On the failure of rank revealing QR factorization software – a case study LAPACK Working Note 176

Zlatko Drmač[*]and Zvonimir Bujanović[†]

Abstract

This note reports an unexpected and rather erratic behavior of the LAPACK software implementation of the QR factorization with Businger–Golub column pivoting. It is shown that, due to finite precision arithmetic, software implementation of the factorization can catastrophically fail to produce triangular factor with the structure characteristic to the Businger–Golub pivot strategy. The failure of current *state of the art* software, and a proposed alternative implementations are analyzed in detail.

1 Introduction

During the implementation and testing of a new Jacobi–type SVD algorithm [8, 9], we encountered an exceptional behavior in one test case: safety switches were triggered and an emergency branch of the code was activated. This worst case scenario was unexpected because the theory had guaranteed smooth run with no need for exceptional treatment of the input matrix. An inspection of control parameters computed by numerical *poka–yoke* devices in our software has shown that the exceptional behavior was caused by an objectionable result of the pivoted QR factorization in the preprocessing phase of the algorithm. Namely, the computed triangular factor failed to have properly ordered diagonal entries. This fact prompted separate testing of the LAPACK routine xGEQP3, which is used in our SVD software. It implements the Businger–Golub [2] pivot strategy which, for $A \in \mathbb{R}^{m\times n}$, computes permutation matrix P, orthonormal Q and upper triangular matrix R such that

$$AP = QR, \quad \text{where } |R_{ii}| \geq \sqrt{\sum_{k=i}^{j} |R_{kj}|^2}, \quad \text{for all } 1 \leq i \leq j \leq n. \tag{1}$$

The issue addressed in this note is relevant to the rank revealing problem, but it is not about the rank revealing capabilities of the factorization (1). Instead, the main object of our study is software implementation of (1).

[*]Department of Mathematics, University of Zagreb, Bijenička 30, 10000 Zagreb, Croatia.
[†]Department of Mathematics, University of Zagreb, Bijenička 30, 10000 Zagreb, Croatia.

It is well known that the QR factorization in floating point arithmetic is backward stable. Therefore, in proper software implementation, the computed $\tilde{Q}$, $\tilde{R}$ and the actually used permutation matrix $\tilde{P}$ should satisfy $(A+\Delta A)\tilde{P} = \tilde{Q}\tilde{R}$ with small ΔA, and, in addition, $\tilde{Q}$ should be numerically orthonormal and $\tilde{R}$ upper triangular. Strictly speaking, $\tilde{Q}$ is close to an orthonormal matrix $\hat{Q}$ such that $(A+\delta A)\tilde{P} = \hat{Q}\tilde{R}$ is a QR factorization with column pivoting. Here δA is similar in size to ΔA. Notice that in the statement of the backward (or mixed) stability we insist on structure: $\tilde{R}$ must be upper triangular and $\tilde{Q}$ must be numerically orthonormal.

But, although the factorization is computed with *pivoting*, which should impose certain *structure* on the triangular factor (cf. the property of R in (1)), the structure of $\tilde{R}$ is never mentioned in backward error analysis. It certainly cannot be taken as granted, as e.g. the triangular form of $\tilde{R}$. It could be that it is tacitly assumed that the structure will be nearly attained (up to roundoff), or that the issue is simply pushed into the forward error – the structure of the computed $\tilde{R}$ is considered not to be the responsibility of backward error analysis.

The purpose of this note is to warn that in currently available implementations (e.g. LAPACK, MATLAB) the code can catastrophically fail to produce such structured triangular factor. Thus, strictly speaking, such implementation is not backward (nor mixed) stable computation of the Businger–Golub QR factorization (1). Since the QR factorization is well understood and numerically stable, without inherent difficulties (such as in e.g. symmetric definite and indefinite factorizations or e.g. in the QR factorization in indefinite inner product spaces), a failure of its implementation with column pivoting of paramount importance is unacceptable and we propose an alternative.

The material is organized as follows. In §2 we give several examples which illustrate how state of the art implementations of the factorization (1) can fail to produce satisfactory results, and how this failure affects solvers based on the pivoted QR factorization. This examples should convince the reader that the problem is serious, and that it may lead to numerical catastrophes in engineering applications. Section 3 offers an analysis of the erratic behavior and proposes modifications of the current LAPACK code. The first modification is a quick fix for the current LAPACK code and it does not affect the input/output specifications of the routines. The second approach we propose requires n extra locations in the work space. In §4, we present new software implementations of the Businger–Golub pivoting. The new software runs at the speed similar to current LAPACK code, and it is fail safe. Finally, section 5 recalls the importance of pivoting from the numerical point of view. We discuss forward error in the computed factorization, and changes of certain condition numbers.

2 Examples of software failure

To make our case, we first give several examples of software failure. The matrix which first exposed the weakness of the code was the famous Kahan [12] matrix $\mathfrak{K} = \mathfrak{K}(n, c)$, but the nature of the weakness was unexpected and, to our best knowledge, not reported elsewhere. Recall, $\mathfrak{K}$

e.g. in the case $n = 6$ reads

$$\mathfrak{K} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & s & 0 & 0 & 0 & 0 \\ 0 & 0 & s^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & s^3 & 0 & 0 \\ 0 & 0 & 0 & 0 & s^4 & 0 \\ 0 & 0 & 0 & 0 & 0 & s^5 \end{pmatrix} \begin{pmatrix} 1 & -c & -c & -c & -c & -c \\ 0 & 1 & -c & -c & -c & -c \\ 0 & 0 & 1 & -c & -c & -c \\ 0 & 0 & 0 & 1 & -c & -c \\ 0 & 0 & 0 & 0 & 1 & -c \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad c^2 + s^2 = 1,$$

and in general,

$$\mathfrak{K}(n, c) = \left(\begin{array}{c|c} 1 & -c \; -c \; \ldots \; -c \\ \hline 0 & s\mathfrak{K}(n-1, c) \end{array} \right), \quad c = \cos\psi, \;\; s = \sin\psi.$$

This matrix is known to be a counterexample for the rank revealing property of the factorization (1) because it is already upper triangular with the property of R from (1), and $|\mathfrak{K}_{nn}|$ overestimates $\sigma_{\min}(\mathfrak{K})$ by a factor of order 2^{n-1}. The QR factorization (1) applied to $A = \mathfrak{K}$ gives $Q = I_n$, $P = I_n$, $R = \mathfrak{K}$. (In fact, even the Powell–Reid complete pivoting [13] leaves $\mathfrak{K}$ unchanged. Moreover, $R = \mathfrak{K}$ is the matrix which almost attains the $O(2^n)$ upper bound for $\|R_r^{-1}\|_2$ in Proposition 5.1.)

