Error bounds for extremely ill-conditioned problems

Siegfried M. Rump

Institute for Reliable Computing, Hamburg University of Technology,
Schwarzenbergstraße 95, Hamburg 21071, Germany
and Waseda University, Faculty of Science and Engineering,
2-4-12 Okubo, Shinjuku-ku, Tokyo 169-0072, Japan.
email: rump@tu-harburg.de.

Abstract— We discuss methods to compute error bounds for extremely ill-conditioned problems. As a model problem we treat matrix inversion. We demonstrate that additive corrections to improve an approximate inverse are useful for ill-conditioned problems, but hardly usable for extremely ill-conditioned problems. Here multiplicative corrections can be used, including the possibility to compute guaranteed error bounds.

1. Introduction

Let us consider the inversion of a real $n \times n$ matrix as a model problem. This can be interpreted as a function from $\mathbb{R}^{n^2}$ to $\mathbb{R}^{n^2}$, which is continuous and differentiable on the set of n^2-vectors which, interpreted as an $n \times n$ matrix A, are nonsingular. Suppose A is nonsingular. The problem of matrix inversion is called ill-conditioned when small changes in the input data A cause large changes in the solution. If A is a matrix of floating point numbers and changes in the unit of the last place of the input data A cause a relative change in the solution of the order of 100 %, then the problem is extremely ill-conditioned. It is common belief that such problems cannot be solved in the same floating point format as the input data is stored. This fact is supported by the well known relation

$$\text{forward error} = \text{condition} \times \text{backward error} \ . \qquad (1)$$

Suppose the floating point format in use is IEEE 754 double precision corresponding to 53 bits in the mantissa. Then this means that the solution of problems with condition number $2^{53} \approx 10^{16}$ or above cannot be reasonably approximated in double precision. However, there are exceptions to that statement, namely, if, for some reason, intermediate operations can be performed exactly without rounding error, for instance in integer computations. A very interesting example are so-called error-free transformations. Here pairs of floating point numbers are transformed into pairs of floating point numbers without error. Consider the following algorithm by Knuth, 1969 [2]:

$$\begin{aligned} &\text{function } [x, y] = \text{TwoSum}(a, b) \\ &\quad x = \text{fl}(a + b) \\ &\quad z = \text{fl}(x - a) \\ &\quad y = \text{fl}((a - (x - z)) + (b - z)) \end{aligned}$$

This algorithm satisfies the following property.

Theorem 1 *Given two floating point numbers $a, b \in \mathbb{F}$, the result (x, y) of* TwoSum *satisfies*

$$x = fl(a + b) \quad \textit{and} \quad x + y = a + b.$$

Note that this statement is also true in the presence of underflow. Using TwoSum we can present the following algorithm for vector transformation.

$$\begin{aligned} &\text{function } p = \text{VecTransform}(p) \\ &\quad \text{for } i = 2 : n \qquad \%\text{length}(p) = n \\ &\quad\quad [p_i, p_{i-1}] = \text{TwoSum}(p_i, p_{i-1}) \end{aligned}$$

Note that we use the same variable p for input and output. One sees immediately by induction that the value of the sum $s := \sum p_i$ is not changed by VecTransform. For such an algorithm the relation (1) cannot hold since all intermediate transformations are exact. One can show [3] that the condition number of $\sum p_i$ drops

In Proceedings of 2006 International Symposium on Nonlinear Theory and its Applications, Bologna, Italy, September 11-14, 2006

basically by a factor 2^{-53} with each application of VecTransform. After sufficiently many transformations, $\mathrm{fl}(\sum p_i)$ will be a very accurate approximation of the exact sum s. Such algorithms are presented in [3] and [6].

The following ingenious method by Dekker splits a 53-bit floating point number $a \in \mathbb{F}$ into two 26-bit parts.

$$\begin{array}{l}\text{function } [x,y] = \mathrm{Split}(a)\\ \quad c = \mathrm{fl}(\mathrm{factor}\cdot a) \qquad \%\mathrm{factor} = 2^{27}+1\\ \quad x = \mathrm{fl}(c-(c-a))\\ \quad y = \mathrm{fl}(a-x)\end{array}$$

The result of this algorithm satisfies $x + y = a$ such that the results x, y do have not more that 26 significant bits. The trick is that the sign bit of x or y is used for the representation. The function Split is used to transform the product of two floating point numbers into the sum of two floating point numbers, again an error-free transformation. This is done by the following algorithm TwoProduct.

