Eldon Hansen

ARO-D Report 67-3

PROCEEDINGS OF THE 1967 ARMY NUMERICAL ANALYSIS CONFERENCE

Sponsored by

The Army Mathematics Steering Committee

on Behalf of

THE OFFICE OF THE CHIEF OF RESEARCH AND DEVELOPMENT

PROGRAMMING INTERVAL ARITHMETIC AND APPLICATIONS

Allen Reiter
Lockheed Missiles and Space Company
Palo Alto, California

INTRODUCTION. This paper discusses the current state-of-the-art in interval arithmetic, both from the programming point of view and from the point of view of applications to date.

Interval arithmetic was first developed formally by R.E. Moore around 1960, although there is essentially nothing new in the concepts involved. Moore originally envisioned interval arithmetic as a means of completely rigorous automatic error control for computational processes using a digital computer. More recently, people have begun to appreciate the potential of interval arithmetic for control theory, and also as a tool in experimental designing on-line, with both a man and a computer as parts of the feedback loop.

There are basically three different sources of error associated with numerical computations. The first, which we may call the data problem, is due to the fact that the value of some given parameter may not be known exactly (this is for example true for physically-determined parameter values), or else may not be exactly represented in a computer (for example, the number π). A second type of error, usually called truncation error, is caused by the necessity to terminate after a finite number of steps some infinite converging process, or (equivalently) by the requirement that some well-defined expression be evaluated at some point whose location is known only approximately (for example, the remainder term of the Taylor series with remainder). The third type of error is round-off error, caused by the necessity to restrict computational processes to operate on numbers which do not exceed some predetermined number of digits in length. Round-off error has traditionally been the most troublesome, primarily because of its non-analyticity. Attempts at rigorous "pencil-and-paper" bounding of round-off either are too difficult or lead to hopelessly pessimistic "bounds".

Interval arithmetic keeps track of the accumulation of error by continually producing an interval, guaranteed to contain the "true" result, and performing the indicated arithmetic operations on the entire interval. Since the implementation of interval arithmetic necessarily involves ordinary arithmetic operations on the end-points of the interval, which in turn involve rounding, care must be taken to perform the rounding properly: "down" for the left-hand end point, and "up" at the right-hand one. Thus, when in the sequel we shall speak of interval arithmetic, it shall be understood that in the implementation of the operations on a computer rounded interval arithmetic is used. However, in the formal discussion of interval arithmetic we shall ignore this fact, and define the formal operations independently of their implementation.

ARITHMETIC RULES. An interval is simply a closed interval on the real line, of the form $[a,b]$. We can also think of an interval as a fuzzy number x of the form $[x-\varepsilon , x+\varepsilon]$; although ε is certainly not restricted to being small in any sense. The arithmetic operations are defined in a natural fashion, and in fact reduce to ordinary arithmetic when $\varepsilon=0$. (When the occasion arises, we shall speak of ordinary real numbers as degenerate intervals.)

Elementary operations are defined as follows. Let $[a,b]$ and $[c,d]$ be a pair of intervals. Then

$$[a,b] + [c,d] = [a+c,b+d] ;$$
$$[a,b] - [c,d] = [a-d,b-c] ;$$
$$[a,b] * [c,d] = [\min(ac,ad,bc,bd), \max(ac,ad,bc,bd)] ;$$
$$[a,b] / [c,d] = [a,b] *[1/d,1/c];$$ (division is defined only if the interval $[c,d]$ does not contain the point zero).

It can be seen that these operations are defined in such a way that the result is precisely the set of all possible values of the operation as the operands range over the argument intervals.

Interval arithmetic is associative, and addition and multiplication are commutative. Unfortunately, the distributive law does not hold; instead we have the "subdistributive" law (I, J, and K being intervals):

$$I * (J + K) \subset I * J + I * K.$$

That the inclusion can indeed be proper can be seen from the example

$$[-3,3] * [0,2] + [-3,3] * [-1,0] = [-6,6] + [-3,3] = [-9,9] ,$$

whereas

$$[-3,3] * ([0,2] + [-1,0]) = [-3,3]* [-1,2] = [-6,6] .$$

The example also illustrates that a given interval number may have many multiplicative units: if y is any real number in $(-1,1)$, then all interval numbers of the form $[-1,y]$ or of the form $[y,1]$ are multiplicative units for the interval number $[-3,3]$.

