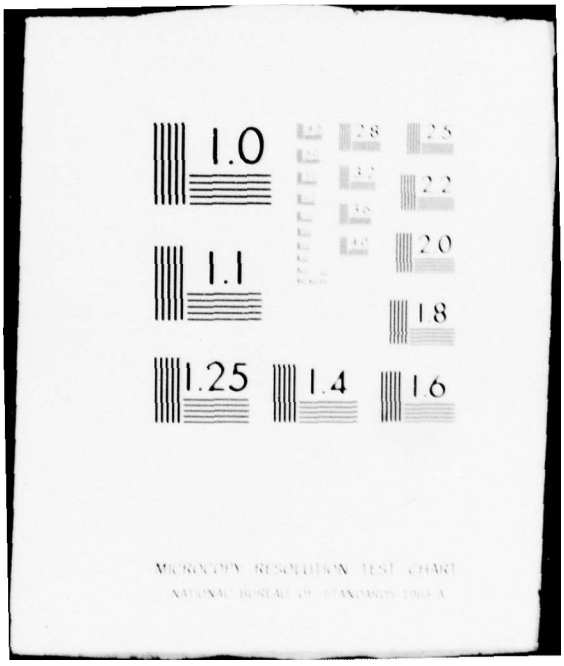
AD-A070 375 SOUTHERN METHODIST UNIV DALLAS TX DEPT OF COMPUTER S--ETC F/G 9/2
IMPLEMENTATION AND EVALUATION OF INTERVAL ARITHMETIC SOFTWARE. --ETC(U)
APR 79 D A COHN, J B POTTER, M GINSBERG DACA39-77-M-0105
UNCLASSIFIED WES-TR-0-79-1 NL

1 OF 2
AD A070375


1.0
1.1
1.25
1.4
1.6
2.0
1.8
2.2
2.5
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

TECHNICAL REPORT O-79-1

IMPLEMENTATION AND EVALUATION OF INTERVAL ARITHMETIC SOFTWARE

Report 5

THE CDC CYBER 70 SYSTEM

by

David A. Cohn, J. Brian Potter, Myron Ginsberg

Department of Computer Science
Southern Methodist University
Dallas, Tex. 75275

April 1979

Report 5 of a Series

Prepared for Office, Chief of Engineers, U. S. Army
Washington, D. C. 20314

Under Contract No. DACA39-77-M-0105

Monitored by Automatic Data Processing Center
U. S. Army Engineer Waterways Experiment Station
P. O. Box 631, Vicksburg, Miss. 39180

Destroy this report when no longer needed. Do not return it to the originator.

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

This program is furnished by the Government and is accepted and used by the recipient with the express understanding that the United States Government makes no warranties, expressed or implied, concerning the accuracy, completeness, reliability, usability, or suitability for any particular purpose of the information and data contained in this program or furnished in connection therewith, and the United States shall be under no liability whatsoever to any person by reason of any use made thereof. The program belongs to the Government. Therefore, the recipient further agrees not to assert any proprietary rights therein or to represent this program to anyone as other than a Government program.

The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

(9) 1. REPORT NUMBER: Technical Report O-79-1

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

(6) 4. TITLE (and Subtitle): IMPLEMENTATION AND EVALUATION OF INTERVAL ARITHMETIC SOFTWARE. Report 5. The CDC CYBER 70 System.

5. TYPE OF REPORT & PERIOD COVERED: Report 5 of a series

6. PERFORMING ORG. REPORT NUMBER

(10) 7. AUTHOR(s): David A. Cohn, J. Brian Potter, Myron Ginsberg

(15) 8. CONTRACT OR GRANT NUMBER(s): Contract No. DACA39-77-M-0105, NSF-MCS77-06611

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Southern Methodist University, Department of Computer Science, Dallas, Tex. 75275

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Intergrated Software Research & Development Program, AT11

11. CONTROLLING OFFICE NAME AND ADDRESS: Office, Chief of Engineers, U. S. Army, Washington, D. C. 20314

(11) 12. REPORT DATE: Apr[illegible]79

13. NUMBER OF PAGES: 103

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): U. S. Army Engineer Waterways Experiment Station, Automatic Data Processing Center, P. O. Box 631, Vicksburg, Miss. 39180

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

(12) 198 p.

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

(18) WES (19) TR-O-79-1

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

CDC CYBER 70 System
Computer systems programs
Interval arithmetic

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This is Report 5 of a series entitled "Implementation and Evaluation of Interval Arithmetic Software." The series concerns implementation and evaluation of an interval arithmetic software package on six different computer systems. The other reports to be published in the series are:

Report 1: The State of the Interval: Evaluation and Recommendations
Report 2: The Honeywell MULTICS System

(Continued)

over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

409 158

20. ABSTRACT (Continued).

Report 3: The Honeywell G635 System
Report 4: The IBM 370, DEC 10, and DEC PDP-11/70 Systems

The potential user of this or any other interval analysis package should take into consideration the limitations of this technique. Some of the limitations are inherent in all interval analysis implementations and others are dependent upon one specific package.

The user should be aware that software interval analysis tends to be very slow; cases run on one CDC CYBER 70 have required as much as 100-fold increase in execution time over a noninterval version. Thus, large production codes involving significant amounts of floating-point computations are not viable candidates for a complete interval implementation; however, small portions of the computation might benefit from use of the package.

The computed bounds using interval analysis can be overly pessimistic as well as being time-consuming to determine.

The tightness of the bounds is dependent upon the arithmetic of the host computer and possible numerical anomalies introduced by compile-time arithmetic.

Another factor affecting the performance of the interval package is the accuracy of the host computer's FORTRAN library routines.

PREFACE

In December 1975, the Automatic Data Processing (ADP) Center of the U. S. Army Engineer Waterways Experiment Station (WES), Vicksburg, Miss., submitted a proposal to implement and evaluate interval arithmetic, a software system for digital computer numerical analysis, on the Corps of Engineers' primary engineering computer--the WES Honeywell G635. The proposal was later expanded to include the implementation and evaluation of an interval arithmetic software package on six different computer systems. Engineering and scientific data problems were selected to be used on each of the six computers with the interval arithmetic software.

The work was funded by the Office, Chief of Engineers, U. S. Army, through the Integrated Software Research and Development (ISRAD) Program, AT11, Engineering Software Research.

This is Report 5 of a series entitled "Implementation and Evaluation of Interval Arithmetic Software." The other reports to be published in the series are:

Report 1: The State of the Interval: Evaluation and Recommendations

Report 2: The Honeywell MULTICS System

Report 3: The Honeywell G635 System

Report 4: The IBM 370, DEC 10, and DEC PDP-11/70 Systems

This report was written by Mr. David A. Cohn, Mr. J. Brian Potter, and Dr. Myron Ginsberg of the Department of Computer Science, Southern Methodist University, Dallas, Tex. Their work was performed under Contract No. DACA39-77-M-0105, dated 24 February 1977, and under National Science Foundation Grant MCS 77-06611. The work concerned implementation and evaluation of an interval arithmetic software system on the CDC CYBER 70 computer system.

Dr. J. Michael Yohe, Director of Academic Computing Services, University of Wisconsin-Eau Claire, developed and wrote the interval arithmetic software

package which was implemented on each of the six computer systems. Dr. Fred D. Crary, formerly with the U. S. Army Mathematics Research Center, University of Wisconsin-Madison, developed and wrote the AUGMENT precompiler which was implemented on each computer system as a front-end to the interval arithmetic software package. Dr. Yohe and Dr. Crary are specially thanked and recognized for their technical contributions and assistance.

Mr. James B. Cheek, Jr., formerly with the ADP Center, WES, provided initial impetus and guidance for the project. Mr. Fred T. Tracy, ADP Center, WES, provided expert advice and technical guidance during the project. Dr. N. Radhakrishnan, Special Technical Assistant, ADP Center, furnished technical guidance and general project supervision. The project and the report were monitored by Mr. William L. Boyt under the general supervision of Mr. D. L. Neumann, Chief of the ADP Center.

Directors of WES during the project and the preparation of the report were COL G. H. Hilt, CE, and COL J. L. Cannon, CE. Technical Director was Mr. F. R. Brown.

Copies of the other reports of the series, computer listings of the interval program and of AUGMENT for each computer system, and runs of the benchmarks for each computer system may be obtained from the ADP Center, WES.

CONTENTS

IMPLEMENTATION OF THE 'AUGMENT' PRECOMPILER AND INTERVAL ARITHMETIC ON THE CDC CYBER 70 SYSTEM

1. Cyber 70 and Operating System Architecture

Architectural differences amongst the 6000 Series, 7600, Cyber 70 Series, and Cyber 170 Series are sufficiently minimal for this report to refer unambiguously to CDC computers.

On Control Data computers, I/O and operating system functions are resident in the ten to twenty peripheral processors (referred to as PPUs) rather than in the central processor. There are no direct communications between the CPU and the PPUs. The CPU cannot be interrupted; requests for operating systems' and I/O functions are detected by a PPU polling loop. All communications between the CPU and the external devices are through main memory, where all user code is resident. The use of PPUs for I/O and the high memory bandwidth are keys to the high system performance. Extended Core Storage (ECS), if present, is often used as a fast swapping drum and for small user files. Some of the relationships amongst the architectural components are illustrated in Figure 1.

The CDC KRONOS Operating System which is used on our host machine is highly modular. It is based on the File Name Table (FNT). Each control card that is encountered causes the FNT to be searched for a user file to be loaded and executed. If the sequential search fails, the FNT is searched again for a PPU program or system CPU utility; if the second search succeeds, that system routine is executed.

The Southern Methodist University Computer Center has a Cyber 70, model 72 machine with 11 peripheral processors, no ECS, and a maximum main memory user space of approximately 200_8K words (i.e. 64_{10}K words). The Fortran

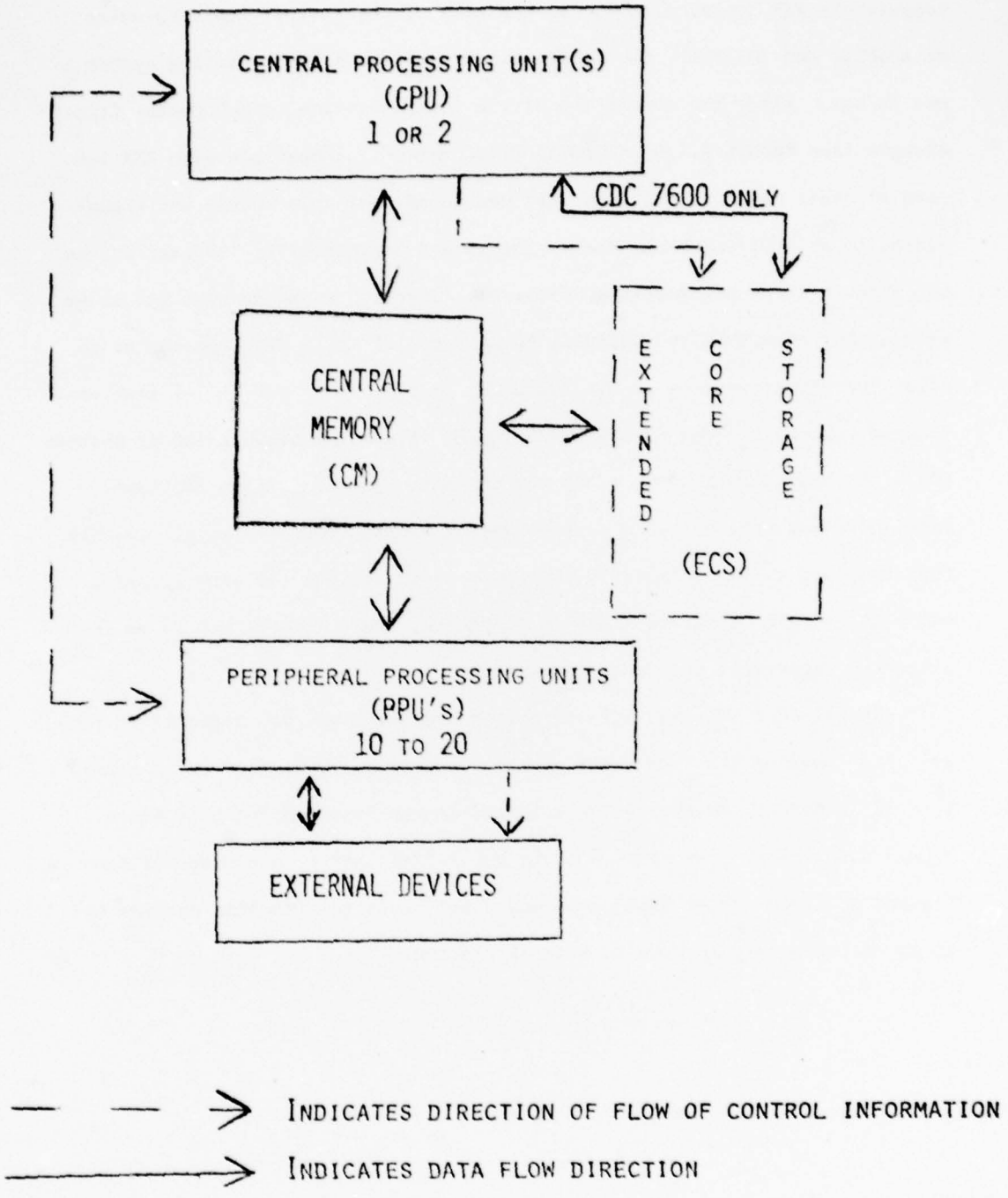


Figure 1: CDC 6000 and 7000 Series Systems Configuration

compiler is FTN (version 4.6). At the time of the initial implementation of AUGMENT and INTERVAL, the KRONOS 2.1.0 version of the operating system was in use. After the completion of the implementation, the Computer Center changed from KRONOS 2.1.0 to KRONOS 2.1.2 which is compatible with NOS 1.0 used at other CDC installations. It was found that this change was transparent to AUGMENT/INTERVAL, due to the upward compatibility inherent in the CDC product (both hardware and software). Inasmuch as no changes had to be made either to AUGMENT or INTERVAL to make use of CDC's FTN Compiler or to allow for either version of the operating system, it is our belief that no changes would be needed if AUGMENT/INTERVAL were to be implemented at another CDC site. None of the code makes any calls to specific system routines. Note, however, that calls to certain UNIVAC Fortran routines (e.g., the FLD function) had to be replaced by in-line code to achieve the same purpose, and that certain UNIVAC routines (e.g. URWALK, CBRT, STRACE) had to be replaced by equivalent routines, coded at SMU.

We did find that AUGMENT, even when overlaid, was too large to be run as a time-sharing job (SMU TELEX permits a maximum job size of 100_8K words, i.e. 32_{10}K words), so that we were forced to run in batch but this was a minor inconvenience, at worst. It is our belief that a host computer configured to allow larger TELEX jobs would not be faced with this problem but it is not necessary to have this configuration.

2. Implementation of AUGMENT

AUGMENT is a Fortran-based precompiler that facilitates the use of non-standard constructs in Fortran programs. It can also be used to aid in translation between various dialects of Fortran, or to implement non-standard operators. It accepts programs written in a user-defined extension of Fortran where the extension consists of new data types, new operators, and/or new t. library functions. It requires a description of the extension, which consists essentially of a description of the user-supplied routines that implement the extension. It then uses this description to translate the extended Fortran into standard Fortran, with the non-standard constructs translated into references to the routines in the supporting packages.

In the SMU implementation of AUGMENT, we found that the non-overlaid version required 170_8K words (i.e. 60_{10}K words) of core memory (and, of course, a disk channel). Using the overlaying suggestions given in [2], we overlaid AUGMENT, which reduced the size requirement to 120_8K words (i.e. 40_{10}K words) of central memory but at the cost of increasing the execution time required. We were able to follow the suggested overlays with the exception that we were forced to move one routine from the suggested overlay to the root overlay.

On CDC computers the output from AUGMENT can be compiled on the FTN compiler with any desired compiler options. The resulting object decks are then loaded with libraries INTLIB and BPALIB which contain, respectively, the routines defining the interval and BPA operations, and the primitives which

perform the basic arithmetic operations for the entire package. The result of the load can then either be executed or written to absolute central processor overlays (see Figure 2). The control card sequence for the use of the AUGMENT precompiler as well as the entire job deck structure is discussed in Section 6 of this report.

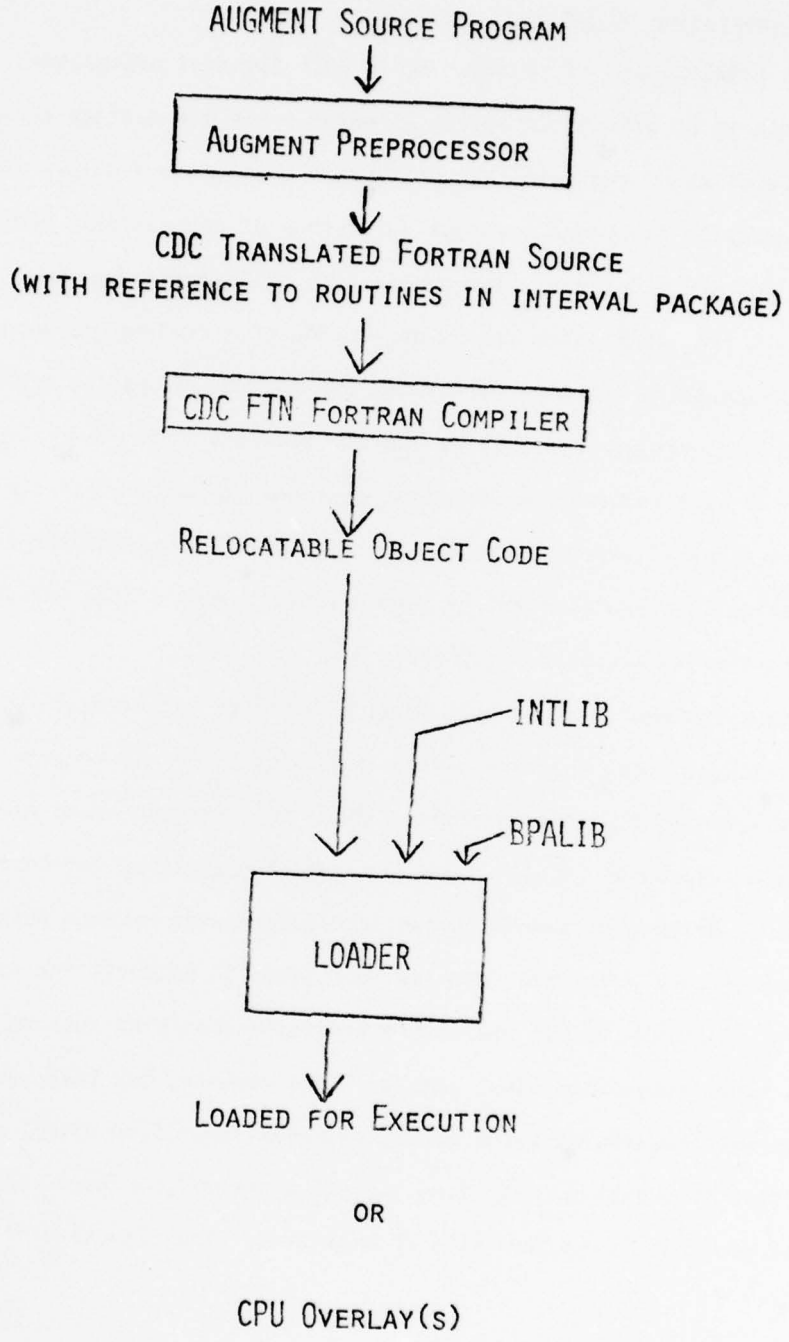


Figure 2: Processing Sequence for Program Using Interval Analysis Package

3. Implementation of Interval Routines

The INTERVAL set of routines implements interval arithmetic. Interval arithmetic is an arithmetic system in which rules are defined to create interval versions of the primitive arithmetic operations and then to combine these intervals to create interval extensions of user-defined mathematical expressions and functions. The application of interval analysis to a computational problem results in the generation of a rectangular parallelpiped which encapsulates the exact solution. In our implementation, all the usual arithmetic operations and Fortran library routines have interval counterparts; in addition, several interval-oriented operations are also available. The reader unfamiliar with the details of interval analysis should read the explanation in Appendix B and/or should consult some of the references given in the bibliography in Appendix A.

Since chained computations, especially those involving multiplications and divisions, cause the size of the resulting intervals to grow rapidly, the concepts of Best Possible Arithmetic (BPA) and directed roundings come into play (see references [3,4]). BPA is a set of algorithms for performing the four basic arithmetic operations while retaining the maximum possible amount of accuracy on a given host machine (CDC Cyber 70 Series). The directed rounding algorithm rounds the double precision result of such an operation into a single precision final result. This rounding may take one of four directions: inwards (towards zero), outwards (away from zero), upwards (towards zero if negative, away from zero if positive), or downwards (towards zero if positive, away from zero if negative). The direction of the rounding

is specified by the computational environment in each interval routine.

Our CDC implementation of this software package has followed the suggestions of Yohe and introduced only those changes necessary to accommodate the CDC software and floating-point arithmetic. These changes are described in the UPDATE deck on our release tape (see Section 4). For the BPA routines, we have implemented two versions. The first version, called BPALIB, contains the following routines:

BPAADD - Yohe's addition algorithm

BPASUB - Yohe's subtraction algorithm

BPAMUL - Yohe's multiplication algorithm

BPADIV - Yohe's division algorithm

BPARND - Yohe's directed rounding algorithm

All five routines implement the algorithms given in [3] with modifications made only to allow for the one's complement Cyber 70 floating-point format.

In addition, we have implemented a second version, called BPFLIB (resident in BPALIB), which contains the following routines:

BPFADD - Sterbenz' algorithm for an ssd addition

BPFSUB - Sterbenz' algorithm for an ssd subtraction

BPFMUL - Sterbenz' algorithm for an ssd multiplication

BPFDIV - Sterbenz' algorithm for an ssd division

BPFRND - a double to single rounding, similar to BPARND

These routines implement the algorithms suggested by Sterbenz [6]. All these routines are coded in CDC 6000 assembly language. In contrast to the BPA routines which implement Yohe's algorithms to perform best possible floating-point arithmetic, the BPF routines use the CDC 6000 floating-point hardware to perform faster, reasonably accurate computations that are within one ulp

of the corresponding BPA results; initial numerical tests of our implementation indicate that BPF routines tend to require from 50 to 80 percent of the execution time of their BPA counterparts. The CDC loader may be used to substitute any BPF routine for any BPA routine if computation time is more critical than computational accuracy (tighter bounds); the LDSET card with SUBST parameter allows the user to switch from any BPA to any BPF routine (see Section 6). The documented listings of BPALIB and BPFLIB are available from the Automatic Data Processing (ADP) Center of the U. S. Army Engineer Waterways Experiment Station (WES), P. O. Box 631, Vicksburg, Miss. 39180.

