal geometry defined through a set of parameters. Camera calibration is a field of study in itself and its description is out of the scope of these notes. We refer the interested reader to [14]. In summary, having the cameras calibrated means that it is possible to know which camera pixel is recording a given 3D point, and conversely, from the 2D coordinates of an image pixel and its depth, we can obtain the 3D coordinates of the point in 3D that projects on the given pixel and is located at distance $d$ from the camera. In its simpler form calibration of a camera $C_i$ is given by a matrix $P_i$ in a way such that it holds $P_i(z) = (x, y)$, where $z$ is a 3D point and $(x, y)$ is the pixel where this 3D point projects on camera $C_i$. An illustration of these relations between the 3D world and the 2D camera pixels is shown in Fig. 4. In the figure, cameras are represented by a line indicating the camera plane (which is actually a plane in 3D) and a point for the camera center so that a 3D point $z$ is imaged at the pixel where the line joining the point $z$ and the camera center intersects its image plane.

Note that every 3D point located in the line joining the pixel $(x, y)$ and the camera center projects on $(x, y)$, thus $P$ can only be invertible if we fix a distance $d$ to the camera. In this case $P_i^{-1}(x, y, d)$ denotes the 3D point that verifies $P_i(z) = (x, y)$ and is located at distance $d$ from camera $C_i$. The knowledge of the projection matrices $P_i$

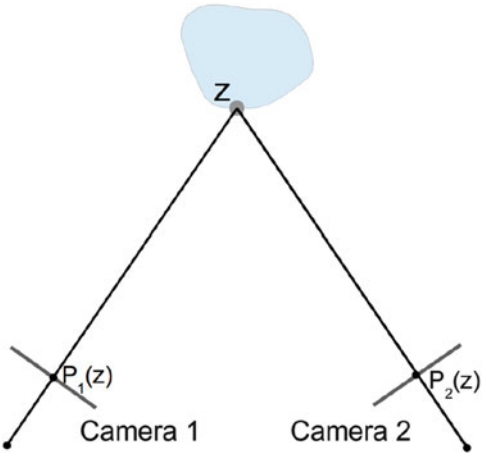


**Fig. 4** Relation between the 3D world and the 2D camera pixels

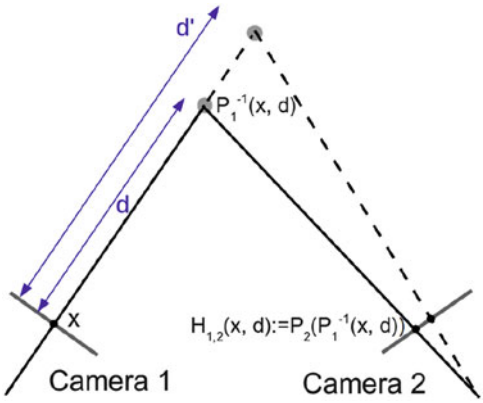


**Fig. 5** Relation between the 3D world and the 2D camera pixels

and $P_j$ corresponding to a pair of cameras $C_i$ and $C_j$ induces a family of applications parametrized by the depth $d$ by

$$H_{ij}(x, y, d) = P_j(P_i^{-1}(x, y, d)).$$

These applications are called homographies and relate pixels in both images assuming that the object imaged in both pixels is at distance $d$ from camera $C_i$. This concept is illustrated in Fig. 5.

Methods for depth estimation are often based on correlations between the observations made by many cameras. For a fixed camera $C_i$ a suitable range of possible depths is discretized into $M$ target depths $d_{\min} = d_1 < \cdots < d_M = d_{\max}$ and for each pixel a test is performed to decide which one among all possible depths best fits to the observations, based on a certain matching score. The observations are of course the images, so color or intensity comparisons between images are used as a decision mechanism. If one adopts a winner-takes-all approach, where the depth with best score is kept as correct, the procedure is called plane sweeping [6]. Different choices for the assignment of a score to a target depth result in different algorithms. Let us assume the simplest case where two cameras are available and the images are recorded in grayscale. Assume that $C_1$ is the reference camera (the one for which we compute depths) and $C_2$ the auxiliary camera (the one used for comparison). Then, a basic plane-sweeping algorithm based on intensity consistency amounts to find, for each pixel $(x, y) \in I_1$ and some functional $J : \mathbb{R} \to \mathbb{R}^+$ the solution of

$$\min_d J(I_1(x, y) - I_2(H_{12}(x, y, d))).$$

The simplest choice is probably to take $J$ as the absolute value, which corresponds to evaluate the intensity difference between corresponding pixels, but this method is far too sensitive to small errors due to calibration and neither is robust to changes in contrast between the images. A better choice is to take $J = SAD(x, y, d)$ (Sum of Absolute Differences) on a patch:

$$SAD(x, y, d) := \sum_{i,j=-s}^{s} |I_1(x + i, y + j) - I_2(H_{12}(x, y, d) + (i, j))| .$$

It is already a simple formula, with an evaluation cost depending on the half-patch size $s$, and partially robust to calibration errors but not to contrast changes. A still better method is based on the normalized cross-correlation on a patch $J = NCC(x, y, d)$ defined by

$$NCC(x, y, d) := \frac{\sum_{i,j=-s}^{s} I_1(x + i, y + j) \cdot I_2(H_{12}(x, y, d) + (i, j))}{\sqrt{\sum_{i,j=-s}^{s} I_1(x + i, y + j) \cdot \sum_{i,j=-s}^{s} I_2(H_{12}(x, y, d) + (i, j))}}. \tag{18}$$

The normalization represented by the term at the denominator induces robustness with respect to contrast changes while maintaining low sensitivity to calibration errors. The evaluation of (18) has a higher complexity but plane-sweeping algorithms based on brute-force minimization can be nowadays run in real-time thanks to the use of GPU computing techniques [10].

### 4.2.1 Variational Formulation

An improvement over the plane-sweeping method consists on adding smoothness to the functional. Piecewise smooth depthmaps correspond to the situation where nearby pixels are located at similar depth as they correspond to the same object, allowing abrupt depth changes only across image edges. This is a reasonable assumption in general and, as discussed above, total variation can be used as a regularizer as it possesses the aforementioned properties. Hence, the multi-label depth computation problem can be stated as the minimization of a functional $J(u)$ defined by

$$J(u) := \int_{\Omega} |\nabla u(x, y)| \, dx \, dy + \int_{\Omega} \rho(x, y, u(x, y)) \, dx \, dy \tag{19}$$

where $u(x, y)$ is the depth of pixel $(x, y)$ with respect to the reference camera and $\rho(x, y, u(x, y))$ is the cost of assigning depth $u(x, y)$ to pixel $(x, y)$. Any of the matching costs discussed above for the plane-sweeping algorithm can be used to define $\rho$, possibly multiplied by a constant to tune the relative influence of the regularization and the data term. In the above definition we deliberately changed

the variable name from $d$ to $u$ to assess the fact that the same functional, for suitable $\rho$, can be used in other multi-label problems.

Minimization of (19) is difficult to perform because it is not convex in general. A method for the convexification of functionals of the form (19) was introduced in [22], where the authors consider an interval $\mathcal{U} = [u_{\min}, u_{\max}]$ defining the range of admissible depths and define the function:

$$\phi(x, y, s) = H(u(x, y) - s),$$

where $H$ is the Heaviside function defined by $H(t) = 1$ if $t > 0$ and $H(t) = 0$ otherwise. The function $\phi$ can be interpreted as the characteristic function of the subgraph of $u$. A first observation is that $\phi$ is a decreasing function of $s$ and that $\phi(x, y, u_{\min}) = 1, \quad \phi(x, y, u_{\max}) = 0$. The following layer-cake type formula allows to recover $u$ from $\phi$:

$$u(x, y) = u_{\min} + \int_{\mathcal{U}} \phi(x, y, s) ds. \tag{20}$$

The goal is to rewrite the functional $J(u)$ in (19) in terms of $\phi$. Let $H_{n-1}$ denote the $(n-1)$-dimensional Hausdorff measure and let $\delta$ be the Dirac delta function. Either from the co-area formula

$$\int_{\Omega} |\nabla u| = \int_{-\infty}^{+\infty} H_{n-1}(u^{-1}(t)) dt,$$

or using

$$\nabla_x \phi = \delta(u(x, y) - s) \nabla u,$$

we can write the total variation of $u$ in terms of $\phi$ as

$$\begin{aligned} \int_{\Omega} |\nabla u(x, y)| \, dx\, dy &= \int_{\Omega} \left\{ \int_{\mathcal{U}} \delta(u(x, y) - s) \, |\nabla u(x, y)| \, ds \right\} dx\, dy \\ &= \int_{\Omega} \left\{ \int_{\mathcal{U}} |\nabla_x \phi(x, y, s)| \, ds \right\} dx\, dy. \end{aligned}$$

On the other hand, from the definition of $\phi$ it follows that

$$|\partial_s \phi(x, y, s)| = \delta(u(x, y) - s),$$

which allows to write the data term as

$$\begin{aligned}\int_{\Omega} \rho(x,y,u(x,y))\,dx\,dy &= \int_{\Omega} \left\{ \int_{\mathcal{U}} \rho(x,y,s)\delta(u(x,y)-s)\,ds \right\} dx\,dy \\ &= \int_{\Omega} \left\{ \int_{\mathcal{U}} \rho(x,y,s)\,|\partial_s \phi(x,y,s)|\,ds \right\} dx\,dy.\end{aligned}$$

Summarizing, the functional

$$J(u) := \int_{\Omega} |\nabla u(x,y)|\,dx\,dy + \int_{\Omega} \rho(x,y,u(x,y))\,dx\,dy$$

can be written in terms of $\phi$ as

$$\int_{\Sigma} \{|\nabla_x \phi(x,y,s)| + \rho(x,y,s)\,|\partial_s \phi(x,y,s)|\}\,d\Sigma,$$

where we have denoted $\Sigma = \Omega \times \mathcal{U}$. The functional

$$E(\phi) := \int_{\Sigma} \{|\nabla_x \phi(x,y,s)| + \rho(x,y,s)\,|\partial_s \phi(x,y,s)|\}\,d\Sigma$$

is convex in $\phi$. The natural function space where to carry the minimization of $E(\phi)$ is the set $D'$ of binary functions that are strictly decreasing on their third variable:

$$D' = \{\phi : \Sigma \to \{0,1\} \mid \phi(x,y,u_{\min}) = 1, \phi(x,y,u_{\max}) = 0\}.$$

Since $D'$ is a non-convex set, the minimization problem

$$\min_{\phi \in D'} E(\phi)$$

is still not convex. To circumvent this difficulty $\phi$ is relaxed to allow continuous values between 0 and 1 and the minimization of $E(\phi)$ is performed on the set

$$D = \{\phi : \Sigma \to [0,1] \mid \phi(x,y,u_{\min}) = 1, \phi(x,y,u_{\max}) = 0, \partial_s \phi < 0\}.$$

The problem

$$\min_{D} \left\{ \int_{\Sigma} \{|\nabla_x \phi(x,y,s)| + \rho(x,y,s)\,|\partial_s \phi(x,y,s)|\}\,d\Sigma \right\} \tag{21}$$

is now a convex problem. It was shown in [22] that if $\phi$ is the minimizer of $E(\phi)$ on $D$ and $\chi$ denotes the characteristic function, then the binary function $\phi^* = \chi_{\{\phi > \mu\}}$ is a minimizer of $E(\phi)$ on $D'$ for almost any $\mu \in [0,1]$.

A minimizer for $J(u)$ is finally obtained through the formula (20) as

$$u^*(x,y) = u_{\min} + \int_{\mathcal{U}} \phi^*(x,y,s)ds.$$

As a summary, the minimization of the non-convex functional (19) has been translated into a convex problem given by (21) at the price of increasing the dimensionality of the problem.

The minimization of (21) can be performed by means of a primal-dual strategy analogous to the one described in Sect. 3.1. In this case we have that

$$|\nabla_x \phi| + \rho\,|\partial_s \phi| = \max\left\{\phi \cdot \operatorname{div} p : \sqrt{p_1^2 + p_2^2} < 1, |p_3| < \rho\right\}.$$

Therefore, the primal–dual form of the problem is

$$\min_{\phi \in D} \max_{p \in C} \int \phi \operatorname{div} p,$$

with

$$C = \left\{ p : \mathbb{R}^2 \times [a,b] \;\middle|\; \sqrt{p_1^2 + p_2^2} < 1,\ |p_3| < \rho \right\}.$$

Adding a proximal point term the following primal–dual iteration is obtained. Starting from initial guesses $\phi^0$ and $p^0$ the algorithm iterates the following steps until convergence:

- Primal Step:

$$\phi^{k+1} = \arg\min_{\phi \in D} \left\{ \int \phi \operatorname{div} p^k + \frac{1}{2\tau_\phi} \int (\phi - \phi^k)^2 \right\}. \tag{22}$$

- Dual Step:

$$p^{k+1} = \arg\max_{p \in C} \left\{ \int \phi^{k+1} \operatorname{div} p + \frac{1}{2\tau_p} \int (p - p^k)^2 \right\}. \tag{23}$$

The solution of (22) is given by

$$\phi^{k+1} = \mathcal{P}_D\left(\phi^k - \tau_\phi \operatorname{div} p\right),$$

where $\mathcal{P}_D$ is the projection to the feasible set $D$ and can be computed by

$$\mathcal{P}_D(\phi)(x,s) = \begin{cases} 0, & \text{if } s = s_{\max}, \\ 1, & \text{if } s = s_{\min}, \\ \max\{0, \min\{\phi(x,s), 1\}\}, & \text{otherwise.} \end{cases}$$

On the other hand, the solution of (23) is given by

$$p^{k+1} = \mathcal{P}_C(p^k - \tau_p \nabla \phi^{k+1}),$$

where $\mathcal{P}_C$ is the projection of $p$ on to the set $C$ and can be computed by

$$\mathcal{P}_C(p)_1(x) = \frac{p_1(x)}{\max\{1, \sqrt{p_1(x)^2 + p_2(x)^2}\}},$$

$$\mathcal{P}_C(p)_2(x) = \frac{p_2(x)}{\max\{1, \sqrt{p_1(x)^2 + p_2(x)^2}\}},$$

$$\mathcal{P}_C(p)_3(x) = \frac{\rho(x)\, p_3(x)}{\max\{\rho(x), |p_3(x)|\}}.$$

### *4.3 3D Reconstruction from Depths*

In this subsection we briefly describe the application of the primal-dual algorithm described at the end of Sect. 3.1 to the problem of 3D reconstruction of scenes. We assume a multi-camera system and that depthmaps are available for at least $N \geq 2$ cameras in the system. In such a situation one can try to recover the 3D scene that has been observed by the cameras. An initial attempt to figure out the real 3D scene is to generate a point cloud composed by the 3D points that result from putting at their respective depths the pixels of the cameras for which depthmaps are available. If we repeat the process for many cameras, we will incrementally complete the scene as each camera observes a different part of it. Unfortunately, the depthmaps contain mismatches due to many factors (mainly calibration errors and wrong depths) and therefore such a deterministic approach would result in a noisy representation of the scene. The usual approach for 3D reconstruction from multiple cameras is to compute the set of 3D surfaces (representing the boundaries of the true scene objects) that most likely will produce the observed depthmaps. This approach is known as 3D reconstruction by depth merging.

Mathematically, if we denote by $P_i$ the operator that projects a 3D point on the 2D image $I_i$ of camera $C_i$, (i.e. $P_i(z) = x$ if the 3D point $z$ corresponds to pixel $x$ in $I_i$) and $d_i(x)$ is the depth computed for pixel $x$ with respect to $C_i$, then the problem can be stated as: find a bi-dimensional surface $S \subset \mathbb{R}^3$ such that if $z \in S$ then the

observed depths $d_i(P_i(z))$ are as coherent as possible with $r_i(z) := \text{dist}(z, C_i)$, for all $i$.

In this context, the particular definition of coherence gives rise to many different approaches for depth merging. We describe here a model based on taking as coherence criterion a weighted sum for all cameras of squared errors computed with respect to an initial reconstruction and adds a smoothness constraint via the total variation. The goal is to decide if a given 3D point is outside, inside or at the boundary of an object in the scene. To this aim objects in 3D space are represented through an occupancy function, which acts as an indicator of whether a point is inside or outside the objects in the scene. An occupancy function $u : \Omega \subset \mathbb{R}^3 \to \{-1, 1\}$ for a known scene can be defined by

$$u(z) = \begin{cases} -1, & \text{if } z \text{ is inside an object,} \\ 1, & \text{if } z \text{ is outside all objects.} \end{cases}$$

If the scene is unknown one can try to recover it by means of the minimization of a functional like

$$J(u) = \int_\Omega |\nabla u(z)| \, dz + \frac{1}{2\theta} \sum_{i=1}^{N} \int_\Omega w_i(z) \, (u(z) - f_i(z))^2 \, dz, \tag{24}$$

where $u : \Omega \subset \mathbb{R}^3 \to [-1, 1]$ is a relaxed occupancy function that can take values on the interval $[-1, 1]$, $f_i : \Omega \to [-1, 1]$ is a truncated distance function built from the depthmaps corresponding to the $i$-th camera as described below, $w_i : \Omega \to [0, 1]$ is a weighting function that ponders the contribution of each camera to the reconstruction and $\theta > 0$.

The algorithms act on a discrete discretization of a certain 3D volume into the 3D equivalent of pixels, called voxels (contraction of volume pixels). The functions $f_i$ can be defined in the following way: if $z$ is a voxel center and $x = P_i(z)$ take

$$f_i(z) = \begin{cases} 1, & \text{if } r_i(z) < d_i(x) - \frac{\delta}{2}, \\ -1, & \text{if } r_i(z) > d_i(x) + \frac{\delta}{2}, \\ \frac{2}{\delta}(d_i(x) - r_i(z)), & \text{if } d_i(x) - \frac{\delta}{2} \le r_i(z) \le d_i(x) + \frac{\delta}{2}, \end{cases}$$

for a fixed $\delta > 0$. The function $f_i$ takes the value 1 for voxels visible from camera $i$, the value $-1$ for voxels occluded by the object surface and an intermediate value for voxels near the surface. On the other hand the weight $w_i$ is defined as

$$w_i(z) = \begin{cases} 0, & \text{if } r_i(z) > d_i(x) + \eta, \\ 1, & \text{otherwise,} \end{cases}$$

for $\eta > \delta$. These weights measure the confidence on the values assigned by $f_i$ so that it takes the maximum confidence for voxels that are either visible or at a distance from the surface smaller than $\eta$. This construction is similar to the one in [7]. The function $u$ represents the 3D volumes in the scene by selecting the relaxed occupancy function which is more coherent with the observations $f_i$ in the sense of (24). The zero level set of $u$, given by $\{z \in \Omega : u(z) = 0\}$ is obviously the sought surface.

Each term of the form

$$\int_\Omega w_i(z)\,(u(z) - f_i(z))^2\,dz$$

measures the weighted squared mean error committed by taking a surface $u$ with respect to the observations $f_i$ for the $i$-th camera. On the other hand, the total variation regularizes this estimation so that the final result is composed by relatively smooth surfaces. Also, note that the functional is convex, and that its Euler-Lagrange equation coincides with the one of the functional

$$J(u) = \int_\Omega |\nabla u(z)|\,dz + \frac{1}{2\theta}\int_\Omega w(z)\,(u(z) - f(z))^2\,dz, \tag{25}$$

where $w(x)$ and $f(x)$ are defined by

$$w(z) = \sum_{i=1}^{N} w_i(z), \qquad f(z) = \frac{1}{w(z)}\sum_{i=1}^{N} w_i(z) f_i(z).$$

Therefore, both functionals have the same minima. Functional (25) can be minimized by means of a primal-dual procedure analogous to the one described in Sect. 3.1.

## 4.4 Optical Flow Estimation

In this section we conclude by briefly introducing the problem of optical flow estimation, an example of a non-convex problem in which the target function is vector-valued, in contrast with the previous problems, in which the unknown was always a scalar function.

Optical flow is the apparent motion of a scene (as observed by a camera) caused by the relative motion of the observer (the camera) and the scene. Ideally, it is the 2D

projection of the true 3D motion of the scene onto the image plane.[4] The case where the displacement in the image is one-dimensional is known as disparity estimation.

Natural applications of optical flow estimation are the tracking of objects (persons in an airport, cars in a highway, the ball in a tennis game, etc.). The input of the problem is given by a set of images $I(x,t)$ and the output is a vector field $d(x,t) = (u(x,t), v(x,t)) \in \mathbb{R}^2$ that represents the apparent motion of a pixel $x$ in the frame $t$ with respect to frame $t+1$.

The most known variational model for optical flow estimation was introduced by Horn and Schunk in the seminal work [15], and is based on the minimization of the functional:

$$\int_\Omega |\nabla d(x,t)|^2 \, dx + \lambda \int_\Omega (I(x+d, t+1) - I(x,t))^2 dx, \quad (\lambda > 0) \tag{26}$$

where the computation has to be done for each fixed $t$. The unknown is now the displacement $d$, but the functional is still in the general variational form (16). Many methods are based on the minimization of (26) or variations of it (see e.g. [27]). A more general formulation using total variation and a (possibly) non-convex data term would lead to the vector analogous of (19). The problem can be written as the minimization, for each $t$ of

$$J(u,v) = \int_\Omega \{|\nabla u(x)| + |\nabla v(x)|\} \, dx + \frac{1}{\lambda} \int_\Omega \rho(x, d(x)) \, dx,$$

with $\rho(x, d(x))$ being a suitable data attachment term, typically depending on

$$|I(x,t) - I(x + d(x), t+1)| \, .$$

An analysis of such a model can be found in [21]. The approach consists on introducing two functions $\phi(x,s) = H(u(x) - s)$ and $\psi(x,t) = H(v(x) - t)$, where $H$ denotes again the Heaviside function and $s, t \in \mathbb{R}$. Proceeding analogously to the scalar case, the functional can be written in terms of $\phi$ and $\psi$ in a way such that the resulting functional is polyconvex. Polyconvex functionals are quasiconvex and quasiconvexity is assumed to be the right extension of the notion of convexity for vector valued functions. Under certain assumptions, it guarantees the existence of minimizers and the well-posedness (in a certain sense) of the energies.