$$\begin{array}{l}\text{function } [x,y] = \mathrm{TwoProduct}(a,b)\\ \quad x = \mathrm{fl}(a\cdot b)\\ \quad [a_1,a_2] = \mathrm{Split}(a)\\ \quad [b_1,b_2] = \mathrm{Split}(b)\\ \quad y = \mathrm{fl}(a_2\cdot b_2 - (((x - a_1\cdot b_1) - a_2\cdot b_1) - a_1\cdot b_2))\end{array}$$

The mathematical property

$$x + y = a \cdot b$$

is true for all $a, b \in \mathbb{F}$ as long as no over- or underflow occurs. With this the dot product of two n-vectors can be transformed into the sum of an $2n$-vector, which in turn can be summed up with the methods described.

Therefore we can calculate accurate approximations of the $\sum(p_i)$ and $x^T y$ for n-vectors p, x, y. Note that the mentioned algorithms use only ordinary floating point addition and multiplication. The algorithms proved to be very fast [3, 6].

Let a matrix $A \in \mathbb{F}^{n\times n}$ be given. One may ask whether it is possible to compute an accurate approximation of the inverse of A using accurate summation and dot product.

2. Additive corrections

Let R be an approximate inverse of A, for example computed by the Matlab command inv(A). The obvious choice of iterative improvement of R is a Newton iteration. This method is known as Schulz iteration in the literature [7]. The first step of the iteration is as follows.

$$\begin{array}{l}X := R\\ X := X - (X\cdot A - I)\cdot X\end{array} \tag{2}$$

Here I denotes the $n \times n$ identity matrix. Consider as a model problem a randomly generated 100×100 matrix of condition number 10^{10}. We used `randsvd` as described in [1, Chapter 28]. In double precision with relative rounding error unit $\mathbf{u} = 2^{-53}$ we can expect $\log_2(\mathbf{u}^{-1}/10^{10}) \sim 6$ correct digits of the solution. We check the accuracy by means of the residual $\|I - R\cdot A\|$. We also check the right residual and obtain

$$\begin{array}{l}\|I - R\cdot A\| = 9.93\cdot 10^{-7} \quad \text{and}\\ \|I - A\cdot R\| = 7.87\cdot 10^{-6}.\end{array} \tag{3}$$

We display the result for one typical example. Note that we used high precision to calculate the residuals, so we can expect them to be correct. As expected the left residual is slightly better since the Matlab routine `inv` computes a left inverse of A. Next we perform one iteration (2) in double precision. We obtain

$$\begin{array}{l}\|I - X\cdot A\| = 6.30\cdot 10^{-7} \quad \text{and}\\ \|I - A\cdot X\| = 1.54\cdot 10^{+2}.\end{array} \tag{4}$$

First, the residual $I - X\cdot A$ in (2) is computed in double precision. This computation is always ill-conditioned, also for well-conditioned matrices, so we cannot expect a significant improvement. The left residual improves slightly, whereas the quality of X as a right inverse deteriorates completely. This is because the residual in (2) is written for a left inverse. The corresponding iteration for the right inverse

$$\begin{array}{l}X := R\\ X := X - X\cdot (A\cdot X - I)\end{array} \tag{5}$$

yields

$$\begin{array}{l}\|I - X\cdot A\| = 1.40\cdot 10^{+2} \quad \text{and}\\ \|I - A\cdot X\| = 8.06\cdot 10^{-7}.\end{array} \tag{6}$$

as expected. We mention that one iteration (2) or (5) with quadruple precision residual yields

$$\|X - A^{-1}\|/\|A^{-1}\| \approx 10^{-13}$$

as expected. However, the quality of the iteration matrix $I - XA$ or $I - AX$ does not improve. Next we concentrate on the left inverse and compute the residual $I - X \cdot A$ in quadruple precision. By a common rule of thumb we may expect an improvement of the relative accuracy of the result by cond $\cdot \mathbf{u} \sim 10^{-6}$. We denote this by

$$\begin{aligned} &X := R \\ &X := X - \text{double}(\text{quad}(X \cdot A - I)) \cdot X. \end{aligned} \tag{7}$$

Here quad$(\cdot)$ indicates that the whole expression inside the parentheses is computed in quadruple precision, and double$(\cdot)$ means to round it back into double precision. The result is as follows.

