published in H. Bulgak and C. Zenger, editors, Error Control and adaptivity in Scientific Computing, pages 279-298. Kluwer Academic Publishers, 1999

VERIFIED SOLUTION OF LARGE LINEAR AND NONLINEAR SYSTEMS

SIEGFRIED M. RUMP*

Abstract. In this paper we describe some of the principles of methods for the verified solution of large systems of linear and nonlinear equations. Verified solution means to produce results with correct error bounds using floating point arithmetic, possibly with directed rounding. Some background is given why and for what kind of problems the methods work, and also why other approaches do not work.

1. Introduction. In numerical analysis it is usually distinguished between backward and forward error analysis. Given an approximation of the true solution of a given problem, the frequently used backward analysis asks: "How large a perturbation of the original input data is necessary such that the computed approximation is the exact solution of the perturbed problem?" For many numerical algorithms this perturbation is small. Consider, for example, the solution of a linear system $Ax = b$ with $A \in \mathbf{M}_n(\mathbb{R}), b \in \mathbb{R}^n$. If $\widetilde{L}, \widetilde{U} \in \mathbf{M}_n(\mathbb{F})$, where $\mathbb{F}$ denotes the set of floating point numbers, are the computed factors and $\widetilde{x} \in \mathbb{F}^n$ is the approximate solution computed by Gaussian elimination with partial pivoting, then $\widetilde{A}\widetilde{x} = b$ where

$$|\widetilde{A} - A| \leq (3\,\varepsilon' + \varepsilon'^2)\,|\widetilde{L}|\,|\widetilde{U}| \quad \text{and} \quad \varepsilon' := n\,\varepsilon/(1 - n\,\varepsilon) \quad .$$

Here ε denotes the relative rounding error unit. This estimation ([23], [18]) is rigorous provided $n\,\varepsilon < 1$. If the factor in backward analysis is small, as for Gaussian elimination, the used algorithm is stable. More precisely, the algorithm does not introduce additional instability. This is not always true. A familiar example is the solution of least squares problems by normal equations instead of some orthogonalization technique, thus unnecessarily squaring the condition number of the problem.

On the other hand, backward analysis does not tell the condition of the problem. In case of Gaussian elimination, for example, the condition can be read from the growth factor. It is well known that for Gaussian elimination with partial pivoting the growth factor may grow exponentially with the dimension. On the other hand, until the work by Wright ([24], see also [7]) it was common belief that this occurs only in constructed, nonpractical examples. Since then the growth factor is monitored in LAPACK routines [3].

In forward error analysis the question is: "How far is the computed solution from the true solution of the problem?" including the question "Is the problem solvable?" At first sight this seems to be the more natural question to ask. This is not necessarily always the case. In many cases a good backward error is totally sufficient. If it is important to know the forward error of an approximation, for example, in problems where safety is important or controllability has to be assured, traditional estimations contain the condition number of the problem. For Gaussian elimination, for example, it is

$$\frac{||x - \widetilde{x}||}{||x||} \quad \leq \quad \frac{\varepsilon\,\mathrm{cond}(A)}{1 - \varepsilon\,\mathrm{cond}(A)}$$

provided $Ax = b$, $(A + E)\widetilde{x} = b$ and $||E|| \leq \varepsilon\,||A||$, $\varepsilon\,\mathrm{cond}(A) = \varepsilon\,||A||\;||A^{-1}|| < 1$ (cf. [10]). A finite condition number implies solvability of the problem. For many problems the reciprocal of the condition number is proportional to the distance to the nearest ill-posed problem. This is well known for normwise distances [6], and has recently proved to be also true for linear systems and componentwise distances [22]. However, for a valid estimate of the forward error we need an upper bound for the condition number, and this is frequently as difficult to compute as to solve the problem.

*Inst. f. Informatik III, Technical University Hamburg-Harburg, Eißendorfer Str. 38, 21071 Hamburg, Germany, (rump@tu-harburg.de).

Traditional forward error analysis gives general bounds, independent of the actual data of the problem. Probably the oldest bounds of this kind are in [16]. Those bounds turned out to be very pessimistic, possibly one reason that today mostly backward bounds are used. In the following we will discuss possibilities to compute forward error bounds for a given problem taking into account the actual data. These bounds will cover all intermediate errors such as representation errors, rounding errors and all kinds of computational errors. The bounds are completely rigorous.

A major ingredient of those methods is the possibility to estimate the error of a sequence of operations and to estimate the range of a function. This is to be described in the next section.

2. Range of a function. One possibility to estimate the error of an individual operation is interval arithmetic. Consider the set of nonempty real intervals $\mathbb{IIR} := \{[\,a_1, a_2\,] \,|a_1, a_2 \in \mathbb{R} \text{ and } a_1 \le a_2\}$. Power set operations over real intervals are defined by

$$A, B \in \mathbb{IIR} \quad \Rightarrow \quad A \circ B := \{a \circ b \,|\, a \in A, b \in B\}$$

where $\circ \in \{+, -, \cdot, /\}$. This definition can also serve as definition of interval operations because for $A, B \in \mathbb{IIR}$ the result of the power set operations is always a real interval (provided the denominator does not contain zero in case of division). For $A = [\,a_1, a_2\,], B = [\,b_1, b_2\,]$, this definition is equivalent to

$$A \circ B = [\,\min(a_1 \circ b_1, a_1 \circ b_2, a_2 \circ b_1, a_2 \circ b_2), \max(a_1 \circ b_1, a_1 \circ b_2, a_2 \circ b_1, a_2 \circ b_2)\,] \quad . \tag{2.1}$$

For intervals with floating point endpoints, the same definition can be used provided operations for the left bound are executed with rounding downwards, and operations for the right bound are executed with rounding upwards. For details see a standard book on interval analysis, among them ([2],[15]). We want to stress that (2.1) is *not* used for computation of an interval result. Here much faster methods are available, see Section 6.