**Acknowledgements** The author is grateful to the organizers and scientific committee of the Jacques-Louis Lions Spanish-French school for the invitation. This research was Partially supported by Spanish MINECO grant MTM 2014-54388. This work is dedicated to the memory of Vicent Caselles.

[4] The determination of the true 3D motion of objects is known as scene flow and will not be analyzed in these notes. The interested reader can refer to [26] and references therein.

## References


1. Chambolle, A.: An algorithm for total variation minimization and applications. J. Math. Imaging Vis. **20**(1–2), 89–97 (2004)
2. Chan, T.F., Vese, L.A.: Active contours without edges. IEEE Trans. Image Process. **10**(2), 266–277 (2001)
3. Chan, T.F., Golub, G.H., Mulet, P.: A nonlinear primal–dual method for total variation–based image restoration. SIAM J. Sci. Comput. **20**(6), 1964–1977 (1999)
4. Chan, T.F., Sandberg, B.Y., Vese, L.A.: Active contours without edges for vector-valued images. J. Vis. Commun. Image Represent. **11**(2), 130–141 (2000)
5. Cok, D.R.: Signal processing method and apparatus for producing interpolated chrominance values in a sampled color image signal. U.S. Patent No. 4,642,678 (1987)
6. Collins, R.: A plane-sweep approach to true multi-image matching. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR), San Francisco, pp. 358–363 (1996)
7. Curless, B., Levoy, M.: A volumetric method for building complex models from range images. In: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH, New Orleans (1996)
8. Faugeras, O., Luong, Q.T.: The Geometry of Multiple Images. MIT, Cambridge (2001)
9. Ferradans, S., Bertalmio, M., Caselles, V.: Geometry-based demosaicking. IEEE Trans. Imgage Process. **18**(3), 665–670 (2009)
10. Gong, M.: Real-time joint disparity and disparity flow estimation on programmable graphics hardware. Comput. Vis. Image Underst. **113**(1), 90–100 (2009)
11. Gunturk, B.K., Glotzbach, J., Altunbasak, Y., Schafer, R.W., Mersereau, R.M.: Demosaicking: color filter array interpolation. IEEE Signal Proc. Mag. **22**(1), 44–54 (2005)
12. Gunturk, B.K., Li, X. (eds.): Image Restoration: Fundamentals and Advances. Digital Imaging and Computer Vision. CRC Press, Boca Raton (2012)
13. Hamilton, J.F., Adams, J.E.: Adaptive color plane interpolation in single sensor color electronic camera. U.S. Patent No. 5,629,734 (1997)
14. Hartley, R.I., Zisserman, A.: Multiple View Geometry in Computer Vision, 2nd edn. Cambridge University Press, Cambridge (2004)
15. Horn, B.K.P., Schunck, B.G.: Determining optical flow. Artif. Intell. **17**, 185–203 (1981)
16. Kolmogorov, V., Zabih, R.: What energy functions can be minimized via graph cuts? IEEE Trans. Pattern Anal. Mach. Intell. **26**(2), 147–159 (2004)
17. Laroche, C.A., Prescott, M.A.: Apparatus and method for adaptively interpolating a full color image utilizing chrominance gradients. U.S. Patent No. 5,373,322 (1994)
18. Menon, D., Calvagno, G.: Color image demosaicking: an overview. Signal Process. Image Commun. **26**(8–9), 518–533 (2011)
19. Mumford, D., Shah, J.: Optimal approximations by piecewise smooth functions and associated variational problems. Commun. Pure Appl. Math. **42**, 577–685 (1989)
20. Noma, A., Graciano, A.B., Cesar, R.M. Jr., Consularo, L.A., Bloch, I.: Interactive image segmentation by matching attributed relational graphs. Pattern Recognit. **45**(3), 1159–1179 (2012)
21. Papadakis, N., Baeza, A., Gargallo, P., Caselles, V.: Polyconvexification of the multi-label optical flow problem. In: Proceedings of the International Conference on Image Processing (ICIP), Hong Kong, pp. 765–768 (2010)
22. Pock, T., Schoenemann, T., Graber, G., Bischof, H., Cremers, D.: A convex formulation of continuous multi-label problems. In: Proceedings of the European Conference on Computer Vision (ECCV), Marseille (2008)
23. Rockafellar, R.T.: Monotone operators and the proximal point algorithm. SIAM J. Control Optim. **14**(5), 877–898 (1976)
24. Rudin, L.I., Osher, S., Fatemi, E.: Nonlinear total variation based noise removal algorithms. Physica D **60**, 259–268 (1992)

25. Tikhonov, A.N., Arsenin, V.Y.: Solutions of ill-posed problems. Winston & Sons, Washington (1977)
26. Vedula, S., Baker, S., Rander, P., Collins, R., Kanade, T.: Three-dimensional scene flow. IEEE Trans. Pattern Anal. Mach. Intell. **27**(3), 475–480 (2005)
27. Zach, C., Pock, T., Bischof, H.: A duality based approach for realtime TV-L1 optical flow. Pattern Recognit. (Proc. DAGM 2007) LNCS **4713**, 214–223 (2007)
28. Zhu, M., Chan, T.: An efficient primal-dual hybrid gradient algorithm for total variation image restoration. Tech. Rep. 08–34, UCLA (2008)

# Modeling and Optimization Techniques with Applications in Food Processes, Bio-processes and Bio-systems

**Eva Balsa-Canto, Antonio A. Alonso, Ana Arias-Méndez, Miriam R. García, A. López-Núñez, Maruxa Mosquera-Fernández, C. Vázquez, and Carlos Vilas**

**Abstract** Food processes, bio-processes and bio-systems are coupled systems that may involve heat, mass and momentum transfer together with kinetic processes. This work illustrates, with a number of examples, how model-based techniques—i.e. simulation, optimization and control—offer the possibility to improve our knowledge about the system at hand and facilitate process design and optimisation even in real time. The contribution is mainly based on the authors experience and illustrates concepts with several examples such as biofilm formation, gluconic acid production, deep-fat frying of potato chips and the thermal processing of packaged foods.

**Keywords** Model identification • Reduced order modelling • Real-time optimization • Food processing • Bio-processes • Bio-systems

## 1 Introduction

Food processes are devoted to transform raw ingredients into food or to transform food products into other forms. Bio-processes are those which make use of living organisms to make useful products. Production may be carried out by using yeasts or bacteria or by using enzymes from organisms. Bio-systems are defined as living organisms or systems of living organisms that can interact with others.

E. Balsa-Canto (✉) • A.A. Alonso • A. Arias-Méndez • M.R. García • M. Mosquera-Fernández • C. Vilas
(Bio)Process Engineering Group, IIM-CSIC, Vigo, Spain
e-mail: ebalsa@iim.csic.es; antonio@iim.csic.es; anaarias@iim.csic.es; miriamr@iim.csic.es; maruxamosquera@iim.csic.es; carlosvf@iim.csic.es

A. López-Núñez • C. Vázquez
Department of Mathematics, University A Coruña, A Coruña, Spain
e-mail: alopeznu@gmail.com; carlosv@udc.es



I. Higueras et al. (eds.), *Numerical Simulation in Physics and Engineering*, SEMA SIMAI Springer Series 9, DOI 10.1007/978-3-319-32146-2_4

The drivers of innovation in the food processing sector can be divided into six major axes, corresponding to general consumer expectations: safety, pleasure, health, physical, convenience and ethics. These consumer demands pose serious challenges to the industry that must comply with a continuously changing market in due time to maintain competitiveness. In the context of biotechnology, the challenge is to define robust bio-processes to produce large quantities of high quality bio-based products in a sustainable and economical way.

Computer-aided simulation and model-based optimization offer a powerful, rational and systematic way to achieve those goals, enabling the possibility to (i) test "what-if" scenarios in a quick and inexpensive way; (ii) improve the understanding of the process or the system at hand; (iii) compute optimal designs or operation conditions given certain objectives and constraints and (iv) control the process operation so as to respond to possible uncertainties and disturbances.

Mathematical models can be roughly classified into three types: white-box models, based on the conservation principles; black-box models, based on data (for example, surface responses or artificial neural networks) and gray-box models which combine first principles with empirical descriptions. In addition models can be classified attending to their mathematical characteristics in linear or non-linear; static or dynamic; lumped or distributed; continuous or discrete; deterministic or stochastic; structured or unstructured.

In recent decades there has been a growing interest in the development of rigorous, mostly hybrid models, to describe food and bio-processes as well as biological systems. Each type of process or system has its own peculiarities. In this work we have selected a set of examples representative of bio-systems (biofilm formation), bio-processes (gluconic-acid production) and processes of the food industry (deep-fat frying of potato chips and thermal processing of packaged foods). The physical, chemical and biological underlying mechanisms are different. However the corresponding mathematical formulations share several properties: they are dynamic, non-linear, continuous, deterministic and unstructured models, and typically distributed.

Despite all efforts on developing rigorous models and the necessary numerical simulation techniques, model validation is still a challenge and it is considered as critical to develop confidence on models use in the food and biotechnological industries. In this scenario it is necessary to develop protocols and to standardise data acquisition so as to obtain transport properties for different food materials, new products and packages, as well as kinetic constants related to microbial and biochemical processes.

In this respect we will describe in some detail how models can be reconciled with experimental data by means of parameter estimation, identifiability analyses and optimal experimental design.

The parameter estimation problem is devoted to find the model parameter values that minimise the distance between model predictions and the experimental data. The identifiability analysis is aimed at evaluating the quality of the model fit and the confidence on the parameter values whereas the optimal experimental design problem is devoted to improve model predictive capabilities.

Once a suitable model with an accurate value for model parameters becomes available, it is possible to formulate optimisation problems to find those operating conditions that achieve a given objective (maximise product quality, minimise energy consumption, etc.) subject to constraints (maximum and minimum processing temperatures, food safety, etc.). We will describe how those problems are mathematically formulated and numerically solved, with special emphasis in the control vector parametrisation approach and the use of reduced order modelling techniques so as to improve computational efficiency.

From the numerical point of view, we will realise that many of the problems of interest: parameter estimation, optimal experimental design, operation design and real time optimisation are formulated as constrained non-linear programming problems including dynamic constraints (the model). Therefore we will describe the type of numerical methods we may use for solving the models, including discretisation based techniques and model reduction techniques, and the most suitable non-linear programming methods with special emphasis in global optimisers.

## 2 Modelling

Mathematical modelling is the art of quantitatively describing from observations certain aspects of the structure and function of a particular process. Model building is an iterative process which starts from the definition of the purpose of the model, that is, the questions to be addressed with the model. In the next step, using the a priori available knowledge and preliminary experimental data, a modelling framework is chosen and a first mathematical model structure is proposed. This first model usually contains unknown non-measurable parameters that may be estimated by means of experimental data fitting. In this regard, we need to know whether it is possible to uniquely determine their values (identifiability analysis) and if so, to estimate them with maximum precision and accuracy (parameter estimation step). This leads to a first working model that must be (in)validated with new experiments, revealing in most cases a number of deficiencies. In this case, a new model structure and/or a new (optimal) experimental design must be planned, and the process is repeated iteratively until the validation step is considered satisfactory.

Most of the models related to food and bio-processes and bio-systems are non-linear dynamic models, typically stated as (ordinary and partial) differential equations (ODEs and PDEs), as follows:

$$\Psi(\mathbf{x}, \mathbf{x}_{\xi}, \mathbf{x}_{\xi\xi}, \mathbf{x}_t, \mathbf{v}_t, \mathbf{v}, \mathbf{u}, \boldsymbol{\theta}, t) = 0 \tag{1}$$

$$\mathbf{x}(\boldsymbol{\xi}, t_0) = \Psi_0(\mathbf{x}(\boldsymbol{\xi}, t_0), \mathbf{u}(t_0), \boldsymbol{\theta}, t_0); \quad \mathbf{v}(t_0) = \Phi_0(\boldsymbol{\theta}, t_0); \tag{2}$$

$$\mathcal{B}(\mathbf{x}, \mathbf{x}_{\xi}, \mathbf{v}, \mathbf{u}, \boldsymbol{\theta}, \boldsymbol{\xi}, t) = 0 \tag{3}$$

where $\boldsymbol{\xi} \in \Omega \subset \mathbb{R}^3$ are the spatial variables, $\mathbf{x}(\boldsymbol{\xi}, t) \in \mathbb{Z} \subset \mathbb{R}^{\nu}$ are the distributed state variables (temperature, water content, microorganisms concentration, etc.),

$\mathbf{x}_{\xi} = \partial \mathbf{x}/\partial \boldsymbol{\xi}$, $\mathbf{x}_{\xi\xi} = \partial^2 \mathbf{x}/\partial \boldsymbol{\xi}^2$, $\mathbf{x}_t = \partial \mathbf{x}/\partial t$, $\mathbf{v} \subset \mathbb{R}^{\mu}$ are the lumped variables, $\mathbf{v}_t = d\mathbf{v}/dt$, $\mathbf{u} \in U \subset \mathbb{R}^{\sigma}$ are the control variables (processing temperature, feeding substrate, valves openings, etc.) and $\boldsymbol{\theta} \in \Theta \subset \mathbb{R}^{\eta}$, time independent parameters (thermo-physical properties, kinetic related constants, etc.). Equations (2) and (3) represent the initial and boundary conditions, respectively.

## 2.1 Parameter Estimation

Given a general set of differential equations explaining the dynamics of a system, Eqs. (1), (2), and (3), the values assigned to the parameters $\boldsymbol{\theta}$ will give rise to different system behaviours. The problem of parameter estimation may be formulated as follows: Find model unknown parameters (e.g. thermo-physical properties, kinetic coefficients, initial conditions, etc.) so as to minimise a measure of the distance among the model predictions and the available experimental data as obtained under a particular *experimental scheme* (illustrated in Fig. 1) [36].

Let us suppose the most general *experimental scheme* where several experiments $\mathcal{E} = 1, \ldots, n_{\mathcal{E}}$ and types of outputs $k = 1, \ldots, n_y^{\mathcal{E}}$ are used for the estimation (for instance, two experiments with different inputs where some concentration and temperature are measured). Due to the discrete nature of these outputs they are located at a given number ($n_t^{\mathcal{E},k}$) of certain sampling times $t_s$ and a number ($n_{\mathcal{S}}^{\mathcal{E},k,s}$) of sensor positions $\boldsymbol{\xi}_p$ for each experiment. Their associated model predictions must be obtained by means of the implementation of the above experiments and evaluating the results at the same sampling times and sensor positions. Considering the general

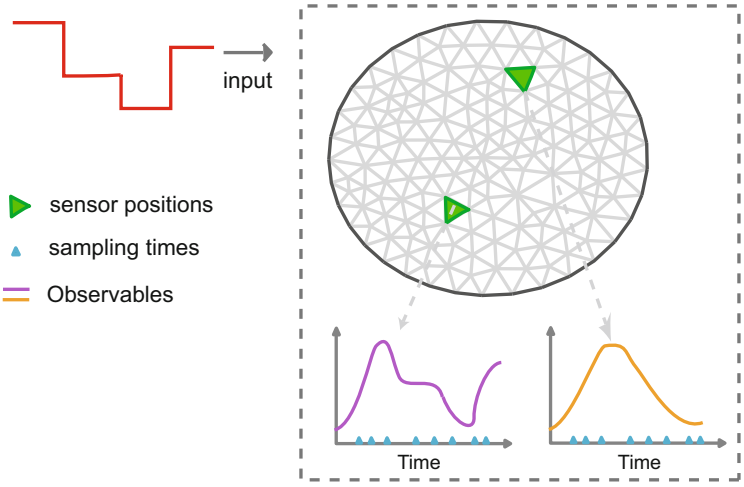


**Fig. 1** Illustrative representation of the *experimental scheme*

non-linear model described in (1), (2), and (3), these predictions are calculated from:

$$\mathbf{v}^{\mathcal{E}}(\boldsymbol{\xi}_p, t_s, \boldsymbol{\theta}) = \mathbf{f}_y^{\mathcal{E}}\left(\mathbf{x}^l(t_s), \mathbf{x}^d(t_s, \boldsymbol{\xi}_p), \boldsymbol{\theta}\right) \tag{4}$$

where in $\mathbf{f}_y^{\mathcal{E}} \in \mathbb{R}^{n_y}$ the implicit influence of the inputs is not made explicit to simplify notation.

For the sake of clarity the measurements and model predictions will be encoded in the following vectors:

$$\boldsymbol{\mathcal{Y}}_{\boldsymbol{m}} = [y_{m1}, \ldots, y_{m\boldsymbol{\ell}}, \ldots, y_{m\boldsymbol{n_\ell}}]^{\mathrm{T}} \in \mathbb{R}^{\boldsymbol{n_\ell}} \tag{5}$$

and

$$\boldsymbol{\mathcal{Y}}(\boldsymbol{\theta}) = [y_1(\boldsymbol{\theta}), \ldots, y_{\boldsymbol{\ell}}(\boldsymbol{\theta}), \ldots, y_{\boldsymbol{n_\ell}}(\boldsymbol{\theta})]^{\mathrm{T}} \in \mathbb{R}^{\boldsymbol{n_\ell}}, \tag{6}$$

where $\boldsymbol{\ell}$ represents a certain data defined by the sub-indexes $p, s, k, \mathcal{E}$ and $\boldsymbol{n_\ell}$ is the total number of such data.

At the time of defining a measure of the distance between the experimental and predicted data, several possibilities exist. Here the maximum likelihood approach is considered. The idea is to find the vector of parameters that gives the highest likelihood to the measured data. Under the assumptions of independent measurements with Gaussian noise, the distance to be minimised becomes:

$$J_{ml} = \sum_{\boldsymbol{\ell}}^{\boldsymbol{n_\ell}} \left(-\frac{1}{2}\right) \left[\log(2\pi) + \log(\sigma_{\boldsymbol{\ell}}^2) + \frac{(y_{m\boldsymbol{\ell}} - y_{\boldsymbol{\ell}}(\boldsymbol{\theta}))^2}{\sigma_{\boldsymbol{\ell}}^2}\right] \tag{7}$$

where

$$\sum_{\boldsymbol{\ell}=1}^{\boldsymbol{n_\ell}} (\cdot) = \sum_{\mathcal{E}=1}^{n_{\mathcal{E}}} \left( \sum_{k=1}^{n_y^{\mathcal{E}}} \left( \sum_{s=1}^{n_t^{\mathcal{E},k}} \left( \sum_{p=1}^{n_S^{\mathcal{E},k,s}} (\cdot) \right) \right) \right).$$

The parameter estimation problem is thus formulated as a nonlinear optimization problem subject to the system dynamics (Eqs. (1), (2), and (3)) and possibly bounds on the parameter values. Therefore, its numerical solution involves an outer iterative procedure to generate values for the unknown parameters and initial conditions, the nonlinear programming method (NLP), and an iterative procedure to solve the differential equations, the boundary value problem (BVP) solver, as shown in Fig. 2.

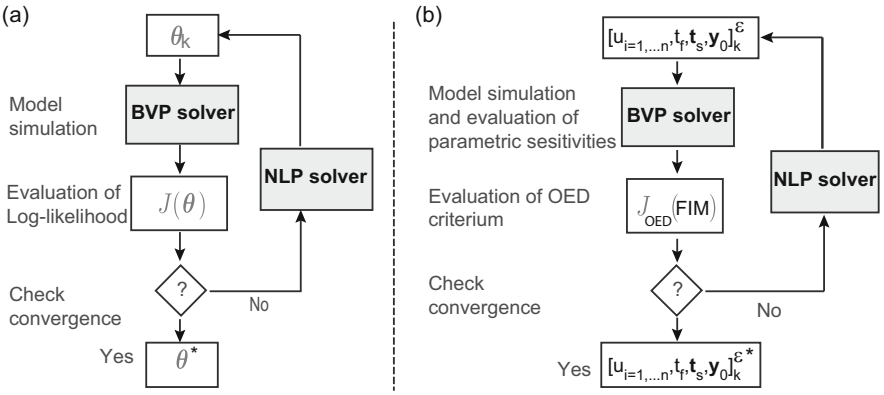


**Fig. 2** Numerical solution of: (**a**) parameter estimation and (**b**) optimal experimental design

## *2.2 Identifiability Analysis*

Identifiability has to do with the possibility of finding a unique solution for the model parameters. At this point, it is important to note that in fact, in the presence of experimental error, there are several equivalent solutions defining the parameter uncertainty region. The shape and the size of such region will determine whether practical identifiably is or not guaranteed. Assuming that the uncertainty region corresponds to a hyper-ellipsoid (typical case), highly elongated hyperellipsoids tend to be associated with poor or lack of identifiability of some parameters.

In order to asses the uncertainty regions, several possibilities exist. Monte-Carlo based approaches allow to compute robust uncertainty regions [9]. However, the associated computational cost makes it difficult to use these methods for large scale models. Alternatively, the confidence interval of $\boldsymbol{\theta}_i^*$ may be obtained through the covariance matrix

$$\pm\, t_{\alpha/2}^{\gamma} \sqrt{\mathbf{C}_{ii}} \tag{8}$$

where $t_{\alpha/2}^{\gamma}$ is given by Students t-distribution, $\gamma = N_d - \eta$ degrees of freedom and $(1-\alpha)100\,\%$ is the confidence interval selected, typically $95\,\%$.

For non-linear models, there is no exact way to obtain the covariance matrix $\mathbf{C}$. Therefore, the use of approximations has been suggested. Possibly the most widely used is based on the Crammèr-Rao inequality which establishes, under certain assumptions on the number of data and non-linear character of the model, that the covariance matrix may be approximated by the inverse of the Fisher information

matrix (FIM) which is formulated as follows [24, 36]:

$$\mathcal{F} = \underset{\mathbf{v}_m|\boldsymbol{\theta}*}{\mathrm{E}} \left\{ \left[ \frac{\partial J_{ml}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \right] \left[ \frac{\partial J_{ml}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \right]^T \right\} \tag{9}$$

where E regards the expected value.

