2

SOFTWARE ENVIRONMENTS

This chapter deals with implementations of the interval arithmetic concepts described in Chapter 1. Such software is, in turn, used in later chapters to construct higher-level software systems to solve nonlinear systems of equations and for global optimization. A `FORTRAN-77` library and a set of Fortran 90 modules built upon this library are described here. Although some information here is specific, underlying principles and philosophy are emphasized, and the packages are readily available, free of cost, to readers.

Alternate packages are also outlined.

2.1 INTLIB

The `FORTRAN-77` package INTLIB is *ACM Transactions on Mathematical Software* Algorithm 737 [123], and is thus available over NETLIB [50, 68], either via FTP (ibid.) or over the World Wide Web[1]. INTLIB consists of standard `FORTRAN-77` subroutines and functions for the elementary interval operations, some of the Fortran standard functions, and certain logical and utility operations. INTLIB achieves portability with simulated directed rounding, although minor modifications to INTLIB allow true directed rounding, where available.

Some simplification of INTLIB would be possible if it were rewritten with Fortran 90 constructs. However, INTLIB is presently directly accessible through Fortran 90, so its ease of use would not be affected.

[1] The address at the time of writing is `http://www.netlib.org/index.html`

Details of the construction, installation, and use of INTLIB that do not appear below can be found in [124], [123], and in the instructions provided with the `FORTRAN-77` source.

2.1.1 Philosophy and Design Goals

The goals and design of INTLIB are consistent with the philosophy stated with the original Level-1 BLAS routines[2] [151]. That is to say, the routines are meant to be well-documented, generally available, production quality, and totally portable. Moreover, the routine names and argument lists are meant to provide a pattern upon which more efficient machine-specific versions can be based. With this scheme, users can transfer application programs from one machine to another, and the program, without modification, will execute efficiently on a variety of machines. An attempt was made in INTLIB to conform to a proposal of Corliss [37].

In addition, the routines in INTLIB are meant to be user-readable, thus providing instruction in the underlying methods and, possibly, allowing enough understanding for the user to improve them. The contents were selected to provide a core set of clearly needed routines, without large numbers of superfluous functions. When there was ambiguity of mathematical interpretation, such as branch points (leading to disconnected ranges, etc.), a routine was not included.

2.1.2 Simulated Directed Rounding

Conceptually simple, simulated directed rounding is a device with which rigorous enclosures to floating point results may be obtained in **`FORTRAN-77`** and other languages that do not have access to true directed rounding, or on machines without hardware-supported directed rounding. The only requirement is that arithmetic parameters, such as the accuracy of an elementary operation, are known. For example, suppose $\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{y}$ is to be computed. Also suppose that the largest relative spacing ϵ_m between numbers[3], as well as the number

[2] although INTLIB is more like a "Level-0" BLAS, since INTLIB deals with scalar interval operations, rather than vector operations, as the Level-1 BLAS, or matrix-vector operations, as the Level-2 BLAS.

[3] For example, in a hypothetical decimal machine with numbers of the form $0.\mathrm{xxx} \times 10^{\mathrm{X}}$, the largest relative spacing between numbers is $0.101 \times 10^1 - 0.100 \times 10^1 = 10^{-2}$.

of units M in the last place by which the result of $+$, $-$, $\times$ or $\div$ can be in error, are known. Then the following procedure can be applied.

Algorithm 6 (Simplified example: addition with simulated directed rounding. See INTLIB routines `ADD` and `RNDOUT`.)

INPUT: $\boldsymbol{x}$ with $\underline{x} > 0$ and $\boldsymbol{y}$ with $\underline{y} > 0$. (The other cases are analogous; see the listing of `RNDOUT`.)

OUTPUT: $\boldsymbol{z}$, a machine interval containing the exact interval sum $\boldsymbol{x} + \boldsymbol{y}$.

1. *Compute* $\underline{z} = \underline{x} + \underline{y}$ *and* $\overline{z} = \overline{x} + \overline{y}$ *with the computer's floating point arithmetic.*
2. IF $\underline{z} \neq 0$ *and the following computations would neither underflow nor overflow*

   THEN

   (a) $\underline{z} \leftarrow (1 - \mathrm{M}\epsilon_m)\underline{z}$

   (b) $\overline{z} \leftarrow (1 + \mathrm{M}\epsilon_m)\overline{z}$

   ELSE

   Do a special computation for rigor.

   END IF

End Algorithm 6

Different formulas must be used when part or all of $\boldsymbol{x}$ or $\boldsymbol{y}$ is less than 0, and special consideration must be given for rigor when underflow or overflow would occur. In practice, Step 2 is performed after each elementary operation (i.e. after performing Step 1 with op replacing $+$, for op $\in \{+, -, \times, \div\}$). The INTLIB routine `RNDOUT` performs Step 2.