The definition can be extended to complex operations, to vector and matrix operations, and to real and complex standard functions. The actual computation of the result for complex standard functions is nontrivial (see [4], [12]).

All interval operations satisfy one fundamental property, the isotonicity. It is

$$\forall\, a \in A \quad \forall\, b \in B: \qquad a \circ b \;\in\; A \circ B \tag{2.2}$$

for all suitable A,B and all suitable interval operations $\circ$. For monadic operations isotonicity is formulated similar to (2.2).

Having isotonicity at hand, it seems natural to apply interval arithmetic in a so-called *naive way*: Replacing every operation by the corresponding interval operation produces a final result which includes the true solution of the problem. This follows trivially by successively applying the isotonicity property (2.2). However, other than in special cases, this produces a severe overestimation of the result. One reason is the so-called wrapping effect. Consider the iteration

$$z_0 = 1; \quad z_{\nu+1} := (a + ib) \cdot z_\nu \quad \text{where} \quad a = b = 1/\sqrt{2}. \tag{2.3}$$

Evaluation of (2.3) in floating point introduces a small roundoff error in the computation of a and b. In the following iterations, the interval inclusion of z_ν is always turned by 45°. The interval inclusion is always a rectangular parallel to the axes. That means, in every iteration step the true length of the axes is prolonged by a factor $\sqrt{2}$, which means more than a factor 10^{16} after 100 iterations. Another reason for overestimation is the fact that one interval is used several times. For example, using interval arithmetic it is

$$e^X - X = [\,0, e\,] \qquad\qquad \text{for } X = [\,0, 1\,] \quad ,$$

instead of $[\,1, e - 1\,]$ using power set operations. This is the dependency problem. If every interval quantity occurs exactly once in an expression, then the final result shows no overestimation. Otherwise, except special

circumstances, the result is afflicted with overestimation. As a more practical example consider Gaussian elimination in interval arithmetic, frequently called IGA (Interval Gaussian Elimination). That is Gaussian elimination with partial pivoting is performed by replacing every operation by its corresponding interval operation.

For the following table we generated a random $n \times n$ matrix with entries uniformly distributed in $[\,-1, 1\,]$. We performed IGA for the original matrix, for the matrix preconditioned with an approximate inverse of its L-factor, and preconditioned with an approximate inverse of the entire matrix. In the second case, the preconditioned matrix is approximately the U-factor, in the third case approximately the identity matrix. We used INTLAB for the computation, see Section 6. The input is

```
for n=10:10:100; a=2*rand(n)-1;
  [l u]=luwpp(intval(a)); r1=rad(u(n,n));
  [l u]=lu(a); [l u]=luwpp(inv(l)*intval(a)); r2=rad(u(n,n));
  [l u]=luwpp(inv(a)*intval(a)); r3=rad(u(n,n));
  [n r1 r2 r3]
end
```

The output display only the radius of the U_{nn} element, where U is the U-factor produced by IGA, an interval matrix. The result is as follows.

```
  10  1.6642e-013  2.8755e-014  1.7764e-015
  20  2.7654e-010  3.9713e-012  3.8303e-015
  30  5.8932e-008  2.5538e-011  1.0991e-014
  40  8.2912e-005   4.318e-009  7.9381e-015
  50     0.027519  1.9748e-007  1.0936e-014
  60          NaN  1.8863e-005  1.1879e-014
  70          NaN   0.00036907  1.4211e-014
  80          NaN     0.026651  1.5155e-014
  90          NaN     0.092109  1.7042e-014
 100          NaN          NaN  2.0428e-014
```

TABLE 2.1. *Radius of U_{nn} for IGA*

Obviously, the table displays the exponential growth of interval radii produced by IGA for the first and second case. An output `NaN` means that computation broke down due to the fact that all pivoting elements contained zero. Only in the third case, where there is almost nothing to do, the algorithm seems to work. Note that the reason for the observed effect is *not* the condition number, it is the mere number of operations. The condition number in all cases is less than 10^3, where computation is performed in double precision (16 decimal places).

On the other hand, interval arithmetic offers the possibility to estimate the range of a function without further knowledge about Lipschitz constants or others. Consider, for example, the following function taken from a paper by Broyden [5]

$$y = \begin{pmatrix} 0.5\,\sin(x_1 x_2) - x_2/4\pi - x_1/2 \\ (1 - 1/4\pi)(e^{2x_1} - e) + e \cdot x_2/\pi - 2ex_1 \end{pmatrix} \quad . \tag{2.4}$$

Then the INTLAB function

```
function y = f(x)
  Pi = typeof(intval('3.14159265358979_'), x);
  E = exp(typeof(intval(1)), x);
  y(1) = .5*sin(x(1)*x(2)) - x(2)/(4*Pi) - x(1)/2;
```

```
y(2) = (1-1/(4*Pi))*(exp(2*x(1))-E) + E*x(2)/Pi - 2*E*x(1);
```

calculates the function (2.4). The call

```
x = [ .5 ; 3.14 ]; y = f(x)
```

calculates the function in pure floating point, whereas the call

```
x = intval([ .5 ; 3.14 ]); y = f(x)
```

calculates the function with verified bounds. If x is specified to be an interval like $x = \text{midrad}\,(\,[\,.5; 3.14\,],\ 1e-5\,)$, then the range of the function over the input interval is estimated. The function `typeof(a,b)` gives back the value of `a` with type adjusted to that of `b`.

