SHARE PROGRAM LIBRARY AGENCY

PROGRAM NUMBER

400 003

University of Miami

1365 MEMORIAL DRIVE - CORAL GABLES, FLORIDA
(305) - 284-6257

SHARE PROGRAM LIBRARY SUBMITTAL FORM

SHARE PROGRAM LIBRARY AGENCY
Triangle Universities Computation Center
Post Office Box 12076
Research Triangle Park, North Carolina
27709 USA
Attention: Mr. Joe Ragland

SPLA CONTROL NUMBER:

This form should be completed and submitted with the program package to the SHARE Program Library Agency at the address shown above. Standards and instructions for submitting programs are in the "SHARE Program Library Standards Manual".

(1) Program Number (to be filled in by SPLA) 360D-40.0.003

(2) System Type (machine) S/360

(3) Search Key ..... INTFORT - Interval Arithmetic Interpreter and Subroutine Package

(4) Programming Language ..... PL/1, FORTRAN, Assembler

(5) Author's Name and Address ..... D.P. Laurie
National Research Institute for Mathematical Sciences of CSIR, P.O. Box 395, Pretoria, South Africa

(6) Direct Inquiries to Name and Address (if different than Author) ..... Author

(7) Title of Program ..... INTFORT - Interval Arithmetic Interpreter and Subroutine Package

(8) Submitter's Installation Membership Code..... PQ

(9) Submitter's Own Program Identification and Suffix(Optional)... INTFORT

(10) Primary Subject Code..... 40.0

(11) Operating or Monitor System Required 360/OS

(12) New or Revision Code (if revision, show prior Program Number in Item 1).. N

(13) Year Completed..... 1972

(14) Date of Submittal..... 31.10.1973

(15) Documentation (number of original pages submitted)..... iii + 12

(16) Abstract (should contain sufficient information for a reader to determine the value of the program). Listed on the reverse side of this form are subjects which may serve as a guide for a descriptive abstract.

SHARE PROGRAM LIBRARY SUBMITTAL FORM

Subject Guide:

a. Purpose
b. Programming Language used
c. Version and modification level or release number
d. Field of application
e. Type of routine (main program, subroutine, etc.)
f. Specific description of machine requirements

ABSTRACT

The INTFORT interpreter converts explicit type declarations, assignment and arithmetic IF statements for interval variables to equivalent FORTRAN state= ments. A subroutine package for performing the interval arithmetic in single or double precision is provided. The interpreter is also suitable for use with any fancy arithmetic subroutines (e.g. multiprecision) that use synonomous subroutines for the arithmetic operations.

DISCLAIMER

Triangle Universities Computation Center (TUCC) serves solely as the distribution agent for contributed programs and does not test or maintain them. They are distributed essentially in the original form submitted by the author. Neither TUCC nor SHARE, INC., makes any warranty, expressed or implied, as to the documentation, function, or performance of the contributed programs.

(Please attach additional pages if necessary).....Total pages attached ____

Permission to Publish

"I hereby give the SHARE Program Library Agency permission to reprint, reproduce, and distribute this program."

(17) Signature of Submitter and Date D. P. Laurie 2·IX·1973

(18) Signature of Installation Addressee L. M. Delves

Magnetic Tape Key

Tape Volume Name : INTERV

This volume contains four files with standard labels [redacted] as follows: (TRACK AND DENSITY ARE AS ORDERED)

File 1 : DSNAME=LOADLIB : Contains unloaded PDS as created by IEHMOVE, including the interpreter INTFORT as well as the interval arithmetic subroutine library.
83 blocks of 800 bytes in EBCDIC.

File 2 : DSNAME=INTERP : Contains PL/1 source of interpreter.
38 blocks of 800 bytes in EBCDIC.

File 3 : DSNAME=SUBROUTS : Contains FORTRAN and Assembler source of interval arithmetic subroutines, preceded by //FORT.SYSIN DD * and //ASM.SYSIN DD * JCL statements respectively.
42 blocks of 800 bytes in EBCDIC.

File 4 : DSNAME=TESTPROG : Contains complete job deck (except JOB card) for a test program, including JCL, extended FORTRAN source and data cards.
11 blocks of 800 bytes in EBCDIC.

