Interval Computations
No 3, 1993

Verified Solution of Linear Systems Based on Common Software Libraries

Carlos Falcó Korn and Christian Ullrich

Usually routines of common software libraries compute approximations for the solution of a given problem and, in certain cases, corresponding accuracy estimations. These estimations are sometimes very precise. In other cases, they are completely wrong, giving rise to misleading interpretations of the computed solution. Therefore, we propose to extend existing software libraries by the capability of computing a guaranteed inclusion of the solution. For linear systems with interval H-matrices, this extension can be done without changing the given library, thus reusing the existing software. Furthermore, runtime and accuracy measurements show that the verification can be achieved with low additional costs—even for linear systems of high dimension.

Верифицированное решение линейных систем на основе библиотек подпрограмм общего назначения

К. Фалько Корн, Х. Уллрих

Как правило, подпрограммы, имеющиеся в библиотеках общего назначения, вычисляют приближенное решение данной задачи и, в некоторых случаях, соответствующие оценки точности. Эти оценки иногда весьма точны, хотя нередко они совершенно ошибочны и могут привести к неверной интерпретации полученного решения. Предложение авторов статьи сводится к тому, чтобы дополнить существующие библиотеки средствами, позволяющими получить гарантированное включение решения. Для линейных систем с интервальными H-матрицами такие средства могут быть добавлены без изменения существующей библиотеки. Кроме того, измерения времени счета и достигнутой точности показывают, что верификация для линейных систем даже большой размерности может быть осуществлена с незначительными дополнительными затратами.

1 Introduction

Many software libraries for solving numerical problems are now available [8]. Some of these packages are widely used because they are distributed freely via electronic mail servers [4]. In addition they are popular because the routines provided are efficient implementations of efficient algorithms. The major drawback when using these libraries is that the correctness of the results is not guaranteed. Frequently the user gets just an estimation for the number of correct digits of the computed "solution". This estimation is sometimes accurate, but in other cases, it is not reliable or is completely misleading. The well-known LINPACK package [5] gives a good example in the routines `matgen` and `DGECO`. The first generates a test matrix of a given dimension `n`; the second performs the LU-decomposition of a matrix and computes an estimation of its condition number, which can be used to estimate the number of correct digits. For `n = 128`, a very high condition number is delivered, thus suggesting an ill-conditioned problem. This may make the user cautious, but he gets no hint that the matrix is singular!

This problem is avoided by interval algorithms, which either compute a guaranteed inclusion of the solution or return a warning that the problem may not be solvable. In spite of this powerful feature, interval algorithms have not been accepted by the great majority of users. We believe that reasons for this lie in the philosophy of existing environments for verified computing. First, a user has no possibility of choice. For example, Pascal–XSC [11] provides just one routine for solving linear systems. That routine is so general that it works inefficiently in many cases. Second, existing software is not reused; everything is built "from scratch." A potential user has to decide between "all or nothing." If he needs guaranteed results, he has to drop the working environment or library he is accustomed to. Third, the proposed interval algorithms are too slow. This is partly due to the ambition of computing inclusions to the maximum accuracy, which in most practical cases is not necessary. A direct consequence of being inefficient is the inability to solve big problems. Finally, existing interval programs are coupled to the environment being used (e.g. Pascal–XSC), thus reducing the portability.

We show how to avoid above drawbacks by using software libraries for solving linear systems:

Reuse of existing software libraries is achieved by decoupling

- the calculation of the approximation using a library routine
- and the verification step.

We add new routines to the existing library procedures that use the computed approximation to construct an inclusion. This implies the subordination of the second phase to the first one, specially to the accuracy of the computed approximation. This strategy leads to a variety of verification methods.

Efficiency: For interval H-matrices, we develop algorithms that compute an inclusion efficiently in the sense that the time needed for the verification is less or of the same magnitude as the computation time for the approximation. Since floating point operations should be executed as fast as possible (using co-processors), we avoid evaluating scalar products to maximum accuracy.

Portability: The algorithms are based only on a floating-point arithmetic conforming the IEEE standard 754 [1]. This guarantees the general applicability of the methods.