4. Limitations of Interval Analysis Package

The potential user of this or any other interval analysis package should take into consideration the limitations of this technique. Some of the limitations are inherent in all interval analysis implementations and others are dependent upon our specific package. First of all, the user should be aware that software interval analysis tends to be very slow; cases run on our CDC machine have required as much as 100 fold increase in execution time over a non-interval version. From 50 to 300 fold increases of execution time over non-interval versions have been observed for various implementations of the Yohe package (see Yohe (1977)). Thus large production codes involving significant amounts of floating-point computation are not viable candidates for a complete interval implementation; however, small portions of the computation might benefit from the use of the package and/or the utilization of a hardware-assisted interval analysis such as that currently being proposed by Intel Corporation for microcomputers (interval versions would be only 2 to 4 times slower than non-interval versions).

Another important aspect of interval analysis is that the computed bounds can be overly pessimistic as well as being time-consuming to determine. The innate structure of interval analysis can make its application insensitive to situations in which common mathematical expressions (such as in a numerator and denominator) are assigned different bounds; this situation is usually referred to as the simultaneity problem (see Sterbenz (1974)). The bounds for a particular computation are dependent on both the form in which the computation is expressed and the data which is imput to the expression; thus it is difficult, if not impossible, for the user to define in all cases the "best" form of the expression as well as the "best" data in order to ensure smallest growth in the interval size associated with the resultant computation.

Furthermore, the tightness of the bounds is dependent upon the arithmetic of the host computer and possible numerical anomalies introduced by compile-time arithmetic.

The I/O routines provided by the interval analysis package do not easily permit as much flexibility as that offered by standard Fortran I/O routines; also the interval I/O is very slow (e.g. ≈10 minutes to input 10,100 interval numbers on the CDC Cyber 72). This lack of flexibility can be partially overcome by using Fortran I/O with explicit lower and upper bounds of interval variables in the argument list.

Further difficulties can arise when converting non-interval arithmetic or logical IF statements to interval versions. Since the space defined by interval arithmetic has no total ordering, any of several possible partial orderings could be employed. The user must be careful when selecting an interval relational operator based on a particular partial ordering that he does not violate the intent of the original non-interval IF statement. This issue is discussed in more detail in Section 5 along with package supported relational operators based on two possible partial orderings.

Another factor affecting the performance of the interval package is the accuracy of the Fortran library routines. In order to provide the tightest possible bounds on these routines, a thorough analysis must be performed of the accuracy of each routine across the entire range of representable numbers for each argument. Bounds (not necessarily optimal) on the errors associated with CDC Fortran library routines used on our host machine, provided by Control Data, are available from the ADP Center, WES. It is this author's opinion that a much more thorough analysis must be conducted but such a task is beyond the scope of the current project; it could be done with the aid of a multiple precision package and with detailed knowledge of the algorithms and implementation

features employed in the Fortran library routines.

Additional constraints on the use of the interval analysis package include memory requirements and provisions for dealing with intervals which are infinite and/or include zero. The latter problem has been somewhat resolved by Kahan and others who have advocated entension of classical interval analysis. The former problem caused some problems at SMU where the maximum user memory space is 200_8k words; even using overlays, the storage requirements of the package prevented application to medium or large scale matrix problems.

5. Using the Interval Analysis Package

This section focuses attention on the mechanics of running the interval analysis package on CDC computers. An overview of the actions taken by the control cards used in this section is given in Section 2. It is assumed that the potential user of the package can modify his Fortran program for use with the interval analysis package in accord with the limitations discussed in Section 4. Here we will discuss the general and specific form of the job control cards used, the input and output of interval quantities, and an example run with the given software.

The following control card sequence is typical for use of the AUGMENT precompiler:

```
  JØB; ACCØUNT.                (installation dependent)
  ATTACH, AUGMENT.             (not required if AUGMENT is system-resident;
                                PN, UN, and NA parameters may be required)
* SETTL(time limit)
  RFL, (size of AUGMENT)       (varies from 20k to 200k
                                depending upon overlay tactics)
  AUGMENT (, SF, SC1)
  REWIND, SF.
* RETURN, SC1.
* RETURN, AUGMENT
* RFL, 50000.
  FTN, I=SF(, your options).
* RFL, 20000.
  ATTACH, INTLIB, BPALIB.
  LIBRARY, INTLIB, BPALIB.
  LDSET (USEP = BPABKDT/INTBKDT (, your options))
  RFL, (program size)
  LGØ (, FILE parameters).
* RFL, 20000.
* LIBRARY.
* RETURN, INTLIB, BPALIB, LGØ.
        :
        .
  (control cards for post-interval analysis processing)
        :
        .
  (end of record)
```

All cards marked with * are optional, but generally have the effect of improving system throughput. Note that one of the options on the LDSET card could be the "substitute" option to replace references to BPA routines with references to the BPF routines (see discussion in Section 3) if the faster arithmetic provided by these routines is desired.

The entire job deck structure is as follows:

```
(control card record)
(end of record)
 description deck
*BEGIN
 program in extended FORTRAN
*END
 (end of record)
 input, if any, for extended FORTRAN program
 (end of record)
 input, if any, for post-interval program
 (end of information)
```

At SMU, the software interval analysis package is run on a CDC Cyber 72 and resides on a private disk pack (PRO1036). The specific deck setup for a run with this package is as follows in our environment:

```
 Bin number.                    name
 ACCØUNT, 7 character alphanumeric account no, password.
 SETTL, appropriate time limit in octal seconds.
 ATTACH, AUGMENT/UN=EACR005, PN=PRO1036, M=W,NA.
 ATTACH, INTLIB/PN=PRO1036, UN=EACR007, M=R,NA.
 RFL, appropriate number of words in octal.
 AUGMENT(INPUT, CØMPILE, ØUTPUT, PL=999999, AUGSRC)
 RT, AUGSCR.
 ATTACH, BPALIB/UN=EACR007, PN=PRO1036, M=R.
 FTN, I, PL=9999999.
 LIBRARY, INTLIB, BPALIB.
 LDSET(USEP=BPABKDT/INTBKDT)
 LGØ.
 7
 8
 9
 *DESCRIBE EXTENDED
  AUGMENT description deck
 *BEGIN
 /          PROGRAM Name(INPUT, ØUTPUT, TAPE5=INPUT, TAPE6=ØUTPUT, DEBUG=ØUTPUT)


 Fortran source program
 * END
 7
 8
 9
   data, if any
 7
 8
 9
 6
 7
 8
 9
```

In the above control sequence all commands are written beginning in column 1. The card 7_{8_9} represents a multiple punch in column 1 signifying end of record and the card $6_{7_{8_9}}$ represents a multiple punch in column 1 indicating end of information. Note that for the CDC PROGRAM card, a slash must be inserted in column 1 in order to indicate to the AUGMENT preprocessor that it is to ignore this statement (otherwise a diagnostic error message is generated). The word PROGRAM begins in column 7 on this card. If the user wishes to use the faster BPF interval arithmetic rather than the BPA interval arithmetic specified by the above control sequence, then two changes are necessary:

```
    replace     ATTACH, BPALIP/UN=EACR007, PN=PRO1036, M=R.
    with        ATTACH, BPALIP=BPFLIB2/UN=EACR007, PN=PRO1036, NA.
and after       LDSET(USEP=BPABKDT/INTBKDT)
    insert      LDSET(SUBST=BPAADD-BPFADD/BPASUB-BPFSUB/BPAMUL-BPFMUL/BPADIV-BPFDIV
```

The above insert can be varied to permit the programmer to mix the use of BPA and BPF arithmetic; for example, the use of the control card

LDSET(SUBST=BPAMUL-BPFMUL/BPADIV-BPFDIV) would result in BPA arithmetic being performed for addition and subtraction and BPF arithmetic being performed for multiplication and division.

From the information provided by the release tape (described in Section 6), the programmer could set up the package to be used with the above control sequence employed at SMU or could devise any other arrangement more convenient to the user and/or his computer installation.

An interval variable or array is declared by the INTERVAL declaration which when processed by AUGMENT generates a REAL declaration with an additional dimension introduced to provide space for both endpoints of each interval element; for example, a declaration of

```
      INTERVAL   A, B(3,3)
```

would generate a declaration of

```
      REAL       A(2), B(2,3,3),
```

where A(1) and A(2) would represent the left and right endpoints, respectively, of interval variable A, B(1,1,1) and B(2,1,1) would represent the left and right endpoints of interval variable B(1,1), etc. Interval quantities can be expressed in a variety of ways both within the program or a data input to the program; a description of these representations is provided by Yohe (1977) and is reproduced as Appendix C to this report.

Interval quantities can be input or output via standard READ, WRITE, or PUNCH statements and/or by the INTRDF, INTRD, or INTWR statements provided with the package. When using the standard Fortran I/O routines with interval arguments, it is necessary to place a / in column 1 (to tell AUGMENT not to process this card) and to refer to the interval quantities in their expanded form as generated by the INTERVAL declaration. Thus if for the interval quantities referred to in the preceding paragraph above we wanted to output the left and right endpoints of A, B(2,1), and B(2,2) we could use the following statement:

```
/     WRITE(6, 100) A(1,2), A(2,2), B(1,2,1), B(2,2,1), B(1,2,2), B(2,2,2)
  100 FØRMAT(1X, 6E21.14)
```

We can proceed in a similar fashion when using READ or PUNCH statements, i.e. slash in column 1 and interval variables expressed in their expanded dimensional form when referenced in the argument list.

The format for the I/O provided by the interval analysis package is as follows:

```
      CALL     INTRDF (UNIT, X)
      CALL     INTRD (UNIT, FMT, A, N)
      CALL     INTWR (UNIT, FMT, A, N)
```

where

UNIT = 5(standard input) or 0(standard reread)

6(standard print unit) or 1(standard punch unit)

FMT = a user supplied format which must be declared by the user as a 3-element integer array for use with input statements and as a 4-element integer array for output statements.

FMT(1) = number of interval variables in each record

FMT(2) = number of characters to be skipped between interval variables in each record

FMT(3) = number of decimal digits in each interval variable

= (for output) 2*Q+15 where Q is number of decimal digits to be displayed for each interval endpoint

FMT(4) = (for output only) Hollerith printer carriage control character ' ' or '0'

For input, the INTRDF routine obtains the next data field from the input stream, converts it and stores the result in the declared interval variable specified as the second argument in the call to the routine (this variable is X in the above routine call); only one value is read by each call to routine INTRDF. In contrast, INTRD can be used to input the first N elements of an array of interval variables whose name is specified as the third argument in the call to the routine (A in the above example); the interval data elements can be expressed in any of the forms given in Appendix C. The number of elements to be read can be expressed explicitly or as any user-defined integer variable given as the fourth argument in the call to routine INTRD. For output, the interpretation of the third and fourth arguments for INTWR is essentially the same as that for INTRD except that the interval quantities are expressed in one form in which each endpoint is represented by a Q decimal digit signed mantissa with a three digit signed exponent as specified by FMT(3) above. Since our

CDC machine has roughly the equivalent of 14 or 15 decimal digits in single precision floating-point arithmetic, the selected value for Q should probably by less than or equal to 15. The value of FMT(3) should not exceed 135 since the width of our printer line is 136 (which includes 1 character for carriage control). Above we have given a very brief account of I/O with the interval analysis package; extensive additional details are given by Yohe (1977).

In the remainder of this section an example is presented which illustrates the use of the interval analysis package on a CDC Cyber 72 computer and how the non-interval version of a Fortran program can be modified to produce an interval version. The main program SPLIST calls subroutines SPLINE and SPLINT which are used for spline interpolation problems. Computational details and analysis of results using these routines can be found in the paper entitled, "Some Numerical Computational Experience with a Software Interval Analysis Package and FORTRAN Preprocessor on Control Data Cyber 70 Series Computers" by Myron Ginsberg and available as report CS 7808 (July 1978) from the Department of Computer Science at Southern Methodist University in Dallas, Texas. The non-interval version of the main program and the accompanying subroutines along with the dayfile (includes job control statements as well as statistics about the execution of the program) are available at WES, as is an interval version with its dayfile.

The interested reader should obtain these from WES and compare the non-interval and interval versions to observe the similarities and differences. It is first noted that the dayfiles are substantially different; this occurs because the non-interval version merely requires a Fortran compilation of the source program whereas the interval version requires, in addition, the use of the Fortran preprocessor AUGMENT and a large collection of interval arithmetic

support routines. The job control statements for the interval version are in accord with the discussion at the beginning of the section. Also the PROGRAM card in the non-interval version has a blank in column 1 whereas a slash appears in column 1 of the PROGRAM card for the interval version; the slash indicates to AUGMENT that it is not to attempt to alter this card but instead to pass it on intact (but without the slash) to the Fortran compiler. Similarly, other statements in the interval version with a slash in column 1 (for example, the PUNCH statements) are not to be processed by AUGMENT. In the earlier part of this section use of slashes was mentioned with I/O statements such as READ or WRITE when interval variables are present in the argument list.

Observe in the interval version that there are INTERVAL statements present at the beginning of the program and each subroutine and that all single percision floating-point variables used in each routine are declared in the accompanying INTERVAL statement. Such statements when processed by AUGMENT generate a REAL statement in which each variable is changed into a variable of the same name but with an additional dimension added as the new first dimension of the original variable; this action is necessary to provide storage for the lower and upper bounds associated with each interval quantity. The REAL statements generated by AUGMENT for our example under discussion are also available from the ADP Center, WES.

Another alteration of a non-interval program required to transform it into an interval version involves the re-writing of arithmetic and logical IF statements which employ interval quantities. Since interval arithmetic has no total ordering, any of several possible partial orderings could be used to provide definitions of relational operators. Two sets of such operators are pro-

vided by the interval analysis package - value operators and set operators. The value operator definitions are as follows:

.VE. where $[a,b] = [c,d]$ if and only if $a = b = c = d$

.VNE. where $[a,b] \neq [c,d]$ if and only if they are disjoint

.VLT. where $[a,b] < [c,d]$ if and only if $b < c$

.VLE. where $[a,b] \leq [c,d]$ if and only if $b \leq c$

.VGT. where $[a,b] > [c,d]$ if and only if $a > d$

.VGE. where $[a,b] \geq [c,d]$ if and only if $a \geq d$

The set operator definitions are as follows:

.SEQ. where $[a,b] = [c,d]$ if and only if $a = c$ and $b = d$

.SNE. where $[a,b] \neq [c,d]$ if and only if $a \neq c$ or $b \neq d$

.SLT. where $[a,b] < [c,d]$ if and only if $a < c$

.SLE. where $[a,b] \leq [c,d]$ if and only if $a \leq c$

.SGT. where $[a,b] > [c,d]$ if and only if $b > d$

.SGE. where $[a,b] \geq [c,d]$ if and only if $b \geq d$

The reader should note that for the value operators, if two intervals are non-empty and with corresponding endpoints equal, then neither equality nor non-equality of these intervals is true; in such cases the user could decide to use the set relational operations.

Several alterations of the non-interval versions of IF statements are present in the program under discussion (e.g. see lines 9510 or 9590 in routine SPLINE). For example line 1580 of routine SPLINT in the non-interval version is transformed from

```
IF(XP - X(1)) 100, 170, 110
```

to

```
IF(XP .VGT. X(1))GØ TØ 110
IF(XP .SEQ. X(1))GØ TØ 170
```

in the interval version. The user must alter all logical and arithmetic IF

statements involving interval variables. When the programmer is making such changes he must be careful to select the appropriate operator which best reflects the intent of the comparison in the non-interval version of the IF statement in the context of the program; such selection of interval operators is particularly crucial for IF statements involving termination and/or tolerance criteria where an inappropriate operator selection could significantly alter the behavior of the interval version of the program from its non-interval counterpart.

As discussed earlier in this section, the user has several ways to provide I/O for interval quantities and a couple of these are present in our example program. Note that an ordinary PRINT statement is used in the main program to output the non-interval quality YP whereas in the interval version, routine INTWR from the interval analysis package is used to output the interval variable YP. Also note that the format used with routine INTWR is provided by integer array FMT which is explicitly declared and initialized by the user in the routine in which INTWR is invoked. Also observe that interval variable YP is output by the use of a PUNCH statement but that in this statement a slash appears in column 1 (so AUGMENT will not process the statement) and that the lower and upper bounds of YP (namely YP(1) and YP(2), respectively) are explicitly referenced in the argument list.

In completing the transformation of the non-interval program into an interval program, the user can utilize several operators that are supported by the interval analysis package and applicable to interval quantities. A list of all such operators and their definitions is available at WES. In our sample program we have used MDPT and LGTH to provide the midpoint and length, respectively, of any interval quantity.

From our discussion above the user should realize that to change his non-interval program into an interval version on CDC machines, he must perform the following steps:

1. Use the job control statements (or equivalent) to invoke the AUGMENT preprocessor, interval analysis package, and Fortran compiler.

2. Place a slash in column 1 of the PROGRAM statement.

3. Introduce an INTERVAL declaration in each routine for all single precision floating-point variables (and arrays) appearing in that routine.

4. Change all arithmetic and logical IF statements involving interval quantities, replacing Fortran relational operators with the appropriate interval value or set operators.

5. For I/O interval quantities use the interval supported routines (INTRDF, INTRD, INTWR) or the regular Fortran I/O routine (but with slash in column 1 and explicit reference to lower and/or upper bounds of each interval quantity in the argument list).

6. Introduce any desired interval oriented operators supported by the interval package.

7. Select an appropriate time limit (use SETTL instruction) and field length (use RFL instruction) for the interval version of your program, keeping in mind that the interval version may require from 50 to 100 times (or more the execution time of the non-interval version and that the storage requirement for all interval variables and arrays is doubled from that of the non-interval representations.

After the non-interval program has been converted into an interval source program using steps 1 through 6 above, the AUGMENT preprocessor generates another Fortran program which invokes calls to subroutines provided by the interval analysis package in order to execute the interval program.

6. Format of the Release Tape for Distribution of the CDC Version

The interval analysis software is currently contained on two CDC UPDATE libraries. One library contains the BPA primitives and the other library contains all other routines including the AUGMENT routines. The format of the release tape is as follows:

7-track

556 BPI

KRONOS Internal Format (512 word blocks)

unlabeled

File No.	Description
1	BPA and BPF Primatives (Source Card Images)
2	INTERVAL and AUGMENT (Source Card Images)
3	BPA and BPF Primatives (UPDATE Source Format)
4	INTERVAL and AUGMENT Package (UPDATE Source Format)

Files 1 and 2 are card images which can be read by almost any machine. Files 3 and 4 are to be used with UPDATE to create new UPDATE libraries. They are basically the same as files 1 and 2 except they have UPDATE directives embedded within them.

7. Problem: To solve for $\underline{x}$ the linear algebraic system of equations

$$A\underline{x} = \underline{b}$$

where A is full (non-sparse) matrix. Test data for A is 100 x 100 case from WES

Code used and references:

FORTRAN code from WES called GAUSS and written by F. T. Tracy.

Conditions under which test was conducted:

The FTN 4.6 FORTRAN compiler was used with no optimization (i.e. opt = 0). Noninterval and interval (both BPA and BPF) cases were run. Timing figures were obtained by using the SECOND function to measure the CPU time for each run excluding time for I/O. The INTRD and INTWR routines were used with all variables declared interval. Inequalities involving interval variables used the interval operators VLE and VGE to replace the noninterval operators LE and GE, respectively.

Execution time for noninterval version:

14.763 seconds CPU time (excludes time involved with I/O)

Execution time for BPF interval version:

1139.251 seconds CPU time [18.99 minutes] (excludes time involved with I/O)

Execution time for BPA interval version:

1424.297 seconds CPU time [23.74 minutes] (excludes time involved with I/O)

Observations:

1. The BPF interval version requires 80% of the execution time used by the BPA version and produces competitive bounds. (This compares with 81% for DECØMP and SØLVE code with WES data.)

2. The BPA and BPF interval versions take 96.477 and 77.169 as much CPU time, respectively as the noninterval version. (This compares with 83.865 and 68.072 for DECØMP and SØLVE code on this same WES data.)

3. The BPF intervals for the x elements are wider than the corresponding BPA intervals; 79% of these intervals have a difference in width with corresponding BPA intervals on the order of 10^{-9} to 10^{-11}. This is the same distribution of interval width differences as occurred with the same problem with the DECØMP and SØLVE code, although the specific values of the interval width differences vary slightly from those of the DECØMP and SØLVE code. The distribution of BPF - BPA interval widths for x elements is as follows:

	difference in width	
	10^{-7}	X
	10^{-8}	XXXXX XXXXX XX
79%	10^{-9}	XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX X
79%	10^{-10}	XXXXX XXXXX XXXXX XXXXX XX
79%	10^{-11}	XXXXX XXXXX XXXXX XXXXX X
	10^{-12}	XXXXX
	10^{-13}	XXX

Observations: (cont.)

4. The distribution of BPA interval widths for the components of the solution vector $\underline{x}$ is the same as for the BPA intervals using the same data and the DECØMP AND SØLVE code. The distribution is as follows:

difference in width		
	10^{-4}	X
	10^{-5}	XXXXX XXXXX XXXXX X
60%	10^{-6}	XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX
60%	10^{-7}	XXXXX XXXXX XXXXX XXXXX XXXXX
	10^{-8}	XXXXX XXXXX XXXXX X
	10^{-9}	XXXX
	10^{-10}	XXX

5. The distribution of BPF interval widths for the components of the solution vector $\underline{x}$ is the same as for the BPA interval widths as well as for the BPA and BPF interval widths from the same data used with the DECØMP and SØLVE code:

	10^{-4}	X
	10^{-5}	XXXXX XXXXX XXXXX X
60%	10^{-6}	XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX
60%	10^{-7}	XXXXX XXXXX XXXXX XXXXX XXXXX
	10^{-8}	XXXXX XXXXX XXXXX X
	10^{-9}	XXXX
	10^{-10}	XXX

Since the distribution of widths of intervals is the same for BPA and BPF versions, it is therefore justified to use BPF bounds especially since BPF bounds require less execution time than BPA bounds (80% of BPA execution time for this problem).

6. The computational results ($\underline{x}$) (both interval and noninterval versions) for both the GAUSS code and the DECØMP and SØLVE code are remarkably similar which is somewhat reassuring considering two different codes (with slightly different algorithms) were used on a problem that seems to slightly ill-conditioned (determinant $\approx .361 \times 10^{-20}$ and condition number ≈ 8176). Possibly with a better conditioned A matrix, the bounds with both codes might have been tighter than the order of magnitude distribution indicated in observations 4 and 5.

The computed results for $\underline{x}$, the residuals, interval widths and midpoints of $\underline{x}$ components, as well as CPU timings are given for the noninterval and interval (BPA and BPF) versions in the attached listings.

