INTLIB: A Portable FORTRAN 77 Interval Standard Function Library

R. B. Kearfott*
Department of Mathematics
University of Southwestern Louisiana
U.S.L. Box 4-1010, Lafayette, LA 70504-1010 USA
email: rbkusl.edu

M. Dawande
Carnegie Mellon University

K. Du
University of Southwestern Louisiana

Ch. Hu
University of Houston–Downtown

Abstract

INTLIB is meant to be a readily available, portable, exhaustively documented interval arithmetic library, written in standard FORTRAN 77. Its underlying philosophy is to provide a standard for interval operations to aid in efficiently transporting programs involving interval arithmetic. The model is the BLAS package, for basic linear algebra operations. The library is composed of elementary interval arithmetic routines, standard function routines for interval data and values, and utility routines. The library can be used with INTBIS (Algorithm 681), and a Fortran 90 module to use the library to define an interval data type is available from the first author.

Keywords: interval arithmetic, standard functions, BLAS, operator overloading, FORTRAN 77, Fortran 90

Subject classifications: AMS: 65G10, 65D15, 26A09. CR: G.1.0 (Computer arithmetic), G.1.2 (standard function approximation), D.2.2 (Software libraries) D.2.7 (documentation, portability)

*This work is partially supported by National Science Foundation Grant no. CCR-9203730.

1 Introduction and Applicability of Interval Arithmetic

Interval arithmetic, introduced in [1], [22] and other texts, is an extension of arithmetic operations to intervals, such that the result of, e.g. a binary interval operation is the set of all possible results of the corresponding operation on real numbers, choosing the first operand from the first interval and the second operand from the second interval. Such arithmetic is useful to

- rigorously bound roundoff error in a variety of numerical computations, and
- compute rigorous bounds on the range of functions.

Interval arithmetic lends certainty to computations through the use of *directed roundings*, in which the exact result of an arithmetic operation[1] is enclosed in a containing interval whose endpoints are machine numbers. Such rigor, combined with tools such as fixed-point iteration, allows *automatic result verification* and *automatic theorem proving.* For example, it is well known among researchers in the field that interval Newton methods can prove existence, uniqueness, or non-existence of roots of a nonlinear system of equations within a given rectangular region in multidimensional space; see [23]. With clever bounding of truncation errors, these computational techniques can also be used in existence proofs for infinite-dimensional problems such as partial differential equations, as described in [10] and more recent work.

Interval arithmetic cannot be used naively (e.g. by replacing all floating-point operations in a program by interval operations), since, due to *interval dependencies*, the interval value of an arithmetic expression can sometimes be a severe overestimation of the range of that expression. Nonetheless, interval techniques have been successful in a variety of applications. For example:

- If an approximate root of a linear or nonlinear system of equations has been found by conventional means, it is often easy for interval techniques to *verify* that an actual root of the system lies within a small rectangular region (box) surrounding the approximate root.
- Rigorous computations of bounds on ranges of functions is a powerful tool in global optimization algorithms, where such bounds can be used

[1] assuming the operands are exact

in exhaustive searches to reject subregions that cannot contain global optima. In fact, even though the interval arithmetic implementation itself may be much slower, interval algorithms can not only be rigorous but also faster than alternate techniques, such as Monte Carlo methods or simulated annealing.

- It has been shown recently ([16]) that use of interval arithmetic to estimate the sensitivity of certain large sparse linear systems can not only lead to tight but correct bounds on the sensitivity, but can, in some cases, be computable with orders-of-magnitude less work than the LINPACK condition estimator.

In fact, hundreds of researchers have made steady progress over the past three decades[2] in increasing applicability of interval techniques to practical problems.

The package presented here originally grew out of the desire to extend the root-finding package INTBIS of [11] to handle transcendental in addition to polynomial systems. However, we have also made the package transparent and have included standard capabilities, for general utility in the contexts mentioned above.

2 Purpose and General Properties of the Package

Various packages have been made available for interval arithmetic support. However, these lack portability, are in obsolete FORTRAN 66 (and thus harder to understand and maintain), or have other undesirable features. INTLIB is meant to be an up-to-date FORTRAN 77 package, quickly installable on most machines. It is not meant to be optimal on any particular machine, but provides a template upon which optimized versions can be designed. Thus, INTLIB is consistent with the BLAS philosophy (Algorithm 539, [20]). We also used Corliss' BIAS proposal ([4]) as a guide in designing INTLIB.

