PERSPECTIVES IN COMPUTING, Vol. 19

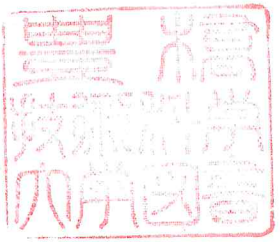
(Formerly "Notes and Reports in Computer Science and Applied Mathematics")

Reliability in Computing

The Role of Interval Methods in Scientific Computing

Edited by

Ramon E. Moore
Department of Computer and Information Science
Ohio State University
Columbus, Ohio



ACADEMIC PRESS, INC.
Harcourt Brace Jovanovich, Publishers

Boston San Diego New York
Berkeley London Sydney
Tokyo Toronto

ACADEMIC PRESS, INC.
1250 Sixth Avenue, San Diego, CA 92101

United Kingdom Edition published by
ACADEMIC PRESS INC. (LONDON) LTD.
24-28 Oval Road, London NW1 7DX

Library of Congress Cataloging-in-Publication Data

Reliability in computing.
(Perspectives in computing ; vol. 19)
Bibliography: p.
1. Electronic data processing—Reliability.
2. Interval analysis (Mathematics) I. Moore, Ramon E.
II. Series.
QA76.9.E94R45 1988 004 88-3479
ISBN 0-12-505630-3

88 89 90 91 9 8 7 6 5 4 3 2 1
Printed in the United States of America

Algorithms for Verified Inclusions: Theory and Practice

Siegfried M. Rump *)
IBM Germany
Development and Research
Schoenaicher Strasse 220
7030 Boeblingen
West Germany

Summary. In the following basic principles of algorithms computing guaranteed bounds are developed from a theoretical and a practical point of view. Some fundamental theoretical facts are repeated where, for more detailed information, the reader is referred to the literature (e.g. [Ru83], [Ru87] and the references mentioned in these papers).

Furthermore practical aspects are discussed, especially how the process of computing a guaranteed result really works out on a digital computer. The verification process is performed by means of checking assumptions of mathematical theorems. This checking process is performed automatically. The various steps from the mathematical theorem down to the practical verification are described in detail.

In contrast, standard floating-point algorithms usually deliver good approximations to the solution of a given numerical problem but there is neither a verification or guarantee about the quality of the approximation nor must a solution to the given problem actually exist. There are simple examples where the floating-point approximation is drastically wrong.

A programming environment has been developed which allows to specify commands to the computer in mathematical notation. Because the system (preliminary name ABACUS) works interactively, no type specificaiton is necessary at all allowing to specify algorithms like in a math book.

---

*) present address: Informatik III, Technical University, Eissendorfer Str. 38, 2100 Hamburg 90, West Germany

ABACUS works right now on IBM System /370 machines under VM operating system. It is planned to have a C version for IBM System /2, SUN work stations and others available by early 1988. Some examples demonstrating the system are presented.

0. Introduction. The theoretical basis for algorithms with guaranteed results has been developed in a series of papers since 1979. The algorithms are based on a so-called inclusion theory providing necessary conditions for standard problems of numerical analysis to be solvable and yielding regions in a constructive way, where there is provably a uniquely defined solution of the given problem. The basic property of all those theorems is that the assumptions can effectively be verified on digital computers.

The inclusion theory provides theorems for a large number of standard numerical problems such as general systems of linear equations, special linear systems with band or symmetric matrix, matrix inversion, algebraic eigenproblems, polynomial zeros, polynomial evaluation, linear, quadratic and convex programming problems, evaluation of arithmetic expressions, over- and underdetermined linear systems, general nonlinear systems of equations, differential equations and others. The key property of the algorithms based on the inclusion theory is that every result is verified to be correct by means of the automatical proof that the problem is solvable (non-singularity) and by delivering bounds for the solution. In case a problem is not solvable (e.g. inversion of a singular matrix) a respecting message is given.

A common approach to estimate the error of a floating-point computation is to evaluate in more than one precision and to compare the results. However, this does not imply any guarantee of the correctness of coinciding figures. Consider the following example. Compute

$$f = 333.75\, b^6 + a^2 \,(\, 11a^2b^2 - b^6 - 121b^4 - 2\,) + 5.5b^8 + a/(2b)$$

$$\text{for} \quad a = 77617.0 \quad \text{and} \quad b = 33096.0 \quad .$$

To calculate the value of the polynomial a FORTRAN program has been written, the computer in use is a S/370 main frame. All input data is exactly representable, the only errors occurring in the computation are rounding errors and mainly cancellation errors. In order to test the arithmetic rather than standard functions every exponentiation is replaced by successive multiplications. The program calculates the values for f in single, double and extended precision equivalent to approximately 6, 17 and 34 decimal digits precision. The obtained values are the following:

single precision : f = + 1.172603 ...
double precision : f = + 1.1726039400531 ...
extended precision : f = + 1.172603940053178 ...

All three values agree in the first 7 figures, whereas the true value for f is

exact value : $f \in -0.827396059946821\frac{4}{3}$

indicating that the first figures -0.827396... are guaranteed and the sixteenth figure after the decimal point is between 3 and 4. This result was obtained by an ACRITH algorithm yielding verified inclusions (cf. [ACR86]). It is guaranteed to be correct.

Analyzing the expression above yields immediately the extreme sensitivity with respect to the input data. The 8th power of a 5-digit number yields a 40 digit result and a 1 figure (left of the decimal point) result on a 34-digit computer is by no means of any significance. On the other hand the polynomial need not to occur at once, the input data may be read from a file and the user can't analyze every operation in a million operation program.

The interactive programming environment to be described and used in chapter 3 adapts the mathematical notation as close as possible. E.g., a complex number is written as 3-5i (as an example), a tolerance can be specified by 5+/-.0001, operators like matrix multiplication or scalar products are directly accessible thru the multiplication operator etc.

The interactive programming environment ABACUS has been developed without knowing MATLAB [MAT86]. Finally many notations in the part of ABACUS which is common to MATLAB turn out to be very similar to the MATLAB notation (without having known them before). This fact gives the author much confidence that the notation is indeed very near the mathematical notation and is easy to learn and to understand.

1. Basic theorems. We start with a brief description of the mathematical basis, the inclusion theory. Consider a system of nonlinear equations $f(x) = 0$, where $f:R^n \to R^n$ is a continuously differentiable function in n unknowns. Consider the Newton iteration

$$x^{k+1} = x^k - \{f'(x^k)\}^{-1} \cdot f(x^k)$$

and the simplified Newton iteration

$$x^{k+1} = x^k - R \cdot f(x^k) \quad ,$$

where R is an approximate inverse of f' at some fixed point

$\tilde{x}$. Consider the function $g:R^n \to R^n$ defined by

$$g(x) = x - R \cdot f(x) \quad .$$

If there is a set X, an element of the power set PR^n of R^n, which is compact and convex such that g maps X in itself, then by the Fixed Point Theorem of Brouwer there is a fixed point $\hat{x}$ of the function g in X.

To obtain a zero of the function f from a fixed point of the function g a contraction principle is used. Consider the following theorem (cf. [Ru87]):

Theorem 1. Let $Z \in PR^n$ be a set of vectors, $C \in PR^{nxn}$ be a set of matrices and $X \in PR^n$ be a compact set of vectors. If then

$$Z + C \cdot X \subseteq \text{int}(X) \quad ,$$

then for every matrix $C \in C$ holds: $\rho(C) < 1$.

