Computing 53, 277–287 (1994)

Computing

PROFIL/BIAS—A Fast Interval Library

O. Knüppel, Hamburg

Received November 15, 1993; revised March 31, 1994

Abstract — Zusammenfassung

PROFIL/BIAS—A Fast Interval Library. The interval data type is currently not supported in common programming languages. Therefore the implementation of algorithms using interval arithmetic requires special programming environments or at least special libraries. In this paper we present the C++ class library PROFIL which provides a user friendly environment for implementing interval algorithms. The main goals in the design of PROFIL were speed and portability. Therefore all interval operations in PROFIL use BIAS (Basic Interval Arithmetic Subroutines) [16]. BIAS defines a concise and portable interface for the basic scalar, vector, and matrix operations. The interface is independent of a specific interval representation or computation but permits machine specific and fast implementations. Based on this general specification we present an implementation in C using a lower/upper bound representation of intervals and directed roundings. By using few assembler instructions for switching the rounding modes and avoiding sign tests and rounding mode switches wherever possible, the computational costs of the interval operations were reduced significantly. This is especially important for RISC machines, where floating point instructions can be executed in few machine cycles. Comparisons with other interval arithmetic packages show an improvement in speed of about one order of magnitude.

AMS Subject Classification: 65G10

Key words: Interval library, IEEE-754 arithmetic standard, C, C++.

PROFIL/BIAS—Eine schnelle Intervallbibliothek. In den weit verbreiteten Programmiersprachen wird der Intervalldatentyp nicht unterstützt. Daher benötigt man für die Implementierung von Algorithmen, die auf Intervallarithmetik basieren, spezielle Programmierumgebungen oder zumindest spezielle Programmbibliotheken. In diesem Artikel stellen wir die C++-Klassenbibliothek PROFIL vor, die eine anwenderfreundliche Umgebung für die Implementierung von Intervallalgorithmen darstellt. Die Entwicklung von PROFIL wurde von den beiden Hauptzielen Geschwindigkeit und Portabilität geleitet. Daher basieren alle Intervalloperationen von PROFIL auf BIAS (Basic Interval Arithmetic Subroutines). BIAS definiert eine einheitliche und portable Schnittstelle für die grundlegenden Intervalloperationen von skalaren bis hin zu Matrixoperationen. Dabei ist die Schnittstelle unabhängig von einer speziellen Intervalldarstellung oder von speziellen Berechnungsmodi, erlaubt aber dennoch maschinenspezifische und schnelle Implementierungen. Basierend auf dieser allgemeinen Spezifikation stellen wir eine Implementierung in C vor, die eine Intervalldarstellung in der Form untere/obere Grenze sowie gerichtete Rundungen verwendet. Durch Verwendung eigener Assemblerroutinen (insgesamt nur ca. 10 Assemblerinstruktionen) zur Umschaltung der Rundung sowie durch weitestgehende Vermeidung überflüssiger Vorzeichentests und Rundungsumschaltungen wird der Aufwand für die Intervalloperationen drastisch reduziert. Dies ist insbesondere für RISC-Architekturen wichtig, auf denen Gleitkommaoperationen in wenigen Maschinentaktzyklen ausgeführt werden können. Vergleiche mit anderen Intervallpaketen zeigen eine Geschwindigkeitssteigerung um etwa eine Größenordnung.

1. Introduction

In a number of problem areas interval arithmetic is proved to be useful [1, 7, 8, 21]. By using interval arithmetic, guaranteed bounds for the solution are calculated

despite rounding errors occurring during the computation. In some cases interval methods are even faster than the traditional floating point approach [11, 12].