8. Problem: To solve for $\underline{x}$ the linear algebraic system of equations

$$\underline{A}\underline{x} = \underline{b}$$

where A is full (non-sparse) matrix
Test data for A is 100 x 100 case from WES

Code used and References

DECØMP and SØLVE from Computer Methods for Mathematical Computations by George E. Forsythe, Michael A. Malcolm, and Cleve B. Moter, Prentice-Hall, Englewood Cliffs, New Jersey, 1977, pp. 52-55.

Conditions under which test was conducted:

The FTN FORTRAN Compiler was used with no optimization (i.e. opt = 0). Noninterval and interval (both BPA and BPF) cases were run. Timing figures were obtained by using the SECØND function to measure the CPU time for each run excluding time for I/O. The INTRD and INTWR routines were used with all variables declared interval. Inequality involving interval variables used the interval operators SEQ, VGT, VLT, VLE to replace the noninterval operators EQ, GT, LT, LE, respectively.

Execution time for noninterval version:

18.902 seconds CPU time (includes calls to DECØMP and SØLVE)

Execution time for BPF interval version:

1286.701 seconds [21.445 minutes] CPU time (includes calls to DECØMP and SØLVE)

Execution time for BPA interval version:

1585.217 seconds [26.42 minutes] CPU time (includes calls to DECØMP and SØLVE)

Observations:

1. The BPF interval version requires 81% of the execution time used by the BPA version and produces competitive bounds.

2. The BPA and BPF interval versions take 83.865 and 68.072 as much time, respectively, as the noninterval version.

3. The BPF intervals for the x elements are wider than the corresponding BPA intervals; 79% of these intervals have a difference in width with corresponding BPA intervals on the order of 10^{-9} to 10^{-11}. The distribution of BPF - BPA interval widths is as follows:

	difference in width	
	10^{-7}	X
	10^{-8}	XXXXX XXXXX XX
79%	10^{-9}	XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX X
79%	10^{-10}	XXXXX XXXXX XXXXX XXXXX XX
79%	10^{-11}	XXXXX XXXXX XXXXX XXXXX X
	10^{-12}	XXXXX
	10^{-13}	XXX

Observations: (cont.)

4. The distribution of BPA interval widths for the components of the solution vector $\underline{x}$ is as follows:

	10^{-4}	X
	10^{-5}	XXXXX XXXXX XXXXX X
60%	10^{-6}	XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX
60%	10^{-7}	XXXXX XXXXX XXXXX XXXXX XXXXX
	10^{-8}	XXXXX XXXXX XXXXX X
	10^{-9}	XXXX
	10^{-10}	XXX

5. The distribution of BPF interval widths for the components of the solution vector $\underline{x}$ is as follows:

	10^{-4}	X
	10^{-5}	XXXXX XXXXX XXXXX X
60%	10^{-6}	XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX
60%	10^{-7}	XXXXX XXXXX XXXXX XXXXX XXXXX
	10^{-8}	XXXXX XXXXX XXXXX X
	10^{-9}	XXXX
	10^{-10}	XXX

We observe that the distritution of widths of intervals is the same for BPA and BPF versions, therefore further justifying the use of BPF bounds since the BPF bounds also require less execution time than BPA bounds (81% of BPA execution time for this problem).

6. We observe that the computed value of the determinant of A is small ($\approx .361 \times 10^{-20}$) and the extimate of the condition number (x 8176) is fairly large, thus indicating some ill-conditioning in the problem.

The computed results for $\underline{x}$, the residuals, interval widths and midpoints of $\underline{x}$ components, as well as CPU timings are given for the noninterval and interval (BPA and BPF) versions in the attached listings.

9. Problem: To solve for $\underline{x}$ the linear algebraic system of equations

$$A\underline{x} = \underline{b}$$

where A is sparse.

Test data for A is 10 x 10 block tridiagonal matrix with 10 x 10 blocks where

$$A = \begin{bmatrix} C & D & & & & \\ B & C & D & & & \\ & B & \ddots & \ddots & & \\ & & \ddots & \ddots & D & \\ & & & C & D & \\ & & & B & C & \end{bmatrix}$$

with

$$B = \begin{bmatrix} -1 & & & & \\ & -1 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & -1 \end{bmatrix}$$

$$C = \begin{bmatrix} 4 & & & & \\ -1 & 4 & & & \\ & -1 & \ddots & & \\ & & \ddots & \ddots & \\ & & & -1 & 4 \end{bmatrix}$$

$$D = \begin{bmatrix} -1.5 & & & \\ & -1.5 & & \\ & & \ddots & \\ & & & 1.5 \end{bmatrix}$$

Code used and references: NSPIV - a fortran routine which uses sparse Gaussian elimination with partial pivoting and was written by Andrew H. Sherman (University of Texas-Austin). The code is listed in NSPIV: A FORTRAN Subroutine for Sparse Gaussian Elimination with Partial Pivoting, Report TR-65, Department of Computer Sciences, University of Texas, Austin, Texas, January 1977. Also see Algorithms for Sparse Gaussian Elimination with Partial Pivoting, Report No. UIUCUCS-R-76-817, Department of Computer Science, University of Illinois, Urbana, Illinois, 1976. Both reports were written by Andrew H. Sherman.

Conditions under which test was conducted:	The FTN 4.6 FORTRAN compiler was used with no optimization (i.e. opt = 0). Noninterval and interval (both BPA and BPF) cases were run. Timing figures were obtained by using the SECØND function to measure the CPU time for each run, excluding time for I/O. The INTWR routine was used with variables declared interval in IF statements involving interval quantities, the operators EQ, GT, and LT are replaced by SEQ, VGT, and VLT, respectively.
Execution time for noninterval version:	.34 seconds CPU time
Execution time for BPF interval version:	11.665 seconds CPU time
Execution time for BPA interval version:	13.963 seconds CPU time
Observations:	1. Slow interval version (BPA) requires 41.07 times as much CPU time as noninterval version. 2. Fast interval version (BPF) requires 34.31 times as much CPU time as noninterval version. 3. The fast interval version (BPF) requires 83.54% of slow (BPA) interval CPU time. 4. The ratios for fast and slow interval versions to the noninterval version are smaller for this 100 x 100 sparse case than for 100 x 100 nonsparse WES case using DECØMP and SOLVE (in that situation BPA version was 84 times as slow as noninterval version and BPF version is 68 times as slow as noninterval version). As a matter of fact, comparing the specific nonsparse and sparse 100 x 100 cases mentioned here, we note that sparse ratios are 1/2 those of the corresponding nonsparse ratios.

10. Problem:

To apply interval analysis to the solution of a linear system of equations, $A\underline{x} = \underline{b}$, where A is non-singular but nearly singular. We want to observe for slight perturbations of b, the effects on the size and/or midpoint of the generated intervals associated with the resulting solution and the relationship between these values and those corresponding to the original problem.

Code used and references:

DECØMP and SØLVE from Computer Methods for Mathematical Computations by George E. Forsythe, Michael A. Malcolm, and Cleve B. Moler, Prentice-Hall, Englewood Cliffs, New Jersey, 1977, pp. 52-55.

Conditions under which test was conducted:

The FTN 4.6 FORTRAN compiler was used with no optimization (i.e. opt = 0). Non-interval and interval (both BPA and BPF) cases were run. Timing figures were obtained by using the SECØND function to measure the CPU time for each run excluding time for I/O. The INTRD and IWTWR routines were used with all variables declared interval. Inequalities involving interval variables used the interval operators SEQ, VGT, VLT, and VLE to replace the non-interval operators EQ, GT, LT, LE, respectively.

Note: The test problem used in this experiment is defined by G.J. Tee in an article entitled, "A Simple Example of an Ill-Conditioned Matrix," SIGNUM Newsletter, Volume 7, Number 3, October 1972, pp. 19-20.

The suggested test matrix A is of order 3 and has a determinant of 1

$$A = \begin{bmatrix} 11 & 10 & 14 \\ 12 & 11 & -13 \\ 14 & 13 & -66 \end{bmatrix}$$

Let

$$b = \begin{bmatrix} 1.001 \\ 0.999 \\ 1.001 \end{bmatrix}$$

then the exact solution of $A\underline{x} = \underline{b}$ is

$$\underline{x} = \begin{bmatrix} -0.683 \\ 0.843 \\ 0.006 \end{bmatrix}$$

Let

$$\underline{b} + \underline{\delta b} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

then the perturbed equation for this experiment is $A\underline{z} = b + \delta b$.

Computational results:

Noninterval Version* for $A\underline{x} = \underline{b}$

Computed values for $\underline{x}$

opt=0 truncated arithmetic
- -.683000000036458
- .843000000039922
- .00600000000012932

opt=0 rounded arithmetic
- -.682999999999616
- .842999999999577
- .00599999999999803

Noninterval Version* for $A\underline{z} = \underline{b} + \delta\underline{b}$

Computed values for $\underline{x}$

opt=0 truncated arithmetic
- 1.00000000005106
- -1.00000000005591
- $-1.81188397628383 \times 10^{-12}$

opt=0 rounded arithmetic
- .999999999996479
- -.999999999996138
- $.133226762955049 \times 10^{-13}$

Computational value of determinant of A = .999999999999545.

Estimate of condition number of A = $.164135602084493 \times 10^{6}$.

*Noninterval versions were also run under opt = 1,2, and 2 U0 with rounded and truncated arithmetic; truncated results under all options gave exact agreement as did rounded results under all options.

Computational results: (cont.)

Interval Version* for $\underline{A}\underline{x} = \underline{b}$

BPA computed intervals for $\underline{x}$

(-.68000000274032,-.682999999775020)

(.842999999762525,.843000000291192)

(.00599999999922828,.00600000000094667)

BPF computed intervals for $\underline{x}$

(-.683000000277098,-.682999999773805)

(.842999999761271,.843000000294467)

(.00599999999922420,.00600000000095733)

BPA	midpoint of x	length of x
	$-.683000000024528$	$.499010610610640 \times 10^{-9}$
	$.843000000026858$	$.528665111687587 \times 10^{-9}$
	$.600000000008749 \times 10^{-2}$	$.161837544193290 \times 10^{-11}$

BPF	midpoint of x	length of x
	$-.683000000025451$	$.503291630593594 \times 10^{-9}$
	$.843000000027871$	$.533194821628058 \times 10^{-9}$
	$.600000000009077 \times 10^{-2}$	$.173311365259110 \times 10^{-11}$

Total CPU time for BPA version $\approx$.198 seconds

Total CPU time for BPF version $\approx$.164 seconds

Total CPU time for Noninterval version $\approx$.005 seconds

*Interval versions were run for BPA and BPF under opt=0,1,2, and $2\cup 0$ with rounded and truncated arithmetic. Truncated and rounded interval bounds were identical under all options for BPA and likewise for all BPF versions.

Computational results: (cont.)

Interval Version* for $Az = b + \delta b$

BPA computed intervals for z

(.999999999947554,1.00000000011969)

(-1.00000000012800,-.999999999945636)

$(-.415667500471075 \times 10^{-12}, .175859327122378 \times 10^{-12})$

BPF computed intervals for z

(.999999999944471,1.00000000012126)

(-1.00000000012963,-.999999999942339)

$(-.420996570989935 \times 10^{-12}, .186517468160098 \times 10^{-12})$

BPA	midpoint of z	length of z
	1.00000000003362	$.172128977737884 \times 10^{-9}$
	-1.00000000003681	$.182353687705472 \times 10^{-9}$
	$.119904086674349 \times 10^{-12}$	$.59126827593453 \times 10^{-12}$

BPF	midpoint of z	length of z
	1.00000000003286	$.17678303]657113 \times 10^{-9}$
	-1.00000000003598	$.187284854291647 \times 10^{-9}$
	$-.117239551414919 \times 10^{-12}$	$.617514039150033 \times 10^{-12}$

Total CPU time for BPA version $\approx$ .197 seconds

Total CPU time for BPF version $\approx$ .163 seconds

Total CPU time for noninterval version $\approx$ .005 seconds

*Interval versions were run for BPA and BPF under opt=0,1,2, and $2 \cup 0$ with rounded and truncated arithmetic; truncated and rounded interval bounds are identical under all options for BPA and likewise for all APF versions.

Observations:

1. $\frac{\text{BPF CPU time}}{\text{BPF CPU time}} \lesssim 83\%$

   (This compares with 81% for the same code when used with 100 X 100 WES case.)

2. $\frac{\text{BPF CPU time}}{\text{Noninterval CPU time}} \approx 33$

   (This compares with 68 for the same code when used with 100 X 100 WES case.)

3. $\frac{\text{BPA CPU time}}{\text{Noninterval CPU time}} \approx 39.5$

   (This compares with 83.9 for the same code when used with 100 X 100 WES case.)

4. The ratios of the widths of the intervals corresponding to the x and z solutions are approximately the same for all components for BPA and BPF versions, i.e.

$$\frac{\text{width of } x_i \text{ interval}}{\text{width of } z_i \text{ interval}} \approx 2.9$$

   Thus the intervals associated with the perturbed problem are actually smaller than the corresponding intervals associated with the x solution and therefore do not alert the user. Although the right-hand side of $A\underline{z}=\underline{b}+\delta\underline{b}$ has been perturbed only .1% from that of b in $A\underline{x}=\underline{b}$, the solution z is actually perturbed $\approx$ >220% (see Observation 5). Thus, interval widths in this experiment do not warn the user of the ill-conditioning of this problem (note that the estimate of the condition number of A is 164,136 which indicates the problem is ill-conditioned). As a matter of fact, the interval lengths for x and z computed components are very small (on the order of 10^{-9} to 10^{-12}) and thereby do not alert the user of numerical difficulty.

5. The absolute value of the relative error of the midpoints of the z intervals are better indicators of the numerical fluctuation, i.e.

$$\left| \frac{z_i - x_i}{x_i} \right|$$

$$\left| \frac{z_1 - x_1}{x_1} \right| \approx 2.464$$

$$\left| \frac{z_2 - x_2}{x_2} \right| \approx 2.186$$

$$\left| \frac{z_3 - x_3}{x_3} \right| \approx 1.00$$

Observations: (cont.)

We note particularly for the larger components of x, the absolute value of the relative error in the midpoint of z is on the order of >218% (i.e. for first z component $\approx$ 246% and for second z component $\approx$ 219%) and this is caused by a .1% variation in the right-hand side of the original system $A\underline{x}=\underline{b}$. Thus the midpoints of the z component intervals are better indicators of the numerical difficulty than the widths of these intervals.

6. Examination of the absolute value of the relative error for the z values in the noninterval version of this problem indicates approximately the smae values given in observation 5 above when dealing with midpoints of the generated solution intervals. Thus the noninterval solution was just as good an indicator of numerical difficulty as were the midpoints of the interval solution and has the added advantage of requiring less time and storage.

7. Note that the residuals associated with the z solution (see attached listings) indicate values on the order of 10^{-13} which might deceive the user into believing that the results are very accurate. The user should realize that for ill-conditioned problems such as this one (recall that the estimate of the condition number is 164,136), small residuals do not indicate that the computed solution is close to the exact solution.

8. The interval solution computed for the perturbed problem (i.e. $A\underline{z}=\underline{b}+\delta\underline{b}$) does not warn the user of numerical difficulty mainly because the widths of the resultant solution bounds are dependent upon the number, type, and sequence of arithmetic operations involved in the linear equation solver but do not take into account the condition number of the amatrix in the problem.

9. The attached listings for the noninterval, BPA, and BPF versions of this test for the original and perturbed problems give the computed solution, timings, and residuals.

11. Problem: To apply the WES SPLINE interpolation routine (in both non-interval and interval versions) to WES 4-point and 10-point data.

Code used and references:

The WES SPLINE routine is adapted from work by Greville (see TS Report No. 893, U.S. Army Mathematic Research Center, University of Wisconsin, June 1968). The documentation for the WES SPLINE routine is contained in WES Miscellaneous Report No. 0-71-2 (July 1971) which is available from the Computer Analysis Branch, ADP Center, U.S. Army Waterways Experiment Station, Vicksburg, Mississippi. The WES routine SPLINT written by Jay Cheek of WES is also used in this experiment to compute Y and Y' values for given X values based on the spline generated by routine SPLINE. The SPLINE routine is discussed in the aforementioned WES paper and documentation is given in the attached listing.

Conditions under which the test was conducted:

The FTN 4.6 FØRTRAN compiler was used with no optimization (i.e. opt = 0). Both BPA and BPF interval versions were run to compare with the noninterval case. Interval variables were output using the IWTWR routine. Timing figures were obtained by using the SECØND function to measure the CPU time. Inequalities involving interval variables used the interval operators SEQ, VGT, and VLT to replace the noninterval operators EQ, GT, and LT, respectively. WES provided the 4-point and 10-point data used with the test. SPLINT was used to produce Y and Y' values for corresponding X values spaced at increments of .25 across the entire X range specified by the 4-point and 10-point data. The 4-point data spans X=1.6 to 10 and the 10-point data spans X=0 to 26. For X values outside the range of the given data set, routine SPLINT extrapolates linearly from the first or last data point using the slope of the spline at that endpoint.

Computational results:

CPU timing (in seconds) for SPLINE: 4-point Case

	With Function Call Overhead	Without Function Call Overhead
Noninterval	.007	.002
BPA	.471	.467
BPF	.394	.389

CPU timing (in seconds) for SPLINE: 10-point Case

	With Function Call Overhead	Without Function Call Overhead
Noninterval	.020	.014
BPA	2.690	2.684
BPF	2.161	2.157

Additional computational results including X, Y, and Y' values and their associated intervals (along with interval width and midpoint) for X values across each data range can be found in the attached

listing. The numerical behavior of these quantities is discussed below.

Observations:

The distribution of interval widths for Y and Y' are given below for BPA and BPF versions of the 4-point and 10-point cases.

I. Distribution of interval widths for 4-point case

A) Y intervals for BPA version

Interval Width	
10^{-11}	XXXXX XXXXX XX
10^{-12}	XXXXX XXXXX XXXXX XXXXX XX
10^{-13}	XX
0	X

B) Y intervals for BPF version

Interval Width	
10^{-11}	XXXXX XXXXX XXXX
10^{-12}	XXXXX XXXXX XXXXX XXXXX X
10^{-13}	X
0	X

C) Y' intervals for BPA version

Interval Width	
10^{-11}	XXXXX XXXXX XXXX
10^{-12}	XXXXX XXXXX XXXXX XXXXX XXX

D) Y' Intervals for BPF version

Interval Width	
10^{-11}	XXXXX XXXXX XXXXX XXX
10^{-12}	XXXXX XXXXX XXXXX XXXX

II. Distribution of interval widths for 10-point case

A) Y intervals for BPA version

Interval Width	
10^{-7}	
10^{-8}	XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXX
10^{-9}	XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX X
10^{-10}	XXXXX XXXXX XX
10^{-11}	XXX
10^{-12}	X
0	XXXXX XXXX

Observations: (cont.)

B) Y intervals for BPF version

10^{-7}	XXXXX XXXXX XXXXX XXXXX XX
10^{-8}	XXXXX XXXXX XXXXX XXXXX XXXX
10^{-9}	XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXX
10^{-10}	XXXXX X
10^{-11}	
10^{-12}	X
0	XXXXX XXXX

C) Y' intervals for BPA version

10^{-8}	XXXXX XXXXX XXXXX XXXXX XXXXX XXX
10^{-9}	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
10^{-10}	XXXXX XXXXX XXXXX XX
10^{-11}	X

D) Y' intervals for BPF version

Interval Width	
10^{-8}	XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX XXXXX
10^{-9}	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
10^{-10}	XXXXX XXXX
10^{-11}	

1. From the tabulated results on interval width distributions given in I and II above, we can make several observations:

   a) Comparing 4-point Y and Y' widths - Y (for both BPA and BPF versions) intervals vary insignificantly in width from corresponding Y' intervals for same X values.

   b) Comparing 10-point Y and Y' widths - same as for a) above.

   c) Comparing 4-point Y and 10-point Y widths - widths are smaller for 4-point case than for 10-point case (see IA, IB, IIA, IIB distributions).

   d) Comparing 4-point Y' and 10-point Y' widths - same as for c) above (see IC, ID, IIC, IID distributions).

   e) Comparing 4-point Y for BPA and BPF versions - widths vary insignificantly (see IA, IB)

   f) Comparing 10-point Y for BPA and BPF versions - BPF widths more noticeably wider than in 4-point case (see IIA, IIB)

   g) Comparing 4-point Y' for BPA and BPF versions - very slight increase in width of BPF intervals over BPA intervals (see IC, ID)

Observations: (cont.)

h) Comparing 10-point Y' for BPA and BPF versions - more noticeable increase in widths for BPF over BPA than in 4-point case (see IIC, IID)

2. The observed increases in interval widths mentioned in comparisons c,d,f, and h of observation 1 above are reasonable considering that more computations are performed for the 10-point case than for the 4-point case, thus widening some of the computed intervals.

3. The widths of the intervals associated with Y and Y' (in the range of 10^{-7} to 10^{-13}) make plotting somewhat impractical. Plots (based on 3 significant figures) of the noninterval 4-point and 10-point cases for Y and Y' are attached.

Our next few observations deal with execution time of the SPLINE routine.

	BPA time	BPF time	BPF time
	Noninterval time	Noninterval time	BPA time
4-point case without overhead	23.35	19.45	.833
4-point case with overhead	67.29	56.29	.837
10-point case without overhead	191.71	154.07	.804
10-point case with overhead	134.50	108.05	.803

4. From the tabulated results above, we see that the BPF interval versions for both the 4-point and 10-point data consistently required 80% of the execution time of their BPA interval counterparts and still produced competitive interval widths.

5. The noticeable differences between the ratios computed with function call overhead and without function call overhead can probably be attributed to the inability of the SECOND function to accurately measure such small execution times (all our values were under 3 seconds). We note that the ratio of interval to noninterval versions is significantly less for the 4-point case than for the 10-point case and there are fluctuations in these ratios depending upon whether or not function call overhead is included.

6. A more consistent execution time pattern emerges if we compare ratios of 10-point to 4-point CPU values for each version tested.

	10-point BPA time	10-point BPF time	10-point Noninterval time
with fcn call overhead	5.7	5.5	2.85
without fcn call overhead	5.7	5.5	7.00

Observations: (cont.)

We therefore see that both interval versions (with or without function call overhead) for the 10-point case require approximately 5.6 times the execution time as their corresponding 4-point interval version, whereas the non-interval 10-point version can take either 2.85 or 7 times as much execution time as the corresponding 4-point non-interval time. There is probably more fluctuation in these noninterval ratios due to the small CPU times involved and the inability to measure them precisely.