Here int(X) denotes the interior of the set X. The operations in theorem 1 and throughout this chapter are the power set operations.

Previous versions of this theorem have been given in [Ru83]. Note that in theorem 1 the set X is not supposed to be convex. Theorem 1 allows to verify the contraction property of a matrix on the computer. In fact, if the sets are represented e.g. as intervals, then the convergence of a matrix can be automatically verified on computers without any norm estimation (which is, in general, an overestimation).

Theorem 1 can be applied to systems of nonlinear equations. With the definition of a Jacobian for sets:

$$f'(X) := \bigcap \{Y \in IR^n \mid f'(x) \in Y \text{ for all } x \in X\}$$

for $X \in PR^n$ we have the following theorem:

Theorem 2. Let $f:R^n \to R^n$ be a continuously differentiable function in n unknowns, $R \in R^{nxn}$ be a matrix, $\tilde{x} \in R^n$ be a vector and $X \in PR^n$ be a compact set of vectors. If then

$$(1) \quad \tilde{x} - R \cdot f(\tilde{x}) + \{ Id - R \cdot f'(\tilde{x} \cup X) \} \cdot (X - \tilde{x}) \subseteq \text{int}(X),$$

then the system of nonlinear equations $f(x)=0$ has a unique solution $\hat{x}$ in X.

Id denotes the identity matrix and $\underline{\cup}$ the convex union. Theorem 2 verifies the existence and uniqueness of a solution of f(x)=0 in the inclusion X. There are corresponding theorems for a large number of numerical standard problems listed above with similar properties (cf. [Ru83]).

Note that theorem 2 does neither require additional information on the matrix R nor on the function f(such as R being nonsingular or f having a zero nearby the approximation x). All properties necessary prove thru assumption (1) which can be verified on computers.

Theorem 2 applies directly to systems of linear equations. We mention a respective theorem explicity because in the subsequent examples we will deal with systems of linear equations for the sake of simplicity. However, it should be mentioned that also larger systems of nonlinear equations have been treated with great success, i.e. very sharp bounds for the solution have been computed (see [Ru83] and [Ru87]). The examples in these papers include ill-conditioned systems of nonlinear equations.

Theorem 3. Let $A \in R^{nxn}$ be a real matrix, $b \in R^n$ be a vector, $R \in R^{nxn}$ be a matrix, $\tilde{x} \in R^n$ be a vector and $X \in PR^n$ be a compact set of vectors. If then

$$(2) \quad R \cdot \{ b - A \cdot \tilde{x} \} + \{ Id - R \cdot A \} \cdot X \subseteq \text{int}(X) \quad ,$$

then the matrices R and A are not singular, the linear system $A \cdot x = b$ is uniquely solvable and the solution $\hat{x}$ satisfies $\hat{x} \in \tilde{x} + X$.

The inclusion theory also allows to handle data afflicted with tolerances. This will be illustrated by an example of a theorem for systems of linear equations.

Theorem 4. Let $A \in PR^{nxn}$ be a set of matrices, $b \in PR^n$ be a set of vectors, $R \in R^{nxn}$ be a matrix, $\tilde{x} \in R^n$ be a vector and $X \in PR^n$ be a compact set of vectors. If then

$$(3) \quad R \cdot \{ b - A \cdot \tilde{x} \} + \{ Id - R \cdot A \} \cdot X \subseteq \text{int}(X) \quad ,$$

then the following is true: For every real matrix A with $A \in A$ and for every real vector b with $b \in b$ the system of linear equations $A \cdot x = b$ is uniquely solvable and the solution $\hat{x}$ satisfies $\hat{x} \in \tilde{x} + X$.

The capability of solving problems afflicted with tolerances is very important. As soon as a single (real) problem within the tolerances is not solvable due to a singularity this fact is reported by the corresponding algorithm. There are

theorems like the one above for a large number of numerical problems (cf. [Ru83]).

The theorems mentioned above apply as well to complex data and to complex data afflicted with tolerances. For more details see e.g. [Ru83].

2. Practical verification on the computer. The power set operations required in theorems 1 to 4 in the previous chapter are in general not executable on digital computers. For the purpose of verifying the assumptions we will use interval operations.

The basic property of interval arithmetic is isotonicity. This property will be used several times in the following discussions. For details about interval arithmetic see for instance [AlHe83] or [Mo79].

In the following illustrations we will use a 3-decimal-digit computer with exponent range large enough to avoid over- and underflow. The number representation will cover 3 significant decimal digits (3 digits in the mantissa).

An interval is a set of real or complex numbers, vectors or matrices. In the following we will use rectangles over real and complex numbers. Then a real interval is a set of the form

$$\{ x \in R \mid a \leq x \leq b \} \quad \text{for some } a,b \in R \quad .$$

A complex interval is defined similarly using the induced order relation:

$$\{ x \in C \mid a \leq x \leq b \} \quad \text{for some } a,b \in C \quad .$$

Interval vectors and matrices are defined to be vectors and matrices of intervals, respectively. We denote the set of intervals, interval vectors and interval matrices over real numbers by IR, IR^n and IR^{nxn}, those over the complex numbers by IC, IC^n and IC^{nxn}, respectively.

If the left and right endpoints of an interval are (real or complex) floating-point numbers, we denote the set of those intervals by IF, IF^n, IF^{nxn}, ICF, ICF^n and ICF^{nxn}, respectively. In this case the two floating-point bounds describe a set of real or complex numbers on the computer.

On a 3-decimal-digit computer the number π can be represented by [3.14,3.15] including the true number π=3.14159...

All basic operations can be performed over intervals, e.g. $\pi \cdot \pi \in$ [9.85,9.93] obtained by multiplying the left and

right bounds, respectively. The bounds of the result are always rounded outwards. Interval calculations do, in general, not cover dependencies between variables. For instance

$$\pi - \pi \in [3.14,3.15] - [3.14,3.15] = [-0.01,+0.01]$$

introducing an overestimation. All standard functions can be applied to intervals as well. E.g.

$$\sin(\pi) \in \sin([3.14,3.15]) = [\sin(3.15),\sin(3.14)] \subseteq [-0.00841,+0.00160]$$

using monotonicity properties of the sine function near π. In case of extrema within the argument interval special care is necessary:

$$\sin(\pi/2) \in \sin([3.14,3.15]/2) \subseteq \sin([1.57,1.58] \subseteq [\min\{\sin(1.57),\sin(1.58)\},1.00] \subseteq [0.999,1.00] \quad .$$

With case selections all standard functions can be extended to interval argument (see [Br87] and [Kr87]).

The essential property of interval analysis, the isotonicity, yields that when replacing all operations including standard functions by the corresponding interval operations and interval standard functions the computed result will definitely include the true result. Because of the earlier mentioned overestimation by interval operations the functions to be evaluated have to be defined in a way to minimize this overestimation. The formulas (1) to (2) in theorems 2 to 4 meet this requirement.

