Approximate inverses of almost singular matrices still contain useful information

S. M. Rump, Hamburg

Abstract. It is well-known that, roughly spoken, a matrix inversion on a computer working in base B with t digits precision in the mantissa applied to a matrix of condition B^k produces approximately $t-k$ correct digits of the inverse. For condition $\gg B^t$ one might conclude that an approximate inverse contains virtually useless information.

In this note we will show that the latter is not true. An approximate inverse may still be useful, e.g. as a preconditioner. An extended set of examples show that preconditioning a matrix using an approximate inverse (computed in t digits precision) lowers the condition number by a factor B^t. As an example we develop an algorithm for solving systems of linear equations up to condition B^{2t} strictly using t digits precision for all calculations and only allowing for double precision accumulation of inner products.

0. Introduction.

It is a commonly accepted rule of thumb (cf. [5]) that Gaussian elimination on a floating-point computer with base B and t digits in the mantissa produces approximately $t - k$ correct digits of the solution for a linear system $Ax = b$ of condition B^k. Using double precision residual correction adds in every step approximately $t - k$ correct digits up to a final accuracy of the order of the relative rounding error.

This rule of thumb holds as long as $k < t$. For $k \geq t$ Gaussian elimination (or matrix inversion) produces an approximation with no correct digits and residual iteration diverges because $\rho(I-U^{-1}L^{-1}A) \geq 1$. It is also common belief that for such an ill-conditioned linear system L and U (resp. an approximate inverse) contains essentially useless information and that a matrix of condition greater than B^t "cannot be inverted" in t digits precision of base B.

In [6] (cited in [1]) this has been expressed in popular terms by Wilkinson:

> "If A is almost singular to working precision the first solution has no correct significant and the same is true for all subsequent refined solutions. A is too ill-conditioned (possibly singular) for solution to be possible without working to higher precision in the factoring."

In this note we will demontrate that an LU-factorization, or better, an approximate inverse of a matrix of condition $\gg B^t$ still contains useful information. For example we give an algorithm showing that a double precision accumulation of inner products suffices to solve linear systems up to condition B^{2t} in t digits precision of base B to high accuracy. This algorithm uses information within a totally incorrect and seemingly useless LU-decomposition resp. inverse of A to produce a suitable preconditioning of the linear system $Ax = b$.

As a general, empirical observation preconditioning lowers the condition by a factor of B^t, virtually independent of the original condition number of the matrix or the precision in use. This observation has been verified by many computational examples. The approach is heuristic, no analysis is given.

We want to stress that the purpose of this note is to show that even for $\text{cond}(A) >> B^t$ a single precision approximate inverse still contains (hidden) useful information.

1. Preconditioning of nearly singular matrices.

Consider a matrix A being exactly representable in the floating-point screen in use and some standard algorithm for matrix inversion. Suppose this algorithm produces an approximate (floating-point) inverse R. We want to monitor the ratio of condition numbers of A und $R \cdot A$, the latter multiplication being executed precisely (without rounding error).

For this purpose a decimal computer arithmetic with specifyable precision was simulated. It does not seem to be trivial to find matrices being small enough in size and exactly representable in few decimal digits but with high condition number. Among five "standard" types of matrices

(Pascal, Hilbert, inverse Hilbert, Zielke, inverse Zielke) the maximum condition number provided the matrix is exactly representable in 4 decimal digits was $4.0_{10}8$. For examples with those matrices see section 4.