It is also well known that rounding errors during the computation may provoke permutation different from the identity, and the computed $\tilde{R} \neq \mathfrak{K}$ is rank–revealing in the sense that $|\tilde{R}_{nn}|$ correctly estimates the magnitude of the minimal singular value $\sigma_{\min}(\mathfrak{K})$ of $\mathfrak{K}$. The phenomenon we are going to describe is of different kind.

Our first examples were generated using the LAPACK xGEQP3 and xGEQPF procedures under a GNU FORTRAN compiler. (Both single and double precision routines failed, for both real and complex matrices.) Later on, we have tested other software packages (MATLAB, SciLab, Octave, Intel Fortran compiler) and found the same problem.

For this presentation we will use examples generated in MATLAB 6.5.

2.1 Loss of structure in the triangular factor

To control the structure of R, we compare $|R_{ii}|$ with

$$\mu_i = \max_{j=i+1:n} |R_{ij}|, \quad i = 1 : n-1.$$

In exact computation, (1) implies that $\max_{i=1:n-1} \mu_i / |R_{ii}| \leq 1$. (Note that (1) implies even stronger structure.)

Example 2.1 Take $n = 300$ and $c = 4.664999999999993e-001$, define $\mathfrak{A} = \mathfrak{K}(n, c)$ and compute $[Q, R, P] = qr(\mathfrak{A})$. If one routinely looks after the sudden drop along the diagonal of R, checking the quotients $|R_{k+1,k+1}/R_{kk}|$ will point to the index 281 where $|R_{281,281}/R_{280,280}| <$

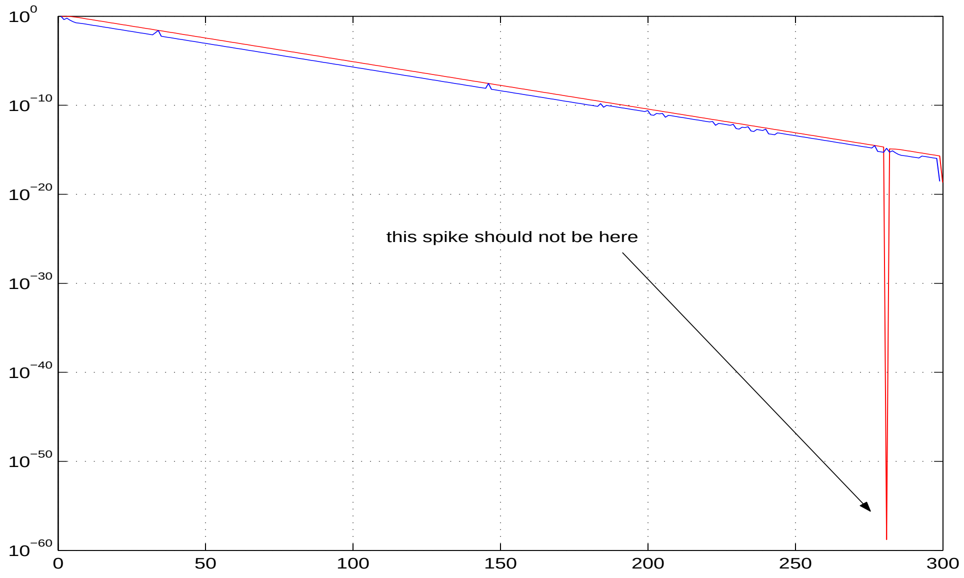

Figure 1: *The values of* $|R_{ii}|$ *(red line) and* μ_i *(blue line) for the matrix* $\mathfrak{K}(300, c)$ *in Example 2.1. Here* $\max_{i=1:n-1} \mu_i/|R_{ii}| \approx 9.2734e + 043$ *(and it should be at most* 1*).*

10^{-44}. Having the property (1) of R in mind, one concludes that $\|R(281 : 300, 281 : 300)\|_F \leq \sqrt{20} \cdot 10^{-44} \cdot |R_{280,280}|$ and that in the partition

$$R = \left(\begin{array}{ccc|ccc} R_{11} & \cdots & R_{1,280} & R_{1,281} & \cdots & R_{1,300} \\ & \ddots & \vdots & \vdots & & \vdots \\ & & R_{280,280} & R_{280,281} & \cdots & R_{280,300} \\ \hline & & & R_{281,281} & \cdots & R_{281,300} \\ & & & & \ddots & \vdots \\ & & & & & R_{300,300} \end{array}\right)$$

the sub–matrix below the line (rows with indices above 280) can be discarded. Visual inspection (Figure 1) shows that we are misled into a wrong conclusion. The same (wrong) conclusion is reached if an incremental condition estimator is deployed with the task to find maximal leading well–conditioned matrix.

If the initial $\mathfrak{A}$ is changed by random permutation of its columns, some permutations will produce satisfactory triangular factor, but some (very quickly found by random search) will lead to a catastrophic loss of diagonal dominance, but always at the positions around 281 (280, 281, 282). If the rows and the columns are permuted simultaneously, catastrophic loss of diagonal dominance is less frequent, but the loss of the non–increasing order of the diagonal entries is found very quickly, see Figures 2, 3.

To show the subtlety of the problem, take $\tilde{c} = c * (1 + eps) = 4.664999999999994e - 001$ (MATLAB notation), and then $\tilde{\mathfrak{A}} = \mathfrak{K}(n, \tilde{c})$. After computing $[\tilde{Q}, \tilde{R}, \tilde{P}] = qr(\tilde{\mathfrak{A}})$, one can easily check that $\tilde{R}$ is diagonally dominant with decreasing absolute values along the diagonal.[1] A

[1]The results of these experiments depend on the way we generate $\mathfrak{K}$ because one bit of difference can change the result. With his/her own code for $\mathfrak{K}$ the reader can find other interesting values of c.

