Interval Computations
No 3(5), 1992

INTLIB: A PORTABLE FORTRAN-77 ELEMENTARY FUNCTION LIBRARY

Baker Kearfott, Milind Dawande, Kaisheng Du and Chenyi Hu

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. In this paper, we (1) outline previous packages and present efforts, (2) explain the overall structure of the package, as well as give descriptions of some of the routines, and (3) mention future enhancements.

INTLIB: ПЕРЕНОСИМАЯ БИБЛИОТЕКА ЭЛЕМЕНТАРНЫХ ФУНКЦИЙ НА ЯЗЫКЕ FORTRAN-77

Б. Кирфотт, М. Дуанде, К. Ду, Ч. Ху

Библиотеку INTLIB предполагалось сделать легко доступной, переносимой, исчерпывающе документированной библиотекой интервальной арифметики, написанной на стандартном языке FORTRAN-77. В ее основе лежит намерение обеспечить стандарт для интервальных операций с тем, чтобы помочь в эффективной переносимости программ, содержащих интервальную арифметику. Прототипом библиотеки является пакет BLAS для основных операций линейной алгебры. В настоящей работе мы (1) дадим обзор предшествующих пакетов и наших достижений, (2) опишем общую структуру пакета и дадим спецификацию некоторых процедур и (3) наметим пути развития пакета.

1. Introduction and purpose

Our INTBIS algorithm [12] for finding all solutions to certain polynomial systems of equations contained portable Fortran 77 routines for evaluation of the basic arithmetic operations. These routines allowed INTBIS to find all roots of polynomial systems of equations. Since INTBIS appeared, various researchers have asked us for additional routines to allow finding roots of systems which involve transcendental functions. This led to INTLIB.

Various packages supporting interval arithmetic have been described in the literature. The first such package was perhaps Herzberger's *Algol-60 Procedures Evaluating Standard Functions in Interval Analysis* [9]. Yohe's Fortran 66 code (*Software for Interval Arithmetic: A Reasonably Portable Package*) [20] followed nine years later. Clemmesen, under the direction of K. Madsen, published language-independent pseudocode in [5]. Computer languages which explicitly support interval arithmetic include Pascal-SC [16] the precompiler TPX for Turbo-Pascal [17] and Fortran-SC [2]. Various libraries [11,7] are subroutine packages in C++, written to take advantage of that language's operator overloading capability. Aberth and Schaefer have provided C++ modules for range arithmetic, a useful and interesting variant of interval arithmetic. Other public implementations of interval computations, such as interval support in the MAPLE symbolic manipulation package [18], are available.

We do not attempt to mention all of the interval arithmetic implementations researchers have produced for their own use, in an ad-hoc way, or as a means to proceed with a higher-level computation. An example of this is the interval arithmetic by Krishchuk, Vasilega, and Kozina implemented in assembler language for IBM PC compatibles [14].

None of the above packages is universally applicable. Yohe's is large, involved, in obsolete Fortran 66, and presently unsupported. The interval arithmetic support in language compilers and precompilers is limited to particular machines. The C++ libraries are valuable, but only if the computational project is carried out in C++.

Corliss has proposed language-independent specifications for a "Basic Interval Arithmetic Subroutines Library" (BIAS), as well as bindings for interval arithmetic in Fortran, Ada, and C [6]. The goal was to standardize interval arithmetic in the same way that the BLAS standardized

numerical linear algebra, aided portability, and aided optimization of applications codes on advanced architecture machines. INTLIB is a realization of this BLAS / BIAS philosophy. INTLIB implements the most important operations in the BIAS proposal.

INTLIB is meant to provide:

- portable, rigorous interval arithmetic to the Fortran community (to prevent wheel re-invention),
- well-documented templates for the design and validation of efficient machine-specific versions.
- a library to support operator overloading in Fortran 90,
- a reasonable testing paradigm, and
- simple but professional error signalling.

Though possibly not optimal on specific architectures, INTLIB is meant to be usable in a production mode. It can be used by itself in Fortran 77 or with operator overloading in Fortran 90.

2. Contents and structure of the package

We have organized the package into

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

Elementary interval arithmetic subroutines.

These routines include subroutines which are essentially unchanged from our previous work INTBIS, subroutines which are found in improved form from INTBIS, and new routines. Routines in a form essentially as in INTBIS include:

ADD, which adds two intervals;

SCLADD, which adds an interval to a point;

SCLMLT, which multiplies an interval by a point; and

SUB, which subtracts two intervals.

The following routines have been improved from INTBIS.

MULT: For multiplying two intervals, the version in INTBIS used the naive definition (2.19) of [15]. The improvement, suggested in [10], is an implementation of the more efficient formulas (2.20) of [15].

POWER: The INTBIS version took a nonnegative integer power of an interval. The routine has been generalized to allow taking a negative power of an interval. Associated error checking is provided. The routine has also been rewritten to be rigorous on systems for which the Fortran intrinsic represented by A**N is not optimally accurate. Thus, this routine is now more properly an elementary function, rather than a basic operation.