NATIONAL RESEARCH INSTITUTE FOR MATHEMATICAL SCIENCES

NRIMS/W/67/1

Interpreter and Subroutine Package for Interval Arithmetic on the IBM/360

by

D.P. Laurie
Numerical Analysis Division

Abstract:

An interpreter is described which converts explicit type declarations, assignment and arithmetic IF statements for interval variables to equivalent FORTRAN statements. A subroutine package performing the interval arithmetic is provided. The interpreter is also suitable for use with any fancy arithmetic subroutines (e.g. extended precision) that require synonymous subroutine calls for each and every arithmetic operation.

C O N T E N T S

0. Machine Interval Arithmetic on the IBM/360

In the great majority of all calculations on computers the numbers involved are subject to imprecision, either because they arise from measurements or because of the fact that computers only carry a finite number of significant digits. Almost always, no attempt is made to keep track of the errors that may be present in the final results.

The purpose of interval arithmetic is to carry along automatic error estimates. This is done by considering, rather than a number, an interval containing the entire range of possible values. An arithmetic operation on two intervals yields an interval containing all possible results of the operation on two elements of each interval. The final resulting intervals are always unduly pessimistic, but sharper bounds are usually very cumbersome to obtain even by analytical methods.

On an actual computer, the exact real interval arithmetic of theory is of course impossible; therefore machine interval arithmetic is used. This means that instead of an exact interval, the smallest machine interval containing it is used. This keeps track automatically of all the possible errors, whether arising from the initial data or from the round-off errors in the computation.

On the IBM/360, truncation is always used rather than rounding, i.e. if a given number (whether given as a decimal expansion or a too long binary expansion, as arises from computation) is not a machine number, it is represented by the nearest machine number that is not greater in absolute value, rather than just the nearest machine number. This simplifies the writing of interval arithmetic subroutines, as it is then known that the next machine number greater in absolute value than the 360 representation of a given real number will then also be greater (absolutely) than the real number. A minimal machine interval can thus readily be found.

The machine interval arithmetic system described in this report uses two implementations, one in single precision and the other in double precision.

1. Interval Arithmetic as a FORTRAN extension

Interval arithmetic is normally used with FORTRAN by declaring an interval variable as an array of dimension 2, and using special subroutines

to perform the interval operations. This is a tedious and error-prone procedure. What is needed is the ability to declare an interval variable as such, and use it in arithmetic and IF statements just like a normal variable.

The INTFORT program goes some way towards remedying this. It accepts as input normal FORTRAN IV programs, but including a few special statements concerning interval variables. These statements are identified by the character I in column 1. It produces as output the same FORTRAN IV program, but with these statements converted to valid FORTRAN IV using the same subroutines for interval arithmetic as previously. The programmer can thus use normal FORTRAN expression format in assignment and IF statements.

Interval Arithmetic Subroutines

Name	Argument list	Argument types	Action
ADD DADD	A,B,C	I4,I4,I4 I8,I8,I8	C ← A+B
NEG DNEG	A,C	I4,I4 I8,I8	C ← -A
EQU DEQU	A,C	I4,I4 I8,I8	C ← A
INT	A,C	E,I4	A is assumed to be the truncated machine representation of a non-machine number. C is set to the smallest machine interval containing that number.
SUB DSUB	A,B,C	I4,I4,I4 I8,I8,I8	C ← A-B
MUL DMUL	A,B,C	I4,I4,I4 I8,I8,I8	C ← A*B
DIV	A,B,C	I4,I4,I4	C ← A/B If 0 is in B, C is set to the largest machine interval.
POW DPOW	A,I,C	I4,F,I4 I8,F,I8	C ← A**I
SINGLI	A,C	I8,I4	C ← The smallest machine interval containing A.
DOUBLI	A,C	I4,I8	C ← The double precision equivalent of A.

