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 CALCULUS) works interactively, no type specification is necessary at all allowing specifying algorithms like in a math book.

CALCULUS 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 early next year. Some examples demonstrating the system are presented.

----

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

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 compare the results. However, this does not imply any guarantee of the correctness of coinciding figures. Consider the following example. Compute

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

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

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

$$\text{exact value} \quad : \quad f = - 0.827396059946821^{4}_{3}$$

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 CALCULUS has been developed without knowing MATLAB. After all many notations 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 $f:\mathbb{R}^n \to \mathbb{R}^n$ 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 the inverse of f' at some fixed point $\tilde{x}$. Consider the function $g:\mathbb{R}^n \to \mathbb{R}^n$ defined by

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

If there is a set X, an element of the power set $\mathbb{P}\mathbb{R}^n$ of $\mathbb{R}^n$, 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 $\mathcal{Z} \in \mathcal{PR}^n$ be a set of vectors, $\mathcal{C} \in \mathcal{PR}^{n\times n}$ be a set of matrices and $X \in \mathcal{PR}^n$ be a compact set of vectors. If then

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

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

Here int(X) denotes then 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:

$$f(X) := \{ f'(x) \mid x \in X \}$$

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

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

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

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 explicitly 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 $A \in R^{n\times n}$ be a real matrix, $b \in R^n$ be a vector, $R \in R^{n\times n}$ be a matrix, $x \in R^n$ be a vector and $X \in PR^n$ be a compact set of vectors. If then

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

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 $\hat{x} \in x+X$.

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 $\mathit{A} \in PR^{n\times n}$ be a set of matrices, $\mathit{b} \in PR^n$ be a set of vectors, $R \in R^{n\times n}$ be a matrix, $x \in R^n$ be a vector and $X \in PR^n$ be a compact set of vectors. If then

$$(3)\quad R\cdot\{\mathit{b}-\mathit{A}\cdot x\} + \{Id - R\cdot \mathit{A}\}\cdot X \subsetneq int(X) \quad ,$$

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

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

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

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

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

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 $\mathit{IR}^{n\times n}$, those over the complex numbers by IC, IC^n and $\mathit{IC}^{n\times n}$, 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 , $\mathit{IF}^{n\times n}$, ICF, ICF^n and $\mathit{ICF}^{n\times n}$, respecitvely. 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 $f:\mathcal{R}^n \rightarrow \mathcal{R}^n$ can be split in the following five steps:

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

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

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

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) $f(e) = e^2 + \pi^2 = 17.25866... \in \mathcal{R}$,

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]:

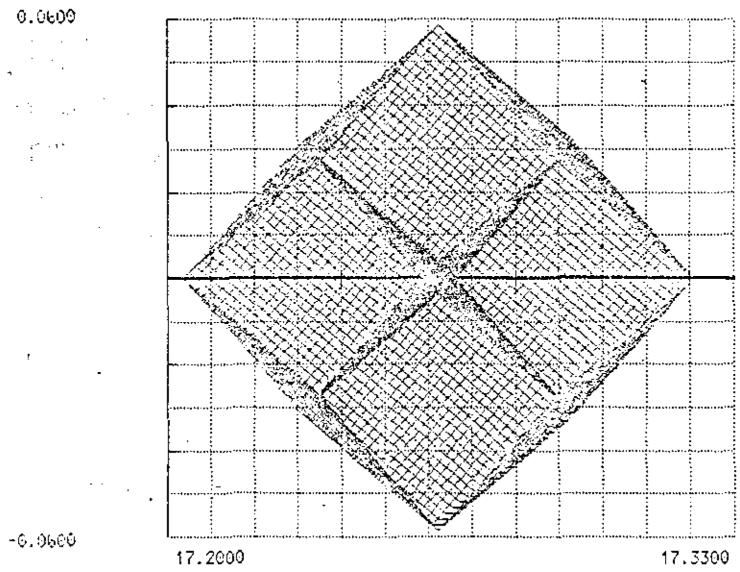
2) $g(E) = \{ f(x) \,|\, x \quad E,\ x \quad \mathcal{R}^n \} =$
$\{ (a+p\bullet i)\bullet(a-p\bullet i) \,|\, a \quad E,\ p \quad P \} =$
$\{ a^2+p^2 \,|\, a \quad E,\ p \quad P \} =$
$\{ x \quad \mathcal{R} \,|\, 17.2037 \le x \le 17.3209 \}$.

The evaluation makes use of the monotonicity of the square function. This evaluation is 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) $h(E) =$
$\{ (E+P\bullet i)\bullet(E-P\bullet i) \,|\, +,-,\bullet$ power set operations $\}$
$\{ (a1+p1\bullet i)\bullet(a2-p2\bullet i) \,|\, a1,a2 \quad E,\ p1,p2 \quad P,$
$+,-,\bullet$ real operations $\}$

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 CALCULUS) is the following:



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) H(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) $f(Y),\ Y \quad \mathit{PR}^n$ with $f(Y) := \{f(y) \mid y \quad Y\}$

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

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

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

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

5) $F(X),\ X \quad \mathit{IF}^n$ 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), (20 or (3) of the preceding chapter, then $F(X) \subseteq \mathrm{int}(X)$ implies immediately $f(X) \subseteq \mathrm{int}(X)$ 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 $F(X) \subseteq \mathrm{int}(X)$ 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-dot-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 5×5 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-dot-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-dot-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-dot-matrix                            O
                                                               O
     1.1088352458                                              O
     1.2032042029                                              O
     1.1393663790                                              O
     0.4080069615                                              O
     0.2359223927                                              O
```

These figures are all correct because an inner product is 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 CALCULUS will be demonstrated. If in an expression in CALCULUS 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 CALCULUS-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 this program we check on the eigenvalues of P.

```
format f5; eig(P)                                              I

ANS = (5,1)-real-dot-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-dot-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+/-0 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+/-0,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. This result is guaranteed to be correct.

CALCULUS 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 CALCULUS 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 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; re(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 15×15 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-dot-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-dot-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)                              O
     ( 0.72009005 +0.03525145i)                              O
     ( 0.82619613 -0.24994861i)                              O
( 0.72101034 -0.03942435i)                               0
( 0.73783840 -0.11319895i)                               0
( 0.77212545 -0.18458822i)                               0
```

Using a today 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.

CALCULUS 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 CALCULUS, which will be programmed in C, will cover part of this.

Because of the extremely powerful operators, code written in CALCULUS 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 CALCULUS instead of FORTRAN. When developing the code in very short time using CALCULUS and then switching to FORTRAN for production code there is the additional advantage that CALCULUS is a specification in mathematical 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 Computations, Academic Press, 1983. ACADEMIC PRESS, New York (1981).

[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).

[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 Seiten, (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).