The aim of our approach is a collection of verification routines with interfaces that are adapted to specific conventional libraries (ITPACK, LINPACK etc.). A user continues to work in his familiar environment, and he can—but does not have to—guarantee his results economically by calling a further subroutine with a familiar interface.

Table 1 lists notation used in the following sections. Matrix variables are denoted by capital letters, vectors and scalars by small letters. We enclose interval variables in brackets to achieve a better readability. For instance, an interval matrix is given by $[A] \in M_n(I\!I\!R)$, and $A \in [A]$ denotes an arbitrary real matrix $A \in M_n(I\!R)$ contained in $[A]$. Subscripts denote the components of a vector or matrix. Thus, $x = (x_i) \in V_n(I\!R)$ and $A = (A_{ij}) \in M_n(I\!R)$ represent a vector and a matrix, respectively. Superscripts identify a sequence of vectors.

$\mathbb{N}$	:	set of natural numbers
$\mathbb{R}$	:	set of real numbers
$I\mathbb{R}$	:	set of real intervals, i.e. $I\mathbb{R} = \{[a,b] \mid a,b \in \mathbb{R}, a \leq b\}$
$V_n(T)$	:	n-dimensional vectors with components in $T \in \{\mathbb{R}, I\mathbb{R}\}$
$M_n(T)$	:	quadratic $n \times n$ matrices with components in $T \in \{\mathbb{R}, I\mathbb{R}\}$

Table 1: Notation for data types

2 Analysis of classical iterative methods

Starting with an interval matrix $[A] \in M_n(I\mathbb{R})$ and an interval vector $[b] \in V_n(I\mathbb{R})$, we are looking for an inclusion $[x] \in V_n(I\mathbb{R})$ of the solution set $X := \{x \in V_n(\mathbb{R}) \mid Ax = b,\ A \in [A],\ b \in [b]\}$. The usual approach is to transform the problem into a fixed point equation $[x] = [T][x] + [g]$ that allows a successful test for inclusion (according to the theorem of Brower), i.e. there exists a $[x] \in V_n(I\mathbb{R})$ with $[T][x] + [g] \subseteq [x]$.

The success of the inclusion test is characterized by the following theorem proved in [13]:

Theorem 1. *Given $[T] \in M_n(I\mathbb{R})$ and the $\delta-$neighborhoods of $0 \in V_n(\mathbb{R})$ $U_\delta(0) \subseteq [\epsilon]^{k+1} \subseteq [\epsilon]^k,\ k = 1,\ 2,\ 3,\ \ldots$ Choose any starting value $[x]^0 \in V_n(I\mathbb{R})$, and perform the iteration $[x]^{k+1} := ([T][x]^k + [g]) + [\epsilon]^k,\ k \geq 0$. Then there exists a $k \in \mathbb{N}$ with $[x]^{k+1} \subseteq [x]^k$ if and only if $\rho(|[T]|) < 1$.*

Standard books of iterative methods contain several statements regarding the condition $\rho(|T|) < 1$. For example, Varga [14] devotes several chapters to M-matrices and regular splittings. They lead to convergent methods with non-negative iteration matrices T, which obviously fulfill above condition. In the following, we give the necessary definitions and results. For further reading, we recommend [7, 12, 14].

Definition 1. *Given $A,\ M,\ N \in M_n(\mathbb{R})$, $A = M - N$ is called a regular splitting of the matrix A if M is nonsingular with $M^{-1} \geq 0$, and $N \geq 0$.*

Definition 2. *A matrix $A \in M_n(\mathbb{R})$ is a M-matrix, if $a_{ij} \leq 0$ $(i \neq j)$, A is nonsingular, and $A^{-1} \geq 0$.*

Definition 3. *Let* $[A] = ([a]_{ij}) \in M_n(I\!R)$*:*

1. *The point matrix* $\langle [A] \rangle = (\alpha_{ij}) \in M_n(I\!R)$ *defined by*

$$\alpha_{ij} = \begin{cases} inf\{|a_{ii}| \mid a_{ii} \in [a]_{ii}\} & \text{for } i = j \\ -|[a]_{ij}| & \text{otherwise} \end{cases}$$

   is called the comparison matrix of $[A]$*.*

2. $[A]$ *is an interval H-matrix, if* $\langle [A] \rangle$ *is a M-matrix.*

Since important classes of matrices (e.g. strict or irreducible diagonal dominant matrices, symmetric matrices which have a positive definite comparison matrix) are H-matrices[1], the following lemma allows us to define simple, but effective verification algorithms.