### 2.3 Optimal Experimental Design

In order to improve the quality of parameter estimates it is possible to use the model to define new experiments. The idea is to formulate an optimisation problem where the objective is to find the experimental scheme (number of experiments, input conditions, number and location of sampling times and sensors, duration of the experiments) which result in maximum information content as measured by, for example, the FIM, subject to the system dynamics Eqs. (1), (2), and (3) plus experimental constraints. The problem can be solved by a combination of the control vector parametrisation (CVP) method and a suitable optimiser enabling the simultaneous design of several dynamic experiments with optimal sampling times [8] and optimal sensor locations [20].

The optimal experimental design problem is thus formulated as a nonlinear optimisation whose numerical solution involves an outer iterative procedure to generate values for the experimental conditions, the nonlinear programming method, and the boundary value problem solver to handle model simulation and the computation of the parametric sensitivities needed to evaluate the FIM, as shown in Fig. 2.

It should be remarked that the recently developed software tool AMIGO (Advanced Model Identification using Global optimisation)[4] covers model simulation, parameter estimation, identifiability analysis and optimal experimental design. Thus facilitating the implementation of the model identification loop for general non-linear dynamic models.

## 3 Optimization of the Operation

### 3.1 Problem Formulation

The optimization of the operation is formulated as a general **dynamic optimization** (DO) problem as follows: Find the controls $\mathbf{u}(t)$ that minimise (or maximise) the objective functional

$$J = \phi\left(\mathbf{x}(\boldsymbol{\xi}, t_f), \mathbf{v}(t_f), \boldsymbol{\theta}, t_f\right) + \int_{t_0}^{t_f} L\left(\mathbf{x}(\boldsymbol{\xi}, t), \mathbf{v}(t), \mathbf{u}(t), \boldsymbol{\theta}, \boldsymbol{\xi}, t\right)\, \mathrm{d}t, \tag{10}$$

where the scalar functions $\phi$ (Mayer term) and $L$ (Lagrangian term) are continuously differentiable with respect to all of their arguments, and the final time $t_f$ can be either fixed or free, subject to the following constraints:

- The system dynamics Eqs. (1), (2), and (3).
- Algebraic constraints on the state and control variables which force the fulfilment of particular operational or biological conditions (for example, microbiological lethality, maximum and minimum processing temperatures, etc.) at particular time points or throughout the process:

$$\mathbf{r}_k^{eq}(\mathbf{x}(\boldsymbol{\xi}, t_k), \mathbf{v}(t_k), \mathbf{u}(t_k), \boldsymbol{\theta}, t_k) = 0; \qquad \mathbf{r}_k^{in}(\mathbf{x}(\boldsymbol{\xi}, t_k), \mathbf{v}(t_k), \mathbf{u}(t_k), \boldsymbol{\theta}, t_k) \leq 0; \tag{11}$$

$$\mathbf{c}^{eq}(\mathbf{x}(\boldsymbol{\xi}, t), \mathbf{v}(t), \mathbf{u}(t), \boldsymbol{\theta}, t) = 0; \qquad \mathbf{c}^{in}(\mathbf{x}(\boldsymbol{\xi}, t), \mathbf{v}(t), \mathbf{u}(t), \boldsymbol{\theta}, t) \leq 0. \tag{12}$$

- Bounds on the control variables:

$$\mathbf{u}^L \leq \mathbf{u}(t) \leq \mathbf{u}^U. \tag{13}$$

### 3.2 Control Vector Parametrisation

There are several alternatives for the solution of dynamic optimization problems from which the direct methods are the most widely used. These methods transform the original problem into a non-linear programming problem by means of the complete parametrisation [12], the multiple shooting [13] or the control vector parametrisation (CVP) [35]. Basically, all of them are based on the use of some type of discretisation and approximation of either the control variables or both the control and state variables. The three alternatives basically differ in: the resulting number of decision variables, the presence or absence of parametrisation related constraints and the necessity of using an initial value problem solver.

While the complete parametrisation or the multiple shooting approaches may become prohibitively expensive in computational terms, the CVP approach allows handling large-scale DO problems, such as those related to PDE systems, without solving very large NLPs and without dealing with extra junction constraints.

The CVP method proceeds dividing the duration of the process into a number $\rho$ of control intervals and the control function is approximated using a low order polynomial form over each interval. Each control variable approximation may be expressed using Lagrange polynomials as follows:

$$u_j(t) = \sum_{i=1}^{M_j} u_{ij} \Phi_i^{(M_j)}(\tau), \tag{14}$$

where, $j = 1, \ldots, \rho$, $t \in [t_0, t_f]$, and $\tau$ is normalized time given by,

$$\tau = \frac{t - t_0}{t_f - t_0} \tag{15}$$

and the Lagrange polynomials of order $M$, $\Phi_i^{(M_j)}$ are defined in the standard form:

- If $M = 1$,

$$\Phi_i^{(M)}(\tau) \equiv 1. \tag{16}$$

- If $M \geq 2$,

$$\Phi_i^{(M)}(\tau) \equiv \prod_{i'=1, i \neq 1}^{M} \frac{\tau - \tau_{i'}}{\tau_i - \tau_{i'}}. \tag{17}$$

The parameters of these polynomials, $u_{ij}$, will be used as decision variables in the optimization process together with time independent parameters. Again the numerical solution of the associated NLP requires an inner iteration to solve the system dynamics, similarly to what is shown in Fig. 2.

## 4 Numerical Methods

### *4.1 Model Simulation*

As explained before, most of the food, bio-processes and bio-systems models exhibit a nonlinear dynamic behavior which makes the analytical solution of models representing such systems rather complicated, if not impossible, for most of the realistic situations. In addition to non-linearity, these processes may present a spatially distributed nature. As a consequence they must be described using PDEs which, in turns, makes the analytical approach even more difficult. Numerical techniques must be, therefore, employed to solve the model equations.

Most of numerical methods employed for solving PDEs, in particular those employed in this work, belong to the family of *methods of weighted residuals* in which the solution of the distributed variables in the system (1), (2), and (3) is approximated by a truncated Fourier series of the form[1] [23]:

$$x(\boldsymbol{\xi}, t) \approx \sum_{i=1}^{N} m_i(t)\, \phi_i(\boldsymbol{\xi}). \tag{18}$$

[1]For the sake of clarity and without loss of generality, the vector field $\mathbf{x}(\boldsymbol{\xi}, t)$ in Eqs. (1), (2), and (3) will be considered as a scalar $x(\boldsymbol{\xi}, t)$.

Depending on the selection of the *basis functions* $\phi_i(\boldsymbol{\xi})$ different methodologies arise. Here two groups will be considered: those using locally defined basis functions as it is the case in classical techniques like the numerical method of lines (NMOL) or the finite element method (FEM) and those using globally defined basis functions.

### 4.1.1 Methods Using Local Basis Functions

The underlying idea is to discretise the domain of interest into a (usually large) number $N$ of smaller sub-domains. In these sub-domains local basis functions, for instance low order polynomials, are defined and the original PDE is approximated by $N$ ordinary differential equations (ODEs). The shape of the elements and the type of local functions allow distinguishing among different alternatives.

Probably the most widely used approaches for this transformation are the NMOL and the FEM. The reader interested on an extensive description of these techniques is referred to the literature [23, 29, 34].

However it must be highlighted that in many food, bio-processes and biological models, especially those in 2D and 3D, the number of discretisation points ($N$) to obtain a good solution might be too large for their application in parameter estimation, experimental design or process optimization.

Methods using global basis functions, which will reduce the computational effort, constitute an efficient alternative [6].

### 4.1.2 Methods Using Global Basis Functions

The use of eigenfunctions obtained from the Laplacian operator, Chevyshev or Legendre polynomials, etc. have been considered over the last decades—see [22] as a means to obtain reduced order descriptions of PDE systems. Probably the most efficient order reduction technique is the one based on the *proper orthogonal decomposition* (POD) approach [31]. In this approach each element $\phi_i(\xi)$ of the set of basis functions in (18) is computed off-line as the solution of the following integral eigenvalue problem [31]:

$$\int_{\mathbb{V}} R(\boldsymbol{\xi}, \boldsymbol{\xi}')\, \phi_i(\boldsymbol{\xi}')\, \mathrm{d}\boldsymbol{\xi}' = \lambda_i\, \phi_i(\boldsymbol{\xi}), \tag{19}$$

where $\lambda_i$ corresponds with the eigenvalue associated with each global eigenfunction $\phi_i$. The kernel $R(\boldsymbol{\xi}, \boldsymbol{\xi}')$ in Eq. (19) corresponds with the two point spatial correlation function, defined as follows:

$$R(\boldsymbol{\xi}, \boldsymbol{\xi}') = \frac{1}{\ell} \sum_{j=1}^{\ell} x(\boldsymbol{\xi}, t_j) x(\boldsymbol{\xi}', t_j), \tag{20}$$

with $x(\boldsymbol{\xi}, t_j)$ denoting the value of the field at each instant $t_j$ (snapshot) and the summation extends over a sufficiently rich collection of uncorrelated snapshots at $j = 1, \ldots, \ell$ [31]. The basis functions obtained by means of the POD technique are also known as empirical basis functions or POD basis. The basis functions are orthogonal and can be normalised so that:

$$\int_{\mathbb{V}} \phi_i \, \phi_j \, \mathrm{d}\boldsymbol{\xi} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$

The dissipative nature of food and bio-processes makes that the eigenvalues obtained from Eq. (19) can be ordered so that $\lambda_i \leq \lambda_j$ for $i < j$, furthermore $\lambda_n \to \infty$ as $n \to \infty$. This property allows to define a finite (usually low) dimensional subset $\phi_A = [\phi_1, \phi_2, \ldots, \phi_N]$ which captures the relevant features of the system [1, 5, 6].

In order to compute the time dependent coefficients in Eq. (18), the original PDE system (1), (2), and (3) is projected onto each element of the POD basis set. Such projection is carried out by multiplying the original PDE by each $\phi_i$ and integrating the result over the spatial domain. As a result the following set of ODEs is obtained:

$$\mathbf{m}_{A\,t} = F(\mathbf{m}_A, \mathbf{x}, \mathbf{v}, \mathbf{u}, \boldsymbol{\theta}, t). \tag{21}$$

At this point the basis functions $\phi_A$ and time dependent coefficients

$$m_A = [m_1, m_2, \ldots, m_N]$$

are known, therefore the original field $x$ can be recovered by applying Eq. (18), this is $x = \phi_{\mathrm{A}} m_A$. The number of elements $N$ in the basis subset $\phi_{\mathrm{A}}$ can be increased to approximate the original state $x$ with an arbitrary degree of accuracy.

## *4.2 Non-linear Programming Methods*

Most of the problems of interest are formulated as non-linear optimisation problems which can be handled by adequate non-linear programming methods. Nonlinear programming methods may be largely classified in two main groups: local and global. Local methods are designed to generate a sequence of solutions, using some type of pattern search or gradient and Hessian information, that will converge to a local optimum, usually the closest to the provided initial guess. However the NLPs with non-linear dynamic constraints (such as in parameter estimation or the ones resulting from the application of the CVP approach) are frequently multimodal (i.e. presenting multiple local optima). Therefore, local methods may converge to local solutions, especially if they are started far away from the global optimum. In order to surmount these difficulties, global methods must be used.

Global methods have emerged as the alternative to search the global optimum [27]. The successful methodologies combine effective mechanisms of exploration of the search space and exploitation of the previous knowledge obtained by the search. Depending on how the search is performed and the information is exploited the alternatives may be classified in three major groups: deterministic, stochastic and hybrid.

Global deterministic methods [18, 28] in general take advantage of the problem's structure and guarantee global convergence for some particular problems that verify specific smoothness and differentiability conditions. Although they are very promising and powerful, there are still limitations to their application, particularly for non-linear dynamic systems, since the computational cost increases rapidly with the size of the considered dynamic system and the number of decision variables.

Global stochastic methods do not require any assumptions about the problem's structure. They make use of pseudo-random sequences to determine search directions toward the global optimum. This leads to an increasing probability of finding the global optimum during the run time of the algorithm, although convergence may not be guaranteed. The main advantage of these methods is that, in practise, they rapidly arrive to the proximity of the solution.

The most successful approaches lie in one (or more) of the following groups: pure random search and adaptive sequential methods, clustering methods or metaheuristics. Metaheuristics are a special class of stochastic methods which have proved to be very efficient in recent years. They include both population (e.g., genetic algorithms) or trajectory-based (e.g., simulated annealing) methods. They can be defined as guided heuristics and many of them try to imitate the behaviour of natural or social processes that seek for any kind of optimality [33].

Despite the fact that many stochastic methods can locate the vicinity of global solutions very rapidly, the computational cost associated to the refinement of the solution is usually very large. In order to surmount this difficulty, hybrid methods and metaheuristics have been recently presented for the solution of dynamic optimisation problems [7, 16] or parameter estimation problems [30]. They speed up these methodologies while retaining their robustness and, provided a gradient based local method is used, they guarantee convergence to a gradient zero solution.

The recently developed Scatter Search based methods [15, 17] have proved to be successful in the solution of parameter estimation and dynamic optimisation problems allowing to overcome typical difficulties of nonlinear dynamic systems optimisation such as noise, flat areas, non-smoothness, and/or discontinuities.

# 5 Illustrative Examples

## *5.1 Modelling and Simulation: Growth of Bacterial Biofilms*

Bacteria, both in natural and pathogenic ecosystems, are found mainly within surface associated cell assemblages, the so called biofilms. It is now recognised that biofilms constitute a source for food related infections. Since they can render their inhabitants more resistant to disinfectants, biofilms have become problematic in a wide range of food industries, including brewing, seafood processing, dairy processing, poultry processing and meat processing [32].

Since resistance is associated to biofilm structure there is a growing interest in the characterisation of pathogenic biofilm structures. In this respect much research has been performed to gain deeper understanding of biofilms formation, adherence and growth. Basically two approaches have been considered: the use of imaging techniques and modelling.

While image quantitative analysis allows direct quantification from images obtained by microscopic techniques [11, 25, 39], mathematical models have been developed to provide mechanistic insight into structure evolution. Proposed models can be divided in three general classes according to the way the biomass is represented: discrete, continuous and hybrid discrete-continuous models (see Wanner et al. [37] for an extensive review).

Here we will consider the continuous model proposed by Eberl et al. [14]. The model represents bacteria and nutrients with two density fields denoted by $m(t, \boldsymbol{\xi})$ and $c(t, \boldsymbol{\xi})$, respectively. Their spatial distributions are represented by the following set of coupled diffusion-reaction mass balance equations:

$$\frac{\partial C}{\partial t} = d_1 \nabla^2 C - F(C, M), \tag{22}$$

$$\frac{\partial M}{\partial t} = \nabla \cdot (d_2(M) \nabla M) + G(C, M), \tag{23}$$

with

$$F(C,M) = K_1 \frac{MC}{K_2 + C},\ G(C,M) = K_3 \frac{CM}{K_2 + C} - K_4 M,\ d_2(M) = m_{max}^{b-a} \left( \frac{\epsilon}{1-M} \right)^a M^b,$$

where

$$k_1 = m_{max} \left( \frac{\mu_m}{Y_{XS}} + m_s \right), \quad k_2 = K_s, \quad k_3 = Y_{XS}/m_{max}, \quad k_4 = m_s m_{max},$$

$$K_1 = m_{max} \frac{k_1}{c_0}, \quad K_2 = \frac{k_2}{c_0}, \quad K_3 = k_3 k_1, \quad K_4 = k_3 k_4,$$

**Table 1** Biofilm growth model parameters

| Parameter | Description | Value |
|---|---|---|
| $d_1$ | Nutrient diffusion coefficient ( $m^2/s$) | $1.6 \times 10^{-9}$ |
| $\mu_m$ | Maximum bacterial growth rate (1/s) | $1.5 \times 10^{-5}$ |
| $Y_{XS}$ | Substrate growth yield factor (*adim.*) | 0.045 |
| $K_s$ | Monod saturation constant ( $kg/m^3$) | $3.5 \times 10^{-5}$ |
| $m_s$ | Maintenance coefficient (1/ s) | $3 \times 10^{-5}$ |
| $m_{max}$ | Maximum biomass ( $kg/m^3$) | 60 |
| $c_0$ | Initial nutrients concentration ( $kg/m^3$) | $4 \times 10^{-3}$ |
| $M_0$ | Initial biomass concentration (*adim.*) | 0.9 |
| $\epsilon$, $a$, $b$ | Parameters to control biomass diffusion (*adim.*) | $5 \times 10^{-5}$, 4, 4 |

and $M$ and $C$ dimensionless variables ($M := m/m_{max}$; $C := c/c_0$). Model parameters are summarised in Table 1.

In the equation describing biomass Eq. (23), the first term in the right-hand side accounts for the diffusion of the biomass and the second term for the production of biomass. Expansion of bacteria depends on the local density of bacteria and takes place only if the biomass density approaches a prescribed maximum value established by $m_{max}$. Elberl et al. proposed a density-dependent expression for the diffusion factor $d_2$ that satisfies this condition. The physical interpretation is that the biomass diffusivity vanishes as $m$ becomes small but increases as $m$ grows due to biochemical reaction.

In our work we developed a numerical approach based on the combination of finite differences schemes in space—with centred differences for the nutrients and a backwards-forward space for the biomass- and the Crank-Nicolson approach in time [10]. The resulting set of non-linear equations is solved using a Newton-Raphson algorithm. Here we illustrate results achieved for one-dimensional growth with merging colonies under symmetric initial and boundary conditions. For this purpose, two equally sized colonies are located in the interval $[0, L]$:

$$C(0, x) = 1, \qquad \forall x \in [0, 1], \tag{24}$$

$$M(0, x) = \begin{cases} M_0, & \text{for } x \in [x_1, x_2] \cup [x_3, x_4], \\ 0, & \text{elsewhere}, \end{cases} \tag{25}$$

with $L = 10^{-4}$, $x_1 = L - x_4$, $x_2 = L - x_3$, $x_3 = L/2 + 3 \times 1.6 \times 10^{-6}$ and $x_4 = L/2 + 4 \times 1.6 \times 10^{-6}$. And symmetric boundary conditions are imposed:

$$C(t, 0) = 1, \qquad C(t, L) = 1 \qquad \forall t \in [0, 1] \tag{26}$$

$$M_x(t, 0) = 0, \qquad M_x(t, L) = 0 \qquad \forall t \in [0, 1]. \tag{27}$$

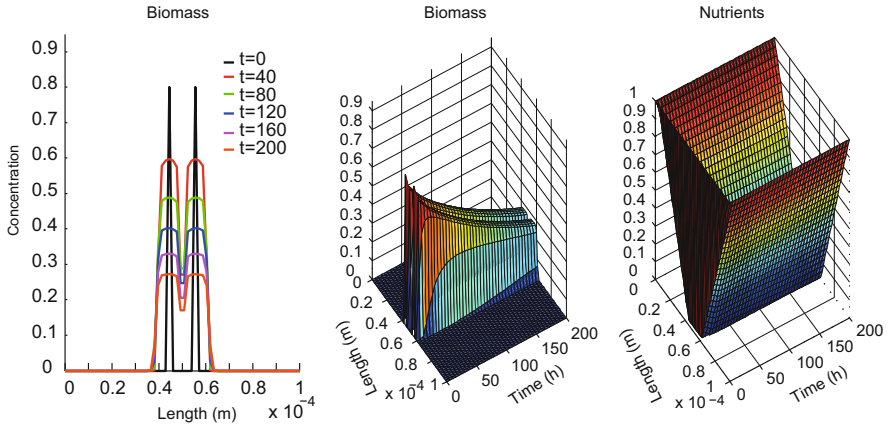


**Fig. 3** Numerical solution of the biofilm growth example

As it can be seen in the Fig. 3, both colonies spread in both directions till collision is produced. It should be noted that a modification on model parameters, for example, an increased nutrient availability or decreased maximum biomass concentration, would accelerate the spatial spreading of biomass and, in consequence, colonies would merge earlier and form a more compact spatial structure. On the contrary, a decrease in nutrient availability or a larger maximum biomass concentration, would slow down the spatial spreading of biomass and colonies would merge later or even would not merge. This reveals the capacity of the model to describe the clusters and tunnels typically observed in the laboratory.

## *5.2 Reduced Order Models: Food Pasteurisation in Tunnels*

Food thermal processing persists as one of the most widely used methods for food preservation. The product is treated at a given temperature for a given period of time to minimise public health hazards due to the presence of pathogenic microorganisms and to extend product shelf-life. Different time-temperature combinations could be used to achieve safety. However, the related time-temperature histories would affect the quality of the product in different ways.

Therefore, the design of thermal processes requires a deep understanding of the heating process of the given product, the impact on the target microorganism and quality factors. The thermal treatment will depend on the thermo-physical characteristics, shape and size of the food product and container; the type and thermal resistance of the microorganisms of interest and the kinetics of quality degradation.

In this section we consider the pasteurisation in tunnels of highly viscous liquid foods such as tomato or carrot puree in cylindrical food jars. The containers are

loaded at one end of the pasteuriser and passed under sprinkles of water as they move along the conveyor belt. Temperature of the water changes in the different zones so as to achieve pasteurisation. The heat transfer occurs between the hot water film and the package surface and from the package to the food product.

The evolution of temperature and velocity within the food product during the pasteurisation is described by means of conservation laws. The package is assumed to be homogeneously heated therefore axial symmetry allows to consider a 2D geometry. The process can be mathematically described as follows [26]:

### 5.2.1 Continuity Equation

$$\frac{\partial u}{\partial z} + \frac{v}{r} + \frac{\partial v}{\partial r} = 0, \tag{28}$$

being $r$ and $z$ being the spatial coordinates (radius and height of the package) while $v$ and $u$ are the velocity field components, i.e., $w = [u, v]^T$.

### 5.2.2 Momentum Conservation

$$\rho_{prod}\left(\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial z} + v\frac{\partial v}{\partial r}\right) = -\frac{\partial p}{\partial r} + \mu_{prod}\left(\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial rv}{\partial r}\right) + \frac{\partial^2 v}{\partial z^2}\right), \tag{29}$$

$$\rho_{prod}\left(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial z} + v\frac{\partial u}{\partial r}\right) = -\frac{\partial p}{\partial z} + \mu_{prod}\left(\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u}{\partial r}\right) + \frac{\partial^2 u}{\partial z^2}\right) + \hat{\rho}\, g, \tag{30}$$

where $p$ is the pressure, $\rho_{prod}$ corresponds with the food stuff density, $g$ is the gravity constant, $T$ represents the temperature distribution inside the food, $\mu_{prod}$ stands for the viscosity expressed as a function of the temperature [26]:

$$\mu_{prod} = a_\mu T^2 - b_\mu T + c_\mu, \tag{31}$$

and the density $\hat{\rho}$ is usually expressed in terms of the fluid temperature as follows:

$$\hat{\rho} = \rho_{ref}\left(1 - \beta\left(T - T_{ref}\right)\right), \tag{32}$$

being $\beta$ the thermal dilatation coefficient and $\rho_{ref}$ and $T_{ref}$ given reference values.