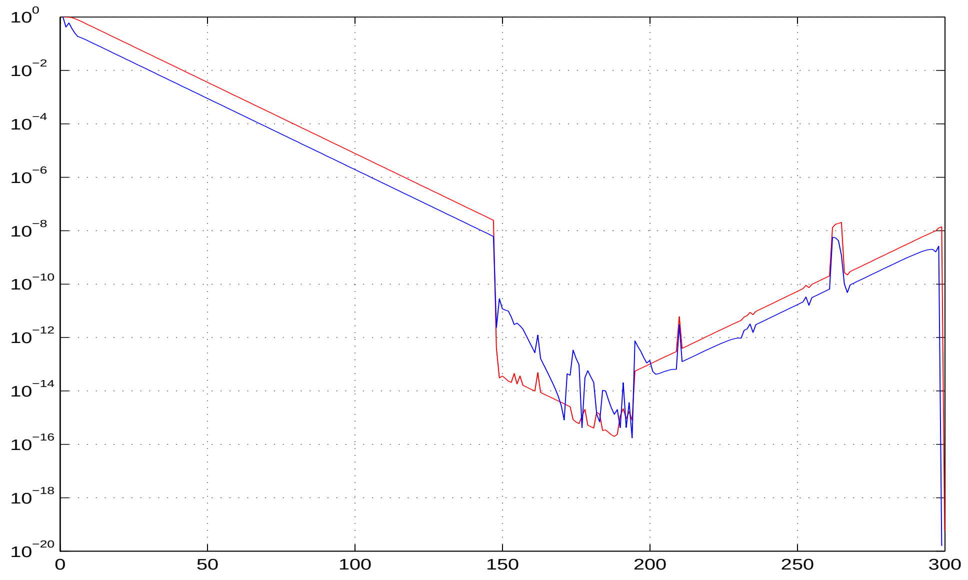

Figure 2: *The values of* $|R_{ii}|$ *(red line) and* μ_i *(blue line) for a row and column permuted matrix* $\mathfrak{K}(300, c)$ *in Example 2.1.*

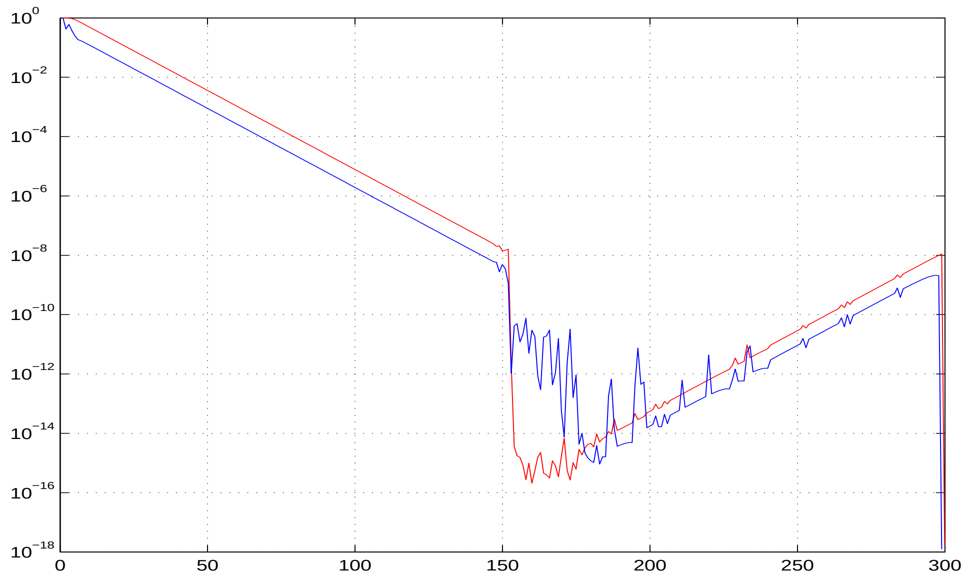

Figure 3: *The values of* $|R_{ii}|$ *(red line) and* μ_i *(blue line) for the row and column permuted matrix* $\mathfrak{K}(300, c)$ *in Example 2.1. Here* $\max_{i=1:n-1} \mu_i/|R_{ii}| \approx 2.7330e + 005$.

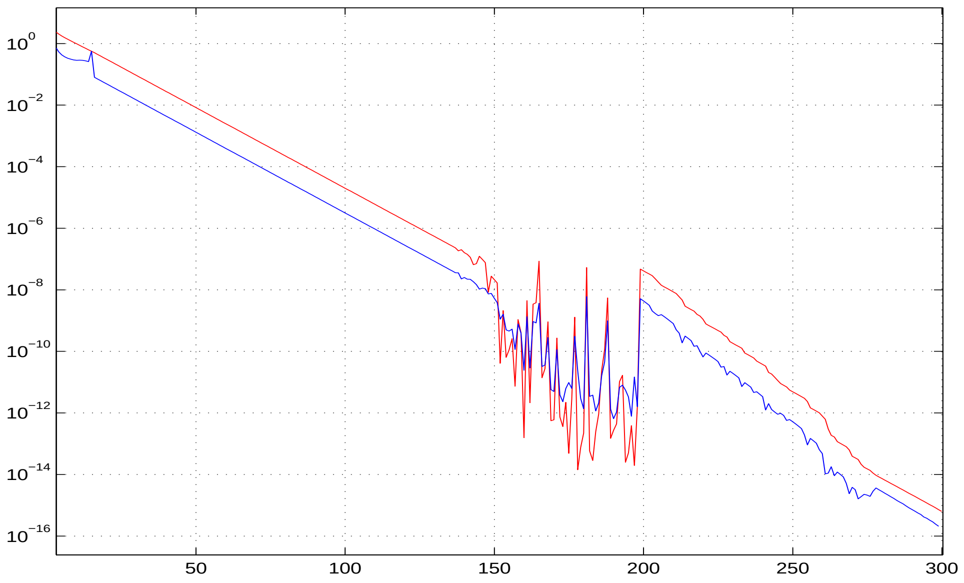

Figure 4: *The values of* $|R_{ii}|$ *(red line) and* μ_i *(blue line),* $i = 1 : 300$ *for the matrix* $\mathfrak{M} = \mathfrak{B} + \mathfrak{B}^T$, $\mathfrak{B} = \mathfrak{K}(300, 0.4630)$. *Here* $\max_{i=1:n-1} \mu_i/|R_{ii}| \approx 1.6633e + 003$.