A real interval $\boldsymbol{a}$ is defined by

$$\boldsymbol{a} := [\underline{a}, \overline{a}] := \{a \in \mathbb{R} | \underline{a} \leq a \leq \overline{a}\}, \qquad \underline{a}, \overline{a} \in \mathbb{R} \tag{1}$$

with $\underline{a} \leq \overline{a}$. The values $\underline{a}$ and $\overline{a}$ are called lower and upper bound, resp. We denote the set of real intervals by $\mathbb{IR}$. The interval operations on $\mathbb{IR}$ with $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{IR}$ are defined by:

$$\boldsymbol{a} * \boldsymbol{b} := \{a * b | a \in \boldsymbol{a}, b \in \boldsymbol{b}\}, \qquad * \in \{+, -, \cdot, /\} \tag{2}$$

with $\boldsymbol{a} * \boldsymbol{b} \in \mathbb{IR}$. It is assumed that $0 \notin \boldsymbol{b}$ in the case of division. The interval operations can easily be realized. For example, the subtraction of two intervals can be calculated by using the identity

$$\boldsymbol{a} - \boldsymbol{b} := [\underline{a} - \overline{b}, \overline{a} - \underline{b}]. \tag{3}$$

The interval operations can be extended to interval vectors and interval matrices by replacing the real operations by the appropriate interval operations. For example, the components of the product of a real matrix $R \in \mathbb{R}^{n \times p}$ and an interval matrix $\boldsymbol{A} \in \mathbb{IR}^{p \times m}$ are defined by

$$(R \cdot \boldsymbol{A})_{ij} := \sum_{k=1}^{p} R_{ik} \cdot \boldsymbol{A}_{kj}. \tag{4}$$

This matrix product can actually be computed in different ways as we will see later.

The key property of interval arithmetic is the isotonicity

$$a \in \boldsymbol{a}, b \in \boldsymbol{b} \quad \Rightarrow \quad a * b \in \boldsymbol{a} * \boldsymbol{b}, \qquad \boldsymbol{a}, \boldsymbol{b} \in \mathbb{IR}, \qquad * \in \{+, -, \cdot, /\} \tag{5}$$

which extends to interval vector, interval matrix operations and to interval standard functions as well. With preserved isotonicity, all interval calculations deliver guaranteed bounds. For a thorough discussion of interval arithmetic and its properties see for example [1].

When implementing the interval operations on a computer, the interval bounds $\underline{a}$ and $\overline{a}$ are floating point numbers, whereas $\boldsymbol{a} = [\underline{a}, \overline{a}]$ represents the set of all real numbers between $\underline{a}$ and $\overline{a}$.

In order to preserve the isotonicity we have to use floating point arithmetic with directed rounding. For this purpose the IEEE 754 arithmetic standard [9] may be used, which is implemented in a number of computers, e.g. in PCs and most workstations by means of a coprocessor. The IEEE 754 standard provides directed rounding modes denoted by ∇ for the downward and Δ for the upward rounding mode. In the ∇-mode resp. Δ-mode, a floating point operation $* \in \{+, -, \cdot, /\}$ is denoted by $\triangledown$ resp. $\triangle$. Therefore, if $\mathbb{F}$ denotes the set of floating point numbers and $a, b \in \mathbb{F}$, then

$$a \triangledown b \leq a * b \leq a \triangle b.$$

Thus the interval subtraction (3) for floating point bounds computed by

$$\boldsymbol{a} - \boldsymbol{b} := [\underline{a} \nabla \overline{b}, \overline{a} \triangle \underline{b}] \tag{6}$$

preserves the isotonicity. As another example we take the multiplication of a floating point value r with an interval $\boldsymbol{a}$. A sign test for r is needed in this case and the resulting interval is computed by

$$r \cdot \boldsymbol{a} := \begin{cases} [r \nabla \underline{a}, r \triangle \overline{a}] & \text{for } r \geq 0 \\ [r \nabla \overline{a}, r \nabla \underline{a}] & \text{otherwise} \end{cases} \tag{7}$$