### 5.2.3 Energy Conservation

$$\frac{\partial T}{\partial t} + v\frac{\partial T}{\partial r} + u\frac{\partial T}{\partial z} = \alpha_{prod}\left(\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial T}{\partial r}\right) + \frac{\partial^2 T}{\partial z^2}\right). \tag{33}$$

The system in Eqs. (28), (29), (30), (31), (32), and (33) is subject to the following initial and boundary conditions:

- Initially the food stuff is at rest ($v = 0$) and at uniform temperature $T(r, z, t = 0) = T_0$.
- The velocity field components ($u$, $v$) are zero in the package walls, i.e.:

$$u|_{z=0} = u|_{z=Z} = u_{|_{r=R}} = 0, \tag{34}$$

$$v|_{z=0} = v|_{z=Z} = v|_{r=R} = 0. \tag{35}$$

- Symmetry conditions are imposed in the symmetry axis ($r = 0$):

$$\left.\frac{\partial T}{\partial r}\right|_{r=0} = \left.\frac{\partial u}{\partial r}\right|_{r=0} = \left.\frac{\partial v}{\partial r}\right|_{r=0} = 0. \tag{36}$$

- The package bottom is touching the transportation belt assumed to be an insulating material:

$$\left.\frac{\partial T}{\partial z}\right|_{z=0} = 0. \tag{37}$$

- At the right and upper sides, the package is in direct contact with the falling film of heating fluid:

$$k_{prod}\left.\frac{\partial T}{\partial r}\right|_{r=R} = h_{jar}\Big(T_{ff} - T|_{r=R}\Big), \tag{38}$$

$$k_{prod}\left.\frac{\partial T}{\partial z}\right|_{z=Z} = h_{jar}\Big(T_{ff} - T|_{z=Z}\Big), \tag{39}$$

with $T_{ff}$ being the temperature of the falling film, $h_{jar}$ the jar heat transfer coefficient and $k_{prod}$ the product thermal conductivity.

Here we compare the solution of the models Eqs. (28), (29), (30), (31), (32), (33), (34), (35), (36), (37), (38), and (39) using the finite element method (FEM) and the reduced order model (ROM) based on the proper orthogonal decomposition approach. The main steps to derive the ROM are the following:

1. *Obtain a set of snapshots* that characterises the spatio-temporal distribution of the variable of interest (temperature, velocity, etc.). In our case all the snapshots are obtained from a FEM based simulation of system (Eqs. (28), (29), (30), (31), (32), (33), (34), (35), (36), (37), (38), and (39)) under different possible experimental conditions ($T_{ff}$, $T_0$). Since product and package properties are unknown, several values of the parameters within physical meaningful bounds have to be considered to obtain the snapshots. The finite element method, with a mesh of 725 discretisation points, was used to solve the system of Eqs. (28), (29), (30), (31), (32), (33), (34), (35), (36), (37), (38), and (39) and generate snapshots (see Fig. 4). Each simulation implies solving 2900 ODEs which takes around 25 s in a standard PC.
2. *Computation of the POD basis*. The snapshots of the previous point are used to compute the so-called POD basis as described above [19].
3. *Projection of the model equations (28), (29), (30), (31), (32), (33), (34), (35), (36), (37), (38), and (39) over the selected POD basis*. Projection is carried out by multiplying the original PDE system by the POD basis and integrating the result over the spatial domain. Note that the FEM structure may be exploited to

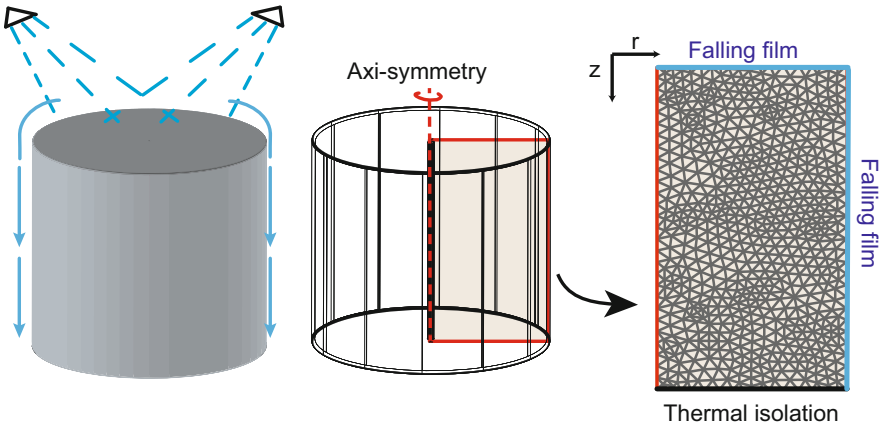


**Fig. 4** Illustrative example of the package, operating conditions and FEM mesh

numerically perform the projection [19]:

$$\int_V \phi_{T,i} \frac{\partial T}{\partial t} \, \mathrm{d}\xi = \int_V \phi_{T,i} \left(\alpha \, \Delta T - w \, \nabla T\right) \mathrm{d}\xi, \tag{40}$$

$$\rho \int_V \phi_{w,i} \frac{\partial w}{\partial t} \, \mathrm{d}\xi = \int_V \phi_{w,i} \left(\mu \Delta w - \rho w \nabla w - \nabla P + \rho g \left(1 - \beta (T - T_0)\right) \mathbf{z}\right) \mathrm{d}\xi, \tag{41}$$

with $i = 1, \ldots, N_x$ and $\mathbf{z}$ being a unitary vector with the direction of the spatial coordinate $z$. Equation (41) with $w = [u, v]^T$ is equivalent to the result of the projections of Eqs. (29) and (30).

Taking into account:

$$T(\xi, t) \approx \sum_{i=1}^{N_T} m_{T_i}(t) \, \phi_{T_i}(\xi), \tag{42}$$

$$w(\xi, t) \approx \sum_{i=1}^{N_w} m_{w_i}(t) \, \phi_{w_i}(\xi), \tag{43}$$

and after some algebraic manipulations, Eqs. (40) and (41) can be rewritten as:

$$\frac{\mathrm{d}\mathbf{m}_T}{\mathrm{d}t} = \left(\alpha_{prod} A_T + B_T + \alpha_{prod} D_T\right) \mathbf{m}_T, \tag{44}$$

$$\rho \frac{\mathrm{d}\mathbf{m}_w}{\mathrm{d}t} = \left(\mu A_w + \rho B_w + \mu D_w\right) \mathbf{m}_w - \rho g \beta C_{T,w} \mathbf{m}_T + \rho g (1 + \beta T_0), \tag{45}$$

where each component of matrices $A_x$, $B_x$, $C_{T,w}$ and $D_x$ are of the form:

$$A_x(i;j) = \int_V \nabla \phi_{x,i} \, \nabla \phi_{x,j} \, \mathrm{d}\xi, \qquad B_x(i;j) = \int_V \phi_{x,i} (w \nabla \phi_{x,j}) \, \mathrm{d}\xi,$$

$$C_{T,w}(i;j) = \int_V \phi_{w,i} \, \phi_{T,j} \, \mathrm{d}\xi, \qquad D_x(i;j) = \int_{\partial V} \phi_{x,i} \, \nabla \phi_{x,j} \, \mathrm{d}\xi,$$

with $\partial V$ denoting the boundary of $V$. The vector of time dependent functions $\mathbf{m}_x$ is of the form $\mathbf{m}_x = [m_{x,1}, m_{x,2}, \ldots, m_{x,N}]^T$.

The larger the number of basis functions used, the better the accuracy of the reduced model. However at the expense of higher computational cost. In order to arrive to a compromise between accuracy and efficiency, several validation experiments were performed for various experimental conditions and parameter values. Table 2 shows the differences emerging from the addition of basis functions. Results are compared in terms of the mean error as compared to the worst validation

**Table 2** FEM vs ROM in the simulation of thermal pasteurisation in tunnels

| Method | Number of ODEs | Mean error T (%) | Mean error $\|w\|$ (%) | Simulation time (s) |
|---|---|---|---|---|
| FEM | 2900 | 0 | 0 | 25 |
| ROM | 10 | 1.08 | 4.3 | 3.1 |
| | 20 | 0.7 | 2.7 | 3.5 |
| | 40 | 0.47 | 1.82 | 4.2 |
| | 100 | 0.47 | 1.58 | 6.5 |

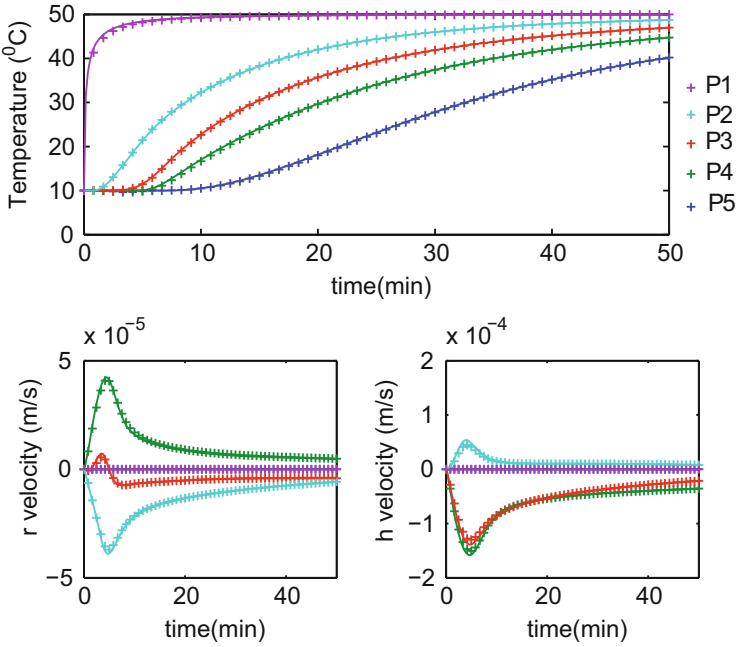


**Fig. 5** Evolution of the state variables, temperature and fluid velocity, using the POD approach (*marks*) and the FEM (*continuous lines*) at different locations within the spatial domain

example: $E = 100\frac{x_{ROM} - x_{FEM}}{x_{FEM}}$ where $x$ represents each of the state variables $T$, $w = [u, v]^T$.

The best compromise quality and computational cost is offered by the ROM with 40 ODEs. It should be noted that the mean error is bellow the 2 % as compared to the FEM based simulation. The dynamic evolution of the temperature and velocity fields at five spatial locations distributed along the diagonal of the spatial domain ($p_1 = (0, 0)$, $p_2 = (0.011, 0.022)$, $p_3 = (0.019, 0.045)$, $p_4 = (0.029, 0.067)$, $p_5 = (0.04, 0.09)$) is presented in Fig. 5 for one validation example. Continuous lines correspond to the FEM simulation while marks represent the solution of the ROM with 40 ODEs. As shown in the figure the ROM is able to reproduce the system behavior.

## 5.3 Model Identification: Production of Gluconic Acid in a Fed-Batch Reactor

Industrial fermentation is based on the conversion of glucose to other substances by the action of microorganisms under highly oxygenated and aerobic growth conditions. This kind of processes are widely employed to obtain for instance bread, wine and cheese in the food industry and biomass, metabolites (ethanol, citric acid, gluconic acid, vitamins, antibiotic) or recombinant products (insulin) in the biotechnology industry or, even, bio-fuels to replace conventional petrol.

Most of the fermentation processes to obtain Gluconic Acid (GA) are carried out by *Aspergillus niger*. The objective here is that of building a model with good predictive capabilities to describe the dynamics of glucose ($G$), oxygen ($O_2$), gluconic acid ($GA$) and biomass ($B$) during the growth phase of *Aspergillus niger*. We consider a fed-batch fermenter with two valves to regulate the incoming flux of glucose and water mixture ($u_1$) and the oxygen transfer rate described by the Henry's law ($u_2$). The controls may take values between: 0, when closed and 1, when opened. Mathematically the process may be described as follows [21]:

$$\frac{\mathrm{d}X}{\mathrm{d}t} = \mu_{max}\frac{G}{K_G + G}\frac{O_2}{K_{O_2} + O_2}, \tag{46}$$

$$\frac{\mathrm{d}GA}{\mathrm{d}t} = Y_{GA}\mu_{max}\frac{G}{K_G + G}\frac{O_2}{K_{O_2} + O_2}, \tag{47}$$

$$\frac{\mathrm{d}G}{\mathrm{d}t} = Y_G\mu_{max}\frac{G}{K_G + G}\frac{O_2}{K_{O_2} + O_2} + \frac{u_1 F_{in}}{V}(G^{in} - G), \tag{48}$$

$$\frac{\mathrm{d}O_2}{\mathrm{d}t} = Y_{O_2}\mu_{max}\frac{G}{K_G + G}\frac{O_2}{K_{O_2} + O_2} + u_2 K_{La}(O_2^* - O_2), \tag{49}$$

$$\frac{\mathrm{d}V}{\mathrm{d}t} = u_1 F_{in}, \tag{50}$$

where $F_{in}$ and $K_{La}$ represent the maximum incoming flux and oxygen transfer rate, respectively; $O_2^*$ is the saturation of dissolved oxygen; $G^{in}$ corresponds to the concentration of glucose in the inlet.

Next in model building loop is to compute model unknowns, in this case $\boldsymbol{\theta} = [\mu_{max}, Y_G, Y_{O_2}, Y_{GA}, K_{La}]$ by measuring $\mathbf{y} = [B, GA, G, O_2]$. To estimate their values we will first consider a qualitative experimental design. Basically two completely different experiments are designed: (i) where the incoming flux valve is almost closed $u_1 = 0.01$ and the oxygen transfer is completely open $u_2 = 1$ and (ii) where the incoming flux valve is completely open $u_1 = 1$ and the oxygen transfer is almost closed $u_2 = 0.01$. Pseudo-experimental data are obtained by direct numerical simulation of the model assuming the following nominal values for the parameters: $K_{La} = 600\,\mathrm{h}^{-1}$, $O_2^* = 0.0084\,\mathrm{g\,l}^{-1}$, $G^{in} = 250\,\mathrm{g\,l}^{-1}$, $F_{in} = 0.5\,\mathrm{min}^{-1}$, $\mu_{max} = 0.2242\,\mathrm{h}^{-1}$, $K_G = 9.9222\,\mathrm{g\,l}^{-1}$, $K_{O_2} = 0.0137\,\mathrm{g\,l}^{-1}$, $Y_GA = 44.8887$,

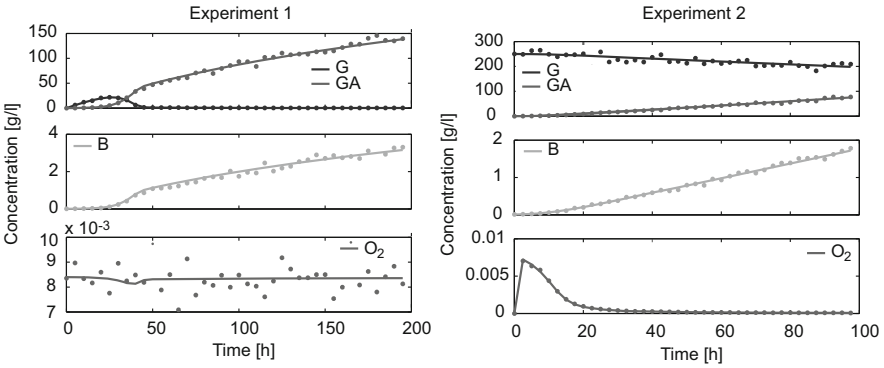


**Fig. 6** Best fit obtained for the qualitative experimental design

$Y_{O_2} = -2.5598$, $Y_G = -51.0365$. Gaussian experimental error is added to the model predictions and 40 equidistant sampling times are used per experiment.

The parameter estimation problem was solved using eSS as incorporated in AMIGO obtaining the following optimal solution:

$$\boldsymbol{\theta}^* = [0.2241, -51.049, -2.1341, 44.923, 500.2].$$

The corresponding optimal fit is shown in Fig. 6.

It should be noted that even though the values obtained for $\mu_{max}$, $Y_{GA}$ and $Y_G$ are within the 1 % of the known global solution, this is not the case for $Y_{O_2}$ and $K_{La}$ where the differences are around the 17 %. In addition the Monte Carlo based identifiability analysis reveals uncertainties over the 20 %.

In view of the results a parallel-sequential optimal experimental design was pursued in order to improve parameter estimates. The two qualitative designs are incorporated in the FIM and two new experiments are designed allowing for constant control profiles that are optimised together with the final time and the initial conditions of glucose and biomass. The OED problem was solved to minimise the ratio between the maximum and the minimum eigenvalue of the FIM and the Monte Carlo based practical identifiability analysis was performed for the resultant experimental scheme so as to compare the expected uncertainty in the parameter estimates.

The parameter estimation problem was then solved by using the four experiments in the optimal experimental scheme. Figures 7 show the two optimally designed experiments together with the optimal fits obtained by the use of SSm that correspond to the following parameter set: $\boldsymbol{\theta}^* = [0.2241, 44.908, -51.04, -2.5606, 600.04]$ which is within the 0.04 % of the optimal value, i.e. with OED it is possible to converge to the *real* parameter values.

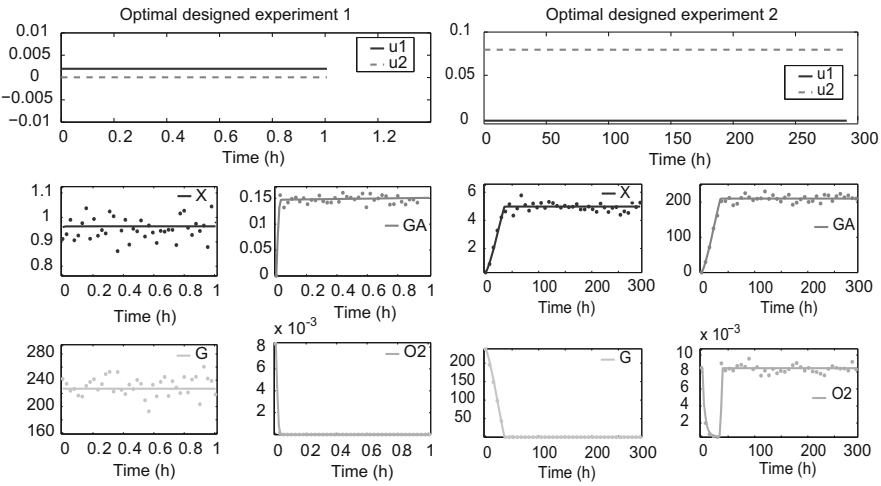


**Fig. 7** Best fit obtained for the optimally designed experiments

## *5.4 Identification and Dynamic Optimisation: Frying of Potato Chips*

In deep-fat frying foodstuff is immersed into oil at high (constant) temperature. This induces water evaporation and the formation of a thin crust. As the temperature increases and moisture is lost, the typical deep-frying sensory characteristics (colour, flavour, texture) are developed. However, the use of high temperatures results in the production of acrylamide, a carcinogen compound. Thus model-based optimisation may assist in the design of those operating conditions that provide the best compromise between quality and safety.

A multiphase porous media based model was formulated to describe heat, mass and momentum transfer and acrylamide kinetics within a potato chip as described in Warning et al. [38]. The model consists of a set of coupled nonlinear PDEs describing the evolution of the saturation of water, oil and vapor ($S_w$,$S_o$, $S_g$), product temperature ($T$), moisture content ($M$), pressure ($P$), water vapour mass fraction ($\omega_v$) and acrylamide content ($c_{AA}$). The potato chip is assumed to be cylindrical and heated from outside therefore axi-symmetry can be assumed. The selected geometry is shown in Fig. 8.

The model was solved in COMSOL©. The *Convection and Diffusion* module was used to solve for water, oil and acrylamide mass conservation while *Maxwell-Stefan Diffusion and Convection* was used to gas mass fraction and *Darcy's Law* and *Convection and Conduction* were used to solve for pressure and temperature respectively. The selected mesh consists of 20 × 10 rectangular elements. The simulation of 1.5 min frying takes around 40 s in a standard PC 3.25 GB RAM and 2.83 GHz.

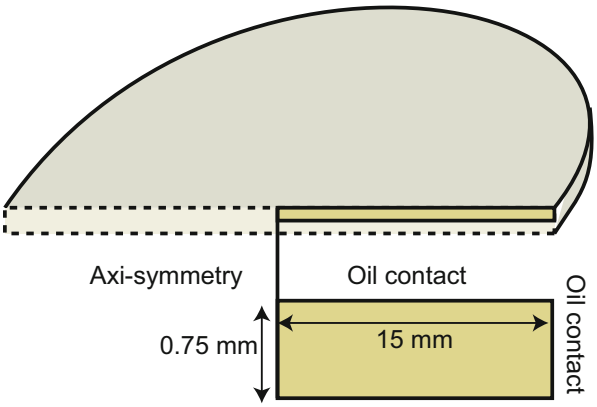


**Fig. 8** Geometry of the potato chip

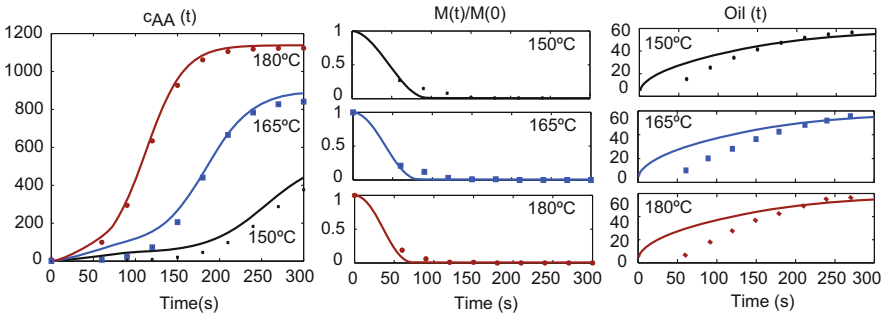


**Fig. 9** Best fit: experimental data (*dots*) vs model predictions (*lines*) of acrylamide ($c_{AA}$), oil and moisture content ($M/M(0)$) at different process temperatures

Unknown parameters, the heat transfer coefficient ($h$) and the surface oil saturation $S_{o,surf}$, were identified from experimental data using AMIGO, details can be found in Arias-Méndez et al. [3].