The evaluation of an inclusion of a function $f:R^n \to R^n$ can be split in the following five steps:

1) $f(x)$, $x \in R^n$ with real operations over R^n
2) $g(X)$, $X \in IF^n$ with $g(X):=\{f(x) \mid x \in X\}$ and $x \in X$
3) $h(X)$, $X \in IF^n \subseteq PR^n$ replacing operations in g by power set operations over R^n
4) $H(X)$, $X \in IF^n \subseteq IR^n$ replacing operations in h by interval operations over R^n
5) $F(X)$, $X \in IF^n$ replacing operations in H by interval operations over F^n

We will illustrate the five steps in the following by means of an example. Let

$$f(x) := (x + \pi i) \cdot (x - \pi i)$$

where we wish to calculate f(e), e being the base of the

natural logarithm. Obviously

$$f(x) = x^2 + \pi^2 \quad .$$

Therefore the result of the first step is a real number

$$1) \quad f(e) = e^2 + \pi^2 = 17.25866... \in R \quad ,$$

which is in fact the true result. In the second step numbers which cannot be exactly representable on the computer are replaced by the smallest interval (with floating-point bounds) containing that number. In our example we have with E=[2.71,2.72] and P=[3.14,3.15]:

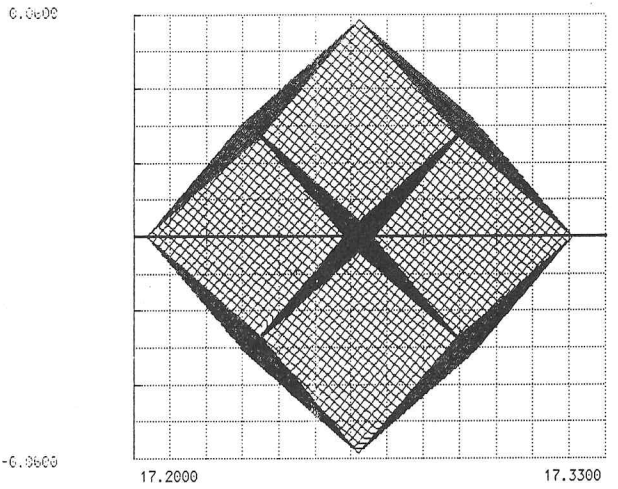
$$2) \quad g(E) = \begin{Bmatrix} f(x) \mid x \in E, \ x \in R^n \end{Bmatrix} = \\ \begin{Bmatrix} (a+p\cdot i)\cdot(a-p\cdot i) \mid a \in E, \ p \in P \end{Bmatrix} = \\ \begin{Bmatrix} a^2+p^2 \mid a \in E, \ p \in P \end{Bmatrix} = \\ \begin{Bmatrix} x \in R \mid 17.2037 \leq x \leq 17.3209 \end{Bmatrix} \quad .$$

The evaluation makes use of the monotonicity of the square function. This evaluation is, in general, not possible on computers because it requires real operations within the computation of the function values of f and requires a complete analysis of the extrema of f.

In the third step the function g is evaluated by using power set operations in a step by step mode, i.e. replacing every operation in the formula for g by its corresponding power set operation. This introduces an overestimation due to variables occurring more than once. In our example we have for E=[2.71,2.72] and P=[3.14,3.15]:

$$3) \quad h(E) = \\ \begin{Bmatrix} (E+P\cdot i)\cdot(E-P\cdot i) \mid +,-,\cdot \text{ power set operations} \end{Bmatrix} \\ \begin{Bmatrix} (a1+p1\cdot i)\cdot(a2-p2\cdot i) \mid a1,a2 \in E, \ p1,p2 \in P, \\ +,-,\cdot \text{ real operations} \end{Bmatrix}$$

The result is a belly out square. Regarding E+P·i and E-P·i as complex intervals this curve can be sketched by taking sample points on the boundary of the first interval and multiplying them by the boundary of the second interval (each product yielding a rectangle) and plotting all result-rectangles in one picture. The result (obtained by ABACUS) is the following:



In general the result set looks quite more complicated. Especially the result is usually not convex and not bounded by straight lines.

In the fourth step the power set operations used in step 3 are replaced by interval operations over R. That means the bounds of the interval are real numbers. The result using E=[2.71,2.72] and P=[3.14,3.15] (all operations are interval operations over R) is:

```
4)   H(E) = (E+P•i)•(E-P•i) =
           ( E•E + P•P ) + i•( P•E - E•P ) =
           ( [7.3441,7.3984] + [9.8596,9.9225] ) +
             i•( [8.5094,8.568] - [8.5094,8.568] ) =
           [17.2037,17.3209  + i•[-0.0586,+0.0586]
```

There is no overestimation introduced in the real part because of the monotonicity of the square function (over the positive real axis). The imaginary part is overestimated compared to step 2.

The final step is replacing the interval operations over the real numbers by interval operations over floating-point numbers. The result using E=[2.71,2.72] and P=[3.14,3.15] (all operations are interval operations over F) is:

```
5)   F(E) = (E+P•i)•(E-P•i) =
           ( E•E + P•P ) + i•( P•E - E•P ) =
           ( [7.34,7.40] + [9.85,9.93] ) +
             i•( [8.50,8.57] - [8.50,8.57] ) =
           [17.1,17.4] + i•[-0.07,+0.07]
```

This is the final result when replacing real numbers which are not exactly representable by floating-point numbers by the corresponding smallest floating-point intervals and replacing real operations by the corresponding interval operations.

Clearly, the result of every step is included in the result of the succeeding step:

$$f(x) \in g(X) \subseteq h(X) \subseteq H(X) \subseteq F(X) \quad .$$

For f(Y), where Y is element of PR^n, an extra step has to be introduced. We regard the calculation of f(Y) as the 0-th step:

0) f(Y), $Y \in PR^n$ with $f(Y) := \{f(y) \mid y \in Y\}$

1) f(Z), $Z \in IR^n$ with $f(Z) := \{f(z) \mid z \in Z\}$ and $Y \subseteq Z$

2) g(X), $X \in IF^n$ with $g(X) := \{f(x) \mid x \in X\}$ and $Z \subseteq X$

3) h(X), $X \in IF^n \subseteq PR^n$ replacing operations in g by power set operations over R^n

4) H(X), $X \in IF^n \subseteq IR^n$ replacing operations in h by interval operations over R^n

5) F(X), $X \in IF^n$ replacing operations in H by interval operations over F^n

Here steps 0 to 2 change the set of vectors Y (which is, in general, not representable on computers) into the interval vector X with floating-point endpoints.

Again, the result of every step is included in the result of the succeeding step. If f is the function occurring on the left hand side of formula (1), (2) or (3) of the preceding chapter, then $F(X) \subseteq int(X)$ implies immediately $f(X) \subseteq int(X)$ proving the assumption of theorems 2, 3 or 4, resp. from which the assertions follow. This is the process of automatic verification on the computer, because $F(X) \subseteq int(X)$ can indeed be checked on digital computers.

3. Interactive Programming Environment. An interactive programming environment has been written with the following objectives:

- All commands close to mathematical notation
- support of all vector and matrix operations
- support of real and complex numbers and intervals
- built-in precise scalar product
- access to a large subroutine library
- generic concepts in the definition of operators

The programming environment runs on all IBM S/370 machines. The interpretative part is written in PASCAL, all numerical subroutines in FORTRAN. Access to LINPACK and EISPACK is provided. The programming has been performed by Dirk Husung in very short time with superb quality.

Although the syntax was defined without knowing MATLAB it turned out to have a number of similarities. This gives the author much confidence that indeed a definition has been found close to mathematical terms.