7. The results from the WES 4-point and 10-point test cases do not seem to strongly justify the use of interval analysis here; the midpoints of the Y and Y' intervals are approximately equal to the computed non-interval values. Possibly tight resultant intervals here might reassure the user of a bound above and below the true solution for some physical problem but I would question how good that bound is, since the given data was to no more than 3 significant figures and it is doubtful that such figures, in practice, would be exact. Interval versions of SPLINE might, however, be applicable to situations in which there is a spike and/or discontinuity in the measured data or where a tolerance (interval) is given to express the uncertainty in measuring one or more data points, in which case we might see the effect of such tolerances on the intervals associated with the use of the generated spline. Even for such a case with one or more tolerances, a least squares approach would probably be more appropriate to the problem than the spline technique. Further testing with a wide variety of practical data sets of varying sizes is advised before determining the efficacy of the interval version of SPLINE. I would also suggest that WES consider using Alan Cline's splines under tension approach (see "Scalar - and Planar - Valued Curve Fitting Using Splines Under Tension", Communications of the Association for Computing Machinery, Volume 17, Number 4, April 1974) for comparison purposes with the routine used in this experiment.

12. Problem: To sum the first 129 terms of (1/x)**(I-1) backward and forward with interval, single and double precision arithmetic, and using WES test data for x values.

Code used and references:

A FØRTRAN routine provided by WES

Conditions under which test was conducted:

The FTN 4.6 FØRTRAN compiler was used with no optimization (i.e. opt = 0). Noninterval (single and double precision) and interval (both BPA and BPF) cases were run. The WRITE routine was used with variables declared interval.

Computational Results:

Case for x = 2.

	Forward Summation
Noninterval (single precision)	1.999999999999999
Noninterval (double precision)	2.000000000000000000000000000
Interval (BPA)	(1.9999999999999,2.00000000000114)
Interval (BPF)	(1.999999999999999,2.000000000000003)

	Backward Summation
Noninterval (single precision)	1.99999999999999
Noninterval (double precision)	2.000000000000000000000000000
Interval (BPA)	(1.999999999999999,2.000000000000000)
Interval (BPF)	(1.999999999999999,2.000000000000003)

Case for 2 = -2.

	Forward Summation
Noninterval (single precision)	.666666666666583
Noninterval (double precision)	.666666666666666666666666667
Interval (BPA)	(.666666666666526,.666666666666810)
Interval (BPF)	(.666666666666494,.666666666666842)

	Backward Summation
Noninterval (single precision)	.666666666666664
Noninterval (double precision)	.666666666666666666666666667
Interval (BPA)	(.666666666666664,.666666666666668)
Interval (BPF)	(.666666666666654,.666666666666679)

Computational Results: (cont.)

Case for x = 2.1

	Forward Summation
Noninterval (single precision)	1.90909090909077
Noninterval (double precision)	1.90909090909091378871230254 5
Interval (BPA)	(1.90909090909077,1.90909090909168)
Interval (BPF)	(1.90909090909077,1.90909090909140)

	Backward Summation
Noninterval (single precision)	1.90909090909091
Noninterval (double precision)	1.90909090909091378871230254 6
Interval (BPA)	(1.90909090909091,1.90909090909092)
Interval (BPF)	(1.90909090909090,1.90909090909091)

Case for x = -2.1

	Forward Summation
Noninterval (single precision)	.677419354838555
Noninterval (double precision)	.67741935483870908591655711 88
Interval (BPA)	(.677419354838559,.677419354838875)
Interval (BPF)	(.677419354838559,.677419354838875)

	Backward Summation
Noninterval (single precision)	.677419354838705
Noninterval (double precision)	.67741935483870908591655711 96
Interval (BPA)	(.677419354838705,.677519354837 15)
Interval (BPF)	(.677419354838705,.677419354838715)

Case for x = 3.0

	Forward Summation
Noninterval (single precision)	1.4999999999991
Noninterval (double precision)	1.49999999999999999999999999
Interval (BPA)	(1.4999999999991,1.50000000000082)
Interval (BPF)	(1.4999999999991,1.50000000000033)

	Backward Summation
Noninterval (single precision)	1.49999999999999
Noninterval (double precision)	1.50000000000000000000000000 0
Interval (BPA)	(1.49999999999999,1.50000000000001)
Interval (BPF)	(1.49999999999999,1.50000000000001)

Computational Results: (cont.)

Case for x = -3.0

	Forward Summation
Noninterval (single precision)	.749999999999901
Noninterval (double precision)	.749999999999901
Noninterval (double precision)	.749999999999999999999999999949
Interval (BPA)	(.749999999999780,.750000000000000)
Interval (BPF)	(.749999999999901,.750000000000117)

	Backward Summation
Noninterval (single precision)	.750000000000000
Noninterval (double precision)	.750000000000000000000000000000
Interval (BPA)	(.749999999999996,.750000000000004)
Interval (BPF)	(.749999999999996,.750000000000004)

Case for x = 5.0

	Forward Summation
Noninterval (single precision)	1.24999999999992
Noninterval (double precision)	1.24999999999999999999999999
Interval (BPA)	(1.24999999999992,1.25000000000083)
Interval (BPF)	(1.24999999999992,1.25000000000021)

	Backward Summation
Noninterval (single precision)	1.24999999999999
Noninterval (double precision)	1.25000000000000000000000000
Interval (BPA)	(1.24999999999999,1.25000000000001)
Interval (BPF)	(1.24999999999999,1.25000000000001)

Case for x = -5.0

	Forward Summation
Noninterval (single precision)	.833333333333258
Noninterval (double precision)	.833333333333333333333333329
Interval (BPA)	(.833333333333105,.833333333333560)
Interval (BPF)	(.833333333333258,.833333333333403)

	Backward Summation
Noninterval (single precision)	.833333333333329
Noninterval (double precision)	.833333333333333333333333333
Interval (BPA)	(.833333333333332,.833333333333336)
Interval (BPF)	(.833333333333332,.833333333333336)

Computational Results: (cont.)

Case for x = 7.0

	Forward Summation
Noninterval (single precision)	1.16666666666660
Noninterval (double precision)	1.16666666666666666666666666
Interval (BPA)	(1.16666666666660,1.66666666666751)
Interval (BPF)	(1.16666666666660,1.16666666666683)

	Backward Summation
Noninterval (single precision)	1.66666666666666
Noninterval (double precision)	1.66666666666666666666666667
Interval (BPA)	(1.16666666666666,1.16666666666667)
Interval (BPF)	(1.16666666666666,1.16666666666667)

Case for x = -7.0

	Forward Summation
Noninterval (single precision)	.874999999999936
Noninterval (double precision)	.87499999999999999999999999997
Interval (BPA)	(.874999999999769,.875000000000224)
Interval (BPF)	(.874999999999936,.875000000000057)

	Backward Summation
Noninterval (single precision)	.875000000000000
Noninterval (double precision)	.87500000000000000000000000000
Interval (BPA)	(.874999999999996,.875000000000004)
Interval (BPF)	(.874999999999996,.875000000000004)

Case for x = 10.0

	Forward Summation
Noninterval (single precision)	1.11111111111106
Noninterval (double precision)	1.11111111111111111111111111
Interval (BPA)	(1.11111111111106,1.11111111111197)
Interval (BPF)	(1.11111111111106,1.11111111111126)

	Backward Summation
Noninterval (single precision)	1.11111111111111
Noninterval (double precision)	1.11111111111111111111111111
Interval (BPA)	(1.11111111111111,1.11111111111111)
Interval (BPF)	(1.11111111111111,1.11111111111111)

Computational Results: (cont.)

Case for x = -10.0

	Forward Summation
Noninterval (single precision)	.909090909090860
Noninterval (double precision)	.909090909090909090909090089
Interval (BPA)	(.909090909090683m,909090909091137)
Interval (BPF)	(.909090909090860,.909090909090960)

	Backward Summation
Noninterval (single precision)	.909090909090907
Noninterval (double precision)	.909090909090909090909090091
Interval (BPA)	(.909090909090907,.909090909090910)
Interval (BPF)	(.909090909090907,.909090909090910)

Case for x = 11.0

	Forward Summation
Noninterval (single precision	1.09999999999996
Noninterval (double precision	1.10000000000000000000000000
Interval (BPA)	1.09999999999996,1.10000000000087)
Interval (BPF)	(1.09999999999996,1.10000000000015)

	Backward Summation
Noninterval (single precision)	1.00000000000000
Noninterval (double precision)	1.10000000000000000000000000
Interval (BPA)	(1.09999999999999,1.10000000000000)
Interval (BPF)	(1.09999999999999,1.10000000000000)

Case for -11.0

	Forward Summation
Noninterval (single precision)	.916666666666618
Noninterval (double precision)	.916666666666666666666666664
Interval (BPA)	(.916666666666440,.916666666666714)
Interval (BPF)	(.916666666666618,.916666666666714)

	Backward Summation
Noninterval (single precision)	.916666666666664
Noninterval (double precision)	.916666666666666666666666667
Interval (BPA)	(.916666666666664 ,.916666666666668)
Interval (BPF)	(.916666666666664,.916666666666668)

Computational Results: (cont.)

Case for x = 13.0

	Forward Summation
Noninterval (single precision)	1.00000000076923
Noninterval (double precision)	1.000000000769230769822485208
Interval (BPA)	(1.00000000076923,1.00000000077014)
Interval (BPF)	(1.00000000076923,1.00000000076925)

	Backward Summation
Noninterval (single precision)	1.00000000076923
Noninterval (double precision)	1.000000000769230769822485208
Interval (BPA)	(1.00000000076923,1.00000000076923)
Interval (BPF)	(1.00000000076923,1.00000000076923)

Case for x = -13.0

	Forward Summation
Noninterval (single precision)	.99999992307682
Noninterval (double precision)	.999999992307692366863904870 3
Interval (BPA)	(.999999992307465,.999999992307920)
Interval (BPF)	(.999999992307686,.999999992307696)

	Backward Summation
Noninterval (single precision)	.999999992307686
Noninterval (double precision)	.999999992307692366863904870 3
Interval (BPA)	(.999999992307689,.999999992307693)
Interval (BPF)	(.999999992307689,.999999992307693)

Observations:

1. The forward and backward double precision summations usually agree or differ slightly (by no more than two decimal digits) with the backward double precision sum being more accurate.

2. Single precision summations differ by at most three decimal digits with the backward sums being more accurate.

3. For the interval cases, BPA bounds tend to be tighter (or agree with) than BPF bounds.

4. BPA and BPF forward sum bounds are larger than BPA and BPF backward sum bounds, as is to be expected since there is more error propagation in the forward summing process.

5. With respect to the given test program and the WES data, interval analysis cannot be justified here on our CDC machine. The double precision result is more accurate and faster than interval analysis. The timing ratios (CPU time) of double precision to single precision summing (both backward and forward) ranges from $\approx$ 1.25 to $\approx$ 2.5, whereas the timing ratios of interval versions (both BPF and BPA in forward and backward summing) to single precision summing ranged from $\approx$ 36 to $\approx$ 74.

13. Problem: To subtract $1./2.^{48}$ from 1.0, repeatedly square the difference, and sum the squares. Forward and backward sums are computed for single precision, double precision, and interval (BPA and BPF) cases. This problem was defined by WES for the case of $1./2.^{27}$, and I have also chosen to run the same test with $1./2.^{48}$ because this quantity is more appropriate to a CDC machine and roughly analogous to $1./2.^{27}$ for the Univac 1110. See my other write-up for details of a Univac 1110 case run on a CDC machine.

Code used and references:

A FØRTRAN routine by WES and modified to include double precision case and use the quantity $1./2.^{48}$ instead of $1./2.^{27}$.

Conditions under which test was conducted:

The FTN 4.6 FØRTRAN compiler was used with no optimization (i.e. opt = 0). Noninterval (single and double precision) and interval (both BPA and BPF) cases were run. The WRITE routine was used with variables declared interval. The standard (default) option of chopped arithmetic was used. The program was then run again, using the CDC ROUND option (see observation 11 for comparison of rounded and truncated results).

Computational results:

	Forward Sum
Noninterval (single precision)	39.9883040991076
Noninterval (double precision)	39.992197661183123269072993 52
Interval (BPA)	(39.9921976607195,39.9921976614171)
Interval (BPF)	(39.9921976607195,40.0000000000000)

	Backward Sum
Noninterval (single precision)	39.9883040991078
Noninterval (double precision)	39.992197661183123269072993 53)
Interval (BPA)	(39.9921976607195,39.9921976614162)

		Terms of Sequence
1	Noninterval (single precision)	.999999999999989
	Noninterval (double precision)	.999999999999992895472642399 0
	Interval (BPA)	(.999999999999993,.999999999999993)
	Interval (BPF)	(.999999999999993,.999999999999993)

Computational Results: (cont.)

		Terms of Sequence
2	Noninterval (single precision)	.999999999999979
	Noninterval (double precision)	.999999999999857891452847980
	Interval (BPA)	(.999999999999986,.999999999999986)
	Interval (BPF)	(.999999999999986,.999999999999993)
3	Noninterval (single precision)	.999999999999957
	Noninterval (double precision)	.999999999999715782905695963)
	Interval (BPA)	(.999999999999972,.999999999999972)
	Interval (BPF)	(.999999999999972,.999999999999993)
4	Noninterval (single precision)	.999999999999915
	Noninterval (double precision)	.999999999999431565811391933
	Interval (BPA)	(.999999999999943,.999999999999943)
	Interval (BPF)	(.999999999999943,.999999999999993)
5	Noninterval (single precision)	.999999999999829
	Noninterval (double precision)	.999999999998863131622783899
	Interval (BPA)	(.999999999999886,.999999999999886)
	Interval (BPF)	(.999999999999886,.999999999999993)
6	Noninterval (single precision)	.999999999999659
	Noninterval (double precision)	.999999999997726263245567926
	Interval (BPA)	(.999999999999773,.999999999999931)
	Interval (BPF)	(.999999999999773,.999999999999993)
7	Noninterval (single precision)	.999999999999931
	Noninterval (double precision)	.999999999995452526491136369
	Interval (BPA)	(.999999999999545,.999999999999545)
	Interval (BPF)	(.999999999999545,.999999999999993)
8	Noninterval (single precision)	.999999999998636
	Noninterval (double precision)	.999999999990905052982274805
	Interval (BPA)	(.999999999999091,.999999999999091)
	Interval (BPF)	(.999999999999091,.999999999999993)
9	Noninterval (single precision)	.999999999997272
	Noninterval (double precision)	.999999999981810105964557882
	Interval (BPA)	(.999999999998181,.999999999998181)
	Interval (BPF)	(.999999999998181,.999999999999993)

Computational Results: (cont.)

		Terms of Sequence
10	Noninterval (single precision)	.999999999994543
	Noninterval (double precision)	.999999999963620211929148851
	Interval (BPA)	(.999999999996362,.999999999996362)
	Interval (BPF)	(.999999999996362,.999999999999993)
11	Noninterval (single precision)	.999999999989086
	Noninterval (double precision)	.999999999927240423858430051
	Interval (BPA)	(.999999999992724,.999999999992724)
	Interval (BPF)	(.999999999992724,.999999999999993)
12	Noninterval (single precision)	.999999999978172
	Noninterval (double precision)	.999999999854480847717389498
	Interval (BPA)	(.999999999985448,.999999999985448)
	Interval (BPF)	(.999999999985448,.999999999999993)
13	Noninterval (single precision)	.999999999956344
	Noninterval (double precision)	.999999999708961695436896
	Interval (BPA)	(.999999999970896,.999999999970896)
	Interval (BPF)	(.999999999970896,.999999999999993)
14	Noninterval (single precision)	.999999999912689
	Noninterval (double precision)	.999999999417923390882263485
	Interval (BPA)	(.999999999941792,.999999999941792)
	Interval (BPF)	(.999999999941792,.999999999999993)
15	Noninterval (single precision)	.999999999925377
	Noninterval (double precision)	.999999999835846781798408288
	Interval (BPA)	(.999999999883585,.999999999883585)
	Interval (BPF)	(.999999999883585,.999999999999993)
16	Noninterval (single precision)	.999999999650754
	Noninterval (double precision)	.999999999767169356373234184 7
	Interval (BPA)	(.999999999767169,.999999999767169)
	Interval (BPF)	(.999999999767169,.999999999999993)
17	Noninterval (single precision)	.999999999301508
	Noninterval (double precision)	.999999999534338712800678477 9
	Interval (BPA)	(.999999999534339,.999999999534339)
	Interval (BPF)	(.999999999534339,.999999999999993)

Computational Results: (cont.)

		Terms of Sequence
18	Noninterval (single precision)	.999999998603016
	Noninterval (double precision)	.999999999068677425818197 3902
	Interval (BPA)	(.999999999068677,.999999999068677)
	Interval (BPF)	(.999999999068677,.999999999999993)
19	Noninterval (single precision)	.999999997206032
	Noninterval (double precision)	.999999998137354852503756 5177
	Interval (BPA)	(.999999998137355,.999999998137355)
	Interval (BPF)	(.999999998137355,.999999999999993)
20	Noninterval (single precision)	.999999994412065
	Noninterval (double precision)	.999999996274709708476959 9808
	Interval (BPA)	(.999999996274710,.999999996274710)
	Interval (BPF)	(.999999996274170,.999999999999993)
21	Noninterval (single precision)	.999999988824129
	Noninterval (double precision)	.999999992549419430831707 7177
	Interval (BPA)	(.999999992549419,.999999992549419)
	Interval (BPF)	(.999999992549419,.999999999999993)
22	Noninterval (single precision)	.999999977648258
	Noninterval (double precision)	.999999985098838917174566 2531
	Interval (BPA)	(.999999985098839,.999999985098839)
	Interval (BPF)	(.999999985098839,.999999999999993)
23	Noninterval (single precision)	.999999955296516
	Noninterval (double precision)	.999999970197678056393734 1224
	Interval (BPA)	(.999999970197678,.999999970197678)
	Interval (BPF)	(.999999970197678,.999999999999993)
24	Noninterval (single precision)	.999999910593033
	Noninterval (double precision)	.999999403953570009658614 752
	Interval (BPA)	(.999999940395335,.999999940395359)
	Interval (BPF)	(.999999940395355,.999999999999993)
25	Noninterval (single precision)	.999999821186073
	Noninterval (double precision)	.999999880790717554645189 9928
	Interval (BPA)	(.999999880790714,.999999880790721)
	Interval (BPF)	(.999999880790714,.999999999999993)

Computational Results: (cont.)

		Terms of Sequence
26	Noninterval (single precision)	.999999642372174
	Noninterval (double precision)	.999999761581449320143401 1219
	Interval (BPA)	(.999999761581439,.999999761581456)
	Interval (BPF)	(.999999761581439,.999999999999993)
27	Noninterval (single precision)	.999999284744472
	Noninterval (double precision)	.999999523162955483692110 5270
	Interval (BPA)	(.999999523162931,.999999523162970)
	Interval (BPF)	(.999999523162931,.999999999999993)
28	Noninterval (single precision)	.999998569489453
	Noninterval (double precision)	.999999046326138340951244 1013
	Interval (BPA)	(.999999046326085,.999999046326167)
	Interval (BPF)	(.999999046326085,.999999999999993)
29	Noninterval (single precision)	.999997138980948
	Noninterval (double precision)	.999998092653186175736899 8850
	Interval (BPA)	(.999998092653076,.999998092653243)
	Interval (BPF)	(.999998092653076,.999999999999993)
30	Noninterval (single precision)	.999994277970078
	Noninterval (double precision)	.999996185310010323342005 3381
	Interval (BPA)	(.999996185309787,.999996185310124)
	Interval (BPF)	(.999996185309787,.999999999999993)
31	Noninterval (single precision)	.999998555972894
	Noninterval (double precision)	.999992370634572506401 3499773
	Interval (BPA)	(.999992370634121,.999992370634800)
	Interval (BPF)	(.999992370634121,.999999999999993)
32	Noninterval (single precision)	.999977112076753
	Noninterval (double precision)	.999984741327352229628 9345357
	Interval (BPA)	(.999984741326447,.999984741327808)
	Interval (BPF)	(.999984741326447,.999999999999993)
33	Noninterval (single precision)	.999954224677360
	Noninterval (double precision)	.999969482887431550229 6847379
	Interval (BPA)	(.999969482885721,.999969482888442)
	Interval (BPF)	(.999969482885721,.999999999999993)

Computational Results: (cont.)

		Terms of Sequence
34	Noninterval (single precision)	.999908451450100
	Noninterval (double precision)	.999938966706357253871381 9093
	Interval (BPA)	(.999938966702736,.999938966708179)
	Interval (BPF)	(.999938966702736,.999999999999993)
35	Noninterval (single precision)	.999816911281336
	Noninterval (double precision)	.999877937137777440624438 8169
	Interval (BPA)	(.999877937130535,.999877937141402)
	Interval (BPF)	(.999877937130535,.999999999999993)
36	Noninterval (single precision)	.999633856084149
	Noninterval (double precision)	.999755889174897215212390 3751
	Interval (BPA)	(.999755889160411,.999755889182183)
	Interval (BPF)	(.999755889160411,.999999999999993)
37	Noninterval (single precision)	.999267846229664
	Noninterval (double precision)	.999511837939889362787164 3624
	Interval (BPA)	(.999511837910923,.999511837954458)
	Interval (BPF)	(.999511837910923,.999999999999993)
38	Noninterval (single precision)	.998536228508470
	Noninterval (double precision)	.999023914181975657035708 1981
	Interval (BPA)	(.999023914124070,.999023914211101)
	Interval (BPF)	(.999023914124070,.999999999999993)
39	Noninterval (single precision)	.997074599643920
	Noninterval (double precision)	.998048781107475462322184 8234
	Interval (BPA)	(.998048780991777,.998048781165672)
	Interval (BPF)	(.998048780991777,.999999999999993)
40	Noninterval (single precision)	.994157757255081
	Noninterval (double precision)	.996101369477011746 9276869595
	Interval (BPA)	(.996101369239170,.996101369586285)
	Interval (BPF)	(.996101369239170,.999999999999993)

Observations:

1. Backward and forward double precision sums (noninterval) agree to 14 decimal digits, as do backward and forward single precision sums (noninterval).

2. Both single precision sums (backward and forward) agree to 3 decimal digits with double precision result. (This compares with 9 decimal digit agreement for WES case using $1./2.^{27}$ instead of $1./2.^{48}$.)

3. The lower bound for both interval versions of the sum agree to 3 decimal digits with the single precision versions for backward sums. (This compares with 15 decimal digit agreement for WES case using $1./2.^{27}$ instead of $1./2.^{48}$.)

4. The lower bound for both interval versions of each term in the sequence agrees to a steadily decreasing number of decimal digits (13 digits to 2 digits) with the single precision results for all 40 terms. (This compares with 15 decimal digit agreement for WES case using $1./2.^{27}$ instead of $1./2.^{48}$.)

5. The BPF intervals for each term are equal to or larger than the corresponding BPA intervals.

6. The BPA intervals for the terms of the sequence are degenerated through term 23. (This compares with only two degenerated intervals for WES case using $1./2.^{27}$.)