Again, the estimation of the range will be afflicted with an overestimation. The objective of verification methods is to diminish the effect of overestimation. One of the possibilities to do that is to formulate the problem in such a way that all interval quantities are multiplied by a small factor. This is possible as demonstrated in the next section.

3. Verified solution of dense linear systems. Consider a (dense) system of linear equations

$$Ax = b \quad \text{for} \quad A \in \mathbf{M}_n(\mathbb{R}), b \in \mathbb{R}^n \quad . \tag{3.1}$$

It turns out that the use of fixed point theorems can take advantage of interval arithmetic and the possibility to estimate the range of a function. We reformulate (3.1) into the fixed point form

$$g(x) := x + R(b - Ax) \quad , \tag{3.2}$$

where $R \in \mathbf{M}_n(\mathbb{R})$ is a preconditioning matrix to be specified. Suppose for some interval vector $X \in \mathbb{IR}^n$, which is nonempty, closed, bounded and convex, it is

$$g(X) \subseteq X \quad . \tag{3.3}$$

Then Brouwer's Fixed Point Theorem implies existence of a fixed point $\widehat{x} \in X$ of g, and (3.2) implies

$$R(b - A\widehat{x}) = 0 \quad . \tag{3.4}$$

If the preconditioner is nonsingular, then a solution of the linear system (3.1) is enclosed in X. The proof of nonsingularity of the preconditioning matrix R together with that of A is given by the following lemma.

LEMMA 3.1. *Let $X, Z \in \mathbb{IR}^n$, $\mathcal{C} \in \mathbb{IM}_n(\mathbb{R})$ and suppose*

$$Z + \mathcal{C} \cdot X \subseteq \text{int}(X).$$

Then $\rho(C) < 1$ for all matrices $C \in \mathcal{C}$.

For the proof of this and the following results in this section see [19]. The naive way to check (3.3) is to replace every operation by its corresponding interval operation. If $X - R\,(\,b - A \cdot X\,) \subseteq \text{int}\,(X)$, then $g(X) \subseteq \text{int}(X)$. However, this will never be satisfied because adding some quantity to an interval can never shrink its diameter. Instead, we use the so-called Krawczyk operator [13]:

$$\widetilde{x} + R\,(\,b - A\widetilde{x}\,) + (\,I - RA\,)\,(\,X - \widetilde{x}\,) \subseteq \text{int}\,(x) \;\Rightarrow\; g(x) \subseteq \text{int}\,(x) \quad ,$$

where $\widetilde{x} \in \mathbb{R}^n$. The proof follows by isotonicity:

$$\begin{aligned} \forall\, x \in X: \quad g(x) &= x + R\,(\,b - Ax\,) \\ &= \widetilde{x} + R\,(\,b - A\widetilde{x}\,) + (\,I - RA\,)\,(\,x - \widetilde{x}\,) \\ &\subseteq \widetilde{x} + R\,(\,b - A\widetilde{x}\,) + (\,I - RA\,)(\,X - \widetilde{x}\,) \\ &\subseteq \text{int}\,(X) \quad . \end{aligned}$$

Together with Lemma 3.1 this implies nonsingularity of the matrices R and A, and therefore $A^{-1}b \in X$. For a practical application it turns out to be superior to enclose the error with respect to an approximate

solution of the linear system. This gives better inclusions for less work. Moreover, it is easy to derive a corresponding theorem for linear systems with uncertain data.

THEOREM 3.2. *Let* $[A] \in \mathbb{IM}_n(\mathbb{R}), [b] \in \mathbb{IIR}^n, R \in \mathbf{M}_n(\mathbb{R}), \widetilde{x} \in \mathbb{R}^n$ *and* $X \in \mathbb{IIR}^n$ *be given. Suppose*

$$R \cdot ([b] - [A]\widetilde{x}) + (I - R \cdot [A])\, X \subseteq int(X) \quad . \tag{3.5}$$

Then the matrix R *and all matrices* $A \in [A]$ *are nonsingular, and*

$$\forall\, A \in [A] \quad \forall\, b \in [b]: \qquad A^{-1}b \in \widetilde{x} + X \quad . \tag{3.6}$$

In a practical computation, the optimal preconditioner is the inverse of the midpoint matrix. Therefore, usually an approximate midpoint inverse is used. For the quality of the inclusion, consider the inclusion formula (3.5). The critical term is $(I - R[A])X$. However, this is the product of two small quantities: For a linear system with precise date, $I - RA$ is of the order $\varepsilon \cdot ||A||$, and X is an inclusion of the *error* of $\widetilde{x}$, which is also small if the problem is not too ill-conditioned.

It turns out in practice that the same limit $1/\varepsilon$ for the condition number applies to the verification algorithm. Therefore, one may ask what verification is useful for. The major difference to a pure floating point algorithm is that every result is verified to be correct. In pure floating point methods one might take a poor approximation to be a correct answer to a problem, see [24], [7]. This is not possible for verification methods.

Still the question remains: If a verification method delivers a poor answer, is this due to overestimations and dependencies or, is it due to the poor condition of the problem. This is solved by the following observation.