Following we give some examples how to use the programming environment. Due to limited space we can only give very few highlights. The input lines (input from the console or from a file) are marked by "I" at the far right hand side, output lines are marked by an "O". All computations are performed in /370 double precision, i.e. 14 hex or approximately 16..17 decimal digits. We start with some examples on accurate scalar products.

```
exec pascal(5,P), P                                          I

P = (5,5)-real-point-matrix                                  O
                                                             O
     2     3     4     5     6                               O
     3     6    10    15    21                               O
     4    10    20    35    56                               O
     5    15    35    70   126                               O
     6    21    56   126   252                               O
```

First we generate a 5x5 Pascal matrix, which is the top of a Pascal triangle. Next we solve a linear system with matrix P and right hand side b purely numerical, i.e. using LINPACK routines.

```
b=(1:5)'; format f10; x=P\b                                  I

x = (5,1)-real-point-matrix                                  O
                                                             O
    -1.5000000000                                            O
     3.3333333333                                            O
    -2.5000000000                                            O
     1.0000000000                                            O
    -0.1666666667                                            O
```

The residual P*x-b is calculated using two separate operations, the matrix-vector multiplication (the result of which is rounded) and the vector-vector subtraction. Due to cancellation few digits of the result will be correct.

```
P*x-b                                                          I

ANS = 1E-14 * (5,1)-real-point-matrix                          O
                                                               O
      1.1102230246                                             O
      1.1990408666                                             O
      1.1324274851                                             O
      0.3996802889                                             O
      0.2442490654                                             O
```

As expected the residual is of small magnitude (common factor 1E-14). Floating-point arithmetic displays just as many figures as are asked for regardless how many figures are correct. We can check on the accuracy of these floating-point figures by calculating the residual using interval arithmetic. The result will be as sharp as possible because there are no overestimations: every variable occurs only once.

```
ival P*x-b                                                     I

ANS = 1E-14 * (5,1)-real-interval-matrix                       O
                                                               O
      1.1_________                                             O
      1.2_________                                             O
      1.1_________                                             O
      0.4_________                                             O
      0.2_________                                             O
```

In our interval format only as many digits are displayed as are correct. To be precise: Adding and subtracting one unit in the digit displayed immediately left to the left-most underline-bar yields a correct inclusion interval of the result.

Finally we calculate the residual using a precise inner product. This is called just by placing an ! left to the expression to be evaluated. If the user wishes to store the result using several residuals (Stetter calls it "staggered correction") he simply specifies k! where k denotes the number of residuals to be stored.

```
!(P*x-b)                                                       I

ANS = 1E-14 * (5,1)-real-point-matrix                          O
                                                               O
      1.1088352458                                             O
      1.2032042029                                             O
      1.1393663790                                             O
      0.4080069615                                             O
      0.2359223927                                             O
```

These figures are all correct because all inner products are calculated with maximum accuracy. As expected it shows that only two figures of the floating-point result are correct. Several other formats are available such as the i-format showing left and right bounds of an interval, e-format with exponent for every number, h-format for hexadecimal output.

Next a non-contextfree feature of ABACUS will be demonstrated. If in an expression in ABACUS at least one variable is of type interval (scalar, vector or matrix), then the entire expression will be evaluated using interval operations (including solution of linear systems, matrix inversion etc. using the inclusion algorithms).

Consider the exponential function of a matrix. The result matrix can be calculated using a truncated Taylor series and Horner's scheme. Following is a small ABACUS-program for that purpose:

```
// C = exp(A) evaluated by Horner's scheme (n terms)        I
                                                            I
module (n,A,C);                                             I
  C = A;                                                    I
  for i=n:2:-1 do  C = ( C/i + Id ) * A ;                   I
  C = C + Id;                                               I
```

Input parameters are the highest exponent n, the input matrix A and the result matrix C. Within the program the generic variable Id is used, which is an identity matrix adjusting its size automatically. This program is stored in the file exp and called by its name.

Before applying the program we check on the eigenvalues of P.

```
format f5; eig(P)                                           I

ANS = (5,1)-real-point-matrix                               O
                                                            O
     0.00528                                                O
     0.13946                                                O
     1.58572                                                O
    15.43123                                                O
   332.83831                                                O
```

We change the matrix P such that the spectral radius is less than one:

```
P=P/333; eig(P)                                          I

ANS = (5,1)-real- point-matrix                           O
                                                         O
      0.00002                                            O
      0.00042                                            O
      0.00476                                            O
      0.04634                                            O
      0.99951                                            O
```

The matrix P has been changed forced to be contracting. Using the Taylor series for the computation of exp(P) up to n=20 therefore yields an error less than 1/20!≈4.1E-19 or at least 18 figures (with exact computations). We check this result against the built-in exponential function for matrices by calculating the norm of the difference matrix (again the notation is very near to mathematical notation):

```
exec exp(20,P,C); D=exp(P); |C-D|                        I

ANS =                                                    O
                                                         O
   1.22663E-15                                           O
```

This is a purely numerical result without any guarantee involved. It suggests that the built-in exponential function for matrices (in this case) is accurate to 15 figures. We can easily change this approximate result into a guaranteed result. We simply replace the matrix P by P+/-O forcing the argument to be of type interval. This implies all operations in the subroutine exp to be performed in interval arithmetic with a result (interval) matrix of error less than 1E-19.

```
exec exp(20,P+/-O,C); D=exp(P); |C-D|                    I

ANS = interval                                           O
                                                         O
   1._____E-15                                           O
```

The result shows that indeed the built-in matrix exponential function is at least accurate to 15 figures in this example. This result is guaranteed to be correct.

ABACUS supports traditional statements such as for, loop, if, exit etc. The syntax is much like as in so-called pseudo-programs frequently listed in math papers.

Many of the constructs are generic. For example the colon is used for matrix and vector arguments to switch from standard mathematical operators to elementwise operators. This is true for a number of operators such as +, -, *, /, relational operators, all trig/exp/log functions, sqrt and

others. This is one concept the user has to learn once and may apply it to many built-in routines.

Performing a sensitivity analysis using ABACUS is very simple. Consider the previously defined matrix P. We invert the matrix P with a relative tolerance of 1E-8. That means every matrix within a relative distance to P of less than or equal 1E-8 will be inverted, the non-singularity of every such matrix will be shown and the set of all inverses of those matrices will be included. The result is displayed below (here only the first three columns are shown; the remaining two look similar):

```
format f8;    inv(P*(1+/-1e-8))                                    I

ANS = 1E+04 * (5,5)-real-interval-matrix                           O
                                                                   O
     0.5828____    -1.166_____    1.049_____    ...                O
    -1.166_____     2.642_____   -2.564_____    ...                O
     1.049_____    -2.564_____    2.647_____    ...                O
    -0.4662____     1.199_____   -1.299_____    ...                O
     0.0833____    -0.2220____    0.2498____    ...                O
```

By changing the format to display 8 figures it is immediately clear (from the number of underline bars displayed) that approximately 4 to 5 figures are lost which means a condition number of between 1.E4 and 1.E5. The condition number can be (numerically, without guarantee) calculated separately:

```
cond(P)                                                            I

ANS =                                                              O
                                                                   O
 63005.45737889                                                    O
```

Finally we will mention an extremely ill-conditioned example. Consider a Hilbert 15x15 matrix. We choose a right hand side such that the solution consists of the components 1..15. For solving the linear system with Hilbert matrix and that right hand side we use the ACRITH linear system solver.

```
n=15; b=(1:n)'; h=hilb(n); lss(h,h*b)                                I

ANS = (15,1)-real-point-matrix                                       O
                                                                     O
    1                                                                O
    2                                                                O
    3                                                                O
    4                                                                O
    5                                                                O
    6                                                                O
    7                                                                O
    8                                                                O
    9                                                                O
   10                                                                O
   11                                                                O
   12                                                                O
   13                                                                O
   14                                                                O
   15                                                                O
```

Using a standard numerical algorithm gives an error message that the linear system is too ill-conditioned.