comparison of the diagonal entries of the computed factors R and $\tilde{R}$ is given in (2).

i	$\lvert R_{ii}\rvert$	$\lvert\tilde{R}_{ii}\rvert$
$\vdots$	$\vdots$	
274	$4.146036291985283e-015$	$4.146036291985283e-015$
275	$3.667256988614787e-015$	$3.667256988614787e-015$
276	$3.243766545541791e-015$	$3.243766545541791e-015$
277	$2.869180271424214e-015$	$2.869180271424215e-015$
278	$2.537850771426260e-015$	$2.537850771426261e-015$
279	$2.244782805101268e-015$	$2.244782805101268e-015$
280	$1.985557976384244e-015$	$1.985557976384244e-015$
281	$\boxed{1.510608517753438\text{e-}059}$	$1.756268120293820e-015$
282	$1.191304454419907e-015$	$1.553456382058059e-015$
283	$1.173764088030493e-015$	$1.374065100352209e-015$
$\vdots$	$\vdots$	$\vdots$
298	$2.180875970060762e-016$	$2.180876095640635e-016$
299	$1.929031096752219e-016$	$1.929031137161460e-016$
300	$2.021325892754739e-019$	$3.350097232677502e-066$

(2)

Example 2.2 Almost identical situation is obtained with $\mathfrak{B} = \mathfrak{K}(300, 0.4630)$. Now, $\mathfrak{M} = \mathfrak{B} + \mathfrak{B}^T$ shows more irregular behavior of the diagonal of the computed triangular factor (Figure 4). One can easily construct more interesting examples. For instance, take $\mathfrak{N} = \begin{pmatrix} \mathfrak{A} & X \\ Y & \mathfrak{M} \end{pmatrix}$ with various X and Y. With $X = Y = 0$ we obtain $\max_{i=1:n-1} \mu_i/|R_{ii}| \approx 4.1037e + 009$. (The structure of the matrix R is shown in Figure 6.)

2.2 Consequences in applications

The QR factorization with column pivoting is computational routine used as core procedure in other solvers in numerical linear algebra, and its failure can have bad impact to their reliability. We give few examples and illustrate how the improper structure of R can fool more complex solvers.

Example 2.3 As we mentioned in the Introduction, we first experienced this problem in context of the new Jacobi type SVD algorithm [8, 9]. To illustrate, we first describe a simplified variant based on the accelerated Jacobi algorithm [18].

Let $A \in \mathbb{R}^{m\times n}$ have full column rank and $AP = Q\begin{pmatrix} R \\ 0 \end{pmatrix}$ be its QR factorization with pivoting as in (1). Instead of implicit diagonalization of $\mathsf{M} \equiv R^T R = P^T(A^T A)P$, accelerated Jacobi diagonalizes $\mathsf{W} \equiv RR^T$. This is easier task to accomplish because W tends to be very strongly diagonally dominant for any full rank A, and suitable implementation of Jacobi iterations can exploit such structure. The relevant condition number (for accuracy and convergence) is $\|R_r^{-1}\|_2$, which is expected to be moderate (See Proposition 5.1).

If the computed $\tilde{R} \approx R$ violates (1), as in examples in §2.1, then all inequalities "$\leq$" in Proposition 5.1 may become "$\gg$". The value of $\|\tilde{R}_r^{-1}\|_2$ may even overflow. This means that preconditioner completely fails – instead of reducing, it drastically increases the condition number. (For instance, computing the QR factorization of such a badly structured $\tilde{R}^T$ is not safe in the context of high accuracy SVD computation.) An occurrence of this situations triggers safety switches in our algorithm [8, 9].

Best illustration is to plot the matrix $\mathsf{W}_s = (\,\mathsf{W}_{ij}/\sqrt{\mathsf{W}_{ii}\mathsf{W}_{jj}}\,)$ using the `mesh()` command in MATLAB. In an ideal case, W_s should have clearly visible unit diagonal which dominates all off–diagonal entries. If the pivoting fails, we can have situation as in Figure 5. Instead being close to identity, $\mathsf{W}_s(150\!:\!200, 150\!:\!200)$ is highly ill–conditioned.

Example 2.4 Linear least squares problem solvers are also at high risk. Brief inspection of the source code of xGELSX and xGELSY in LAPACK is enough to conclude that failure of xGEQPF and xGEQP3 will go undetected, and that the least squares solution will have unnecessary and unacceptably large error. The reason is that the numerical rank is detected by a naive incremental condition estimator (ICE) which gets fooled by the first deep drop on the diagonal of the computed triangular factor.[2] We use the term *naive* ICE to denote an ICE which merely records the condition numbers of leading principal submatrices, without any attempt to recompute the triangular factor and find better submatrix in each particular step. Naive ICE relies on the assumed structure of the triangular factor.

For the sake of brevity, we will not list bad examples generated using LAPACK. Instead, we illustrate the nature of failure using one example generated in MATLAB. We use the *slash* operator to solve $\|Ax - d\|_2 \to \min$.

Let $A = \mathfrak{N}(:, 1 : 560)$, where $\mathfrak{N}$ is the matrix from Example 2.2. To solve $\|Ax - d\|_2 \to \min$ with randomly generated right hand side d, we use the *slash*, $A\backslash d$, which computes the solution using the QR factorization with column pivoting of the coefficient matrix A. The result is delivered with the warning

[2]See the source code http://www.netlib.org/lapack/single/sgelsy.f.

