Japan J. Indust. Appl. Math., 26 (2009), 249–277 Area ⟨2⟩

Inversion of Extremely Ill-Conditioned Matrices in Floating-Point

Siegfried M. Rump

Institute for Reliable Computing, Hamburg University of Technology
Schwarzenbergstraße 95, Hamburg 21071, Germany, and
Visiting Professor at Waseda University, Faculty of Science and Engineering
3–4–1 Okubo, Shinjuku-ku, Tokyo 169–8555, Japan
E-mail: rump@tu-harburg.de

Received March 17, 2008
Revised December 5, 2008

Let an $n \times n$ matrix A of floating-point numbers in some format be given. Denote the relative rounding error unit of the given format by `eps`. Assume A to be extremely ill-conditioned, that is $\mathrm{cond}(A) \gg \mathtt{eps}^{-1}$. In about 1984 I developed an algorithm to calculate an approximate inverse of A solely using the given floating-point format. The key is a multiplicative correction rather than a Newton-type additive correction. I did not publish it because of lack of analysis. Recently, in [9] a modification of the algorithm was analyzed. The present paper has two purposes. The first is to present reasoning how and why the original algorithm works. The second is to discuss a quite unexpected feature of floating-point computations, namely, that an approximate inverse of an extraordinary ill-conditioned matrix still contains a lot of useful information. We will demonstrate this by inverting a matrix with condition number beyond 10^{300} solely using double precision. This is a workout of the invited talk at the SCAN meeting 2006 in Duisburg.

Key words: extremely ill-conditioned matrix, condition number, multiplicative correction, accurate dot product, accurate summation, error-free transformations

1. Introduction and previous work

Consider a set of floating-point numbers $\mathbb{F}$, for instance double precision floating-point numbers according to the IEEE 754 standard [3]. Let a matrix $A \in \mathbb{F}^{n \times n}$ be given. The only requirement for the following algorithm are floating-point operations in the given format. For convenience, assume this format to be double precision in the following.

First we will show how to compute the dot product $x^{\mathrm{T}}y$ of two vectors $x, y \in \mathbb{F}$ in K-fold precision with storing the result in one or in K floating-point numbers. This algorithm to be described in Section 2 uses solely double precision floating-point arithmetic and is based on so-called error-free transformations [7, 13, 12]. The analysis will show that the result is of a quality "as if" computed in K-fold precision.

The relative rounding error unit in IEEE 754 double precision in rounding to nearest is $\mathtt{eps} = 2^{-53}$. Throughout the paper we assume that no over- or underflow occurs. Then every single floating-point operation produces a result with relative error not larger than `eps`.

This research was partially supported by Grant-in-Aid for Specially Promoted Research (No. 17002012: Establishment of Verified Numerical Computation) from the Ministry of Education, Science, Sports and Culture of Japan.

Throughout this paper we use the Frobenius norm $\|A\|_{\mathrm{F}} := \left(\sum a_{ij}^2\right)^{1/2}$. It is unitary as the often used spectral norm, but it is much easier to compute and of similar size as by $\|A\|_2 \le \|A\|_{\mathrm{F}} \le \sqrt{\mathrm{rank}(A)} \cdot \|A\|_2$.

For a matrix $A \in \mathbb{R}^{n\times n}$, the condition number $\mathrm{cond}(A) = \|A^{-1}\| \cdot \|A\|$ characterizes the sensitivity of the inverse of A with respect to small perturbations in A. More precisely, for small enough ε and $\|\Delta A\| = \varepsilon\|A\|$,

$$\frac{\|(A+\Delta A)^{-1} - A^{-1}\|}{\varepsilon\|A^{-1}\|} \le \mathrm{cond}(A). \tag{1.1}$$

Practical experience verifies that for a general perturbation ΔA we can expect almost equality in (1.1). Now consider an extremely ill-conditioned matrix $A \in \mathbb{F}^{n\times n}$. By that we mean a matrix A with $\mathrm{cond}(A) \gg \mathtt{eps}^{-1}$. As an example taken from [10] consider

$$A_4 = \begin{pmatrix} -5046135670319638 & -3871391041510136 & -5206336348183639 & -6745986988231149 \\ -640032173419322 & 8694411469684959 & -564323984386760 & -2807912511823001 \\ -16935782447203334 & -18752427538303772 & -8188807358110413 & -14820968618548534 \\ -1069537498856711 & -14079150289610606 & 7074216604373039 & 7257960283978710 \end{pmatrix}. \tag{1.2}$$

This innocent looking matrix is extremely ill-conditioned, namely

$$\mathrm{cond}(A_4) = 6.4 \cdot 10^{64}.$$

An approximate inverse R of extremely ill-conditioned matrices calculated by any standard algorithm such as Gaussian elimination is so severely corrupted by round-off errors that we cannot expect a single correct digit in R. For our matrix A_4 we obtain

$$\begin{aligned} \mathtt{inv}_{\mathrm{fl}}(A_4) &= \begin{pmatrix} -3.11 & -1.03 & 1.04 & -1.17 \\ 0.88 & 0.29 & -0.29 & 0.33 \\ -2.82 & -0.94 & 0.94 & -1.06 \\ 4.00 & 1.33 & -1.34 & 1.50 \end{pmatrix}, \\ \mathrm{fl}(A_4^{-1}) &= \begin{pmatrix} 8.97\cdot 10^{47} & 2.98\cdot 10^{47} & -3.00\cdot 10^{47} & 3.37\cdot 10^{47} \\ -2.54\cdot 10^{47} & -8.43\cdot 10^{46} & 8.48\cdot 10^{46} & -9.53\cdot 10^{46} \\ 8.14\cdot 10^{47} & 2.71\cdot 10^{47} & -2.72\cdot 10^{47} & 3.06\cdot 10^{47} \\ -1.15\cdot 10^{48} & -3.84\cdot 10^{47} & 3.85\cdot 10^{47} & -4.33\cdot 10^{47} \end{pmatrix}. \end{aligned} \tag{1.3}$$

Here $R := \mathtt{inv}_{\mathrm{fl}}(A_4)$ denotes an approximate inverse of A calculated in floating-point, for example by the Matlab command $R = \mathtt{inv}(A)$, whereas $\mathrm{fl}(A_4^{-1})$ denotes the true inverse A_4^{-1} rounded to the nearest floating-point matrix. Note that in our example R is almost a scalar multiple of A_4^{-1}, but this is not typical. As can be seen, R and A_4^{-1} differ by about 47 orders of magnitude. This corresponds to a well-known rule of thumb in numerical analysis [2].

One may regard such an approximate inverse R as useless. The insight of the algorithm to be described is that R contains a lot of useful information, enough

information to serve eventually as a good preconditioner for A. The challenge will be to extract this information out of R.

The key will be a multiplicative correction rather than a Newton-type additive correction. A Newton-type iteration converges only in a neighborhood of the true solution. But the ordinary approximate inverse R computed in double precision is far away from the true inverse, see the example above. Thus the additive correction is of the order of the approximation and causes only cancellation but no reasonable correction. The multiplicative correction, however, is capable to extract the information hidden in R.

Since our matrices are so extremely ill-conditioned that almost anything can happen, including R to be singular (which is, in fact, rare, but may happen), we may not expect a completely rigorous analysis. However, we give a reasoning how and why the algorithm works. A main observation to be explained in the following is that floating-point operations per se introduce a certain smoothing effect, similar to a regularization process applied to ill-posed problems.

As an answer to a question posed by Prof. Shin'ichi Oishi from Waseda University we mention that our algorithm may be applied to a sum $\sum A_i$ of m matrices as well. This is can be viewed as a representation of the input matrix in m-fold precision and may be helpful to generate extremely ill-conditioned matrices. As we will see this does not influence the analysis. An example with $m = 5$ is presented in Table 4.5.

The paper is organized as follows. In the following section we state some basic definitions, and we introduce and analyze a new algorithm producing a result "as if" computed in K-fold precision. In Section 3 we state and analyze the algorithm I developed in about 1984. The paper is concluded with computational results. Some proofs are moved to an appendix, where we also give the complete Matlab-code for our algorithms.

2. Basic facts

We denote by $\mathrm{fl}(\,\cdot\,)$ the result of a floating-point computation, where all operations within the parentheses are executed in working precision. If the order of execution is ambiguous and is crucial, we make it unique by using parentheses. An expression like $\mathrm{fl}\big(\sum p_i\big)$ implies inherently that summation may be performed in any order.

In mathematical terms, the fl-notation implies $|\mathrm{fl}(a \circ b) - a \circ b| \le \mathtt{eps}|a \circ b|$ for the basic operations $\circ \in \{+, -, \cdot, /\}$. In the following we need as well the result of a dot product $x^{\mathrm{T}}y$ for $x, y \in \mathbb{F}^n$ "as if" calculated in K-fold precision, where the result is stored in 1 term $\mathtt{res} \in \mathbb{F}$ or in K terms $\mathtt{res}_1, \ldots, \mathtt{res}_K \in \mathbb{F}$. We denote this by $\mathtt{res} := \mathrm{fl}_{K,1}(x^{\mathrm{T}}y)$ or $\mathtt{res} := \mathrm{fl}_{K,K}(x^{\mathrm{T}}y)$, respectively. The subindices K, K indicate that the quality of the result is "as if" computed in K-fold precision, and the result is stored in K parts. For completeness we will as well mention an algorithm for $\mathrm{fl}_{K,L}$, i.e., storing the result in L terms, although we don't actually need it for this paper.

The mathematical assumption on $\mathrm{fl}_{K,K}(x^{\mathrm{T}}y)$ is not very strict, we only require

$$\mathtt{res} = \mathrm{fl}_{K,K}(x^{\mathrm{T}}y) \implies \left| \sum_{i=1}^{K} \mathtt{res}_i - x^{\mathrm{T}}y \right| \leq \varphi \cdot \mathtt{eps}^K |x^{\mathrm{T}}|\,|y| \tag{2.1}$$

for a reasonably small constant φ. Note that this implies the relative error of $\sum \mathtt{res}_i$ to $x^{\mathrm{T}}y$ to be not much larger than $\mathtt{eps}^K \operatorname{cond}(x^{\mathrm{T}}y)$. More general, the result may be stored in L terms $\mathtt{res}_{1\cdots L}$. We then require

$$\mathtt{res} = \mathrm{fl}_{K,L}(x^{\mathrm{T}}y) \implies \left| \sum_{i=1}^{L} \mathtt{res}_i - x^{\mathrm{T}}y \right| \leq \varphi' \cdot \mathtt{eps}^L |x^{\mathrm{T}}y| + \varphi'' \cdot \mathtt{eps}^K |x^{\mathrm{T}}|\,|y| \tag{2.2}$$

for reasonably small constants φ' and φ''. In other words, the result `res` is of a quality "as if" computed in K-fold precision and then rounded into L floating-point numbers. The extra rounding imposes the extra term $\varphi'\mathtt{eps}^L$ in (2.2). For $L = 1$, the result of Algorithm 4.8 (`SumK`) in [7] satisfies (2.2), and for $L = K$ the result of Algorithm `SumL` in [14] satisfies (2.1). In this paper we need only these two cases $L = 1$ and $L = K$. A value $L > K$ does not make much sense since the precision is only K-fold.

It is not difficult and nice to see the rationale behind both approaches. Following we describe an algorithm to compute $\mathrm{fl}_{K,L}(x^{\mathrm{T}}y)$ for $1 \leq L \leq K$. We first note that using algorithms by Dekker and Veltkamp [1] two vectors $x, y \in \mathbb{F}^n$ can be transformed into a vector $z \in \mathbb{F}^{2n}$ such that $x^{\mathrm{T}}y = \sum_{i=1}^{2n} z_i$, an error-free transformation. Thus we concentrate on computing a sum $\sum p_i$ for $p \in \mathbb{F}^n$ "as if" calculated in K-fold precision and rounded into L floating-point numbers. In [13, 12] an algorithm was given to compute a K-fold faithfully rounded result. Following we derive another algorithm to demonstrate that faithful rounding is not necessary to invert extremely ill-conditioned matrices.