Lemma 1. *Let* $A \in M_n(I\!R)$ *be a H-matrix. Let* $\omega_0 \in I\!R$ *be defined by*

$$\omega_0 = \frac{2}{1 + \rho(|J|)}$$

where J *is the matrix resulting from the Jacobi method. Then* $\omega_0 > 1$*, and the Jacobi, Gauss-Seidel, and SOR methods (for* $0 < \omega < \omega_0$*) are convergent.*

Proof: The proof is trivial for the Jacobi method; the rest is shown in [7, pp. 169–171]. □

The proof shows $\rho(|J|) < 1$, $\rho(|G|) < 1$, and $\rho(|H(\omega)|) < 1$ for $0 < \omega < \omega_0$.

3 Symmetric inclusions

3.1 Reducing the runtime

The idea of using symmetric intervals is not new. Collatz [3] already used this approach to reduce the computation time. The following lemma

[1] An H-matrix is a point matrix $A \in M_n(I\!R)$ that fulfills Definition 3.

proved in [6] shows how to bound the solution set of an interval linear system (with H-matrices) by solving a single linear system with point data.

Lemma 2. *Let $[A] \in M_n(I\mathbb{R})$ be an interval H-matrix, and $[b] \in V_n(I\mathbb{R})$. Let $z \in V_n(\mathbb{R})$ be the solution of $\langle [A] \rangle z = |[b]|$. Then every $x \in \{x \mid Ax = b,\ A \in [A],\ b \in [b]\}$ satisfies $|x| \leq z$.*

Now let $u \in V_n(\mathbb{R})$ be an approximate solution of $Ax = b$ with $A \in [A]$ and $b \in [b]$, and let $bd \in V_n(\mathbb{R})$ be an upper bound for the solution of

$$\langle [A] \rangle z = |[b] - [A]u| \,.$$

Then $u + [-bd, bd]$ is an inclusion of the solution set of $[A][x] = [b]$. The computation of bd can be done using any regular splitting $\langle [A] \rangle = \langle M \rangle - |N|$ and iterating as shown in Table 2. Upon termination, $u + [-z^{k+1}, z^{k+1}]$ is an inclusion of the solution set X.

Choose	$0 \leq z^0 \in V_n(\mathbb{R})$
Iterate	$z^{k+1} := \langle M \rangle^{-1} \lvert N \rvert z^k + \langle M \rangle^{-1} \lvert [b] - [A]u \rvert$
until	$z^{k+1} \leq z^k$

Table 2: Computing a symmetric inclusion for interval H-matrices

Note that z^{k+1} is computed without interval operations once $|[b]-[A]u|$ is available. In Section 3.2, we show how to bound $|[b] - [A]u|$ without interval arithmetic. Thus the total execution time can be halved.

For usual splittings (Jacobi, Gauss-Seidel, SOR methods) the system $\langle M \rangle z = r$ can be solved in a sufficiently "simple" way. In these cases, computing an upper bound of z^{k+1} can be achieved by setting the upward rounding mode when evaluating the expression that defines z^{k+1}. In addition we should comment that, as stated in Theorem 1, an inflation has to be performed. In [6] it is shown that the well-known epsilon inflation is not suitable, and a new inflation strategy is introduced.

3.2 Reducing the memory requirement

Having used symmetric intervals to reduce the runtime, we consequently consider a possible reduction of the memory requirements for interval in-

put data. Table 3 lists the data involved in each step of the inclusion computation.