However, there are cases where a standard numerical algorithm produces drastically wrong results. Consider a lower triangular matrix with all 1's in the diagonal and 2's below the diagonal (cf. [Neu87]):

```
for i=1:20 do for j=1:20 do if i>j  then a(i,j)=2                    I
                         else if i==j then a(i,j)=1                  I
```

Computing the eigenvalues of that matrix (which are all 1 and clearly very ill-conditioned) using EISPACK yields the following result:

```
eig(A)                                                               I

ANS = (20,1)-complex-point-matrix                                    O
                                                                     O
     ( 1.38726248 +0.07525836i)                                      O
     ( 1.38790457 -0.07174266i)                                      O
     ( 1.32675171 +0.20581631i)                                      O
     ( 1.32850272 -0.20283533i)                                      O
     ( 1.22516701 +0.29567067i)                                      O
     ( 1.10782683 +0.33721132i)                                      O
     ( 1.22762994 -0.29351367i)                                      O
     ( 0.99507345 +0.33601092i)                                      O
     ( 1.11063798 -0.33589205i)                                      O
     ( 0.89858074 +0.30299316i)                                      O
     ( 0.99804708 -0.33543569i)                                      O
     ( 0.82305723 +0.24897531i)                                      O
     ( 0.76913374 +0.18256656i)                                      O
     ( 0.90166407 -0.30313694i)                                      O
```

```
( 0.73550008 +0.10996239i)                                    0
( 0.72009005 +0.03525145i)                                    0
( 0.82619613 -0.24994861i)                                    0
( 0.72101034 -0.03942435i)                                    0
( 0.73783840 -0.11319895i)                                    0
( 0.77212545 -0.18458822i)                                    0
```

Using a today's inclusion algorithm will give an error message that no inclusion could be computed because the problem is too ill-conditioned. It would definitely not give a wrong result.

ABACUS has been used in several courses and seminars. It turned out to be extremely useful in a teaching environment as well as a research tool. It should not be used as a production like program. The next version of ABACUS, which will be programmed in C, will cover part of this.

Because of the extremely powerful operators, code written in ABACUS is very short, easy to read and easy to debug. The number of lines of code in a typical numerical application program decreases by an order of magnitude using ABACUS instead of FORTRAN. When developing the code in very short time using ABACUS and then switching to FORTRAN for production code there is the additional advantage that ABACUS is a specification in m thematical notation which is executable, i.e. against which can be tested.

4. References.

[ACR86] ACRITH, High-Accuracy Arithmetic Subroutine Library, Program Description and User's Guide, IBM Publications, Document Number SC33-6164-3 (1986).

[AlHe83] Alefeld, G. and Herzberger, J.: Introduction to Interval Computation, Academic Press, 1983.

[Br87] Hochgenaue Standardfunktionen fuer reelle und komplexe Punkte und Intervalle in beliebigen Gleitpunktrastern, Ph.D., University of Karlsruhe, 1987.

[Kr87] Inverse Standardfunktionen fuer reelle und komplexe Intervallargumente mit a priori Fehlerabschaetzungen fuer beliebige Datenformate, Ph.D., University of Karlsruhe, 1987.

[KuMi81] Kulisch, U. and Miranker, W.L.: Computer Arithmetic in Theory and Practice, ACADEMIC PRESS, New York (1981).

[KuRu87] Kulisch, U. and Rump, S.M.: Rechnerarithmetik und die Behandlung algebraischer Probleme, in Buchberger/Feilmeier/Kratz/Kulisch/Rump: Rechnerorientierte Verfahren, B.G. Teubner (1986).

[MAT86] PC-MATLAB for MS-DOS Personal Computers, Version 2.2, (C. Moler, D. Little, S. Bangert, S. Kleiman), The Math Works, Inc. (1986).

[Mo79] Moore, R.E.: Methods and Applications of Interval Analysis, SIAM Studies in Applied Mathematics, (1979).

[Neu87] Neumaier, A.: Private communication.

[PAS87] Allendoerfer, Boehm, Bohlender, Gruener, Kaucher, Kirchner, Klatte, Kulisch, Neaga, Rall, Rump, Saier, Schindele, Ullrich, Wippermann, Wolff von Gudenberg: PASCAL-SC General Information Manual and User's Guide with Computer Software, Teubner and John Wiley, 1987.

[Ru83] Rump, S.M.: Solving Algebraic Problems with High Accuracy in "A New Approach to Scientific Computation", Edts. U.W. Kulisch and W.L. Miranker, ACADEMIC PRESS, p. 51-120 (1983).

[Ru84] Rump, S.M.: Solution of linear and nonlinear algebraic problems with sharp, guaranteed bounds, COMPUTING Supplementum 5, 23 pages, (1984).

[Ru87] Rump, S.M.: New Results on Verified Inclusions, in "Accurate Scientific Computations", eds. W.L. Miranker and R.A. Toupin, Springer Lecture Notes in Computer Science 235, New York 1987.

[SIE86] ARITHMOS, Benutzerhandbuch, Siemens AG, Bestellnummer U 2900-J-Z87-1, (1986).

AN OVERVIEW OF GLOBAL OPTIMIZATION USING INTERVAL ANALYSIS

Eldon Hansen

Abstract. We give an overview of the general ideas involved in solving global optimization problems using interval analysis. We include a tutorial discussion of a simplified global optimization algorithm for a function of one variable. We discuss common misconceptions about interval analysis and the optimization algorithm.

1. Introduction. A primary goal of this paper is to give an elementary introduction to the ideas involved in using interval analysis to solve global optimization problems. Another goal is to point out misconceptions held by some persons having little knowledge of interval analysis. Hopefully, this will help workers in interval analysis make their papers more understandable by a wider audience.

When possible, explicit comments are cited which exhibit the lack of understanding of interval methods. These comments are by referees of a paper written by the author.

One common misconception about interval analysis is that rounding causes loss of certainty in various contexts. A recurring theme in this paper is an explanation of why this is not so.

2. The fundamental theorem. In the author's opinion, the following theorem should be called the Fundamental Theorem of Interval Analysis. It is what most (all?) of interval analysis is based upon. In particular, it is what makes it possible to solve the global optimization problem with certainty.

Theorm 2.1 (Moore [7]). Let $f(x_1, \ldots, x_n)$ be a rational function of n variables. Consider any sequence of

arithmetic steps which serve to evaluate f with given arguments $x_1, \ldots, x_n$. Suppose we replace the arguments x_i by corresponding intervals X_i $(i = 1, \ldots, n)$ and replace the arithmetic steps in the sequence used to evaluate f by the corresponding interval arithmetic steps. The result will be an interval $f(X_1, \ldots, X_n)$. This interval contains the value of $f(x_1, \ldots, x_n)$ for all $x_i \in X_i$ $(i = 1, \ldots n)$.

As a simple example, consider the function of one variable $f(x) = x(x^2 - 1)$. Suppose we evaluate this function with interval argument $X = [-2, 2]$. We first compute $X^2 = [0, 4]$, next $X^2 - 1 = [-1, 3]$ and, finally, $X(X^2 - 1] = [-6, 6]$. That is, $f([-2, 2]) = [-6, 6]$. Theorem 2.1 assures that $-6 \leq f(x) \leq 6$ for all x $[-2, 2]$.