All algorithms use extensively the following error-free transformation of the sum of two floating-point numbers. It was given by Knuth in 1969 [4] and can be depicted as in Fig. 2.1.

ALGORITHM 2.1. *Error-free transformation for the sum of two floating-point numbers.*

---

function $[x, y] = \mathtt{TwoSum}(a, b)$
  $x = \mathrm{fl}(a + b)$
  $z = \mathrm{fl}(x - a)$
  $y = \mathrm{fl}((a - (x - z)) + (b - z))$

Knuth's algorithm transforms any pair of floating-point numbers (a, b) into a new pair (x, y) with

$$x = \mathrm{fl}(a + b) \quad \text{and} \quad x + y = a + b. \tag{2.3}$$

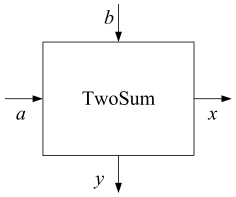

Fig. 2.1. Error-free transformation $x+y=a+b$ of the sum of two floating-point numbers.

Note that this is also true in the presence of underflow. First we repeat Algorithm 4.2 (`VecSum`) in [7]. For clarity, we state the algorithm without overwriting the input vector.

ALGORITHM 2.2. *Error-free vector transformation.*

function $q = \texttt{VecSum}(p)$
  $q_1 = p_1$
  for $i = 2 : n$
    $[q_i, q_{i-1}] = \texttt{TwoSum}(p_i, q_{i-1})$

Successive application of (2.3) to a vector $p \in \mathbb{F}^n$ yields

$$\sum_{i=1}^{n} q_i = \sum_{i=1}^{n} p_i, \tag{2.4}$$

the transformation is error-free. Moreover, Lemma 4.2 in [7] implies

$$\sum_{i=1}^{n-1} |q_i| \le \gamma_{n-1} \sum_{i=1}^{n} |p_i|, \tag{2.5}$$

where $\gamma_k := k\texttt{eps}/(1 - k\texttt{eps})$ as usual [2]. Note that $q_1, \ldots, q_{n-1}$ are the errors of the intermediate floating-point operations and q_n is the result of ordinary recursive summation, respectively. Now we modify Algorithm 4.8 (`SumK`) in [7] according to the scheme as in Fig. 2.2. Compared to Fig. 4.2 in [7] the right-most "triangle" is omitted. We first discuss the case $L = K$, i.e., summation in K-fold precision and rounding into K results. The algorithm is given as Algorithm 2.3 (`SumKK`). Again, for clarity, it is stated without overwriting the input vector.

ALGORITHM 2.3. *Summation "as if" computed in K-fold precision with K results.*

function $\{\texttt{res}\} = \texttt{SumKK}(p^{(0)}, K)$
  for $k = 0 : K - 2$
    $p^{(k+1)} = \texttt{VecSum}\big(p^{(k)}_{1\cdots n-k}\big)$
    $\texttt{res}_{k+1} = p^{(k+1)}_{n-k}$
  $\texttt{res}_K = \mathrm{fl}\big(\sum_{i=1}^{n-K+1} p_i^{(K-1)}\big)$

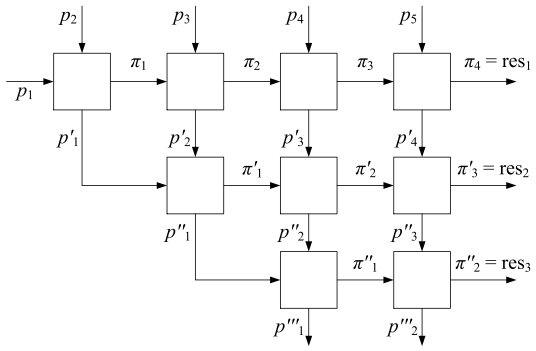

Fig. 2.2. Summation "as if" in K-fold precision.

Note that the vectors $p^{(k)}$ become shorter with increasing values of k, more precisely $p^{(k)}$ has $n-k$ elements. In the output of the algorithm we use braces for the result $\{\mathtt{res}\}$ to indicate that it is a collection of floating-point numbers.

Regarding Fig. 2.2 and successively applying (2.3) yields

$$s := \sum_{i=1}^{n} p_i = p'_1 + \pi_1 + \sum_{i=3}^{n} p_i = p'_1 + p'_2 + \pi_2 + \sum_{i=4}^{n} p_i = \cdots = \sum_{i=1}^{n-1} p'_i + \mathtt{res}_1.$$

In the same way we conclude

$$\sum_{i=1}^{n-1} p'_i = \sum_{i=1}^{n-2} p''_i + \mathtt{res}_2 \quad \text{and} \quad \sum_{i=1}^{n-2} p''_i = \sum_{i=1}^{n-3} p'''_i + \mathtt{res}_3.$$

Applying this scheme to Algorithm 2.3 it follows

$$\begin{aligned}\sum_{i=1}^{n} p_i^{(0)} &= \sum_{i=1}^{n} p_i^{(1)} = \mathtt{res}_1 + \sum_{i=1}^{n-1} p_i^{(1)} \\ &= \mathtt{res}_1 + \mathtt{res}_2 + \sum_{i=1}^{n-2} p_i^{(2)} = \cdots = \sum_{k=1}^{K-1} \mathtt{res}_k + \sum_{i=1}^{n-K+1} p_i^{(K-1)}.\end{aligned}$$

Now (2.5) gives

$$\sum_{i=1}^{n-K+1} \left|p_i^{(K-1)}\right| \le \gamma_{n-K+1} \cdot \sum_{i=1}^{n-K+2} \left|p_i^{(K-2)}\right| \le \cdots \le \left(\prod_{\nu=n-K+1}^{n-1} \gamma_\nu\right) \cdot \sum_{i=1}^{n} \left|p_i^{(0)}\right|.$$

The error of the final floating-point summation to compute $\mathtt{res}_K$ satisfies [2]

$$\left|\mathtt{res}_K - \sum_{i=1}^{n-K+1} p_i^{(K-1)}\right| \le \gamma_{n-K} \cdot \sum_{i=1}^{n-K+1} \left|p_i^{(K-1)}\right|.$$

Combining the results proves the following theorem.

THEOREM 2.4. *Let a vector $p \in \mathbb{F}^n$ be given and define $s := \sum_{i=1}^{n} p_i$. Then for $K \geq 1$ the result* $\mathtt{res} \in \mathbb{F}^K$ *of* Algorithm 2.3 *satisfies*

$$\left| \sum_{k=1}^{K} \mathtt{res}_k - s \right| \leq \left(\prod_{\nu=n-K}^{n-1} \gamma_\nu \right) \cdot \sum_{i=1}^{n} |p_i| \leq \gamma_{n-1}^{K} \cdot \sum_{i=1}^{n} |p_i|. \tag{2.6}$$

We note that the sum $\sum \mathtt{res}_k$ may be ill-conditioned. The terms $\mathtt{res}_k$ need neither to be ordered by magnitude, nor must they be non-overlapping. However, this is not important: The only we need is that the final error is of the order $\mathtt{eps}^K$ as seen in (2.6).

To obtain a result stored in less than K results, for example $L = 1$ result, we might sort $\mathtt{res}_i$ by magnitude and sum in decreasing order. A simple alternative is to apply the error-free vector transformation `VecSum` $K-L$ times to the input vector p and then compute the final L results by the following Algorithm 2.5. Since the transformation by `VecSum` is error-free, Theorem 2.4 implies an error of order $\mathtt{eps}^L$.

The corresponding algorithm to compute $\mathrm{fl}_{K,L}\left(\sum p_i\right)$ is given as Algorithm 2.5, now with overwriting variables. Note that Algorithm 2.5 and Algorithm 2.3 are identical for $L = K$.

ALGORITHM 2.5. *Summation "as if" computed in K-fold precision and rounded into L results* $\mathtt{res}_{1\cdots L}$.

---

function $\mathtt{res} = \mathtt{SumKL}(p, K, L)$
  for $i = 1 : K - L$
    $p = \mathtt{VecSum}(p)$
  for $k = 0 : L - 2$
    $p_{1\cdots n-k} = \mathtt{VecSum}(p_{1\cdots n-k})$
    $\mathtt{res}_{k+1} = p_{n-k}$
  $\mathtt{res}_L = \mathrm{fl}\left(\sum_{i=1}^{n-L+1} p_i\right)$

We note that the results of Algorithm `SumKL` is much better than may be expected by Theorem 2.4. In fact, in general the estimations (2.1) and (2.2) are satisfied for factors φ and φ' not too far from 1.

As has been noted before, we need Algorithm `SumKL` only for $L = K$ and $L = 1$. For general L this is Algorithm 2.5, for $L = K$ this is Algorithm 2.3, and for $L = 1$ this is Algorithm 4.8 (`SumK`) in [7]. For the reader's convenience we repeat this very simple algorithm.

ALGORITHM 2.6. *Summation "as if" computed in K-fold precision and rounded into one result* `res`.

---

function $\mathtt{res} = \mathtt{SumK1}(p, K)$
  for $i = 1 : K - 1$
    $p = \mathtt{VecSum}(p)$
  $\mathtt{res} = \mathrm{fl}\left(\sum_{i=1}^{n} p_i\right)$

As has been mentioned, there is an error-free transformation of a dot product of two n-vectors x, y into the sum of a $2n$-vector [7]. If this sum is calculated by `SumKL`, we will refer to the result of the dot product as $\mathrm{fl}_{K,L}(x^{\mathrm{T}}y)$.

3. The algorithm and its analysis

First let us clarify the notation. For two matrices $A, B \in \mathbb{F}^{n\times n}$ the ordinary product in working precision is denoted by $C = \mathrm{fl}(A \cdot B)$.

If the product is executed in k-fold precision and stored in k-fold precision, we write $\{P\} = \mathrm{fl}_{k,k}(A \cdot B)$. We put braces around the result to indicate the P comprises of k matrices $P_\nu \in \mathbb{F}^{n\times n}$. This operation is executed by firstly transforming the dot products into sums, which is error-free, and secondly by applying Algorithm 2.3 (`SumKK`) to the sums.

If the product is executed in k-fold precision and stored in working precision, we write $P = \mathrm{fl}_{k,1}(A \cdot B) \in \mathbb{F}^{n\times n}$. This operation is executed as $\mathrm{fl}_{k,k}(A \cdot B)$ but Algorithm 2.6 (`SumK1`) is used for the summation. The result is a matrix in working precision.

For both types of multiplication in k-fold precision, either one of the factors may be a collection of matrices, for example $\{P\} = \mathrm{fl}_{k,k}(A \cdot B)$ and $\{Q\} = \mathrm{fl}_{m,m}(\{P\} \cdot C)$ for $C \in \mathbb{F}^{n\times n}$. In this case the result is the same as $\mathrm{fl}_{m,m}\big(\sum_{\nu=1}^{k} P_\nu C\big)$, where the dot products are of lengths kn. Similarly, $Q = \mathrm{fl}_{m,1}(\{P\} \cdot C)$ is computed. Note that we need to allow only one factor to be a collection of matrices except when the input matrix A itself is a collection of matrices.

In the following algorithms it will be clear from the context that a factor $\{P\}$ is a collection of matrices. So for better readability we omit the braces and write $\mathrm{fl}_{m,m}(P \cdot C)$ or $\mathrm{fl}_{m,1}(P \cdot C)$.

Finally we need an approximate inverse R of a matrix $A \in \mathbb{F}^{n\times n}$ calculated in working precision; this is denoted by $R = \mathtt{inv}_{\mathrm{fl}}(A)$. Think of this as the result of the Matlab command $R = \mathtt{inv}(A)$. The norm used in the following Algorithm 3.1 (`InvIllCoPrel`) to initialize $R^{(0)}$ is only for scaling; any moderate approximate value of some norm is fine.

