INTERVAL_ARITHMETIC: A Fortran 90 Module for an Interval Data Type

R. BAKER KEARFOTT
Department of Mathematics
University of Southwestern Louisiana

Interval arithmetic is useful in *automatically verified computations*, that is, in computations in which the algorithm itself rigorously proves that the answer must lie within certain bounds. In addition to rigor, interval arithmetic also provides a simple and sometimes sharp method of bounding ranges of functions for global optimization and other tasks.

Convenient use of interval arithmetic requires an interval data type in the programming language. Although various packages supply such a data type, previous ones are machine-specific, obsolete and unsupported, for languages other than Fortran, or commercial. The Fortran 90 module `INTERVAL_ARITHMETIC` provides a portable interval data type in Fortran 90. This data type is based on two double precision real Fortran storage units. Module `INTERVAL_ARITHMETIC` uses the FORTRAN 77 library INTLIB (ACM TOMS Algorithm 737) as supporting library. The module has been employed extensively in the author's own research.

Categories and Subject Descriptors: G1.0 [**Numerical Analysis**]: General—*Computer arithmetic*; *Error analysis*; *Numerical Algorithms*

General Terms: Programming Languages, Portability

Additional Key Words and Phrases: Interval arithmetic, operator overloading

1. INTRODUCTION AND SYNOPSIS

Interval arithmetic, when practical, allows rigor in scientific computations, and can provide tests of correctness of hardware, compilers, and function libraries for floating point computations. Interval arithmetic can also be useful in sensitivity analysis. Additionally, since interval arithmetic provides rigorous bounds on the ranges of functions, it is appropriate in applications in which Lipschitz constants or bounds on moduli of continuity are required. In fact, interval arithmetic is a convenient, and sometimes the sharpest, means of obtaining such information in algorithms. For example, evaluation of $f(x) = x^4 + x^3 + x$ over the interval $[1, 2]$ results in $[1, 16] + [1, 8] + [1, 2] = [3, 26]$, which happens to be the range[1] of f over

[1] Conditions under which the interval value is the range can be found in discussions of the properties of interval arithmetic, such as in [Neumaier 1990]. In general, interval values are just bounds on the range.

Author's address: R. B. Kearfott, Department of Mathematics, University of Southwestern Louisiana, USL Box 4-1010, Lafayette, LA 70504-1010 USA. Email: rbk@usl.edu
This work was supported in part by National Science Foundation grant CCR-9203730.

```
C  This standard FORTRAN-77 routine uses INTLIB directly.
      PROGRAM TEST_INTLIB

C  Intervals are represented as double precision arrays
C  with two elements --
      DOUBLE PRECISION X(2), F(2)
      DOUBLE PRECISION TMP2(2), TMP3(2)

C Initialize machine constants and interval constants used in
C the standard functions --
      CALL SIMINI

C The range over [1,2] will be computed --
      X(1) = 1D0
      X(2) = 2D0

C Round out in case the decimal-to-binary conversion is
C not exact --
      CALL RNDOUT(X,.TRUE.,.TRUE.)

C Compute X**4 + X**3 + X --
      CALL POWER(X,4,TMP2)
      CALL POWER(X,3,TMP3)
      CALL ADD(TMP2,TMP3,TMP2)
      CALL ADD(TMP2,X,F)

      WRITE(6,*) F(1), F(2)

      END
```

Fig. 1. Interval evaluation in FORTRAN 77 using INTLIB

$[1, 2]$.

The INTLIB package, ACM TOMS Algorithm 737, consists of a portable FORTRAN 77 library for interval arithmetic and standard functions, so that interval values can be obtained for most functions that can be calculated as Fortran subroutines. However, direct use of INTLIB is cumbersome. For example, an interval computation of $f([1, 2])$ for $f(x) = x^4+x^3+x$, using INTLIB directly, would require a program similar to that in Figure 1.

Fortunately, operator overloading in Fortran 90 allows a computation equivalent to Figure 1 to be expressed in simple syntax. With module `INTERVAL_ARITHMETIC`, the computation can be expressed as in Figure 2. The actual machine operations corresponding to Figure 2 will consist of a series of calls to INTLIB routines as in Figure 1, with the order of these calls depending on the particular compiler.