7. The number of decimal digits in agreement for upper and lower interval bounds on the terms of the sequence drifts downward as follows: 15 digits (up to term 23), 14 digits (term 24), 13 digits (term 25), 12 digits (term 28), 10 digits (term 30), 9 digits (term 38), 8 digits (term 39). (This compares with a decrease from 15 digits to 0 digits for WES case with $1./2.^{27}$, but for that case decrease started by term 4, whereas here decrease doesn't start until term 24 and never goes below 8 digits.)

8. As each term in the sequence is computed, the upper and lower interval bounds are monotonicly decreasing, i.e., the sequence of lower interval bounds is steadily decreasing from terms 1 to 40 and the sequence of upper interval bounds is also steadily decreasing from terms 1 to 40. The resulting intervals for the terms of the sequence are drifting to the left, however, unlike the WES case with $1./2.^{27}$, the intervals for $1./2.^{48}$ terms do not collapse to near zero; this is because the difference between 1 and $1/2^{48}$ is much closer to 1 than the difference between 1 and $1/2^{27}$; hence powers in the former case are larger than in the latter case.

Observations: (cont.)

9. The single precision resultant of the sum is not contained in any of the computed intervals tot the sum. (Recall for WES case with $1./2.^{27}$ it was the double precision resultant of the sum rounded or chopped to single precision that was not contained in any of the computed intervals for the sum.

10. The number of decimal digits in agreement in the corresponding upper and lower interval bounds for the sum is 13 and 15, respectively. (This compares with 13 [for both bounds] for the WES case for $1./2.^{27}$.) For each interval for the backward and forward sums there is a 10 decimal digit agreement between upper and lower bounds. (This compares with 8 decimal digit agreement in WES case for $1./2.^{27}$.)

11. When the program was run with the CDC rounded option we get the results given in the following table (truncated results and interval results are listed for comparison).

	Forward Sum
Noninterval (single precision truncated)	39.9883040991076
Noninterval (single precision rounded)	40.0000000000000
Interval (BPA)	(39.9921976607195,39.9921976614171)
Interval (BPF)	(39.9921976607195,40.0000000000000)

	Backward Sum
Noninterval (single precision truncated)	39.9883040991078
Noninterval (single precision rounded)	40.0000000000000
Interval (BPA)	(39.9921976607195,39.9921976614162)
Interval (BPF)	(39.9921976607195,40.0000000000000)

We note that the noninterval single precision rounded and truncated results (for both backward and forward sums) are not in the BPA intervals for their respective sums (in contrast to WES case for $1./2.^{27}$) although the rounded single precision results are contained in the BPF intervals for sums. The single precision rounded terms, unlike their single precision truncated counterparts, are all exactly equal, namely 1.000000000000000; thus for the rounded single precision case all the differences are erroneous (rounded to 1) and thus the rounded sum of 40 is also inaccurate.

12. I strongly suspect for this test with $1./2.^{48}$ on our CDC machine, that the best result is the BPA interval associated with the backward sum. The overall results for this test are probably significantly worse than the results for the WES case with $1./2.^{27}$, because in our case with $1./2.^{48}$ we are pressing the roundoff limits of our machine for the powers of the difference between $1-1/2^{48}$ (i.e. $1-1./2^{48}$) and its powers are much more inaccurate that ($1-1./2^{27}$) and its powers, especially since our machine has a 48-bit binary mantissa).

14. Problem: To subtract $1./2.^{27}$ from 1.0, repeatedly square the difference, and sum the squares. Forward and backward sums are computed for single precision, double precision, and interval (BPA and BPF) cases. This problem was defined by WES.

Code used and references:

A FØRTRAN routine provided by WES and modified to include double precision case.

Conditions under which test was conducted:

The FTN 4.6 FØRTRAN compiler was used with no optimization (i.e. opt = 0). Noninterval (single and double precision) and interval (both BPA and BPF) cases were run. The WRITE routine was used with variables declared interval. The standard (default) option of chopped arithmetic was used. The program was then run again using the CDC round option. See Observation II for comparison of rounded and truncated results.

Computational results:

	Forward Sum
Noninterval (single precision)	25.6672525468367
Noninterval (double precision)	25.667252621719278006774071 84
Interval (BPA)	(25.6672525468367,25.6672526755431)
Interval (BPF)	(25.6672525468367,25.6672531915557)

	Backward Sum
Noninterval (single precision)	25.6672525468373
Noninterval (double precision)	25.667252621719278006774071 84
Interval (BPA)	(25.6672525468373,25.6672526755418)
Interval (BPF)	(25.6672525468373,25.6672541915555)

		Terms of Sequence
1	Noninterval (single precision)	.999999985098839
	Noninterval (double precision)	.999999985098838806152343750 0
	Interval (BPA)	(.999999985098839,.000000095098839)
	Interval (BPF)	(.999999985098839,.999999985098839)
2	Noninterval (single precision)	.999999970197678
	Noninterval (double precision)	.999999970197677834349292425 0
	Interval (BPA)	(.999999970197678,.999999970197678)
	Interval (BPF)	(.999999970197678,.999999970197685)

Computational results: (cont.)

		Terms of Sequence
3	Noninterval (single precision)	.999999940395355
	Noninterval (double precision)	.999999940395356556887699153
	Interval (BPA)	(.999999940395355,.999999940395359)
	Interval (BPF)	(.999999940395355,.999999940395377)
4	Noninterval (single precision)	.999999880790714
	Noninterval (double precision)	.999999880790716666467502612 4
	Interval (BPA)	(.999999880790714,.999999880790721)
	Interval (BPF)	(.999999880790714,.999999880790760)
5	Noninterval (single precision)	.999999761581439
	Noninterval (double precision)	.999999761581447543788238 1192
	Interval (BPA)	(.999999761581439,.999999761581456)
	Interval (BPF)	(.999999761581439,.999999761581538)
6	Noninterval (single precision)	.999999523162931
	Noninterval (double precision)	.999999523162951930982631553 8
	Interval (BPA)	(.999999523162931,.999999523162970)
	Interval (BPF)	(.999999523162931,.999999523163137)
7	Noninterval (single precision)	.999999046326085
	Noninterval (double precision)	.999999046326131235535674282 0
	Interval (BPA)	(.999999046326085,.999999046326167)
	Interval (BPF)	(.999999046326085,.999999046326504)
8	Noninterval (single precision)	.999998092653076
	Noninterval (double precision)	.999998092653171964919312744 7
	Interval (BPA)	(.999998092653076,.999998092653243)
	Interval (BPF)	(.999998092653076,.999998092653243)
9	Noninterval (single precision)	.999996185309767
	Noninterval (double precision)	.999996185309981901761040973 0
	Interval (BPA)	(.999996185309787,.999996185310124)
	Interval (BPF)	(.999996185309787,.999996185310124)
10	Noninterval (single precision)	.999992370634121
	Noninterval (double precision)	.999992370634515663456260288 6
	Interval (BPA)	(.999992370634121,.999992370634800)
	Interval (BPF)	(.999992370634121,.999992370637525)

Computational Results: (cont.)

		Terms of Sequence
11	Noninterval (single precision)	.999984741326447
	Noninterval (double precision)	.999984741327238544606106 3617
	Interval (BPA)	(.999984741326447,.999984741327808)
	Interval (BPF)	(.999984741326447,.999984741333261)
12	Noninterval (single precision)	.99996948288572k
	Noninterval (double precision)	.999969482887304183653393 4933
	Interval (BPA)	(.999969482885721,.999969482888442)
	Interval (BPF)	(.999969482885721,.999969482899353)
13	Noninterval (single precision)	.999938966702736
	Noninterval (double precision)	.999938966705902534595942 2444
	Interval (BPA)	(.999938966702736,.999938966708179)
	Interval (BPF)	(.999938966702736,.999938966730003)
14	Noninterval (single precision)	.999877937130535
	Noninterval (double precision)	.999877937136868057579589 8196
	Interval (BPA)	(.999877937130535,.999877937141420)
	Interval (BPF)	(.999877937130535,.999877937185072)
15	Noninterval (single precision)	.999755889160411
	Noninterval (double precision)	.999755889173078671126487 8294
	Interval (BPA)	(.999755889160411,.999755889182183)
	Interval (BPF)	(.999755889160411,.999755889182183)
16	Noninterval (single precision)	.999511837910923
	Noninterval (double precision)	.999511837936253162467957 1691
	Interval (BPA)	(.999511837910923,.999511837954458)
	Interval (BPF)	(.999511837910923,.999511838129013)
17	Noninterval (single precision)	.999023914124070
	Noninterval (double precision)	.999023914174706806507384 6316
	Interval (BPA)	(.999023914124070,.999023914211101)
	Interval (BPF)	(.999023914124070,.999023914560045)
18	Noninterval (single precision)	.998048780991777
	Noninterval (double precision)	.998048781092951951309418 5973
	Interval (BPA)	(.998048780991777,.998048781862881)
	Interval (BPF)	(.998048780991777,.998048781862881)

Computational results: (cont.)

		Terms of Sequence
19	Noninterval (single precision)	.996101369239170
	Noninterval (double precision)	.996101369441127124400513234 9
	Interval (BPA)	(.996101369239170,.996101369586285)
	Interval (BPF)	(.996101369239170,.996101370977986)
20	Noninterval (single precision)	.992217937800149
	Noninterval (double precision)	.992217938202488826231362 2151
	Interval (BPA)	(.992217937800149,.992217938491674)
	Interval (BPF)	(.992217937800149,.992217938491674)
21	Noninterval (single precision)	.984496436092378
	Noninterval (double precision)	.984496436890797935303646 7808
	Interval (BPA)	(.984496436092378,.984496437464667)
	Interval (BPF)	(.984496436092378,.984496442966631)
22	Noninterval (single precision)	.969233232678594
	Noninterval (double precision)	.969233234250676881798718 6284
	Interval (BPA)	(.969233232678594,.969233235380624)
	Interval (BPF)	(.969233232678594,.969233246213953)
23	Noninterval (single precision)	.939414059328594
	Noninterval (double precision)	.939413062376027485732454 4856
	Interval (BPA)	(.939413059328594,.939413064566395)
	Interval (BPF)	(.939413059328594,.939413085566443)
24	Noninterval (single precision)	.882496896037107
	Noninterval (double precision)	.882496901762706107677569 4322
	Interval (BPA)	(.882496896037107,.882496905878028)
	Interval (BPF)	(.882496896037107,.882496945333472)
25	Noninterval (single precision)	.778800771515126
	Noninterval (double precision)	.778800718620775354380175 2689
	Interval (BPA)	(.778800771515216,.778800788884293)
	Interval (BPF)	(.778800771515126,.778800858522914)
26	Noninterval (single precision)	.606530641712553
	Noninterval (double precision)	.606530657453130623019026 5813
	Interval (BPA)	(.606530641712553,.606530668766798)
	Interval (BPF)	(.606530641712553,.606530777236031)

Computational results: (cont.)

		Terms of Sequence
27	Noninterval (single precision)	.367879419336241
	Noninterval (double precision)	.367879438430526871878426 3932
	Interval (BPA)	(.367879419336241,.367879452154700)
	Interval (BPF)	(.367879419336241,.367879583734547)
28	Noninterval (single precision)	.135335276171169
	Noninterval (double precision)	.135335281219959816161701 3234
	Interval (BPA)	(.135335267171169,.135335291317642)
	Interval (BPF)	(.135335267171169,.135335388128705)
29	Noninterval (single precision)	.0183156345
	Noninterval (double precision)	.018315638342885608106286 70080
	Interval (BPA)	(.0183156345402917,.0183156410760311)
	Interval (BPF)	(.0183156345402917,.0183156672799473)
30	Noninterval (single precision)	.000335462468613526
	Noninterval (double precision)	.000335462607907381464539 7657185
	Interval (BPA)	(.000335462468613526,.000335462708025999)
	Interval (BPF)	(.000335462468613526,.000335463667909734)
31	Noninterval (single precision)	$.112535067848281 \times 10^{-6}$
	Noninterval (double precision)	$.112535161304021553099316 8484 \times 10^{-6}$
	Interval (BPA)	$(.112535067848281 \times 10^{-6},.11253522847613 \times 10^{6})$
	Interval (BPF)	$(.112535067848281 \times 10^{-6},.112535872487453 \times 1^{-6}$
32	Noninterval (single precision)	$.126641414956172 \times 10^{-13}$
	Noninterval (double precision)	$.126641625297221499434324 4580 \times 10^{-13}$
	Interval (BPA)	$(.126641414956172 \times 10^{-13},.126641776481763 \times 10^{-1})$
	Interval (BPF)	$(.126641414956172 \times 10^{-13},.126643225965123 \times 10^{-13})$
33	Noninterval (single precision)	$.160380479821014 \times 10^{-27}$
	Noninterval (double precision)	$.160381012579218524349394 7999 \times 10^{-27}$
	Interval (BPA)	$(.160380479821014 \times 10^{-27},.160381395504569 \times 10^{-2})$
	Interval (BPF)	$(.160380479821014 \times 10^{-27},.160385066828534 \times 10^{-27})$
34	Noninterval (single precision)	$.257218983076186 \times 10^{-55}$
	Noninterval (double precision)	$.257220691959534505449002 5814 \times 10^{-55}$
	Interval (BPA)	$(1.257218983076186 \times 10^{-55},.257221920239931 \times 10^{-5})$
	Interval (BPF)	$(.257218983076186 \times 10^{-55},.257233696615933 \times 10^{-55})$

Computational results: (cont.)

		Terms of Sequence
35	Noninterval (single precision)	$.661616052547472 \times 10^{-55}$
	Noninterval (double precision)	$.661624843720491395273518 1012 \times 10^{-55}$
	Interval (BPA)	$(.257218983076186 \times 10^{-55}, .257221920239931 \times 10^{-55})$
	Interval (BPF)	$(.257218983076186 \times 10^{-55}, .257233696615933 \times 10^{-55})$
36	Noninterval (single precision)	$.437735800988497 \times 10^{-222}$
	Noninterval (double precision)	$.437747433828164662080492 5622 \times 10^{-222}$
	Interval (BPA)	$(.437735800988497 \times 10^{-222}, .437755795216480 \times 10^{-222})$
	Interval (BPF)	$(.437735800988497 \times 10^{-222}, .437835967713080 \times 10^{-222})$
37	Noninterval (single precision)	0.
	Noninterval (double precision)	0.
	Interval (BPA)	$(0., .156575653125701 \times 10^{-293})$
	Interval (BPF)	$(0., .156575653125701 \times 10^{-293})$
38	Noninterval (single precision)	0.
	Noninterval (double precision)	0.
	Interval (BPA)	$(0., .156575653125701 \times 10^{-293})$
	Interval (BPF)	$(0., .156575653125701 \times 10^{-293})$
39	Noninterval (single precision)	0.
	Noninterval (double precision)	0.
	Interval (BPA)	$(0., .156575653125701 \times 10^{-293})$
	Interval (BPF)	$(0., .156575653125701 \times 10^{-293})$
40	Noninterval (single precision)	0.
	Noninterval (double precision)	0.
	Interval (BPA)	$(0., .156575653125701 \times 10^{-293})$
	Interval (BPF)	$(0., .156575653125701 \times 10^{-293})$

Observations:

1. Backward and forward double precision sums (noninterval) agree exactly whereas backward and forward single precision sums (noninterval) agree to 13 decimal digits out of 15.

2. Both single precision sums (backward and forward) agree to 9 decimal digits with double precision result.

3. The lower bound for both interval versions of the sum agree exactly with the single precision versions for backward and forward sums.

4. The lower bound for both interval versions of each term in the sequence agrees with the single precision results for all 40 terms.

Observations: (cont.)

5. The BPF intervals for both sums and for each term are equal or larger than the corresponding BPA intervals.

6. The number of decimal digits (not including leading zeros) in agreement for upper and lower interval bounds on the terms of the sequence drifts downward as follows: 15 digits (terms 1 and 2), 13 digits (term 4, 12 digits (term 7), 12 digits (term 7), 11 digits (term 11), 10 digits (term 14), 9 digits (term 17), 8 digits (term 20), 7 digits (term 23), 6 digits (term 30), 5 digits (term 33), 4 digits (term 34), 0 digits (term 37).

7. As each term in the sequence is computed, the upper and lower interval bounds are monotonic decreasing, i.e., the sequence of lower interval bounds is steadily decreasing from terms 1 to 40 and the sequence of upper interval bounds is also steadily decreasing from terms 1 to 40. The resulting intervals for the terms of the sequence are drifting to the left approaching zero. Finally, the interval essentially collapse from term 34 onward.

8. The double precision resultant of the sum if rounded or chopped to single precision is contained in any of the computed intervals for the sum.

9. The number of decimal digits in agreement in the corresponding upper and lower interval bounds for the sum is 13. For each interval for the backward and forward sums, there is an 8 decimal digit agreement between upper and lower bounds.

10. According to the comments in this WES listing, the results for the subtractions on the Univac 1110 were correct using interval but not exact without interval due to truncation error in 1110 floating point. I wonder if interval result was better, not entirely because of truncation problem, but because interval implementation possibly used double precision on the 1110 which I believe has a wider exponent range than single precision and thus could keep track of smaller quantities, rather than set them equal to zero as in single precision floating-point arithmetic. On the 1110 version of this test were double precision non-interval arithmetic results for the sum and the terms compared with the interval results?

11. When the program was run with the CDC rounded option we get the results given in the following table (truncated results and interval results are listed for comparison):

	Forward Sum
Noninterval (single precision truncated)	25.6672525468367
Noninterval (single precision rounded)	25.6672525895078
Interval (BPA)	(25.6672525468367,25.6672526755431)
Interval (BPF)	(25.6672525468637,25.6672531915557)

Observations: (cont.)

	Backward Sum
Noninterval (single precision truncated)	25.6672525468373
Noninterval (single precision rounded)	25.6672523895078
Interval (BPA)	(25.6672525468373,25.6672526755418)
Interval (BPF)	(25.6672525468373,25.6672531915555)

We note that the noninterval single precision rounded and truncated results (for both backward and forward sums) are in both BPA and BPF intervals for their respective sums. The single precision rounded terms, although differing slightly from the corresponding truncated versions, exhibit the same monotonic decrease to zero that the unrounded terms indicated.

12. Although not provided with the exact results from the Univac 1110, I would strongly suspect that on our CDC machine the best result is the BPA interval associated with the backward sum.

15. Problem: To use Kahan's problem involving roundoff error propagation to check out consistency of compiler generated arithmetic rules versus execution time arithmetic rules and to determine the effects of the application of interval analysis to this problem.

Code used and references:

We used as a basis for our test the FØRTRAN code given by F. Dorr and C. Moler, SIGNUM Newsletter, Volume 8, Number 2, April 1973, pp. 24-26. This code is as follows:

```
Program 1                                   Program 2
                                            ØNE = 1.0
                                            TWØ = 2.0
                                            THREE = 3.0
                                            FIVE = 5.0
H = 1.0/2.0                                 H = ØNE/TWØ
X = 2.0/3.0-H                               X = TWØ/THREE-H
Y = 3.0/5.0-H                               Y = THREE/FIVE-H
                      E = (X+X+X)-H
                      F = (Y+Y+Y+Y+Y)-H
                      Q = 2.0*F/E
                      PRINT,Q
```

Note 1: The code in the middle belongs to both Programs 1 and 2. The correct values for E and F are both Zero, which means that Q should be indeterminate; however, on most (if not all) machines, the value for Q is finite! For example, results for Q on the CDC 6600 and CDC 7600 as reported by Dorr and Moler and on the CDC Cyber 70, model 72 as reported by Ginsberg are as follows:

	Program 1	Program 2
CDC 6600 (truncated arithmetic)	-6.0	+3.0
CDC 6600 (rounded arithmetic)	-6.0	-6.0
CDC 7600 (truncated arithmetic)	-6.0	+3.0
CDC 7600 (rounded arithmetic)	-6.0	+4.0
CDC Cyber 70, model 72 (truncated arithmetic)	-6.0	+3.0
CDC Cyber 70, model 72 (rounded arithmetic)	+4.0	-6.0

Note 2: The difficulty with this problem is that the evaluation of arithmetic subexpression can be in a different order and/or use different rounding procedures if done at translation time than the hardware would perform at execution.

Note 2: (cont.)

Furthermore, this inconsistency is compounded by the subtractive cancellation and roundoff error propagation introduced throughout the problem (especially in expressions for E and F).

Note 3: Ideally, we would like the application of interval analysis to this and similar problems to automatically warn the naive user that reported finite values of computed results (e.g. Q in this test) can be in serious error. We would like interval analysis to warn us in both Programs 1 and 2.

Conditions under which test was conducted:

The FIN 4.6 FØRTRAN compiler was used with no optimization (i.e. opt = 0). Noninterval and interval (both BPA and BPF) cases were run with RØUND arithmetic and truncated arithmetic. INTWR was used to output interval variables. Also for noninterval tests, both rounded and truncated arithmetic options were used under each compiler option (i.e. opt = 0,1,2,2,VO).

Observations:

1. The results are as follows on our CDC Cyber 72 with opt = 0.

Values of Q

	Program 1	Program 2
Noninterval(trun)	-6.0	+3.0
Noninterval(rdd.)	+4.0	-6.0
Interval(BPA)	(-6.0,-6.0)	$(-.1265014083\ 10^{323}, .1265014083\ 10^{323})$
Interval(BPF)	(-6.0,-6.0)	$(-.1265014083\ 10^{323}, .1265014083\ 10^{323})$

Program 1 is a test of compile time arithmetic behavior
Program 2 is a test of execution time arithmetic behavior

2. From the results given in Observation 1 above, we see that BPA and BPF interval results agree for each program; however, both interval versions were deceived in Program 1 because the interval routines use 2./3. and 3./5. as arguments which permits the compiler to do the arithmetic here, and unfortunately, the compiler does not evaluate these expressions and/or use the same rounding rules as are used at execution time. Thus, compile-time arithmetic deceived the interval analysis into indicating the true result for Q is -6, when in reality Q should be 0./0.

3. The intervals for Q in Program 2 were not deceived because no compile time arithmetic was involved in computing the intervals; such is the case when interval analysis is applied to arithmetic

Observations: (cont.)

expression where all values are hidden behind variable names and no explicit arithmetic which may be done in several different ways is present. Thus, for Program 2 the interval bounds for Q imply that the value of Q can be between -00 and +00. As a matter of fact, Program 2 is halted by interval package by warning of possible division by zero. Thus, in Program 2, involving no compile time arithmetic, the interval bounds properly warn the user that the result for Q is probably in serious error.

4. From this test and the comments in Observation 3, we see that when faced with a computational problem in which we intend to use interval analysis, we should re-write (or avoid) all explicit floating-point arithmetic expressions (make them implicit, i.e. hide all floating-point numbers behind variable names) which may be evaluated by the compiler in a different manner than the hardware uses at execution time. This precaution will decrease the user's chance of being deceived by the resulting interval bounds. It will not, however, guarantee that no deception will occur. Peculiarities in hardware behavior and attributes of interval analysis always allow room for the introduction of deceptive bounds.