For details and clarification, the `FORTRAN-77` source for `RNDOUT` should be consulted directly. This routine is meant to be comprehensible by the user.

Within INTLIB, calls to `RNDOUT` may be replaced by similar calls to an assembler language routine, such as that described in [135]. This results in slightly

Name	function
ADD	interval addition: $\boldsymbol{z} \leftarrow \boldsymbol{x} + \boldsymbol{y}$
CANCEL	cancellation subtraction as explained on page 5
IDIV	ordinary interval division $\boldsymbol{z} \leftarrow \boldsymbol{x}/\boldsymbol{y}$, $0 \notin \boldsymbol{y}$
MULT	interval multiplication $\boldsymbol{z} \leftarrow \boldsymbol{x}\boldsymbol{y}$
RNDOUT	simulated directed rounding, as in §2.1.2
SCLADD	addition of an interval and a real number
SCLMLT	multiplication of an interval and a real number
SUB	interval subtraction $\boldsymbol{z} \leftarrow \boldsymbol{x} - \boldsymbol{y}$

Table 2.1 Elementary arithmetic routines in INTLIB

narrower result intervals and slightly faster execution. In fact, this was done for Sun SPARC computers (six low-level routines in INTLIB were changed), with approximately a factor of two increase in speed[4].

2.1.3 Contents of INTLIB

INTLIB is organized into subsets of 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. Table 2.1 lists the arithmetic routines in INTLIB, while Table 2.2 lists the INTLIB implementations of standard functions.

INTLIB also contains utility functions, whose mathematical functions are elementary, but whose existence is justified by the error-checking they provide and their usefulness in operator overloading procedures. Table 2.3 lists these. Within this table, the arguments to a function are $\boldsymbol{x}$ and $\boldsymbol{y}$ for binary interval functions, $\boldsymbol{x}$ for unary interval functions, x and $\boldsymbol{x}$ for binary real / interval functions, and x and y for binary real / real functions. The results are $\boldsymbol{z}$ for interval-valued functions, `.TRUE.` / `.FALSE.` for logical-valued functions, and r for floating-point-valued functions. All intervals are considered to be closed.

Finally, INTLIB contains error-printing routines, a routine to initialize global constants and parameters, a suite of routines for testing, and sample test output. For details, see [123].

[4] Details are available from the author, or consult his home page on the World Wide Web at `ftp://interval.usl.edu/pub/interval_math/www/kearfott.html`.

Name	function
`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`	interval hyperbolic sine
`ISQRT`	interval square root
`POWER`	integer power of an interval

Table 2.2 Standard function routines in INTLIB

Name	function
`ICAP`	$\boldsymbol{z} \leftarrow \boldsymbol{x} \cap \boldsymbol{y}$ (Error signalled if empty)
`IDISJ`	`.TRUE.` if $\boldsymbol{x} \cap \boldsymbol{y} = \emptyset$
`IHULL`	$\boldsymbol{z} \leftarrow [\min\{\underline{x}, \underline{y}\}, \max\{\overline{x}, \overline{y}\}]$
`IILEI`	`.TRUE.` if $\boldsymbol{x} \subseteq \boldsymbol{y}$
`IILTI`	`.TRUE.` if $\underline{x} > \underline{y}$ and $\overline{x} < \overline{y}$
`IINF`	$r \leftarrow \underline{x}$
`IMID`	$r \leftarrow (\underline{x} + \overline{x})/2$ (computation in floating point)
`IMIG`	"mignitude:" $r \leftarrow \min\{\lvert\underline{x}\rvert, \lvert\overline{x}\rvert\}$ if $0 \notin \boldsymbol{x}$, and $r \leftarrow 0$ otherwise
`INEG`	negation with round out: $\boldsymbol{z} \leftarrow -\boldsymbol{x}$
`INTABS`	"magnitude:" $r \leftarrow \max\{\lvert\underline{x}\rvert, \lvert\overline{x}\rvert\}$
`IRLEI`	`.TRUE.` if $x \in \boldsymbol{x}$
`IRLTI`	`.TRUE.` if $x > \underline{x}$ and $x < \overline{x}$
`ISUP`	$r \leftarrow \overline{x}$
`IVL1`	construct an interval from a point: $\underline{z} \leftarrow x$, $\overline{z} \leftarrow x$
`IVL2`	construct an interval from two points: $\underline{z} \leftarrow x$, $\overline{z} \leftarrow y$
`IVLABS`	range of $\lVert \circ \rVert$ over an interval: $\boldsymbol{z} \leftarrow \{\lvert x\rvert, x \in \boldsymbol{x}\}$
`IVLI`	assign one interval to another: $\boldsymbol{z} \leftarrow \boldsymbol{x}$
`IWID`	upwardly rounded width: $r \leftarrow \overline{x} - \underline{x}$

Table 2.3 Utility functions in INTLIB

2.1.4 An Example

Figure 2.1 contains a `FORTRAN-77` routine that computes and prints rigorous bounds on the range[5] over $[1,2]$ of the function $f(x) = x^4+x^3+x$ of Example 1.3 on page 37. The operations in this program are done in the order implied by Table 1.2; `TMP2` and `TMP3` represent the intermediate variables x_2 and x_3 in the first two computations. Note, however, that separate storage locations are not needed for x_4 and x_5, if all that is desired[6] is an interval enclosure $\boldsymbol{f}(\boldsymbol{x})$.