There are numerous previous packages for interval arithmetic, including those based on the operator overloading technique here; the oldest is the Augment precompiler [Crary 1976] in conjunction with a FORTRAN 66, somewhat transportable interval library [Yohe 1979]. More recently, several C++ packages, including the package mentioned in [Leclerc 1993], the package of [Knüppel 1994], and C-XSC

```
PROGRAM EVALUATE_EXAMPLE

!  This standard Fortran 90 routine
!  evaluates X**4 + X**3 + X over [1,2].

USE INTERVAL_ARITHMETIC
      TYPE(INTERVAL) X, F
      CALL SIMINI

      X = INTERVAL(1,2)
      F = X**4 + X**3 + X
      WRITE(6,*) F

END PROGRAM EVALUATE_EXAMPLE
```

Fig. 2. Fortran 90 Interval evaluation with module `INTERVAL_ARITHMETIC`

[Klatte et al. 1993] have been developed. C-XSC, as well as Pascal-XSC [Hammer et al. 1993], are commercial languages available on a variety of machines. An extension of FORTRAN 77, ACRITH-XSC [Walter 1993a] is an IBM product running under the VM operating system. The preprocessor TPX [Husung 1989] translates to Turbo-Pascal, and an alternate set of Fortran 90 modules [Walter 1993b] is under development. However, our module `INTERVAL_ARITHMETIC`, based on a minimalist philosophy, is the first polished, publicly available and portable access to an interval data type. It is reasonably efficient and practical in a variety of applications, and is simple to maintain and extend.

`INTERVAL_ARITHMETIC` is the most universally useful portion of the package described in [Kearfott 1995].

In module `INTERVAL_ARITHMETIC`, the endpoints of the interval are represented as double precision Fortran variables. The module should be useful to persons requiring portable interval arithmetic in standard Fortran 90, with reasonable, but not optimal, accuracy and efficiency.

2. SYNTAX DEFINED IN INTERVAL_ARITHMETIC

The module `INTERVAL_ARITHMETIC` defines the four elementary operations ($+$, $-$, $*$, and $/$), as well as negation (i.e. unary minus), on interval data types. Mixed-mode operations are allowed only between intervals and double precision, or between intervals and integer numbers[2]. Exponentiation $**$ is defined for interval-to-interval, interval-to-integer, double precision-to-interval, interval-to-double precision, and integer-to-interval.

The module defines the generic names

`ACOS`, `ACOT`, `ASIN`, `ATAN`, `COS`, `COT`, `EXP`, `LOG`, `SIN`,
`SINH`, `SQRT`, and `TAN`, (1)

[2] This is because intervals are stored and rounded out as double precision numbers. Arithmetic between a single-length real and interval would first involve converting the single-length real to double precision, then rounding according to the double precision machine epsilon. However, the single-precision value may be only accurate to single precision, so the rounded interval would not contain the theoretical value. Rigor would thus be sacrificed.

Table I. Special interval functions in module INTERVAL_ARITHMETIC

Syntax	Corresponding INTLIB routine	function
`Z = ABS(X)`	IVLABS	$z \leftarrow \{\lvert x\rvert, x \in x\}$
`R = WID(X)`	IWID	$r \leftarrow \overline{x} - \underline{x}$
`R = MID(X)`	IMID	$r \leftarrow (\overline{x} - \underline{x})/2$
`R = MAG(X)`	INTABS	$r \leftarrow \max\{\lvert\underline{x}\rvert, \lvert\overline{x}\rvert\}$
`Z = MAX(X,Y)`	–	$z \leftarrow [\max\{\underline{x}, \underline{y}\}, \max\{\overline{x}, \overline{y}\}]$
`Z = MIN(X,Y)`	–	$z \leftarrow [\min\{\underline{x}, \underline{y}\}, \min\{\overline{x}, \overline{y}\}]$
`R = MIG(X)`	IMIG	"mignitude:" $r \leftarrow \min\{\lvert\underline{x}\rvert, \lvert\overline{x}\rvert\}$ if $0 \notin x$, and $r \leftarrow 0$ otherwise

each of which returns bounds on the range, to within roundout error, of the corresponding point-valued function. In each case, a corresponding INTLIB routine [Kearfott et al. 1994] is used, with `TAN` and `COT` employing the INTLIB routines `ISIN` and `ICOS`. (Here, "roundout error" is the excess width of the interval result caused by rounding the lower endpoint down and the upper endpoint up after each operation in a series of interval computations.)
Additionally, the special interval functions in Table I are defined[3]. The definitions of `ABS` and `MAG` vary slightly from those in ACRITH-XSC [Walter 1993a] and other languages with interval data types: the function `ABS` in ACRITH-XSC corresponds to the `INTERVAL_ARITHMETIC` function `MAG`, and is consistent with use of $\lvert \circ \rvert$ throughout the literature on interval computations. However, when coding objective functions for interval branch and bound algorithms, it is more natural for `ABS` to return the range of $\lvert \circ \rvert$.