The true range of $f(x)$ for $x \in X$ is $[-6, 6]$. We have obtained exact bounds on the range of f by a single evaluation of f with argument X. This is despite the fact that f has both a minimum and a maximum in the interior of X.

In general, we will not get sharp bounds on the range. But they are always correct. In fact, we do not get sharp bounds for this problem if we evaluate f as follows. Write f as $x^3 - x$. Compute $X^3 = [-8, 8]$ and then $f(X) = X^3 - X = [-10, 10]$. This interval is not a sharp bound for the range of f; but it does bound the range, as is guaranteed by the Theorem 2.1.

In practice, when rounding is necessary, we round left endpoints downward and right endpoints upward. This "outward rounding" assures that the theorem remains true even when rounding is present.

As a trivial example, consider $f(x) = 5.67x$. The range of f for $x \in [2, 3]$ is $[11.34, 17.01]$. Suppose we evaluate $f([2, 3])$ using three decimal digit outwardly rounded interval arithmetic. We round the left endpoint down to 11.3 and the right endpoint up to 17.1. Our rounded interval $[11.3, 17.1]$ contains the range of f over X.

A referee of a paper by the author commented: "The cost of extending real functions to interval functions is not addressed. If the function is not monotone, it seems that a lot of work is required." But there is no such cost! As we have seen, we merely replace real arithmetic operations by interval arithmetic operations. Of course, interval arithmetic is slower; but that is a separate issue. As we have seen, no monotonicity is required.

It should be noted, however, that monotonicity can be utilized and permits sharp bounds on the range to be obtained. However, it is not necessary, and is seldom used in solving optimization problems.

3. Newton's method. An important use of Theorem 2.1 is in bounding the remainder term in a Taylor expansion. As a simple example, consider

$$g(y) = g(x) + (y - x)g'(t) \qquad (3.1)$$

where g is a rational function which is continuous in a given interval, X. Assume x and y are in the interval X. The mean value theorem assures the existence of a number, t, between x and y such that (3.1) holds. Hence, t is in X.

If y is a zero of g, then $g(y) = 0$. Therefore, from

(3.1),

$$y = x - g(x)/g'(t). \tag{3.2}$$

Theorem 2.1 assures that $g'(t) \in g'(X)$. Hence, from (3.2), any root $y \in X$ is also in

$$N(x, X) = x - g(x)/g'(X). \tag{3.3}$$

When evaluating $g(x)$, and then $N(x, X)$, interval arithmetic is used to bound rounding errors as well as the range of values. As a result, $N(x, X)$ is a bound on any zero of g in X, not just an approximation.

When applying an interval Newton step to an interval, X, we are temporarily interested only in zeros of g which are in X. Hence, the next iterate is chosen to be $X \cap N(x, X)$. If this intersection is empty, then there is no zero of g in X. If $0 \in g'(X)$, this intersection may be composed of two intervals. When this is the case, we store one interval for later processing and iterate on the other until convergence. See [4] for details.

Note also, that $N(x, X)$ is a bound on <u>every</u> zero of f in X. This fact makes it a straightforward task to find <u>all</u> zeros of a function in a given interval. Equation (3.3) is the basis of the interval Newton method. The algorithm, itself, subdivides the initial interval and separately finds and bounds all the zeros in the initial interval. For details, see [4].

If any point of the initial interval is discarded by the algorithm, then it is guaranteed not to be a zero of g. Rounding errors merely cause fewer points to be discarded but do not invalidate this statement. When the algorithm terminates (as described in Section 10), each zero of g is bounded

by a small interval. At this stage, the entire initial interval has been exhaustively searched. No zero of g is missed. These properties carry over to the interval Newton method for the multidimensional case.

A referee of a paper by the author remarked that the interval Newton method has to be expensive because it is "tantamount to exhaustive search". The point is that it <u>is</u> an exhaustive search. Its virtue is that it does an exhaustive search while taking about the same number of algorithmic steps as the non-interval Newton method when the latter works well. If the non-interval method cycles or converges poorly for other reasons, computational experience shows that the interval method generally takes fewer iterations.

The multidimensional interval Newton method is as follows: Let J denote the Jacobian of g. Given an initial box (interval vector) $X^{(0)}$, solve

$$g(x^{(i)}) + J(X^{(i)})(y - x^{(i)}) = 0 \quad (i = 0, 1, 2, \ldots) \quad (3.4)$$

for a box Y containing the set of solutions y. (Methods for doing this are described, for example, in [4]). The new box is obtained as $X^{(i+1)} = X^{(i)} \cap Y$. Let $m(X)$ denote the center of a box, X. In (3.4), we generally choose $x^{(i)} = m(X^{(i)})$.

In an interval Newton algorithm, no subset of the initial box is deleted unless the algorithm has guaranteed that no zero of g is deleted. This guarantee takes rounding into account. At termination, any zero of g must be in the final set of boxes. Any final box will be small (as prescribed by the termination condition described in Section 10). Therefore, every zero of g is isolated in a small region;

although more than one zero may reside in a single box. It is possible that a final box may not contain any zero. Stricter convergence criteria would be necessary in this case to provide greater certainty of the precise location of zeros. In any case, all zeros (if any) reside in the final box(es).

Persons unversed in interval Newton methodology seem to have difficulty understanding this fact that it is always true that all zeros in the initial box are found.

4. Existence. Multidimensional interval Newton methods differ in how they compute the box Y bounding the solution set of (3.4). For most published methods it has been proved that the following theorem holds (e.g., see [8]).

Theorem 4.1. If $Y^{(i)}$ is in the interior of $X^{(i)}$, there exists a zero of g in $X^{(i)}$.

Many papers have been published which discuss this theorem. However, a referee recently remarked: "The claim that the interval Newton method will prove existence of a solution seems a bit strong and unjustified." Perhaps this statement was made because the referee did not realize that the hypothesis of Theorem 4.1 is so often satisfied in practice at some stage of the iteration. In fact, however, it is unusual not to obtain proof of existence in practice when solving for a simple zero. The important point is that further educating is needed if interval methods are to be more widely understood and used.

5. Introductory remarks on optimization. One way of viewing an interval Newton method is that, rather than directly seek a solution, it deletes that part of the original box

which cannot contain a solution. It proceeds until only a small box (or set of boxes) remains which, therefore, must contain the solution(s). The global minimization method also proceeds in this way. It is applied in a fail-safe way so that, despite rounding, no solution is ever deleted. For optimization, additional means are available for deleting sub-boxes of the initial box.

For the unconstrained optimization problem in which we wish to minimize a function, f, a sub-box, X, is deleted if

(1) an interval Newton method proves there is no stationary point in X, or

(2) the gradient g of f is not zero in X, or

(3) f is not convex anywhere in X, or

(4) throughout X, the function f is greater than a known value, $\overline{f}$, of f.

For the constrained minimization problem, a sub-box, X, is deleted if

(1) an interval Newton method proves that the Fritz John conditions are not satisfied anywhere in X, or

(2) every point in X is infeasible, or

(3) for every point in X, the function, f, is greater than a known value, $\overline{f}$, of f at a point known to be feasible.

For all these processes, outward rounding is used and the sub-box, X, is deleted only if it is certain that it does not contain the global minimum.