Interval Arithmetic Functions

Name	Argument list	Argument types	Function type	Action
IUNE IUND	A,B,C	I4,I4,I4 I8,I8,I8	F	If A and B intersect, C is set to the union of A and B, and the value 1 is returned. Else C is unchanged, and 0 is returned.
ISECTE ISECTD	A,B,C	I4,I4,I4 I8,I8,I8	F	If A and B intersect, C is set to the intersection of A and B, and 1 is returned. Else C is unchanged, and 0 is returned.
ICONTE ICONTD	A,B	I4,I4 I8,I8	F	-2 is returned if A and B overlap -1 is returned if B contains A 0 is returned if A and B are equal 1 is returned if A contains B 2 is returned if A and B are disjunct
HALF DHALF	A	I4 I8	E D	The truncated midpoint of A is returned
WIDTH DWIDTH	A	I4 I8	E D	The truncated width of A is returned
INTSGE INTSGD	A	I4 I8	F F	If A contains 0, 0 is returned. Else sign (A) is returned.

Note: The above four subprograms are also callable as subroutines when the returned function value is not required.

Explanation of Types

F	INTEGER*4
E	REAL*4
D	REAL*8
I4	INTERVAL*4
I8	INTERVAL*8

Table 2.1

2. The Interval Subroutines and Interpreter

In order to understand the use of interval variables, we must know how the eventual FORTRAN program will look. This requires knowledge of the

subroutines used. (See Table 2.1).

The INTFORT program acts as an interpreter: the special statements involving interval variables are converted to FORTRAN type declarations, subroutine calls and IF statements. It is designed to be compatible with the G and H level IBM/360 FORTRAN compilers.

The interval statements resemble FORTRAN type declaration, assignment and arithmetic IF statements, and have the obvious meaning implied by the notation used. An interval is regarded as zero if it contains zero, and positive or negative if the entire interval is positive or negative, respectively.

3. The Interval Statements

General Format:

Each interval statement is identified by the letter I in column 1. Columns 2-5 are available for a statement number if the statement is an executable one, while column 6 must be blank, since no continuation cards are permitted. Columns 7-72 may be used in accordance with the format for the particular statement (described below), while columns 73-80 can be used by the programmer for any purpose he thinks fit.

Action of the Interpreter:

Each interval statement is reproduced unchanged as a comment statement, with a C replacing the I in column 1. The remaining statements produced are FORTRAN statements, details about which are given in section 4, "Programming Notes and Hints". Self-explanatory diagnostics are provided, as listed in Appendix A.

Subroutines and Functions:

If these occur, each END statement in the program should be marked with an I in column 1 to allow recognition by the interpreter.

Type of operand 1	Type of operand 2			
	REAL*4	REAL*8	INTERVAL*4	INTERVAL*8
REAL*4	REAL*4	REAL*8	INTERVAL*4	illegal [1]
REAL*8	REAL*8	REAL*8	INTERVAL*4 [2]	INTERVAL*8
INTERVAL*4	INTERVAL*4	INTERVAL*4 [2]	INTERVAL*4	INTERVAL*8
INTERVAL*8	illegal [1]	INTERVAL*8	INTERVAL*8	INTERVAL*8

1) The result will have type INTERVAL*8 with probability $\frac{1}{2}$, and give a specification exception in the other cases.

2) The long part of the double precision number will be ignored.

Table 3.1: Type of result of arithmetic operations +, -, *, /

Type Declaration Statement

Format: INTERVAL*s $a_i(k_i),\ldots,a_n(k_n)$

Where: *s represents the length in bytes that each end point of the interval will occupy. s may be 4 or 8; if *s is omitted, *4 is assumed.

a_i represents a valid FORTRAN variable name.

k_i is a list of dimension extents, to be omitted if a_i is not an array. Each k is a list of one to six integer constants; if a_i appears in a subprogram parameter list, integer variables may also appear in the list.

The interval declaration statement is necessary for all interval variables that will be used.

Examples

```
INTERVAL A
INTERVAL*8 A,B(5,5)
```

Assignment Statement

Format: x = y

Where: x represents the name of a variable or array element that has appeared in a preceding type declaration statement.

y represents an expression formed as in FORTRAN, containing interval variables (or array elements) and (optionally) sub-expressions of ordinary FORTRAN variables, constants, functions or array elements. Such sub-expressions are not examined for type, and the user must take care that they have legal type (see table 3.1). An interval variable may only be raised to an integral power.

The expression y is evaluated according to normal operator precedence rules, and assigned to x. If two quantities of unlike type are combined (except for the ** operator), one is first converted to the type of the other so that both have the type of the result as specified in table 3.1.