Unfortunately, on most computers the evaluation of the transcendental functions is not affected by the setting of the rounding mode. To preserve the isotonicity for these operations, too, we use a faithful rounding [3, 23] which is based on the estimation of the rounding error. Let $\mathscr{S}: \mathbb{R} \to \mathbb{R}$ be a transcendental function and $\mathscr{S}_{\mathbb{F}}: \mathbb{F} \to \mathbb{F}$ the floating point equivalent as implemented on the computer. Then bounds for $\mathscr{S}(x)$, $x \in \mathbb{F}$ can be calculated by

$$|\mathscr{S}(x) - \mathscr{S}_{\mathbb{F}}(x)| \leq \varepsilon |\mathscr{S}_{\mathbb{F}}(x)| + \eta \tag{8}$$

where ε denotes the maximal relative error of the floating point function $\mathscr{S}_{\mathbb{F}}$, and η denotes a small absolute error which is only needed in case of arithmetic underflow.

A more detailed description of the tools of Interval Arithmetic can be found e.g. in [1].

2. BIAS—Basic Interval Arithmetic Subroutines

In numerical linear algebra the set of commonly used FORTRAN callable functions for Basic Linear Algebra Subroutines (BLAS) is very popular [4, 5, 20]. The idea of BLAS is to provide an interface for basic vector and matrix operations with specific and fast implementations on various machines, the latter frequently provided by the manufacturers. Along these lines we have presented a specification for Basic Interval Arithmetic Subroutines (BIAS) in [16]. The specification defines a concise interface for basic interval operations from scalar up to matrix operations as well as for all important auxiliary functions (midpoint, diameter, ...). For performance reasons, special routines for specific operand combinations (e.g. floating point/interval matrix multiplication) are also contained in the interface.

The interface is independent of the specific interval representation and computation. This is because only the interval data type `BIASINTERVAL` is referenced by the programs using BIAS. All access to the components of this data type is performed only through BIAS (e.g. access to the lower bound of the interval). Especially all programs based on BIAS need not to be modified and can be kept portable even if used on different hardware architectures.

In the following we are going to discuss a special implementation of BIAS in C using a lower/upper bound interval representation and directed roundings. The directed

rounding modes are available on the large class of machines conforming the IEEE 754-standard [9], especially PCs and most workstations.

The implementation of BIAS is written in ANSI C. No special machine dependencies are used except the 3 small routines

```
BiasRoundUp ()
BiasRoundDown ()
BiasRoundNear ()
```

for switching the rounding mode which are written in assembler (about 10 assembler statements totally). Together with very few informations (e.g. the byte order) these routines contain the machine dependent part of the BIAS library. For a number of common architectures the needed assembler routines are contained in the BIAS package. We have decided to use our own assembler routines instead of the ones provided by the vendor, because the latter differ from machine to machine and, in general, are slower by orders of magnitude than our routines.

C has been chosen as implementation language, because it is widely available and the most common target language for source converters (e.g. from Fortran, Modula-2, Pascal). In general, C compilers also produce good optimized code.

As an example of the BIAS implementation let us consider the subtraction of two intervals. The subroutine taken from BIAS is

```
VOID BiasSubII (const PBIASINTERVAL pR,
                const PBIASINTERVAL pA,
                const PBIASINTERVAL pB)
{
 BiasRoundDown ();
 pR->inf = pA->inf - pB->sup;

 BiasRoundUp ();
 pR->sup = pA->sup - pB->inf;

 SetRoundToNearest();
}
```

where `PBIASINTERVAL` denotes the type of a pointer to an interval, `pR` points to the resulting interval and `pA`, `pB` denote pointers to the operands of the subtraction. Optionally, the rounding mode can be set to nearest after any operation. This is defined at compile time by changing the definition of the macro `SetRoundToNearest`, which is either equal to `BiasRoundNear` or empty. In the latter case, the setting of the rounding mode is undefined after executing any routine from BIAS (in almost all cases, it is the upward rounding), but this option saves up to 10% computing time for the scalar operations.

For the scalar operations, there is not much room for improvements, therefore we concentrate on the more interesting vector and matrix operations.