RNDOUT: Called after each elementary operation, this routine performs simulated directed rounding, to make the interval arithmetic mathematically rigorous. This is done using a simplified model of the number system (similar to that in [3]). The version in INTLIB has improved behavior near the underflow threshold.

The following routines are new in INTLIB.

IDIV does interval division of ordinary intervals. It signals an error if the result is an extended interval.

CANCEL performs cancellation-type subtraction on an accumlated sum. For example, if one knew that X and Y were the same interval, then the result of applying CANCEL to X and Y would be an interval containing zero whose width would be on the order of the unit roundoff error.[1]

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

[1] Of course, ordinary interval subtraction $X - X$ would not yield zero unless the width of X were zero.

```
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).)

In the basic package, we have not attempted to implement extended interval arithmetic, or indeed, any operation whose result may be anything other than a standard interval. The set of extended intervals is not closed under intersection, etc., and additional support functions and error checking would be advisable. Also, the precise definitions of extended interval arithmetic are still subject to some modification.

Interval elementary function subroutines.

The following routines have been completed at the time of writing of this report.

ICOS	(Interval cosine)
IEXP	(Interval e^x)
ILOG	(Interval natural logarithm)
ISIN	(Interval sine)
ISQRT	(Interval square root)
IATAN	(Single argument arctangent)
IACOT	(Single argument arccotangent)
IASIN	(Interval arcsine)
IACOS	(Interval arccosine)
IIPOWR	(Nonnegative interval to an interval power)
ISINH	(Hyperbolic sine)

In general, we used various argument reduction techniques coupled with Taylor polynomial approximations. Actual details appear in in-line documentation. In our design, we weighed conflicting goals of portability, clarity, accuracy (tightness of the bounds), and efficiency.

The interval square root, an elementary operation in the IEEE floating point standards [19], requires a more sophisticated algorithm for non IEEE arithmetic. Since we do not assume IEEE arithmetic in the portable version of INTLIB, we used a combination of argument reduction, Taylor polynomial approximation, and an interval Newton method.

Likewise, we assume nothing about the Fortran-supplied intrinsic func-

tions. Thus, even for monotone functions, we supply our own code to compute bounds on the values at the endpoints of the domain intervals.

The value of the two argument arctangent would possibly consist of more than one interval, due to branch points. Due to this complication, we excluded the two argument arctangent from the initial package.

Utility routines.

These routines are needed to support operator overloading, as well as for reliable exception handling. The list is modified from Corliss' proposed BIAS specifications.

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)

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)

IWID (Outwardly rounded width)

The error handling routine.

Error handling is roughly similar to that in INTBIS, but is better developed.

Errors are checked within the individual routines at the points where they possibly occur. If an error condition occurs, an integer flag E is set, and the error routine ERROUT is called. Subsequent action depends on E and on two other flags P and S. The errors are assigned three severities[2], similar to the scheme in the SLATEC library [8][3]. A print flag P controls which severity of errors is printed, and a stop flag S controls which error severity causes termination of execution.

The flag E, set by the package, and the flags P and S, set by the user, are global[4]. An additional global flag R, set by the package, indicates in which routine the error occurred. An additional global variable indicates the unit number for output of error messages.

Routine to set constants.

This is one of the few routines which may need some modification in transporting the library from one machine to another.

This routine grew out of routine SIMINI in INTBIS. Its purpose is to set machine constants (unit roundoff error, etc.) and interval inclusions for mathematical constants such as π, e, and related values. These are then stored in global variables (common blocks) for efficient use by the elementary function routines. Installation requires setting the maximum number of units in the last place by which an elementary operation can be in error. This value is 1 for IEEE arithmetic.

The inclusions for the mathematical constants are obtained by expressing the constants to more digits than the precision, then using simulated directed rounding. We assume that conversion of decimal numbers to the machine format is as accurate as a floating point operation. Othewise, the representation of the constants will need to be changed.

The representation of the constants may also need to be changed if double precision is more highly accurate than on most machines

Some simplifications of this routine would be possible with the Fortran 90 machine query functions.

[2](0, 1, and 2, corresponding to warning, error, and fatal error)

[3]However, to keep calling sequences simple for operator overloading, the severity flag is stored in a common block instead of passed as an argument.

[4](in the common block ERR)

3. The testing programs

Each elementary function has its own testing routine, with a standardized name[5]. These routines are distributed as part of the package, to check the installation. Each testing routine evaluates interval inclusions at

a) natural special points of the function,

b) crossover points and extreme points in the argument reduction and approximations,