5. The noninterval results reported in Observation 1 for Programs 1 and 2 are not consistent (for both rounded and truncated versions) because of inconsistencies between compiler time and execution time arithmetic which are further magnified by roundoff error propagation induced by subtractive cancellation and non-existence of exact floating point representation for some of the quantities involved (e.g. 3./5.).

6. The noninterval version of Programs 1 and 2 were run on our Cyber 72 under options 0,1,2 (using both rounded and chopped arithmetic) as well as under U.O. (unsave optimizations with rounded and chopped arithmetic). The results for E, F, and Q are given below for each case:

Noninterval Results on Cyber 72

		Program 1	Program 2	Program 3
OPT=0, Trun	E	.3552271367880050 X 10^{-14}	- .710542735760100 X 10^{-14}	- .710542735760100 X 10^{-14}
	F	-.106581410364015 X 10^{-13}	- .106581410364015 X 10^{-13}	- .106581410364015 X 10^{-13}
	Q	-6.00000000000000	3.00000000000000	3.00000000000000
OPT=0, Rdd	E	.355271367880050 X 10^{-14}	.355271367880050 X 10^{-14}	.355271367880050 X 10^{-14}
	F	.710542735760100 X 10^{-14}	- .106581410364015 X 10^{-13}	- .106581410364015 X 10^{-13}
	Q	4.00000000000000	-6.00000000000000	-6.00000000000000
OPT=1, Trun	E	.355271367880050 X 10^{-14}	- .710542735760100 X 10^{-14}	- .710542735760100 X 10^{-14}
	F	-.106581410364015 X 10^{-13}	- .106581410364015 X 10^{-13}	- .106581410364015 X 10^{-13}
	Q	-6.00000000000000	3.00000000000000	3.00000000000000

Noninterval Results on Cyber 72 (cont.)

		Program 1	Program 2	Program 3
OPT=1, Rdd	E	.355271367880050 10^{-14}	.355271367880050 10^{-14}	.355271367880050 10^{-14}
	F	.710542735760100 10^{-14}	- .106581410364015 10^{-13}	- .106581410364015 10^{-13}
	Q	4.00000000000000	-6.00000000000000	-6.00000000000000
OPT=2, Trun	E	.355271367880050 10^{-14}	- .710542735760100 10^{-14}	- .710542735760100 10^{-14}
	F	- .106581410364015 10^{-13}	- .106581410364015 10^{-13}	- .106581410364015 10^{-13}
	Q	-6.00000000000000	3.00000000000000	3.00000000000000
OPT=2, Rdd	E	.355271367880050 10^{-14}	.355271367880050 10^{-14}	.355271367880050 10^{-14}
	F	.710542735760100 10^{-14}	- .106581410364015 10^{-13}	- .106581410364015 10^{-13}
	Q	4.00000000000000	-6.00000000000000	-6.00000000000000
OPT=2, UO,Trun	E	.355271367880050 10^{-14}	- .710542735760100	.355271367880050 10^{-14}
	F	.710542735760100 10^{-14}	- .106581410364015 10^{-13}	- .106581410364015 10^{-13}
	Q	4.00000000000000	-6.00000000000000	-6.00000000000000
OPT=2, UO,Trun	E	.355271367880050 10^{-14}	- .710542735760100 10^{-14}	- .710542735760100 10^{-14}
	F	- .1065814010364015 10^{-13}	- .106581410364015 10^{-13}	- .106581410364015 10^{-13}
	Q	-6.00000000000000	3.00000000000000	3.00000000000000
OPT=2, UO,Rdd	E	.355271367880050 10^{-14}	.355271367880050 10^{-14}	.355271367880050 10^{-14}
	F	.710542735760100 10^{-14}	- .106581410364015 10^{-13}	- .106581410364015 10^{-13}
	Q	4.00000000000000	-6.00000000000000	-6.00000000000000

A Program 3 was added which is identical to Program 2, except the constants 1,2,3,5 are read in. The interval versions of this test were not run under round and truncated options (and under other opt levels) because they would have no numerical effect on the interval versions, since interval arithmetic is written in assembly language and the round or truncate option available with the CDC FTN FØRTRAN compiler only is applied to FØRTRAN explicit arithmetic expressions, not to assembly language code or library routines.

7. Observing the results given in Observation 6, we note that

   a) All truncated results are consistent for each program under all options and all rounded results are consistent for each program under all options, but all Program 1 and corresponding Program 2 results differ for all cases, and are not consistent for rounding and truncating under the same option.

7. (cont.)

   b) Corresponding results of Programs 2 and 3 are consistent under all options.

   c) The results for program 1 and corresponding Program 2 under truncated and rounded options is identical pattern for all opt = 0,1,2,2U0

   d) Although Programs 1,2,3 look equivalent, we see that we obtain values of -6,4 and 3 for Q when the exact result should be 0/0!!!

8. Although the CDC 6600,7600, and Cyber 72 all have the same floating-point arithmetic format, both Note 1 results above and Observation 6 results indicate differences in values of Q on the same machine as well as on different machines. (Particularly obvious is how the Cyber 72 result on Program 1 differs from Program 1 results on the other machines).

16. Problem: To test the performance of sin, cos, and tan routines with interval analysis as specified by WES.

Code used and references:

A FØRTRAN program provided by WES which output interval results via use of INTRAP routine in interval analysis package.

Conditions under which test was conducted:

The FTN 4.6 FØRTRAN compiler was used with no optimization (i.e. opt = 0). Both BPA and BPF interval versions were run. Interval variables were output via use of INTRAP routine in interval package.

Observations:

1. Both BPA and BPF versions gave identical results (for both decimal and corresponding octal values) for corresponding interval bounds.

2. All interval endpoints were checked for accuracy. All correct trig values and corresponding arguments are indicated by double arrows on the output provided in the attached listing.

3. As suggested by a comment in the WES test deck, we checked endpoints against adjacent computations. The arrows on our listing indicate the correspondences. There was no further specific indication of the pattern of correspondences. Presumably the pattern observed is correct. (We know the correspondences indicated between arguments and endpoints via double arrows are correct because these were verified.)

4. A few modifications were made in the WES test deck before the program was run. In the WES version of subroutine NTRAP, the arrays MØN, NAMES, TYPA, TYB, TYPR in CØMMON block INTRCM were each specified as a single array of dimension 32 which disagrees with the dimension of these arrays as specified by Yohe in his interval package. Thus, I changed the dimension of each of the aforementioned arrays to 40 to be in agreement with Yohe's package. Also, this test deck requires the use of subroutine INTUNN (for the UNION operator) in the Yohe package, but we were not provided with that routine in the version given to us. I have written a version of INTUNN (see attached listing) which satisifies Yohe's definition of the behavior of the UNION operator.

5. Since BPA and BPF interval versions produced the same results, I have attached only the BPA listing. It shows all the arguments and interval endpoints (in decimal and octal) produced by INTRAP and also provides a copy of the test program as run on our Cyber 72.

17. Problem: To use the non-interest and interval versions of the WES FFT program with sin input data provided by WES.

Code used and references:

The WES FFT program is written in FORTRAN and based on the paper by Glenn D. Bergland entitled "A Fast Fourier Transform Algorithm for Real-Valued Series" which appeared in the Communications of the ACM, Vol. 11, No. 10, October 1968, pp. 703-710.

Conditions under which test was conducted:

The FTN 4.6 FORTRAN compiler was used with no optimization (i.e. opt = 0). Non-interval and interval (both BPA and BPF) cases were run. Timing figures were obtained by using the SECØND function to measure the CPU time for each run excluding time for I/O. Inequalities involving interval variables used the interval operators SEQ, VGT, and VLT to replace the non-interval operators EQ, GT, AND LT, respectively. The INTWR routine was used to optput values of interval variables. PARAMETER statements were replaced by arithmetic statements. The modifications to the WES FFT code and driver program given by Bill Boyt were used with the exception of two lines in the modification of original line no. 1160 in the FFT code; these two lines were incorrect as stated and were corrected by M. Ginsberg (see Observation 1). Also the right-hand side of the expression for Y(I) in the FFT driver program had to be altered by taking its absolute value (see Observation 4)

Note: The input test data was provided by a WES FFT driver program which used 4096 elements of an array AT where the Ith element of the array is represented by

```
AT(I) = SIN (ØMEGA * X(I))
```

for

```
ØMEGA = ATAN2(1.,0.) * 800.
 X(I) = FLØAT(I-1) * DT
   DT = 1. (FLØAT(N) * DF)
   DF = .5
    N = 4096
```

The exact result from the FFT routine should be found in Array Y and should contain all zeros except for a value of + 1/2 corresponding to X(200) where in the FFT driver program

```
X(I) = FLØAT(I-1) * DF
  DF = .5
```

Computational results:

(A) Timings for the FFT Test

Computation results: (cont.)

Noninterval version ≈2.8 seconds of CPU time

BPA interval version ≈251 seconds of CPU time

BPF interval version ≈214 seconds of CPU time

(B) Computer values of Y(I) corresponding to X(100)

Noninterval

.49999 99999 99988

BPA interval

(.49999 99999 97866, .50000 00000 02125)

midpoint of interval = .49999 99999 99996

length of interval = $.268585154117318 \times 10^{-10}$

Note: All the values of Y(I) for I = 1 to 2049 are given for noninterval version, all values of Y(I) other than the one stated in B above are on the order of 10^{-12}, 10^{-13}, or 10^{-14}. For the BPA version, all Y values other than the one in B are on the order of 10^{-11} and for BPF version on the order of 10^{-10}.

Observations:

1. Two of the statements in the WES modification for line 1160 in the FFT code are incorrect as given, namely

WP(L2) = X

and

WP(L1) = Y

In the section of code immediately preceding line 1160 W is being computed to the necessary powers and Euler's formula is being used. o/e/. $e^{iz} = \cos z + i \sin z$. The mistake made in the WES modification was to put the real part of this relationship in the imaginary location and the imaginary part in the real location in the array WP. Thus the correct modification should be

WP(L1) = X

WP(L2) = Y

If these changes are not made, the code produces several nonzero values for Y(I) and not at the expected place (i.e., corresponding to X(200))

2. From the computational results, we see that the noninterval and interval versions produced values close to the exact result of .5. The noninterval version is slightly less than the exact value of .5 and both interval versions included the solution (as well as the noninterval resultant). As in our other tests with the interval package, the BPF version produces a slightly wider interval. We also note that the midpoint of the BPA results is less than the exact value but the BPF midpoint is slightly more than the exact value. For both the BPA and BPF

Observations: (cont.)

interval versions, the intervals for the other values of Y were small nonnegative neighborhoods of zero; all of these intervals for BPA and BPF versions were bounded by exactly zero on the left and the lengths of these intervals were on the order of 10^{-11} and 10^{-10} for BPA and BPF versions, respectively.

3. Examining the timing results for this test we note that

$$\frac{\text{BPF interval version CPU time}}{\text{BPA interval version CPU time}} \approx .85$$

$$\frac{\text{BPA interval version CPU time}}{\text{Noninterval version CPU time}} \approx 89.6$$

$$\frac{\text{BPF interval version CPU time}}{\text{Noninterval version CPU time}} \approx 76.4$$

Thus we see that both interval versions require substantially more CPU time than the noninterval version and that the BPF interval version requires only 85% of the CPU time needed by the BPA interval version and provides an interval result only slightly larger than the BPA interval.

4. Part of the WES modification for line 310 in the FFT driver program included

```
Y(I) = SQRT(AFR * AFR + AFI * AFI)
```

When the interval versions of the FFT test were run with this expression for Y(I), the programs halted with a diagnostic indicating that the program was attempting to calculate the square root of a negative quantity. To understand why this happened, recall that the interval definition for the multiplication of two intervals is

$$[a,b] * [c,d] = [\min(a*c,\ a*d,\ b*c,\ b*d),\ \max(a*c,\ a*d,\ b*c,\ b*d)]$$

So when both intervals are the same, i.e., $[a,b] = [c,d]$ then

$$[a,b] * [a,b] = [\min(a^2,\ a*b,\ b^2),\ \max(a^2,\ a*b,\ b^2)]$$

It turns out that for most of the Y(I) cases, the left endpoints of the intervals for AFR and AFI are negative (relatively small on the order of 10^{-11}) and the right endpoints are positive and so that multiplying such intervals by themselves does not produce positive intervals, i.e., if $a < 0$ and $b > 0$ then

$$[a,b,] * [a,b] = [a*b] = [a*b,\ \max(a^2, b^2)]$$

which would contain negative numbers since $a*b < 0$ and thus would make the interval square root routine halt because it

Observations: (cont.)

recognizes that the interval is not entirely nonnegative. To correct this situation we rewrite Y(I) as

```
Y(I) = SQRT [ABS(AFR * AFR + AFI + AFI)]
```

which results in the interval version of the square root routine working with an entirely nonnegative interval. This situation is a direct result of the rules for interval arithmetic, i.e., an interval multiplied by itself does not necessarily produce a nonnegative resultant interval even though for noninterval arithmetic on real numbers, the product of any real number with itself is nonnegative.

5. In the WES Input for the FFT, 4096 values of the SIN routine were used with arguments between 0 and $\approx 800\pi$. Since increasingly larger angles were used and errors in the SIN routine tend to get bigger as the angles increase beyond 2π, further examination of the SIN results was conducted. The values of the WES SIN inputs were compared with the double precision SIN of the double precision expansion of the original single precision arguments and with the resultants chopped to single precision for comparison with the WES SIN Input. In addition, the WES SIN argument expressions were calculated in double precision and then used with double precision SIN with the results chopped to single precision for comparison with the WES SIN values. For both sets of comparisons with the WES SIN values, it was noted that there was a slight gradual loss of accuracy in the SIN values as the angles increased in size; there was never more than a 4 or 5 decimal digit loss, i.e., error never larger than on the order of 10^{-10} for the abovementioned comparisons. Of course with such comparisons, we are tacitly assuming that the CDC FTN version of DSIN is more accurate for larger angles (when the double precision resultant is chopped to single precision) than the SIN routine is. An examination of the intervals associated with the elements in the array AT which contains the resultants of the WES SIN inputs also reflects this loss of digits, i.e., the lengths of the intervals associated with the elements in array AT increase slightly as the SIN angles get larger; for the BPA version the interval lengths are initially 0 or on the order of 10^{-13} and decrease to the order of 10^{-11} or 10^{-10} as the arguments of the SIN approach 800π and for the BPF version the intervals are initially on the order of 10^{-13} and decrease to the order of 10^{-10} as the angle approaches 800π.

6. More and varied cases would have to be run beyond the WES test before we could make more conclusive assessments of the behavior of the interval versions of the WES FFT program with respect to exact results and/or noninterval runs. Also it would be informative to perform interval and noninterval tests on other high quality FFT routines such as that provided by the IMSL Library or Subroutine FOURT by Norman Brenner (described in "Three Fortran Programs that Perform the Cooley-Tukey Fourier Transform," Technical Note 1967-2, MIT Lincoln Laboratory, Lexington, Massachusetts, July 28, 1967 (also available from NTIS under AD 657 019)).

REFERENCES

[1] Crary, F. D., "The AUGMENT Precompiler; I: User Information," Technical Summary Report No. 1469, Mathematics Research Center, University of Wisconsin-Madison, Dec 1974 (revised Apr 1976).

[2] Crary, F. D., "The AUGMENT Precompiler; II: Technical Documentation," Technical Summary Report No. 1470, Mathematics Research Center, University of Wisconsin-Madison, Oct 1975.

[3] Yohe, J. M., "Best Possible Floating Point Arithmetic," Technical Summary Report No. 1054, Mathematics Research Center, University of Wisconsin-Madison, Mar 1970.

[4] Yohe, J. M., "Roundings in Floating-Point Arithmetic," IEEE Trans. Computers, Vol C-22, No. 6, Jun 1973, pp. 577-586.

[5] Yohe, J. M., "The Interval Arithmetic Package," Technical Summary Report No. 1755, Mathematics Research Center, University of Wisconsin-Madison, Jun 1977.

[6] Sterbenz, P. H., Floating-Point Computation, Prentice-Hall, Englewood Cliffs, N. J., 1974.

NOTE: Additional references are cited in Sections 7-17 of this report as they pertain to the examples.

The following are additional computer listings and runs available from the Automatic Data Processing Center, U. S. Army Engineer Waterways Experiment Station, P. O. Box 631, Vicksburg, Miss. 39180:

SMU primitives for Interval
SMU primitives for AUGMENT
Source of Interval
Source of AUGMENT
CAUSE (real and interval)
FFT (real and interval)
Mathematical Expression (real and interval)
Trigonometric functions (real and interval)
SPLINE (real and interval)
Kahan's problem (real and interval)

APPENDIX A: A GUIDE TO THE LITERATURE ON INTERVAL ANALYSIS

I. THEORETICAL CONSIDERATIONS

1. Chuba, W. and W. Miller, "Quadratic Convergence in Interval Arithmetic, Part I," BIT, Vol. 12, 1972, pp. 284-290.
2. Good, D. I. and R. L. London, "Computer Interval Arithmetic: Definition and Proof of Correct Implementation," J. Assoc. Comput. Mach., Vol. 17, No. 4, October 1970, pp. 603-612.
3. Hanson, R. J., Interval Arithmetic as a Closed Arithmetic System on a Computer, Technical Memorandum No. 197, Section 314, Jet Propulsion Laboratory, Pasadena, California, 1968.
4. Miller, W., "The Error in Interval Arithmetic," Interval Mathematics, edited by K. Nickel, Vol. 29, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1975, pp. 246-250.
5. Miller, W., "Quadratic Convergence in Interval Arithmetic, Part II," BIT, Vol. 12, 1972, pp. 291-298.
6. Miller, W., "More on Quadratic Convergence in Interval Arithmetic," BIT, Vol. 13, 1973, pp. 76-83.
7. Moore, R. E., "The Automatic Analysis and Control of Error in Digital Computation Based on the Use of Interval Numbers," Error in Digital Computation, Vol. 1, edited by L. B. Rall, J. Wiley and Sons, New York, 1965, pp. 61-130.
8. Moore, R. E., "Automatic Local Coordinate Transformations to Reduce the Growth of Error Bounds in Interval Computation of Solutions of Ordinary Differential Equations," Error in Digital Computation, Vol. 2, edited by L. B. Rall, J. Wiley and Sons, New York, 1965, pp. 103-140.
9. Moore, R. E., Interval Analysis, Prentice-Hall, Englewood Cliffs, New Jersey, 1966.
10. Nickel, K., The Contraction Mapping Fixed Point Theorem in Interval Analysis, Technical Summary Report No. 1334, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, 1973.
11. Nickel, K., "Error Bounds and Computer Arithmetic," Proc. Information Processing 68, Vol. 1, North-Holland, Amsterdam, The Netherlands, 1969, pp. 54-60; "Invited Commentary," by W. Kahan, Ibid., pp. 60-62.
12. Nickel, K. (ed.), Interval Mathematics, Vol. 29, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1975.
13. Nickel, K., Stability and Convergence of Monotonic Algorithms, Technical Summary Report No. 1340, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, 1973.
14. Ratschek, H., "Some Basic Concepts of Interval Arithmetic," Computing, Vol. 4, No. 1, 1969, pp. 43-55.

II. IMPLEMENTATION CONSIDERATIONS

1. Braun, J. and R. E. Moore, A Program for the Solution of Differential Equations Using Interval Arithmetic (DIFEQ), Technical Summary Report No. 901, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, June 1968.
2. Kahan, W. M., "Interval Arithmetic" in Implementation of Algorithms, Chapter 12, Technical Report No. 20, Department of Computer Science, University of California, Berkeley, 1973; available as AD 769 124 from National Technical Information Service, Springfield, Virginia.

3. Kahan, W. M., "A More Complete Interval Arithmetic," Lecture Notes, University of Michigan summer course, Ann Arbor, Michigan, June 1968.
4. Ris, F. N., "Tools for the Analysis of Interval Arithmetic," Interval Mathematics, edited by K. Nickel, Vol. 29, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1975, pp. 75-98.
5. Smith, R. and E. R. Hansen, A Computer Program for Solving a System of Linear Equations and Matrix Inversion with Automatic Error Bounding Using Interval Arithmetic, Report LMSC4-22-66-3, Lockheed Missiles and Space Co., Palo Alto, California, 1968.

III. APPLICATIONS TO NUMERICAL ANALYSIS PROBLEM AREAS

1. Aird, T. J. and R. E. Lynch, "Accurate Upper and Lower Error Bounds for Approximate Solutions of Linear Algebraic Systems," ACM Trans. Math. Software, Vol. 1, No. 3, September 1975, pp. 217-231.
2. Arthur, D. W., "The Use of Interval Arithmetic to Bound the Zeros of Real Polynomials," J. Inst. Math. Appl., Vol. 10, No. 2, October 1972, pp. 231-237.
3. Barth, W., "Computing of Roots with Interval Analysis," Computing, Vol. 8, No. 3-4, 1971, pp. 320-328.
4. Berti, S. N., "The Solution of an Interval Equation," Mathematica, Vol. 11, No. 34, 1969, pp. 189-194.
5. Berti, S. N., "Some Relations between Interval Functions (I)," Mathematica, Vol. 14, No. 37, 1972, pp. 9-26.
6. Chartres, B. A., "Automatic Controlled Precision Calculations," J. Assoc. Comput. Mach., Vol. 13, No. 3, July 1966, pp. 386-403.
7. Chartres, B. A. and J. C. Geuder, "Computable Error Bounds for Direct Solution of Linear Equations," J. ACM, Vol. 14, No. 1, January 1967, pp. 63-71.
8. Cryer, C. W., On the Computation of Rigorous Error Bounds for the Solution of Linear Equations with the Aid of Interval Arithmetic, Technical Report No. 70, Computer Science Dept., University of Wisconsin, Madison, Wisconsin, 1969.
9. Dargel, R. H., F. R. Loscalzo, and T. H. Witt, "Automatic Error Bounds on Real Zeros of Rational Functions," Comm. ACM, Vol. 9, 1966, pp. 806-809.
10. Grant, J. A. and G. D. Hitchins, "The Solution of Polynomial Equations in Interval Arithmetic," Comput. J., Vol. 16, 1973, pp. 69-72.
11. Hansen, E. R., "On the Solution of Linear Algebraic Equations with Interval Coefficients," Linear Algebra and Appl., Vol. 2, 1969, pp. 153-165.
12. Hansen, E. R., "On Solving Systems of Equations Using Interval Arithmetic," Math. Comp., Vol. 22, 1968, pp. 374-384.
13. Hansen, E. R., "On Solving Two-Point Boundary-Value Problems Using Interval Arithmetic," Topics in Interval Analysis, edited by E. R. Hansen, Clarendon Press, Oxford, 1969, pp. 74-90.
14. Hansen, E. R. (ed.), Topics in Interval Analysis, Oxford University Press, London, 1969.
15. Hansen, E. R., "Interval Arithmetic in Matrix Computations, Part 1," SIAM J. Numer. Anal., Vol. 2, No. 2, 1965, pp. 308-320.
16. Hansen, E. R. and R. Smith, "Interval Arithmetic in Matrix Computations, Part 2," SIAM J. Numer. Anal., Vol. 4, No. 1, March 1967, pp. 1-9.
17. Hanson, R. J., "Automatic Error Bounds for Real Roots of Polynomials Having Interval Coefficients," Comput. J., Vol. 13, 1970, pp. 284-288.
18. Krawczyk, R., "Enclosing Zeros with an Interval Arithmetic," Computing, Vol. 5, No. 4, 1970, pp. 356-370.