Therefore we used some extremely ill-conditioned matrices especially constructed for this purpose (cf. [3]). The following matrix

$$A = \begin{bmatrix} 6566 & -5202 & -4040 & -5524 & 1420 & 6229 \\ 4104 & 7449 & -2518 & -4588 & -8841 & 4040 \\ 5266 & -4008 & 6803 & -4702 & 1240 & 5060 \\ -9306 & 7213 & 5723 & 7961 & -1981 & -8834 \\ -3782 & 3840 & 2464 & -8389 & 9781 & -3334 \\ -6903 & 5610 & 4306 & 5548 & -1380 & 3539 \end{bmatrix}$$

has a condition number

$$\operatorname{cond}(A) = \|A\|_1 \cdot \|A^{-1}\|_1 = 1.2_{10}25 \text{ .}$$

According to Skeel [4] the sensitivity of the matrix w.r.t. inversion, that is the relative change of elements of the inverse under small perturbations of the matrix, can be explicitely calculated. For our example the minimum sensitivity s satisfies

$$s = \min_{i,j} \lim_{\varepsilon\to 0} \left\{ \frac{|\tilde{A}^{-1}_{ij} - A^{-1}_{ij}|}{|A^{-1}_{ij}|} \;\middle|\; |\tilde{A}-A| \le \varepsilon\cdot|A| \right\} =$$

$$= \min_{i,j} \frac{\{|A^{-1}|\cdot|A|\cdot|A^{-1}|\}_{ij}}{|A^{-1}_{ij}|} > 5.01 \cdot 10^{24} \qquad (1.1)$$

Next we calculate an approximate inverse R using Gauß-Jordan with precision p (i.e. p decimal digits in the mantissa). In the following table the condition number of $R \cdot A$ and the ratio $\mathrm{cond}(A)/\mathrm{cond}(R \cdot A)$ is displayed. For comparison we also display the same numbers for R^* which is the exact inverse of A rounded to p decimal digits.

p	cond(R·A)	cond(R*·A)	$\frac{\mathrm{cond}(A)}{\mathrm{cond}(R \cdot A)}$	$\frac{\mathrm{cond}(A)}{\mathrm{cond}(R^* \cdot A)}$
13	$7.7_{10}13$	$9.3_{10}15$	$1.5_{10}{+11}$	$1.3_{10}{+9}$
12	$1.3_{10}15$	$2.2_{10}17$	$9.2_{10}{+9}$	$5.3_{10}{+7}$
11	$6.6_{10}15$	$2.0_{10}17$	$1.8_{10}{+9}$	$5.8_{10}{+7}$
10	$3.0_{10}17$	$1.5_{10}24$	$3.9_{10}{+7}$	$7.7_{10}{+0}$
9	$3.1_{10}17$	$2.8_{10}26$	$3.8_{10}{+7}$	$4.2_{10}{-2}$
8	$9.1_{10}18$	$4.2_{10}27$	$1.3_{10}{+6}$	$2.8_{10}{-3}$
7	$2.8_{10}20$	$7.2_{10}26$	$4.1_{10}{+4}$	$1.6_{10}{-2}$
6	$7.2_{10}20$	$6.1_{10}25$	$1.6_{10}{+4}$	$1.9_{10}{-1}$
5	$4.6_{10}21$	$4.6_{10}27$	$2.5_{10}{+3}$	$2.5_{10}{-3}$
4	$3.4_{10}22$	$1.6_{10}23$	$3.4_{10}{+2}$	$7.5_{10}{+1}$

Table 1. Improving the condition number through preconditioning, rounding to nearest

Needless to say that an approximate inverse computed in 4 decimal digits of a matrix of condition 10^{25} has nothing to do with the correct inverse. But although it differs by approximately 20 orders of magnitude from A^{-1} it still improves the condition number when used as a preconditioner. On the other hand the rounded precise inverse is a poor preconditioner and for $p \leq 9$ it even makes the condition worse. This result would be expected.

It is reasonable to look at the condition number of $R \cdot A$ with exact evaluation of the product. A calculation of $R \cdot A$ using the working precision in which R was computed would yield a significantly better condition number and ratio because of the finite floating-point grid.