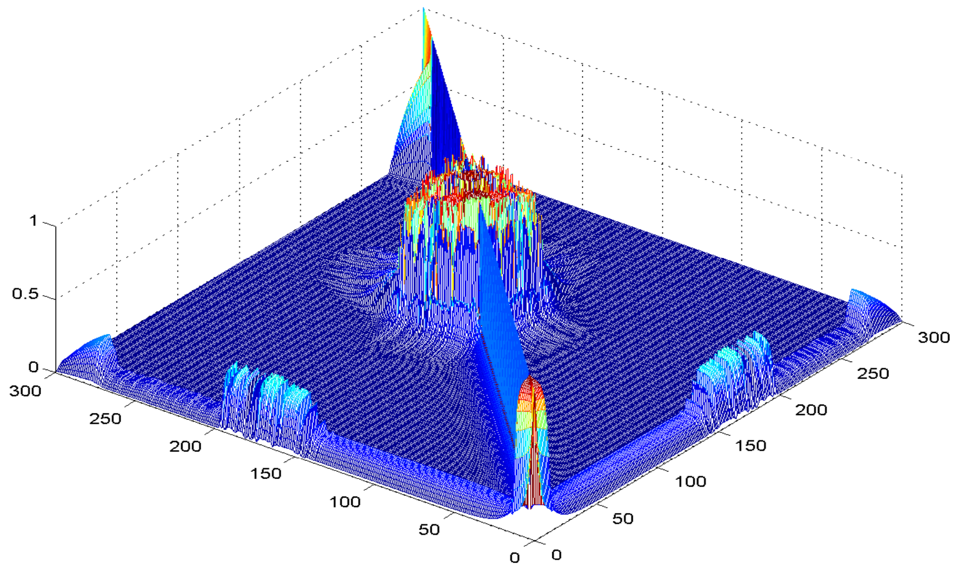

Figure 5: *The matrix* W_s, *where* $\tilde{R}$ *is the matrix used in Figure 4. The ill–conditioned "tower"* $\mathsf{W}_s(150\!:\!200, 150\!:\!200)$ *is a result of wrong pivot choices.*

```
Warning:  Rank deficient, rank = 304 tol = 1.0994e-012.
```

Note that in this case `rank(`A`,1.0994e-12)` returns 466. (Since $\|A\|_2 < 20$, 466 can be taken as correct number of singular values above the threshold.) We first note that 304 severely underestimates the numerical rank 466, as defined by the singular value threshold. (Expected is the opposite, due to the Eckart–Young–Mirsky theorem.)

To understand what caused this warning, we compute `[Q,R,P] = qr(A)` and analyze the structure of R. See Figure 6. It is clear where 304 comes from – here $|R(305, 305)| \approx 9.7560e-013$ is the first diagonal absolute value below the threshold `1.0994e-012`. Then, R is assumed to have block partition

$$R = \begin{pmatrix} R_{[11]} & R_{[12]} \\ 0 & R_{[22]} \end{pmatrix}, \text{ where } R_{[11]} = R(1:304, 1:304), \quad \|R_{[22]}\|_F \leq 1.5610e-011.$$

Setting $R_{[22]}$ to zero implicitly defines a rank 304 matrix $A + \delta A$ with $\|\delta A\|_F \leq 1.5610e - 011$. If we compute the singular values of A, then the computed value $2.0470e - 008$ of $\sigma_{305}(A)$ is sufficiently accurate to conclude that the distance to the closest matrix of rank at most 304 is more than $2.0470e - 008$. On the other hand, if we just count the number of $|R(i,i)|$'s above `1.0994e-012` we obtain $\underline{\text{exactly 466}}$.

Example 2.5 To compute the GSVD of the pair $(A, B) \in \mathbb{R}^{m\times n} \times \mathbb{R}^{p\times n}$, it can be advantageous to reduce the pair to joint triangular form by orthogonal transformations:

$$U^T A Q \;=\; \begin{array}{c} \\ k \\ \ell \\ m-k-\ell \end{array} \begin{array}{c} \begin{array}{ccc} n-k-\ell & k & \ell \end{array} \\ \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \\ 0 & 0 & 0 \end{pmatrix} \end{array} \text{ for } m-k-l \geq 0,$$

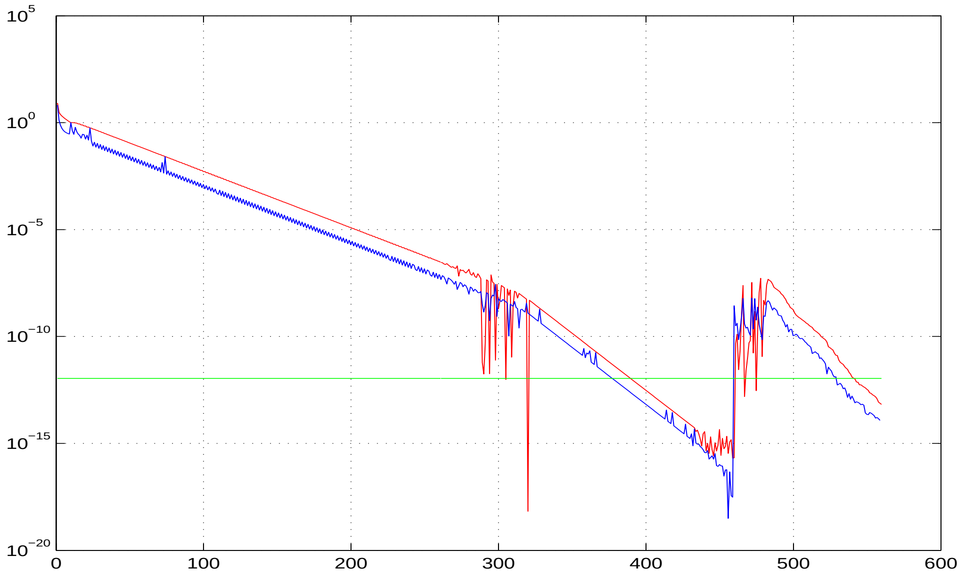

Figure 6: *The values of* $|R_{ii}|$ *(red line) and* μ_i *(blue line),* $i = 1 : 559$ *for the matrix* A *in Example 2.4. The green line marks the tolerance* `1.0994e-012` *used to determine numerical rank.*

$$
\text{or}\quad U^T A Q = \begin{array}{c} \\ k \\ m-k \end{array}\begin{array}{c} \begin{array}{ccc} n-k-\ell & k & \ell \end{array} \\ \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \end{pmatrix} \end{array}, \ \text{if}\ m-k-\ell < 0;
$$

$$
V^T B Q = \begin{array}{c} \\ \ell \\ p-\ell \end{array}\begin{array}{c} \begin{array}{ccc} n-k-\ell & k & \ell \end{array} \\ \begin{pmatrix} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{pmatrix} \end{array}.
$$

Here A_{12} and B_{13} are upper triangular nonsingular matrices. Note that reduction of B represents the URV decomposition which starts with the QR factorization with column pivoting, $B\tilde{P} \approx \tilde{Q}_B\tilde{R}$. The numerical rank $\tilde{\ell}$ of B (computed ℓ) is determined by counting the number of diagonal entries in the triangular factor $\tilde{R}$ which are above certain threshold. Such strategy may give correct value of the numerical rank, but the problem arises when $\tilde{R}$ is replaced with the first $\tilde{\ell}$ rows of $\tilde{R}$. (See SGGSVP.F in LAPACK.)