2.1 *Vector and Matrix Operations*

In the last years the computing paradigms changed: In the past, the floating point operations were the most expensive ones (in terms of computing time). But this is no longer true for modern RISC architectures, where the computational costs of floating point operations, rounding mode switches, sign tests, and integer operations are nearly the same. For example on IBM RS/6000 architectures, a floating point multiply-and-add instruction can be executed in one machine cycle, whereas an already optimized sign test needs several machine cycles. Therefore sign tests and rounding mode switches cannot be neglected and must be avoided whereever possible in order to increase the performance.

As an example we consider the multiplication of an interval vector by a floating point scalar, i.e. $w = a \cdot v$, $a \in \mathbb{R}$, $v \in \mathbb{IR}^n$. The standard implementation for this operation would reduce the vector operation to a multiplication of a with each element of the vector v:

$$\text{for } i = 1, \ldots, n \text{ do } w_i = a \cdot v_i \tag{9}$$

This implementation needs n sign tests and $2n$ rounding mode switches. In the BIAS implementation, there is only 1 sign test and 2 rounding mode switches:

$$\begin{aligned}
&\text{if } a > 0 \text{ then}\\
&\quad \text{set rounding mode downwards}\\
&\quad \text{for } i = 1, \ldots, n \text{ do } \underline{w_i} = a \cdot \underline{v_i}\\
&\quad \text{set rounding mode upwards}\\
&\quad \text{for } i = 1, \ldots, n \text{ do } \overline{w_i} = a \cdot \overline{v_i}\\
&\text{else} \ldots
\end{aligned} \tag{10}$$

On IBM RS/6000 architectures, the standard implementation (9) needs about 5 times as long as the BIAS implementation (10).

As a second example the multiplication of a floating point matrix by an interval matrix is considered, i.e. $C = R \cdot A$, $R \in \mathbb{R}^{n \times n}$, $\mathbf{A} \in \mathbb{IR}^{n \times n}$. For simplicity we take square matrices.

The standard implementation looks like

$$C_{ij} := \sum_{k=1}^{n} R_{ik} \cdot A_{kj}, \qquad i, j = 1, 2, \ldots, n \tag{11}$$

which needs n^3 sign tests and $2n^3$ rounding mode switches. The BIAS implementation uses a row update which reduces the number of sign tests to n^2 and the number of rounding mode switches to $2n^2$:

$$\begin{aligned}
&\text{for } i = 1, 2, \ldots, n \text{ do}\\
&\quad C_{i*} = 0\\
&\quad \text{for } j = 1, 2, \ldots, n \text{ do}\\
&\qquad C_{i*} = C_{i*} + R_{ij} \cdot A_{j*}
\end{aligned} \tag{12}$$

where A_{j*} denotes the j-th row of A.

On an IBM Model 350 (RS/6000 architecture) with $n = 300$, the standard implementation (11) needs 33 seconds, where the BIAS implementation (12) needs only 6 seconds.

By using these techniques, a ratio of approx. 2 can be achieved in many cases between interval and the appropriate floating point operations, where floating point means *standard C* using a *standard compiler* with full optimization.

3. PROFIL—A C++ Class Library Based on BIAS

The routines discussed above are low-level routines and it is not very convenient to use them directly to write larger programs. Especially, there is no memory management of temporary values and no operator overloading. Because of this, we developed the more user friendly C++ library PROFIL (Programmer's Runtime Optimized Fast Interval Library) [17] which is based on BIAS. PROFIL is easily extendable by defining new classes for new data types. An optional runtime index check and an improved memory management avoiding unnecessary copies of large data structures are provided. Libraries like IMSL, NAG, and LAPACK can be used directly in PROFIL. Only very few simple lines of code describing the interface have to be written.

Currently, the following data types are supported by PROFIL: boolean, integer, real, interval, vectors and matrices of these types, and complex.