The results of table 1 are obtained using a decimal arithmetic with p digits precision and rounding to nearest. The computation is very sensitive and therefore a difference should be visible when changing the rounding mode. The following table displays the results obtained when rounding towards zero. As can be seen, the results for preconditioning using the Gauß-Jordan inverse are the same whereas the results for R* (the rounded A^{-1}) change significantly. This suggests a certain stability of the entire computational process although the individual components of the approximate inverse R react extremely sensitive to smallest perturbations or changing the rounding mode. This observation was verified in many examples.

p	cond(R·A)	cond(R*·A)	$\frac{\text{cond(A)}}{\text{cond(R}\cdot\text{A)}}$	$\frac{\text{cond(A)}}{\text{cond(R*}\cdot\text{A)}}$
13	$7.7_{10}13$	$1.7_{10}16$	$1.5_{10}+11$	$6.7_{10}+8$
12	$1.3_{10}15$	$2.7_{10}17$	$9.2_{10}+9$	$4.3_{10}+7$
11	$6.6_{10}15$	$2.8_{10}22$	$1.8_{10}+9$	$4.1_{10}+2$
10	$3.0_{10}17$	$1.2_{10}24$	$3.9_{10}+7$	$9.6_{10}+0$
9	$3.1_{10}17$	$1.1_{10}25$	$3.8_{10}+7$	$1.0_{10}+0$
8	$9.1_{10}18$	$1.4_{10}27$	$1.3_{10}+6$	$8.4_{10}-3$
7	$2.8_{10}20$	$1.1_{10}28$	$4.1_{10}+4$	$1.1_{10}-3$
6	$7.2_{10}20$	$1.4_{10}30$	$1.6_{10}+4$	$8.5_{10}-6$
5	$4.6_{10}21$	$1.4_{10}30$	$2.5_{10}+3$	$8.4_{10}-6$
4	$3.4_{10}22$	$8.3_{10}31$	$3.4_{10}+2$	$1.4_{10}-7$

Table 2. Improving the condition number through preconditioning, rounding towards zero

Similar computations have been performed with many matrices with different condition numbers and varying precisions. All results confirm the rule of thumb that preconditioning improves the condition number by a factor B^t when using t digits precision and base B.

2. Application to linear systems. In the following an algorithm with preconditioning for solving ill-conditioned systems of linear equations will be introduced. To be perfectly clear, every detail of the algorithm will be specified. In the following we use a MATLAB-notation so that

the algorithm can be implemented and tested very quickly.

Unfortunately MATLAB does not offer the possibility to work with different precisions. Therefore we first simulate a rounding to a specified precision. For demonstration purposes we use base 10. The following algorithm rounds the input x to **precision** decimal digits:

```
function res = rnd(x);
      a = abs(x); k = round(log(a + (a==0))/log(10) - .5);
      c = 10 . ^ (precision - k - 1);
      res = sign(x) .* (round(a.*c)./c);
```

<u>Algorithm 3.</u> Rounding to **precision** decimal digits

Note that **precision** is a global variable and that the addition of a==0 covers the case x = 0 (not very elegant but it works). The algorithm works for vectors and matrices as well rounding every component to **precision** decimal digits.

Next we define a double precision summation of inner products by means of the following algorithm:

```
function res = mul(a,b);
      [l,m] = size(a); [m,n] = size(b); res = nulls(l,n);
      precision = 2*precision;
      for i = 1:m,
          res = rnd(res + rnd(a(:,i)*b(i,:)));
      end;
      precision = precision/2; res = rnd(res);
```

<u>Algorithm 4.</u> Double precision accumulation of inner products

nulls denotes a structure of zero elements nulls(1,n)=0*ones(1,n). The algorithm is designed for (compatible) vectors and matrices a and b using rank-1-corrections. Note that all multiplications and summations of the inner product are performed in double precision with one final rounding to single precision. The global variable **precision** is set to its old value after leaving the function. The rounding of the product a(:, i)*b(i, :) is added to avoid misinterpretations. It is not necessary because the product is exactly representable in 2***precision** digits.