THEOREM 3.3. *With the assumptions of Theorem 3.2, for all* $i, 1 \le i \le n$ *there exist* $A_1, A_2 \in [A]$ *and* $b_1, b_2 \in [b]$ *such that with*

$$Z := R \cdot ([b] - [A]\widetilde{x}) \quad \textit{and} \quad \Delta := (I - R[A])\, X$$

it is

$$(A_1^{-1}b_1)_i \le \inf(Z_i) + \sup(\Delta_i) \quad \textit{and} \quad \sup(Z_i) + \inf(\Delta_i) \le (A_2^{-1}b_2)_i \quad .$$

Note that Theorem 3.2 implies

$$\inf(Z_i) + \inf(\Delta_i) \le (A^{-1}b)_i \le \sup(Z_i) + \sup(\Delta_i)$$

for all $A \in [A], b \in [b]$ and for all $i, 1 \le i \le n$. Henceforth, the quality of the inclusion depends on the width of Δ, and due to the preceeding discussion this is the product of two small quantities and therefore small. This is the reason why inclusions obtained by Theorems 3.2 and 3.3 are very sharp. We mention that an intrinsic assumption is that all uncertain data are independently varying within the given tolerances. If there are dependencies between input data, consult [11] and [19].

As an example consider the INTLAB input

```
n=500; A=2*rand(n)-1; b=A*ones(n,1); cond(A)
X = verifylss(A,b); X(1), X(n)
```

producing the output

```
8.753696537725864e+002
[   0.999999999989908,   1.000000000010085]
[   0.999999999990334,   1.000000000009646]
```

According to the condition number, the radius 10^{-11} is about what to expect in double precision. If we afflict the input data with tolerances like

```
X = verifylss(midrad(A,1e-10),b); X(1), X(n),
```

which means all matrix entries are afflicted with an absolute uncertainty of 10^{-10}, then the result is as follows:

```
[   0.999997448889150,   1.000002551110845]
[   0.999997561317640,   1.000002438682342]
```

The radii $2 \cdot 10^{-6}$ of the inclusion reflect the condition of the problem. Note that usually only approximate information about the condition number is available. Those approximations may be inaccurate (see the discussion in [10], Chapter 14 and the many papers cited over there).

4. Verified results for sparse linear systems. The methods described in the previous section are not suitable for sparse systems because an approximate inverse is used as preconditioner. For sparse systems we use an estimation of the smallest singular value of the matrix. In the following $||\cdot||$ denotes always the spectral norm; the results may be formulated for other norms as well. For $A\widetilde{x} = b$ and a given approximation $\widetilde{x}$ it is

$$||\widetilde{x} - \widehat{x}|| = ||A^{-1}(A\widetilde{x} - b)|| \leq \sigma_n(A)^{-1} \cdot ||A\widetilde{x} - b|| \quad , \tag{4.1}$$

where $\sigma_n(A)$ denotes the smallest singular value of A. For the moment we suppose A to be symmetric positive definite. In the course of estimations it will be *proved* that A has this property. Later, the concepts will be extended to general symmetric and unsymmetric matrices.

For the estimation we need a verified *lower* bound for the smallest singular value of A. Such a positive bound implies nonsingularity of A. One possibility is to compute an approximation s for $\sigma_n(A)$, and to verify that $A - sI$ is positive definite. This in turn is true if a Cholesky decomposition of $A - sI$ exists. One might try to prove this by performing an interval Cholesky decomposition. However, the same remarks as for IGA apply: due to overestimation and dependencies this works only for small dimension. Consider

$$A = 0.1 \cdot GG^T \text{ with } G = \begin{pmatrix} 1 & & & & \\ 1 & 1 & & 0 & \\ 1 & 1 & 1 & & \\ & 1 & 1 & 1 & \\ 0 & \ddots & \ddots & \ddots & \end{pmatrix} \quad .$$

We perform a Cholesky decomposition of A with interval operations and monitor the radius of the lower right element of the Cholesky factor. The result is as follows.

n	10	20	30	40
rad(G_nn)	7e-13	1e-8	2e-4	failed
cond(A)	200	600	1000	2000

Again, the failure of the approach is due to the number of operations, not due to the condition of the matrix. In contrast we use perturbation analysis of the symmetric eigenvalue problem. It is well known that for symmetric A and E and suitable numbering the following bounds are valid:

$$|\lambda_i(A + E) - \lambda_i(A)| \leq ||E||_2 \qquad \text{for } 1 \leq i \leq n \quad . \tag{4.2}$$

Here, λ_i denotes the *i-th* eigenvalue of a (symmetric) matrix. Suppose, s is an approximation to the smallest singular value of a symmetric matrix, and $HH^T \approx A - sI$ is an *approximate* Cholesky decomposition. If H has no zero on the diagonal, HH^T is positive definite, and by (4.2) it is for all i

$$|\lambda_i(A - sI)| \geq -||\Delta||, \qquad \text{where } \Delta := A - sI - HH^T \quad .$$

This implies

$$\sigma_n(A) \geq s - ||\Delta|| \quad ,$$

a lower bound for the smallest singular value of A. The only computation to be performed in interval arithmetic in this verification process is the computation of Δ. Therefore, a verification may look as follows.

1. Compute an approximate Cholesky decomposition $A \approx GG^T$
2. Use G to compute an approximate solution $\widetilde{x}$
3. Use G and inverse power iteration to compute an approximation t to the smallest singular value of A
4. Set $s := 0.9t$
5. Compute an approximate Cholesky decomposition $A - sI \approx HH^T$
6. Compute an upper bound δ of $||A - sI - HH^T||$
7. If $s - \delta > 0$, then A is positive definite and $||A^{-1}b - \widetilde{x}|| \leq (s-\delta)^{-1} \cdot ||A\widetilde{x} - b||$.