The final model exhibits good predictive capabilities (see Fig. 9) enabling the possibility to analyse alternative operating conditions. The objective was to compute the oil temperature profile ($T_{oil_{min}} \leq T_{oil} \leq T_{oil_{max}}$) that guaranties the desired quality attributes (colour and crispness) while minimising final acrylamide content subject to the process dynamics. The problem was solved by means of a combination of the CVP approach and eSS [17].

In a first approximation to the problem the typical industrial process at constant oil temperature was designed. As expected, the lower the oil temperature the lower the acrylamide content and the longer the process. Results reveal that a reduction in the oil temperature from 180 °C to 150 °C translates into a reduction of around the 4 % in acrylamide content and an increase of the 25 % in the process duration. Since the process duration is critical for the production rate, and no recommendations or

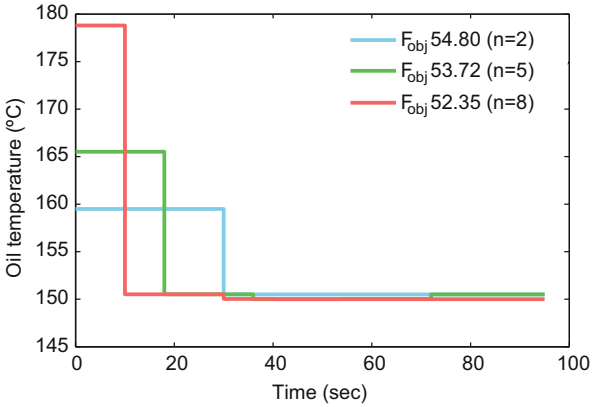


**Fig. 10** Optimal operation profiles (oil temperatures) using different numbers of heating zones ($t_{final} = 95$ s)

constraints are yet available on the maximum admissible acrylamide content, a good compromise would be to use intermediate temperature values 165–170 °C during 80–85 s.

The general dynamic optimisation problem was then solved for different maximum process durations (80, 85, 90 and 95 s) and different numbers of maximum heating zones. Results show that using two heating zones significantly reduces the final acrylamide content with respect to typical constant operating profiles. The optimal profile corresponds to the use of a higher temperature at the beginning of the process, this helping to satisfy the constraint on the moisture content, followed by a lower temperature to minimise the final acrylamide content (Fig. 10).

## *5.5 Real Time Optimisation: Thermal Sterilisation of Packaged Foods*

In this example we consider the thermal sterilisation of packaged solid foods in steam retorts. The product is introduced in a steam retort where it is subjected to a given heating-cooling cycle so as to get a pre-specified degree of microbial inactivation indicated by the microbiological lethality. However, some organoleptic properties or nutrients can be negatively affected by the heat action. The objective is, therefore, to optimise operation conditions to maximise quality while guaranteeing safety. In this example, we go a step further, and propose a real time optimisation (RTO) architecture to handle the optimisation during processing and in the presence of uncertainty or sudden disturbances. The performance of the proposed RTO architecture was experimentally validated for tuna paté at the pilot plant in the IIM-CSIC.

The dynamic representation of the plant couples the description of the temperature inside the retort, temperature distribution inside the food product and the corresponding distribution of nutrients and microorganisms:

### 5.5.1 Retort Dynamics

$$\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{f}(\mathbf{z};\boldsymbol{\theta}) + \mathbf{g}(\mathbf{z},\mathbf{u};\boldsymbol{\theta}), \tag{51}$$

here $\mathbf{f}$ and $\mathbf{g}$ are nonlinear vector fields of appropriate dimensions; $\mathbf{z}$ denotes the temperature and pressure in the retort $[T_R, P_R]$; $\mathbf{u}$ stands for the control variables: valve positions for input and output streams. Finally, $\boldsymbol{\theta}$ denotes the vector of unknown parameters. For a detailed description the reader is referred to [2].

### 5.5.2 Temperature Distribution Inside the Food Product

$$\frac{\partial T_{prod}}{\partial t} = \alpha \nabla^2 T_{prod}, \quad n(k\nabla T_{prod}) = h(T_R - T_{prod}), \tag{52}$$

where $T_{prod}$ is the temperature of the food stuff and $h$, $k$, $\alpha$ stand for the heat transfer coefficient of the package and the food thermal conductivity and diffusivity, respectively.

### 5.5.3 Quality and Safety Models

$$\frac{\mathrm{d}C_i(t)}{\mathrm{d}t} = -\left(\frac{\ln 10}{D_{i,ref}}\right) C_i(t) \exp\left(\frac{T_{prod}(\boldsymbol{\xi},t) - T_{\boldsymbol{\xi},ref}}{z_{i,ref}}\right), \tag{53}$$

where subindex $C_i$ refers to the concentration of either microorganisms or nutrients.

The unknown parameters of the model, the functional dependencies of fluxes on valves openings and the valves related constants were identified by means of parameter estimation, identifiability analysis and multi-experimental optimal design, using AMIGO toolbox.

For the case of the evolution of temperature inside the retort, the resulting model presents excellent predictive capabilities taking into account that a maximum error of around 3 % is observed in fast transitions.

The product was packed in glass containers with metal top. The corresponding geometry and the FEM mesh used for simulation purposes are depicted in Fig. 11. Selected mesh consists of 184 nodes which translates into 553 ODEs. Three model parameters were estimated from the temperature measurements, namely,

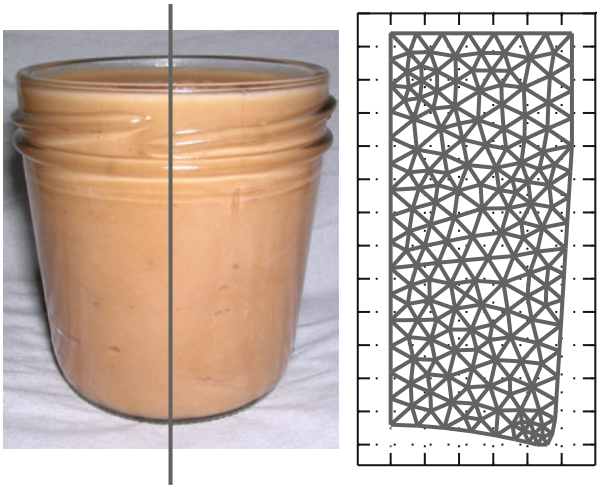

**Fig. 11** Geometry of the food package

the product thermal conductivity, and the glass/steam and the metal/steam heat transfer coefficients. After the model identification, the differences between model predictions and experimental data are lower than 1 %.

Once a satisfactory model became available, a POD-based ROM model was developed to be used within the RTO scheme, it should be noted that each simulation of the ROM takes less than 1 s. In addition, the optimal operating conditions were computed off-line using the CVP and scatter search methods.

Real time implementation of the optimal control needs to consider the effect of unmeasured disturbances not being part of the prediction model. To that purpose, feedback was implemented by regularly measuring the current retort variables and observing the relevant variables of the packaged product to compute efficient on-line optimisation. Optimal operation conditions are then re-computed any time a difference between predicted value and off-line optimal solution is detected. A combination of a local optimiser and SSm was designed so as to guarantee feasibility and optimality of the solution even in the presence of significant perturbations or plant/model mismatch (see details in [2]).

Figures 12 and 13 illustrate the performance of the RTO architecture in an experimental case were large perturbations occur. The implementation of the optimal off-line heating profile leads to a product that does not fulfill the lethality requirement ($F_c = 8$ min). The RTO architecture proposed in the work was able to drive the system to feasibility and optimality by means of re-computing optimal profiles on-line and slightly extending the duration of the heating phase.

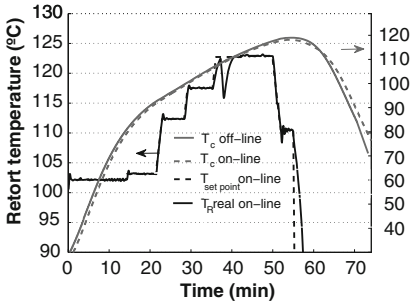


**Fig. 12** Comparison of off-line and on-line optimal profiles under large perturbations in the retort at the pilot plant (IIM-CSIC)

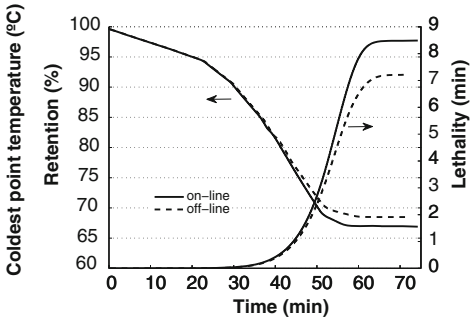


**Fig. 13** Comparison of off-line and on-line optimal profiles surface nutrient retention and lethality value at the coldest point

## 6 Conclusions

Computer-aided simulation and model-based optimisation offer a powerful, rational and systematic way to improve food, bio-process and biological systems understanding or performance. In recent decades there has been a growing interest in the development of rigorous models, based on first principles, that enable not only to perform experiments in silico, but to design and to optimise operation policies.

However several problems have to be faced mostly related to (i) insufficient a priori knowledge to deduce the right model structure or model parameter values; (ii) the complexity of the processes that combine physical, chemical and biological phenomena on a wide range of time and space scales; (iii) the complexity of the associated models that calls for sophisticated numerical simulation techniques and (iv) the complexity of the associated optimisation problems due mainly to multimodality.

In this work we have used a number of examples taken from the food and biotechnology industry to illustrate how those problems emerge and to present some alternatives to tackle them. Special emphasis was paid to describe the model identification loop, which involves parameter estimation, identifiability analyses and model based experimental design as well as the dynamic optimisation problem. Most of the problems can be formulated as non-linear optimisation problems whose solution requires adequate model simulation techniques, including accurate and efficient reduced order modelling approaches and the use of global optimisation

methods. To finish with, all elements were combined to design and implement a real-time optimisation architecture, which is able to assure high operational stability, process reproducibility and optimal operation.

**Acknowledgements** The authors acknowledge financial support from CSIC [PIE201270E075]. A. Arias-Méndez and M. Mosquera-Fernández acknowledge financial support from the JAE-CSIC program, A. López-Nuñez acknowledges financial support from Xunta de Galicia.

## References

1. Alonso, A.A., Frouzakis, C., Kevrekidis, I.: Optimal sensor placement for state reconstruction of distributed process systems. AIChE J. **50**(7), 1438–1452 (2004)
2. Alonso, A., Arias-Méndez, A., Balsa-Canto, E., García, M., Molina, J., Vilas, C., Villafin, M.: Real time optimization for quality control of batch thermal sterilization of prepackaged foods. Food Control **32**(2), 392–403 (2013)
3. Arias-Méndez, A., Warning, A., Datta, A., Balsa-Canto, E.: Quality and safety driven optimal operation of deep-fat frying of potato chips. J. Food Eng. **119**(1), 125–134 (2013)
4. Balsa-Canto, E., Banga, J.: AMIGO, a toolbox for advanced model identification in systems biology using global optimization. Bioinformatics **27**(16), 2311–2313 (2011)
5. Balsa-Canto, E., Alonso, A.A., Banga, J.R.: A novel, efficient and reliable method for thermal process design and optimization. Part i: theory. J. Food Eng. **52**, 227–234 (2002)
6. Balsa-Canto, E., Alonso, A.A., Banga, J.R.: Reduced-order models for nonlinear distributed process systems and their application in dynamic optimization. Ind. Eng. Chem. Res. **43**, 3353–3363 (2004)
7. Balsa-Canto, E., Vassiliadis, V., Banga, J.: Dynamic optimization of single- and multi-stage systems using a hybrid stochastic-deterministic method. Ind. Eng. Chem. Res. **44**(5), 1514–1523 (2005)
8. Balsa-Canto, E., Alonso, A.A., Banga, J.: Computational procedures for optimal experimental design in biological systems. IET Syst. Biol. **2**(4), 163–172 (2008)
9. Balsa-Canto, E., Alonso, A., Banga, J.: An iterative identification procedure for dynamic modeling of biochemical networks. BMC Syst. Biol. **4**(11) (2010). doi: 10.1186/1752-0509-4-11
10. Balsa-Canto, E., López-Núñez, A., Vázquez, C.: Numerical solution of a biofilm model based on pdes. Poster Presented in the XVI Jacques-Louis Lions Spanish-French School on Numerical Simulation in Physics and Engineering, EHF2014, Pamplona (2014)
11. Beyenal, H., Lewandowski, Z., Harkin, G.: Quantifying biofilm structure: facts and fiction. Biofouling **20**, 1–23 (2004)
12. Biegler, L., Cervantes, A., Wächter, A.: Advances in simulaneous strategies for dynamic process optimization. Chem. Eng. Sci. **57**(4), 575–593 (2002)
13. Bock, H., Plitt, K.: A multiple shooting algorithm for direct solution of optimal control problems. In: Proceedings of the 9th IFAC World Congress, pp. 242–247. Pergamon Press, New York (1984)
14. Eberl, H.J., Parker, D.F., Loosdrecht, M.C.V.: A new deterministic spatio temporal continuum model for biofilm development. J. Theor. Med. **3**, 161–175 (2001)
15. Egea, J., Rodríguez-Fernández, M., Banga, J., Marti, R.: Scatter search for chemical and bio-process optimization. J. Glob. Optim. **37**(3), 481–503 (2007)
16. Egea, J., Balsa-Canto, E., García, M., Banga, J.: Dynamic optimization of nonlinear processes with an enhanced scatter search method. Ind. Eng. Chem. Res. **48**(9), 4388–4401 (2009)
17. Egea, J., Martí, R., Banga, J.: An evolutionary method for complex-process optimization. Comput. Oper. Res. **37**(2), 315–324 (2010)

18. Floudas, C.: Deterministic Global Optimization: Theory, Methods and Applications. Kluwer Academics, Dordrecht (2000)
19. García, M.R., Vilas, C., Banga, J., Alonso, A.: Optimal field reconstruction of distributed process systems from partial measurements. Ind. Eng. Chem. Res. **46**(2), 530–539 (2007)
20. García, M.R., Vilas, C., Banga, J.R., Alonso, A.A.: Exponential observers for distributed tubular (bio)reactors. AIChE J. **54**(11), 2943–2956 (2008)
21. García, M., Vilas, C., Balsa-Canto, E., Lyubenovac, V., Ignatovac, M., Alonso, A.: On-line estimation in a distributed parameter bioreactor: application to the gluconic acid production. Comput. Chem. Eng. **35**(1), 84–91 (2011)
22. Gottlieb, D., Orszag, S.A.: Numerical Analysis of Spectral Methods: Theory and Applications. Society for Industrial and Applied Mathematics, Philadelphia (1977)
23. Lapidus, L., Pinder, G.: Numerical Solution of Partial Differential Equations in Science and Engineering. Wiley, New York (1999)
24. Ljung, L.: System identification: theory for the user. Prentice Hall, Englewood Cliffs (1999)
25. Mosquera-Fernández, M., Rodríguez-López, P., Cabo, M., Balsa-Canto, E.: Numerical spatio-temporal characterization of listeria monocytogenes biofilms. Int. J. Food Microbiol. **182**, 26–36 (2014)
26. Palazoglu, T., Erdogdu, F., Uyar, R.: Experimental comparison of natural convection and conduction heat transfer. J. Food Process Eng. **33**, 85–100 (2010)
27. Pardalos, P., Romeijn, H., Tuyb, H.: Recent developments and trends in global optimization. J. Comput. App. Math. **124**, 209–228 (2000)
28. Pinter, J.: Global Optimization in Action. Continuous and Lipschitz Optimization: Algorithms, Implementations and Applications. Kluwer Academics, Dordrecht (1996)
29. Reddy, J.: An Introduction to the Finite Element Method. McGraw-Hill, New York (1993)
30. Rodríguez-Fernández, M., Egea, J., Banga, J.: Novel metaheuristic for parameter estimation in nonlinear dynamic biological systems. BMC Bioinformatics **7**, 483 (2006)
31. Sirovich, L.: Turbulence and the dynamics of coherent structures. Part I: coherent structures. Q. Appl. Math. **45**(3), 561–571 (1987)
32. Srey, S., Jahid, I., Ha, S.D.: Biofilm formation in food industries: a food safety concern. Food Control **31**(2), 572–585 (2013)
33. Talbi, E.G.: Metaheuristics: From Design to Implementation. Wiley, Hoboken (2009)
34. Vande Wouwer, A., Saucez, P., Vilas, C.: Simulation of ODE/PDE Models with MATLAB, OCTAVE and SCILAB: Scientific and Engineering Applications. Springer, Cham (2014)
35. Vassiliadis, V.S., Pantelides, C.C., Sargent, R.W.H.: Solution of a class of multistage dynamic optimization problems. 1. Problems without path constraints. Ind. Eng. Chem. Res. **33**(9), 2111–2122 (1994)
36. Walter, E., Pronzato, L.: Identification of Parametric Models from Experimental Data. Springer, Masson (1997)
37. Wanner, O., Eberl, H.J., Morgenroth, E., Noguera, D.R., Picioreanu, C., Rittmann, B.E., van Loosdrecht, M.C.: Mathematical modeling of biofilms. Technical report, IWA Task Group on Biofilm Modeling (2006)
38. Warning, A., Dhall, A., Mitrea, D., Datta, A.: Porous media based model for deep-fat vacuum frying potato chips. J. Food Eng. **110**(3), 428–440 (2012)
39. Yang, X., Beyenal, H., Harkin, G., Lewandowski, Z.: Quantifying biofilm structure using image analysis. J. Microbiol. Methods **39**, 109–119 (2000)

# Part III
# Advanced Computational Techniques

# An Introduction to GPU Computing for Numerical Simulation

**José Miguel Mantas, Marc De la Asunción, and Manuel J. Castro**

**Abstract** Graphics Processing Units (GPUs) have proven to be a powerful accelerator for intensive numerical computations. The massive parallelism of these platforms makes it possible to achieve dramatic runtime reductions over a standard CPU in many numerical applications at a very affordable price. Moreover, several programming environments, such as NVIDIA's Compute Unified Device Architecture (CUDA) have shown a high effectiveness in the mapping of numerical algorithms to GPUs. These notes provide an introduction to the development of CUDA programs for numerical simulation using CUDA C/C++, the most popular GPU programming toolkit. An overview of CUDA programming will be illustrated through the CUDA implementation of simple numerical examples for PDEs. These CUDA implementations will be studied and run on modern GPU-based platforms.

**Keywords** Graphics processing units • CUDA • Numerical solution of PDEs • CUDA C programming

## 1 Introduction to GPU Computing

A *graphics processing unit (GPU)* is a programmable single-chip processor which is used primarily for things such as: rendering of 3D graphics scenes, 3D object processing and 3D motion. All these tasks are computing-intensive and highly parallel. A GPU performs arithmetic operations in parallel on multiple data to render images. Generally, GPUs are embedded in stand-alone cards which include their own memory. In order to use them, these cards must be connected to the motherboard of a CPU-based computer system, by using the suitable high speed connection (Peripheral Component Interconnect Express bus, PCI-Express [11]) (Fig. 1).

J.M. Mantas (✉)
Departamento de Lenguajes y Sistemas informáticos, Universidad de Granada, Granada, Spain
e-mail: jmmantas@ugr.es

M. De la Asunción • M.J. Castro
Dpto. Análisis Matemático, Universidad de Málaga, Málaga, Spain
e-mail: marcah@uma.es; castro@anamat.cie.uma.es



I. Higueras et al. (eds.), *Numerical Simulation in Physics and Engineering*,
SEMA SIMAI Springer Series 9, DOI 10.1007/978-3-319-32146-2_5

**Fig. 1** GPU cards

Modern GPUs are highly programmable and can be used for non-graphics applications. They offer hundreds or thousands of processing units optimized for massively performing floating-point operations in parallel. In fact, they can be a cost-effective way to obtain a substantially higher performance in computationally intensive tasks [5, 25, 26, 28].

In many compute-intensive applications, there is a large performance gap between GPUs and multicore CPUs [28]. Nowadays the peak performance of a modern and powerful GPU is approximately seven times the peak performance of a corresponding CPU in single precision and close to five times in double precision [5, 21]. This is mainly due to two reasons:

- While CPUs are designed to minimize the execution latency of a single task (latency-oriented platforms), the design goal of GPUs is to maximize the total execution throughput of a large number of parallel tasks (they are massively parallel throughput-oriented computing platforms).
- The graphics chip memories are usually much faster than the corresponding CPU chip memories.

Moreover, since massively parallel GPUs are easily accessible in comparison with huge cluster-based parallel machines, GPU-based platforms have made it possible to increase the use and spread of the high performance computing in scientific and engineering environments.

*GPU computing* consists of using GPUs together with CPUs to accelerate the solution of compute-intensive science, engineering and enterprise problems. Since the numerical simulation based on Partial Differential Equations (PDEs) exhibits a lot of exploitable parallelism, there has been an increasing interest in the acceleration of these simulations by using GPU-based computer systems.

Initially, graphics-specific programming languages and interfaces [8, 9, 14, 16, 26, 27] were used to program GPUs. However, the use of these languages

complicates the programming of GPUs in scientific applications, and the code obtained is difficult to understand and maintain. Later, NVIDIA developed the CUDA programming toolkit [21] which includes an extension of the C language which facilitates the programming of GPUs for general purpose applications [10] by preventing the programmer to deal with the graphics details of the GPU. Additionally, several languages and interfaces such as OpenCL [7, 12], OpenACC [24] or Thrust [4], have been developed and spread to make the GPU computing easier.