More disruptive is the fact that although an additive unit is unique ($[0,0]$), interval numbers do not in general possess additive inverses. (This reflects the fact that once uncertainty or error has been introduced into a computational process, it cannot be cancelled out, but must be carried along till the end.) This last property is responsible for almost all of the difficulties in interval arithmetic, and frequently necessitates very delicate handling of the specification of a computational algorithm - something that the current state-of-the-art is not quite up to. (In spite of this handicap, useful areas of application have already been found.)

The usefulness of interval arithmetic for error bounding comes from the fact that

1) The elementary arithmetic operations are continuous mappings from $I_1 \times I_2$ onto I_3 (the I's are arbitrary intervals);

2) Since the elementary operations are defined in such a manner that the range of the operator as the operands range over the argument intervals is contained in the result interval, the same is true for any well-defined grouping of such operations on argument intervals; in other words, for all rational functions. Of course, rational operations is all computers are capable of executing; thus, any computable function can be bounded by the use of interval arithmetic.

Let $f(x_1,\ldots x_n)$ be a given <u>formal</u> rational function in the indeterminates $x_1,\ldots x_n$. When the indeterminates take on real values, f denotes a real-valued function. There may be many different ways of representing this function, which are all algebraically equivalent; we will fix a representation $f_1(x_1,\ldots x_n)$. If we let the indeterminates take on interval values $X_1,\ldots X_n$, then the function f_1 is still well-defined (we can regard f_1 as a computer program, with a sequence of arithmetic operations to be carried out in a certain order); we however choose to call this interval-valued function $F_1(X_1,\ldots X_n)$. Note that the fact that f_1 and f_2 may be algebraically equivalent to f (and to each other) certainly does not imply that F_1 and F_2 are equivalent (this is primarily due to the failure of the cancellation law for interval arithmetic). The basic theorem of interval arithmetic however states that for the purposes of error bounding any representation will do:

<u>Theorem</u>. Let f be a given rational function, $f = f(x_1,\ldots x_n)$, and let F be any representation of f, F to be evaluated in interval arithmetic. Let $X_1,\ldots X_n$ be a collection of closed intervals on the real line. Then the range of f as each variable x_i ranges over X_i is contained in $F(X_1,\ldots X_n)$.

The theorem assures us that interval arithmetic is sufficient to compute bounds on the range of a rational function over a compact rectangle in E_n. Note that since the evaluation of F can be done using <u>rounded</u> interval arithmetic, the round-off error is included in the final bounds produced by F. (It is worth while stressing though that nothing is said about bounding the round-off that might occur in evaluating f. The round-off process is not a continuous operation. On some computers, in particular on the IBM SYSTEM/360, it is easy to cook up examples where f evaluated at some point p inside the rectangle turns out to be outside the interval obtained by evaluating F. This is but another aspect of "dirty" floating-point hardware. The <u>true</u> range of f is however always contained in F.)

As already noted, the width of the interval obtained by evaluating

F may be considerably greater than the width of the true range of f; it is also generally quite sensitive to the choice for the particular representation F. This shall be discussed below.

SOME APPLICATIONS OF INTERVAL ARITHMETIC. Aside from the obvious advantage of providing error bounds, interval arithmetic can be used by a computer to control the growth of error. While potentially the realm of applications is unlimited, the author knows only of the following contexts in which interval arithmetic has been studied:

a) The initial-value problem for ordinary differential equations;
b) Finding roots of polynomials;
c) Matrix inversion, and the eigen-value problem for matrices;
d) Solution of systems of simultaneous (non-linear) equations;
e) The two-point boundary-value problem.

In these areas, analytic techniques are being developed which make use of interval arithmetic evaluations, and which also address themselves to the peculiar problems which arise in using interval arithmetic.

THE INITIAL VALUE PROBLEM FOR ORDINARY DIFFERENTIAL EQUATIONS. Let $dy/dx = f(x,y)$ denote a system of n first-order ordinary differential equations, and let $y_0 = y(x_0)$ be given. The application of interval arithmetic to the automatic generation of solutions to this problem was the first application suggested by Moore. He designed a computer program using interval arithmetic which gave solutions with automatic error bounds.

His method is described in [7]. Briefly stated, the solution is expanded in a Taylor series with remainder (up to a specified number of terms) at a given point. To bound the remainder term, the required derivative is evaluated over a whole rectangle (using interval arithmetic) which is guaranteed to contain the point at which the derivative should be evaluated. Iterative procedures can be specified which limit the growth of the width of the resulting interval.