We have already discussed the interval Newton method. In what follows, we describe how the other procedures for

deleting sub-boxes can be implemented. We describe simplified procedures easy to understand. More sophisticated versions are used in practice for most of them. For discussions of the better procedures, see [3] for the unconstrained problem, [5] for the inequality constrained problem, and [6] for the equality constrained problem.

6. Monotonicity. Suppose we evaluate the gradient g of f over a box, X. Let the result for the i-th component of g be

$$g_i(X) = [g_i^L(X), g_i^R(X)] \qquad (i = 1, \ldots, n).$$

By Theorem 2.1, if $g_i^L(X) > 0$, then $g_i(x) > 0$ for all $x \in X$. If $g_i^R(X) < 0$, then $g_i(x) < 0$ for all $x \in X$. In either case, X cannot contain a stationary point of f. If the computed interval is wider than the exact one, we still draw the correct conclusion that X does not contain a stationary point of f if the computed interval does not contain zero.

7. Concavity. In the one-dimensional case, suppose $f''(x) < 0$ for all x in some interval, X. Then f is concave in X and f cannot have a minimum in the interior of X. Suppose we seek a minimum in some initial interval containing X. Suppose we evaluate $f''(X)$ and obtain the interval $[f_L''(X), f_R''(X)]$. If $f_R''(X) < 0$, then by Theorem 2.1, $f''(x) < 0$ for all $x \in X$ and we can delete X. If outward rounding is used in evaluating $f''(X)$, then the correct value of $f_R''(X)$ is negative if its computed value is negative. Hence, the correct decision will be made about whether f is concave in X.

In the multidimensional case, the corresponding condition is that the Hessian, $H(x)$, of $f(x)$ be positive semi-definite at an unconstrained minimum. To be positive semi-definite, it is necessary that the diagonal elements $H_{ii}(x)$ $(i = 1, \ldots, n)$ be nonnegative. (We ignore the other necessary conditions since they are harder to check.) If $H_{ii}(X) < 0$ for a given box, X, then, by Theorem 2.1, $H_{ii}(x) < 0$ for all $x \in X$. Hence, the interior of X cannot contain a minimum of f. Outward rounding permits this condition to be used without loss of certainty.

8. Deletion of boxes where f is large. As an interval algorithm for global minimization proceeds, it dynamically subdivides the initial box (See [3]; also, see step 3 of the algorithm in Section 11). As each new sub-box is generated, the value of f at its center is computed. Let x be such a point. When evaluating $f(x)$ we obtain, say, $[f^L(x), f^R(x)]$. Because outward rounding is used, $f^R(x) \geq f(x)$. In the unconstrained case, therefore, $f^R(x)$ is an upper bound for the globally minimum value, f^*, of f.

Let $\overline{f}$ denote the smallest of such upper bounds for the points previously sampled at some stage of the solution process. Let X be some sub-box of the original box. If $f(X) > \overline{f}$, then, by Theorem 2.1, $f(x) > \overline{f}$ for all $x \in X$ and, hence, X cannot contain the global minimum.

When we compute $f(X)$, we obtain, say, $[f^L(X), f^R(X)]$. Because of outward rounding, $f^L(X) \leq f(X)$. Hence, if $f^L(X) > f$, then $f(X) > \overline{f}$. That is, despite rounding, we know the global minimum is not in X.

9. Feasibility. Consider the inequality constrained problem where conditions $p_i(x) \leq 0$ $(i = 1, \ldots, m)$ are imposed. Suppose, for some sub-box, X, we compute $p_i(X)$ $(i = 1, \ldots, m)$ and, using outward rounding, obtain $[p_i^L(X), p_i^R(X)]$. If $p_i^L(X) > 0$, then $p_i(X) > 0$. Therefore, by Theorem 2.1, $p_i(x) > 0$ for all $x \in X$. That is, x is infeasible for every $x \in X$. In particular, X cannot contain the global solution.

For the equality constrained problem, we have conditions $q_i(x) = 0$ $(i = 1, \ldots, r)$. In the same way, if $q_i(X) > 0$ or $q_i(X) < 0$, then, by Theorem 2.1, $q_i(x) \neq 0$ for any $x \in X$. That is, there is no feasible point in X.

10. Termination. The algorithm proceeds until all remaining boxes are small according to a given tolerance. It then enters a phase which assures that the range of f is small over any final box. We now describe this phase.

Consider the one dimensional case. For any remaining interval, X, we evaluate $f(X)$ getting $[f^L(X), f^R(X)]$ and $w[f(X)] = f^R(X) - f^L(X)$. If $w[f(X)] < \varepsilon_F$ for a given tolerance ε_F, we stop.

Note that $\overline{f}$ is the smallest value of f at the center of any generated interval, including the final ones. For a final interval, X, we have $f^L(X) \leq \overline{f}$ since, otherwise, X will have been deleted. Let $\underline{f}$ denote the smallest value of $f^L(X)$ for the remaining intervals, X. Then $\underline{f} \leq f^* \leq \overline{f}$ and $\overline{f} - \underline{f} \leq \varepsilon_F$.

The process is similar for the multidimensional case.

11. A global optimization algorithm. We now give a

simplified algorithm for solving unconstrained minimization problems in one dimension. For a more sophisticated algorithm, see [2]. The purpose of the algorithm is to illustrate, in a simple way, the ideas expressed above. In Section 12, we apply the algorithm to a simple example and trace the progress. As we shall see, the algorithm works rather well despite its simplicity. For numerical results for a comparable, but more sophisticated, algorithm in the multidimensional case, see [9].

Let the objective function, f, and the initial interval, $X_0 = [a, b]$, be given. Assume a box size tolerance, ε_X, and a range width tolerance, ε_F, are specified. The steps of the algorithm are as follows:

To initialize, put X_0 in a list, L_1. Evaluate $f(X_0)$ getting $[f^L(X_0), f^R(X_0)]$.

1. Evaluate $f(a)$ getting $[f^L(a), f^R(a)]$ and evaluate $f(b)$ getting $[f^L(b), f^R(b)]$. Set $\overline{f} = \min\{f^R(a), f^R(b)\}$.

2. Denote the number of intervals in list L_1 by s. If L_1 is empty, then $s = 0$. If $s = 0$, go to step 13. Denote the intervals (if any) in L_1 by X_i $(i = 1, \ldots, s)$. Choose the current interval to be $X = X_j$ $(j = 1, \ldots, s)$ if $f^L(X_j) \leq f^L(X_i)$ for all $i = 1, \ldots, s$. Delete X from L_1.

3. Set $x = m(X)$. Evaluate $f(x)$ getting $[f^L(x), f^R(x)]$.

4. If $f^R(x) < \overline{f}$, set $\overline{f} = f^R(x)$.

5. Evaluate $f'(X)$. If $0 \notin f'(X)$, go to step 2. (See Section 6.)

6. Evaluate $f''(X)$. If $f''(X) < 0$, go to step 2. (See Section 7.)

7. Do one step of the interval Newton method (See Section 3)

applied to the interval X. Denote the result by Y. If Y is the empty set, go to step 2.

8. If Y is a single interval, go to step 9. Otherwise, go to step 11.

9. Evaluate $f(Y)$ getting $[f^L(Y), f^R(Y)]$. If $f^L(Y) > \overline{f}$, go to step 2. (See Section 8.)

10. Let $w(Y)$ denote the width of Y. If $w(Y) < \varepsilon_X$, put Y in list L_2. Otherwise, put Y in list L_1. In either case, go to step 2.