INTLIB does not assume IEEE arithmetic, nor does it assume anything about the accuracy of the standard functions supplied with the Fortran compiler. However, it is generally structured top-down so modifications to higher-level routines will allow efficient use of accurate FORTRAN 77 supplied functions, when available.

[2]since the establishment of interval arithmetic, by Moore

INTLIB uses the machine's intrinsic double precision floating point arithmetic, and simulates directed roundings for rigor.

We took care when designing INTLIB to make it compatible with INTBIS (Algorithm 681), a package that uses an interval Newton method combined with exhaustive search to rigorously find all roots of a nonlinear system of algebraic equations. (See [11].) We provide instructions for incorporating INTLIB into INTBIS in §5.3 below.

INTLIB contains a set of subroutines for support of an interval data type, assuming operator overloading is available. We have designed a Fortran 90 module, described in [13], containing interfaces to INTLIB routines. This module and INTLIB, without modification, provide an interval data type in Fortran 90. The interface has been designed to be, as much as possible, consistent with a simple design philosophy and consistent with the language ACRITH-XSC (known earlier as FORTRAN-SC; see [25]).

INTLIB provides error handling through a set of flags in a common block. Level of message printing, as well as level of error that terminates execution, can be user-specified. Errors normally handled by the Fortran system, such as overflows and underflows, are not trapped. However, we make some effort in INTLIB to avoid overflows and destructive underflows resulting from intermediate computations in INTLIB.

Alternatives to INTLIB exist in many cases. For example, the "XSC" languages have been developed as a coordinated effort over the years at the University of Karlsruhe. Using operator overloading and other object-oriented concepts, they represent portable extensions to common languages. The XSC languages include Pascal-XSC [7], C-XSC [19], and Fortran-XSC[3] [24]. These commendable languages are complete, sophisticated systems that, unlike INTLIB, feature maximal accuracy and an accurate dot product[4], and are available commercially. However, none of them is available in strict FORTRAN 77 environments. Additionally, INTLIB is meant to be a simple framework, suitable for many production-type computations, but transparent enough for the user to comprehend and improve.

INTLIB and the aforementioned languages represent interval computations within traditional programming environments. Interval computations are also available within interactive systems such as Maple [3] and Mathe-

[3] Fortran-XSC is still under development at the time of writing; we briefly discuss it in §6 below.

[4] Maximal accuracy and accurate dot products are useful in many contexts, especially when dealing with ill-conditioned linear systems. However, they are not indispensable in many situations.

matica [14]. Also, additional packages are mentioned in [12].

3 Package Contents

We have organized the package into

- elementary interval arithmetic subroutines,
- interval standard function subroutines,
- utility routines,
- an error-printing routine,
- a routine to set mathematical constants and machine-dependent constants, and
- testing programs.

Calls to all routines in INTLIB are similar. For example, ADD, which adds intervals A and B, returning the sum in RESULT, is structured as follows.

```
SUBROUTINE ADD(A, B, RESULT)
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
DOUBLE PRECISION A(2), B(2), RESULT(2)
```

(Univariate routines have the argument list (A, RESULT).)

All routines are "safe" in the sense that they give the expected result when input arguments and output arguments are the same. For example, the statement

```
CALL MULT(A, B, A)
```

causes A to be replaced by $A * B$. (The authors first saw this concept in [6].)

We list the user-callable routines in INTLIB below.

3.1 Elementary Interval Arithmetic Subroutines

ADD, which adds two intervals;

CANCEL, which performs cancellation-type subtraction on an accumulated sum.

IDIV, which does interval division of ordinary intervals;

MULT, which multiplies two intervals

RNDOUT, which performs simulated directed rounding;

SCLADD, which adds an interval to a point;

SCLMLT, which multiplies an interval by a point; and

SUB, which subtracts two intervals.

The simulated directed rounding in RNDOUT is done by multiplication of computed results by $(1+\epsilon)$, for appropriate ϵ; see §6 below and the source files for RNDOUT and SIMINI.)