Since this method depends on the ability of the computer to evaluate higher-order derivatives of f, it is handy to have a computer program which can do analytic differentiation. Such computer programs have indeed been written, either tailored for the purpose at hand [9], or in more general settings, such as the FORMAC capability for the FORTRAN IV compiler on the IBM 7094.

The success of interval arithmetic in this setting is somewhat difficult to evaluate. The problem is that for reasonably complex systems of equations and for long ranges of integration with respect to the independent variable, the resulting interval tends to be too wide to be of much practical value. Attempts at elaborate transformations to reduce the error growth due to the remainder term evaluation being too crude have in general been defeated by the fact that the structure of interval arithmetic (lack of additive inverses) causes growth of widths of intervals due to too many operations. Also, on some computers (such as the CDC 1604) the floating-point hardware structure of the computer is so unfriendly that interval arithmetic operations are rather time-consuming. For short integrations, and for qualitative estimates, interval arithmetic may be very valuable.

ROOTS OF POLYNOMIALS. Moore suggested that a simple procedure for localizing zeroes of rational functions can be developed using interval arithmetic. Such a procedure was indeed programmed [3]. The method is based on the simple fact that if P is given rational form in n variables, R a rectangle in E^n, and P(R) evaluated in interval arithmetic does not contain the point 0, then P (as a function of real variables) cannot possibly have any zeroes in R.

An iterative procedure can be implemented based on the fact that if R_1 and R_2 are two rectangles in E^n each of which contains a given zero of P, then their intersection must necessarily also contain that zero. Thus, an extension of Newton's method is possible, as long as care is taken at each iteration to intersect the new interval (which may not be contained in the one obtained at the previous iteration) with the old one, thus guarding against divergence. This is called by Moore "the method of interval contractions". Clearly any such procedure must converge, but the limit will in general be an interval, rather than a point. If the limit interval is too wide, the process may be repeated by subdividing the original rectangle R into smaller ones.

Similar results were obtained for the complex domain (Boche [2] having extended the concept of interval arithmetic to the complex plane) by Hansen [6] and Bennett [1] .

For this problem, interval arithmetic may well be the best (computationally speaking) method of obtaining results, especially if it is desirable to find regions guaranteed not to contain any zeroes of some given function.

MATRIX INVERSION AND THE EIGENVALUE PROBLEM. The problem of inverting matrices in the context of interval arithmetic comes from two distinct sources. Problem one: given a matrix with real elements, obtain a (real) inverse with automatic error bounding of round-off. Problem two: given a method of obtaining solutions of some problem in ordinary arithmetic (for example, Newton's method in n variables) which calls for inverting matrices, extend this method to the case where interval arithmetic will be used for the solution (possibly because the coefficients are only approximately known). That is, in problem two we are asked to invert a matrix with interval elements.

Since it is not a priori clear what we mean by an "inverse" of an interval-valued matrix, we define this inverse to be the set of inverses of all of the real matrices contained in the given interval matrix. It is understood that the inverse is defined only if the interval matrix does not contain any singular real matrices.

Hansen ([4] and[5]) has worked extensively on this problem. He shows that a direct extension of the standard methods for matrix inversion (such as modifications of Gauss - Seidel) to interval arithmetic is not very useful, because of the many arithmetic operations involved, and (again) because of the lack of additive inverses. Instead, he develops several methods, all based on essentially the same principle. What he does is to compute an (approximate) real inverse of the real center of the interval matrix, and then (using some iterative procedure) compute in interval arithmetic bounds for the width of each element of the true inverse of the interval matrix. The variations in the iterative procedures consist of trying to represent things in such a way as to have as many terms as possible be non-interval.

Similar considerations apply to the problem of finding eigenvalues and associated eigenvectors of real-valued or interval-valued matrices. Again, direct extensions of the standard techniques used for real arithmetic are not satisfactory. Hansen [6] suggests iterative procedures using interval arithmetic once approximate solutions are obtained using real arithmetic.

The numerical results quoted by Hansen suggest that very good accuracy can be obtained using interval arithmetic. His methods do converge, although he does not discuss the rate of convergence. Note that in Hansen's methods it frequently pays to carry out the real computations involved using extended-precision arithmetic, since in general multiple-precision arithmetic is much faster than the interval arithmetic procedures required, and it is worthwhile to go to great lengths to save an iterative step.