Similar problems arise in computation of the generalized QR factorization with pivoting, URV decomposition etc. We omit the details for the sake of brevity.

Example 2.6 More sophisticated rank revealing factorizations [3], [14], [1] postprocess the computed triangular factor. The initial triangular factor is computed by Businger–Golub pivoting restricted to a sliding window (for better use of memory hierarchy), see TOMS Algorithm 782. Then, fast condition estimators is used to detect and move suspicious columns to the rear and the triangular form is corrected by a sequence of Givens rotations. (This is a general scheme for sophisticated rank revealing QR factorizations – an initial factorization is refined in a process of optimization of suitably chosen objective functions.) Inspection of the code of TOMS ♯ 782 shows the same problem in the initial phase (see SGEQPC.F, SGEQPW.F, SGEQPB.F).

Numerical experiments with xGEQPX have shown the same sort of problems as with xGEQPF and xGEQP3 – as preprocessing module in our SVD algorithm it failed catastrophically.

3 Analysis of the problem

It is well known that $\mathfrak{K}$ is tough case for rank revealing QR factorization, so one is tempted to offer that fact as an explanation, without any need to look closer to identify the source of the problem. However, different manifestations of the problem in various computational tasks might have substantially different origins, and thus each of them definitely deserves separate analysis.

In the examples in §2 the pivoting was rank revealing, it has successfully exposed linear dependence by creating small columns in the submatrices generated in the algorithm, but the problem was that those small columns have, mistakenly, been taken for pivots in the ensuing steps.

3.1 Experimental analysis. Compiler issues.

To analyze, explain and correct the erratic behavior of the code, we need to go into the details of a concrete implementation. We first focus to SGEQPF, because: *(i)* the code is simpler than its BLAS 3 implementation SGEQP3; *(ii)* it has less numerical uncertainties than SGEQP3. (The same problem occurs in SQRDC from LINPACK. It should be stressed that xGEQP3 and xGEQPF are not numerically equivalent.) Our machine is Intel Pentium 4 based HP X2100 Workstation running under MS Windows/Suse Linux.[3]

To remove the unknown factor of machine optimized libraries we use BLAS and LAPACK compiled from the source code from the NETLIB repository. (Our first bad cases were discovered while using BLAS from the Intel's MKL library.)

First approach is experimental. Several experiments under different circumstances will provide useful hints for the ensuing numerical analysis. So far, it is clear that the problem is in wrongly determined pivotal columns. For the reader's convenience, we display in Table 1 part of SGEQPF.F which is critical for column norms of submatrices in the factorization process. (The updating strategy is the same as in the LINPACK procedure SQRDC.) What follows is a brief description of our course of action after facing this problem.
First attempt to resolve the problem is the obvious one – enforce explicit norm computation by SNRM2() in all cases. With this modification, the problem is gone. That is, of course, not satisfactory – we still do not know the exact source of the problem that we have removed so easily by using an expensive modification, which is not even feasible in the block (BLAS 3) implementation xGEQP3. However, this points to the main suspect: the IF statement which chooses between the updating formula and explicit norm computation.

We return the code to its original version in order to study the switching mechanism which chooses between update formula and explicit norm computation. An old fashion debugging practice calls for writing out the values of the key variable TEMP2. The outcome was one of the most feared – the run was smooth and the computed R had proper structure. The

[3] Same problems are discovered using Athlon and Pentium Xeon processors.

```
      DO 30 J = I+1, N
         IF ( WORK( J ).NE.ZERO ) THEN
            TEMP = ONE - ( ABS( A( I, J ) ) / WORK( J ) )**2
            TEMP = MAX( TEMP, ZERO )
            TEMP2 = ONE + 0.05*TEMP*( WORK( J ) / WORK( N+J ) )**2
            IF( TEMP2.EQ.ONE ) THEN
               IF( M-I.GT.0 ) THEN
                  WORK( J ) = SNRM2( M-I, A( I+1, J ), 1 )
                  WORK( N+J ) = WORK( J )
               ELSE
                  WORK( J ) = ZERO
                  WORK( N+J ) = ZERO
               END IF
            ELSE
               WORK( J ) = WORK( J )*SQRT( TEMP )
            END IF
         END IF
30    CONTINUE
```

Table 1: Critical part in the column norm update. (For the full source see http://www.netlib.org/lapack/single/sgeqpf.f)

result was as it should be. A bug?! In our code? (Our test code is very simple and, say, easily checked to be correct. It generates the matrix, calls SGEQPF and checks the structure of the computed upper triangular factor.) In LAPACK? (There is a W^3 page for LAPACK bugz, http://icl.cs.utk.edu/lapack-forum/bugz/) In MATLAB? (MATLAB uses LAPACK as computing engine.) Or in the compiler? (We have found 1999. reports on a bug in g77-2.95 19990629 (prerelease). According to Mathias Fröhlich at the Universität Tübingen, SLAPMT (which applies permutation returned by SGEQPF) was trapped in an infinite loop, and he was able to trace the error back to misscompiled SGEQPF.[4] But, we have encountered the same problem with the Intel Fortran compiler.)

Since finding a bug in program which returns correct result is difficult, we removed the WRITE statements and restored erratic behavior. Just to produce a different executable, we recompiled SGEQPF with the optimizer switched off. Needless to say, the result (data from our collection of bad matrices) was as it should be, with no anomalies. Finally a familiar situation, well known in the computing community.