As a small example, consider the following complete subroutine for an interval linear system solver (cf. [22]):

```
INTERVAL_VECTOR ILSS (INTERVAL_MATRIX & A,
                      INTERVAL_VECTOR & b, INT & info);
{
  INT             dim = Dimension (b);
  INT             k, done;
  MATRIX          R (dim, dim);
  VECTOR          xs (dim);
  INTERVAL_VECTOR x (dim), y (dim), z (dim), Inflat (dim);
  INTERVAL_MATRIX C(dim,dim);
  INTERVAL        eps (0.9,1.1);

  Initialize(Inflat, SymHull (Machine::MinPositive));

  R = Inverse (Mid (A));
  xs = R * Mid (b);
  x = z = R * (b - A * xs);
  C = Id (dim) - R * A;
  k = 0;
```

```
  do {
    y = eps * x + Inflat;
    x = z + C * y;
    done = (x < y);
    k++;
  } while (!done && (k < 10));
  info = done;
  return (xs + x);
}
```

4. Performance Results

To show the overall performance of PROFIL together with BIAS, we first consider the interval linear system solver from above for some midrange dimensions n. In table 1 the total time needed to solve the linear system is displayed. Additionally, we give the average speed in MFlops. Apart from lower order terms, which are neglectable except for small dimensions, there are in total $6n^3$ floating point operations needed for the above interval linear system solver (addition and multiplication counting each as one operation). This value has been used to compute the average speed in Table 1. All times are obtained using the IBM Model 350 which is a 19 MFlops double precision LINPACK benchmark computer. For moderate dimensions Table 1 shows that this performance is also nearly achieved for interval computations when using PROFIL/BIAS.

The next example is the test suite taken from Corliss [2]. This test suite has been designed to compare interval arithmetic software packages. It consists of several small programs which exercise different parts of the software packages. The programs are:

Test 1: Exercise $+$, $-$, $\times$, and $/$.
Test 2: Exercise elementary functions.
Test 3: Exercise vector and matrix operations.
Test 4: One-variable interval Newton method.
Test 5: One-variable global optimization.

Table 1. Time needed for solving an interval linear system

dim	total time	MFlops
50	60 ms	12.5
100	400 ms	15.0
150	1190 ms	17.0
200	2730 ms	17.6
250	5350 ms	17.5

The third test is available in 2 versions: The first one named "Test 3" by Corliss uses an accurate scalar product and the second one named "Test 3a" by Corliss uses the ordinary floating point operations. In order to estimate the cost of interval arithmetic and the overhead caused by the software packages, the first 3 test programs have been modified such that all interval operations and values are replaced by their floating point equivalents. These modified programs are compiled under Fortran and can be used as machine dependent references.

For most of his tests Corliss used a 80486 50 MHz PC as reference. Some of his tests have also been run on a SUN Sparc 1+, the performance of which is comparable to the PC for these tests [2]. We run all our tests on a SUN SparcServer 330, which is also comparable to the 50 MHz PC used by Corliss, as the run time of the Fortran reference programs in Table 2 show.

Table 3 contains the run time in seconds for the different packages reported by Corliss as well as the run time of our PROFIL programs. Additionally, Table 3 also

Table 2. Execution times (sec) of raw Fortran reference programs

	Test 1	Test 2	Test 3a
Raw Fortran (no intervals) PC, MS Fort 7.0	0.86	0.33	0.97
Raw Fortran (no intervals) SparcServer 330, f77	0.9	0.4	0.8

Table 3. Run time (sec) for different interval arithmetic software packages

	Test 1	Test 2	Test 3	Test 3a	Test 4	Test 5
Clemmeson PC, MS Fort 7.0	19.99	—	—	12.58	—	—
INTLIB PC, MS Fort 7.0	89.86	89.86	—	13.73	142.48	288.80
INTLIB Spar 1+, f77	35.58	74.96	—	11.65	110.71	221.71
C-XSC PC, Borl C++ 3.1	52.34	56.85	78.64	25.65	27.95	50.15
Pascal-XSC PC, Borl C++3.1 Win	40.65	74.26	106.77	32.90	50.70	81.74
Pascal-XSC Sparc 1+	35.20	356.43	63.83	26.78	131.06	186.53
Pascal-XSC SparcServer 330, gcc	32.0	341.8	61.3	24.5	122.7	178.1
PROFIL/BIAS SparcServer 330, gcc	8.5	12.2	27.8	1.4	5.5	5.4