Phase	Input	Output
1. Approximation	$A \in [A]$, $b \in [b]$	$u \in V_n(\mathbb{IR})$
2. Defect	$[A]$, $[b]$, u	$d = \lvert[b] - [A]u\rvert \in V_n(\mathbb{IR})$
3. Verification	$\langle[A]\rangle \in M_n(\mathbb{IR})$, d	$z \in V_n(\mathbb{IR})$, with $X \subseteq u + [-z, z]$

Table 3: Data needed for symmetric inclusions using interval H-matrices

Let us start with the handling of the matrices. Notice that in the first and third steps, only a point matrix is needed. Since we can choose in the approximation phase any matrix contained in the interval input, it is reasonable to choose $A \in [A]$ such that $\langle[A]\rangle$ is defined by taking the absolute values (see Definition 3). Therefore, we can use the same matrix A in the first and third phases.[2]

The computation of the defect is more complicated since we need all of the information contained in $[A]$. Attempts to avoid the storage of the interval matrix lead to a loss of information, i.e. accuracy. For the moment, let $[b] = b \in V_n(\mathbb{IR})$. Then for all $\widehat{A} \in [A]$, we have

$$|b - \widehat{A}u| = |b - Au + (A - \widehat{A})u| \leq |b - Au| + |A - \widehat{A}||u| \leq |b - Au| + d([A])|u|$$

where $d([A])$ is the diameter matrix of $[A]$. So far, we have not gained anything since A and $d([A])$ require the same amount of memory as $[A]$. Considering that in many cases intervals occur because of converting decimal into binary data or measurement errors in experiments, we assume that all components have a similar relative error. Then the diameter matrix can be estimated by $d([A]) \leq m_A|A|$ with $0 \leq m_A \in \mathbb{IR}$. This means for the defect:

$$|b - [A]u| \leq |b - Au| + m_A|A||u|.$$

Hence, the defect can be bounded by the same matrix $A \in M_n(\mathbb{IR})$, which suffices for all three phases in Table 3. This way we have halved the memory requirements for the matrix.

[2]This choice reduces not only the memory requirements, but usually also the accuracy compared with the center matrix of $[A]$. The results in Section 5 show that this accuracy loss is not important.

The handling of the interval right hand sides follows a similar pattern. For the approximation phase, we can choose any $b \in [b]$. For the defect, we easily get

$$|[b] - [A]u| \leq m_b * |b| + |b - Au| + m_A|A||u|$$

with $0 \leq m_b \in \mathbb{IR}$, and $d([b]) \leq m_b|b|$.

Summarizing, we have shown how to map intervals (in the matrix and in the right hand side) to point data and an additional number.[3] This makes the adaptation of the verification routines to the given library an easy task.

Before doing so, we should remark that the user is still responsible for generating the correct data. He must compute sequentially the elements $[a]_{ij}$ (using interval arithmetic) and determine the values $a_{ij} \in [a]_{ij}$ and m_{ij} with

$$d([a]_{ij}) \leq m_{ij} * |a_{ij}|.$$

Then $m_A := \max\{m_{i,j} \mid 1 \leq i, j, \leq n\}$. The user should not determine the values a_{ij} and m_{ij} by hand, but by calling special routines provided in an interval library (see [6]). Two different routines are necessary to compute the elements of the comparison matrix (see Definition 3). A further routine treats the right hand side.

4 Extension of ITPACK

As stated in [9, p. 74]: "It is not usual to find library subroutines for iterative solution methods. However, a collection of such routines can be found in ITPACK, ..." This package provides basic methods (such as Jacobi and SOR iteration) that are accelerated by semi-iterative or conjugate gradient techniques. The reference manual [10] points out that "...ITPACK routines can be called with any linear system containing positive diagonal elements, they are most successful in solving systems with symmetric positive definite or mildly nonsymmetric coefficient matrices." Matrices are stored in the well-known "compressed sparse row format," which means

[3] This estimation is very inaccurate if $[A]$ or $[b]$ have elements containing 0 without being equal to [0]. For instance, m_A has to be 2 if $[a]_{ij} = [-\epsilon, \epsilon]$ for some i, j. This does not occur in our examples. In other cases, another estimate is necessary.