Examples:

If A and B are INTERVAL*4, C is INTERVAL*8 and D is REAL*8, the following statements are valid:

```
A = B
A = B + C
A = D*(B + C)
A = 3.14159265
A = A**2
```

The following statements may cause errors:

```
C = 1.          (Correct is C = 1.D0)
A = A**2.
```

Arithmetic IF statement

Format: IF (y) s_1, s_2, s_3

Where: y is an expression as in the assignment statement, containing at least one interval variable.

s_1, s_2, s_3 are FORTRAN statement numbers.

7/.....

If $\underline{y}$ is wholly negative, control is transferred to statement $\underline{s}_1$; if $\underline{y}$ contains 0, control is transferred to statement $\underline{s}_2$; if $\underline{y}$ is wholly positive, control is transferred to statement $\underline{s}_3$.

Example:

If A is of type INTERVAL*4, the following statement is valid:

```
IF(A) 1,2,3
```

4. Programming notes and hints

Each interval variable (or array) is interpreted as an array of real variables with the same length and with one more dimension (of extent 2) than the corresponding interval variable (or array). To ensure that the end points of each interval occupy contiguous storage locations, the extra dimension is added at the left of the subscript list. An INTERVAL statement might appear thus in the outputted FORTRAN:

```
C       INTERVAL*8 A,B(5,5)
        REAL*8 A(2), B(2,5,5)
```

When referring to an unsubscripted interval variable in a sub= routine call, the name itself is sufficient; when referring to a subscripted array name, however, the left dimension must have the subscript 1, thus:

```
C       A = B(3,2)
        CALL DEQU(B(1,3,2),A)
```

Note that the interpreter will not examine such a subscript list in any way. The following error will thus only be discovered by the FORTRAN compiler:

```
C       INTERVAL X(ANYOLDJUNK)
        REAL*4 X(2,ANYOLDJUNK)
```

In an assignment statement, mixed mode in an expression is tole= rated with the limitations set out in table 3.1.

The interpreter classifies all variables in three groups: those that have been declared as INTERVAL*8, those that have been declared as INTERVAL*4, and those that have not been declared as either. Before

an arithmetic operation between an interval A and an ordinary variable X is performed, X is assigned to a dummy interval variable of the same length as A. This assignment is only compatible when X is of the same length as A. The programmer thus must use the FORTRAN built-in functions SNGL, DBLE, FLOAT and DFLOAT if necessary to ensure such compatibility. Similarly, the IFIX function may be needed to ensure that the exponent after the ** operator is of type integer.

Interval constants do not exist; assignments of constants to interval variables can be made in two ways:

(i) Straightforward assignment to the end points, e.g.:

```
A(1) = 1.D0
A(2) = 1.D0
```

(ii) Interval assignment statement, e.g.:

```
C     A = 1.1D0
      CALL DINT (1.1D0,A)
```

The first method must be used when the number is an exact machine number, as the second method always assumes the constant to be the truncated machine number nearest to, but lower than the constant (which is the normal situation for a FORTRAN constant).

5. Other Subroutine Libraries

Since the subroutines needed are only provided at linkage time, the interpreter can be used for any arithmetic requiring logical words of two normal variables, e.g. extended precision routines.

These subroutines may be supplied either as source decks included at compile time, or as compiled (object or load) modules included at linkage time. Only those subroutines to which actual calls are gene= rated by the interpreter need be provided.

For details about the subroutines, refer to table 2.1.

APPENDIX A: Diagnostic Messages

Warning (Severity code 4)

NAME name TRUNCATED TO SIX CHARACTERS

Interpreting continues with the shortened name appearing in the FORTRAN output.

Errors (Severity code 8)

01 UNRECOGNIZABLE STATEMENT

02 INVALID LENGTH SPECIFICATION
Byte length must be 4 or 8.

03 RIGHT PARENTHESIS MISSING
Dimension extent list is not closed.

04 SYNTAX ERROR IN INTERVAL STATEMENT
Some other symbol was found where a separating comma was expected.

05 NULL ASSIGNMENT STATEMENT
Nothing appears after the "=" sign.

06 ILLEGAL CHARACTER "c" FOUND IN ASSIGNMENT STATEMENT

07 LEFT SIDE OF ASSIGNMENT STATEMENT NOT OF TYPE INTERVAL

08 MYSTERIOUS SYNTAX ERROR OR INTERPRETER BUG
Hopefully, this will never occur.

09 EXTRA "c" FOUND IN ARITHMETIC CONSTANT
An arithmetic constant contains more than one decimal point, exponent designator (E or D).