19. Krückeberg, F., "Ordinary Differential Equations," chapter 8 in Topics in Interval Analysis, edited by E. R. Hansen, Oxford University Press, London, 1969, pp. 91-97.
20. Krückeberg, F., "Partial Differential Equations," chapter 9 in Topics in Interval Analysis, edited by E. R. Hansen, Oxford University Press, London, 1969, pp. 98-101.
21. Madsen, K., "On the Solution of Nonlinear Equations in Interval Arithmetic," BIT, Vol. 13, 1973, pp. 428-433.
22. Miller, W., "On an Interval-Arithmetic Matrix Method," BIT, Vol. 12, 1972, pp. 213-219.
23. Moore, R. E., "Practical Aspects of Interval Computation," Aplikace Matematiky, Vol. 13, No. 1, 1968, pp. 52-92.
24. Moore, R. E., "Two-Sided Approximation to Solutions of Nonlinear Operator Equations - A Comparison of Methods from Classical Analysis, Functional Analysis, and Interval Analysis," Interval Mathematics, edited by K. Nickel, Vol. 29, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1975, pp. 31-47.
25. Nickel, K., The Application of Interval Analysis to the Numerical Solution of Differential Equations, Report No. 69/9, Institute for Information, Karlsruhe University, Karlsruhe, West Germany, 1969.
26. Nickel, K., On the Newton Method in Interval Analysis, Technical Summary Report No. 1136, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, December 1971.
27. Nickel, K., "Zeros of Polynomials and Other Topics," Topics in Interval Analysis, edited by E. R. Hansen, Oxford University Press, London, 1969, pp. 25-34.
28. Oliveira, F. A., "Interval-Extension of Quasilinearization Method," Interval Mathematics, edited by K. Nickel, Vol. 29, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1975, pp. 270-278.
29. Oliveira, F. A., "Interval Analysis and Two-Point Boundary Value Problems," SIAM J. Numer. Anal., Vol. 11, No. 2, April 1974, pp. 382-391.
30. Ris, F. N., Interval Analysis and Applications to Linear Algebra, Ph.D. thesis, Oxford University, Oxford, England, 1972.
31. Rokne, J., "Explicit Calculation of the Lagrangian Interval Interpolating Polynomial," Computing, Vol. 9, 1972, pp. 149-157.
32. Rokne, J., Interval Arithmetic in the Solution of Equations, Research paper 217, Yellow Series, Department of Mathematics, Statistics, and Computing Science, University of Calgary, Alberta, Canada, 1974.
33. Rokne, J. and P. Lancaster, "Complex Interval Arithmetic," Comm. ACM, Vol. 14, No. 2, February 1971, pp. 111-112.
34. Schroeder, G., "Differentiation of Interval Functions," Proc. Amer. Math. Soc., Vol. 36, 1972, pp. 485-490.
35. Skelboe, S., "Computation of Rational Interval Functions," BIT, Vol. 14, 1974, pp. 87-95.
36. Smith, L. B., "Interval Arithmetic Determinant Evaluation and Its Use in Testing for a Chebyshev System," Comm. ACM, Vol. 12, No. 2, February 1969, pp. 89-93.
37. Stewart, N. F., "Interval Arithmetic for Guaranteed Bounds in Linear Programming," J. Optimization Theory Appl., Vol. 12, No. 1, July 1973, pp. 1-5.
38. Talbot, T. D., Guaranteed Error-Bounds for Computed Solutions of Nonlinear Two-Point Boundary Value Problems, Technical Summary Report No. 875, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, June 1968.

39. Wiggins, R. A., "Evaluation of Computational Algorithms for Associated Legendre Polynomials by Interval Analysis," Bull. Seismological Society of America, Vol. 61, No. 2, 1971, pp. 375-381.
40. Yohe, J. M., "Interval Bounds for Square Roots and Cube Roots," Computing, Vol. 11, No. 1, 1973, pp. 51-57.

IV. ALTERNATIVES TO CLASSICAL INTERVAL ARITHMETIC

1. Ames, W. F. and M. Ginsberg, "Bilateral Algorithms and Their Applications," Computational Mechanics, edited by J. T. Oden, Vol. 461, Lecture Notes in Mathematics, Springer-Verlag, Berlin, 1975, pp. 1-31.
2. Ashenhurst, R. L., "Experimental Investigation of Unnormalized Arithmetic," Error in Digital Computation, Vol. 2, edited by L. B. Rall, J. Wiley and Sons, New York, 1965, pp. 3-37.
3. Ashenhurst, R. L., "Number Representation and Significance Monitoring," Mathematical Software, edited by J. R. Rice, Academic Press, New York, 1971, pp. 67-92.
4. Breiter, M. C., C. L. Keller, and T. E. Reeves, A Program for Computing Error Bounds for the Solution of a System of Differential Equations, Report No. 69-0054, Aerospace Research Laboratories, Wright-Patterson Air Force Base, Ohio, March 1969.
5. Brent, R. P., A FORTRAN Multiple-Precision Arithmetic Package, report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, May 1976.
6. Ginsberg, M., Introduction to the Study of Algorithms for Computing Upper and Lower Bounds to the Exact Solution of Problems in Numerical Analysis, Technical Report No. CP 73024, Department of Computer Science, Southern Methodist University, Dallas, Texas, September 1973.
7. Guderley, K. G. and C. L. Keller, Ellipsoidal Bounds for the Solutions of Systems of Ordinary Linear Differential Equations, Report No. ARL 69-005, Aerospace Research Labs, Wright-Patterson Air Force Base, Ohio, January 1969.
8. Guderley, K. G. and M. Valentine, "On Error Bounds for the Solution of Systems of Ordinary Differential Equations," Blanch Anniversary Volume, Aerospace Research Labs, Wright-Patterson Air Force Base, Ohio, 1967, pp. 45-89.
9. Hansen, E. R., "A Generalized Interval Arithmetic," Interval Mathematics, edited by K. Nickel, Vol. 29, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1975, pp. 7-18.
10. Hull, T. E. and J. J. Hofbauer, Language Facilities for Multiple Precision Floating-Point Computation, with Examples and the Description of a Pre-processor, Technical Report No. 63, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, February 1974.
11. Jackson, L. W., Automatic Error Analysis for the Solution of Ordinary Differential Equations, Tech Report No. 28, Department of Computer Science, University of Toronto, Ontario, Canada, March 1971.
12. Jackson, L. W., "Interval Arithmetic Error-Bounding Algorithms," SIAM J. Numer. Anal., Vol. 12, No. 2, April 1975, pp. 223-238.
13. Kahan, W. M., "An Ellipsoidal Error Bound for Linear Systems of Ordinary Differential Equations," (Unpublished), University of Toronto, Toronto, Ontario, Canada, 1967.
14. Kahan, W. M., "A Computable Error Bound for Systems of Ordinary Differential Equations," (abstract), SIAM Rev., Vol. 8, No. 4, October 1966, pp. 568-569.
15. Rall, L. B. (ed.), Error in Digital Computation, Vols. 1 and 2, J. Wiley and Sons, New York, 1965.

16. Richman, P. L., Variable-Precision Interval Arithmetic, Technical Memo No. MM-69-1374-26, Bell Telephone Laboratories, Murray Hill, New Jersey, 1969.
17. Richman, P. L., Error Control and the Midpoint Phenomenon, Technical Memorandum No. MM-69-1374-29, Bell Telephone Laboratories, Murray Hill, New Jersey, 1969.
18. Tienari, M., Varying Length Floating-Point Arithmetic: A Necessary Tool for the Numerical Analyst, Technical Report No. 62, Computer Science Department, Stanford University, Stanford, California, 1967.
19. Wyatt, W. T., Jr., D. W. Lozier, and D. J. Orser, "A Portable Extended Precision Arithmetic Package and Library with Fortran Precompiler," ACM Trans. Math. Software, Vol. 2, No. 3, September 1976, pp. 209-231.

V. BIBLIOGRAPHIES ON INTERVAL ANALYSIS AND RELATED TOPICS

1. Bierbaum, F., Intervall-Mathematik. Eine Literaturübersicht, Report No. 74/2, Institut für Praktische Mathematik, University Karlsruhe, Karlsruhe, West Germany, 1974.
2. Bierbaum, F., Intervall - Mathematik Eine Literaturübersicht Nachtrag, Interval Report No. 75/3, Institut für Praktische Mathematik, Karlsruhe University, Karlsruhe, West Germany, 1975.
3. Bierbaum, F., Intervall-Mathematik. Eine Literaturübersicht, edition 2, Interval Report No. 76/4 Institut für Praktische Mathematik, University Karlsruhe, Karlsruhe, West Germany, July 1976.
4. Ginsberg, M., "A Guide to the Literature of Modern Numerical Mathematics," Bibliography 36, Computing Reviews, Vol. 16, No. 2, February 1975, pp. 83-97.

VI. THE MRC INTERVAL ARITHMETIC PACKAGE AND RELATED TOPICS

1. Crary, F. D., The AUGMENT Precompiler, I. User Information, Technical Summary No. 1469, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, December 1974, (Revised April 1976).
2. Crary, F. D., The AUGMENT PRECOMPILER, II. Technical Documentation, Technical Summary Report No. 1470, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, October 1975.
3. Crary, F. D., Language Extension and Precompilers, Technical Summary Report No. 1317, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, February 1973.
4. Ladner, T. D. and J. M. Yohe, An Interval Arithmetic Package for the Univac 1108, Technical Summary Report No. 1055, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, May 1970.
5. Rosser, J. B. and J. M. Yohe, Cancellation and Rounding Errors, Technical Summary Report No. 1588 Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, September 1976.
6. Yohe, J. M., Accurate Conversion between Number Bases, Technical Summary Report No. 1109, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, October 1970.
7. Yohe, J. M., Best Possible Floating-Point Arithmetic, Technical Summary Report No. 1054, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, March 1970.
8. Yohe, J. M., Implementing a Nonstandard Data Type, Technical Summary Report No. 1545, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, April 1975.
9. Yohe, J. M., Implementation of the Package Interval II on Other Hardware, memo, draft, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, June 1976.

10. Yohe, J. M., The Interval Arithmetic Package, Technical Summary Report No. 1755, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, June 1977.

11. Yohe, J. M., "Roundings in Floating-Point Arithmetic," IEEE Trans. Computers, Vol. C-22, No. 6, June 1973, pp. 577-586.

APPENDIX B: INTRODUCTION TO ERROR-BOUNDING TECHNIQUES AND INTERVAL ANALYSIS

In this appendix the reader is introduced to interval analysis as an error-bounding technique. We discuss the motivation for error-bounding methods and give some examples. We then move on to a discussion of engineering applications and the advantages of using such techniques. Finally, we present some interval analysis definitions, theorems, and elementary properties. Limitations (theoretical and practical) of interval analysis are described in Section of this report. The reader desiring additional information on interval analysis is urged to consult Moore (1966) and/or any of the references given in the bibliography of this appendix.

Most scientific and engineering problems require solutions of an extensive range of mathematical equations - algebraic, transcendental, integral, differential, etc. Quite often, an approximate or numerical solution is needed because an analytic or exact solution is not known or is computationally awkward to work with. Such approximate methods introduce their own inherent roundoff or truncation errors. Thus, the scientist or engineer who utilizes these techniques frequently would like to know the relationship between the approximate and exact solutions, especially when dealing with math models used in designing various physical objects such as airplanes, electronic equipment, or nuclear power plants.

Unfortunately, many numerical solutions give little or no indication as to whether they are above, below, or oscillate about the exact solution.

For example, Table 1 and Table 2 from Ralston (1965) show some output from approximation techniques such as Hamming's Method, Milne's Method, and the Hermite Method when applied to the following two initial value problems:

(a) $y' = -y \qquad y(0) = 1$

(b) $y' = \frac{1}{1+\tan^2 y} \qquad y(0) = 0.$

Scattered throughout these tables (underlined entries) are changes in sign between two adjacent entries in the error columns; these places represent positions at which the approximate solution oscillates about the exact solution. Figure 1 from Lapidus and Seinfeld (1971) reveals that the fifth order Nystrom approximation for problem (a) above is initially equal to the exact solution at x=0, goes below the exact solution for $0 < x < .5$ and then rises above the exact solution for $x > .5$. This type of alternating behavior is quite commonly exhibited by numerical solutions.

A more pronounced example of approximation fluctuation, with respect to an exact solution, is exhibited by Greenspan (1967). Three possible approximate solutions to an initial value problem involving the linearized Emden differential equation are $y^{(1)}$ and $y^{(2)}$ in Figure 2 and $y^{(3)}$ in Figure 3. The exact solution is shown by the solid line curve in Figures 2 and 3. It is apparent from these graphs that all three numerical solutions oscillate about the exact solution. The user of any of these approximate solutions would not have any automatic indication as to whether his numerical solution, for any given abscissa, is above or below the exact solution.

The above examples clearly demonstrate the changing positional relationships between approximate and exact solutions for even relatively simple initial value problems. At present, there are few ways to determine the

nature of this relationship. One naive approach is to construct a graph of both the approximate and exact solutions for certain test problems. In general, however, plotting is usually very ineffective; indeed, accurate plotting may be difficult to perform if the exact solution is very complicated. Frequently, the user employs a numerical method because the analytic solution is inaccessible, or he has little knowledge of its exact form, thus making plotting impossible.

Most approximation methods provide a theoretical error bound for the maximum distance the exact solution can deviate from the numerical solution. Unfortunately, most of these error bounds are defined for the absolute value of the deviation so that the user can still not be certain as to which side the exact solution resides with respect to the numerical solution. More importantly, these bounds are usually difficult to evaluate in a practical manner and often require the introduction of many simplifying assumptions. Even when they can be evaluated, these bounds are usually not optimal and/or do not take into account the floating-point roundoff error propogation (which can be substantial in some problems) encountered when an algorithm is implemented and executed on a computer.

From the above discussion, it should be apparent that in many situations the user of numerical methods ordinarily has no efficient, automatic, and practical indicators of the positional relationship between approximate and exact solutions. One way to establish an indicator is to develop numerical methods which provide upper and lower bounds to the exact solution; techniques which bound the exact solution from above and below are called "bilateral" (two-sided) methods. One such bounding approach is known as interval analysis; this technique produces upper and lower bounds to the

solution of large collections of scalar and matrix mathematical problems.

Before examining the properties of interval analysis, we shall look briefly at the engineering applications and other advantages offered by the use of bilateral methods in general. Even though bilateral algorithms have not as yet been widely available, several applications have already been described in the literature. For example, Baranov (1964) reports that nonlinear differential equations representing flexural and torsional vibration of beams or buckling of rods often cannot be easily solved either analytically or by existing numerical methods in such a way that the positional relationship between the approximate and exact solutions is accurately revealed. A bilateral approach is helpful here; Baranov has developed such a method for his specific equations. In naval warfare problems involving the use of Lanchester equations, Fabry (1970) has defined a two-sided bounding scheme which offers a means of studying variations of certain parameters before the equations are solved by some conventional (non-bounding) method. Boley (1963) has used a bilateral approach for problems of heat conduction in melting or solidifying slabs; his work can also provide approximations for certain types of aerodynamic ablation problems. Appl and Hung (1964) have applied a two-sided technique to continuous equilibrium problems such as a fin temperature problem in which there is internal heat generation. Ispolov and Appl (1971) have employed a bilateral method for a problem of self-sustained vibration of an autonomous system. Kahan (1972) has developed an ellipsoidal bounding technique which can be applied to the N-body problem. Two-sided bounding methods have also been employed by Weinstein and Stenger (1972) for problems involving vibrations of cantilever plates or for energy levels in quantum mechanics. In most of the above examples, no general techniques have been devised, just methods

to solve very specific problems.

In addition to the aforementioned engineering applications, there are several other benefits to be gained from the utilization of two-sided algorithms for the solution of problems in numerical analysis. For example, the bounds produced by a bilateral method can provide information about the region in which a solution resides; this may be sufficient for some applications (e.g. see Willson (1968)), including the solution of algebraic equations, ordinary, and partial differential equations, and eigenvalue problems. Acquired knowledge of the solution region via efficient bilateral techniques could be of assistance in a variation of parameter study; such was the motivation of Fabry's nested bound approach (1970). Also, the insight acquired about the solution space could aid in selecting the most appropriate conventional (nonbounding) numerical method to use on a set of problems having solutions in the same or neighboring regions. Furthermore, good starting values for these numerical methods could be produced by two-sided algorithms.

A bilateral approach can help in error estimation. The error of a numerical solution obtained by conventional techniques can be compared with the bounds obtained by a two-sided scheme, thus providing an explicit estimate of the error associated with the approximate solution. Also, the average of the upper and lower bounds produced by a bilateral method can serve as an improved approximation to the exact solution, in which case this new approximation is in error by no more than the absolute value of half the difference between the utilized upper and lower bounds (if it can be assumed that propagated computational error in calculating the bounds is relatively small). Thus, bilateral methods not only guarantee a priori the relative positions of their iterates with respect to the exact solution, but also

assist in providing an approximate solution of known accuracy. This aforementioned trait is quite significant, since most currently available methods cannot offer an explicit, easily computable, and precise estimate of the error (theoretical or actual) associated with their approximations.

In our above comments concerning engineering applications of error bounding methods, we gave a few examples of the assistance a bilateral approach offers when dealing with differential equations which arise from many physical problems; no doubt numerous other applications await discovery. An error bounding approach should be considered when the exact solution is quite difficult (or impossible) to obtain or easily represent, and when most standard numerical techniques do not clearly reveal the relative position of the approximate and exact solutions. This was indeed the situation, as reported by Baranov (1964) for some mechanics problems. In some areas of communications theory (see Strakhov and Kurz (1968)), the exact solution is known only for relatively simple cases; in complex cases, an upper or lower solution bound could be very helpful. Information about the solution region obtained from a bilateral approach may be beneficial in designing a piece of equipment (e.g. in defining a device's range of limitation). Efficient computer implementations of two-sided schemes can be used in a quantitative analysis of parameters and initial conditions; such an application may produce competitive or significantly better results than those obtained by conventional techniques. From the above discussion it should be readily apparent that the major advantage offered to engineers and scientists is a possible, practical way of automatically assessing the positional relationship between approximate and exact solutions. Thus, interval analysis is one means of providing such a tool.

Now we proceed to discuss some of the specific attributes of interval analysis. The reader interested in extensive coverage of interval analysis is urged to consult Moore (1966); details relating to the MRC interval analysis package are covered here and in greater depth by Yohe (1977). We focus attention on some of the elementary properties of interval analysis and list without proof (proofs can be found in Greenspan (1971) and Moore (1966)) some of the relationships which users of the interval package should be aware of. When interval analysis is applied to a mathematical method it generates bounds which form a rectangular parallelpiped which encapsulates the exact solution to a problem. The technique utilizes interval versions of the primitive arithmetic operations and combines these intervals using certain rules (defined below) to decompose mathematical expressions and problems; such decomposition is applicable to an extensive range of numerical problems.

The intervals we are dealing with are closed and denoted by [a,b] where

$$[a,b] \equiv \{x \mid a \leq x \leq b\}.$$

The elementary rules of arithmetic for intervals are as follows:

$$[a,b]+[c,d] \equiv \{x+y \mid x\in[a,b],\ y\in[c,d]\} = [a+c, b+d]$$

$$[a,b]-[c,d] \equiv \{x-y \mid x\in[a,b],\ y\in[c,d]\} = [a-d, b-c]$$

$$[a,b] \cdot [c,d] \equiv \{x\cdot y \mid x\in[a,b],\ y\in[c,d]\} = [\min(ac,ad,bc,bd), \max(ac,ad,bc,bd)]$$

If $0 \notin [c,d]$ then

$$\frac{[a,b]}{[c,d]} = \{\frac{x}{y} \mid x\in[a,b],\ y\in[c,d]\} = [\min(\frac{a}{d},\frac{a}{c},\frac{b}{d},\frac{b}{c}), \max(\frac{a}{d},\frac{a}{c},\frac{b}{d},\frac{b}{c})]$$

From the above rules it can also be inferred that

$$-[a,b] = [-b,-a]$$

$$\frac{1}{[c,d]} = [\frac{1}{d},\frac{1}{c}] \quad \text{if } 0 \notin [c,d].$$

Several theorems which are analogous to those for the real line can be established for intervals:

commutative law of addition $[a,b]+[c,d]=[c,d]+[a,b]$

commutative law of multiplication $[a,b] \cdot [c,d]=[c,d] \cdot [a,b]$

associative law of addition $([a,b]+[c,d])+[e,f]=[a,b]+([c,d]+[e,f])$

associative law of multiplication $([a,b] \cdot [c,d]) \cdot [e,f]=[a,b] \cdot ([c,d] \cdot [e,f])$

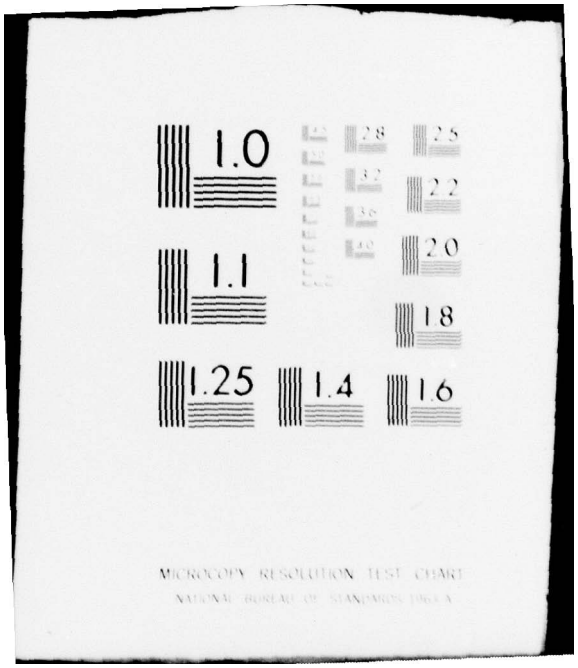
Any real number a, has an interval (degenerate) counterpart, namely $[a,a]$. Thus the interval $[0,0]$ corresponds to 0 and $[1,1]$ corresponds to 1 and hence $[0,0]$ serves as an additive interval identity (i.e. $[a,b]+[0,0]=[a,b]$) and $[1,1]$ is a multiplicative interval identity (i.e. $[a,b] \cdot [1,1]=[a,b]$). Furthermore, the above-mentioned rules for addition, subtraction, multiplication, and division of intervals when applied to degenerate intervals lead to obvious interval results analogous to the non-interval results:

AD-A070 375 SOUTHERN METHODIST UNIV DALLAS TX DEPT OF COMPUTER S--ETC F/G 9/2
IMPLEMENTATION AND EVALUATION OF INTERVAL ARITHMETIC SOFTWARE. --ETC(U)
APR 79 D A COHN, J B POTTER, M GINSBERG DACA39-77-M-0105

UNCLASSIFIED WES-TR-0-79-1 NL

2 OF 2
AD
A070376

END
DATE
FILMED
7-79
DDC


1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

$$[a,a] + [b,b] = [a+b,\ a+b]$$

$$[a,a] - [b,b] = [a-b,\ a-b]$$

$$[a,a] \cdot [b,b] = [ab,ab]$$

$$\frac{[a,a]}{[b,b]} = [\frac{a}{b},\frac{a}{b}] \qquad \text{for } b \neq 0 .$$

Extensions of interval operations to functions are also possible. For example, Moore defines for $f(x_1,x_2,\ldots x_n)$ a real rational expression in which each independent variable x_i occurs only once an interval extension

$$F(X_1,X_2,\ldots X_n)=\{f(x_1,x_2,\ldots, x_n)\,|\,x_i \epsilon X_i,\ i = 1,2,\ldots n\}$$

where each x_i ranges independently over the corresponding interval X_i .