3.2 Interval Standard Function Subroutines

IACOS (Interval arc cosine)

IACOT (Single argument arc cotangent)

IASIN (Interval arc sine)

IATAN (Single argument arc tangent)

ICOS (Interval cosine)

IEXP (Interval e^x)

IIPOWR (Nonnegative interval to an interval power)

ILOG (Interval natural logarithm)

ISIN (Interval sine)

ISINH (Hyperbolic sine)

ISQRT (Interval square root)

POWER (Integer power of an interval)

Additionally, the routines ASNSER, ATNRED, ATNSER, ISNRED, ISHSER, ISNVAL, RRPOWR, RCOS, REXP, RLOG, and RSQRT are associated with the interval standard function routines, but are not normally called by the user. These routines perform argument reductions or rigorous (rounded out) evaluations at endpoints.

3.3 Utility Routines

These routines have simple functions, but are useful if one wants error checking and assurance of proper rounding. They are also necessary if an interval data type is to be defined through overloading.

ICAP (Intersection)

IDISJ (Two intervals disjoint)

IHULL (Convex hull)

IILEI (Set inclusion in closure of interval)

IILTI (Set inclusion in interior)

IINF (Return left endpoint)

IMID (Return approximation to midpoint using available floating point arithmetic)

IMIG (Mignitude)

INEG (Unary negation)

INTABS (Interval absolute value – a double precision value)

IRLEI (Point inclusion in closure of interval)

IRLTI (Point inclusion in interior of interval)

ISUP (Return right endpoint)

IVL1 (Construct interval from a point)

IVL2 (Construct interval from its endpoints)

IVLABS (Range of $\| \circ \|$ over an interval)

IVLI (Assign one interval to another)

IWID (Outwardly rounded width)

3.4 Miscellaneous and Machine Dependent Routines

ERRTST: This routine is used for printing information about error conditions as they occur. Since no files are attached here, there should be no machine dependencies.

SIMINI: This routine is used to set parameters for a particular machine's arithmetic and mathematical constants. A quantity MAXERR, indicating the number of units in the last place by which double precision addition, subtraction, multiplication, or division may be in error must be set. Additionally, some representations of mathematical constants may need to be changed for unusual machines. (See §5 below.)

3.5 Arithmetic Query Routines

INTLIB uses the SLATEC routines D1MACH and I1MACH to obtain machine constants such as the maximum relative distance between two floating point numbers. The target machine may have these routines already installed. Otherwise, they must be installed properly for INTLIB to give correct results.

3.6 The Testing Routines

The test programs check the INTLIB routines at special points of the standard functions, at crossover points and extreme points in the argument reduction, at points near the limits of the floating point system, and on a set of fixed pseudo-random intervals. The test data is scaled according to the particular floating point system, so output will differ from machine to machine. However, the test output indicates widths of resulting intervals and whether the intrinsic Fortran library function values are contained in the corresponding interval bounds.

The test routines are driven by the program TSTDRV, which calls the routines **TIACOS**, **TIACOT**, **TIASIN**, **TIATAN**, **TICOS**, **TIEXP**, **TIIPWR**, **TILOG**, **TINPWR**, **TISIN**, **TISINH**, **TISQRT**, and **TIUTIL**. The test programs also include the auxiliary routines DACOT1, EVAL,

EVAL2, EVAL3, RPOWR, UOUT1, UOUT1B, UOUT1C, UOUT2, UOUT2B, UOUTL, AND UOUTLB.

The paper [12] contains a brief discussion of the design principles for the test routines. Also see §5.2 below.

4 Approximation and Argument Reduction Principles

The routines are structured to determine ranges (i.e. interval values) from rigorous lower and upper bounds of values at the endpoints[5]. However, since specific accuracy of the Fortran intrinsic functions is not assumed[6], INTLIB contains routines to compute rigorous (and as sharp as possible, without using extended precision) lower and upper bounds of the values of the standard functions at points. Generally, these latter routines use standard argument reduction techniques and Taylor polynomial evaluation.

The Taylor polynomials, though used only to obtain "point" values, are evaluated in interval arithmetic to rigorously bound the roundoff errors, and the (small-width) interval error term is then added. Similarly, the argument reductions, though applied to "point" inputs, are carried out in interval arithmetic. (However, see comments in §6 below.)