ALGORITHM 4.1. *Verified solution of s.p.d. linear systems*

The applicability of Algorithm 4.1 is limited by $\mathrm{cond}(A) \lesssim \varepsilon^{-1}$, the same bound as for any floating point algorithm. Algorithm 4.1 may also serve as a verified condition estimator. Sometimes it is even faster than an approximate condition estimator. Consider a linear system with $A := LL^T$, where L is an $n \times n$ random band matrix with band width 7. Therefore, A is of bandwidth 14. We calculate the right hand side such that the true solution of the linear system is $\widehat{x}_i = (-1)^i/i$. In the following table we display the maximum error achieved by Algorithm 4.1, the condition number of A and the ratio

$$\rho := \frac{t(\text{LAPACK condition estimator})}{t(\text{verified inclusion by Algorithm 4.1})} \quad ,$$

the ratio between the time for the LAPACK condition estimator divided by the total time to achieve the verified inclusion by Algorithm 4.1.

n	$\|\|\widehat{x} - \widetilde{x}\|\|_\infty$ / $\|\|\widehat{x}\|\|_\infty$	cond(A)	ρ
1 000	2.0e-9	2.5e7	1
10 000	2.4e-6	3.5e10	40
100 000	3.0e-5	4.3e11	1100

TABLE 4.2. *Computing time for s.p.d. systems*

The main principle of this and many other verification methods is that interval computations are used only if absolutely necessary. Almost everything is done in floating point; interval arithmetic is only used to verify validity of certain assumptions of an inclusion theorem. To cite Wilkinson: "In general it is the best in algebraic computations to leave the use of interval arithmetic as late as possible so that it effectively becomes an a posteriori weapon."

For symmetric indefinite systems, the eigenvalues and singular values do not coincide. The problem can still be solved by perturbation theorems for symmetric matrices. Suppose, we have an approximation s to the smallest singular value of A. We perform an LDL^T-decompositions of $A - sI$ and $A + sI$:

$$L_1 D_1 L_1^T \approx A - sI \quad \text{and} \quad L_2 D_2 L_2^T \approx A + sI \quad .$$

We use symmetric pivoting together with the method of Bunch-Kaufmann ([8], Chapter 4.4). Furthermore, we use a limited pivoting strategy to reduce fill-in [14]. By the perturbation result for symmetric eigenvalue problems, the inertia of $A - sI$ and $A + sI$ cannot differ by more than

$$\delta := \max\left(||A - sI - L_1 D_1 L_1^T||\,,\, ||A + sI - L_2 D_2 L_2^T|| \right)$$

from the inertia of D_1 and D_2. If the inertias of D_1 and D_2 are equal, then it is not difficult to see that $s - \delta$ is a lower bound for the smallest eigenvalue in absolute value of A, which is the smallest singular value

of A. Therefore, the approach is similar to Algorithm 4.1 with the exception that the verification process is two-fold, the factorization of $A - sI$ and $A + sI$.

For indefinite systems we use the fact that the set of eigenvalues of

$$B := \begin{pmatrix} 0 & A^T \\ A & 0 \end{pmatrix}$$

is $\pm\sigma_i(A)$. Thus the condition number of B and A are the same. The verification process uses the above approach for symmetric linear systems. A band minimization can be performed in advance by taking advantage of the special structure of the matrix B.

Computational results for Gregory/Karney test cases 4.16 and 4.20 are as follows. The first matrix is positive definite, the second indefinite. The column "error" shows $||\widehat{x} - \widetilde{x}||_\infty \,/\, ||\widehat{x}||_\infty$.

	Example (4.16)		Example (4.20)	
n	cond(A)	error	cond(A)	error
1000	1.7e11	3.8e-14	6.2e2	1.0e-18
10000	1.6e15	5.4e-10	5.5e3	7.6e-17
20000	2.6e16	1.8e-8	1.1e4	2.6e-16
50000	1.0e18	failed	3.2e4	4.1e-15
100000			6.3e4	7.6e-14

TABLE 4.3. *Verified solution of banded systems*

5. Verified solution of nonlinear systems. The methods derived in the previous sections can be used for the inclusion of the solution of nonlinear systems of equations. The following results hold in a more general setting; for simplicity, we state the results with rather strong preassumptions. Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be a continuously differentiable function, and let $\widetilde{x} \in \mathbb{R}^n$ and $X \in \mathbb{IR}^n$ be given. We call $S(\widetilde{x}, X) \subseteq \mathbf{M}_n(\mathbb{R})$ an *expansion* of f in X with respect to $\widetilde{x}$ if

$$\forall\, x \in X \;\exists\, M \in S(\widetilde{x}, X) : \quad f(x) = f(\widetilde{x}) + M \cdot (x - \widetilde{x}) \quad .$$

Such an expansion can be computed by means of automatic differentiation ([17], [9]), automatic slopes ([15], [20]) or other techniques. Given such an expansion the following is true.

THEOREM 5.1. *Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be a continuously differentiable function, let $\widetilde{x} \in \mathbb{R}^n$, $x \in \mathbb{IR}^n$, $R \in \mathbf{M}_n(\mathbb{R})$ and $S(\widetilde{x}, x) \subseteq \mathbf{M}_n(\mathbb{R})$ be given. If*

$$-R \cdot f(\widetilde{x}) + (I - R \cdot S(\widetilde{x}, x)) \cdot X \subseteq \mathrm{int}(X) \quad , \tag{5.1}$$

then there exists some $\widehat{x} \in \widetilde{x} + X$ with $f(\widehat{x}) = 0$.

The proof can be found in [19]. Condition (5.1) can be verified on the computer. Again, the critical part $(I - R \cdot S(\widetilde{x}, x)) \cdot X$ subject to overestimation is the product of two small quantities. In a practical application, the matrix R is usually choosen to be a good preconditioner, for example an approximate inverse of the midpoint of the expansion matrix $\mathrm{mid}(S(\widetilde{x}, x))$.