Additional properties of interval analysis can differ somewhat from our experience with non-interval arithmetic. For instance, the definition for $[a,b]^n$ for n a positive integer is

$$[a,b]^n=\{x^n\,|\,x\epsilon[a,b]\} = \begin{cases} [a^n,b^n] \text{ if } a > 0 \\ [0,\max(a^n,b^n)] \text{ if } 0\epsilon[a,b] \text{ and } n \text{ is even} \\ [a^n,b^n] \text{ if } 0\epsilon[a,b] \text{ and } n \text{ is odd} \\ [b^n,a^n] \text{ if } b < 0 \text{ and } n \text{ is even} \\ [a^n,b^n] \text{ if } b < 0 \text{ and } n \text{ is odd} \end{cases}$$

But $[a,b]^n$ is not necessarily equal to n factors $[a,b] \cdot [a,b] \ldots [a,b]$

e.g. $[-1,1]^2 = [0,1]$ but

$[-1,1] \cdot [-1,1] = [-1,1]$.

Another example is given by the lack of an exact counterpart of the distributive law for real numbers; instead we have subdistributivity

$$[a,b]([c,d]+[e,f]) \subseteq [a,b][c,d]+[a,b][e,f]$$

i.e. exact equality does not necessarily hold for all intervals. If, however, for any real number t or for $[a,b] \cdot [c,d] > 0$, then the distributive law does hold:

$$t([e,f] \cdot [g,h]) = t[e,f] + t[g,h]$$

$$[e,f]([a,b] + [c,d]) = [e,f][a,b] + [e,f][c,d]$$

Also interval arithmetic exhibits the property of inclusion monotonicity, i.e. for $[a,b] \subset [c,d]$ and $[e,f] \subset [g,h]$

then $[a,b] + [e,f] \subset [c,d] + [g,h]$

$[a,b] - [e,f] \subset [c,d] - [g,h]$

$[a,b] \cdot [e,f] \subset [c,d] \cdot [g,h]$

$[a,b]/[e,f] \subset [c,d]/[g,h]$ if $0 \notin [e,f]$.

From the rules for interval arithmetic it is apparent that the bounds for the computed intervals usually involve several arithmetic operations which do not necessarily produce exact results, especially when dealing with finite floating-point representations as used on most machines; this error in computing the error bounds requires special attention to ensure that the endpoints of the interval resultant are specified so as not to exclude values (including the exact or true result) that should be included in the interval by the above-mentioned rules for interval arithmetic. This dilemma is resolved by using so-called directed rounding with all interval resultants, i.e. the left endpoint is rounded down and the right endpoint is rounded up; further details are given by Yohe (1973, 1977). Greenspan (1971) provides a simple decimal example of this situation. Assume that a computer can only store 4 decimal digits for each interval endpoint but that the machine determines that the result lies in the interval $J = [\cdot 11127, \cdot 21123]$. If symmetric rounding to four decimal places is used on the endpoints, then we have $J_1 = [\cdot 1113, \cdot 2112]$ but we note that a possible true solution x contained in J need not be contained in J_1 because J is not contained in J_1. By using the above-mentioned directed rounding on the interval J we obtain $J_2 = [\cdot 1112, \cdot 2113]$ and note that J_2 does indeed include all values from interval J. Of course, when applying directed rounding, one should n unduly widen the interval beyond the amount necessary to compensate for the

roundoff error propagation or otherwise the interval widths can grow unnecessarily large. Thus even though in the above example directed rounding was applied at the decimal digit level, in practice directed rounding is applied at the bit level and is dependent on the arithmetic pecularities of the floating-point arithmetic on the host machine on which interval analysis is performed.

Further consequences of using interval arithmetic are discussed in Section 5 of this paper.

TABLE 1 (from Ralston (1965))

RESULTS OF NUMERICAL SOLUTION OF $y' = -y$, $y(0) = 1$

x	Analytic solution e^{-x}	Hamming's method y	Hamming's method Error	Milne's method y	Milne's method Error	Hermite method y	Hermite method Error
.5	.60653066	.60653078	-1.2×10^{-7}	.60653078	-1.2×10^{-7}	.60653092	-2.6×10^{-7}
.6	.54881164	.54881160	4.0×10^{-8}	.54881183	-1.9×10^{-7}	.54881188	-2.4×10^{-7}
.8	.44932896	.44932865	3.1×10^{-7}	.44932910	-1.4×10^{-7}	.44932918	-2.2×10^{-7}
.9	.40656966	.40656924	4.2×10^{-7}	.40656964	2.0×10^{-8}	.40656986	-2.0×10^{-7}
1.0	.36787944	.36787894	5.0×10^{-7}	.36787953	-9.0×10^{-8}	.36787963	-1.9×10^{-7}
13.0	2.2603294×10^{-6}	2.2602384×10^{-6}	9.1×10^{-11}	6.3541098×10^{-6}	-4.1×10^{-6}	2.2603344×10^{-6}	-5.0×10^{-12}
13.5	1.3709591×10^{-6}			-3.4646962×10^{-6}	4.8×10^{-6}		
14.0	8.3152872×10^{-7}	8.3149279×10^{-7}	3.6×10^{-11}	6.5434628×10^{-6}	-5.7×10^{-6}	8.3153061×10^{-7}	-1.9×10^{-12}
14.5	5.0434766×10^{-7}			-6.2426818×10^{-6}	6.7×10^{-6}		
15.0	3.0590232×10^{-7}	3.0588805×10^{-7}	1.4×10^{-12}	8.2755880×10^{-6}	-8.0×10^{-6}	3.0590305×10^{-7}	-7.3×10^{-13}

TABLE 2 (from Ralston (1965))

RESULTS OF NUMERICAL SOLUTION OF $y' = \frac{1}{1 + \tan^2 y}$, $y(0) = 0$

x	Analytic solution $\tan^{-1} x$	Hamming's method y	Hamming's method Error	Milne's method y	Milne's method Error	Hermite method y	Hermite method Error
.4	.38050638	.38050639	-1.0×10^{-8}	.38050639	-1.0×10^{-8}	.38050639	-1.0×10^{-8}
.5	.46364761	.46364675	8.6×10^{-7}	.46364707	5.4×10^{-7}	.46364772	-1.1×10^{-7}
2.0	1.1071487	1.1071434	5.3×10^{-6}	1.1071478	9.0×10^{-7}	1.1071490	-3.0×10^{-7}
3.0	1.2490458	1.2490442	1.6×10^{-6}	1.2490451	7.0×10^{-7}	1.2490458	0.0
4.0	1.3258177	1.3258166	1.1×10^{-6}	1.3258169	8.0×10^{-7}	1.3258177	0.0
5.0	1.3734008	1.3733999	9.0×10^{-7}	1.3734000	8.0×10^{-7}	1.3734009	-1.0×10^{-7}
6.0	1.4056476	1.4056473	3.0×10^{-7}	1.4056468	8.0×10^{-7}	1.4056478	-2.0×10^{-7}
7.0	1.4288993	1.4288989	4.0×10^{-7}	1.4288984	9.0×10^{-7}	1.4288993	0.0
8.0	1.4464413	1.4464410	3.0×10^{-7}	1.4464403	1.0×10^{-6}	1.4464413	0.0
9.0	1.4601391	1.4601390	1.0×10^{-7}	1.4601379	1.2×10^{-6}	1.4601393	-2.0×10^{-7}
10.0	1.4711277	1.4711278	-1.0×10^{-7}	1.4711264	1.3×10^{-6}	1.4711280	-3.0×10^{-7}
11.0	1.4801364	1.4801369	-5.0×10^{-7}	1.4801350	1.4×10^{-6}	1.4801367	-3.0×10^{-7}
12.0	1.4876551	1.4876561	-1.0×10^{-6}	1.4876535	1.6×10^{-6}	1.4876551	0.0

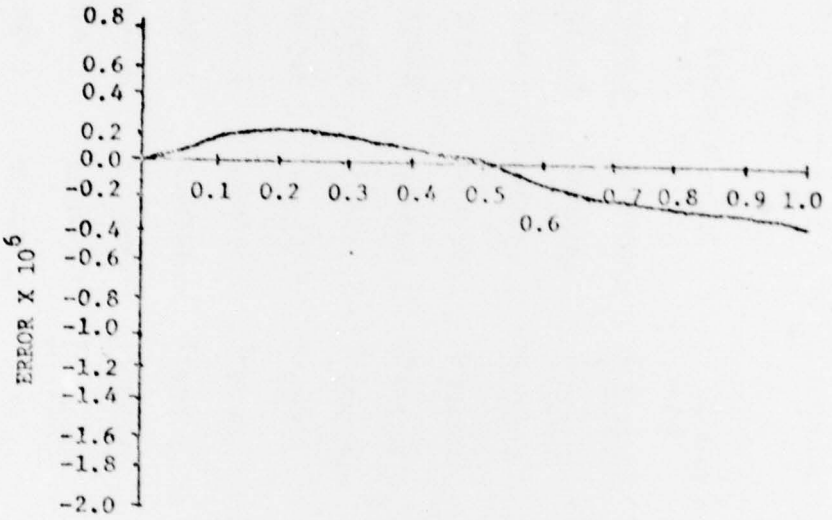


FIGURE 1 (from Lapidus and Seinfeld (1971))

ERROR VERSUS x FOR THE FIFTH ORDER NYSTROM METHOD

APPLIED TO PROBLEM (a)

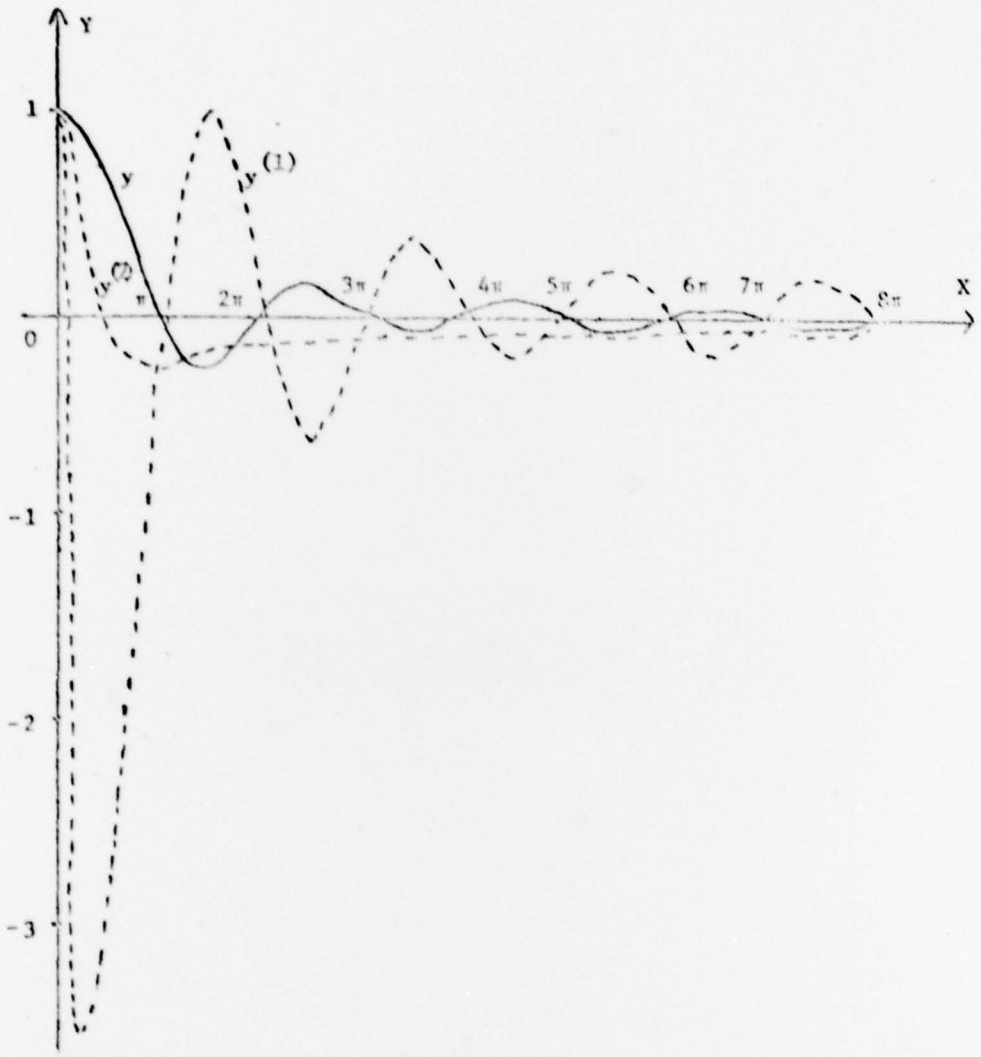


FIGURE 2 (from Greenspan (1967))

A GRAPH OF NUMERICAL SOLUTIONS $y^{(1)}$ AND $y^{(2)}$ FOR A LINEARIZED EMDEN DIFFERENTIAL EQUATION

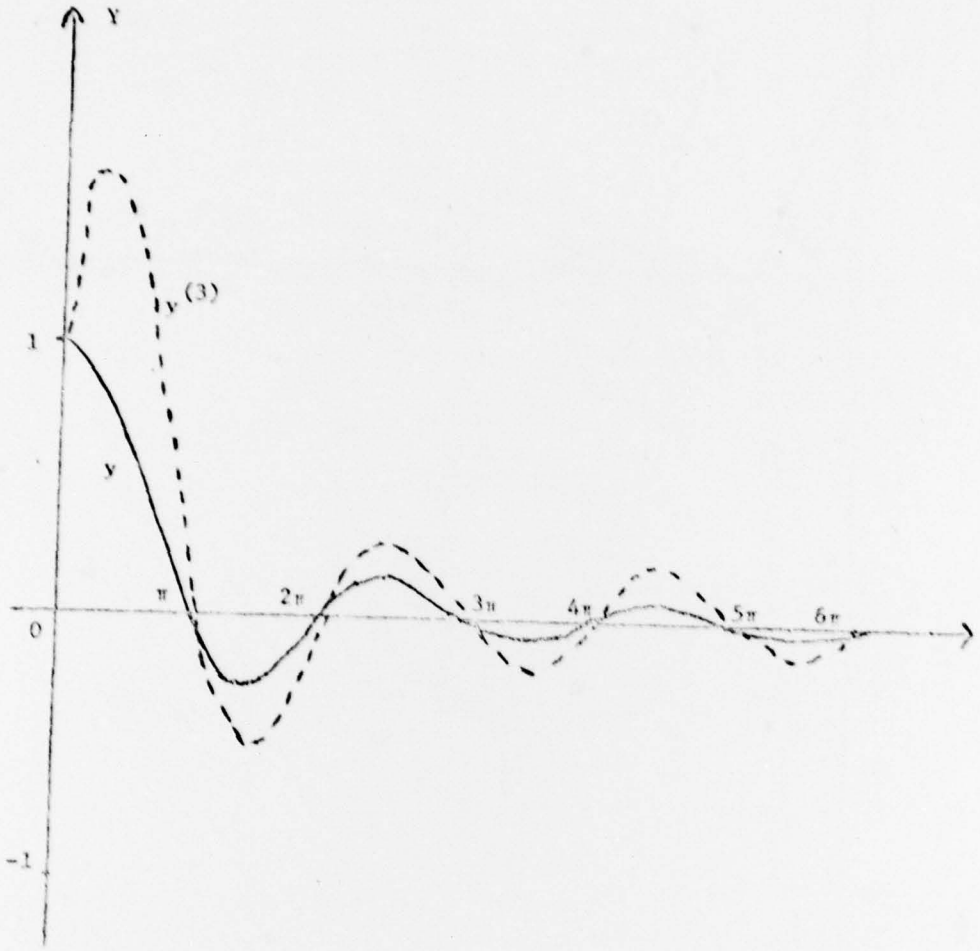


FIGURE 3 (from Greenspan (1967))

A GRAPH OF NUMERICAL SOLUTION $y^{(3)}$ FOR A LINEARIZED EMDEN DIFFERENTIAL EQUATION

References

1. Appl, F. C. and H. M. Hung, "A Principle for Convergent Upper and Lower Bounds," Intern. J. Mech. Sci., Vol. 6, 1964, pp. 381-389.

2. Baranov, A. V., "New Method for Solving Differential Equation of the Type

$$F_n \frac{d}{dx} F_{n-1}(x) \frac{d}{dx} \ldots F(x) \frac{dy}{dx} - y(x) = F(x)",$$

Zh. Vych. Math., Vol. 4, No. 5, 1964, pp. 920-926.

3. Boley, B. A., "Upper and Lower Bounds for the Solution of a Melting Problem," Quart. App. Math., Vol. 21, No. 1, April 1963, pp. 1-11.

4. Fabry, C., Nested Bounds for Solutions of Differential Equations, SACLANTCEN Technical Memorandum No. 155 (unclassified), SACLANT ASW Research Center, La Spezia, Italy, 1970.

5. Greenspan, D., Approximate Solution of Initial Value Problems for Ordinary Differential Equations by Boundary Value Techniques, MRC Technical Summary Report No. 752, Mathematics Research Center, University of Wisconsin, Madison, June 1967, pp. 15-16.

6. Greenspan, D., Introduction to Numerical Analysis and Applications, Markham, Chicago, 1971, pp. 147-163.

7. Ispolov, Y. G. and F. C. Appl, "Bounds for Limit Cycles of Self-Sustained Vibrations," J. Sound Vib., Vol. 15, No. 2, 1971, pp. 163-173.

8. Kahan, W., "Ellipsoidal Bounds for the Propagation of Uncertainty Along Trajectories," presentation at Conference on the Numerical Solution of Ordinary Differential Equations, University of Texas, Austin, October 20, 1972.

9. Lapidus, L. and J. H. Seinfeld, Numerical Solution of Ordinary Differential Equations, Academic Press, New York, 1971, p. 88.

10. Moore, R. E., Interval Analysis, Prentice-Hall, Englewood Cliffs, New Jersey, 1966.

11. Ralston, A., A First Course in Numerical Analysis, McGraw-Hill, New York, 1965, pp. 205-209.

12. Strakhov, N. A. and L. Kurz, "An Upper Bound on the Zero Crossing Distribution," Bell System Tech. J., Vol. 47, No. 4, April 1968, pp. 529-547.

13. Weinstein, A. and W. Stenger, Methods of Intermediate Problems for Eigenvalues, Academic Press, New York, 1972.

14. Willson, A. N., Jr., "On the Solution of Equations for Nonlinear Resistive Networks," Bell System Tech. J., Vol. 47, No. 8, October 1968, pp. 1755-1773.

15. Yohe, J. M., The Interval Arithmetic Package, MRC Technical Summary Report No. 1755, Mathematics Research Center, University of Wisconsin, Madison, June 1977.

16. Yohe, J. M. "Roundings in Floating-Point Arithmetic," IEEE Trans. Computers, Vol. C-22, No. 6, June 1973, pp. 577-586.

APPENDIX C: STANDARD FORTRAN NUMBER AND INTERVAL NUMBER REPRESENTATIONS

```
DIGIT        ::= 0|1|2|3|4|5|6|7|8|9

SIGN         ::= +|-

INTEGER      ::= NULL|<SIGN>|<INTEGER><DIGIT>

RADIX        ::= .

FIXEDPOINT   ::= <INTEGER><RADIX>|<FIXEDPOINT><DIGIT>

EXPSEP       ::= E|D

EXPONENT     ::= <SIGN>|<EXPSEP>|<EXPSEP><SIGN>|<EXPONENT><DIGIT>

NUMBER       ::= <INTEGER>|<FIXEDPOINT>|<INTEGER><EXPONENT>|
                 <FIXEDPOINT><EXPONENT>

ENDPTSEP     ::= :

COMMA        ::= ,

INTERVAL     ::= <NUMBER>|(<NUMBER>)|<NUMBER><ENDPTSEP><NUMBER>|
                 (<NUMBER><ENDPTSEP><NUMBER>)|(<NUMBER><COMMA><NUMBER>)
```

In accordance with letter from DAEN-RDC, DAEN-ASI dated 22 July 1977, Subject: Facsimile Catalog Cards for Laboratory Technical Publications, a facsimile catalog card in Library of Congress MARC format is reproduced below.

Cohn, David A
Implementation and evaluation of interval arithmetic software; Report 5: The CDC CYBER 70 System / by David A. Cohn, J. Brian Potter, Myron Ginsberg, Department of Computer Science, Southern Methodist University, Dallas, Tex. Vicksburg, Miss. : U. S. Waterways Experiment Station ; Springfield, Va. : available from National Technical Information Service, 1979.
78, [28] p. : ill. ; 27 cm. (Technical report - U. S. Army Engineer Waterways Experiment Station ; O-79-1, Report 5)
Prepared for Office, Chief of Engineers, U. S. Army, Washington, D. C., under Contract No. DACA39-77-M-0105.
Includes bibliographies.

1. CDC CYBER 70 System. 2. Computer systems programs. 3. Interval arithmetic. I. Ginsberg, Myron, joint author. II. Potter, J. Brian, joint author. III. Dallas. Southern Methodist University. Dept. of Computer Science. IV. United States. Army. Corps of Engineers. V. Series: United States. Waterways Experiment Station, Vicksburg, Miss. Technical report ; O-79-1, Report 5.
TA7.W34 no.O-79-1 Report 5