The routines for IASIN, IACOS, IATAN, IACOT and ISINH, for the arcsine, arccosine, arctangent, arccotangent and hyperbolic sine, respectively, use special reductions and adaptive choice of Taylor series order; these techniques are explained in [9]. The routines for the trigonometric, exponential, and logarithm functions use fixed-order Taylor series, while the square root function uses argument reduction, a low-order Taylor series approximation, and an interval Newton method. The routine IIPOWR, to raise an interval to an interval power, uses the interval exponential and logarithm functions.

Accuracy is limited in some cases by the restriction that the arithmetic be based on the machine's underlying double precision arithmetic. For example, for large arguments, returned bounds for the values of the trigonometric functions, given point input, are wider (sometimes even with relative widths of 10^{-5}) for large arguments; for arguments larger than $N\pi$, where N is the largest representable integer, a warning is signalled, and the correct but wide interval $[-1, 1]$ is returned.

[5]This is straightforward for monotonic functions, but somewhat more involved for the sine and cosine.

[6]and indeed, all Fortran standards are silent on this point

Details of the argument reduction, as well as details of the Taylor series evaluations, are clearly marked in-line in the source.

5 Installation and Use

5.1 The Distribution Files

INTLIB is distributed in eight parts (or files), totalling around 721,018 bytes. These parts are

D1I1MACH.FOR (approx. 28,380 bytes), containing the routines D1MACH and I1MACH from the SLATEC package, used to set machine parameters[7].

BASICOPS.FOR (approx. 32,773 bytes), containing the elementary interval arithmetic subroutines as in §3.1. Documentation for these routines appears in their prologues.

UTILFUNS.FOR (approx. 31,964 bytes), containing basic utility functions in §3.3.

ELEMFUNS.FOR (approx. 164,999 bytes), containing routines for the standard functions in §3.2.

MISCMACH.FOR (approx. 30,122 bytes) which contains the routines of §3.4.

TSTPROGS.FOR (approx. 166,618 bytes), containing the driver and the test subroutines of §3.6.

INTLIBTS.OUT (approx. 266,162 bytes), an output file produced by running TSTDRV on an MS-DOS system[8] with the Microsoft Fortran compiler version 4.1.

[7]These routines may be absent, depending on alternate availability and ACM and copyright policy.

[8]INTLIB has also been tested on Sun SPARC systems, on an IBM 3090 with the VM operating system and VS-Fortran, on a Convex, and on the MS-DOS and SPARC systems using the NAG Fortran 90 compiler, versions 1.0, 1.2, and 2.0.

5.2 Steps for Installation

Installation consists of

1. Altering D1MACH and I1MACH for the particular machine in use. As provided, these routines may be set for MS-DOS systems. They will need to be changed for other systems, as indicated in their in-line documentation. Also, D1MACH and I1MACH may not contain machine constants for some newer machines; such machine constants must be provided. D1MACH and I1MACH are possibly already on the system; in that case, the file file D1I1MACH is not needed with INTLIB.

   CAUTION: Use of the machine constants in D1MACH is not rigorous on machines that do not have guard digits. See Kulisch / Miranker [18, pp. 5–6], On such machines INTBIS can be made rigorous by using a larger value instead of D1MACH(4) (representing the largest relative distance between floating point numbers). This is analogous to choosing a larger ϵ^* in the Kulisch / Miranker analysis (ibid.), and is equivalent to assuming the computer's wordlength is somewhat smaller. In effect, the technique uses part of the computer word as guard digits, to avoid the type of subtraction error illustrated in Kulisch / Miranker. Sometimes, only one or two such guard digits are needed. For example, on a hypothetical decimal machine with D1MACH(4) $= 10^{-8}$, use of 10^{-7} instead should do. Input data should then be stored into intervals using the routines IVL1 and IVL2, which will round out in this lower precision.