ALGORITHM 3.1. *Inversion of an extremely ill-conditioned matrix, preliminary version.*

---

function $\{R^{(k)}\} = \mathtt{InvIllCoPrel}(A)$
  $R^{(0)} = \mathrm{fl}_{1,1}\big(\frac{1}{\|A\|}\big) \cdot I$; $k = 0$
  repeat
    $k = k + 1$
    $P^{(k)} = \mathrm{fl}_{k,1}(R^{(k-1)} \cdot A)$ *% stored in working precision*
    $X^{(k)} = \mathrm{inv}_{\mathrm{fl}}(P^{(k)})$ *% floating-point inversion in working precision*
    $\{R^{(k)}\} = \mathrm{fl}_{k,k}(X^{(k)} \cdot R^{(k-1)})$ *% stored in k-fold precision*
  until $\mathrm{cond}(P^{(k)}) < \mathtt{eps}^{-1}/100$

As can be seen, $R^{(0)}$ is just a scaled identity matrix, generally a very poor approximate inverse, and $R^{(1)}$ is, up to rounding errors, the approximate inverse computed in ordinary floating-point. In practice the algorithm may start with $k = 1$ and $R^{(1)} := \mathtt{inv}_{\mathrm{fl}}(A)$. However, the case $k = 0$ gives some insight, so for didactical purposes the algorithm starts with $k = 0$.

For our analysis we will use the "$\sim$"-notation to indicate that two quantities are of the same order of magnitude. So for $e, f \in \mathbb{R}$, $e \sim f$ means $|e| = \varphi|f|$ for a factor φ not too far from 1. The concept applies similarly to vectors and matrices.

Next we informally describe how the algorithm works. First, let A be an ill-conditioned, but not extremely ill-conditioned matrix, for example $\mathrm{cond}(A) \sim \mathtt{eps}^{-1}/10$. Then $\mathrm{cond}(P^{(1)}) = \mathrm{cond}(A)$, the stopping criterion is not satisfied, and basically $R^{(1)} \sim \mathtt{inv}_{\mathrm{fl}}(A)$ and $R^{(1)} \in \mathbb{F}^{n\times n}$. We will investigate the second iteration $k = 2$. For better readability we omit the superindices and abbreviate $R := R^{(1)}$ and $R' := R^{(2)}$, then

$$P \approx R \cdot A, \quad X \approx (RA)^{-1} = A^{-1}R^{-1} \quad \text{and} \quad R' \approx X \cdot R \approx A^{-1}.$$

For A not too ill-conditioned, we can expect R to be a good enough precondition matrix for A, so that RA is not too far from the identity matrix. This means $\mathrm{cond}(P) \approx 1$. Yet, due to the condition of A, we expect $\|I - RA\|$ to be less than, but not much less than 1. Thus we can expect X to be an approximate inverse of RA of reasonable quality.

The final step $R' \approx X \cdot R$ needs special attention. The aim is to produce a preconditioner R' of A so that $\|I - R'A\|$ is very small, at best of the order $\mathtt{eps}$. But the i-th column of $R'A$ is an approximate solution of the linear systems $Ax = e_i$, were e_i denotes the i-th column of the identity matrix. Thus the relative error of that approximate solution is expected to be $\mathrm{cond}(A) \cdot \mathtt{eps}$, which is 1/10 in our example. This means, for $R' \in \mathbb{F}^{n\times n}$ the entries of $I - R'A$ will be at least of order 0.1, and $\|I - R'A\|$ can't be much less than 1.

This argument is true as long R' is a matrix with floating-point entries, i.e., with precision $\mathtt{eps}$. Therefore it is mandatory to store $R' = R^{(2)}$ in two matrices, as indicated by $\{R^{(2)}\} = \mathrm{fl}_{2,2}(X^{(2)} \cdot R^{(1)})$.

A key observation in this example is that $\mathrm{cond}(RA) \approx \mathtt{eps} \cdot \mathrm{cond}(A)$. We claim that this observation is generally true, also for extremely ill-conditioned matrices. More precisely, for $\mathrm{cond}(A) \gg \mathtt{eps}^{-1}$ and $R := \mathtt{inv}_{\mathrm{fl}}(A)$ we claim $\mathrm{cond}(RA) \approx \mathtt{eps} \cdot \mathrm{cond}(A)$. To develop arguments for this will be a main task in the following.

To illustrate this behavior, consider again the extremely ill-conditioned 4×4 matrix A_4 in (1.2). Recall $\mathrm{cond}(A_4) = 7.5 \cdot 10^{64}$. In the following Table 3.1 we display for $k \geq 2$ the condition numbers $\mathrm{cond}(R^{(k-1)})$, $\mathrm{cond}(R^{(k-1)}A)$ and $\mathrm{cond}(P^{(k)})$ as well as $\|I - R^{(k)}A\|$. The numbers confirm that in each iteration $\mathrm{cond}(R^{(k-1)})$ increases by a factor $\mathtt{eps}^{-1}$ starting at $\mathrm{cond}(R^{(1)}) = 1$, and that $\mathrm{cond}(R^{(k-1)}A)$ decreases by a factor $\mathtt{eps}$ starting at $\mathrm{cond}(R^{(1)}A) = \mathrm{cond}(A) = 6.4 \cdot 10^{64}$. Note that $R^{(k-1)}A$ denotes the exact product of $R^{(k-1)}$ and A, whereas $P^{(k)}$ is this product calculated in k-fold precision and rounded into working precision.

Table 3.1. Computational results of Algorithm 3.1 (`InvIllCoPrel`) for the matrix A_4 in (1.2).

k	$\mathrm{cond}(R^{(k-1)})$	$\mathrm{cond}(R^{(k-1)}A)$	$\mathrm{cond}(P^{(k)})$	$\|I - R^{(k)}A\|$
2	$1.68 \cdot 10^{17}$	$2.73 \cdot 10^{49}$	$2.31 \cdot 10^{17}$	3.04
3	$1.96 \cdot 10^{32}$	$2.91 \cdot 10^{33}$	$2.14 \cdot 10^{17}$	5.01
4	$7.98 \cdot 10^{48}$	$1.10 \cdot 10^{17}$	$1.83 \cdot 10^{17}$	1.84
5	$6.42 \cdot 10^{64}$	8.93	8.93	$3.43 \cdot 10^{-16}$

As we will see, rounding into working precision has a smoothing effect, similar to regularization, which is important for our algorithm. This is why $\mathrm{cond}(P^{(k)})$ is constantly about $\mathtt{eps}^{-1}$ until the last iteration, and also $\|I - R^{(k)}A\|$ is about 1 until the last step. We claim and will see that this behavior is typical.

The matrices $P^{(k)}$ and $R^{(k)}$ are calculated in k-fold precision, where the first is stored in working precision in order to apply a subsequent floating-point inversion, and the latter stored in k-fold precision. It is interesting to monitor what happens when using another precision to calculate $P^{(k)}$ and $R^{(k)}$. This was a question by one of the referees.

First, suppose we invest more and compute both $P^{(k)}$ and $R^{(k)}$ in 8-fold precision for all k, much more than necessary for the matrix A_4 in (1.2). More precisely, we replace the inner loop in Algorithm 3.1 (`InvIllCoPrel`) by

$$
\begin{array}{l}
P^{(k)} = \mathrm{fl}_{prec,1}(R^{(k-1)} \cdot A) \\
\qquad \% \text{ computed in } prec\text{-fold and stored in working precision} \\
X^{(k)} = \mathrm{inv}_{\mathrm{fl}}(P^{(k)}) \\
\qquad \% \text{ floating-point inversion in working precision} \\
\{R^{(k)}\} = \mathrm{fl}_{prec,prec}(X^{(k)} \cdot R^{(k-1)}) \\
\qquad \% \text{ computed in } prec\text{-fold and stored in } prec\text{-fold precision}
\end{array}
\tag{3.1}
$$

with fixed $prec = 8$ for all k. Note that 8-fold precision corresponds to a relative rounding error unit of $\mathtt{eps}^8 = 2.3 \cdot 10^{-128}$. If matrix inversion would be performed in that precision, the approximate inverse of the matrix A_4 would be correct to the last bit.

The results are displayed in the following Table 3.2. As can be seen there is not much difference to the data in Table 3.1, the quality of the results does not increase. Computational experience and our analysis suggest that this behavior is typical, so that it does not make sense to increase the intermediate computational precision. The reason is that everything depends on the approximate inverses $X^{(k)}$ which is computed in working precision, so that there is no more information to squeeze out.

Second, one may try to reduce the computational effort and compute both $P^{(k)}$ and $R^{(k)}$ in 4-fold precision for all k. For $k < 4$ this is more than in Algorithm 3.1 (`InvIllCoPrel`), for $k > 4$ it is less. That means we replace the inner loop in Algorithm 3.1 for the matrix A_4 by (3.1) with $prec = 4$. The results are displayed

Table 3.2. Results of Algorithm 3.1 (`InvIllCoPrel`) for the matrix A_4 in (1.2) using fixed 8-fold precision.

k	$\mathrm{cond}(R^{(k-1)})$	$\mathrm{cond}(R^{(k-1)}A)$	$\mathrm{cond}(P^{(k)})$	$\|I - R^{(k)}A\|$
2	$1.68 \cdot 10^{17}$	$2.73 \cdot 10^{49}$	$2.66 \cdot 10^{17}$	3.59
3	$2.73 \cdot 10^{32}$	$4.37 \cdot 10^{33}$	$2.80 \cdot 10^{17}$	4.78
4	$6.49 \cdot 10^{48}$	$9.05 \cdot 10^{17}$	$1.12 \cdot 10^{18}$	17.8
5	$1.76 \cdot 10^{67}$	7750	7750	$3.86 \cdot 10^{-13}$
6	$7.45 \cdot 10^{64}$	4.0	4.0	$1.79 \cdot 10^{-16}$

in the following Table 3.3. As expected by the previous example, there is not much difference to the data in Table 3.1 for $k \leq 4$, the quality of the results does not increase. The interesting part comes for $k > 4$. Now the precision is not sufficient to store $R^{(k)}$ for $k \geq 5$, as explained before for $R^{(2)}$, so that the grid for $R^{(k)}$ is too coarse to make $I - R^{(k)}A$ convergent, so $\|I - R^{(k)}A\|$ remains about 1. That suggests the computational precision in Algorithm 3.1 is just sufficient in each loop.

Table 3.3. Results of Algorithm 3.1 (`InvIllCoPrel`) for the matrix A_4 in (1.2) using fixed 4-fold precision.

k	$\mathrm{cond}(R^{(k-1)})$	$\mathrm{cond}(R^{(k-1)}A)$	$\mathrm{cond}(P^{(k)})$	$\|I - R^{(k)}A\|$
2	$1.68 \cdot 10^{17}$	$2.73 \cdot 10^{49}$	$2.66 \cdot 10^{17}$	3.59
3	$2.73 \cdot 10^{32}$	$4.37 \cdot 10^{33}$	$2.80 \cdot 10^{17}$	4.78
4	$6.49 \cdot 10^{48}$	$9.05 \cdot 10^{17}$	$4.52 \cdot 10^{17}$	16.9
5	$3.51 \cdot 10^{64}$	34.8	41.1	5.37
6	$1.34 \cdot 10^{65}$	37.0	19.2	2.75
7	$1.39 \cdot 10^{65}$	22.3	26.4	6.42
8	$1.43 \cdot 10^{65}$	18.5	19.4	3.79

In [9] our Algorithm 3.1 (`InvIllCoPrel`) was modified in the way that the matrix $P^{(k)}$ was artificially afflicted with a random perturbation of size $\sqrt{\mathtt{eps}}\,|P^{(k)}|$. This simplifies the analysis; however, the modified algorithm also needs about twice as many iterations to compute an approximate inverse of similar quality. We will come to that again in the result section (cf. Table 4.3). In the following we will analyze the original algorithm.