For a linear system Ax = b in n unknowns the residual b-A*x can be calculated with double precision accumulation of the inner products by

```
     residue = mul([ b a ] , [ 1 ; -x ]);
```

The algorithm works properly as long as 2***precision** is less than the working precision.

For our algorithm we require some standard floating-point algorithm for the solution of a system of linear equations. However, several experiments showed that LU-decomposition with backward substitution yields significantly weaker results than multiplication of the right hand side by an approximate inverse. Knowing that this method is highly not recommendable in numerical analysis we nevertheless formulate our algorithm in that way.

In the following we assume an algorithm for matrix inversion to be given strictly performing all operations in precision decimal digits. In our experiments Gauß-Jordan with row-pivoting performed best. For convenience we list this algorithm. It is designed for good readability rather than best performance.

```
function res=gj(A);
  [m,n]=size(A); row=nulls(n,1); col=row;
  for i=1:n,
    pivot=0; np=0;
    for j=1:n,
      if col(j)==0,
        if abs(A(i,j))>pivot,
          np=j; pivot=abs(A(i,j));
        end;
      end;
    end;
    pivot=A(i,np); col(np)=i; row(i)=np;
    A(i,:)=rnd(-A(i,:)/pivot);
    for k=1:n,
      for j=1:n,
        if (k~=i)*(j~=np),
          A(k,j)=rnd(A(k,j)+rnd(A(i,j)*A(k,np)));
        end;
      end;
    end;
    A(:,np)=rnd(A(:,np)/pivot); A(i,np)=rnd(1.0/pivot);
  end;
  i=1;
  while i<=n,
    k=col(i);
    if i~=k,
      d=A(:,k); A(:,k)=A(:,i); A(:,i)=d;
      col(i)=col(k); col(k)=k;
    else
      i=i+1;
    end;
  end;
  for i=1:n,
    A(row,i)=A(:,i);
  end;
  res=A;
```

<u>Algorithm 5.</u> Gauß-Jordan with row-pivoting in precision decimal digits

For very ill-conditioned matrices it happens that all pivots equal zero. In order to avoid this we introduce a small perturbation in case of an exact zero as an intermediate result. Thus the most inner loop would be replaced by

```
Akj = rnd(A(k,j) + rnd(A(i,j) * A(k,np)));
if Akj ~= 0,
  A(k,j)= Akj;
else
  A(k,j)= rnd(10^(-precision)*rand*Akj);
end;
```

Finally we state the linear system solver using the approximate inverse R produced by algorithm 5 as a preconditioner. In ill-conditioned cases R is totally wrong compared to A^{-1} but still good enough to serve as a preconditioning matrix (see below).

```
function x = solve(A,b);
     R = gj(A); S = mul(R,A); S = gj(S);
     x = mul(S, mul(R,b));
     while x "not good enough",
       residue = mul([b A], [ 1 ; -x ]);
       x = rnd(x + mul(S,mul(R,residue)));
     end;
```

<u>Algorithm 6.</u> Solution of very ill-conditioned linear systems

The stopping criterion is discussed later.

<u>3. A simple example.</u> In the following we demonstrate the algorithm by a very simple example. Consider the Zielke-Matrix defined by

$$A_{ij} := \frac{\binom{n+i-1}{i-1} \cdot n \cdot \binom{n-1}{n-j}}{i+j-1} \tag{3.1}$$

Zielke matrices have the nice property that a checkerboard-like sign distribution on A yields its inverse.

In the following we use

precision = 3 and a 4×4-Zielke matrix,

i.e. we are calculating with 3 decimal digits in the mantissa and

$$A = \begin{bmatrix} 4 & 6 & 4 & 1 \\ 10 & 20 & 15 & 4 \\ 20 & 45 & 36 & 10 \\ 35 & 84 & 70 & 20 \end{bmatrix} \tag{3.2}$$

It is

$\mathrm{cond}(A) = 1.82 \cdot 10^4$.