We focus to a possibility that the GNU FORTRAN optimizer keeps the variable TEMP2 in long register (80 bit, 64 bit mantissa) which can change some equivalence relations we are used to take as granted. Note that the test IF (TEMP2 .EQ. ONE) was meant to be an equivalent way of asking

$$\text{IF (0.05*TEMP*(WORK(J) / WORK(N+J))**2 .LT. EPS)}, \tag{3}$$

where EPS=SLAMCH('Epsilon') is the working precision. It is known that this is a bad idea if TEMP2 is kept in a 80 bit register, which is precisely what happens in this case. The two IF's are not equivalent.

To prevent the compiler to use TEMP2 in extended precision, the code is compiled with the `-ffloat-store` option added to the `-O`. A quick look at the assembler code shows the effect of this option: TEMP2 is stored from the register to the memory and reloaded. The same happens without `ffloat-store` if WRITE(*,*) TEMP2 is inserted, because writing TEMP2

[4]We believe that this is pure coincidence that (if) this (is) compiler bug (that it) was found while using SGEQPF. However, believing in coincidences does not mean trusting them.

requires popping it from the stack. Our FORTRAN code compiled with `-O -ffloat-store` has successfully factorized all our bad examples.

IF(TEMP2 .EQ. ONE) THEN ... $\rightsquigarrow$

g77 -O -S	g77 -O -ffloat-store -S
	fstps -24(%ebp)
	flds -24(%ebp)
flds -104(%ebp)	flds -116(%ebp)
fxch %st(1)	fxch %st(1)
fucompp	fucompp
fnstsw %ax	fnstsw %ax
sahf	sahf
jne L41	jne L41
jp L41	jp L41

Table 2: *Fragments of the assembler code that correspond to the key IF statement.*

On the other hand, extra precision (extra 11 bits to the 53 bit mantissa of double precision) is priceless in floating point arithmetic and switching it off to have numerical program running correctly (and probably slower) is simply wrong. Extra steps would be required from processor to prevent it from using extra precision in numerical software!

Careless, aggressive optimizer can destroy numerical accuracy, but is switching it off because of this story with TEMP2 reasonable? Suppose we had an optimizer which is (almost) perfect from the numerical point of view. Would we expect it to keep storing and reloading TEMP2, and would that be the best way around this problem?

One possible way out of this situation is to replace the test IF (TEMP2 .EQ. ONE) with the one that explicitly uses the machine epsilon (3). For xGEQP3, one has to go to xLAQPS.F and xLAQP2.F to find the critical IF statements and make this change.[5] As the result of this change, the code compiled with `-O` runs fine in all our previous cases.

Unfortunately, this is not end of this story. Now, if we turn the optimizer off, or if we have the optimizer on, but with `-O -ffloat-store`, the problems reappear. So, in this case the optimizer works in favor of the accuracy. Again, assembler code reveals that the key point is keeping TEMP in long register. But, it is simply a matter of time and little (bad) luck to find bad cases for the `-O` option. Should we then try some other constellations of compiler options and hope not to see any bad result? (In fact, using different compiler options for routines called by xGEQP3 may give such a constellation for which new bad examples have to be constructed.) What if changing the rounding mode results in substantially different results? What if the problem is somewhere else, and everything we have observed thus far are just coincidences?

The problem and the solution can be found only by numerical analysis.

[5]Note that this changed code compiled with `-O` and the old code compiled with `-O -ffloat-store` are not necessarily numerically equivalent.

3.2 Updating strategy – numerical analysis

Consider the elimination step in the k–th column. Let $A^{(1)} = A$ and let

$$A^{(k)}\Pi_k = \begin{pmatrix} \cdot & \cdot & \cdot & \odot & \cdot & \oplus & \cdot \\ & \cdot & \cdot & \odot & \cdot & \oplus & \cdot \\ & & \cdot & \odot & \cdot & \oplus & \cdot \\ & & & \blacksquare & \cdot & \circledast & \cdot \\ & & & \circledcirc & \cdot & * & \cdot \\ & & & \circledcirc & \cdot & * & \cdot \\ & & & \circledcirc & \cdot & * & \cdot \\ & & & \circledcirc & \cdot & * & \cdot \end{pmatrix}, \quad \mathbf{a}_j^{(k)} = \begin{pmatrix} \oplus \\ \oplus \\ \oplus \\ \circledast \\ * \\ * \\ * \\ * \end{pmatrix} \equiv \begin{pmatrix} \mathbf{x}_j^{(k)} \\ \eta_j^{(k)} \\ \mathbf{y}_j^{(k)} \end{pmatrix}, \quad \begin{array}{l} \eta_j^{(k)} = \circledast \equiv (A^{(k)})_{kj}, \\ \mathbf{z}_j^{(k)} = \begin{pmatrix} \eta_j^{(k)} \\ \mathbf{y}_j^{(k)} \end{pmatrix}. \end{array} \qquad (4)$$

Elements to be annihilated are denoted by $\circledcirc$, and $\blacksquare$ denotes the element R_{kk}, computed in the k–th step, after the $\circledcirc$'s have been eliminated.

Let $\omega_j^{(k)} = \|\mathbf{z}_j^{(k)}\|_2$. Permutation Π_k ensures that $|R_{kk}| \geq \omega_j^{(k)}$ for all $j \geq k$. Let H_k be Householder reflector such that

$$\mathrm{H}_k \begin{pmatrix} \eta_k^{(k)} \\ \mathbf{y}_k^{(k)} \end{pmatrix} = \begin{pmatrix} R_{kk} \\ 0 \end{pmatrix}, \text{ and let, for } j > k, \begin{pmatrix} \beta_j^{(k+1)} \\ \mathbf{z}_j^{(k+1)} \end{pmatrix} = \mathrm{H}_k \mathbf{z}_j^{(k)}. \qquad (5)$$

The goal is to compute $\omega_j^{(k+1)} = \|\mathbf{z}_j^{(k+1)}\|_2$ by a simple scalar formula with guaranteed and controlled number of correct digits whenever numerically feasible. Clearly, possible loss of accuracy by catastrophic cancellation should be avoided by a failsafe safety switch, which should react in cases of considerable norm reduction.

The following proposition shows that sharp norm reduction is related to the condition number of A_c (cf. §5.2). In other words, cancellation indicates ill–conditioning.