Only the matrix R in Algorithm 3.1 (`InvIllCoPrel`) is represented by a sum of matrices. As mentioned before, the input matrix A may be represented by a sum $\sum A_i$ as well. In this case the initial $R^{(0)}$ can be taken as an approximate inverse of the floating-point sum of the A_i. Otherwise only the computation of P is affected.

For the preliminary version we use the condition number of $P^{(k)}$ in the stopping criterion. Since $X^{(k)}$ is an approximate inverse of $P^{(k)}$ we have $\mathrm{cond}(P^{(k)}) \sim \|X^{(k)}\| \cdot \|P^{(k)}\|$. The analysis will show that this can be used in the final version `InvIllCo` in the stopping criterion without changing the behavior of the algorithm.

Following we analyze this algorithm for input matrix A with $\mathrm{cond}(A) \gg \mathtt{eps}^{-1}$. Note that in this case the ordinary floating-point inverse $\mathtt{inv}_{\mathtt{fl}}(A)$ is totally corrupted by rounding errors, cf. (1.3).

We call a vector "generic" if its entries are pseudo-random values uniformly distributed in some interval. Correspondingly, a matrix $A \in \mathbb{R}^{n\times n}$ is said to be "generic" if its columns are generic. Similarly a vector is called generic on the unit ball if it is chosen randomly on the unit ball with uniform density. We first show that the distance of a generic vector $x \in \mathbb{R}^n$ to its rounded image $\mathrm{fl}(x)$ can be expected to be nearly as large as possible. The proof is given in the appendix.

LEMMA 3.2. *Let a nonnegative vector $d = (d_1, \ldots, d_n) \in \mathbb{R}^n$ be given. Suppose x_i, $1 \le i \le n$ are independent variables uniformly distributed in $[-d_i, d_i]$. Then the expected Euclidean length of $x = (x_1, \ldots, x_n)^{\mathrm{T}}$ satisfies*

$$\frac{1}{2\sqrt{n}} \cdot \|d\|_1 \le E(\|x\|_2) \le \|d\|_2. \tag{3.2}$$

For an equidistant grid, i.e., $d_1 = \cdots = d_n$, we have

$$\frac{1}{2} \cdot \|d\|_2 \le E(\|x\|_2) \le \|d\|_2.$$

Note that the maximum Euclidean length of x is $\|d\|_2$. In other words, this repeats the well-known fact that most of the "volume" of an n-dimensional rectangle resides near the vertices. The same is true for matrices, the distance $\|\mathrm{fl}(A) - A\|$ can be expected to be nearly as large as possible.

COROLLARY 3.3. *Let a generic matrix $A \in \mathbb{R}^{n\times n}$ be given, where no entry a_{ij} is in the overflow- or underflow range. Denote the rounded image of A by $\tilde{A} = \mathrm{fl}(A)$. Then*

$$\frac{\mathtt{eps}}{2n} \cdot \|\tilde{A}\|_{\mathrm{F}} \le \frac{\mathtt{eps}}{2n} \cdot \sum_{i,j} |\tilde{a}_{ij}| \le E(\|\tilde{A} - A\|_{\mathrm{F}}) \le \mathtt{eps} \cdot \|\tilde{A}\|_{\mathrm{F}}. \tag{3.3}$$

If the entries of A are of similar magnitude, then

$$E(\|\tilde{A} - A\|_{\mathrm{F}}) = \beta \mathtt{eps} \cdot \|\tilde{A}\|_{\mathrm{F}}$$

for a factor β not far from 1.

Proof. For generic A we can expect $\tilde{a}_{ij} = \mathrm{fl}(a_{ij})$ not to be a power of 2. Then $a_{ij} - \tilde{a}_{ij}$ is in the grid $[-d_{ij}, d_{ij}]$ with $d_{ij} = \mathtt{eps} \cdot |\tilde{a}_{ij}|$. Regarding A as an element in $\mathbb{R}^{n^2}$ and applying Lemma 3.2 proves the result. □

We cannot expect a totally rigorous analysis of Algorithm 3.1 (`InvIllCoPrel`) since, for example, the inversion of $P^{(k)}$ in working precision may fail (we discuss a cure of even this later). Thus we collect a number of arguments to understand the principle of the algorithm. The main point will be the observation that even an approximate inverse of an arbitrarily ill-conditioned matrix does, in general, contain useful information.

This is due to a kind of regularization by rounding into working precision. In fact, Corollary 3.3 shows that the rounded image of a real matrix A in floating-point can be expected to be far away from A itself. This is in particular helpful if A is extremely ill-conditioned: the rounded image has, in general, a much better condition number. More precisely, we can expect the condition number of a generic floating-point matrix to be limited to not much more than $\texttt{eps}^{-1}$. In the appendix we will show the following.

OBSERVATION 3.4. *Let A be generic on the variety of singular matrix $\mathbb{R}^{n\times n}$ be given. Then the expected Frobenius-norm condition number of A rounded into a floating-point matrix $\mathrm{fl}(A)$ satisfies*

$$\frac{\sqrt{n^2-2}}{0.68}\cdot \texttt{eps}^{-1} \le E(\mathrm{cond}_{\mathrm{F}}(\mathrm{fl}(A))) \le \frac{n\sqrt{n^2-1}}{0.3}\cdot \texttt{eps}^{-1}, \tag{3.4}$$

where eps *denotes the relative rounding error unit. If the entries of A are of similar magnitude, then*

$$E(\mathrm{cond}_{\mathrm{F}}(\mathrm{fl}(A))) = n\beta\cdot \texttt{eps}^{-1}$$

for a factor β not far from 1.

The arguments for Observation 3.4 are based on the following fact which states that the expected angle between two generic vectors is approaching $\pi/2$ with increasing dimension. This is surely known. However, we did not find a proper reference so we state the result and prove it in the appendix.

THEOREM 3.5. *Let $x, y \in \mathbb{R}^n$ be generic vectors on the unit ball and suppose $n \ge 4$. Then the expected value of $|x^{\mathrm{T}}y|$ satisfies*

$$\frac{0.61}{\sqrt{n-1}} \le E(|x^{\mathrm{T}}y|) \le \frac{0.68}{\sqrt{n-2}}. \tag{3.5}$$

Using this we first estimate the magnitude of the elements of an approximate inverse $R := \texttt{inv}_{\mathrm{fl}}(A)$ of $A \in \mathbb{F}^{n\times n}$ and, more interesting, the magnitude of $\|I - RA\|$. We know that for not extremely ill-conditioned matrices the latter is about $\texttt{eps}\cdot\mathrm{cond}(A)$. Next we will see that we can expect it to be not much larger than 1 even for extremely ill-conditioned matrices.

OBSERVATION 3.6. *Let $A \in \mathbb{F}^{n\times n}$ be given. Denote its floating-point inverse by $R := \texttt{inv}_{\mathrm{fl}}(A)$, and define $C := \min(\texttt{eps}^{-1}, \mathrm{cond}(A))$. Then $\|R\| \sim C/\|A\|$ and*

$$\|I - RA\| \sim \texttt{eps}\cdot C. \tag{3.6}$$

In any case

$$\|RA\| \sim 1. \tag{3.7}$$

Argument. If A is not extremely ill-conditioned so that $\mathrm{cond}(A) \lesssim \mathtt{eps}^{-1}$, say, then R is an approximate inverse of reasonable quality, so that $\|R\| \sim \|A^{-1}\| = C/\|A\|$. Now suppose A is extremely ill-conditioned. Matrix inversion is commonly performed using Gaussian elimination or alike. After the first elimination step, the matrix of intermediate results is generally corrupted by rounding errors. By Observation 3.4 the condition number of this intermediate matrix can be expected to be not much larger than $\mathtt{eps}^{-1}$. Thus, if no exceptional operations such as division by zero occur, we can expect in any case $\|R\| \sim C/\|A\|$. A standard estimation (cf. [2]) states

$$|I - RA| \leq \gamma_n |R|\,|L|\,|U|$$

for an approximate inverse R computed via LU-decomposition. Note that this estimation is independent of $\mathrm{cond}(A)$ and is valid with or without pivoting. In fact, the estimation is used as an argument for the necessity of pivoting in order to diminish $|U|$. Furthermore,

$$|u_{ij}| \leq \varrho_n \cdot \max|a_{ij}|,$$

where ϱ_n denotes the growth factor. Recall the growth factor for an L,U-decomposition is defined by $\varrho(A) := \max_{i,j,k}\big|a_{ij}^{(k)}\big| / \max_{i,j}|a_{ij}|$, where $a_{ij}^{(k)}$ denote all intermediate values during the elimination process.

With pivoting we can expect ϱ_n to be small, so $\big\||L| \cdot |U|\big\| \sim \|A\|$, and (3.6) follows. For extremely ill-conditioned A we have $C \sim \mathtt{eps}^{-1}$ and $\|I - RA\| \sim 1$, and therefore (3.7). Otherwise, R is of reasonable quality, so that $RA \sim I$ and also (3.7). □

Let $A \in \mathbb{F}^{n\times n}$ be extremely ill-conditioned, and let $R := \mathtt{inv}_{\mathrm{fl}}(A)$ be an approximate inverse calculated in working precision. The main point in Algorithm 3.1 is that even for an extremely ill-conditioned matrix A its floating-point inverse R contains enough information to decrease the condition number by about a factor $\mathtt{eps}$, i.e., $\mathrm{cond}(RA) \sim \mathtt{eps} \cdot \mathrm{cond}(A)$. To see this we need a lower bound for the norm of a matrix.

LEMMA 3.7. *Let a matrix $A \in \mathbb{R}^{n\times n}$ and a vector $x \in \mathbb{R}^n$ be given which are not correlated. Then*

$$\|A\| \geq E\left(\frac{\|Ax\|}{\|x\|}\right) \geq \frac{0.61}{\sqrt{n-1}} \cdot \|A\| \tag{3.8}$$

for $\|\cdot\|_2$ and $\|\cdot\|_{\mathrm{F}}$.

Proof. With the singular value decomposition of $A = U\Sigma V^{\mathrm{T}}$ we have

$$\|Ax\|_2 = \|\Sigma V^{\mathrm{T}} x\|_2 \geq \|A\|_2 \big|v_1^{\mathrm{T}} x\big|,$$

where v_1 denotes the first column of V. Since x is generic, Theorem 3.5 proves the result. □

Geometrically this means that representing the vector x by the base of right singular vectors v_i, the coefficient of v_1 corresponding to $\|A\|$ is not too small in magnitude. This observation is also useful in numerical analysis. It offers a simple way to approximate the spectral norm of a matrix with small computational cost. Next we are ready for the anticipated result.