   On some Cray machines, however, the situation is more complicated. On such machines, division is implemented as an iterative algorithm, and more than one bit is lost, depending on the numerator and denominator. However, the authors have been informed (Hartmut Schwandt, Peter Tang, private conversations) that it is safe to assume the elementary operations on Cray machines to be accurate to within 20 units in the last place. Thus, INTLIB could be made rigorous on Cray machines by using a value at least twenty times the value given by D1MACH(4). However, see the comments in §6 below.

   The authors would like to be contacted if there are any doubts.

2. Carefully reading the assumptions in the prologue to SIMINI (in file MISCMACH.FOR), and possibly altering SIMINI. In particular, SI-

MINI stores mathematical constants as twenty-five digit decimal representations. It is assumed that the arithmetic is not more accurate than this, and that the conversion error from the decimal representation to the machine's internal floating point is accurate to the same relative error as the elementary operations. If these assumptions are not valid, then the constants will need to be represented in a different form. Also, SIMINI contains a data statement where MAXERR, the maximum number of units in the last place by which double precision addition, multiplication, subtraction, division can be in error, is set. For IEEE machines, MAXERR = 1. However, approximation errors in conversion of ASCII input data to floating point may may need to be be taken into account in some contexts, as above.

3. Possibly altering the test driver TSTDRV.FOR of §3.6. The only items that should require alteration are the OPEN statement, the first executable statement, and the parameter IOUTUN, giving the unit number for output.

4. No other files should need to be altered.

5. Compiling the files D1I1MACH.FOR, BASICOPS.FOR, UTILFUNS.FOR, ELEMFUNS.FOR, and MISCMACH.FOR. The object code should be made available as libraries or in other formats, so that the individual routines in these files are available to user calling programs. Note that D1I1MACH may already be installed on the system.

6. Compiling TSTPROGS.FOR.

7. Linking and running TSTPROGS. No input files are required. If possible, the system should be set so that execution continues after overflows and underflows. Underflows should be set to zero.

8. Carefully examining the output file. The philosophy underlying the testing is explained in [12]. The file INTLIBTS.OUT serves as a guide of what to expect. A test is successful provided the containing intervals corresponding to point data contain the corresponding floating point values from the compiler-supplied intrinsic functions. Successful tests do not prove that the functions in the package will never give erroneous results (though we have done our best to eliminate bugs, thus ensuring that they never do). An unsuccessful test may have the following causes:

(a) improper installation,

(b) inaccurate intrinsic functions supplied with the Fortran compiler, or

(c) a bug or bugs in the package.

Though the output file from the testing can be compared with the output files INTLIBTS.OUT, the output will vary from system to system, especially for the last twenty data intervals. These intervals have pseudo-random endpoints scaled to individual floating point systems.

Some of the evaluations will cause error messages to be printed. This is normal, as is illustrated in the sample output file.

The summary sections in the output file should be interpreted carefully. In particular, the "maximum relative error with point input" and "maximum absolute error with point input" are given. It is possible that the routines are functioning correctly, yet both error measures may be large. There are two reasons for this. First, some of the "point" data actually consists of intervals with small relative widths. Second, in some regions of the range of some of the functions, relative error is a natural measure, whereas, in other regions, absolute error is more appropriate. For sin and cos, absolute error is meaningful, but relative error is reasonable for the exponential function. Also, in many cases, the functions are tested very strictly, near the extremes of the particular floating point system. Of course, both relative and absolute accuracy could be improved if a higher precision data type were assumed to be available. However, despite large relative or abolute interval widths over parts of the range, the INTLIB functions provide useful accuracy.

5.3 Installation for Use with INTBIS

If the package is to be used with INTBIS ([11]) then routines in INTBIS of the same name as routines in this package should be deleted, and the new routines (in this package) should be used instead. In particular, the old versions of ADD, MULT, POWER, RNDOUT, SCLADD, SCLMLT, SIMINI, and SUB should be deleted.

5.4 Installation with Optimally Accurate Intrinsics

If the return-values of the Fortran intrinsic functions are within one unit in the last place of the corresponding mathematical values of the function, then considerable simplification and speedup are possible. Basically, calls to underlying routines like RCOS (routines R$*$) can be replaced by calls to corresponding standard functions. Such modifications will require some study of the package structure, but the package is meant to be transparent.

Similarly, simplifications are possible in various places if IEEE arithmetic is used. However, see §6 below.