that the matrix

$$\begin{pmatrix} 11.0 & 0.0 & 0.0 & 14.0 & 15.0 \\ 0.0 & 22.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 33.0 & 0.0 & 0.0 \\ 14.0 & 0.0 & 0.0 & 44.0 & 45.0 \\ 15.0 & 0.0 & 0.0 & 45.0 & 55.0 \end{pmatrix}$$

is represented by three arrays:

$$\begin{aligned} A &= [11.0, 14.0, 15.0, 22.0, 33.0, 14.0, 44.0, 45.0, 15.0, 45.0, 55.0], \\ JA &= [1, 4, 5, 2, 3, 1, 4, 5, 1, 4, 5], \\ IA &= [1, 4, 5, 6, 9, 12]. \end{aligned}$$

These three parameters are part of the interface of any of the seven routines provided by ITPACK:

```
subroutine name(n, IA, JA, A, b, u, iwksp, nw, wksp, iparm,
                                                rparm, ier).
```

Most of the remaining parameters are self-explanatory; `n` is the dimension, `b` is the right hand side, and `u` is the approximation. Temporary results are stored in the arrays `iwksp` and `wksp` (of size `n+1` and `nw`). Runtime errors are signaled via the last parameter. The 12-element integer array `iparm` is mostly used to enter values controlling the functionality of the routine, such as the maximum number of iterations, etc. The 12-element floating-point array `rparm` is mostly used to return values like the estimated number of correct digits. ITPACK offers a procedure named `DFAULT` that sets defaults for all 24 values in `iparm` and `rparm`.

The interface of a verification routine is given by

```
long name(n, IA, JA, A, b, u, iparm, rparm, bound).
```

The name of this function is constructed by the following convention:

ElementType_MatrixType_KindOfInclusion_MethodUsed.

ElementType expresses whether the matrix contains intervals. MatrixType distinguishes the class of the matrix (M-matrix, H-matrix etc.). KindOfInclusion tells whether a symmetric or a nonsymmetric inclusion is computed. Finally, MethodUsed names the iterative method (Jacobi, Gauss-Seidel, etc.). Thus the routine

```
long I_M_S_Jac(n, IA, JA, A, b, u, iparm, rparm, bound)
```

computes a symmetric inclusion $u+[-bound, bound]$ for interval M-matrices using the Jacobi method. It is important to note that

- The function returns `1` if the inclusion was successful, otherwise `0`.
- Intervals on the right hand side are handled via $b \in V_n(\mathbb{IR})$ and $m_b \in \mathbb{IR}$ satisfying $d([b]) \leq m_b|b|$. The value of m_b is passed to the function via `rparm[5]`.
- Intervals in the matrix are handled via $A \in M_n(\mathbb{IR})$ and $m_A \in \mathbb{IR}$ satisfying $d([A]) \leq m_A|A|$. The value of m_A is passed to the function via `rparm[6]`.

The remaining parameters are the same as in the ITPACK routines. We also supply a procedure `DFAULT_V` that sets the arrays `iparm` and `rparm` to default values (see [6]). Since the approximation and verification routines look almost the same and are handled in a similar way, a potential user will not be discouraged from verifying his results.

5 Results

5.1 Example 1

The first example uses the matrix which arises from the discretization of the well-known Dirichlet problem $\Delta u = 0$ on the unit square $\Omega = \{(x, y)|0 < x, y < 1\}$ with boundary conditions [14, p. 202–205]:

$$A = \overbrace{\begin{pmatrix} B & C & & \\ C & \ddots & \ddots & \\ & \ddots & \ddots & C \\ & & C & B \end{pmatrix}}^{n},$$

with

$$B = \underbrace{\begin{pmatrix} 4 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{pmatrix}}_{\sqrt{n}} \quad \text{and} \quad C = \underbrace{\begin{pmatrix} -1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & -1 \end{pmatrix}}_{\sqrt{n}}.$$

To test the inclusion routines, we construct a linear system $Ax = [b]$ such that the i-th component of the true solution is $x_i = \frac{1}{i}$. We use interval arithmetic to compute an inclusion $[b] \in V_n(I\mathbb{R})$ of Ax. We then choose $b \in [b]$ and determine a number $m_b \in \mathbb{R}$ such that $d([b]) \leq m_b|b|$. With these values, we use ITPACK and the verification routines to achieve an inclusion. For a more realistic problem, see Section 5.2.