For sparse nonlinear systems this is not possible because the inverse of the Jacobian is likely to be a full matrix. Nevertheless, we choose the preconditioner to be the *optimal* preconditioner, namely the *exact* inverse of $\mathrm{mid}(S(\widetilde{x}, x))$. Define

$$R := \mathrm{mid}(S(\widetilde{x}, x))^{-1} \quad .$$

Then the basic rules of interval analysis ([2], [15]) yield

$$\begin{aligned} I - R \cdot S(\widetilde{x}, x) &= I - R \cdot [\mathrm{mid}(S(\widetilde{x}, x)) \pm \mathrm{rad}(S(\widetilde{x}, x))] \\ &= [-|R| \cdot \mathrm{rad}(S(\widetilde{x}, x)) \,,\; +|R| \cdot \mathrm{rad}(S(\widetilde{x}, x))] \quad . \end{aligned}$$

A short calculation proves the following corollary of the preceeding theorem.

COROLLARY 5.2. *Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be a continuously differentiable function, let $\widetilde{x} \in \mathbb{R}^n$ and $0 < \rho \in \mathbb{R}$ be given and define $X := \{x \in \mathbb{R}^n \ : \ ||x|| \leq \rho\}$. Furthermore, let $S(\widetilde{x}, x) \in \mathbb{IM}_n(\mathbb{R})$ be given with*

$$\forall\, x \in X \ \exists\, M \in S(\widetilde{x}, x) : \quad f(x) = f(\widetilde{x}) + M \cdot (x - \widetilde{x}) \quad .$$

If $0 < \tau \in \mathbb{R}$ satisfies $\tau < \sigma_n(\mathrm{mid}(S(\widetilde{x}, x)))$ and

$$||f(\widetilde{x})|| + ||\mathrm{rad}(S(\widetilde{x}, x))|| \cdot \rho \leq \tau \cdot \rho \quad ,$$

then there exists $\widehat{x} \in \mathbb{R}^n$ with $||\widehat{x} - \widetilde{x}|| \leq \rho$ and $f(\widehat{x}) = 0$.

The advantage of the formulation of Corollary 5.2 is that it is suitable for sparse systems, and that it uses the optimal preconditioner without explicitly computing it. Note that if f is sparse, the expansion interval matrix $S(\widetilde{x}, x)$ is also sparse.

As an example consider Emden's equation

$$-\Delta U = U^2 \qquad \text{where } U = 0 \text{ on } \partial\,\Omega \text{ for } \Omega = (0, l^{-1}) \times (0, l) \quad .$$

This equation is not too difficult to solve on the unit square; for values of l larger than 2 the discretized system becomes rapidly ill-conditioned. For an approximate solution $\widetilde{u} := \lambda {x_1}^2 (1 - x_1)^2 {x_2}^2 (1 - x_2)^2$ with suitable parameter λ we perform some Newton steps and apply Corollary 5.2. The discretization uses central differences. The result for different values of l for the final error $e = ||\widehat{u} - \widetilde{u}|| \,/\, ||\widetilde{u}||$ is as follows.

l	N	m1	m2	cond	e
1	32385	255	127	1.3e4	7.7e-13
2	32385	255	127	8.0e6	4.5e-10
2.5	32385	255	127	7.9e9	4.6e-7

TABLE 5.3. *Computational results for Emden's equation*

Note that verification of the result applies to the discretized system. The column N denote the total dimension of the nonlinear system, columns $m1$ and $m2$ denote number of grid points in the two variables, and *cond* the condition number of the nonlinear system.

For larger values of l the discretized system becomes truely ill-conditioned. Moreover, plenty of solutions appear and turn out to be ghost solutions produced by floating point errors. For example, consider $l = 3.1$, $m_1 = 127$, $m_2 = 31$ such that $N = 3937$. Denote the floating point iterates by $\widetilde{u}^k$. Then the weighted residue shows the following behaviour.

k	$\|\|f(\widetilde{u}^k)\|\| \,/\, \|\|\widetilde{u}^k\|\|$
1	1.3e-3
2	9.8e-5
3	2.6e-7
4	3.1e-12
5	1.2e-4

TABLE 5.4. *Floating point iteration for Emden's equation*

What looks like an almost quadratic convergence finally turns out to be a fake. However, a stopping criterion may stop at iteration 4 and give back the computed approximation.

6. Implementation issues. Implementation of interval arithmetic is more involved than it looks at first sight. Consider, for example, the realization of (3.5). The computing intensive part in interval arithmetic is the multiplication $T = R \cdot [\, A \,]$, a real matrix times an interval matrix. A top-down implementation would be

```
T = zeros(n);
for i=1:n
  for j=1:n
    for k=1:n
      T(i,j) = T(i,j) + R(i,k)*A(k,j);
    end
  end
end
```

However, the most inner loop contains an interval multiplication and an interval addition. This has terrible consequences for the performance. First, the multiplication and addition each require two switches of the rounding mode, totally $4n^3$. Second, the compiler looses every possibility to optimize the inner loop, where this is exactly the strength of a floating point inner product. The latter would in fact use a multiply-and-add instruction, what is of course impossible in the above approach.