contains the run times we obtained for the Pascal-XSC programs on the Sparc-Server 330. Entries marked with a dash denote that the appropriate test program cannot be executed by the package because the required features (e.g. interval standard functions) are not available in that package. Note that PROFIL uses a general multiple precision arithmetic for the accurate scalar product which is slower than a special long accumulator. On the other hand it is much more universal (operands with different precision can be combined). The numbers show that the PROFIL programs (except for "Test 3") run about one order of magnitude faster than the programs of the other software packages. Especially the vector/matrix test program ("Test 3a") displays the efficient implementation of the vector and matrix operations in BIAS. The packages INTLIB and Pascal-XSC were designed for maximum portability (although the Pascal-XSC versions used in the tests uses the hardware directed roundings).

Table 4 contains the width of the achieved results, i.e. the tightness of the operations. The width achieved by PROFIL/BIAS is comparable with most other packages (except for "Test 2" and "Test 5", where the faithful rounding for elementary functions leads to a small overestimation).

The compile times and the size of the executable programs depend very much on the operating system and the compiler being used. The corresponding results for PROFIL are comparable with the other packages.

Table 4. Width of results for different interval arithmetic software packages

	Test 1	Test 2	Test 3	Test 3a	Test 4	Test 5
Clemmeson PC, MS Fort 7.0	3.86E-14	—	—	2.20E-7	—	—
INTLIB PC, MS Fort 7.0	2.15E-5	2.38E-11	—	1.54E-3	1.13E-13	9.16E-11
INTLIB Sparc 1+, f77	2.15E-5	2.38E-11	—	1.54E-3	1.13E-13	9.17E-11
C-XSC PC, Borl C++ 3.1	6.84E-11	1.98E-12	1.40E-7	2.36E-4	1.05E-14	2.97E-11
Pascal-XSC PC, Borl C++ 3.1 Win	6.84E-11	1.83E-12	1.41E-7	3.90E-4	1.06E-14	3.53E-11
Pascal-XSC Sparc 1+	6.84E-11	1.83E-12	1.41E-7	3.90E-4	1.06E-14	3.53E-11
Pascal-XSC SparcServer 33p, gcc	6.84E-11	1.83E-12	1.41E-7	3.90E-4	1.06E-14	3.53E-11
PROFIL/BIAS SparcServer 33p, gcc	6.86E-11	1.61E-11	1.40E-7	4.92E-4	2.46E-14	4.02E-9

5. Availability and Conclusion

The complete source code of PROFIL and BIAS is available for non-commercial use via anonymous ftp. Assembler subroutines for PCs (80386/387 and 80486), SUN Sparc, IBM RS/6000 series, and HP 9000/700 series are provided. For other architectures, the adaption ist mainly restricted to the 3 small asssembler subroutines providing the directed roundings. The long real and long interval arithmetic is currently a beta-test version and will be available soon.

An extension package [18] containing a set of test matrices, general linear lists, and automatic differentiation is available for non-commercial use. This package is constantly expanded by new data types and application routines. In the future we plan to further increase the BIAS performance by adaption to specific architectures (e.g. using blocked algorithms as for BLAS, see [6]).

PROFIL/BIAS is already used in different research areas (e.g. for a nonlinear system solver and for a global optimization package).

The complete PROFIL/BIAS package as well as the extension package can be obtained via anonymous ftp from the server

```
ti3sun.ti3.tu-harburg.de
```

The packages itself are contained in the directory `/pub/profil` and the documentation is available as compressed PostScript files in the directory `/pub/reports` as `report93.3.ps.Z`, `report93.4.ps.Z`, and `report.93.5.ps.Z`.