$$\begin{aligned} \|I - X \cdot A\| &= 1.64 \cdot 10^{-7} \quad \text{and} \\ \|I - A \cdot X\| &= 2.06 \cdot 10^{-7}. \end{aligned} \tag{8}$$

Interestingly, now the right residual is of the same quality as the right residual, however, almost no improvement is visible. The next step is to perform the multiplication by X in quadruple as well.

$$\begin{aligned} &X := R \\ &X := X - \text{double}(\text{quad}(X \cdot A - I) \cdot X). \end{aligned} \tag{9}$$

Now the entire correction $(X \cdot A - I) \cdot X$ is executed in quadruple, rounded to double and the added to X. The result is as follows.

$$\begin{aligned} \|I - X \cdot A\| &= 2.41 \cdot 10^{-7} \quad \text{and} \\ \|I - A \cdot X\| &= 3.09 \cdot 10^{-7}. \end{aligned} \tag{10}$$

Again we see no improvement. Finally, we may execute the whole computation of X in (2) in quadruple precision and round the result to double precision:

$$\begin{aligned} &X := R \\ &X := \text{double}(\text{quad}(X - (X \cdot A - I) \cdot X)). \end{aligned} \tag{11}$$

The result is as follows.

$$\begin{aligned} \|I - X \cdot A\| &= 2.09 \cdot 10^{-7} \quad \text{and} \\ \|I - A \cdot X\| &= 2.96 \cdot 10^{-7}. \end{aligned} \tag{12}$$

Strange enough the Newton iteration does not improve the result at all although all computations have been performed in quadruple precision and only the final result is rounded into double precision. Finally, we use the nearest floating point matrix to the exact inverse A^{-1}, the latter computed by some multiple precision routine, that is

$$X := \text{double}(A^{-1}). \tag{13}$$

Then the left and right residuals are

$$\begin{aligned} \|I - X \cdot A\| &= 2.26 \cdot 10^{-7} \quad \text{and} \\ \|I - A \cdot X\| &= 3.60 \cdot 10^{-7}. \end{aligned} \tag{14}$$

So there seems not much chance to decrease the left or right residual using a double precision matrix X as preconditioner. However, a residual iteration using R needs the residual matrix to be convergent. For extremely ill-conditioned matrices a double precision X seems unsufficient. A remedy is to store X in two parts. This was already used in [5] and was later called staggered correction. One method is to store the original X and the correction into two different parts.

$$\begin{aligned} &X := R \\ &X_1 := X, \\ &X_2 := \text{double}(\text{quad}(X \cdot A - I)) \cdot X. \end{aligned} \tag{15}$$

Note that only the residual is computed in quadruple precision and rounded into double, the multiplication by X is in double precision. Now the result is as follows.

$$\begin{aligned} \|I - (X_1 + X_2) \cdot A\| &= 1.18 \cdot 10^{-13} \quad \text{and} \\ \|I - A \cdot (X_1 + X_2)\| &= 1.17 \cdot 10^{-12}. \end{aligned} \tag{16}$$

As expected, the improvement is of the order $\mathbf{u} \cdot \text{cond}\,(A) \approx 10^{-6}$. As a result we see that for ill-conditioned problems the preconditioner R still contains enough information to produce small residuals, but only if the new preconditioner is stored in multiple precision. For extremely ill conditioned problems, however, things change again. Let A be a 100×100 matrix with condition number 10^{15}. This is at the limit of what can be solved in double precision. Then (15) with $R = \text{inv}(A)$ yields

$$\begin{aligned} \|I - (X_1 + X_2) \cdot A\| &= 4.09 \cdot 10^{+4} \quad \text{and} \\ \|I - A \cdot (X_1 + X_2)\| &= 8.87 \cdot 10^{+5}, \end{aligned} \tag{17}$$

so no information at all. Also with the exact inverse by (13) all information is gone:

$$\begin{aligned} \|I - X \cdot A\| &= 3.87 \cdot 10^{+4} \quad \text{and} \\ \|I - A \cdot X\| &= 4.27 \cdot 10^{+5}. \end{aligned} \tag{18}$$

The reason is that the initial approximation X, an approximate inverse of A, is so far from the true result A^{-1} that an additive correction does not work.