OBSERVATION 3.8. *Let $A \in \mathbb{F}^{n\times n}$ and $1 \le k \in \mathbb{N}$ be given, and let $\{R\}$ be a collection of matrices $R_\nu \in \mathbb{F}^{n\times n}$, $1 \le \nu \le k$. Define*

$$\begin{aligned} P &:= \mathrm{fl}_{k,1}(R \cdot A) =: RA + \Delta_1, \\ X &:= \mathtt{inv}_{\mathrm{fl}}(P), \\ \{R'\} &:= \mathrm{fl}_{k,k}(X \cdot R) =: XR + \Delta_2, \\ C &:= \min(\mathtt{eps}^{-1}, \mathrm{cond}(P)). \end{aligned} \tag{3.9}$$

Assume

$$\|R\| \sim \frac{\mathtt{eps}^{-k+1}}{\|A\|} \tag{3.10}$$

and

$$\|RA\| \sim 1 \tag{3.11}$$

and

$$\mathrm{cond}(RA) \sim \mathtt{eps}^{k-1}\,\mathrm{cond}(A). \tag{3.12}$$

Then

$$\|R'\| \sim C\ \frac{\mathtt{eps}^{-k+1}}{\|A\|}, \tag{3.13}$$

$$\|R'A\| \sim 1 \tag{3.14}$$

and

$$\mathrm{cond}(R'A) \sim \frac{\mathtt{eps}^{k-1}}{C}\,\mathrm{cond}(A). \tag{3.15}$$

REMARK 1. Note that for ill-conditioned P, i.e., $\mathrm{cond}(P) \gtrsim \mathtt{eps}^{-1}$, we have $C = \mathtt{eps}^{-1}$, and the estimations (3.13) and (3.15) read

$$\|R'\| \sim \frac{\mathtt{eps}^{-k}}{\|A\|} \quad \text{and} \quad \mathrm{cond}(R'A) \sim \mathtt{eps}^{k}\,\mathrm{cond}(A).$$

REMARK 2. Furthermore note that $RA \in \mathbb{R}^{n\times n}$ in (3.11) is the exact product of R and A. However, we will estimate the effect of rounding when computing this product in finite precision.

REMARK 3. Finally note that R denotes a collection of matrices. We omit the braces for better readability.

Argument. First note that (3.10) and the computation of P in k-fold precision imply

$$\|\Delta_1\| \lesssim \mathtt{eps}\|RA\| + \mathtt{eps}^k\|R\| \cdot \|A\| \sim \mathtt{eps}, \tag{3.16}$$

where the extra summand $\mathtt{eps}\|RA\|$ stems from the rounding of the product into a single floating-point matrix. Therefore with (3.11)

$$\|P\| \sim \|RA\| \sim 1.$$

Hence Observation 3.6 yields

$$\|X\| \sim \frac{C}{\|P\|} \sim C \quad \text{and} \quad \|I - XP\| \sim C\mathtt{eps}, \tag{3.17}$$

and with the assumption (3.10) it follows

$$\|R'\| \sim \|X\| \cdot \|R\| \sim C\ \frac{\mathtt{eps}^{-k+1}}{\|A\|}. \tag{3.18}$$

Therefore, the computation of R' in k-fold precision gives

$$\|\Delta_2\| \lesssim \mathtt{eps}^k\|X\| \cdot \|R\| \lesssim \frac{C\mathtt{eps}}{\|A\|}. \tag{3.19}$$

Putting things together, (3.9), (3.17), (3.16) and (3.19) imply

$$\|I - R'A\| = \|I - XP + X\Delta_1 - \Delta_2 A\| \lesssim C\mathtt{eps} + C\mathtt{eps} + C\mathtt{eps} \sim C\mathtt{eps}. \tag{3.20}$$

The definition of C implies $\|I - R'A\| \lesssim 1$, so that (3.14) follows.

To see (3.15) suppose first that P is not extremely ill-conditioned, so that in view of (3.9) and (3.16) we have $C \sim \mathrm{cond}(RA) < \mathtt{eps}^{-1}$. Then $R'A$ is not far from the identity matrix by (3.20), which means $\mathrm{cond}(R'A) \sim 1 \sim C^{-1}\,\mathrm{cond}(RA) \sim C^{-1}\mathtt{eps}^{k-1}\,\mathrm{cond}(A)$ using (3.12).

Second, suppose P is extremely ill-conditioned, so that $C = \mathtt{eps}^{-1}$. For that case we give two arguments for (3.15). In that case (3.13) and (3.14) imply

$$\mathtt{eps}^{-k}/\|A\| \sim \|R'\| \le \|R'A\| \cdot \|A^{-1}\|$$

and therefore $\mathrm{cond}(A) \gtrsim \mathtt{eps}^{-k}$. Denote the ν-th column of R' by $r'^{(\nu)}$. Then

$$\|(R'A)^{-1}r'^{(\nu)}\| = \|A^{-1}e_\nu\| \sim \|A^{-1}\|,$$

where e_ν denotes the ν-th column of the identity matrix. We have $\mathrm{cond}(P) \sim \mathtt{eps}^{-1}$ by Observation 3.4, so that the entries of X differ from those of P^{-1} in the first digit. This introduces enough randomness into the coefficients of $R' = \mathrm{fl}_{k,k}(RA)$,

so that Lemma 3.7 is applicable. Using $\|R'\| \sim \|r'^{(\nu)}\|$, (3.18) and $C = \texttt{eps}^{-1}$ we can expect

$$\|(R'A)^{-1}\| \sim \frac{\|A^{-1}\|}{\|r'^{(\nu)}\|} \sim \texttt{eps}^k \cdot \mathrm{cond}(A).$$

Thus, (3.14) implies (3.15). The second argument uses the fact that by (3.14) R' is a preconditioner of A. This means that the singular values of R' annihilate those of A, so that accurately we would have $\sigma_\nu(R') \sim 1/\sigma_{n+1-\nu}(A)$. However, the accuracy is limited by the precision $\texttt{eps}^k$ of $R' = \mathrm{fl}_{k,k}(XR)$. So because $\sigma_1(A)/\sigma_n(A) = \mathrm{cond}(A) \gtrsim \texttt{eps}^{-k}$, the computed singular values of R' increase starting with the smallest $\sigma_n(R') = 1/\sigma_1(A)$ only until $\texttt{eps}^{-k}/\sigma_1(A)$. Note this corresponds to $\|R'\| \sim \texttt{eps}^{-k}/\|A\|$ in (3.13). Hence the large singular values of $R'A$, which are about 1, are annihilated to 1, but the smallest will be $\texttt{eps}^{-k}/\sigma_1(A) \cdot \sigma_n(A)$. Therefore

$$\|(R'A)^{-1}\| \sim 1 \cdot \texttt{eps}^k \sigma_1(A)/\sigma_n(A) = \texttt{eps}^k \,\mathrm{cond}(A),$$

and again using (3.14) finishes the argument. □

Observation 3.9. Algorithm 3.1 *(*`InvIllCoPrel`*) will stop for nonsingular* $A \in \mathbb{F}^{n\times n}$ *after some*

$$k \sim \frac{2 + \log_{10} \mathrm{cond}(A)}{-\log_{10} \texttt{eps}}$$

steps. For the computed approximate inverse R *we can expect*

$$\|I - RA\| \lesssim \texttt{eps} \cdot \mathrm{cond}(P^{(k)}) \lesssim 0.01.$$

If an additional iteration is performed, then $\|I - RA\| \sim \texttt{eps}$.

Remark 4. Basically, the condition number of RA improves by a factor `eps` in each iteration. For example, for $\mathrm{cond}(A) \sim 10^{100}$ and computing in IEEE 754 double precision with $\texttt{eps} = 2^{-53} \sim 10^{-16}$, the algorithm should finish after some 6 or 7 iterations.

Argument. The assumptions (3.10), (3.11) and (3.12) of Observation 3.8 are obviously satisfied for $k = 0$. The results follow by an induction argument and a straightforward computation. □

For the final version of our Algorithm 3.1 (`InvIllCoPrel`) we introduce two improvements. First, as has been mentioned before, the floating-point inversion of $P^{(k)}$ may fail in floating-point. Although this is very rare, we can cure it by a afflicting $P^{(k)}$ with a random relative perturbation of size $\texttt{eps}|P^{(k)}|$. Since $P^{(k)}$ must be extremely ill-conditioned if the inversion fails, it is clear that such a perturbation does not change the analysis.

Second, a suitable approximation of the condition number of $P^{(k)}$ is already at hand with the approximate inverse $X^{(k)}$, namely $\mathrm{cond}(P^{(k)}) \sim \|X^{(k)}\| \cdot \|P^{(k)}\|$. Hence the stopping criterion contains only computed quantities.

For the final Algorithm 3.10 (`InvIllCo`) the analysis as in Observation 3.9 is obviously valid as well. In the final version we also overwrite variables. Moreover we perform a final iteration after the stopping criterion $\|P\| \cdot \|X\| < \texttt{eps}^{-1}/100$ is satisfied to ensure $\|I - RA\| \sim \texttt{eps}$. Finally, we start with a scalar for R to cover the case that the floating-point inversion of A fails.

ALGORITHM 3.10. *Inversion of an extremely ill-conditioned matrix, final version.*

---

function $\{R\} = \texttt{InvIllCo}(A)$
  $R = \mathrm{fl}(1/\|A\|)$; $P = X = \infty$; $k = 0$
  repeat
    finished $= (\|P\| \cdot \|X\| < \texttt{eps}^{-1}/100)$;
    $k = k + 1$
    $P = \mathrm{fl}_{k,1}(R \cdot A)$ *% stored in working precision*
    $X = \mathrm{inv}_{\mathrm{fl}}(P)$ *% floating-point inversion in working precision*
    while "inversion failed"
      $P = P + \Delta P$ *% relative perturbation* ΔP *of size* $\texttt{eps}|P|$
      $X = \mathrm{inv}_{\mathrm{fl}}(P)$ *% floating-point inversion in working precision*
    end while
    $\{R\} = \mathrm{fl}_{k,k}(X \cdot R)$ *% stored in k-fold precision*
  until finished

We mention that in Algorithm `AccInv3` in [8] the stopping criterion $\|P - I\| < 10^{-3}$ was used. Moreover, in that algorithm all matrix products were calculated with faithful rounding using our algorithms in [13, 12]. The same stopping criterion can be used in Algorithm 3.10, however, our criterion avoids one extra multiplication $P = \mathrm{fl}_{k,1}(R \cdot A)$.

If the algorithm is used to prove nonsingularity of the matrix A by $\|I - RA\| < 1$, then the error in the computation of $\|P - I\|$ has to be estimated. It seems preferable to use Algorithm 3.10 with computations in k-fold precision and prove $\|I - RA\| < 1$ by computing $\|I - RA\|$ with faithful rounding. In that case only one matrix product has to be computed with faithful rounding. However, the advantage is marginal since our algorithms to compute a matrix product with faithful rounding are almost as fast as computation in k-fold precision, sometimes even faster.

4. Computational results

All computational results are performed in Matlab [5] using IEEE 754 [3] double precision as working precision. Note that this is the only precision used.

The classical example of an ill-conditioned matrix is the Hilbert matrix, the ij-th component being $1/(i + j - 1)$. However, due to rounding errors the floating-point Hilbert matrix is not as ill-conditioned as the original one, in accordance to Observation 3.4. This can nicely be seen in Fig. 4.1, where the condition number

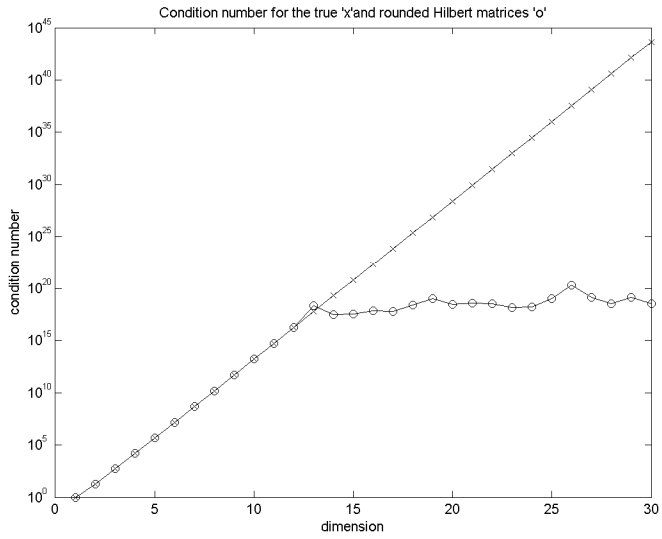

Fig. 4.1. Condition number of the true (×) and rounded Hilbert matrices (○).

of the rounded Hilbert matrices are limited to about $\texttt{eps}^{-1}$. Therefore we use the Hilbert matrix scaled by the least common multiple of the denominators, i.e.,

$$H_n \in \mathbb{F}^{n\times n} \quad \text{with } h_{ij} := \texttt{lcm}(1, 2, \ldots, 2n-1)/(i+j-1).$$

The largest dimension for which this matrix is exactly representable in double precision floating-point is $n = 21$ with $\mathrm{cond}(H_{21}) = 8.4 \cdot 10^{29}$. The results of Algorithm 3.10 are shown in Table 4.1. All results displayed in the following tables are computed in infinite precision using the symbolic toolbox of Matlab.