10 ILLEGAL SYNTACTICAL ELEMENT "element" TO RIGHT OF EQUAL SIGN
A right parenthesis or arithmetic operator other than "+" or "-" follows the "=" sign.

11 UNBALANCED parity PARENTHESIS

12 ILLEGAL SYNTACTICAL ELEMENT "element 1" TO RIGHT OF ELEMENT "element 2"
Juxtaposition of two operators, variables, constants or parenthesized expressions constituting a syntax violation.

13 EXPRESSION IN IF STATEMENT NOT OF TYPE INTERVAL

The severity code is passed to the job scheduler and may affect execution of following job steps.

10/.....

APPENDIX B: Job Control Language

```
//JOBLIB DD DSN=NNWW.P5408.INTERVAL,DISP=SHR
//INT EXEC PGM=INTFORT
//SYSPRINT DD SYSOUT=X,DCB=(RECFM=FBA,BLKSIZE=7182,LRECL=133)
//FORTIN DD UNIT=SPOOL,SPACE=(TRK,(20,10)),DCB=(RECFM=F,BLKSIZE=80),
       DISP=(NEW,PASS)
//OUT DD UNIT=SPOOL,SPACE=(TRK,(10,5)),DCB=(RECFM=F,BLKSIZE=80)
//SYSIN DD *
       (Input to interpreter)
/*
// EXEC FORTGCG,COND=(5,LT)
//FORT.SYSIN DD DSN=*.INT.FORTIN,DISP=(OLD,DELETE)
       (See note)
//GO.SYSLIB DD DSN=SYS1.FORTLIB,DISP=SHR
//          DD DSN=*.JOBLIB,DISP=SHR
//GO.SYSIN DD *
       (data cards)      Optional
/*
```

Note: If FORTRAN subroutines that need not go through the interpreter are used, they are to be inserted at this point as follows:

```
//   DD *
     (FORTRAN source)
/*
```

11/.....

APPENDIX C: An Example

The following subroutine calculates the value and derivative of a poly= nomial by Horner's scheme. The coefficients of the polynomial are assumed to be stored in the interval array A.

```
      SUBROUTINE FUN(X,F,D,N)
I     INTERVAL X,F,D,HORNER(11),A(11)
      N1=N+1
I     HORNER(N1)=A(N1)
      DO 1 J=1,N
      I=N1-J
I     HORNER(I)=X*HORNER(I+1)+A(I)
    1 CONTINUE
I     F=HORNER(1)
I     D=HORNER(N1)
      NM=N-1
      DO 2 J=1,NM
I     D=X*D+HORNER(I)
    2 CONTINUE
      RETURN
      END
```

N.B. Note that the end of a DO loop may not be an interval statement, as all such statements are translated to subroutine calls and are thus transfer statements.

The generated FORTRAN statements are (excluding the comments, which are reproductions of the statements marked I):

```
      SUBROUTINE FUN(X,F,D,N)
      REAL*4 ¢EO1(2)
      REAL*4 X(2),F(2),D(2),HORNER(2,11),A(2,11)
      N1=N+1
      CALL EQU(A(1,N1),HORNER(1,N1))
      DO 1 J=1,N1
      I=N1-J
      CALL MUL(X,HORNER(1,I+1),¢EO1)
```

12/.....

```
      CALL ADD (ØEO1,A(1,I),HORNER(1,I))
    1 CONTINUE
      CALL EQU(HORNER(1,1),F)
      CALL EQU(HORNER(1,N1),D)
      NM=N-1
      DO 2 J=1,NM
      I=N1-J
      CALL MUL(X,D,ØEO1)
      CALL ADD(ØEO1,HORNER(1,I),D)
    2 CONTINUE
      RETURN
      END
```

This routine was used in a program to compute intervals containing the roots of a polynomial that may itself have interval coefficients. The sensitivity of roots to changes in the coefficients could thus be studied.

CJ/DPL/EB
21.2.1972