3. Multiplicative corrections

For a multiplicative correction we use the fact that an approximate inverse still contains a lot of structure. It does not contain adequate information to solve the residual equation directly, however, it may serve as a preconditioner. For a 100×100 matrix A with condition number 10^{21} we use multiple precision arithmetic to compute $\mathrm{cond}\,(A)$, $X := R \cdot A$ and $\mathrm{cond}\,(X)$, where R is the approximate inverse of A by $R := \mathrm{inv}(A)$.

$$\begin{array}{rcl} \mathrm{cond}\,(A) & = & 3.92 \cdot 10^{21} \quad \text{and} \\ \mathrm{cond}\,(X) & = & 2.69 \cdot 10^{7}. \end{array} \tag{19}$$

It is a general observation that for an *arbitrarily* ill-conditioned matrix A we can compute an approximate inverse $R := \mathrm{inv}(A)$ in double precision such that the condition number of the preconditioned matrix $X := R \cdot A$ drops by roughly a factor $\mathbf{u}$. The product $R \cdot A$ is computed using algorithms presented in [3] or [6]. This corresponds to a result *as if* computed in quadruple precision and then rounded to double precision or, with faithful rounding, respectively. The dropping of the condition number is also observed when computing $R \cdot A$ only in double precision.

These observations were used in about 1984, when we derived a method for inverting arbitrarily ill-conditioned matrices. This method, which we never published, requires the possibility to calculate a dot product $x^T y$ in k-fold precision and store into working precision as in [3] or [6]. The first step of the method is as follows.

$$\begin{array}{l} \text{function } [X1, X2] = \mathrm{AccInv}(A) \\ \quad R = \mathrm{inv}(A) \qquad \% \text{ double precision} \\ \quad P = \mathrm{inv}(\mathrm{double}(\mathrm{quad}(R \cdot A))) \\ \quad X1 + X2 = \mathrm{quad}(P \cdot R) \end{array}$$

As we have seen before we need higher precision for a preconditioner to produce a small residual. The last line in our algorithm computes the product $P \cdot R$ in quadruple precision and stores the result in two double precision parts $X1, X2$. This produces

$$\begin{array}{l} \|I - (X_1 + X_2) \cdot A\| = 3.17 \cdot 10^{-9} \quad \text{and} \\ \|I - A \cdot (X_1 + X_2)\| = 2.03 \cdot 10^{+7}. \end{array} \tag{20}$$

The left residual is small enough to produce a converge iteration, for example, for solving a system of linear equations. The right residual is large since P is computed using R as a left preconditioner. Calculating P as an approximate inverse of $A \cdot R$ and X by $R \cdot P$ yields

$$\begin{array}{l} \|I - (X_1 + X_2) \cdot A\| = 2.02 \cdot 10^{+8} \quad \text{and} \\ \|I - A \cdot (X_1 + X_2)\| = 3.56 \cdot 10^{-5}, \end{array} \tag{21}$$

as expected. The presented algorithm `AccInv` is the first step of an iterated algorithm, the latter being able to compute an approximate inverse of an arbitrarily ill-conditioned matrix in double precision floating point. It uses only the basic operations in double and an accurate dot product as presented in [3] or [6]. Very recently a partial analysis of this algorithm was presented in [4]. It is still quite mysterious how purely double precision computation can invert arbitrarily ill-conditioned matrices, and a full analysis is still open.

References

[1] N.J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM Publications, Philadelphia, 2nd edition, 2002.

[2] D.E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison Wesley, Reading, Massachusetts, 1969.

[3] T. Ogita, S.M. Rump, and S. Oishi. Accurate Sum and Dot Product. *SIAM Journal on Scientific Computing (SISC)*, 26(6):1955–1988, 2005.

[4] S. Oishi, K. Tanabe, T. Ogita, and S.M. Rump. Convergence of Rump's Method for Inverting Arbitrarily Ill-Conditioned Matrices. accepted for publication in JCAM, 2006.

[5] S.M. Rump. *Kleine Fehlerschranken bei Matrixproblemen*. PhD thesis, Universität Karlsruhe, 1980.

[6] S.M. Rump, T. Ogita, and S. Oishi. Accurate Floating-Point Summation. Technical Report 05.1, Faculty of Information and Communication Science, Hamburg University of Technology, 2005.

[7] G. Schulz. Iterative Berechnung der reziproken Matrix. *Zeitschrift für Angewandte Mathematik und Mechanik (ZAMM)*, 13:57–59, 1933.