<u>SYSTEMS OF SIMULTANEOUS EQUATIONS</u>. Let $f(x)$ denote the set of n rational forms $f_i(x)$ in the n formal variables x_j, and let it be desirable to find a solution to $f(x) = 0$ in the vicinity of some point x_0 in E^n. A method proposed by Moore goes as follows.

Let y be a solution near x_0; i.e. let $f(y) = 0$. (Of course, we do not know y explicitly.) Expanding $f(x)$ as a Taylor series with remainder about y, we have $f(x_0) = f(y) + (x_0 - y)J(z)$, where z is some point "between" x_0 and y, and J is the Jacobian matrix evaluated at z. Expressing z as $y + \theta(x_0 - y)$, where θ is a vector with elements between 0 and 1, it can be seen that if R is a rectangle which contains both x_0 and y, then R also contains z. Hence, we can try to solve $(x_0-y)J(R) = f(x_0)$ for y. This will yield a new rectangle R' which contains y, and which can then be intersected with R to yield a (hopefully) smaller rectangle R". We now solve $(R''-y)J(R'')$ for y, etc; this will eventually converge to (we hope) a small interval containing the real solution y.

Hansen gives a slight improvement in the method [6]; this is essentially a slightly better way of writing things down for computation.

It can be seen that this is a variant of Newton's method, adapted for interval arithmetic. It requires that f contain no other zeroes near the point in question, for otherwise the Jacobian J becomes singular. Again it pays to obtain as precise an initial guess as possible, using ordinary (possibly extended-precision) arithmetic.

The author knows of no numerical experimentation with solving large systems of equations using interval arithmetic.

THE TWO-POINT BOUNDARY-VALUE PROBLEM. This problem is currently under investigation by Hansen. He has devised a general method for tackling the solution of

$$y^{(n)} = f(x,y,\dots y^{(n-1)})$$

with a total of n conditions prescribed at the end points $x = 0$ and $x = 1$. His method, based on an adaptation of a finite-difference method, gives sharp bounds at the mesh points and less sharp bounds throughout the interval. It will be described in a forthcoming paper.

OTHER POSSIBLE APPLICATIONS. Interval arithmetic may have potentially many uses. It has been suggested that it can be used in control theory, where it is desirable to let parameters in differential equations range over certain restricted domains. Another potential area of utility is in design, where it can be used in conjunction with an on-line computer system. A designer, seated in front of a terminal in communication with a computer, can experiment with various possible designs by letting some variables range over a set of interval values. With instant feedback from the computer, the designer can begin to get a feel for the effects of perturbations in the design parameters. Using interval arithmetic in this setting is particularly attractive because sharp bounds are not required - the qualitative estimates would be produced in relatively little time, and would at the same time be completely rigorous, covering all possible cases.

THE REPRESENTATION PROBLEM. The major trouble with interval arithmetic is that due to the lack of inverses forms normally considered algebraically equivalent are computationally quite different. It is always advisable when using interval arithmetic to eliminate entirely expressions of the form $x - x$. Other reductions of this type suggest themselves.

The general problem can be stated as follows. Suppose that f is a given function (from E^n into the reals) and suppose that it is desired to obtain bounds on the range of values of f over some rectangle R using interval arithmetic. What is the "best" way of representing f from the point of view of obtaining the narrowest bound?

There are three different approaches to this problem. One can try to obtain an optimal representation for f. (The author strongly suspects that this approach is not in general workable; that is, given a general function f, there is no algorithmic procedure that would allow the selection of a "best" form.) A second approach can be based on the following: if

f_1 and f_2 are two different representations for f, and $f_1(R) = I_1$, $f_2(R)=I_2$, then $I_1 \wedge I_2$ also contains the range of f over R. It may be possible by judiciously choosing among different representations for f to obtain successively better approximations to the range of f. Although there is probably no algorithmic procedure guaranteed to converge for an arbitrary function f, it may be possible to find some programmable heuristics which greatly reduce growth of interval widths. The third approach consists of subdividing the original rectangle R into smaller rectangles and performing the required evaluations on each of the small pieces. This process will generally result in narrower bounds, and is in fact guaranteed to converge to the exact range of f regardless of the representation chosen. The convergence is however so slow compared to the overhead for repeating the computations for each one of the smaller intervals that this approach is not very practical.