Finally, `INTERVAL_ARITHMETIC` defines the operators exhibited in Table II. The binary operations in Table II that correspond to Fortran intrinsic relational operators on default data types (i.e. `.LT.`, `.GT.`, `.LE.`, `.GE.`, `.NE.`, and `.EQ.`) admit mixed mode operations between intervals and double precision or integers, while either or both arguments of `.CH.` may be double precision. Also, note that, in standard Fortran 90, the relational operations may be given either by `.LT.`, `.GT.`, `.LE.`, `.GE.`, `.NE.`, and `.EQ.` or by `<`, `<=`, `>`, `>=`, `/=`, and `==`, respectively. For example, the expression "`A < B`", where `A` and `B` are intervals, is equivalent to "`A.LT.B`".

With the exception of `ABS` and `MAG`, the operators and functions in the list (1) and Tables I and II act similarly to those in ACRITH-XSC.

The interval data type in module `INTERVAL_ARITHMETIC` is a user-defined sequenced structure with two components (e.g. the interval `X` has components `X%LOWER` and `X%UPPER`). In contrast, the interval data type in INTLIB consists of a singly-dimensioned array with two elements, e.g. the lower bound `X(1)` and the upper bound `X(2)`. In a non-portable version of `INTERVAL_ARITHMETIC`, the Fortran 90 interval data type was associated directly and efficiently with corresponding INTLIB intervals through subroutine calls. This technique worked, since the storage sequence of the Fortran 90 data type is the same as the storage sequence of an interval in INTLIB. However, this implicit equivalencing is non-standard, and cannot be expected to be possible on all systems, i.e., it cannot be assumed that an interval

[3]In Table I, the Fortran 90 interval variables X, Y and Z are identified with intervals $x = [\underline{x}, \overline{x}]$, $y = [\underline{y}, \overline{y}]$ and $z = [\underline{z}, \overline{z}]$, while r is identified with the double precision variable R.

Table II. Relational operators defined in module `INTERVAL_ARITHMETIC`

Syntax	Corresponding INTLIB routine	function
`Z = X.IS.Y`	ICAP	$z \leftarrow x \cap y$
`Z = X.CH.Y`	IHULL	$z \leftarrow [\min\{\underline{x},\underline{y}\},\max\{\overline{x},\overline{y}\}]$
`X.SB.Y`	IILEI	`.TRUE.` if $x \subseteq y$
`X.SP.Y`	IILEI	`.TRUE.` if $x \supseteq y$
`X.DJ.Y`	IDISJ	`.TRUE.` if $x \cap y = \emptyset$
`R.IN.X`	IRLEI	`.TRUE.` if $r \in x$
`Y.IN.X`	IRLEI	`.TRUE.` if y is in the interior of x
`Y.LT.X`	–	`.TRUE.` if $\overline{y} < \underline{x}$
`Y.GT.X`	–	`.TRUE.` if $\underline{y} > \overline{x}$
`Y.LE.X`	–	`.TRUE.` if $\overline{y} \geq \underline{x}$
`Y.GE.X`	–	`.TRUE.` if $\underline{y} \geq \overline{x}$
`Y.NE.X`	–	`.TRUE.` if $y \neq x$ (set inequality)
`Y.EQ.X`	–	`.TRUE.` if $y = x$ (set equality)

as defined by `INTERVAL_ARITHMETIC` can be passed directly as an actual argument to an INTLIB routine. For this reason, the module `INTERVAL_ARITHMETIC` uses the (less efficient) technique of actually moving the data between the data types. (`INTERVAL_ARITHMETIC` also contains Fortran 90 versions of some low-level INTLIB routines, for efficiency.)

Assignment of interval values can be done using the default Fortran 90 assignment to structures, e.g. `X = INTERVAL(.3D0,.3D0)`. However, this scheme is not recommended, since the values may not be properly rounded when converted from character strings to floating point numbers. The function `IVL`, which accepts either one or two double precision or integer arguments, causes the internally-stored result to be rigorously rounded[4]. Also, the module `INTERVAL_ARITHMETIC` overloads assignment (`=`), so that, when an integer or double precision number is assigned to an interval, the result is properly rounded. For example, `X=IVL(.3D0)`, `X=IVL(.3D0,.3D0)`, and `X=.3D0` each[5] cause a properly rounded inclusion of the number .3 to be stored in the interval variable `X`.