There is a widespread use of CUDA-based platforms to accelerate numerical solvers for PDEs [1–3, 6]. For this reason, these notes intend for making it easier the exploitation of CUDA-enabled platforms to accelerate PDE-based numerical simulations, by providing the suitable CUDA C programming foundations.

## 2 CUDA Introduction

The *CUDA (Compute Unified Device Architecture)* framework [21] is a hardware and software platform that makes it possible to exploit efficiently the potential of NVIDIA GPUs to accelerate the solution of many costly computational problems. This framework includes a unified architectural view of the GPU and a multithreaded programming model which uses an extension to the C programming language (called CUDA C) to implement general-purpose applications on NVIDIA GPUs. Other languages are also supported [21] for CUDA programming (C++, Fortran, Java, etc.).

According to the CUDA framework, a GPU is viewed as a computing device which works as a coprocessor for the main CPU (host). Both the CPU and the GPU maintain their own Dynamic Random Access Memory (DRAM) (see Fig. 2) and it is possible to copy data from CPU memory to GPU memory and vice versa.

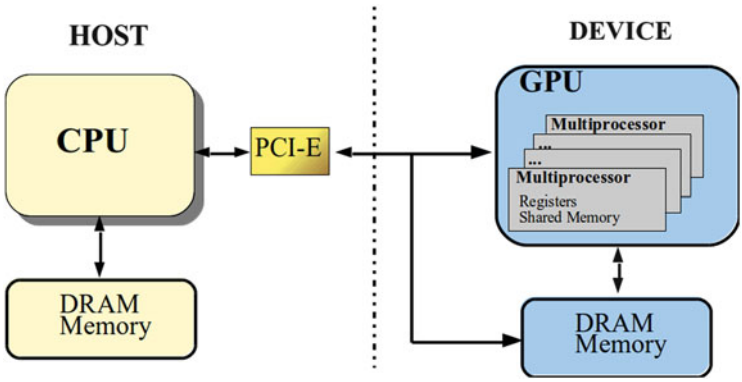


**Fig. 2** GPU system (device) as a coprocessor of the CPU system (host)

## 3 Brief Introduction to the CUDA Hardware Model

In the CUDA framework, we consider that a CUDA enabled GPU is formed by $N$ multiprocessors ($N$ depends on the particular GPU), each one having $M$ processors or cores (see Fig. 4) where the value of $M$ also depends on the specific GPU architecture (it can be 8, 32, 192, ...). At any clock cycle, each core of the multiprocessor executes the same instruction, but operates on different data. The multiprocessors of the GPU are specialized in the parallel execution of multiple CUDA threads. A *CUDA thread* represents a sequential computational task which executes an instance of a function. Each CUDA thread is executed logically in parallel with respect to other CUDA threads (associated to the same function but operating on different data) on the cores of a GPU multiprocessor (see Fig. 3).

### *3.1 CUDA Memory Model*

A CUDA thread that runs on a multiprocessor of the GPU has access to the following memory spaces (see Fig. 4):

- **On-chip memories**:
  - *Registers*: Each thread has its own readable and writeable registers. Each GPU model has a particular number of registers per multiprocessor, which are split and assigned to the threads that run concurrently on that multiprocessor.
  - *Shared memory*: Each multiprocessor includes a small memory (between 16 and 48 KB) called *shared memory*. This memory is readable and writeable

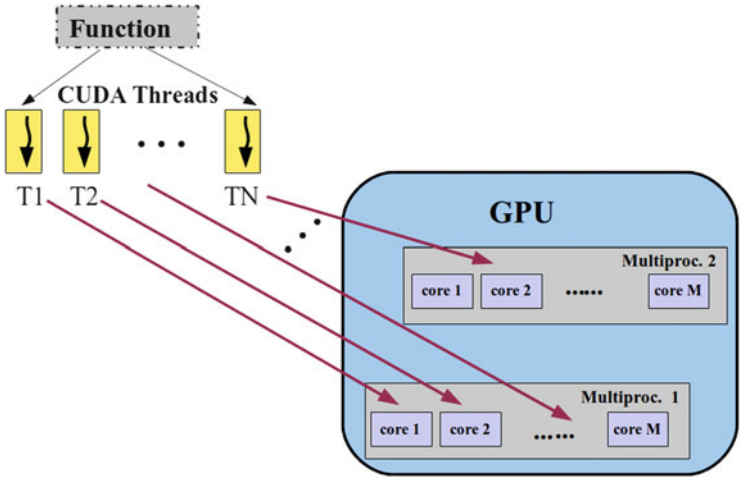


**Fig. 3** Execution of multiple CUDA Threads (associated with the same function)

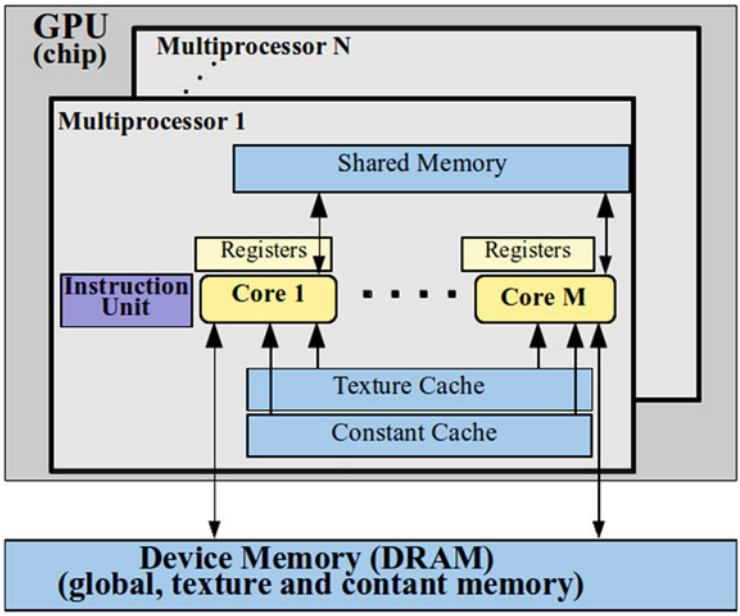


**Fig. 4** CUDA hardware model

only from the CUDA threads running on the particular multiprocessor of the GPU and is much faster than global memory.

- **Off-chip memories** (in DRAM memory): These spaces are not resident in the GPU chip and they are shared by all CUDA threads which are being executed on the GPU (on several multiprocessors).
  - *Global memory*: This memory is readable and writeable from CPU and GPU. It is slow in comparison with the shared memory due to its high latency (500 times slower approximately).
  - *Constant memory*: It is readable from GPU and writeable from CPU. It is cached, making it faster than global memory if the data is in cache.
  - *Texture memory*: It is readable from GPU and writeable from CPU. Under several circumstances, the use of the texture memory can provide performance benefits because it is cached (there is a small texture cache memory for each multiprocessor) and optimized for 2D spatial locality. Texture cache size varies between 6 and 8 KB per multiprocessor.

# 4 CUDA Programming Model

## *4.1 CUDA Kernels and Threads*

In order to specify the function to be executed by each thread on the GPU, CUDA C allows the programmer to define special C functions, called CUDA *kernels*. A CUDA kernel function is called from the CPU and is executed *N* times on the GPU by an array of *N* CUDA threads (see Fig. 5).

CUDA Threads are extremely light because they exhibit a very fast creation and context switching. As a consequence, it is recommendable to use fine-grain parallel decompositions (running very many thousands of CUDA threads) to obtain high efficiency.

Every thread executes the same code (the kernel function) but the specific action (and the data subdomain which is processed) to be performed depends on the thread identifier. The thread identifier can be obtained from the built-in variable `threadIdx` and is used to compute memory addresses and take control decisions (see Fig. 5).

Listing 1 shows a naive CUDA program which includes the declaration of a straightforward kernel and the corresponding kernel launch statement. The kernel (`VecAdd`) is used to add two vectors (with *N* floating point elements), `A` and `B`, obtaining the result in a vector `C`. As can be seen, a kernel function is declared using the modifier `__global__` and it must return `void` type. We can see that each thread of the kernel computes one element of the output vector `C`. In order to launch a kernel, the programmer must specify several parameters which determine, in particular, the number of CUDA threads which are to be created to execute it. In this example, the kernel invocation code indicates that an array of *N* threads (one for each element of the vectors) is launched on the GPU. The number and organization of the threads used to execute a kernel is given by several parameters between `<<<` and `>>>` as we will see later.

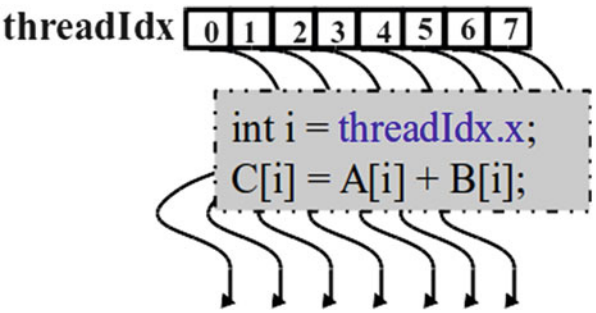


**Fig. 5** Thread array for a naive CUDA kernel

```
...
__global__ void VecAdd(float* A, float* B, float* C)
{ int i = threadIdx.x;
   C[i] = A[i] + B[i]; }
...
int main()
 {
   ...
   // Kernel call with N Threads
   VecAdd <<<1, N>>> (A, B, C);
 }
```

**Listing 1** Outline of a naive one-block CUDA program to add two vectors

## *4.2 CUDA Thread Organization: Grids and Blocks*

The set of CUDA threads which are created in a kernel launch is organized forming a grid of thread blocks that run logically in parallel [21].

### 4.2.1 CUDA Thread Blocks

Each thread is identified with a (1D, 2D or 3D) thread Index (stored in the predefined variable `threadIdx`) in a *thread block*. A CUDA block is the unit used to map threads to multiprocessors. Each thread block runs on a single multiprocessor of the GPU.

Each multiprocessor can process a maximum number of blocks at a particular moment in time, and each block can include a maximum number of threads. It depends on the so-called *Compute Capability* of the GPU. The predefined variable `blockDim`, with type `dim3`, stores the block dimensions for a kernel (see Fig. 6). The `dim3` type is a `struct` with 3 unsigned integer fields (`x`, `y` and `z`).

The threads of a block can communicate among themselves using the shared memory of the multiprocessors assigned to that block.

As can be seen in the naive code of Listing 1, that kernel is designed to launch only one thread block. Consequently the size of the vectors ($N$) can not be greater than the maximum number of threads per block (which depends on the particular GPU architecture).

### 4.2.2 Grid of Blocks

As we have seen, several CUDA threads which will be executed on the same multiprocessor, are organized forming a (1D, 2D or 3D) matrix of threads called block. In a similar way, all the blocks which are created in a kernel launch are

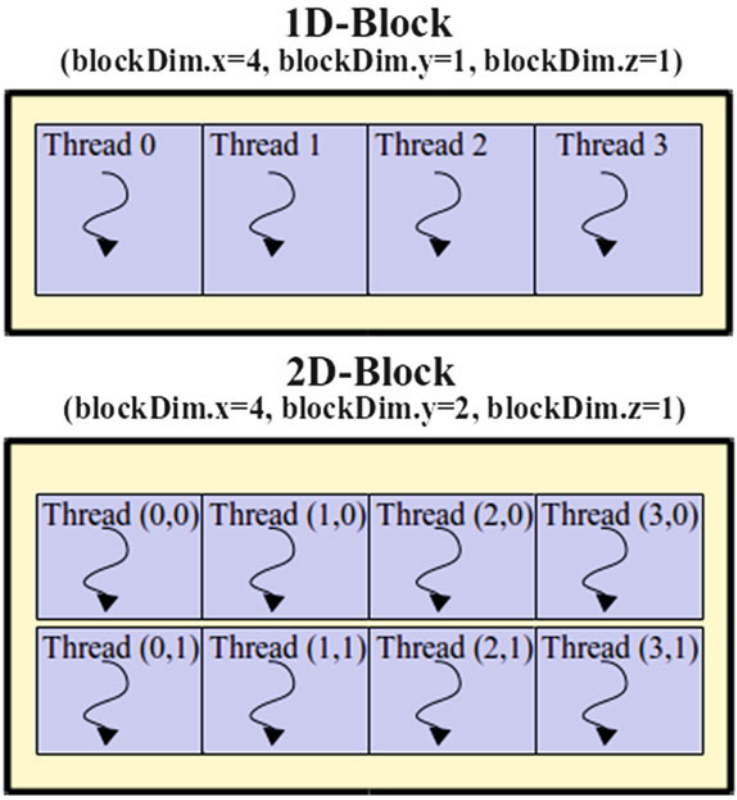


**Fig. 6** Structure of 1D and 2D blocks of CUDA threads

organized as a (1D, 2D or 3D) matrix called *grid* (see Fig. 7). In general, the grid is a 3D array of blocks but the programmer can choose to use fewer dimensions

The number of threads per block and the number of blocks per grid used to launch a kernel are specified with the following syntax:

```
<<<   blocks_per_grid, threads_per_block >>>
```

The first parameter gives the number of thread blocks in the grid. The second gives the number of threads in each thread block. In the program of Listing 1, we run one grid with only one block of $N$ threads. This program would only exploit one multiprocessor of the GPU. A more practical CUDA program to add vectors is given in Listing 2, where a one-dimensional grid with $\lceil \frac{N}{256} \rceil$ one-dimensional blocks with 256 threads is used (see Fig. 8).

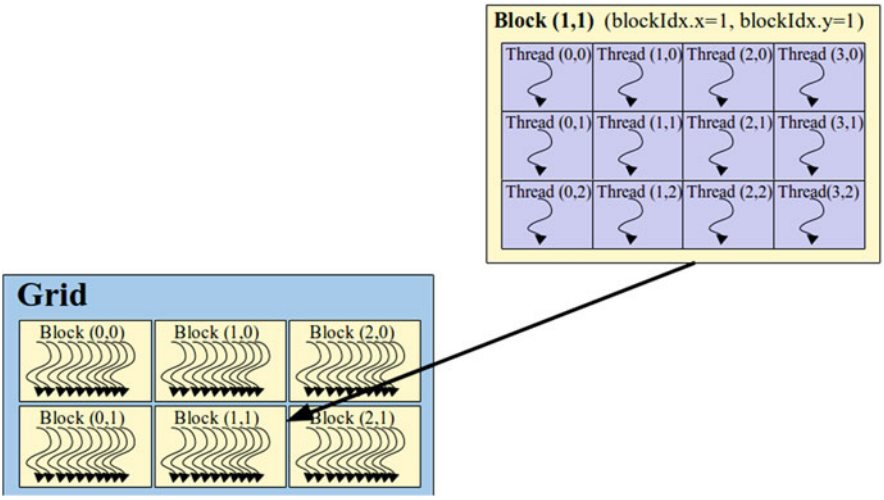


**Fig. 7** Grid of thread blocks

```
#define BSIZE 256
__global__ void VecAdd(float* A, float* B, float* C, int N)
   { int i = blockIdx.x * blockDim.x + threadIdx.x;
     if (i<N)   C[i] = A[i] + B[i];
   }
...
int main()
 {
  ...
  // Kernel call with N Threads using 256 threads 1D-blocks
  VecAdd <<<ceil((float)N/BSIZE), BSIZE>>> (A,B,C,N);
  ...
 }
```

**Listing 2** A more general CUDA program to add vectors

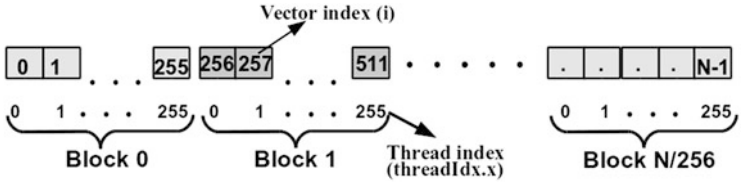


**Fig. 8** Mapping of a vector to one-dimensional blocks

In order to identify several parameters of a particular thread (to distinguish it among the others), programmers can use the following predefined variables within the kernel function:

- `uint3 blockIdx`: identifies the coordinates of a particular block in the grid. The `uint3` type has the same structure as `dim3` (`blockIdx.x`, `blockIdx.y`, `blockIdx.z`).
- `dim3 blockDim`: identifies the dimensions of the block.
- `dim3 gridDim`: maintains the grid dimensions.

Using these spatial indices (including `threadIdx`), the programmer can specify what particular data subdomain will be operated by each CUDA thread. In Listing 2, these indices are used to identify what elements of the input and output vectors are processed by a particular thread. Since not all threads will compute elements of the output vector, a conditional `if` statement is needed to test if the global index values of a thread are within the valid range (if $i \geq N$ nothing is computed).

The grid and block dimensions for a kernel must be chosen by the programmer depending on the nature of the data in order to maximize efficiency. For example, to add two 2D matrices of `float` elements, it can be convenient to use a 2D grid which consists of 2D blocks of threads, where each thread computes one element of the output matrix. Figure 9 shows this type of 2D mapping to compute the addition of $45 \times 45$ matrices. In this figure, it is assumed that we are using $16 \times 16$ 2D-blocks (`blockDim.x` $=$ `blockDim.y` $= 16$, `blockDim.z` $= 1$). As can be seen, we need a $3 \times 3$ grid of blocks (`gridDim.x` $=$ `gridDim.y` $= 3$, `gridDim.z` $= 1$). Note that we have three extra threads in the $x$ and $y$ directions.

Listing 3 shows an outline of a CUDA program to add $N \times N$ matrices. In the `MatAdd` kernel, we initially compute the position of the current thread in the horizontal and vertical directions, using the built-in variables `blockIdx` and `threadIdx` (see Fig. 9). These position values (`i` and `j`) are used to derive the global 1D index for the input and output matrices. This 1D index is used to compute the corresponding element of matrix `C` if the previously computed position values (`i` and `j`) of the thread are within the valid range.

### *4.3 Transparent Scalability*

A CUDA thread block can be executed independently with respect the rest of CUDA blocks of a kernel. Therefore the execution of several blocks can follow any relative execution order. Taking into account that the thread blocks are automatically mapped onto multiprocessors by the CUDA runtime system, this flexibility in the execution order enables the automatic adaptation of a kernel execution to the number of multiprocessors of the available GPU (see Fig. 10). This is called *transparent scalability*, because the blocks are mapped to the specific number of multiprocessors of a particular GPU in an efficient way without any intervention from the user.

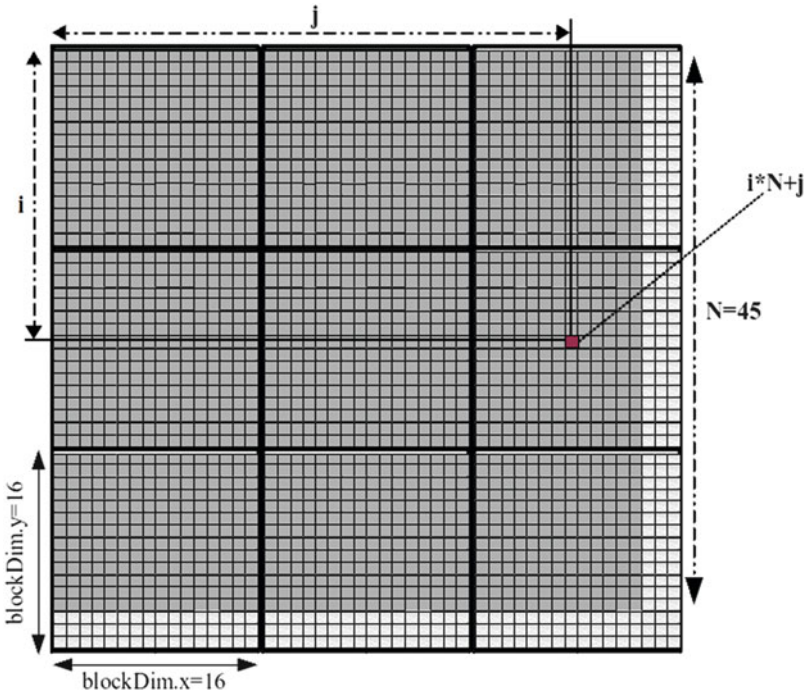


**Fig. 9** Mapping of a 2D grid of 16 × 16 blocks to a 45 × 45 matrix

## *4.4 CUDA Program Structure*

A CUDA C program must include the code for the CPU which includes the calls to several CUDA kernels and the specification of these kernel functions using the suitable C extensions (see Fig. 11). A kernel can only start on a GPU when the previous CUDA function call has returned.

The qualifiers for functions which run on GPU are the following:

- The qualifier `__global__` introduces a function (a kernel) which runs on GPU (device) but is called from the host:

```
__global__ void Kernel_name (...) {.....}
```

```
...
__global__ void MatAdd (float *A, float *B, float * C, int N)
    {
     // Compute row index in A, B, and C
     int j = blockIdx.x * blockDim.x + threadIdx.x;
     // Compute column index in A, B, and C
     int i = blockIdx.y * blockDim.y + threadIdx.y;
     // Compute global 1D index in A, B, and C
     int index=i*N+j;
     // Thread computes C element if within valid range
     if (i < N && j < N)
        C[index] = A[index] + B[index];
    }

  int main()
    {  .......
     // Kernel Launch code
     dim3 threadsPerBlock (16, 16);
     dim3 numBlocks( ceil ((float)(N)/threadsPerBlock.x),
                 ceil ((float)(N)/threadsPerBlock.y) );
     MatAdd <<<numBlocks, threadsPerBlock>>> (A, B, C, N);
     }
```

**Listing 3** CUDA program to add matrices

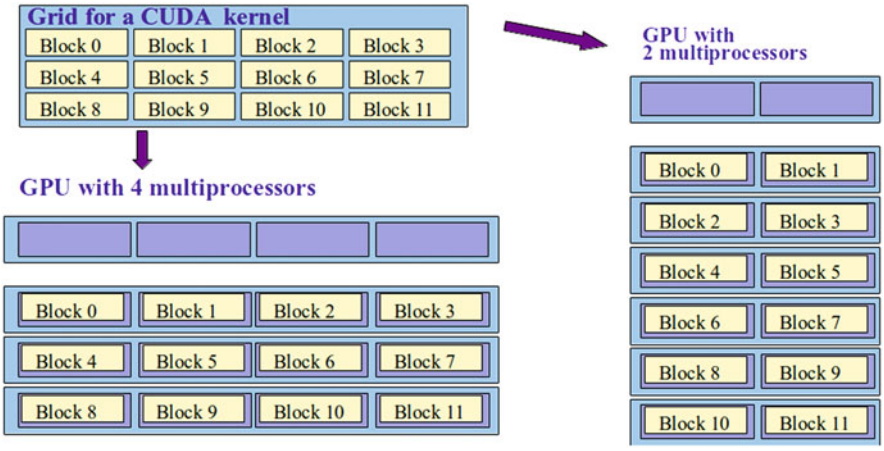


**Fig. 10** Transparent scalability for CUDA programs

- The qualifier `__device__` introduces a function which runs on GPU and is called from the GPU (device):

```
__device__ int Function_name (...) {.....}
```

These device functions are invoked in the kernel function.