Moore has noticed that a certain representation, which he calls the centered form, will frequently yield good results. Briefly, this scheme goes as follows: Given a formal function f of (say) one variable x, and assuming that we are interested in evaluating f over the interval $[a,b]= [m - \frac{1}{2}(b-a), m + \frac{1}{2}(b-a)]$, we represent f as expanded about the midpoint m. That is, we obtain a form g by the relation $g(x-m) = f(x) - f(m)$, so that $g(y) = f(y+m) - f(m)$. g has to be represented in the most "economical" way possible, so that the number of occurences of the term y cannot further be reduced. Since $f([a,b]) = g([-\frac{1}{2}(b-a), \frac{1}{2}(b-a)])$ we have moved the required interval evaluation to be centered about zero.

For an example, let $f(x) = x - x^2$, and let the interval in question be $[0,1]$. The actual range of values of f is of course $[0,\frac{1}{4}]$. Evaluation of f as written yields $[0,1] - [0,1] * [0,1] = [0,1]-[0,1]= [-1,1]$. Writing f in "nested" form as $x*(1-x)$ yields $[0,1] * [0,1] = [0,1]$; an improvement, but still not very good. Writing f in centered form, we have (with $y = x-\frac{1}{2}$) $g(y) = -y^2 + \frac{1}{4}$, so that f(x) is represented as $-(x-\frac{1}{2})^2 + \frac{1}{4}$; interval evaluation of this form yields $- [-\frac{1}{2}.\frac{1}{2}] * [-\frac{1}{2},\frac{1}{2}] + \frac{1}{4} = [0,\frac{1}{2}]$. This turns out the best that can be done for any given representation with the evaluation of only one interval. If however we are willing to evaluate separately the range of f on $[0,\frac{1}{2}]$ and also on $[\frac{1}{2},1]$, then by using the centered form it turns out that we can bound the range of f by $[0,3/8]$. In fact, if we keep halving the width of the (equal) intervals, it can be shown that interval evaluations approach the upper bound $\frac{1}{4}$ linearly with the width.

Lest the reader conclude that the centered form is always the best representation, consider the function $f(x) = x + x^2$, and let the interval in question be $[2-s,2+s]$ where $0 \leq s \leq 2$. Then both straightforward interval evaluation and the nested form give $[s^2-5s+6,s^2+5s+6]$, which is the exact range of values. In centered form, however, we represent f as $(x-2)(x+3)+6$; evaluation of this yields $[-s^2-5s,s^2+5s] +6$, which exceeds the actual width by $2s^2$.

It is possible (and desirable) to modify the rules of interval arithmetic in order to reduce spurious growth of intervals. One obvious and easily programmable change is to define, for all intervals I,

$$I^n = \{x^n : x \varepsilon I\}$$

This in general yields smaller intervals than the computation of $I_1*I_2*...*I_n$ for $I_1 = I_2 = ... = I_n = I$. Other modifications of this sort, which take into account known and easily computable exact ranges of values of a set of elementary common forms, may improve the performance (and possibly even speed up the operation of the system, as generally fewer multiplications will have to be performed during the computations).

Note that with changes of this sort, some of the properties of interval operations no longer hold. For example, with the change indicated above for raising to powers, subdistributivity no longer holds in its original form; the interval I*(I+1) need no longer be contained in the interval $I^2 + I$ (whether it is or not depends on I). If I = [-1,1], then

$$I*(I+1) = [-1,1] * [0,2] = [-2,2];$$

while

$$I^2 + I + [0,1] + [-1,1] = [-1,2].$$

This tends to complicate the representation problem even further, since it becomes desirable to have a representation contain as many (in some sense) as possible of the forms whose ranges of values are exactly computable. The changes are all for the better, however; the complications result because we now have better ways of representing functions than formerly.

SYSTEMS PROGRAMMING FOR INTERVAL ARITHMETIC. Programming for interval arithmetic is somewhat similar to writing (general real) computational routines in the early days of computing, before the hardware implementation of floating-point arithmetic. At level 1, the systems programmer has to build the basic tools for performing interval computations: an adder, a multiplier, an inverter for producing an interval (1/d,1/c) given the interval (c,d), and (if exponentiation is desired) functions that compute good bounds on the range of values of the EXP and LOG operators. (Similarly, other elementary transcendental functions such as SIN should be incorporated.)