All following computations are performed in IEEE double format on a Macintosh IIcx. The approximation is computed with the ITPACK routine `SSORSI`. The verification is computed with `P_M_NS_GS`. The routine `SSORSI` is in all cases the fastest of all the ITPACK routines, i.e. `JCG`, `JSI`, `SOR`, and `SSORCG` each need more time to compute an approximation of the same quality. Table 4 compares for different dimensions the runtime of `SSORSI` (requiring 10 correct digits) and `P_M_NS_GS`.

dimension	100	484	900	1444	1764	2116	2916	40000
approximation	61	382	762	1413	1663	2118	3312	68396
verification	16	150	352	655	855	1283	1858	76252

Table 4: Runtime of the approximation and the verification

The runtime is measured in ticks, which correspond to $\frac{1}{60}$ seconds. Table 4 shows for increasing dimension how the verification time approaches and overtakes the approximation time. This behavior is not bad if we consider that ITPACK does not provide the desired accuracy. For the dimension 40000, an approximation with only 3 correct digits was computed (we required 10). The worst and best inclusions guarantee 2.4 and 12.8 digits, respectively.

To illustrate this point, we run the following test. For fixed dimension 2916, we demanded from ITPACK approximations of different accuracies.

The results are condensed in Table 5, where the error of the computed approximation to the vector $x_i = \frac{1}{i}$ (we will call this the "exact" error), the accuracy of the best and worst inclusions, as well as both ITPACK estimations are given. The values represent the number of digits which are obtained using the negative of the logarithm base ten. Negative values mean that no digits are correct.

digits required	5	7	9	11	13
"exact" error	-2.9	-0.5	1.7	4	6.1
worst inclusion	-3.4	2.8	1.1	2.9	5.3
best inclusion	6.6	9	11.1	12.4	14.3
estimation # 1	5.1	7	9.2	11.3	13.6
estimation # 2	7.3	9.7	11.9	13.2	15.4

Table 5: Accuracy achieved vs. accuracy required

The worst inclusion is clearly influenced by the "exact" error, while the best inclusion is always close to the required accuracy and both estimations. An explanation for this behavior, specially for the misleading information delivered by ITPACK, can be obtained when considering for example the second estimation. Given an approximation u, the error is estimated by

$$\frac{\|x-u\|_2}{\|x\|_2} \approx \frac{\|b-Au\|_2}{\|b\|_2}.$$

Although the reference manual states that the negative of the logarithm base ten of this value "... is the approximate number of digits..." [10, p. 9], it should be stressed that it is not a relative but an absolute accuracy, which is only valid for the large components of the solution (see [6]). As shown by the "exact" error, smaller components may have considerably fewer correct digits. This discrepancy is determined automatically when intervals are used.

The considerations above are confirmed by setting $x_i = 1$. The gap between the best and worst inclusions is narrower, and the inclusion guarantees at least 2 to 3 digits in all. For the dimension 40000, we obtain (demanding 10 digits from ITPACK) 8 – 10 guaranteed digits. In this case the approximation took 81055, the verification 74350 ticks.

5.2 Example 2

The two-dimensional elliptic problem

$$A(x,y)u_{xx} + C(x,y)u_{yy} + D(x,y)u_x + E(x,y)u_y + F(x,y)u = G(x,y)$$

on the unit square $\Omega = \{(x,y)|0 < x,y < 1\}$ with the linear boundary conditions of third kind

$$\begin{aligned}
\text{left}: &\quad \alpha_0(y)u_x(0,y) + \beta_0(y)u(0,y) = \gamma_0(y),\\
\text{right}: &\quad \alpha_1(y)u_x(1,y) + \beta_1(y)u(1,y) = \gamma_1(y),\\
\text{down}: &\quad \delta_0(x)u_y(x,0) + \epsilon_0(x)u(x,0) = \lambda_0(x),\\
\text{up}: &\quad \delta_1(x)u_y(x,1) + \epsilon_1(x)u(x,1) = \lambda_1(x),
\end{aligned}$$

was solved in [2] with ACRITH by partitioning Ω into an equidistant mesh using the step width $h = \frac{1}{m}$. This delivers a total of $(m+1)^2$ points or unknowns in the plane. Interior points are handled by inserting second order central differences into the partial differential equation, which leads to $(m-1)^2$ linear equations. The boundaries (without the 4 corners) are approximated using forward/backward differences. This gives $4(m-1)$ additional equations. Setting the corners to zero, we obtain the last 4 equations of the quadratic linear system (for details see [2, 6]).