5.5 An Interval Data Type

The authors have developed an interval data type using INTLIB and a Fortran 90 interface. It is presently available for testing purposes.

5.6 Use

The most straightforward use involves simply calling the routines in INTLIB from the application program. This requires only FORTRAN 77 and the ability to link the INTLIB libraries. However, the INTLIB routine SIMINI must be called once, at the beginning of the program, before computations begin. Not doing so is a common error that, due to uninitialized constants, leads to INTLIB signalling numerous seemingly unrelated warnings and errors. The authors know of no portable way of checking in FORTRAN 77 whether SIMINI has been called.

It is strongly recommended to use the interval data type the authors provide, if Fortran 90 is available. Doing so frees the program developer from a large programming burden, makes the resulting code more transparent, and does not incur a large run-time penalty.

6 Future Improvements

The present package's simplicity and capabilities are constrained by requirements of

- FORTRAN 77 compatibility,
- total portability,
- no assumption of IEEE arithmetic,

- rigor (i.e. guaranteed results), and
- transparency (ease of reading) of the source code.

However, on a certain test battery (of [5]) and on various machines, INTLIB arithmetic runs roughly twenty times slower than ordinary floating point arithmetic. Thus, improvements are possible.

If some portability is sacrificed, then better accuracy and much better speed can be achieved. For example, accurate standard functions, for reduced arguments, are available on 80486 and Pentium chips; these could be used to directly build interval evaluations. Also, the IEEE standard dictates an extended precision, beyond floating point, that can be used in argument reductions, though this precision is not accessible through FORTRAN 77.

Non-trivial improvements are nonetheless possible, even while maintaining total portability and simplicity. For example, to bound roundoff error, INTLIB presently evaluates Taylor polynomials in interval arithmetic. However, as shown in [2] and [17], given the parameters in the floating point system, the roundoff error in argument reduction and Taylor polynomial evaluation can be bounded a priori. With such bounds, the Taylor polynomials can then be evaluated using floating point arithmetic. Preliminary experiments on the exponential function indicate this technique to be about five times faster. However, significant additional analysis is necessary to incorporate it into INTLIB. Similar speedups may be possible, using totally different approximation techniques, as indicated in [21].

Significant execution time in INTLIB is spent in the routine RNDOUT that simulates directed roundings by multiplying results x by $1+\epsilon$, $1-\epsilon$ or by setting results to zero or plus or minus the smallest machine number, depending on the sign and size of x. However, *true* directed roundings are available in IEEE arithmetic. In [15], such directed roundings are implemented in C++ software for Sparc and 80X86 systems with a single assembly language instruction. The rounding must be changed between "down" and "up" to get lower bounds and upper bounds. Minimizing this changing in vector operations, the package of [15] is reported to execute interval arithmetic no more than five times slower than floating point arithmetic.

Finally, Fortran 90 syntax could simplify the code. For example, environment inquiry functions would obviate some of the need for providing the explicit machines constants of the SLATEC routines D1MACH and I1MACH. The cumbersome common blocks holding mathematical constants could be replaced by modules. The standard function routines could also be written with a subset of the interval data type, to replace long strings of subroutine calls with arithmetic expressions. In fact, this design path has been taken in [24], although that work is at present ongoing.

7 Summary

INTLIB is a portable, available, tested, well-documented and supported FORTRAN 77 system for interval arithmetic. Its underlying design philosophy includes rigor, transparency of the source code to the user, and total portability. In conjunction with operator overloading in Fortran 90, the authors are presently using it extensively in their own research on interval arithmetic algorithms; various others have also tested and used it. It can form the basis for machine-optimized and improved versions.

8 Credits

We wish to acknowledge George Corliss at Marquette University. Many discussions with him both encouraged us and allowed us to improve the package. We also wish to acknowledge Rebecca Yun, a one-time student at the first author's university, and Abdulhamid Awad, an undergraduate at the University of Houston–Downtown who worked on several of the routines.

References

[1] Alefeld, Götz, and Herzberger, Jürgen, *Introduction to Interval Computations*, Academic Press, New York, etc., 1983.

[2] Braune, K. D., *Hochgenaue Standardfunktionen für reele und komplexe Punkte und Intervalle in beliebigen Gleitpunktrastern*, Ph.D. dissertation, Universität Karlsruhe, 1987.