11. Y is the union of two intervals. Denote them by Y_1 and Y_2. For $i = 1, 2$, delete Y_i if $f^L(Y_i) > \overline{f}$.

12. If Y_i $(i = 1, 2)$ was not deleted in step 11, put Y_i in list L_2 if $w(Y_i) < \varepsilon_X$ and put Y_i in list L_1, otherwise. Go to step 2.

13. Let r denote the number of intervals in list L_2. Denote the intervals by $X_1, \ldots, X_r$. For each i $(i = 1, \ldots, r)$, (if any) such that $f^R(X_i) - f^L(X_i) > \varepsilon_F$, put X_i in list L_1 and go to step 2.

14. Delete any interval X_i in L_2 for which $f^L(X_i) > \overline{f}$. (See Section 8.)

15. Stop.

For each remaining interval X_i, we have $w(X_i) < \varepsilon_X$ and $f(x) \leq f^* + \varepsilon_F$ for all $x \in X$. Also, we have $\overline{f} - \varepsilon_F \leq f^* \leq \overline{f}$.

Note that in the steps of the algorithm, interval arithmetic with outward rounding is used in any computation. The rounding may prevent a sub-box from being deleted when exact interval arithmetic would allow deletion. But, rounding never causes deletion of the global minimum.

12. An example. We now apply the algorithm just described to the function

$$f(x) = 24x^4 - 142x^3 + 303x^2 - 276x + 93.$$

The global minimum of this function is $f^* = 1$ at $x^* = 2$. There is a local minimum at $x = 1$ and a maximum at $x = 1.4375$.

For the initial interval, we choose $X = [0, 3]$. This interval contains the maximum and both minima.

Figures 1 and 2 show the progress of the algorithm. The intervals remaining (stored in lists L_1 and L_2) at each stage are shown. Note that only one interval is processed during each iteration. The largest number of intervals stored at any stage was ten. Small intervals and small gaps between them are sometimes exaggerated in length to make them visible. The function, f, which is a lower case letter in the text is shown as the capital letter, F, in the figure. About eleven significant decimal digits were used in the computations. However, fewer digits are given in the figure.

Unless otherwise indicated in the figure, each change in the remaining set of intervals is due to an application of the interval Newton method to a selected interval. Each improvement in the upper bound, $\overline{f}$, is shown at the stage in which it was obtained.

As indicated in the figure, after 34 iterations, only the interval $X^* = [1.999999998, 2.000000002]$ remains. Since the global minimum could not have been deleted, it must be in this interval. We compute $f(X^*) = [0.9999987778, 1.000001225]$. This interval must contain f^*. But the algorithm obtained the upper bound $\overline{f} = f(2.000001) = 1.000000062$.

Hence, $0.9999987778 \leq f^* \leq 1.000000062$.

Note that at step 24, the current interval, X, contains the local minimum. But $f(X) > \overline{f}$ so X is eliminated. At steps 26, 27, 28, and 32, the objective function f is shown to be concave over the current interval and the interval is deleted. Otherwise, any deletion of a subinterval is by the interval Newton method.

The simplified procedure described herein requires 34 steps to solve this example to the prescribed accuracy. Our actual program, using more sophisticated versions of these procedures, requires only 10 steps.

13. Other examples. In the above example, the initial interval was rather small. A natural question to ask is how the algorithm would perform if the initial interval were large. For the above example, very little increase in run time occurs even when a huge increase is made in the initial interval. We show why below. In [9], various multidimensional problems were run with both large and small initial boxes. In all cases, when the box was greatly increased, the runtime increased by only a small factor.

For the above example, let X be any nonpositive interval. Then we would compute $f'(X) < 0$. This assures there is no stationary point in X so X can be deleted. For example, suppose the initial interval is $X = [-10^{30}, 10^{30}]$ and the first application of the interval Newton method deleted a tiny gap about $x = 0$. We now have (approximately) the two intervals $X = [-10^{30}, 0]$ and $Y = [0, 10^{30}]$. Computing $f'(X)$ yields $[-0.9603 \times 10^{92}, -278]$. Assuming no

overflow, we thus find $f'(X) < 0$ and, hence, X cannot contain a stationary point of f. We thus delete X, which has a width of 10^{30}.

Similarly, if $X > 426/96 = 4.44$, we would compute $f'(X) > 0$. Thus, any large positive interval would be rapidly reduced to a relatively small one.

We now consider a second example which illustrates how it is possible to quickly find the global minimum of a function with many local minima. Consider

$$f(x) = x^2(2 + \sin \pi x).$$

It is easily shown that the derivative, $f'(x)$, of this function has a zero in the interval $[n, n + 1]$ for all $n = \pm 2, \pm 3, \ldots$. To see this, note that $f'(0) = 0$, $f'(\pm n) > 0$ for n even and nonzero, and $f'(\pm n) < 0$ for n odd and $n \geq 3$. Also, $f'(x) = 0$ for $x = 1.118$, approximately. Thus, f' has at least $2n + 1$ zeros in the interval $[-n, n]$ for all $n = 1, 2, 3, \ldots$.

Suppose the initial interval is $X = [-10^{30}, 10^{30}]$ so that there are $2x10^{30} - 1$ extrema in X. Suppose, also, that we have sampled the value of f at $x = 1$. Since $f(1) = 2$, we know that $f^* \leq 2$. Consider the interval $X = [2, 10^{30}]$. We compute $f(X) = [4, 3x10^{60}]$. That is, $f(x) \geq 4$ for all $x \in X$. In particular, $f(x) > \overline{f} = 2$ for all $x \in X$. Thus, the global minimum is not in X. With one interval evaluation of f, we have shown that the global minimum is not in the interval X. In so doing, we have proved that $10^{30} - 2$ local extrema are not global minima.

14. Conclusion. As stated above, an interval global optimization algorithm isolates the global minimum by

deleting sub-boxes of the original box which cannot contain the solution. Our theme in this paper has been that none of the procedures for deleting sub-boxes ever deletes the global solution even though rounding is present. This is also true for the more sophisticated procedures used in practice.

References

[1] E. R. Hansen, "A globally convergent interval method for computing and bounding real roots", BIT, 18(1978), 415-424.

[2] E. R. Hansen, "Global optimization using interval analysis - the one-dimensional case", Jour. Optim. Theory Applic., 29(1979), 331-334.

[3] E. R. Hansen, "Global optimization using interval analysis - the multidimensional case", Numer. Math., 34(1980), 247 - 280.

[4] E. R. Hansen and R. I. Greenberg, "An interval Newton method", Applied Math. Comput., 12(1983), 89-98.

[5] E. R. Hansen and S. Sengupta, "Global constrained optimization using interval analysis", pages 25-47 of "Interval mathematics 1980" (edited by K. Nickel), Academic Press, 1980.

[6] E. R. Hansen and G. W. Walster, "Nonlinear equations and optimization", submitted for publication in Math. Prog.

[7] R. E. Moore, "Interval analysis", Prentice-Hall, Englewood Cliffs, N. J., 1966.

[8] A. Neumaier, "Interval iteration for zeros of systems of equations", BIT, 25(1985), 256-273.

[9] G. W. Walster, E. R. Hansen, and S. Sengupta, "Test results for a global optimization algorithm", pages 272-287

of "Numerical optimization 1984" (edited by P. T. Boggs, R. H. Byrd, and R. B. Schnabel), SIAM Publ., 1985.