A much better way is the following. First, rewrite the interval matrix $[\,A\,]$ in midpoint-radius form, then multiply using BLAS routines. Suppose, A.inf and A.sup represent the matrices of lower and upper bounds of $[\,A\,]$, respectively. Then an algorithm looks as follows.

```
SetRoundUp
Amid = A.inf + 0.5*(A.sup-A.inf);
Arad = Amid - A.inf;
Trad = abs(R)*Arad;
Tsup = R*Amid + Trad;
SetRoundDown
Tinf = R*Amid - Trad;
SetRoundNear
```

ALGORITHM 6.1. *Real matrix times interval matrix*

The algorithm uses totally three matrix multiplications. However, these are calls to BLAS routines and are very fast on a variety of machines. Apparently this seems to be the fastest way to perform interval matrix multiplication. Note that the compiled `Tinf, Tsup` satisfy for all $A \in [\,A\,]$

$$\texttt{Tinf} \leq R * A \leq \texttt{Tsup} \quad .$$

For the other operations similar considerations apply. Those operations are collected in a Matlab toolbox called INTLAB [21]. The main advantage of INTLAB is that it is entirely written in Matlab code, with the exception of exactly the 3 routines for switching the rounding mode. The above examples are calculated in INTLAB.

An example of INTLAB code is the following algorithm for solving dense systems of equation according to Theorem 3.2. The following is original INTLAB code.

```
function X = denselss(A,b)        % linear system solver
  R = inv( mid(A) ) ;             % for dense matrices
  xs = R * mid(b) ;
  Z = R * (b-intval(A)*xs) ;
  C = speye(size(A)) - R*intval(A);
  Y = Z;
  E = 0.1*rad(Y)*hull(-1,1) + midrad(0,10*realmin);
  k = 0; kmax = 15; ready = 0;
  while ~ready & k<kmax
    k = k+1;
    X = Y + E;
```

```
  Y = Z + C * X;
  ready = in0(Y,X);
end
if ready
  X = xs + Y;
else
  disp('no inclusion achieved for \');
  X = NaN;
end
```

ALGORITHM 6.2. *Solution of dense linear systems*

A timing in seconds on a 120 Mhz Pentium I Laptop for Algorithm 6.2 is as follows.

dimension	pure floating point	verified point	verified interval
100	0.09	0.53	0.70
200	0.56	3.35	4.23
500	8.2	50.9	67.6

TABLE 6.3. *Solution of real linear systems*

Looking into the code of Algorithms 6.1 and 6.2 reveals that the total number of operations for Algorithm 6.2 for interval linear systems is $4n^3$, n^3 for the matrix inversion and $3n^3$ for the point matrix times interval matrix multiplication $R \cdot A$. Gaussian elimination needs $n^3/3$ operations, but the measured computing times show only a factor of 7 to 8 instead of 12. The reason is that matrix multiplication does not take 3 times as long as Gaussian elimination. Consider

```
n=200; A=2*rand(n)-1; b=A*ones(n,1); k=10;
tic; for i=1:k, inv(A); end, toc/k
tic; for i=1:k, A*A; end, toc/k
tic; for i=1:k, A\b; end, toc/k
```

producing

1.24 seconds	for matrix inversion,
0.78 seconds	for matrix multiplication, and
0.56 seconds	for Gaussian elimination.

Finally, we display the code for the solution of a system of nonlinear equations according to Theorem 5.1.

```
function [ X , xs ] = verifynlss(f,xs)
%VERIFYNLSS   Verified solution of nonlinear system
%
%   [ X , xs ] = verifynlss(f,xs)
%
% f is name of function, to be called by f(x), xs is an approximation
% optional output    xs  improved approximation
%

% floating point Newton iteration
  n = length(xs);
  xsold = 2*xs;
  k = 0;
```

```
  while ( norm(xs-xsold)>1e-10*norm(xs) & k<10 ) | k<1
    k = k+1;                    % at most 10, at least 1 iteration performed
    xsold = xs;
    x = initvar(xs);
    y = feval(f,x);
    xs = xs - y.dx\y.x;
  end

% interval iteration
  R = inv(y.dx);
  Z = - R * feval(f,intval(xs));
  X = Z;
  E = 0.1*rad(X)*hull(-1,1) + midrad(0,realmin);
  ready = 0; k = 0;
  while ~ready & k<10
    k = k+1;
    Y = hull( X + E , 0 );      % epsilon inflation
    Yold = Y;
    x = initvar(xs+Y);
    y = feval(f,x);             % f(x) and Jacobian by
    C = eye(n) - R * y.dx;      % automatic differentiation
    i=0;
    while ~ready & i<2          % improved interval iteration
      i = i+1;
      X = Z + C * Y;
      ready = in0(X,Y);
      Y = intersect(X,Yold);
    end
  end
  if ready
    X = xs+Y;                   % verified inclusion
  else
    X = NaN;                    % inclusion failed
  end
```

ALGORITHM 6.4. *Verified solution of nonlinear systems*

Again this is original INTLAB code. We want to stress that the function f need not to be changed to cover floating point arguments or interval arguments. The operator overloading generates automatically correct operations and code.

As an example consider a problem given by Abbott and Brent in [1], the discretization of

```
3 y" y + y'^2 = 0   with   y(0)=0; y(1)=20;
```

We solve the discretized system, not the continuous equation. In the paper, the initial (poor) approximation is a vector all entries of which are equal to 10, the true solution is `20*x^.75`. The discretized problem is specified by the following INTLAB function.

```
function y = f(x)
  y = x;
  n = length(x); v=2:n-1;
  y(1) = 3*x(1)*(x(2)-2*x(1)) + x(2)*x(2)/4;
```

```
y(v) = 3*x(v).*(x(v+1)-2*x(v)+x(v-1)) + (x(v+1)-x(v-1)).^2/4;
y(n) = 3*x(n).*(20-2*x(n)+x(n-1)) + (20-x(n-1)).^2/4;
```

Note the vectorized formulation of the function. The timing on the 120 Mhz Pentium I Laptop by the following statement

```
tic; X = verifynlss('f',10*ones(n,1)); toc
```

is

5.4 seconds	for dimension n=50,
8.5 seconds	for dimension n=100, and
20.3 seconds	for dimension n=200.