Table 4.1. Computational results of Algorithm 3.10 (`InvIllCo`) for the Hilbert 21×21 matrix.

k	$\mathrm{cond}(R)$	$\mathrm{cond}(RA)$	$\mathrm{cond}_{\mathrm{mult}}(R \cdot A)$	$\mathrm{cond}(P)$	$\mathrm{cond}_{\mathrm{mult}}(X \cdot R)$	$\|I - R'A\|$
1	21.0	$8.44 \cdot 10^{29}$	2.00	$1.67 \cdot 10^{18}$	2.06	158
2	$7.06 \cdot 10^{18}$	$8.57 \cdot 10^{15}$	$9.78 \cdot 10^{16}$	$8.55 \cdot 10^{15}$	7.96	0.06
3	$8.49 \cdot 10^{29}$	21.0	$6.66 \cdot 10^{30}$	21.0	2.06	$9.03 \cdot 10^{-16}$
4	$8.44 \cdot 10^{29}$	21.0	$4.53 \cdot 10^{46}$	21.0	2.00	$3.32 \cdot 10^{-16}$

The meaning of the displayed results is as follows. After the iteration counter k we display in the second and third column the condition number of R and of RA, respectively, *before* entering the k-th loop. So in Table 4.1 the first result $\mathrm{cond}(R) = 21.0$ is the Frobenius norm condition number of the scaled identity matrix, which is $\sqrt{n} \cdot \sqrt{n} = n$. In the following fourth column "$\mathrm{cond}_{\mathrm{mult}}(R \cdot A)$" we display the product condition number of the matrix product R times A (not of the result of the product). The condition number of a dot product $x^{\mathrm{T}}y$ is well-known to

be $\mathrm{cond}(x^{\mathrm{T}}y) = 2|x^{\mathrm{T}}|\,|y|/|x^{\mathrm{T}}y|$, where the absolute value is taken componentwise. Therefore we display

$$\mathrm{cond}_{\mathrm{mult}}(R \cdot A) = \mathtt{median}\left\{ 2\frac{(|R|\,|A|)_{ij}}{|RA|_{ij}} \right\}$$

with the `median` taken over all n^2 components. It follows the condition number of P, and similarly $\mathrm{cond}_{\mathrm{mult}}(X \cdot R)$ denotes the condition number of the matrix product X times R. Finally, the last column shows $\|I - RA\|$ as an indicator of the quality of the preconditioner R.

We mention again that all quantities in the tables are computed with the symbolic toolbox of Matlab in infinite precision. For larger matrices this requires quite some computing time.

For better understanding we display the results of a much more ill-conditioned matrices. The first example is the following matrix A_6 with $\mathrm{cond}(A_6) = 6.2 \cdot 10^{93}$, also taken from [10].

$$A_6 = \begin{pmatrix}
1810371096161830 & -2342429902418850 & 2483279165876947 & -3747279334964253 & -3262424857701958 & -3083142983440029 \\
-4670543938170405 & -1397606024304777 & 60011496034489 & 1689416277492541 & -1500903035774466 & 3966198838365752 \\
-1064600276464066 & -7561599142730362 & 4805299117903646 & -6663945806445749 & -7071076225762059 & -52156788818356 \\
13002500911063530 & 2223055422646289 & -1553584081743862 & -5252100317685933 & 7524433713832350 & -6396043249912008 \\
-395183142691090 & -2180846347388541 & 1450541682835654 & -362949820992570 0 & -1866168768872108 & 1230298410784196 \\
2337744233608461 & 1359019382927754 & 1241733688092475 & 1803080888028433 & -2648685047371017 & -7046414836143443
\end{pmatrix}.$$

Then $\|\mathtt{inv}_{\mathrm{fl}}(A_6)\| = 12.4$ for the floating-point approximate inverse of A_6, whereas $\|A_6^{-1}\| = 2.4 \cdot 10^{77}$ for the true inverse of A_6. So the approximate inverse and the true inverse differ by about 76 orders of magnitude. Nevertheless there is enough information hidden in the approximate inverse to serve as a reasonable preconditioner so that $\mathrm{cond}(RA)$ is about `eps` times $\mathrm{cond}(A)$.

The computational results of Algorithm 3.10 (`InvIllCo`) for the matrix A_6 are displayed in Table 4.2. Let us interpret them. As expected (see (3.13)) the condition number of R increases in every step by about a factor $\mathtt{eps}^{-1}$ up to about the final condition number of the input matrix A_6. According to (3.12) the condition number of RA decreases in each step by a factor `eps` down to about 1 in the last iteration. The quality of R as a preconditioner becomes better and better, and thus the product condition number of the matrix product $R \cdot A$ must increase. This also shows why it is mandatory to calculate this product in k-fold precision.

The result of the product $R \cdot A$ computed in k-fold precision is rounded into working precision, into the matrix P. The smoothing effect of rounding as explained in Observation 3.4 limits the condition number of P to about $\mathtt{eps}^{-1}$, as is seen in the fifth column of Table 4.2. The product condition number of the matrix product $X \cdot R$ stays close to 1, but nevertheless it is mandatory to compute the product $R = \mathrm{fl}_{k,k}(X \cdot R)$ with increasing number of results to achieve the intended quality of the preconditioner R.

We note that for $k = 2$ and for $k = 3$ the floating-point inversion failed so that P had to be slightly perturbed. Obviously this is not visible in Table 4.2.

Table 4.2. Computational results of Algorithm 3.10 (`InvIllCo`) for the matrix A_6.

k	$\mathrm{cond}(R)$	$\mathrm{cond}(RA)$	$\mathrm{cond}_{\mathrm{mult}}(R \cdot A)$	$\mathrm{cond}(P)$	$\mathrm{cond}_{\mathrm{mult}}(X \cdot R)$	$\|I - R'A\|$
1	6.00	$6.21 \cdot 10^{93}$	2.00	$9.90 \cdot 10^{17}$	11.1	37.4
2	$4.55 \cdot 10^{18}$	$5.15 \cdot 10^{79}$	$4.56 \cdot 10^{16}$	$2.16 \cdot 10^{18}$	162	7.88
3	$2.19 \cdot 10^{31}$	$1.46 \cdot 10^{64}$	$6.98 \cdot 10^{30}$	$3.39 \cdot 10^{18}$	3.85	9.54
4	$1.87 \cdot 10^{47}$	$4.13 \cdot 10^{48}$	$2.37 \cdot 10^{46}$	$1.24 \cdot 10^{17}$	9.06	4.71
5	$6.42 \cdot 10^{62}$	$3.41 \cdot 10^{32}$	$1.08 \cdot 10^{62}$	$1.99 \cdot 10^{20}$	2.08	6.26
6	$2.18 \cdot 10^{79}$	$9.95 \cdot 10^{15}$	$5.51 \cdot 10^{78}$	$9.91 \cdot 10^{15}$	2.16	0.98
7	$7.00 \cdot 10^{93}$	6.78	$1.42 \cdot 10^{94}$	6.78	2.89	$2.02 \cdot 10^{-16}$

Finally we observe the quality of R as a preconditioner. In all iterations but the last one the norm of $I - RA$ is about 1 in agreement with (3.14). In the final iteration, $\|I - RA\|$ becomes about `eps`, which is best possible.

The results for the Hilbert 21×21-matrix as displayed in Table 4.1 are very similar. For comparison we display the behavior of the modified algorithm as considered in [9] for H_{21}. The artificial, relative perturbation of size $\sqrt{\texttt{eps}}$ of P reduces the improvement of R as a preconditioner to a factor $\sqrt{\texttt{eps}}$ rather than `eps`. This can be nicely observed in Table 4.3.

Table 4.3. Computational results of the modified algorithm [9] for the Hilbert 21×21 matrix.

k	$\mathrm{cond}(R)$	$\mathrm{cond}(RA)$	$\mathrm{cond}_{\mathrm{mult}}(R \cdot A)$	$\mathrm{cond}(P)$	$\mathrm{cond}_{\mathrm{mult}}(X \cdot R)$	$\|I - R'A\|$
1	21.0	$8.44 \cdot 10^{29}$	2.00	$1.65 \cdot 10^{10}$	6.77	10.1
2	$1.65 \cdot 10^{10}$	$7.90 \cdot 10^{21}$	$1.95 \cdot 10^{9}$	$3.00 \cdot 10^{10}$	5.79	53.3
3	$4.10 \cdot 10^{18}$	$8.40 \cdot 10^{14}$	$1.42 \cdot 10^{17}$	$9.10 \cdot 10^{10}$	2.27	91.3
4	$1.29 \cdot 10^{26}$	$5.47 \cdot 10^{7}$	$2.83 \cdot 10^{24}$	$5.47 \cdot 10^{7}$	4.19	0.01
5	$8.44 \cdot 10^{29}$	21.0	$1.86 \cdot 10^{32}$	21.0	2.00	$3.73 \cdot 10^{-8}$
6	$8.44 \cdot 10^{29}$	21.0	$2.17 \cdot 10^{40}$	21.0	2.00	$3.61 \cdot 10^{-8}$
7	$8.44 \cdot 10^{29}$	21.0	$3.76 \cdot 10^{48}$	21.0	2.00	$3.17 \cdot 10^{-8}$

For clarity we display in Table 4.3 the results for the following iterations after $\|I - RA\|$ is already of the order $\sqrt{\texttt{eps}}$. As can be seen $\mathrm{cond}(R)$ does not increase for $k > 5$ because R is already very close to the true inverse of A. Nevertheless the off-diagonal elements of RA are better and better approximations of zero, so that the condition number $\mathrm{cond}_{\mathrm{mult}}(R \cdot A)$ of the product R times A still increases with k. Note that due to the artificial perturbation of P the norm of the residual $\|I - RA\|$ does not decrease below $\sqrt{\texttt{eps}}$.

Finally we will show some results for a truly extremely ill-conditioned matrix. It is not that easy to construct such matrices which are exactly representable in floating-point. In [10] an algorithm is presented to construct such matrices in an arbitrary floating-point format.[1] The matrix A_4 in (1.2) and A_6 are produced by

[1]Recently Prof. Tetsuo Nishi from Waseda University presented a refined discussion of this approach at the international workshop INVA08 at Okinawa in March 2008, and at the NOLTA conference in Budapest in 2008, cf. [6].

this algorithm. It is included as Algorithm "randmat.m" in INTLAB [11], the Matlab toolbox for reliable computing.

Using Algorithm "randmat.m" we generated a 50×50 matrix with condition number $\mathrm{cond}(A) = 7.4 \cdot 10^{305}$. The results of Algorithm 3.10 (`InvIllCo`) for this matrix are displayed in Table 4.4. Our algorithm easily allows much larger dimensions, however, the symbolic computation of the results in Table 4.4 took about a day on a Laptop.