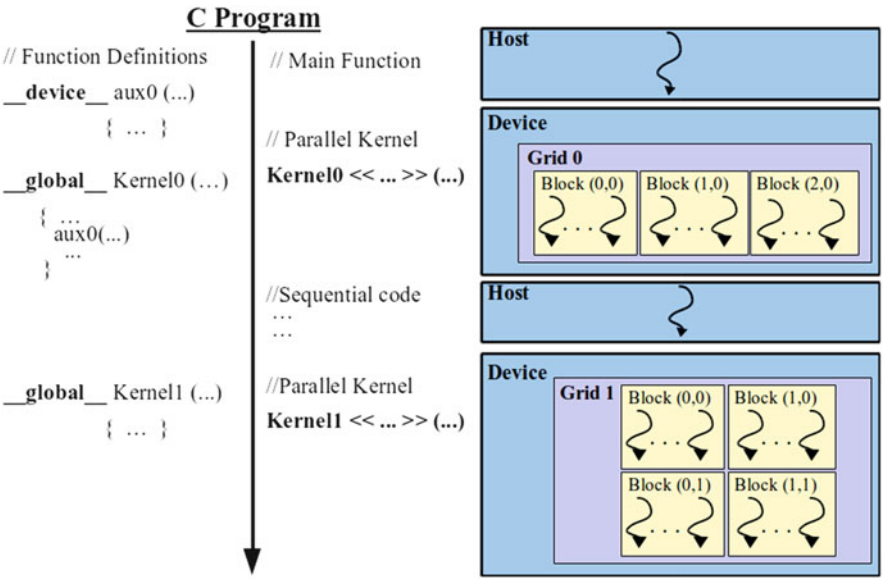


**Fig. 11** Structure of a CUDA program

## *4.5 Memory Organization and Variable Declarations*

As we introduced in Sect. 3.1, CUDA supports several types of memory. Each memory type has usage constraints and can be used by programmers declaring suitably the variables in the CUDA program. The adequate use of the available memory types is very important to obtain a good performance in a kernel execution. Each type of memory has different characteristics that we can denote by:

- **Functionality**: it is the intended use of the memory space.
- **Size**: it is measured in number of Kbytes.
- **Average access speed**: it indicates the global efficiency in accessing data which are allocated in the particular memory space.
- **Variable declaration**: it specifies how we must declare the variables which are resident in that memory space.
- **Scope**: it identifies the group of threads which can access a variable which is resident in that memory space.
- **Lifetime**: it denotes the phase of the program execution where the variable (resident in that memory space) is available for use.

Now we present the properties of the multiple memory spaces where CUDA threads can access data during the execution of a CUDA program (see Fig. 12):

- **Registers**: This memory space is used to hold variables which are frequently accessed by each thread in a kernel function. Each CUDA thread has its own

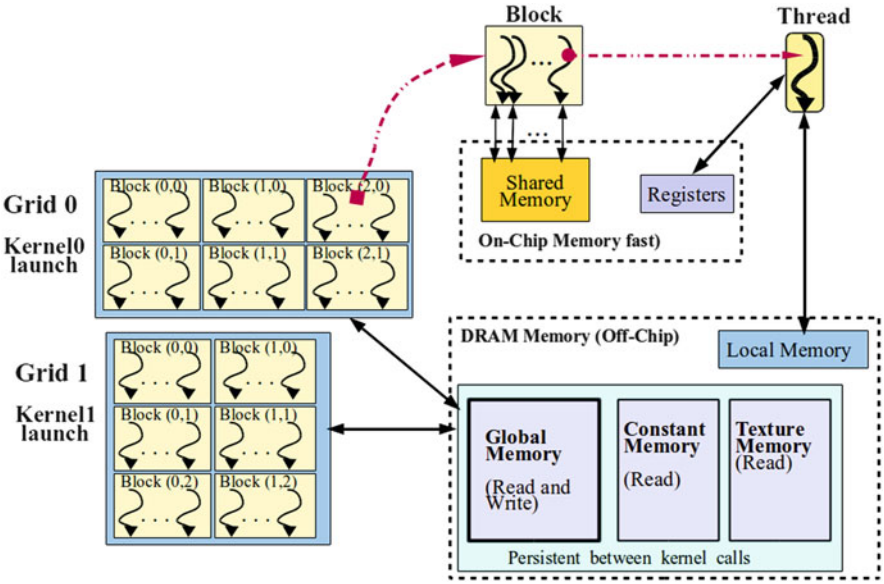


**Fig. 12** Memory organization in a CUDA program

private registers. The speed access to registers is extremely high. While the capacity of the register storage is not exceeded, all scalar variables (not arrays) declared (without qualifiers) in kernel and device functions are placed into registers. The scope of these variables is the single owner thread (when the thread finishes its execution, these variables are also terminated). The number of 32-bit registers per thread and multiprocessors for several modern GPU architectures is shown in Table 1.

- **Local memory**: Array variables declared (without qualifiers) in a kernel and several scalar variables (also declared without qualifiers) which exceed the limited capacity of the register storage are not stored in registers. These variables have the same scope and lifetime as register variables but they are not stored into the registers (they are usually stored in DRAM device memory). Therefore, the speed access to local memory could be very low and this can generate long access delays.
- **Shared memory**: Shared memory is part of the memory space which resides on the processor chip. This implies that the access speed is also very high (in comparison with the access to variables resident in global memory) but lower than accessing directly to registers. The variables resident in shared memory (shared memory variables) are accessible, to read and write, for all the threads within the same block (the threads of the block share that space memory) and only last for the thread block activity period. They provide an excellent means by which threads within a block can communicate and collaborate efficiently on computations.

**Table 1** CUDA Compute Capability of Fermi, Kepler and Maxwell GPUs

| | Fermi GF100 | Fermi GF104 | Kepler GK104 | Kepler GK110 | Maxwell GM107 |
|---|---|---|---|---|---|
| Compute capability | 2.0 | 2.1 | 3.0 | 3.5 | 5.0 |
| Threads/warp | 32 | 32 | 32 | 32 | 32 |
| Max warps/multiprocessor | 48 | 48 | 64 | 64 | 64 |
| Max threads/multiprocessor | 1536 | 1536 | 2048 | 2048 | 2048 |
| Max blocks/multiprocessor | 8 | 8 | 16 | 16 | 16 |
| 32-bit registers/multiprocessor | 32,768 | 32,768 | 65,536 | 65,536 | 65,536 |
| Max numb. registers/thread | 63 | 63 | 63 | 255 | 255 |
| Max numb. threads/block | 1024 | 1024 | 1024 | 1024 | 1024 |
| Shared memory config. | 16K | 16K | 16K | 16K | 16K |
| | 48K | 48K | 48K | 48K | 48K |
| | | | 32K | 32K | 32K |

In order to declare a shared memory variable in a kernel or device function, we use the qualifier `__shared__`. Hence, for example, the following code can be used to declare a shared memory vector of 32 `float` values in a kernel code:

```
__shared__ float vector[32];
```

- **Global memory**: Variables in global memory can be written and read by the host by calling API functions as we will see in Sect. 5.4. These variables are located off the processor chip, into the so-called device DRAM memory (as the local memory variables). Since the DRAM technology is used to implement this memory, the speed access is much lower than in on-chip memories (registers and shared memory).

  The lifetime of a variable resident in global memory is the duration of the entire CUDA application (it is shared and available for all kernels in the application).

  The `__device__` qualifier is used to declare device variables which are resident in global memory. These variables are accessible from all threads through the entire CUDA application.

```
__device__ float Global_A[32];
```

Global memory variables can be used to implement the thread block collaboration in a kernel call, but the main goal of these variables is providing a way to communicate data between different CUDA grids associated to different kernel calls (see Fig. 12).

We can define pointers to global memory data in CUDA. Thus, the programmer can declare a pointer variable in a host function (for example, in the `main` function) and then to use the function `cudaMalloc` to allocate global device memory for that pointer. For example, to declare and allocate a pointer to a vector of 1024 integers in global memory, we can write:

```
int numbytes = 1024*sizeof(int);
int *a_d;
cudaMalloc( (void**)&a_d, numbytes );
```

This pointer can be passed to a kernel function as a parameter. For example, the parameters `A`, `B` and `C` of the kernels shown in Listings 1, 2 and 3 are samples of this class of pointers. Listing 4 shows the declaration, allocation, usage in a kernel and freeing of several pointers to global memory data.

```
__global__ void VecAdd(float* A, float* B, float* C,int N){ ...
     }
...
int main()
{ int N = ...; int size = N * sizeof(float);
float* h_A = (float*) malloc(size); float* h_B = (float*)
    malloc(size);
float* h_C = (float*) malloc(size);
Initialize(h_A, h_B, N);
float* d_A; float* d_B;float* d_C; cudaMalloc((void**)&d_C,
    size);
cudaMalloc((void**)&d_A, size); cudaMalloc((void**)&d_B, size);

// Step 1
cudaMemcpy (d_A, h_A, size, cudaMemcpyHostToDevice);
cudaMemcpy (d_B, h_B, size, cudaMemcpyHostToDevice);

// Step 2
VecAdd <<<(ceil((float) N/BLOCKSIZE), BLOCKSIZE>>> (d_A, d_B,
    d_C, N);

// Step 3
cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);

cudaFree(d_A);cudaFree(d_B); cudaFree(d_C);}
```

**Listing 4** CUDA program with host-device data transfers

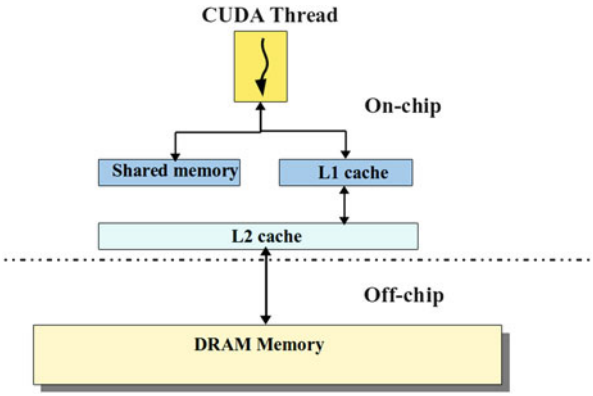


**Fig. 13** Memory hierarchy in modern CUDA-enabled GPUs

- **Constant memory**: The constant variables are global variable declarations which should be outside any function in the source file. The `__constant__` qualifier is used to declare device variables in constant memory. The following code declares a constant vector of 64 `float` values:

```
__constant__ float A[64];
```

These variables are accessible from all threads in a CUDA application but only to be read. They last for the entire CUDA application. These variables are also stored in the device DRAM memory but they are cached (using a small cache memory) and this makes it possible to obtain high speed access if the patterns to read the data are suitable.

These variables are used to provide input values (they will not change during the execution) to a kernel function.

It is important to note that in modern CUDA-enabled GPUs, there exists an on-chip space memory that can be partially dedicated to cache memory for each kernel call (see Fig. 13). This cache memory might reduce the global memory access cost and avoids the explicit control of the programmer [17]. Moreover, the programmer can specify the amount of memory which is allocated to L1 cache and shared memory.

## *4.6 CUDA Thread Scheduling*

Each active block resulting from a kernel launch is divided into warps to be executed on the assigned multiprocessor. A *warp* is a group of 32 threads (see Table 1) with consecutive indices (the thread ordering depends on `ThreadIdx`) that run physically in parallel on a multiprocessor. All threads in a warp execute the same

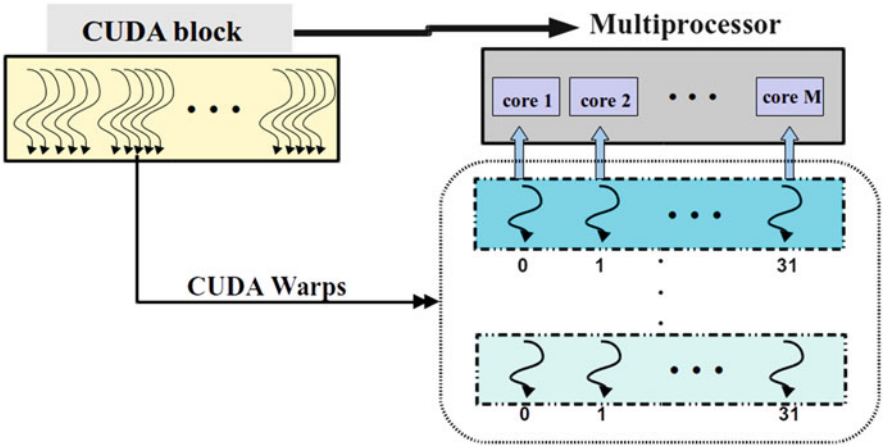


**Fig. 14** Execution of a thread block by generating several warps

instruction of the kernel simultaneously, following the *Single Instruction Multiple Data* (SIMD) execution model (Fig. 14).

When a block is assigned to a multiprocessor of the GPU, the block is split into warps and periodically a scheduler of the multiprocessor switches from one warp to another (without penalties). This allows to hide the high latency when accessing the GPU device memory, since some threads can continue their execution while other threads are waiting. Therefore, the definition of a high number of threads when launching a kernel improves the efficiency in the kernel execution because makes it possible to hide the latency.

The decomposition of a thread block into warps is based on thread indices (the `threadIdx` field values). When only `threadIdx.x` is used (1D blocks), the threads of a warp and its ordering are imposed by the values of `threadIdx.x`. So, for example, for the CUDA program of Listing 2 (the block size is 256 threads), the arrangement of the threads would be ($T_i$ denotes the thread with `threadIdx.x` $= i$):

$$
\begin{array}{ll}
\text{First warp:} & T_0, \ldots\ldots\ldots, T_{31}. \\
\text{Second warp:} & T_{32}, \ldots\ldots\ldots, T_{63}. \\
\ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots\ldots \\
\ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots\ldots \\
\text{Last warp (8th):} & T_{224}, \ldots\ldots\ldots, T_{255}.
\end{array}
$$

When the block size is not a multiple of 32, the last warp will be completed with additional threads. For that reason it is usual to define a block size that is a multiple of 32.

For a 2D or 3D block, the dimensions are mapped to a linear 1D arrangement in order to decompose the block into warps. This linear order is obtained by advancing forward firstly the $x$-dimension, secondly the $y$-dimension and finally the $z$-dimension. For instance, in Listing 3, where 2D $16 \times 16$ blocks are used, we would have the following arrangement ($T_{i,j}$ denotes the thread with `threadIdx.x` $= i$ and `threadIdx.y` $= j$):

First warp: $T_{0,0},\ T_{1,0}, \ldots\ldots, T_{15,0},\ T_{0,1},\ T_{1,1}, \ldots\ldots, T_{15,1}$.

Second warp: $T_{0,2},\ T_{1,2}, \ldots\ldots, T_{15,2},\ T_{0,3},\ T_{1,3}, \ldots\ldots, T_{15,3}$.

$\ldots\ldots\ldots\ldots$ $\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$

$\ldots\ldots\ldots\ldots$ $\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$

Last warp (8th): $T_{0,14},\ T_{1,14}, \ldots\ldots, T_{15,14},\ T_{0,15},\ T_{1,15}, \ldots\ldots, T_{15,15}$.

We can check that the mapping of threads to elements of the input and output matrices in Listing 3 implies that the threads of a warp access to matrix elements with are located contiguously in the global memory. This way of mapping threads to data is very important to obtain a good performance because makes it possible to take advantage of the on-chip cache for the global memory space which is available in modern GPUs.

## 4.7 Block Thread Synchronization

Although there are no synchronization constraints between the CUDA blocks of a kernel, CUDA provides a mechanism to coordinate the execution of the threads in the same block, provoking a barrier-type synchronization. The following function

```
void __syncthreads();
```

can be used to synchronize all the threads in the same block of a kernel launch. It establishes a synchronization barrier for all the threads in a block. When this function is called in a particular point of a kernel function, all threads in each generated block will wait until the rest of threads also reaches the same execution point in the kernel.

It is used to prevent the threads from accessing variables before these have been written by other threads, as will be seen later. This function should only be called in a conditional code when the condition has the same evaluation for all threads in the block.

## 5 CUDA C/C++ Programming Interface

CUDA C/C++ is a minimal extension of C to define kernels as C++ functions. In order to program with a high abstraction level in CUDA C, one must use the CUDA runtime API [19], which provides functions to perform the following tasks:

- To query the properties of multiple devices.
- To allocate and free device memory.
- To transfer data between host memory and device memory.

The CUDA Runtime API requires the usage of the compiler driver **nvcc**.

It is also possible to develop CUDA programs using the CUDA driver API [18] but it is harder to program with this interface.

### *5.1 Querying GPU Properties*

The CUDA Runtime API provides functions to find out the available resources and capabilities of the underlying GPU hardware. The CUDA runtime system numbers all the available GPUs in the system from 0 to `num_GPUs` − 1.

The `cudaGetDeviceCount` function makes it possible to know the number of available CUDA GPUs in a particular computer system. Thus, for instance, to obtain the number of GPUs into an integer variable, we write:

```
int num_GPUs;
cudaGetDeviceCount ( &num_GPUs );
```

In order to know the characteristics of a particular GPU, CUDA Runtime API provides the built-in type `cudaDeviceProp` which is a C structure with fields representing different properties of a CUDA GPU. Using this type, the function `cudaGetDeviceProperties` can be used to query the characteristics of a GPU in the system. For example, this code piece:

```
cudaDeviceProp GPU_prop; int dev_id=...;
cudaGetDeviceProperties ( &GPU_prop, dev_id);
```

returns the properties of the GPU which has the number `dev_id` in the `GPU_prop` variable. Some of the properties we can query in `GPU_prop` are:

- `GPU_prop.totalGlobalMem`: Global memory available on device in bytes.
- `GPU_prop.sharedMemPerMultiprocessor`: Shared memory available per multiprocessor in bytes.
- `GPU_prop.maxThreadsPerBlock`: maximal number of threads allowed in a block (512, 1024,...).

- `GPU_prop.multiProcessorCount`: number of multiprocessors.
- `GPU_prop.maxThreadsDim[dim_id]`: maximal number of threads allowed along dimension `dim_id` of a block.
- `dev_prop.maxGridSize [dim_id]`: maximal number of blocks allowed along dimension `dim_id` of a grid.
- `dev_prop.warpSize`: Number of threads in a warp (unit of thread scheduling in multiprocessors).

The *CUDA Compute Capability* (CCC) indicates the architecture and features of the particular GPU. It is defined by a major revision number and a minor revision number `X.X`. Devices with the same major revision number denote the same core architecture. Table 1 shows some features of modern GPU architectures (Fermi, Kepler and first generation of Maxwell architecture) with different CCC.

## 5.2 Reporting Errors

All CUDA API calls return an error code (with built-in type `cudaError_t`). This error can be caused by:

- Error in the API call itself. For example:

```
cudaError_t err; err=cudaGetDevice(&devID);
if (err!=cudaSuccess) {cout<<"ERROR!!"<<endl;}
```

- Error in an earlier asynchronous operation (for example in a kernel call):

```
kernel<<<blocksPerGrid,threadsPerBlock >>>(a,b,c);
err = cudaGetLastError();
if (err != cudaSuccess) {...}
```

We can get a string which describes the particular error by using the following code piece:

```
char *cudaGetErrorString(cudaError_t)
printf("%s\n", cudaGetErrorString(cudaGetLastError()));
```

## 5.3 Compiling CUDA Code

The CUDA compilation includes two stages [22]:

- *Independent on GPU* (Virtual): It generates a code written in a lower level intermediate language called PTX code (Parallel Thread eXecution) [23].
- *Physical*: It generates object code for a particular GPU.

The CUDA Toolkit includes a compilation tool called `nvcc`, which is the compiler driver. It performs the calls to the required compilers and tools: cudac, g++, etc. This tool decouples CPU and GPU code. The CPU C code must be compiled by another tool. PTX object code must be adapted to a particular GPU architecture and CUDA Capability.

In order to execute code on devices of an specific CUDA Compute Capability, it is necessary to load code that is compatible with this CCC. Which PTX and binary code gets embedded in a CUDA C application is controlled by the `-arch` and `-code` compiler options or, more concisely, by using the -gencode compiler option.

- `-arch=<compute_xy>`: it generates PTX code for capability `x.y`.
- `-code=<sm\_xy>`: it generate binary code for capability `x.y`, by default same as `-arch`.

*Example*

```
nvcc filename.cu -gencode arch=compute_35,
                  code=sm_35 -o filename
```

This command generates binary and PTX code compatible with CCC 3.5.

Host code is generated to automatically select at runtime the most appropriate code.

In order to ensure 64-bit or 32-bit compatibility, we can use the following switches:

- `m64`: compile device code in 64-bit mode.
- `m32`: compile device code in 32-bit mode.

## 5.4 Device Memory Management

CPU and GPU have separated memory spaces. To enable the data management and the communication, there are runtime functions to (Fig. 15):

- Allocate and free DRAM device global memory.
- Set a value in several positions of device global memory.
- Transfer data between global (device) memory and host memory.