The following three examples studied in [2]

1) $-(u_{xx} + u_{yy}) = 0,\ u(x,0) = u(0,y) = 0,\ u(x,1) = u(1,y) = 1$

2) $-(u_{xx} + u_{yy}) = \frac{0.1}{x^2+y^2},\ u(x,0) = u(0,y) = u(x,1) = u(1,y) = 0$

3) $-(u_{xx} + u_{yy}) = e^{-(x-\frac{1}{2})^2-(y-\frac{1}{2})},\ u(x,0) = u(0,y) = u(x,1) = u(1,y) = 0$

lead to exactly representable point matrices (in binary arithmetic). The right hand side has to be included by intervals. Table 6 shows for the first example and different step widths the ratio of the computation time for the verification and the approximation (using ITPACK and its extension). For the approximation, we chose the ITPACK routine `SSORSI`, which was again the fastest of the five ITPACK routines.

Note that $h = \frac{1}{2^8}$ leads to a system with 66049 equations. In [2] results are given just for $h = \frac{1}{2^3} = \frac{1}{8}$. In this case the ratio between verification and approximation time equals 16.8!

step width	$\frac{1}{2^3}$	$\frac{1}{2^4}$	$\frac{1}{2^5}$	$\frac{1}{2^6}$	$\frac{1}{2^7}$	$\frac{1}{2^8}$
$\frac{\text{verification}}{\text{approximation}}$	0.14	0.23	0.30	0.48	0.32	0.38

Table 6: Relation of the approximation and verification times

ACRITH guaranteed 5 digits for $h = \frac{1}{8}$, while the ITPACK extension guarantees in all cases 7 – 10 digits. To achieve this, we require 10 correct digits from the approximation routine in ITPACK. Similar (accuracy and runtime) results are valid for all other examples.

The remaining examples studied in [2]

4) $-(e^y u_{xx} + e^x u_{yy}) = 0,\ u(x,0) = u(0,y) = 0,\ u(x,1) = u(1,y) = 1$

5) $-(e^{-y} u_{xx} + e^{-x} u_{yy}) = 0,\ u(x,0) = u(0,y) = 0,$
$u(x,1) = \frac{1}{1-0.9x},\ u(1,y) = \frac{1}{1-0.9y}$

6) $-(pu_x)_x - (pu_y)_y + u = 0$ with $p(x,y) = e^{x^2+y^2}$,
$u(x,0) = u(0,y) = 0,\ u(x,1) = \frac{x}{1-0.9x},\ u(1,y) = \frac{y}{1-0.9y}$

lead to interval matrices which are treated by computing a real number m_A with $d([A]) \leq m_A|A|$ (see Section 3.2). The right hand side is included in the same way.

Table 7 shows excellent results for the fourth example, where 10 correct digits were demanded from ITPACK. The two other examples lead to similar values.

step width	$\frac{1}{2^3}$	$\frac{1}{2^4}$	$\frac{1}{2^5}$	$\frac{1}{2^6}$	$\frac{1}{2^7}$
$\frac{\text{verification}}{\text{approximation}}$	0.23	0.21	0.32	0.33	0.36
worst inclusion	9	9	9	9	8

Table 7: Runtime and accuracy of the inclusion

6 Conclusion and further work

We have seen that in the case of interval H-matrices that it is possible to extend software libraries (for solving linear systems) by verification routines for a highly efficient inclusion of the solution basing on approximations delivered by the library routines. Our extension to ITPACK consists of 6 routines (each 300 lines of C code). Their application requires

- ITPACK, which consists of 9000 lines of FORTRAN. It is public domain software and can be obtained via `netlib`.
- a floating-point arithmetic that provides operations with directed roundings (this is satisfied by all co-processors supporting the IEEE standard 754 [1]) and a high level language that allows control of the rounding direction.