Therefore it should not be possible to invert A in our given precision. We choose some random right hand side $b = (236, -247, -152, 122)^T$ yielding the exact solution

$$A^{-1} \cdot b = \begin{bmatrix} 1696 \\ -4532 \\ 9143 \\ -15928 \end{bmatrix} \tag{3.3}$$

Then algorithm 5 (Gauß-Jordan) yields

$$R = \begin{bmatrix} -21.0 & 43.8 & -33.6 & 9.1 \\ 66.6 & -133.0 & 101.0 & -27.1 \\ -143.0 & 281.0 & -211.0 & 56.5 \\ 259.0 & -504.0 & 376.0 & -100.0 \end{bmatrix} \qquad (3.4)$$

This hardly represents the magnitude of

$$A^{-1} = \begin{bmatrix} 4 & -6 & 4 & -1 \\ -10 & 20 & -15 & 4 \\ 20 & -45 & 36 & -10 \\ -35 & 84 & -70 & 20 \end{bmatrix} .$$

Computing S=R*A with double precision accumulation of the inner products yields

$$S = \begin{bmatrix} 0.5 & 2.4 & 0.4 & 0.2 \\ 7.9 & 8.2 & 10.4 & 2.6 \\ -4.5 & 13.0 & 2.0 & 1.0 \\ 16.0 & -6.0 & 12.0 & 3.0 \end{bmatrix} \qquad (3.5)$$

In a well-conditioned case we would have $R \approx A^{-1}$ and therefore $S \approx I$. In our example S is far from the identity matrix, but it is simpler to invert:

$$\mathrm{cond}(S) = 86.4 .$$

Even in 3 decimal digits precision this should work. We obtained after applying algorithm 5 to S:

$$S = \begin{bmatrix} 0.7650 & 0.0102 & -0.1520 & -0.0091 \\ 0.3410 & 0.0801 & -0.0696 & -0.0695 \\ -1.2000 & 0.5500 & -0.2710 & -0.3080 \\ 1.3900 & -2.0900 & 1.7600 & 1.4700 \end{bmatrix} \qquad (3.6)$$

A precise evaluation of S*R yields

$$S*R = \begin{bmatrix} 3.99 & -5.96 & 3.97 & -0.99 \\ -9.87 & 19.80 & -14.80 & 3.95 \\ 20.80 & -46.60 & 37.20 & -10.30 \\ -39.30 & 92.50 & -76.40 & 21.70 \end{bmatrix}$$

which is not too far from A^{-1}. The spectral radii of the iteration matrices using R resp. S*R as an approximate inverse of A are

$$\rho(I-R*A) = 17.4 \quad \text{and} \quad \rho(I-S*R*A) = 0.011.$$

Accordingly using R alone yields the following residual iterates (each column representing an iterate)

$$\begin{matrix} -9.5_{10}3 & -8.4_{10}4 & -6.5_{10}5 & -4.9_{10}6 & \cdots \\ 3.0_{10}4 & 2.6_{10}5 & 2.0_{10}6 & 1.5_{10}7 & \cdots \\ -6.2_{10}4 & -5.5_{10}5 & -4.2_{10}6 & -3.2_{10}7 & \cdots \\ 1.1_{10}5 & 9.9_{10}5 & 7.6_{10}6 & 5.8_{10}7 & \cdots \end{matrix}$$

whereas algorithm 5 stops after 3 iterations:

$$\begin{matrix} 1690 & 1740 & 1700 & 1700 \\ -4460 & -4670 & -4530 & -4530 \\ 9590 & 9470 & 9130 & 9140 \\ -18300 & -16600 & -15900 & -15900 \end{matrix} .$$

The last iteration is correct to all 3 figures (compare to (3.3)). Note that the iteration only performs corrections in the last (3rd) decimal digit of the mantissa.