Table 4.4. Computational results of Algorithm 3.10 (`InvIllCo`) for a 50×50 matrix with condition number $1.14 \cdot 10^{305}$.

k	$\mathrm{cond}(R)$	$\mathrm{cond}(RA)$	$\mathrm{cond}_{\mathrm{mult}}(R \cdot A)$	$\mathrm{cond}(P)$	$\mathrm{cond}_{\mathrm{mult}}(X \cdot R)$	$\|I - R'A\|$
1	50.0	$7.36 \cdot 10^{305}$	2.00	$4.59 \cdot 10^{18}$	7.72	30.7
2	$3.25 \cdot 10^{18}$	$1.39 \cdot 10^{291}$	$2.94 \cdot 10^{16}$	$1.39 \cdot 10^{291}$	62.3	106
3	$2.60 \cdot 10^{33}$	$8.89 \cdot 10^{276}$	$9.07 \cdot 10^{30}$	$5.14 \cdot 10^{23}$	68.2	65.5
4	$6.40 \cdot 10^{47}$	$1.58 \cdot 10^{263}$	$1.38 \cdot 10^{45}$	$8.86 \cdot 10^{18}$	138	121
5	$1.29 \cdot 10^{63}$	$5.26 \cdot 10^{249}$	$1.81 \cdot 10^{59}$	$3.47 \cdot 10^{18}$	669	36.7
6	$9.16 \cdot 10^{74}$	$1.23 \cdot 10^{235}$	$4.36 \cdot 10^{72}$	$1.43 \cdot 10^{18}$	48.9	5.32
7	$8.62 \cdot 10^{87}$	$3.85 \cdot 10^{219}$	$1.44 \cdot 10^{87}$	$9.97 \cdot 10^{17}$	4.93	35.7
8	$3.04 \cdot 10^{105}$	$9.30 \cdot 10^{206}$	$4.47 \cdot 10^{102}$	$9.92 \cdot 10^{17}$	$2.79 \cdot 10^{3}$	8.53
9	$1.47 \cdot 10^{116}$	$1.37 \cdot 10^{192}$	$3.87 \cdot 10^{115}$	$4.59 \cdot 10^{17}$	20.9	23.5
10	$1.68 \cdot 10^{131}$	$2.17 \cdot 10^{177}$	$2.20 \cdot 10^{130}$	$4.81 \cdot 10^{17}$	16.8	4.54
11	$6.19 \cdot 10^{145}$	$7.42 \cdot 10^{161}$	$5.67 \cdot 10^{145}$	$4.28 \cdot 10^{17}$	6.51	3.63
12	$1.03 \cdot 10^{161}$	$1.95 \cdot 10^{146}$	$7.34 \cdot 10^{160}$	$1.58 \cdot 10^{18}$	3.67	5.83
13	$5.99 \cdot 10^{176}$	$5.79 \cdot 10^{130}$	$4.46 \cdot 10^{176}$	$9.72 \cdot 10^{17}$	3.86	5.44
14	$2.28 \cdot 10^{192}$	$1.43 \cdot 10^{115}$	$3.31 \cdot 10^{192}$	$3.33 \cdot 10^{17}$	5.31	8.04
15	$1.05 \cdot 10^{208}$	$5.27 \cdot 10^{99}$	$1.41 \cdot 10^{208}$	$9.76 \cdot 10^{17}$	9.31	152
16	$5.55 \cdot 10^{224}$	$1.07 \cdot 10^{86}$	$2.22 \cdot 10^{223}$	$4.93 \cdot 10^{18}$	225	28.2
17	$3.57 \cdot 10^{238}$	$8.91 \cdot 10^{70}$	$2.39 \cdot 10^{237}$	$3.36 \cdot 10^{18}$	19.0	10.1
18	$2.75 \cdot 10^{252}$	$7.20 \cdot 10^{55}$	$3.68 \cdot 10^{252}$	$4.54 \cdot 10^{17}$	11.8	7.18
19	$1.91 \cdot 10^{267}$	$2.08 \cdot 10^{40}$	$3.54 \cdot 10^{267}$	$2.39 \cdot 10^{18}$	4.32	21.7
20	$1.48 \cdot 10^{283}$	$2.45 \cdot 10^{25}$	$8.92 \cdot 10^{282}$	$3.18 \cdot 10^{18}$	13.0	7.80
21	$1.24 \cdot 10^{298}$	$4.55 \cdot 10^{9}$	$2.23 \cdot 10^{298}$	$4.55 \cdot 10^{9}$	8.53	$1.58 \cdot 10^{-7}$
22	$7.36 \cdot 10^{305}$	50.0	Inf	50.0	2.00	$5.64 \cdot 10^{-16}$

We can nicely observe how the condition number of R as well as $\mathrm{cond}_{\mathrm{mult}}(R \cdot A)$ increase in each step by a factor $\mathtt{eps}^{-1}$, and eventually $\|I - RA\|$ becomes less than 1 after some 22 iterations. Note that we expect this to happen after about $\lceil 305/16 \rceil + 1 = 21$ iterations. The condition number $\mathrm{cond}(A) = 7.4 \cdot 10^{305}$ of the matrix is close to the overflow range, the largest floating-point number in double precision is $1.8 \cdot 10^{308}$.

Note in particular that for $k = 2$ we have the exceptional value $\mathrm{cond}(P) = 1.39 \cdot 10^{291} \approx \mathrm{cond}(RA)$. This may happen accidentally, but obviously it does not influence the general behavior of the iteration.

In the last iteration $\mathrm{cond}_{\mathrm{mult}}(R \cdot A)$ causes already overflow. In this particular example the artificial relative perturbation of P in Algorithm 3.10 (`InvIllCo`) by about `eps` was not necessary.

Finally we note a question posed by Prof. Kunio Tanabe from Waseda University about the distribution of singular values of the generated, extremely ill-conditioned matrices and the possible effect on the performance of our algorithm. Indeed it is likely that for matrices generated by the method in [10] the singular values σ_1 to σ_{n-1} are close to $\|A\|$, whereas σ_n is extremely small.

However, our analysis showed no relevance of this to our algorithm. Fortunately we may enter a sum of matrices $\sum A_i$ into Algorithm 3.10 (`InvIllCo`) to check on this. We choose the original Hilbert matrix with $h_{ij} := 1/(i+j-1)$ and approximate the individual entries by several floating-point numbers. As is well known, the singular values of the Hilbert matrix cover the interval $[\sigma_n, \sigma_1]$ linearly on a logarithmic scale.

As an example we show the result of the Hilbert 50×50 matrix stored as a sum of 5 matrices. As can be seen in Table 4.5 the computational behavior of Algorithm 3.10 (`InvIllCo`) is quite similar to the previous examples.

Table 4.5. Computational results of Algorithm 3.10 (`InvIllCo`) for the Hilbert 50×50 matrix stored as the sum of 5 matrices.

k	$\mathrm{cond}(R)$	$\mathrm{cond}(RA)$	$\mathrm{cond}_{\mathrm{mult}}(R \cdot A)$	$\mathrm{cond}(P)$	$\mathrm{cond}_{\mathrm{mult}}(X \cdot R)$	$\|I - R'A\|$
1	50.0	$1.50 \cdot 10^{74}$	2.00	$1.03 \cdot 10^{19}$	5.25	89.7
2	$7.17 \cdot 10^{18}$	$3.35 \cdot 10^{59}$	$1.85 \cdot 10^{17}$	$2.03 \cdot 10^{19}$	12.6	874
3	$7.00 \cdot 10^{34}$	$8.03 \cdot 10^{46}$	$6.61 \cdot 10^{31}$	$2.65 \cdot 10^{20}$	8.57	345
4	$1.34 \cdot 10^{48}$	$8.85 \cdot 10^{32}$	$4.28 \cdot 10^{45}$	$5.35 \cdot 10^{19}$	19.9	33.4
5	$7.81 \cdot 10^{60}$	$1.54 \cdot 10^{18}$	$1.34 \cdot 10^{59}$	$1.99 \cdot 10^{18}$	22.8	6.76
6	$2.64 \cdot 10^{73}$	417	$2.71 \cdot 10^{72}$	417	7.24	$2.02 \cdot 10^{-14}$
7	$1.50 \cdot 10^{74}$	50.0	$7.24 \cdot 10^{87}$	50.0	2.00	$4.76 \cdot 10^{-16}$

Appendix.

In the following we present auxiliary results, proofs and arguments for some results of the previous section.

Proof of Lemma 3.2. We have

$$E(\|x\|_2) = \int_{-d_n}^{d_n} \cdots \int_{-d_1}^{d_1} \|x\|_2 \, dx_1 \cdots dx_n \Big/ \prod_{i=1}^{n} (2d_i) =: I \Big/ \prod_{i=1}^{n} d_i,$$

where

$$I := \int_{0}^{d_n} \cdots \int_{0}^{d_1} \|x\|_2 \, dx_1 \cdots dx_n.$$

Now $\sqrt{n} \cdot \|x\|_2 \geq \|x\|_1$, so $x \geq 0$ implies

$$\sqrt{n} \cdot I \geq \int_0^{d_n} \cdots \int_0^{d_1} \sum_{i=1}^{n} x_i \, dx_1 \cdots dx_n = \frac{1}{2} \left(\prod_{i=1}^{n} d_i \right) \left(\sum_{i=1}^{n} d_i \right)$$

and therefore the left inequality in (3.2). Moreover,

$$\int_0^c \sqrt{x^2 + a} \, dx \leq c \sqrt{c^2 + a} \quad \text{for } a, c \geq 0,$$

so

$$I \leq d_1 \int_0^{d_n} \cdots \int_0^{d_2} \left(d_1^2 + \sum_{i=2}^{n} x_i^2 \right)^{1/2} dx_2 \cdots dx_n \leq \cdots \leq \left(\prod_{i=1}^{n} d_i \right) \cdot \|d\|_2.$$

The proof is finished. □

The surface $S(n)$ of the n-dimensional unit sphere satisfies $S(n) = 2\pi^{n/2} / \Gamma(n/2)$. Recall the Gamma function defined by $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} \, dt$ interpolates the factorial by $(n-1)! = \Gamma(n)$. To prove Theorem 3.5 we need to estimate the surface of a cap of the sphere. For this we need the following lemma.

LEMMA A.1. *For $x > 1$ we have*

$$\frac{1}{\sqrt{x - 1/2}} < \frac{\Gamma(x - 1/2)}{\Gamma(x)} < \frac{1}{\sqrt{x - 1}}.$$

Proof. The function $\Gamma(x-1/2)/\Gamma(x)$ is monotonically decreasing for $x > 0.5$, so it follows

$$\begin{aligned} \frac{1}{x - 1/2} &= \frac{\Gamma(x - 1/2)}{\Gamma(x + 1/2)} = \frac{\Gamma(x - 1/2)}{\Gamma(x)} \cdot \frac{\Gamma(x)}{\Gamma(x + 1/2)} \\ &< \left(\frac{\Gamma(x - 1/2)}{\Gamma(x)} \right)^2 < \frac{\Gamma(x - 1/2)}{\Gamma(x)} \cdot \frac{\Gamma(x - 1)}{\Gamma(x - 1/2)} = \frac{1}{x - 1}. \end{aligned}$$

□

Proof of Theorem 3.5. Denote by $\mathcal{O}_n(\Phi)$ the surface of the cap of the n-dimensional unit sphere with opening angle Φ, see Fig. A.1. Then

$$\mathcal{O}_n(\Phi) = 2\mathcal{O}_{n-1}(\pi/2) \cdot \int_0^{\Phi} \sin^{n-2} \varphi \, d\varphi \tag{A.1}$$

and

$$\mathcal{O}_n(\pi/2) = \frac{\pi^{n/2}}{\Gamma(n/2)}, \tag{A.2}$$

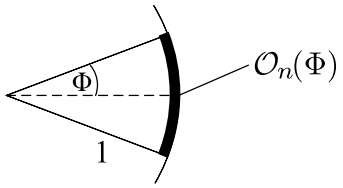

Fig. A.1.

the latter denoting half of the surface of the n-dimensional sphere. The expected value of $|x^{\mathrm{T}}y| = |\cos \measuredangle(x, y)|$ satisfies $E(|x^{\mathrm{T}}y|) = \cos \hat{\Phi}$ for the angle $0 < \hat{\Phi} < \pi/2$ with

$$\mathcal{O}_n(\hat{\Phi}) = \frac{1}{2}\mathcal{O}_n(\pi/2). \tag{A.3}$$

Now

$$\mathcal{O}_n(\Phi_1) \geq \frac{1}{2}\mathcal{O}_n(\pi/2) \geq \mathcal{O}_n(\Phi_2) \iff \cos \Phi_1 \leq E(|x^{\mathrm{T}}y|) \leq \cos \Phi_2. \tag{A.4}$$