The first and last components of the inclusion are

```
X(1:4)
intval ans =
     0.346256418326_
     0.6045521734322
     0.8305219234696
     1.0376691412984
X(197:200)
intval ans =
    19.7005694833674
    19.775568557350_
    19.8504729393822
    19.9252832242374
```

An "_" in the output means that subtracting 1 from and adding 1 to the last displayed figure before the underscore produces a correct interval for the result. An output without underscore is correct up the the last digit. The output is also written in Matlab and is rigorous and correct.

7. Conclusion. Self-validating methods are designed to produce rigorous error bounds for the solution of numerical problems. We want to stress again that those methods are *not* thought to replace existing numerical methods whatsoever. Traditional numerical methods usually produce correct results within a certain error margin. There are situations where numerical methods produce poor or sometimes erreneous results. However, those situations are rare. To cite Vel Kahan: "Numerical errors are rare, rare enough not to care about them all the time, but yet not rare enough to ignore them."

Self-validating methods should be used when another degree of safety is necessary and/or if there is doubt about reliability of some approximation. The scope of applicability of self-validating methods is still limited, although always increasing. The very large problems treated today in numerical analysis are clearly out of the scope of current self-validating algorithms. However, there is the possibility of treating data with uncertainties which may also be advantageous beside the correctness of all results.

REFERENCES

[1] J.P. Abbott and R.P. Brent. Fast Local Convergence with Single and Multistep Methods for Nonlinear Equations. *Austr. Math. Soc. 19 (Series B)*, pages 173–199, 1975.

[2] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, New York, 1983.

[3] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and Sorensen D.C. *LAPACK User's Guide, Resease 2.0.* SIAM Publications, Philadelphia, second edition edition, 1995.

[4] K.D. Braune. *Hochgenaue Standardfunktionen für reelle und komplexe Punkte und Intervalle in beliebigen Gleitpunktrastern.* Dissertation, Universität Karlsruhe, 1987.

[5] C.G. Broyden. A new method of solving nonlinear simultaneous equations. *Comput. J.*, 12:94–99, 1969.

[6] J.B. Demmel. Condition Numbers and the Distance to the Nearest Ill-posed Problem. *Numer. Math. 51*, pages 251–289, 1987.

[7] L.V. Foster. Gaussian elimination with partial pivoting can fail in practice. *Siam J. Matrix Anal. Appl.*, 14:1354–1362, 1994.

[8] G.H. Golub and C. Van Loan. *Matrix Computations.* John Hopkins University Press, second edition, 1989.

[9] A. Griewank. On Automatic Differentiation. In *Mathematical Programming 88.* Kluwer Academic Publishers, Boston, 1989.

[10] N.J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM Publications, Philadelphia, 1996.

[11] C. Jansson. Interval Linear Systems with Symmetric Matrices, Skew-Symmetric Matrices, and Dependencies in the Right Hand Side. *Computing 46*, pages 265–274, 1991.

[12] W. Krämer. *Inverse Standardfunktionen für reelle und komplexe Intervallargumente mit a priori Fehlerabschätzung für beliebige Datenformate.* Dissertation, Universität Karlsruhe, 1987.

[13] R. Krawczyk. Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken. *Computing 4*, pages 187–201, 1969.

[14] T. Leistikow. Verifizierte Lsung groer Gleichungssysteme mit Bandstruktur. Technical report, Inst. f. Informatik III, TU Hamburg-Harburg, 1998.

[15] A. Neumaier. *Interval Methods for Systems of Equations, Encyclopedia of Mathematics and its Applications.* Cambridge University Press, 1990.

[16] J.v. Neumann and H.H. Goldstine. Numerical Inverting of Matrices of High Order. *Bull. Amer. Math. Soc. 53*, pages 1021–1099, 1947.

[17] L.B. Rall. Automatic Differentiation: Techniques and Applications. In *Lecture Notes in Computer Science 120.* Springer Verlag, Berlin-Heidelberg-New York, 1981.

[18] C. Reinsch. Die Behandlung von Rundungsfehlern in der numerischen Analysis. In S. D. Chatterji et al., editors, *Jahrbuch Überblicke Mathematik 1979*, pages 43–62, Mannheim, Wien, Zürich, 1979. Bibliographisches Institut.

[19] S.M. Rump. Verification Methods for Dense and Sparse Systems of Equations. In J. Herzberger, editor, *Topics in Validated Computations — Studies in Computational Mathematics*, pages 63–136, Elsevier, Amsterdam, 1994.

[20] S.M. Rump. Expansion and Estimation of the Range of Nonlinear Functions. *Mathematics of Computation*, 65(216):1503–1512, 1996.

[21] S.M. Rump. INTLAB - Interval Laboratory. submitted for publication, http://www.ti3.tu-harburg.de/rump/intlab/index.html, 1998.

[22] S.M. Rump. Ill-conditioned Matrices are componentwise near to singularity. *SIAM Review, to appear*, 1999.

[23] W. Sautter. *Fehlerfortpflanzung und Rundungsfehler bei der verallgemeinerten Inversion von Matrizen.* Dissertation, TU München, 1971.

[24] S.J. Wright. A collection of problems for which Gaussian elimination with partial pivoting is unstable. *SIAM J. Sci. Comput.*, 14(1):231–238, 1993.