At level 2, tools must be provided for convenient interfacing with the user. For a simple example: subtraction can obviously be implemented very simply using the adder of level 1; at the same time, it is clearly not desirable to have the user perform this implementation every time he wishes to execute subtraction. Thus, a set of subroutines must be provided for the user which he can conveniently call. There are likely to be a large number of such subroutines, for the following reason. It is generally desirable to allow the user to mix the mode of the variables freely; he should be allowed to add a integer-valued variable or constant to an integer-valued one. By the time all possible combinations of modes for operands are accounted for, the number of different subroutines provided is staggering. (Actually, there are typically about eight different routines, each of which has many entry points.)

It is clear that any such package of subroutines should be FORTRAN compatible. While the level 1 subroutines usually have to be written in machine language, there is usually no reason why the level 2 routines themselves cannot be written in the FORTRAN language.

The representation of interval numbers within a computer for FORTRAN might have been quite awkward were it not for the fact that formally an interval number looks just like a complex number. Any FORTRAN language compiler equipped to handle complex numbers can be tricked into handling interval numbers by the appropriate TYPE declarations. This is very handy for getting interval numbers in a decent format into and out of the computer, and also for defining interval-valued constants. (Arrays of interval numbers are also easier to handle if they are defined as being of TYPE COMPLEX.)

The arithmetic operations have to be performed by calls to the appropriate routines. Some computers (for example, the CDC 1604 and 3600) have a feature in their FORTRAN compilers which allow the definition of other (non-standard) variable types. What this means is that the compiler, when it encounters a variable of non-standard type, generates a call automatically to the appropriate arithmetic routine. This simplifies usage of interval arithmetic greatly, since the user, once he defines a variable as being of TYPE INTERVAL, can use it in statements as if it were any other type (integer or real). In fact, should this prove desirable, it is possible to define variables as being of type "double-precision interval" (the appropriate routines would have to be provided). For an example of an interval-arithmetic package of the sort just described, see [8].

The level 2 routines will depend to some extent on the exact working of the FORTRAN compiler. The level 1 routines are essentially compiler-independent; they are however heavily dependent on the way the given computer performs floating-point operations. (For convenience of interfacing with FORTRAN, the interval endpoints should usually be represented as floating-point numbers.) The (real) operations have to be performed at each end point in roughly the sequence: 1) perform the operation in a double length accumulator by using both the A and the Q registers without rounding; 2) normalize the result; 3) round to a single-precision floating-point number by adding (or subtracting) a 1 in the last place, unless the result was exact. If the computer does not allow this sequence of operations to be performed using the hardware floating-point instructions, then these operations have to be simulated by software, using fixed-point instructions.

Similar considerations apply to the computation of the transcendental functions. The functions should be computed in such a way that the result is off by at most one in the least significant bit of the single-precision answer.

Exponentiation can be implemented using the LOG and EXP routines. The system should however first determine if the exponent is an integer (even if represented as a floating-point number). As indicated, a substantial reduction in the growth of the widths of intervals can be effected if integer exponentiation is computed by repeated multiplications, using the true-range-of-values for raising to powers.

REFERENCES. The first place any interested reader should look is Moore [7]; aside from its definitive nature, it contains a rather complete bibliography of relevant literature. For a more up-to-date list, see Bennett [1].

[1] G.K. Bennett, Jr. "A Method for Locating the Zeros of a Polynomial using Interval Arithmetic." Report published by the Computer Center, Texas Technological College, June 1967.

[2] R.E. Boche, "Complex Interval Arithmetic with same Applications" Unpublished Master's Thesis, San Jose State College, 1966.

[3] R.H. Dargel, F.R. Loscalzo, and T.H. Witt. "Automatic Error Bounds on Real Zeros of Rational Functions." Communications of the ACM, Vol. 9, Number 11, Nov. 1966.

[4] E.R. Hansen, "Interval Arithmetic in Matrix Computations, Part I." SIAM Journal on Numerical Analysis, Series B, Vol. 2, Number 2 (1965).

[5] E.R. Hansen and R. Smith, "Interval Arithmetic in Matrix Computations, Part II." SIAM Journal on Numerical Analysis, Series B (to appear).

[6] E.R. Hansen, "On Solving Systems of Equations Using Interval Arithmetic." Mathematics of Computation (to appear).

[7] R.E. Moore, Interval Analysis. Prentice-Hall, Inc. 1966.

[8] A. Reiter, "Interval Arithmetic Package: INTERVAL". MRC Library Program #2, Mathematics Research Center, University of Wisconsin.

[9] A. Reiter, "Automatic Generation of Taylor Coefficients: TAYLOR". MRC Library Program #3, Mathematics Research Center, University of Wisconsin.