The left and right endpoints of an interval `X` are double precision numbers accessed as `X%LOWER` and `X%UPPER`, respectively. These expressions may occur on either side of an assignment statement. The lower and upper endpoints of an interval may also be accessed in an expression (but not on the left side of an assignment statement) with `INF(X)` and `SUP(X)`, respectively. Implicit conversion between intervals and other data types is not allowed: conversions are done with `INF`, `SUP`, and `MID`.

3. INSTALLATION AND USE

The system consists of the following components.

[4]assuming that INTLIB has been installed properly to take account of the conversion errors; see [Kearfott et al. 1994].

[5]IVL also accepts integer arguments, such as `X=IVL(3)`, `X=IVL(3,3D0)`, or `X=3`, where `X` is an interval variable. However, many machines can store small integers exactly in floating point formats; for such machines, outward rounding is not necessary, and `X=INTERVAL(3,3)` would be more logical.

IVL_DEF:	a small Fortran 90 module that defines the interval data type;
INTERVAL_ARITHMETIC:	the Fortran 90 module, approximately 866 lines, that defines the syntax of §2;
TEST_INTERVAL_ARITHMETIC:	a short program to test proper installation of the system;
TEST_INTERVAL_SYSTEM:	a lengthier program for more extensive testing[6] of the installation.
SAMPLE.OUT:	sample output from TEST_INTERVAL_SYSTEM.
TINYSMPL.OUT	sample output from TEST_INTERVAL_ARITHMETIC
INTLIB:	the FORTRAN 77 library that provides the actual elementary operations and standard functions [Kearfott et al. 1994].
D1MACH:	a portable, Fortran 90 version of this SLATEC routine, that can replace the version in INTLIB.

Installation of the system is in the following order.

(1) Install INTLIB according to the instructions of [Kearfott et al. 1994].
(2) Compile IVL_DEF in the directory for Fortran 90 modules.
(3) Compile INTERVAL_ARITHMETIC in the directory for Fortran 90 modules, making sure the compiler has access to the module IVL_DEF.
(4) Compile and link TEST_INTERVAL_ARITHMETIC, making sure the compiler has access to the modules IVL_DEF and INTERVAL_ARITHMETIC; the linker should have access to the object code for IVL_DEF and INTERVAL_ARITHMETIC, as well as to the object library for INTLIB.
(5) Run TEST_INTERVAL_ARITHMETIC (or TEST_INTERVAL_SYSTEM) to check proper installation.

The program TEST_INTERVAL_ARITHMETIC provides a template for use of the system. The syntax is as in §2.

The program TEST_INTERVAL_SYSTEM provides a somewhat more exhaustive test of module INTERVAL_ARITHMETIC. The program TEST_INTERVAL_SYSTEM causes each executable statement in module INTERVAL_ARITHMETIC to be executed, and checks that operators and functions give the proper results. Since most of the results are intervals with integer endpoints, the exact results are input explicitly as integer constants, then converted, e.g. $[1, 2]$ is input as INTERVAL(1,2). The computed results are compared to the exact results so input, and an error is flagged if the computed results do not contain the exact results. If no errors are flagged, then a message stating that module INTERVAL_ARITHMETIC appears to be installed correctly is printed at the end of the output. Output, to the file INTARITH.OUT, also includes printouts of the results of the explicit conversion routine IVL, results of rounding out near the underflow threshold, and results of precipitating two types of errors that INTLIB catches. Examining the less significant digits in the printed

[6] Note that INTLIB comes with its own set of tests, for the arithmetic itself. The tests provided with the module INTERVAL_ARITHMETIC check the syntax defined in INTERVAL_ARITHMETIC.

results of `IVL` and the rounding out may help the installer to determine that the simulated directed rounding is operating correctly. The output should correspond *roughly* to the sample file `SAMPLE.OUT` that is supplied with the algorithm, except that the endpoints may differ slightly depending on the characteristics of the arithmetic and on the details of the particular implementation of binary-to-decimal conversion for formatted output. This should not be a problem, assuming INTLIB, and, in particular, `D1MACH`, have been installed correctly. Correct simulated directed rounding is important for mathematical rigor in the computations.