We will identify such angles Φ_1 and Φ_2 to obtain bounds for $\hat{\Phi}$. Define

$$R := \frac{1}{2}\frac{\mathcal{O}_n(\pi/2)}{\mathcal{O}_{n-1}(\pi/2)}. \tag{A.5}$$

Then

$$R = \frac{\sqrt{\pi}}{2}\frac{\Gamma(n/2 - 1/2)}{\Gamma(n/2)} = \int_0^{\pi/2} \sin^{n-2} \varphi \, d\varphi \tag{A.6}$$

by (A.2) and (A.1), so that

$$\int_0^{\Phi} \sin^k \varphi \, d\varphi = \int_0^{\pi/2} \sin^k \varphi \, d\varphi - \int_0^{\pi/2-\Phi} \cos^k \varphi \, d\varphi \tag{A.7}$$

and $1 - \varphi^2/2 \leq \cos \varphi \leq 1$ for $0 \leq \varphi \leq \pi/2$ and (A.6) yield

$$R - \alpha \leq \int_0^{\Phi} \sin^{n-2} \varphi \, d\varphi \leq R - \alpha\left(1 - \frac{(n-2)\alpha^2}{6}\right) \tag{A.8}$$

with $\alpha := \pi/2 - \Phi$. Set

$$\Phi_1 := \frac{\pi}{2} - \frac{\sqrt{\pi}}{4} \cdot \frac{1}{\sqrt{n/2 - 1/2}}. \tag{A.9}$$

Then Lemma A.1 and (A.6) give

$$\frac{\pi}{2} - \Phi_1 \leq \frac{\sqrt{\pi}}{4}\frac{\Gamma(n/2 - 1/2)}{\Gamma(n/2)} = \frac{1}{2}R, \tag{A.10}$$

and using (A.1), (A.8), (A.10) and (A.5) we see $\mathcal{O}_n(\Phi_1) \geq 2\mathcal{O}_{n-1}(\pi/2)\cdot\left(R-\frac{1}{2}R\right) = \frac{1}{2}\mathcal{O}_n(\pi/2)$. Hence (A.4) implies for $n \geq 4$,

$$E(|x^{\mathrm{T}}y|) \geq \cos\Phi_1 = \sin\frac{\sqrt{2\pi}}{4\sqrt{n-1}} =: \sin\beta > \beta(1-\beta^2/6) > \frac{0.61}{\sqrt{n-1}}.$$

This proves the first inequality in (3.5). To see the second one define

$$\Phi_2 := \pi/2 - \gamma\frac{\sqrt{2\pi}}{4\sqrt{n-2}} \quad \text{with } \gamma := 1.084.$$

Then $\frac{\pi}{48}\gamma^3 - \gamma + 1 < 0$ implies

$$\gamma^{-1} < 1 - \frac{n-2}{6}\cdot\frac{\pi}{16}\cdot\frac{2}{n-2}\gamma^2$$

and

$$\frac{\sqrt{2\pi}}{4\sqrt{n-2}} = \delta\cdot\gamma^{-1} < \delta\left(1-\frac{n-2}{6}\delta^2\right) \quad \text{for } \delta := \gamma\frac{\sqrt{2\pi}}{4\sqrt{n-2}} = \pi/2 - \Phi_2.$$

Hence (A.6) and Lemma A.1 give

$$\frac{1}{2}R = \frac{\sqrt{\pi}}{4}\frac{\Gamma(n/2-1/2)}{\Gamma(n/2)} < \frac{\sqrt{\pi}}{4\sqrt{n/2-1}} < \delta\left(1-\frac{n-2}{6}\delta^2\right),$$

so that with (A.1), (A.8) and (A.5) we have

$$\mathcal{O}_n(\Phi_2) \leq 2\mathcal{O}_{n-1}(\pi/2)\cdot\left(R-\frac{1}{2}R\right) = \frac{1}{2}\mathcal{O}_n(\pi/2).$$

Thus (A.4) yields

$$E(|x^{\mathrm{T}}y|) \leq \cos\Phi_2 = \sin\frac{\gamma\sqrt{2\pi}}{4\sqrt{n-2}} \leq \frac{0.68}{\sqrt{n-2}}.$$

The theorem is proved.

Next we come to the Observation 3.4. We give two different arguments. Let $A \in \mathbb{R}^{n\times n}$ be generic on the variety $\mathcal{S}$ of singular $n\times n$-matrices. Then the normal vector N to $\mathcal{S}$ in A is well-defined. The situation is as in Fig. A.2, where $\tilde{A} := \mathrm{fl}(A)$. Assuming $\mathcal{S}$ to be locally linear, the distance $d := \min\{\|\tilde{A}-B\|_{\mathrm{F}}\colon \det B = 0\}$ of $\tilde{A}$ to the nearest singular matrix satisfies

$$d = \|\tilde{A}^{-1}\|_{\mathrm{F}}^{-1} = \|\tilde{A}-A\|_{\mathrm{F}}\cdot\cos\varphi \tag{A.11}$$

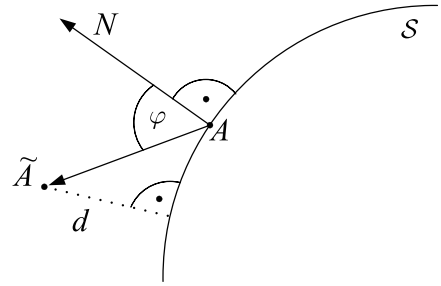

Fig. A.2. Distance of fl(A) to the nearest singular matrix.

since $\|\cdot\|_{\mathrm{F}}$ is unitarily invariant. Accepting that $\tilde{A}$ is generic to $\mathcal{S}$ we use Theorem 3.5 to see

$$\frac{0.61}{\sqrt{n^2-1}} \le E(\cos\varphi) \le \frac{0.68}{\sqrt{n^2-2}}.$$

Then (A.11) and Corollary 3.3 show (3.4).

For the second argument, let u and v denote a left and right singular vector of A to the singular value $\sigma_n(A) = 0$, respectively. Note that $\sigma_{n-1}(A) > 0$ for generic A. Denote $\tilde{A} = \mathrm{fl}(A) = A + E$. First-order perturbation theory for singular values tells

$$|\sigma_n(A+E) - \sigma_n(A)| = |u^{\mathrm{T}} E v| + \mathcal{O}(\mathtt{eps}).$$

Assuming E and v are general to each other, Lemma 3.7 gives

$$\|Ev\| \ge \frac{0.61}{\sqrt{n-1}}\|E\|.$$

For simplicity we assume the entries of A to be of similar magnitude. Then we can expect $\|E\| \sim \mathtt{eps}\|A\|_{\mathrm{F}}$ by Corollary 3.3. Assuming Ev is general to u we may apply Theorem 3.5 to see

$$|u^{\mathrm{T}} E v| \ge \frac{0.61}{\sqrt{n-1}}\|Ev\| \sim \frac{0.4\mathtt{eps}}{n}\|A\|_{\mathrm{F}}.$$

Since $\sigma_n(A) = 0$, $\sigma_n(A+E) \approx |u^{\mathrm{T}} E v|$ is the distance $\delta = \|\tilde{A}^{-1}\|_2^{-1}$ of $A+E$ to the nearest singular matrix in the spectral norm. Hence

$$\mathrm{cond}_{\mathrm{F}}(\tilde{A}) = \delta^{-1}\|\tilde{A}\|_{\mathrm{F}} \sim 5n\mathtt{eps}^{-1}.$$

The following is executable Matlab-code for Algorithm 3.10 (`InvIllCo`). The routines `TwoSum`, `VecSum`, `Split` and `TwoProduct` are listed in [7]. The code for `ProdKL` is straightforward transforming dot products into sums and applying `SumKL`. All routines are included in INTLAB [11], the Matlab toolbox for reliable computing.

Algorithm A.2. *Executable Matlab code for* Algorithm 3.10 *(*`InvIllCo`*).*

```
function R = invillco(A)
% Inversion of extremely ill-conditioned matrices by Rump's algorithm.
% The result R is stored in matrices R_1,...,R_k
%
  n = size(A,1);
  R = eye(n)/norm(A,'fro');
  X = inf; P = inf; k = 0;
  while 1
    k = k+1;
    finished = ( norm(X,'fro')*norm(X,'fro')<.01/eps );
    P = ProdKL(R,A,k,1);
    X = inv(P);
    while any(any(isinf(X)))
      disp('perturbation of P')
      X = inv(P.*(1+eps*randn(n)));
    end
    R = ProdKL(X,R,k,k);
    if finished, break, end
  end
%
function res = SumKL(p,K,L)
% Sum(p_i) in approximately K-fold precision stored in L elements
%
% Adaptation of SumK in
%   T. Ogita, S.M. Rump, S. Oishi: Accurate Sum and Dot Product,
%      SIAM Journal on Scientific Computing (SISC), 26(6):1955-1988, 2005
% to L outputs.
%
  n = length(p); res = zeros(1,L);
  for i=1:K-L
    p = VecSum(p);
  end
  for k=0:L-2
    p = VecSum(p(1:n-k));
    res(k+1) = p(n-k);
  end
  res(L) = sum(p(1:n-L+1));
```

Acknowledgement. The author wishes to thank Shin'ichi Oishi, Tetsuo Nishi, Takeshi Ogita and Kunio Tanabe from Waseda University for their interesting comments. Moreover my dearest thanks to the anonymous referees, who provided very valuable suggestions and remarks.

References

[1] T. Dekker, A floating-point technique for extending the available precision. Numerische Mathematik, **18** (1971), 224–242.

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

[3] ANSI/IEEE 754-2008: IEEE Standard for Floating-Point Arithmetic. New York, 2008.

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

[5] MATLAB User's Guide, Version 7. The MathWorks Inc., 2004.

[6] T. Nishi, T. Ogita, S. Oishi and S. Rump, A method for the generation of a class of ill-conditioned matrices. 2008 International Symposium on Nonlinear Theory and Its Applications, NOLTA '08, Budapest, Hungary, September 7–10, 2008, 53–56.

[7] T. Ogita, S. Rump and S. Oishi, Accurate sum and dot product. SIAM Journal on Scientific Computing (SISC), **26** (2005), 1955–1988.

[8] T. Ohta, T. Ogita, S. Rump and S. Oishi, Numerical verification method for arbitrarily ill-conditioned linear systems. Transactions on the Japan Society for Industrial and Applied Mathematics (Trans. JSIAM), **15** (2005), 269–287.

[9] S. Oishi, K. Tanabe, T. Ogita and S. Rump, Convergence of Rump's method for inverting arbitrarily ill-conditioned matrices. J. Comput. Appl. Math., **205** (2007), 533–544.

[10] S. Rump, A class of arbitrarily ill-conditioned floating-point matrices. SIAM J. Matrix Anal. Appl. (SIMAX), **12** (1991), 645–653.

[11] S. Rump, INTLAB—INTerval LABoratory. Developments in Reliable Computing, T. Csendes (ed.), Kluwer Academic Publishers, Dordrecht, 1999, 77–104.

[12] S. Rump, T. Ogita and S. Oishi, Accurate floating-point summation, part II: Sign, K-fold faithful and rounding to nearest. Accepted for publication in SISC, 2005–2008.

[13] S. Rump, T. Ogita and S. Oishi, Accurate floating-point summation, part I: Faithful rounding. SIAM J. Sci. Comput., **31** (2008), 189–224.

[14] N. Yamanaka, T. Ogita, S. Rump and S. Oishi, A parallel algorithm of accurate dot product. Accepted for publication, 2007.