Figure 2.1 illustrates various aspects of INTLIB. Nonetheless, it is **not** recommended that INTLIB generally be used in this way, if a Fortran 90 compiler is available. Programming with an interval data type, as in §2.2.1 below, is, with a few exceptions, a more desirable way to program interval operations.

2.1.5 Exercises

1. Obtain and install INTLIB. Use INTLIB to evaluate the range (to within roundout error) of

$$\phi(x) = \left[\sin\left(\pi x^3\right)\right]^2$$

over the interval $\boldsymbol{x} = [0.99, 1.01]$.

Recall that INTLIB may be obtained either through NETLIB at

`http://www.netlib.org/index.html`

or via anonymous FTP to

`interval.usl.edu`,

in the directory

`pub/interval_math/intlib`.

Hint: It is always necessary to call `SIMINI` *first. After* `SIMINI` *has been called, an interval enclosure for* π *is available as variable* PI *in the common block* `MTHCNS`.

2. Study the program `SIMINI`.

   (a) What assumptions are made here on the decimal to binary conversions of constants? How could you modify SIMINI to work with machine / compiler combinations for which these assumptions are not valid?

[5] Over $[1,2]$ and with this particular function, the bounds are the actual range, to within roundout error.

[6] However, see §7 below.

```
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
```

Figure 2.1 A `FORTRAN-77` program using INTLIB

(b) What assumptions are made on the word length of the machines upon which INTLIB is to be installed? What could you do to provide rigorous computation with INTLIB on machines with larger word lengths?

(c) What would you do to install INTLIB rigorously on Cray machines? *Hint: See the comments on Cray installation in [123].*

2.2 FORTRAN 90 INTERVAL AND CODE LIST SUPPORT

Throughout this section, it is assumed that the reader is familiar with Fortran 90. An excellent introduction, among others, is [22].

Several tools are available for conveniently accessing the routines in INTLIB through Fortran 90. These include an interval data type and a set of modules and functions for generating and interpreting code lists, as introduced in §1.4.5. Available freely, these will be called "`INTLIB_90`." A set of additional, higher-level tools to actually solve nonlinear equations and optimization problems, explained in §4.4 and §5.3 below, will be called "`INTOPT_90`."

2.2.1 An Interval Data Type

The interval data type is defined in a Fortran 90 module `INTERVAL_ARITHMETIC`. The module takes its support routines from INTLIB [123].

For example, `INTERVAL_ARITHMETIC` may be used for the same computation as in Figure 2.1; corresponding Fortran 90 code appears in Figure 2.2. The output to the program in Figure 2.2 should be similar to the output[7] to the program in Figure 2.1. However, the order in which the operations are done depends on how the particular compiler parses the statement

```
F = X**4 + X**3 + X,
```

and may not be the same as that in Table 1.2 or Figure 2.1.

The module `INTERVAL_ARITHMETIC` defines the four elementary operations (+, −, ∗, and /), as well as negation on interval data types (i.e. unary minus).

[7] namely, $[2.9999999999999982, 26.0000000000000391]$ or something approximately equal, depending on the compiler and the machine

```
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
```

Figure 2.2 Illustration of use of the interval data type

Mixed-mode operations are allowed only between intervals and double precision (or equivalent type), or between intervals and integer numbers[8]. Exponentiation ** is also defined for interval-to-integer, interval-to-interval, double precision-to-interval, and interval-to-double precision.

The module defines the generic names

$$\begin{array}{l}\texttt{ACOS, ACOT, ASIN, ATAN, COS, COT, EXP, LOG, SINH,}\\ \texttt{SIN, SQRT, }\text{and }\texttt{TAN,}\end{array} \tag{2.1}$$

each of which returns bounds on the range[9], to within roundout error, of the corresponding point-valued function. In each case, the corresponding INTLIB routine from Table 2.2 is used, with `TAN` and `COT` making use of the INTLIB routines `ISIN` and `ICOS`.

Additionally, the special interval functions in Table 2.4 are defined[10]. The definitions of `ABS` and `MAG` vary slightly from those in ACRITH-XSC [237] and

[8] This is because intervals are stored and rounded out as double precision numbers. Arithmetic between a real and interval would first involve converting the 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.