References

[1] Alefeld, G., Herzberger, J.: Introduction to interval computations. New York: Academic Press 1983.
[2] Corliss, G. F.: Comparing software packages for interval arithmetic. Preprint presented at SCAN '93, Vienna, 1993.
[3] Daumas, M., Matula, D. W.: Rounding of floating point intervals (to appear).
[4] Dongarra, J., Du Croz, J., Duff, I., Hammarling, S.: A set of level 3 basic linear algebra subprograms. ACM Trans. Math. Softw. *16*, 1–17 (1990).
[5] Dongarra, J., Du Croz, J., Hammarling, S., Hanson, R.: An extended set of Fortran basic linear algebra subroutines. ACM Trans. Math. Softw. *14*, 1–17 (1988).
[6] Dongarra, J. J., Mayes, P., Radicati di Brozolo, G.: The IBM RISC System/6000 and linear algebra operations. Science Tech Report CS-90-122, University of Tennessee, 1991.
[7] Hansen, G.: Global optimization interval analysis. New York Basel Hongkong: Marcel Dekker 1992.
[8] Herzberger, J.: Topics validated computations. Studies in Computational Mathematics (to appear).
[9] IEEE Standard for binary floating-point arithmetic, 1985. ANSI/IEEE Standard 754.
[10] Jansson, C.: A global optimization method using interval arithmetic. In: Computer arithmetic and enclosure methods, pp. 259–268. Amsterdam: Elsevier 1992.
[11] Jansson, C., Knüppel, O.: A global minimization method: The multi-dimensional case. Bericht 92.1, TU Hamburg-Harburg, Technische Informatik III, Jan. 1992.
[12] Jansson, C. Knüppel, O.: Eine intervallanalytische Methode für globale Optimierungsprobleme. Z. Ang. Math. Mech. *73*, T741–T743 (1993).
[13] Kearfott, R. B., Dawande, M., Du, K., Hu, C.: INTLIB: A portable Fortran-77 interval standard function library (to appear) in ACM Trans. Math. Software, 1994).

[14] Kearfott, R. B., Novoa, M.: INTBIS, a portable interval newton/bisection package. ACM Trans. Math. Software *16*, 152–157 (1990).
[15] Klatte, R., Kulisch, U., Neaga, M., Ratz, D., Ullrich, C.: Pascal-XSC: Language reference with examples. Berlin Heidelberg New York Tokyo: Springer 1992.
[16] Knüppel, O.: BIAS—Basic interval arithmetic subroutines. Bericht 93.3 des Forschungsschwerpunktes Informations- und Kommunikationstechnik der TU Hamburg-Harburg, TU Hamburg-Harburg, Technische Informatik III, July 1993.
[17] Knüppel, O.: PROFIL—Programmer's runtime optimized fast interval library. Bericht 93.4 des Forschungsschwerpunktes Informations- und Kommunikationstechnik der TU Hamburg-Harburg, TU Hamburg-Harburg, Technische Informatik III, July 1993.
[18] Knüppel, O., Simenec, T.: PROFIL/BIAS extensions. Bericht 93.5 des Forschungsschwerpunktes Informations- und Kommunikationstechnik der TU Hamburg-Harburg, TU Hamburg-Harburg, Technische Informatik III, Nov. 1993.
[19] Lawo, C.: C-XSC, a programming environment for verifield scientific computing and numerical data processing. In: Adams, E., Kulisch, U. (eds.) Scientific computing with automatic result verification, pp. 71–86. Orlando: Academic Press 1992.
[20] Lawson, C., Hanson, R., Kincaid, D., Krogh, F.: Basic linear algebra subprograms for Fortran usage. ACM Trans. Math. Software *5*, 308–323 (1979).
[21] Moore, R. E.: Methods and applications of interval analysis. Philadelphia: SIAM 1979.
[22] Rump, S. M.: Solving algebraic problems with high accuracy. In: Kulisch, U., Miranker, W. (eds.): A new approach to scientific computation, pp. 51–120. New York: Academic Press 1983.
[23] Schulze, J.: Die schwache Rundung und ihr Rundungsfehlerverhalten (to appear).

O. Knüppel
Technische Informatik III
Technische Universität Hamburg-Harburg
D-21071 Hamburg
Federal Republic of Germany