The efficiency of the verification routines is excellent for various reasons. The use of symmetric input data and symmetric inclusions nearly halves the execution time. Furthermore, a new inflation variant is built in that accelerates the verification step considerably. Finally, we have refused to supply any (software) implementation of the exact dot product. Basing only on hardware operations we may loose some accuracy, but we gain all the power of the existing co-processors.

All the results have been produced on a Macintosh IIcx containing the processors MC68030 and MC68881 (which provides the IEEE standard 754) and 8 MBytes of RAM. Since ITPACK is written in FORTRAN and our routines in C, two compilers are needed (you can avoid this problem by rewriting the latter in FORTRAN). The C compiler must allow to control the rounding direction. We have listed these requirements on purpose to stress the minimality (and thus portability) of our approach.

In our opinion, further research should follow these paths:

- Development of more efficient verification methods: On one hand, more refined inflation variants should be developed and tested. On the other hand, the convergence of the verification could be accelerated using preconditioners. It should even be possible to reuse the parameters of the acceleration methods computed by ITPACK.

- Extension of libraries based on direct methods: Following a different approach, verification methods basing on the LU or Cholesky factorizations, etc., can be developed [6] to extend popular libraries as LINPACK or LAPACK.

- Implementation on parallel computers: First experiments have shown that it is possible to transfer successfully our approach to parallel computers. Promising results with the Laplacian matrix for dimensions up to 1,000,000 are given in [6].

References

[1] American National Standards Institute/Institute of Electrical and Electronic Engineers, *A standard for binary floating-point arithmetic.* ANSI/IEEE Std 754–1985, New York, 1985.

[2] Ames, W. F. and Nicklas, R. C. *Accurate elliptic differential equation solver.* In: Miranker, W. L. and Toupin, R. A. (eds), "Accurate Scientific Computations", Lecture Notes in Computer Science, Vol. 235, Springer Verlag, 1985, pp. 70–81.

[3] Collatz, L. *Funktionalanalysis und numerische Mathematik.* Springer-Verlag, Berlin-Heidelberg-New York, 1964.

[4] Dongarra, J. J. and Grosse, E. *Distribution of mathematical software via electronic mail.* Communications of the ACM **30** (5) (1987), pp. 403–407.

[5] Dongarra, J. J., Moler, C. B., Bunch, J. R., and Stewart, G. W. *LINPACK Users' Guide.* SIAM, Philadelphia, 1979.

[6] Falcó Korn, C. *Die Erweiterung von Software-Bibliotheken zur effizienten Verifikation der Approximationslösung linearer Gleichungssysteme.* PhD Thesis, University Basel, 1993.

[7] Frommer, A. *Lösung linearer Gleichungssysteme auf Parallelrechnern.* Vieweg Verlag, Braunschweig, 1990.

[8] Hager, W. W. *Applied numerical linear algebra.* Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[9] Kahaner, D., Moler, C., and Nash, S. *Numerical methods and software.* Prentice Hall, Englewood Cliffs, May 1989.

[10] Kincaid, D. R., Respess, J. R., Young, D. M., and Grimes, R. G. *ITPACK 2C: a FORTRAN package for solving large sparse linear systems by adaptive accelerated iterative methods.* ACM Transactions on Mathematical Software **8** (1982), pp. 302–322.

[11] Klatte, R., Kulisch, U., Neaga, M., Ratz, D., and Ullrich, Ch. *PASCAL–XSC: language reference with examples.* Springer Verlag, Berlin-Heidelberg, 1991.

[12] Ortega, J. M. *Numerical analysis: a second course.* SIAM, Philadelphia, 1990.

[13] Rump, S. M. *Lineare probleme.* In: Kulisch, U. (ed.) "Wissenschaftliches Rechnen mit Ergebnisverifikation — Eine Einführung". Vieweg Verlag, 1989, pp. 129–135.

[14] Varga, R. S. *Matrix iterative analysis.* Prentice-Hall, Englewood Cliffs, New Jersey, 1962.

Institut für Informatik, Universität Basel
Mittlere Str. 142, CH-4056 Switzerland
e-mail: carlos@ifi.unibas.ch
e-mail: ullrich@ifi.unibas.ch