[3] Connell, A. and Corless, R. M., *An Experimental Interval Arithmetic Package in Maple*, Interval Computations, in press.

[4] Corliss, G. F., *Proposal for a Basic Interval Arithmetic Subroutines Library (BIAS)*, preprint, 1990.

[5] Corliss, G. F., *Comparing Software Packages for Interval Arithmetic*, talk presented at the IMACS / GAMM International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics, Vienna, Austria, September 26–29, 1993.

[6] Crary, F., *The AUGMENT Precompiler*, technical report no. 1470, Mathematics Research Center, The University of Wisconsin, Madison, 1976.

[7] Hammer, R., Neaga, M., Ratz, D., *PASCAL-XSC, New Concepts for Scientific Computation and Numerical Data Processing*, in Scientific computing with automatic result verification, pp. 15–44, Academic Press, New York, etc., 1993.

[8] Hansen, E. R., *Global Optimization using Interval Analysis*, Marcel Dekker, Inc., New York, 1992.

[9] Hu, C. and Kearfott, R. B., *On Bounding the Range of Some Elementary Functions in FORTRAN 77*, Interval Computations, in press.

[10] Kaucher, E. W. and Miranker, W. L., *Self-Validating Numerics for Function Space Problems*, Academic Press, Orlando, 1984.

[11] Kearfott, R. B., and Novoa, M., *INTBIS, A Portable Interval Newton/Bisection Package (Algorithm 681)*, ACM Trans. Math. Software **16** (2), pp. 152–157, 1990.

[12] Kearfott, R. B., Dawande, M., Du K.-S. and Hu, C.-Y., *INTLIB: A Portable FORTRAN 77 Elementary Function Library*, Interval Computations, in press.

[13] Kearfott, R. B., *A Fortran 90 Environment for Research and Prototyping of Enclosure Algorithms for Constrained and Unconstrained Nonlinear Equations*, submitted to the ACM Trans. Math. Software.

[14] Keiper, J. B., *Interval Arithmetic in Mathematica*, Interval Computations, in press.

[15] Knüppel, O., *BIAS – Basic Interval Arithmetic Subroutines*, talk presented at the IMACS / GAMM International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics, Vienna, Austria, September 26–29, 1993.

[16] C. F. Korn and Ch. Ullrich, *Extending LINPACK by Verification Routines for Linear Systems*, talk presented at the IMACS / GAMM International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics, Vienna, Austria, September 26–29, 1993.

[17] Krämer, W., *Inverse Standardfunktionen für reelle und komplexe Intervallargumente mit a priori Fehlerabschätzungen*, Ph.D. dissertation, Universität Karlsruhe, 1987.

[18] Kulisch, Ulrich W., and Miranker, Willard L., *A New Approach to Scientific Computation*, Academic Press, New York, 1983.

[19] Lawo, C., *C-XSC A Programming Environment for Verified Scientific Computing and Numerical Data Processing*, in Scientific Computing with Automatic Result Verification, pp. 71–86, Academic Press, New York, etc., 1993.

[20] Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., *Algorithm 539: Basic Linear Algebra Subprograms for FORTRAN Usage*, ACM Trans. Math. Software **5** (3), pp. 308–325, 1979.

[21] Luther, W. and Otten, W., *Computation of Standard Interval Functions in Multiple-Precision Interval Arithmetic*, technical report no. SM-DU-233, Universität Duisburg, 1993.

[22] Moore, R. E., *Methods and Applications of Interval Analysis*, SIAM, Philadelphia, 1979.

[23] Neumaier, A., *Interval Methods for Systems of Equations*, Cambridge University Press, Cambridge, England, 1990.

[24] Walter, W. V., *FORTRAN-XSC: A Portable Fortran 90 Module Library for Accurate and Reliable Scientific Computing*, Computing (Suppl.) **9**, pp. 265–286, 1993.

[25] Walter, W. V., *ACRITH-XSC: A Fortran-Like Language for Verified Scientific Computing*, Academic Press, New York, etc., in Scientific Computing with Automatic Result Verification, pp. 45–70, Academic Press, New York, etc., 1993.