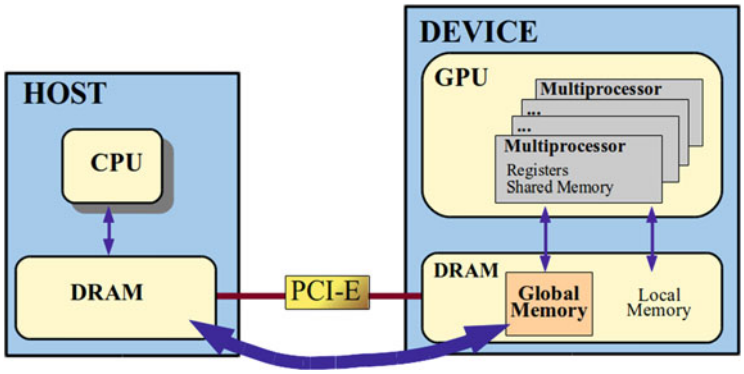


**Fig. 15** Host-device memory transfer

We can use the function `cudaMalloc` to allocate memory device with the following syntax:

```
cudaMalloc(void ** pointer, size_t numbytes)
```

The `size_t` type is an unsigned integer type used to represent the sizes of data objects as a number of bytes.

We use `cudaFree` to free memory device:

```
cudaFree(void* pointer)
```

The function `cudaMemset` makes it possible to set a particular `value` in linear device memory:

```
cudaMemset(void * pointer, int value, size_t nbytes)
```

*Example* Reset all elements of a vector with 1024 integers

```
int numbytes = 1024*sizeof(int);
int *a_d;
cudaMalloc( (void**)&a_d, numbytes );
cudaMemset( a_d, 0, nbytes);
    ...
    ...
cudaFree(a_d); // Free a_d when it is not needed
```

The function `CudaMemcpy` is used to transfer data between global memory and host memory:

```
CudaMemcpy (void *dest, void *source, size_t nbytes,
                    enum cudaMemcpyKind direction);
```

This function copies `nbytes` bytes from the memory area pointed to by `source` to the memory area pointed to by `dest`, where the location (host or device) for `dest` and `source` is given by `direction`:

- `cudaMemcpyHostToDevice`: From host to device.
- `cudaMemcpyDeviceToHost`: From device to host.
- `cudaMemcpyDeviceToDevice`: Between two global memory positions in device.

The invocation of this function blocks the CPU thread and only returns when the data copy has been completed. The data transfer does not begin if any previous CUDA call have not returned.

The program in Listing 4 shows the CUDA calls which are necessary in order to execute the vector addition kernel. After initializing vectors *A* and *B* (stored in a memory area pointed to by `h_A` and `h_B`), the area for the input and output vectors in device memory is allocated using `cudaMalloc` (pointers `d_A`, `d_B` and `d_C`). Then input vectors are transferred to the memory area pointed to by `d_A` and `d_B` before the kernel launch. After the kernel execution, the resultant value of *C*, obtained in device global memory (pointed to by `d_C`), must be copied to the host memory. Finally, the device memory area pointed to by `d_A`, `d_B` and `d_C` is freed. Figure 16 illustrates the main steps of this program.

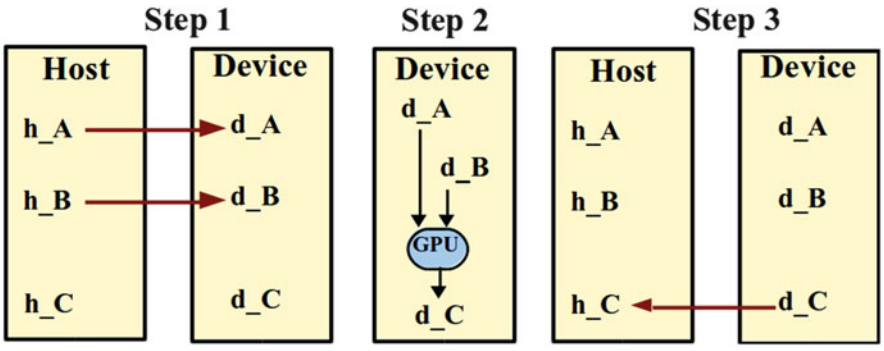


**Fig. 16** Description of data transfers and kernel launch in a CUDA program

### 5.5 CUDA Samples

To assist in the development of CUDA software, the CUDA Samples [20] provide several examples with source code which can be useful to build a wide range of interesting applications.

## 6 Solving the 1D Linear Advection Equation in CUDA

### 6.1 1D Linear Advection

The 1D advection equation [15] is the simplest hyperbolic equation and can be written as:

$$\frac{\partial \phi}{\partial t} + u\frac{\partial \phi}{\partial x} = 0, \tag{1}$$

where $\phi = \phi(x,t)$ is a real function with $x \in [a,b]$ and $t > 0$. The scalar $u \in \mathbb{R}$ represents the speed at which information propagates along the $x$-direction and is assumed to be constant. The initial state for this equation is specified as

$$\phi(x,0) = \phi_0(x),$$

and we assume periodic boundary conditions.

We can verify that the exact solution for this equation is given by

$$\phi(x,t) = \phi_0(x-ut).$$

### 6.2 The Lax-Friedrichs Method

We discretize the spatial and time domain using uniform grids which consist of:

- $n+1$ spatial grid points:

$$x_i, \quad i = 0,\ldots,n, \text{ where } x_i = a + i\Delta x, \qquad \text{with } \Delta x = \frac{b-a}{n}.$$

- $M$ time steps:

$$t^1, t^2, \ldots, t^M \text{ with a constant } \Delta t, \quad (t^{k+1} = t^k + \Delta t).$$

Now, we can define the *Lax-Friedrichs* finite difference scheme to approximate the solution of equation (1) as follows (see [15]):

$$\phi_i^{k+1} = \frac{1}{2}\left(\phi_{i-1}^k + \phi_{i+1}^k - C(\phi_{i-1}^k - \phi_{i+1}^k)\right), \tag{2}$$

$$C = \frac{u\,\Delta t}{\Delta x}, \quad i = 0, \ldots, n, \quad k = 1, \ldots, M.$$

In the previous scheme $\phi(x_i, t^k)$ denotes $\phi_i^k$.

The boundary conditions are imposed by defining:

$$\phi_{-1}^k = \phi_n^k, \qquad \phi_{n+1}^k = \phi_0^k.$$

This method is linearly stable and convergent under the usual CFL condition provided $|C| \leq 1$ (see [15]).

### 6.3 A Sequential C Program to Implement the Method

Listing 5 shows a sequential C program to approximate the 1D linear advection equation using the Lax-Friedrichs scheme on a uniform grid. As we can see, at each time step, each data element of the output vector `phi_new` is computed from two neighbouring input elements of vector `phi`. Figure 17 shows graphically the data dependencies between the input and output vectors at each time step. Vector `phi` includes elements at the edges to store the ghost values needed to perform the stencil computation.

After each state vector update the pointers, `phi` and `phi_new`, must be swapped (using the `swap_pointers` function), because the state vector computed in a time step will be the previous state vector in the next time step.

### 6.4 A CUDA Linear Advection Solver

Since the calculation of all output elements of `phi_new` for a time step can be done in parallel (see Listing 5), we can assign a different thread to the computation of each element of `phi_new` at each time step. We will need to use at least $n + 1$ threads and the body of each thread is given by a kernel function with four arguments:

- A pointer to array, `d_phi`, denoting the initial state vector ($\phi^k$ in Eq. (2)).
- A pointer to array, `d_phi_new`, denoting the new state vector $\phi^{k+1}$.
- An integer `cu` representing the value of the Courant number (`cu` $= \frac{u\,\Delta t}{\Delta x}$).
- The integer value `n`, which denotes the value of $n$ in Eq. (2).

The definition of this kernel function is given in Listing 6.

```
...
void swap_pointers(float * *a, float * *b)
  {float * tmp=*a;*a=*b;*b=tmp;}
...
int main(int argc, char* argv[]){
   // Define Delta x, Delta t and velocity (u)
  int n= ... ;float dx = ...;float dt=..; float u=...
  // Compute the Courant number
  float cu = u*dt/dx;
  //Declare the initial and final state vectors
  float * phi =new float[n+3]; float * phi_new=new float[n+3];
  ..
  Initialize(phi); // Set phi values at t=0
  // Time steps
  for(int k=1; k <= M; k++)
   {// Impose Homogeneous Neumann Boundary Conditions
    phi[index(-1)] =phi[index(n)];
    phi[index(n+1)]=phi[index(0)];

    //Lax-Friedrichs Update
    for(int i=0;i<=n;i++)
         phi_new[index(i)]=0.5*((phi[index(i+1)]+phi[index(i-1)])
                        -cu*(phi[index(i+1)]-phi[index(i-1)]));

    swap_pointers (&phi,&phi_new);
   }
...}
```

**Listing 5** Sequential C program which implements the Lax-Friedrichs scheme

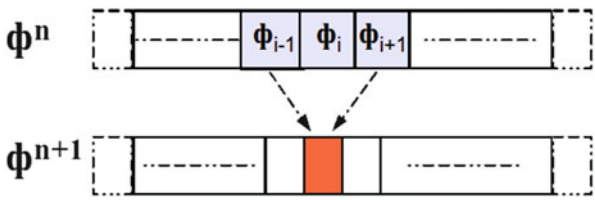


**Fig. 17** Two-point stencil of the Lax-Friedrichs method

As can be seen in Listing 6, we follow a similar approach as the vector addition kernel (see Listing 2). All threads read two elements of the same input vector except for two threads which read an additional element to impose the boundary conditions. The work of threads which are not assigned to elements of `d_phi_new` is avoided (threads verifying that $i \geq n+2$). The same threads which have computed the values close to the edges of `d_phi_new` (positions `1` and `n+1`) impose the boundary conditions by assigning values on the edges of `d_phi_new`.

```
__global__ void FD_kernel1(float * d_phi,
                           float * d_phi_new,
                                float cu, int n)

{int i=threadIdx.x+blockDim.x*blockIdx.x+1;

// Lax-Friedichs Stencil
if (i<n+2)
  d_phi_new[i]=0.5*((d_phi[i+1]+d_phi[i-1])
                         -cu*(d_phi[i+1]-d_phi[i-1]));

// Impose Boundary Conditions
if (i==1) d_phi_new[0]=d_phi_new[n+1];
if (i==n+1) d_phi_new[n+2]=d_phi_new[1];}
```

**Listing 6** CUDA kernel to apply the Lax-Friedrichs method stencil

Listing 7 shows a CUDA C program to implement the Lax-Friedrichs scheme using the previously introduced kernel (`FD_kernel1`). Before launching the kernel, the memory of the input and output state vectors (`d_phi` and `d_phi_new`) must be allocated on the device using calls to `cudaMalloc()`. Using `cudaMemcpy()`, the initial values of the state vector (computed by calling a initialization function) are copied to the device memory. Then, at each time step, we launch the `FD_kernel` from the host code by using blocks of 256 threads. After each kernel launch the memory device pointers, `d_phi` and `d_phi_new`, must be swapped. Finally, when the time integration loop has finished, the final state vector is copied from device memory to the host memory.

## 6.5 A Tiled CUDA Advection Solver

In the kernel of Listing 6, the ratio of floating-point calculations regarding with the global memory accesses is low. This prevents us from achieving high performance. One approach that could improve the performance consists of managing the shared memory which is much faster than global memory. This involves to replace global memory accesses with shared memory accesses (into each block). This approach frequently involves a reorganization of the code to enable the reuse of data which are placed in shared memory.

The typical approach to use shared memory in order to reduce the traffic to global memory consists of two steps:

1. The input data elements are organized in small pieces, called *tiles*.
2. Threads of a thread block collaborate to load a tile into shared memory before the threads use these elements.

```
#define BLOCKSIZE 256

... //FD_kernel function definition

int main(int argc, char* argv[]){
  int size=(n+3)*sizeof(float);
  float * d_phi=NULL; float * d_phi_new=NULL;
  cudaMalloc((void **) &d_phi, size);
  cudaMalloc((void **) &d_phi_new, size);

  Initialize(phi); // Set phi values at t=0

  cudaMemcpy(d_phi,phi,size,cudaMemcpyHostToDevice);
  // Time Steps
  for(int k=1; k <= M ;k++)
   {

    int blocksPerGrid =(int) ceil((float)(n+1)/BLOCKSIZE);

    // ********* CUDA Kernel Launch
        ***********************************
    FD_kernel1<<<blocksPerGrid, BLOCKSIZE >>> (d_phi, d_phi_new,
         cu, n);

    swap_pointers (&d_phi,&d_phi_new);
   }

  cudaMemcpy(phi_GPU, d_phi, size, cudaMemcpyDeviceToHost);
  ...
}
```

**Listing 7** CUDA program to implement the Lax-Friedrichs method

3. Once the threads of the same block have loaded its corresponding tile, they re-use the elements by accessing the shared memory.

In these so-called *tiled algorithms* [13], the size of the tiles must be chosen to fit into the shared memory (which is quite small).

In order to apply this approach to the Lax-Friedrichs kernel of Listing 6, we can load all input elements of `d_phi`, which are needed to compute all output elements of `d_phi_new` for a thread block, into the shared memory. The number of elements to be loaded must be equal to `BLOCKSIZE` + 2 because we need two additional elements at the edges to compute `BLOCKSIZE` elements of `d_phi_new`. Listing 8 shows the tiled kernel which implements this computation.

```
__global__ void FD_kernel2 (float * d_phi, float * d_phi_new,
                            float cu, const int n)
 {
  int li=threadIdx.x+1; //local index in shared memory vector
  int gi= blockDim.x*blockIdx.x+threadIdx.x+1; // global
      memory index
  int lstart=0; // start index in the block (left value)
  int lend=BLOCKSIZE+1; // end index in the block (right value
      )

  __shared__ float s_phi[BLOCKSIZE + 2]; //shared mem. vector

  float result;

  // Load internal points of the tile in shared memory
  if (gi<n+2) s_phi[li] = d_phi[gi];

  // Load the halo points of the tile in shared memory
  if (threadIdx.x == 0) // First Thread (in the current block)
    s_phi[lstart]=d_phi[gi-1];

  if (threadIdx.x == BLOCKSIZE-1) // Last Thread
       if (gi>=n+1) // Last Block
         s_phi[(n+2)%BLOCKSIZE]=d_phi[n+2];
       else
         s_phi[lend]=d_phi[gi+1];


  __syncthreads();

 if (gi<n+2)
  {
   // Lax-Friedrichs Update
   result=0.5*((s_phi[li+1]+s_phi[li-1])
      -cu*(s_phi[li+1]-s_phi[li-1]));
   d_phi_new[gi]=result;
  }


 // Impose Boundary Conditions
 if (gi==1) d_phi_new[0]=d_phi_new[n+1];
 if (gi==n+1) d_phi_new[n+2]=d_phi_new[1];

 }
```

**Listing 8** Lax-Friedrichs CUDA kernel using shared memory

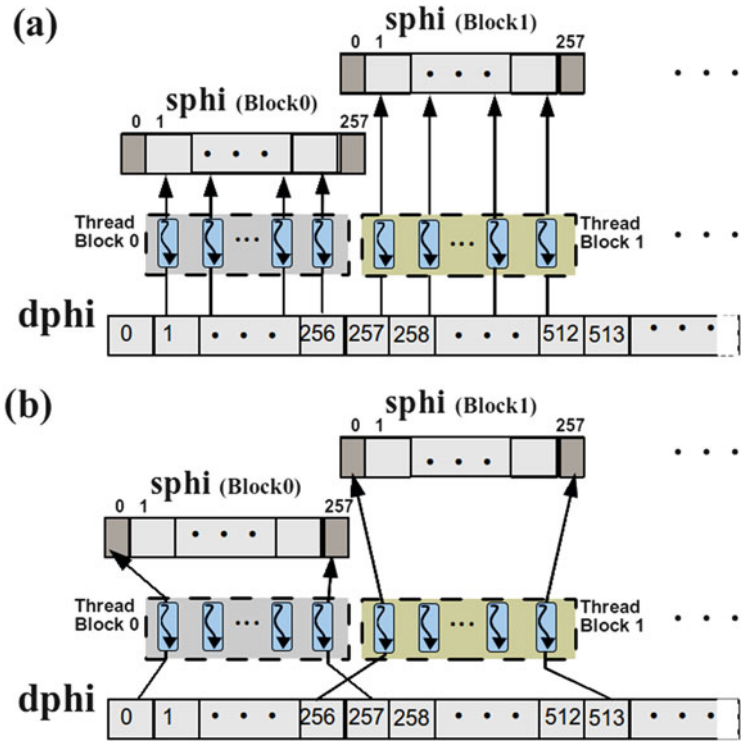


**Fig. 18** Loading the tile in shared memory: (**a**) Loading internal points; (**b**) Loading the halo points

Initially, a shared memory array with `BLOCKSIZE` + 2 elements, `s_phi`, is declared to hold all elements to be loaded in order to compute the corresponding `BLOCKSIZE` elements of `d_phi_new`. Then the tile is loaded in shared memory in two phases (see Fig. 18):

(a) All threads in the block load in parallel the internal elements of the input tile (without taking into account the elements on the edges).
(b) The first and last threads in the block load the elements at the edges of the tile (the halo elements).

After loading the tile in shared memory, it is necessary to make sure that all threads in a block have finalized the load before using the tile to compute the output data elements. This is done by invoking the function `__syncthreads()`.

When the tile is completely loaded for the particular thread block, we can compute the output `BLOCKSIZE` elements of `d_phi_new` reading only elements of `s_phi`.

Finally, the boundary conditions are imposed in a similar way as in Listing 6.

**Acknowledgements** This work has been partially supported by FEDER and the Spanish and Andalusian research projects MTM2014-52056-P, MTM2012-38383-C02-01, P11-FQM8179 and P11-RNM7069.

## References

1. de la Asunción, M., Castro, M.J., Fernández-Nieto, E.D., Mantas, J.M., Ortega, S., González, J.M.: Efficient GPU implementation of a two waves TVD-WAF method for the two-dimensional one layer shallow water system on structured meshes. Comput. Fluids **80**, 441–452 (2012)
2. de la Asunción, M., Mantas, J.M., Castro, M.J.: Programming CUDA-based GPUs to simulate two-layer shallow water flows. In: D'ambra, P., Guarracino, M., Talia, D. (eds.) Euro-Par 2010, Ischia. Lecture Notes in Computer Science, vol. 6272, pp. 353–364. Springer (2010)
3. de la Asunción, M., Mantas, J.M., Castro, M.J.: Simulation of one-layer shallow water systems on multicore and CUDA architectures. J. Supercomput. **58**(2), 206–214 (2011)
4. Bell, N., Hoberock, J.: Thrust: a productivity-oriented library for CUDA. In: GPU Computing Gems Jade Edition, pp. 359–371. Morgan Kaufmann, Waltham (2011)
5. Brodtkorb, A.R., Hagen, T.R., SæTra, M.L.: Graphics processing unit (GPU) programming strategies and trends in GPU computing. J. Parallel Distrib. Comput. **73**(1), 4–13 (2013)
6. Castro, M.J., Ortega, S., de la Asunción, M., Mantas, J.M., Gallardo, J.M.: GPU computing for shallow water flow simulation based on finite volume schemes. Comptes Rendus Mécanique **339**(2–3), 165–184 (2011)
7. Fang, J., Varbanescu, A., Sips, H.: A comprehensive performance comparison of cuda and opencl. In: 2011 International Conference on Parallel Processing (ICPP 2011), Taipei, pp. 216–225 (2011)
8. Fernando, R.: GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics. Addison-Wesley, Boston (2004)
9. Fernando, R., Kilgard, M.J.: The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics. Addison-Wesley, Boston (2003)
10. Hubert, N.: GPU Gems 3: Programming Techniques for High-Performance Graphics and General-Purpose Computation. Addison-Wesley Professional, Boston (2007)
11. Jackson, M., Budruk, R., Winkles, J., Anderson, D.: PCI Express Technology 3.0. MindShare Press, Monument (2012)
12. Khronos OpenCL Working Group: The OpenCL Specification. https://www.khronos.org/opencl/ (2015)
13. Kirk, D., Wen-mei, H.: Programming Massively Parallel Processors. A Hands-on Approach, 2nd edn. Morgan Kaufmann, Waltham (2012)
14. Lastra, M., Mantas, J.M., Ureña, C., Castro, M.J., García, J.A.: Simulation of shallow-water systems using graphics processing units. Math. Comput. Simul. **80**(3), 598–618 (2009)
15. Leveque, R.: Finite Difference Methods for Ordinary and Partial Differential Equations. SIAM, Philadelphia (2007)
16. Matt, P., Randima, F.: GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation. Addison-Wesley Professional, Upper Saddle River (2005)
17. NVIDIA: CUDA C Best Practices Guide. http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html (2014)
18. NVIDIA: Cuda driver api. http://docs.nvidia.com/cuda/cuda-driver-api/index.html (2014)
19. NVIDIA: CUDA Runtime API. http://docs.nvidia.com/cuda/cuda-runtime-api/index.html (2014)
20. NVIDIA: CUDA Samples. http://docs.nvidia.com/cuda/cuda-samples (2014)
21. NVIDIA: CUDA Zone. http://www.nvidia.com/object/cuda_home.html (2014)

22. NVIDIA: NVIDIA CUDA Compiler Driver NVCC. http://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html (2014)
23. NVIDIA: Parallel tread execution isa 3.2. http://docs.nvidia.com/cuda/pdf/ptx_isa_3.2.pdf (2014)
24. OpenACC-Standard.org: The OpenACC Application Programming Interface, Version 1.0. http://www.openacc.org/sites/default/files/OpenACC.1.0_0.pdf (2011)
25. Owens, J.D., Houston, M., Luebke, D., Green, S., Stone, J.E., Phillips, J.C.: GPU computing. Proc. IEEE **96**(5), 879–899 (2008)
26. Rumpf, M., Strzodka, R.: Graphics processor units: new prospects for parallel computing. In: Bruaset, A.M., Tveito, A. (eds.) Numerical Solution of Partial Differential Equations on Parallel Computers. Lecture Notes in Computational Science and Engineering, vol. 51, pp. 89–134. Springer, Berlin (2006)
27. Shreiner, D., Woo, M., Neider, J., Davis, T.: OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2.1. Addison-Wesley Professional, Upper Saddle River (2007)
28. Ujaldon, M.: High performance computing and simulations on the GPU using CUDA. In: 2012 International Conference on High Performance Computing & Simulation (HPCS 2012), Madrid, pp. 1–7. Curran Associates (2012)