If some of the tests in `TEST_INTERVAL_SYSTEM` fail, it can be due to inaccurate conversion of the character strings representing integer and decimal floating point constants, either in the program `TEST_INTERVAL_SYSTEM` or in the INTLIB routine `SIMINI`. It is assumed in `TEST_INTERVAL_SYSTEM` that such conversions give the closest machine number to the decimal string representation, and it is assumed in INTLIB that such conversions are of the same accuracy as the four basic arithmetic operations. Failure of the tests in `TEST_INTERVAL_SYSTEM` can also be due to an insufficient number of decimal digits in the representation, if `DOUBLE PRECISION` on the target machine corresponds to more than 30 digits.

Sample output to the less exhaustive test program `TEST_INTERVAL_ARITHMETIC` is in the file `TINYSMPL.OUT`, while running `TEST_INTERVAL_ARITHMETIC` produces the file `TEST_F90_INTARITH.OUT`. The output to `TEST_INTERVAL_ARITHMETIC` will be the same on many systems.

The source code to `INTERVAL_ARITHMETIC` is meant to be readable and modifiable by the user, although a conservative approach to modifications is recommended. In particular, it is hoped that this module will provide a basis for standardization of syntax for interval computations in Fortran.

4. ADDITIONAL ASSUMPTIONS AND COMMENTS

In addition to the assumptions for which INTLIB is rigorous, it is assumed that 0 and 1 are represented exactly in double precision, and that conversion from integer to double precision (e.g. with `DBLE(I)`) yields an exact representation. The integer-to-double conversions are used in the mixed mode binary operations; if such conversions are not exact, then the conversion should be done explicitly using the function `IVL`. Representations of 0 and 1 are assigned explicitly in the module to the parameter variables `ZERO` and `ONE`.

The four elementary operations, unary negation, and the routine `RNDOUT` were redefined in this module (i.e. the corresponding INTLIB routines are not always used) for a combination of portability and efficiency considerations. The corresponding INTLIB routines were used as templates.

Finally, as with INTLIB, the author has a non-portable version, using assembler language for the directed rounding as in [Knüppel 1994], available for Sun Sparc systems, that runs roughly twice as fast on those systems.

ACKNOWLEDGMENTS

I wish to thank my students Kaisheng Du, Xiaofa Shi and Shiying Ning, as well as Claire Adjiman and George Corliss, who have used the system over a period of several years, and have provided valuable suggestions. I also wish to thank John

Reid for the extensive help he gave. Finally, I thank the referee, whose extensive comments led to substantial changes.

REFERENCES

CRARY, F. 1976. The AUGMENT precompiler. Technical Report 1470, MRC, University of Wisconsin, Madison.

HAMMER, R., NEAGA, M., AND RATZ, D. 1993. PASCAL-XSC, New concepts for scientific computation and numerical data processing. In ADAMS, E. AND KULISCH-U. (Ed.), *Scientific computing with automatic result verification*, New York, etc., pp. 15–44. Academic Press.

HUSUNG, D. 1989. Precompiler for scientific computation (TPX). Technical Report 91.1, Inst. for Comp. Sci. III, Technical University Hamburg–Harburg.

KEARFOTT, R. B. 1995. A Fortran 90 environment for research and prototyping of enclosure algorithms for constrained and unconstrained nonlinear equations. *ACM Trans. Math. Software 21,* 1 (March), 63–78.

KEARFOTT, R. B., DAWANDE, M., DU, K.-S., AND HU, C.-Y. 1994. Algorithm 737: INTLIB: A portable FORTRAN 77 interval standard function library. *ACM Trans. Math. Software 20,* 4 (December), 447–459.

KLATTE, R., KULISCH, U., WIETHOFF, A., LAWO, C., AND RAUCH, M. 1993. *C-XSC A C++ Class Library for Extended Scientific Computing*. Springer-Verlag, New York.

KNÜPPEL, O. 1994. PROFIL/BIAS — A fast interval library. *Computing 53*, 277–287.

LECLERC, A. 1993. Parallel interval global optimization in C++. *Interval Computations 1993,* 3, 148–163.

NEUMAIER, A. 1990. *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge, England.

WALTER, W. V. 1993a. ACRITH-XSC: A Fortran-like language for verified scientific computing. In ADAMS, E. AND KULISCH, U. (Ed.), *Scientific Computing with Automatic Result Verification*, New York, etc., pp. 45–70. Academic Press.

WALTER, W. V. 1993b. FORTRAN-XSC: A portable Fortran 90 module library for accurate and reliable scientific computing. *Computing (Suppl.) 9*, 265–286.

YOHE, J. M. 1979. Software for interval arithmetic: A reasonably portable package. *ACM Trans. Math. Software 5,* 1 (March), 50–53.