[9] These bounds are not tight in the sense of being the smallest machine intervals that enclose the actual range. This is because, for portability, only double precision arithmetic is used in INTLIB. However, the bounds are rigorous inclusions of the range, and are reasonably tight.

[10] In Table 2.4, 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.

Syntax	Corresponding INTLIB routine	function
`R = MAG(X)`	INTABS	$r \leftarrow \max\{\lvert\underline{x}\rvert, \lvert\overline{x}\rvert\}$
`R = WID(X)`	IWID	$r \leftarrow \overline{x} - \underline{x}$
`R = MID(X)`	IMID	$r \leftarrow (\overline{x} - \underline{x})/2$
`R = MIG(X)`	IMIG	"mignitude:" $r \leftarrow \min\{\lvert\underline{x}\rvert, \lvert\overline{x}\rvert\}$ if $0 \notin \boldsymbol{x}$, and $r \leftarrow 0$ otherwise
`Z = ABS(X)`	IVLABS	$\boldsymbol{z} \leftarrow \{\lvert x\rvert, x \in \boldsymbol{x}\}$
`Z = MAX(X,Y)`	—	$\boldsymbol{z} \leftarrow [\max\{\underline{x}, \underline{y}\}, \max\{\overline{x}, \overline{y}\}]$ (cf. Chapter 6)

Table 2.4 Special interval functions in module `INTERVAL_ARITHMETIC`

other languages with interval data types: in [237], the function `ABS` corresponds to the `INTERVAL_ARITHMETIC` function `MAG`, and is consistent with use of $|\circ|$ 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 $|\circ|$, as `ABS` does if `INTERVAL_ARITHMETIC` is used.

Finally, `INTERVAL_ARITHMETIC` defines the binary interval and logical-valued operators exhibited in Table 2.5. The binary operations in Table 2.5 that correspond to Fortran intrinsic operators on default data types (i.e. `.LT.`, `.GT.`, `.LE.`, `.GE.`, and `.NE.`) admit mixed mode operations between intervals and double precision or integers, while either or both arguments of `.CH.` may be double precision or integer.

With the exception of `ABS` and `MAG`, the operators and functions in the list (2.1) and Tables 2.4 and 2.5 act identically to those in ACRITH-XSC [237].

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 arguments, causes the internally-stored result to be rigorously rounded. Also, assignment (`=`) is overloaded in the module `INTERVAL_ARITHMETIC`, so that the result is properly rounded when an integer or double precision number is assigned to an interval. For example, `X = IVL(0.3D0)`, `X = IVL(0.3D0,0.3D0)`, and `X=0.3D0` each[11]

[11] `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

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

Table 2.5 Logical and interval operators in module `INTERVAL_ARITHMETIC`

cause a properly rounded inclusion to the number 0.3 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 can occur on either side of an assignment statement[12]. Additionally, `INF(X)` will return the lower bound on an interval `X`, and `SUP(X)` will return the upper bound[13] on an interval `X`. Implicit conversion from interval is not allowed: the only conversion from interval to floating point is through the functions `MID`, `INF`, and `SUP`.

An additional example, the program used to generate Table 1.3, appears in Figure 2.3.

point formats; for such machines, outward rounding is not necessary, and `X=INTERVAL(3,3)` would be more logical.

[12] The interval `X` is just a Fortran 90 sequenced structure with double precision components `X%LOWER` and `X%UPPER`. This is compatible with INTLIB, which stores intervals in `FORTRAN-77` as arrays with two elements.

[13] The routines `INF` and `SUP` were included since it is slightly easier with them than with `X%LOWER` and `X%UPPER` to make programs language-independent.

```
PROGRAM INTERVAL_NEWTON_ITERATION_1_D
   USE INTERVAL_ARITHMETIC
   IMPLICIT NONE

   DOUBLE PRECISION XP
   TYPE(INTERVAL) :: X, X_IMAGE
   INTEGER K
   DOUBLE PRECISION WIDTH_OF_X, OLD_WIDTH

   CALL SIMINI
      WIDTH_OF_X = 4
      X = IVL(1,2)

      DO K = 1,10000
         OLD_WIDTH = WIDTH_OF_X
         WIDTH_OF_X = IWID(X)
         XP = DBLE(X)
         WRITE(6,*) K, X, XP, WIDTH_OF_X, &
                    WIDTH_OF_X/OLD_WIDTH**2
         X_IMAGE = XP -  ( IVL(XP)**2 - IVL(4) ) / (2*X)
         IF(WIDTH_OF_X.LT.1D-11) EXIT
         X = X_IMAGE
      END DO

END PROGRAM INTERVAL_NEWTON_ITERATION_1_D
```

Figure 2.3 A univariate interval Newton method program, used to generate the data in Table 1.3