As in chapter 1 the computation depends on the rounding mode. In the following we use rounding to nearest in 3 decimal digits with rounding the exact midpoint between two 3-decimal-digit floating-point numbers towards zero (instead of towards plus or minus infinity, as in the previous example), a very minor modification. Using algorithm 4 to calculate R then yields

$$R = \begin{bmatrix} -3.69 & 10.10 & -8.51 & 2.42 \\ 14.60 & -31.70 & 25.30 & -7.05 \\ -34.20 & 69.00 & -52.90 & 14.40 \\ 64.70 & -126.00 & 94.00 & -25.00 \end{bmatrix} . \qquad (3.7)$$

This R is entirely different from (3.4). There is exactly one single case where changing the rounding in the way described affects the result, namely for $i = 1$, $k = 3$, $j = 4$ in algorithm 4 where the exact intermediate result is -7.515 rounded to -7.52 producing (3.4) and rounded to -7.51 producing (3.7). Similar to (3.5) we obtain

$$S = \begin{bmatrix} 0.74 & 0.19 & -0.22 & 0.01 \\ 0.65 & -0.10 & 0.20 & -0.20 \\ -0.80 & 3.90 & 1.80 & 0.80 \\ 3.80 & -1.80 & 2.80 & 0.70 \end{bmatrix}$$

whereas now

$$\text{cond}(S) = 22.0 .$$

Inverting S in itself using algorithm 4 yields

$$S = \begin{bmatrix} 0.855 & 0.210 & -0.0094 & 0.0581 \\ 0.457 & 0.392 & 0.1910 & -0.1140 \\ -1.200 & 0.860 & 0.1330 & 0.1140 \\ 1.340 & -3.650 & 0.0077 & 0.3610 \end{bmatrix}$$

with

$$\rho(I-S*R*A) = 0.005 \; .$$

Finally almost the same residual iteration is produced stopping after 2 iterations with an approximation correct to all 3 figures.

It seems interesting that one single difference in the last digit produces an entirely different approximate inverse but nevertheless both approximations may serve as a good preconditioner.

An attempt to analyse algorithm 6 failes fairly quickly, at least when using traditional techniques. In Wilkinsons backward analysis (see [5]) usually $\text{cond}(A) < B^t$ is assumed. When dropping this assumption the well-known results are no longer true. For example, according to backward error analysis a computed approximate inverse should be the exact inverse of a slightly perturbed matrix. Assuming $R = (A+E)^{-1}$ in our simple example yields

$$E = \begin{bmatrix} -2.2998 & -5.3776 & -4.3400 & -1.2061 \\ -7.9317 & -18.4393 & -14.9137 & -4.1860 \\ -18.5370 & -42.3662 & -34.2926 & -9.6159 \\ -35.5199 & -80.3510 & -64.8960 & -18.1624 \end{bmatrix}$$

with

$$\|E\|_2 = 127.9 \quad \text{where } \|A\|_2 = \|A^{-1}\|_2 = 134.9 \ .$$

Therefore the necessary perturbation of A is of the order of A itself. Moreover

$$\|AR-I\|_2 = 103.4 \quad \text{and } \|I-RA\|_2 = 24.3$$

indicating no convergance of an attempted residual correction.

4. Numerical results.

In the following we list some more numerical results for algorithm 6. As the basic precision we used IEEE 754 single precision, i.e. 24 binary digits precision including implicit one. Double precision accumulation of inner products has been performed using IEEE 754 double precision, i.e. 53 binary digits precision including implicit one. Thus the accumulation of inner products is performed in slightly more than twice single precision. However, extended tests showed that this does not influence the numerical results.