c) points near the limits of the floating point system, and

d) 20 intervals shaped from preset pseudorandom numbers according to the particular function and the particular machine arithmetic.

Interval output is printed along with point output from Fortran-supplied elementary functions, if they are available. The evaluation and printing for the elementary functions with one argument is standardized with a single, shared routine EVAL.[6] Summaries of correctness and accuracy are provided.

Without assuming availability of higher precision or of optimal accuracy of the Fortran intrinsic elementary functions, rigorous testing is difficult. However, using the same input data for each machine is inconsistent with testing the behavior of the algorithm near the extremes of a particular number system. We have opted to program the tests so that the input is scaled according to the individual machine parameters. The tests are then successful if the point evaluations at the end points are within the interval values. An unsuccessful test means either that the interval routines are not functioning correctly or that the Fortran-supplied elementary function routines are inaccurate.

On systems with multiple precision elementary function libraries, the multiple precision functions can be easily substituted for the standard functions, to give a more rigorous test of the routines. In fact, in our own validation efforts, we independently computed more accurate values and ranges.

---

[5]For example, TICOS tests ICOS and TIEXP tests IEXP.

[6]The functions TIIPWR and TINPWR, with two arguments, use similar routines EVAL2 and EVAL3, and various modifications are used for testing the utility routines.

4. Future enhancements

Our hope is that INTLIB will be both widely used in a production mode and used as a template for efficient, machine-specific versions. Besides this, our own future efforts may include

1. providing Fortran 90 code to access the library through operator overloading,
2. providing versions of the package for IEEE arithmetic machines and for compilers with libraries of Fortran intrinsic functions which are optimally accurate,
3. providing extended interval arithmetic routines,
4. providing complex interval arithmetic routines, and
5. further improving the elementary functions, such as with an accurate dot product. We may make use of other packages, such as that presently being developed by W. Walter.

Another, more ambitious extension may be to provide routines for slope arithmetic [13].

5. Credits

Besides my co-authors, I wish to acknowledge George Corliss at Marquette University. Many discussions with him both encouraged me and allowed me to improve the package. I also wish to acknowledge my one-time student Rebecca Yun and Abdulhamid Awad, an undergraduate student at the University of Houston-Downtown who worked on several of the routines.

References

1. Aberth, O. and Schaefer, M. *Precise computation using range arithmetic, via C++.* Preprint, Dept. Math., Texas A&M Univ., College Station, 1990.
2. Bleher, J. H., Rump, S. M., Kulisch, U., Metzger, M., Ullrich, C. and Walter, W. *Fortran-SC – a study of a fortran extension for engineering and scientific computation with access to ACRITH.* Computing **39** (2) (1987), pp. 93–110.
3. Brown, W. S. *A simple but realistic model of floating-point computation.* ACM Trans. Math. Software **7** (4) (1981), pp. 445–480.

4. Bundy, A. *A generalized interval package and its use for semantic checking.* ACM Trans. Math. Software **10** (4) (1984), pp. 397–409.

5. Clemmesen, M. *Interval arithmetic implementations using floating point arithmetic.* ACM SIGNUM News **19** (4) (1984).

6. Corliss, G. F. *Proposal for a basic interval arithmetic subroutines library (BIAS).* Preprint, 1990 (available electronically from georgec@boris.mscs.mu.edu).

7. Ely, J. *Prospects for using variable precision interval software in C++ for solving some contemporary scientific problems.* Ph.D. dissertation, Department of Computer Science, Ohio State University, 1990.

8. Fong, K. W., Jefferson, T. H. and Suyehiro, T. *Guide to the SLATEC common mathematical library.* Preprint, Lawrence Livermore National Laboratory, 1990.

9. Herzberger, J. *ALGOL-60 procedures evaluating standard functions in interval analysis.* Computing **5** (4) (1970), pp. 377–384.

10. Jiang, Hong. *Implementing a basic interval arithmetic subroutines library.* Technical report no. 333, Dept. of Math., Stat., and Comp. Sci., Marquette University, Milwaukee, 1990.

11. Jüllig, H.-P. *Algorithmen mit Ergebnisverification mit C++/2.0.* Preprint, Technische Universität Hamburg-Harburg, 1991.

12. Kearfott, R. B. and Novoa, M. *INTBIS, a portable interval Newton/bisection package (Algorithm 681).* ACM Trans. Math. Software **16** (2) (1990), pp. 152–157.

13. Krawczyk, R. and Neumaier, A. *Interval slopes for rational functions and associated centered forms.* SIAM J. Numer. Anal. **22** (3) (1985), pp. 604–616.

14. Krishchuk, V. N., Vasilega, N. M. and Kozina, G. L. *The experience of applying interval calculation for the analysis of the strength characteristics of radio-electronic devices.* Preprint, 1992.

15. Moore, R. E. *Methods and applications of interval analysis.* SIAM, Philadelphia, 1979.

16. Rall, L. B. *An introduction to the scientific computing language Pascal-SC.* Computers and mathematics with applications **14** (1) (1987), pp. 53–59.

17. Rump, S. M. *Accuracy control and estimation, self-validating systems and software environments for scientific computation.* IFIP Trans. A, Comput. Sci. Technol. (Netherlands) **A-2** (1992), pp. 49–56.

18. *Maple, version 4.2.1.* Brooks/Cole, Monterey, California, 1990.

19. *IEEE standard for binary floating point arithmetic (ANSI/IEEE 754-1985).* IEEE, New York, 1985.

20. Yohe, J. M. *Software for interval arithmetic: a reasonably portable package.* ACM Trans. Math. Software **5** (1) (1979), pp. 50–53.