The algorithm is intended to treat very ill-conditioned linear systems. As has been pointed out before it is difficult to find appropriate matrices being exactly representable in floating-point. In the following we use 6 different types of matrices. Pascal-matrices P defined by

$$P_{ij} := \begin{bmatrix} i+j-1 \\ i \end{bmatrix} ,$$

scaled Hilbert-matrices H with

$$H_{ij} := \mathrm{lcm}(1,\ldots,2n-1) \;/\; (i+j-1) ,$$

inverse Hilbert-matrices H^{-1} (inverse of the original Hilbert-matrix with entries $1/(i+j-1)$ implying integer components), Zielke-matrices Z defined by (3.1), inverse Zielke-matrices Z^{-1} where

$$Z^{-1}_{ij} = (-1)^{i+j} \cdot Z_{ij}$$

and special ill-contitioned matrices Q taken from [Ru90]. It is absolutely necessary that all entries of those matrices are exactly representable in the floating-point mesh in use because a single rounding of some component destroys the property of ill-conditioness. In the following table we display the maximum dimension of the first five "standard" matrices up to which they are exactly representable in IEEE 754 single precision, and the corresponding condition numbers.

matrix	n_{max}	condition $\|A\|_1 \cdot \|A^{-1}\|_1$
P	15	$7.0_{10}16$
H	11	$1.2_{10}15$
H^{-1}	6	$2.9_{10}7$
Z	10	$1.1_{10}15$
Z^{-1}	10	$1.1_{10}15$

Table 7. Largest, in single precision representable matrices

The 6-th class of matrices Q is constructed to contain exactly representable matrices of arbitrary high condition number. We use them to explore the scope of applicability of the algorithm.

We use three different right hand sides to test algorithm 6:

$b_1 = e_1$, the first unit vector,

$b_2 = (1, -1, 1, -1, \dots)^T$ and

b_3 = randomly chosen uniformly distributed within $[-10^4, +10^4]$.

The first r.h.s. produces as the solution the first column of the inverse of the matrix. For each of those r.h.s. we apply standard Gaussian elimination (std.) and algorithm 6 (new), both with residual correction with double precision accumulation of inner products. In the first three columns of the following table we display the type of the matrix A, the number of rows and columns n and the condition number (in 1-norm). In the latter six columns the number of iterations are displayed which are necessary such that the relative error

$\|x^k - A^{-1}b\|_\infty / \|A^{-1}b\|_\infty$ against the true solution is less than $2\cdot 10^{-7}$ $(=2^{-23})$ resp. less than $2\cdot 10^{-4}$ (numbers in paranthesis). If in 100 iterations the relative error stays larger than $2\cdot 10^{-4}$ then **div.** is displayed.

			b_1		b_2		b_3	
matrix	n	cond	std.	new	std.	new	std.	new
P	8	$4.0_{10}8$	5	0	6	0	5	0
	9	$5.8_{10}9$	div.	1	div.	1	div.	1
	10	$9.0_{10}10$	(56)	1	(71)	1	95	1
	11	$1.3_{10}12$	div.	2	div.	3	div.	2
	12	$2.0_{10}13$	div.	2	div.	3	div.	2
	13	$3.0_{10}14$	div.	4	div.	6	div.	5
	14	$4.7_{10}15$	div.	32	div.	(6)	div.	(7)
	15	$7.0_{10}16$	div.	(28)	div.	(83)	div.	(89)
H	6	$2.9_{10}7$	2	1	3	0	3	1
	7	$9.9_{10}8$	9	1	div.	1	(5)	1
	8	$3.4_{10}10$	div.	3	div.	2	div.	2
	9	$1.1_{10}12$	div.	(2)	div.	3	div.	(1)
	10	$3.5_{10}13$	div.	(4)	div.	(2)	div.	(2)
	11	$1.2_{10}15$	div.	(16)	div.	(89)	div.	(15)
H^{-1}	6	$2.9_{10}7$	1	0	5	0	1	0
Z	6	$1.1_{10}8$	3	0	3	0	3	0
	7	$6.1_{10}9$	11	1	12	1	11	1
	8	$3.4_{10}11$	div.	2	div.	1	div.	1
	9	$1.9_{10}13$	div.	3	div.	2	div.	6
	10	$1.1_{10}15$	div.	(3)	div.	8	div.	(3)
Z^{-1}	6	$1.1_{10}8$	3	0	4	0	3	0
	7	$6.1_{10}9$	11	1	27	1	11	1
	8	$3.4_{10}11$	div.	2	div.	2	div.	2
	9	$1.9_{10}13$	div.	3	div.	3	div.	(2)
	10	$1.1_{10}15$	div.	(3)	div.	(6)	div.	(3)
Q	4	$1.9_{10}12$	div.	1	div.	1	div.	1
	4	$1.2_{10}14$	div.	3	div.	2	div.	3
	6	$5.0_{10}17$	div.	14	div.	9	div.	12
	6	$3.9_{10}19$	div.	div.	div.	div.	div.	div.

Table 8. Number of residual corrections k for $\|x^k-A^{-1}b\|_\infty/\|A^{-1}b\|_\infty < 2\cdot10^{-7}$ resp. $2\cdot10^{-4}$

Similar computations have been performed in double precision as the base precision and totally similar results were obtained. As in section 1 we monitor the ratio cond(A)/cond(R*A) of improvement of the condition number. It is typically about 10^7 according to 24 bits of precision (computing in single precision). For example we obtained for Hilbert matrices

n	cond(A)/cond(R*A)
6	$2.2_{10}7$
7	$1.3_{10}7$
8	$1.4_{10}6$
9	$1.1_{10}6$
10	$1.1_{10}7$
11	$1.6_{10}7$

Table 9. Improvement of condition by preconditioning

As we mentioned before the computation of the approximate inverses R and S can be replaced by an LU-decomposition with backward substitution. The modifications to the algorithm are obvious. However this version performs not as good as algorithm 6. For illustration we display the part of table 8 for Hilbert matrices which is typical for the other examples.

			b_1		b_2		b_3	
matrix	n	cond	std.	new	std.	new(LU)	std.	new
H	6	$2.9_{10}7$	3	0	2	1	3	1
	7	$9.9_{10}8$	(2)	1	(8)	1	(2)	1
	8	$3.4_{10}10$	div.	2	div.	2	div.	3
	9	$1.1_{10}12$	div.	(3)	div.	6	div.	(4)
	10	$3.5_{10}13$	div.	div.	div.	div.	div.	div.
	11	$1.2_{10}15$	div.	div.	div.	div.	div.	div.

Table 10. Using LU-decomposition instead of gj (algorithm 5)

One might try to use S*R as an improved approximate inverse of A. This does not work as long as S*R is stored to single precision. A valid heuristic seems to be that for cond(A) > 2^{24} (the inverse of the relative rounding error unit) single precision does not suffice to store an approximate inverse R allowing $\rho(I-RA) < 1$.

The practical applications of algorithm 6 are very limited. It is expensive ($3n^3$ or at least $5/3 \cdot n^3$ for the LU-version), more expensive than Gaussian elimination in double precision (which is $4/3 \cdot n^3$ counting one double precision operation as 4 in single precision, an assumption which can be adapted when the doubled precision has to be simulated).

5. Bibliography.

[1] Björck, A.: Iterative Refinement and Reliable Computing, in "Advances in Reliable Numerical Computing", eds. M. Cox and S. Hammarting, Oxford University Press.

[2] MATLAB User's Guide, The MathWorks Inc., 1987.

[3] Rump, S.M.: A class of arbitrarily ill-conditioned floating-point matrices, appears in SIAM Journal on Matrix Analysis and Applications, 1990.

[4] Skeel, R.: Scaling for Numerical Stability in Gaussian Elimination, Journal of the ACM, Vol. 26, No. 3, 494-526, 1979.

[5] Wilkinson, J.H.: The Algebraic Eigenvalue Problem, Oxford University Press, 1988.

[6] Wilkinson, J.H.: The use of single precision residuals in the solution of linear systems, Unpublished manuscript, NPL (1977).