Proposition 3.1 *For each j, k, with $\|\mathbf{z}_j^{(k+1)}\|_2 \neq 0$, $\|A_c^\dagger\|_2 \geq \|\mathbf{z}_j^{(k)}\|_2 / \|\mathbf{z}_j^{(k+1)}\|_2$. (If $\mathbf{z}_j^{(k)} = \mathbf{0}$ for some j, k, then A is singular.)*

Remark 3.1 Even if we decided to compute the values $\omega_j^{(k)}$ by explicit norm computations, we could not guarantee that the computed triangular factor $\tilde{R}$ would satisfy (1). Instead, the structure of $\tilde{R}$ would be (22) with parameters $\tilde{\rho}_i$ close to one. $\boxtimes$

For the purpose of error analysis, computed quantities will be denoted by tildas, e.g. $\tilde{\omega}_j^{(k)}$ is the computed floating point value of $\omega_j^{(k)}$. Basic operations $+$, $-$, $\cdot$, $\div$, $\sqrt{\centerdot}$ will be denoted by $\oplus$, $\ominus$, $\odot$, $\oslash$, $sqrt()$, respectively. The set of floating point numbers is denoted by $\mathfrak{F}$, and $\mathfrak{e}$ is the gap between one and its first neighbor in $\mathfrak{F}$.

3.2.1 LAPACK (LINPACK) updating strategy

Orthogonality of H_k implies that $\omega_j^{(k)} = \sqrt{(\beta_j^{(k+1)})^2 + \|\mathbf{z}_j^{(k+1)}\|_2^2}$, and thus

$$\omega_j^{(k+1)} = \sqrt{(\omega_j^{(k)})^2 - (\beta_j^{(k+1)})^2}. \qquad (6)$$

Since $\omega_j^{(1)} = \|\mathbf{a}_j\|_2$ (computed explicitly as vector norm), $\omega_j^{(k+1)}$ can be recursively computed from $\omega_j^{(k)}$ and $\beta_j^{(k+1)}$, using (6). This is the approach taken in LAPACK, see Table 1.

Note that $(\omega_j^{(k)})_{k\geq 1}$ is nonincreasing sequence, obtained by successive substractions. If at some step k the update (6) is not considered to be numerically safe, the corresponding value $\tilde{\omega}_j^{(k)}$ is computed explicitly as vector norm. In that case, the value of $\tilde{\omega}_j^{(k)}$ is also stored in the variable $\tilde{\nu}_j$, $\tilde{\nu}_j = \tilde{\omega}_j^{(k)}$. Thus, at any moment in the algorithm, $\tilde{\nu}_j$ contains the last explicitly computed partial column norm in the j–th column. Initially, $\tilde{\nu}_j$'s are the computed column norms of A.

The safety switch in LAPACK which allows using update by (6) has simple and elegant structure:

$$\mathit{computed}(\underbrace{\left(1-\left(\frac{\tilde{\beta}_j^{(k+1)}}{\tilde{\omega}_j^{(k)}}\right)^2\right)}_{\mathit{predicted}}\cdot\underbrace{\left(\frac{\tilde{\omega}_j^{(k)}}{\tilde{\nu}_j}\right)^2}_{\mathit{memorized}}) > \mathit{tol},\quad \mathit{tol}\approx 20\mathfrak{e}, \tag{7}$$

where *predicted* part estimates loss of accuracy in computing $\tilde{\omega}_j^{(k+1)}$ from $\tilde{\omega}_j^{(k)}$, and the *memorized* part memorizes the cumulative loss of accuracy (by cancellations) since the last update by explicit norm computation. The two factors together indicate how accurately $\tilde{\omega}_j^{(k+1)}$ approximates the corresponding partial column norm.

Remark 3.2 Let $\tilde{t}_j^{(k)} = \max\{1 \ominus (\tilde{\beta}_j^{(k+1)} \oslash \tilde{\omega}_j^{(k)})**2, 0\}$. The LAPACK test

$$\mathit{if}\ \ (\underbrace{1 \oplus 0.05 \odot \tilde{t}_j^{(k)} \odot (\tilde{\omega}_j^{(k)} \oslash \tilde{\nu}_j)**2}_{\text{TEMP2}}\ \ .eq.\ \ 1\ ) \tag{8}$$

probes whether or not $\tilde{\omega}_j^{(k+1)} = \mathit{sqrt}(\tilde{t}_j^{(k)}) \odot \tilde{\omega}_j^{(k)}$ (see (6)), sharply drops as compared to $\tilde{\nu}_j$. However, depending on the compiler, the optimizer, and given options, (8) implicitly tests

$$\mathit{if}\ \ (\ 0.05 \odot \tilde{t}_j^{(k)} \odot (\tilde{\omega}_j^{(k)} \oslash \tilde{\nu}_j)**2 < \ell\cdot\mathfrak{e}), \tag{9}$$

where $\ell \in (0,1]$ denotes the extra precision factor if long registers are used. So, for instance, if TEMP2 is kept in a long register, it can have the value of $1+\mathfrak{e}/2 \neq 1$. If the code is forced to spill TEMP2 back to working precision, the resulting value can be $1(=1)$ or $1+\mathfrak{e}(\neq 1)$, depending on implementation.
Following (8), if $\dfrac{\tilde{\omega}_j^{(k+1)}}{\tilde{\nu}_j}$ is below $\sqrt{20}\sqrt{\ell\mathfrak{e}}(1+O(\mathfrak{e}))$ the value of $\tilde{\omega}_j^{(k+1)}$ is obtained by explicit norm computation. Else, $\tilde{\omega}_j^{(k+1)} = \mathit{sqrt}(\tilde{t}_j^{(k)}) \odot \tilde{\omega}_j^{(k)}$. Here the use of long registers actually lowers the threshold and weakens the safety switch. ⊠

From now on, we assume that the test is explicit as in (9), with $\ell = 1$.

How to analyze the accuracy of the $\tilde{\omega}_j^{(k)}$'s? It makes little sense to compare them with the exact $\omega_j^{(k)}$'s. Instead, we have to attach their values to the norms of the actually computed vectors $\tilde{\mathbf{z}}_j^{(k)}$. Let $\tilde{\omega}_j^{(k)} = \|\tilde{\mathbf{z}}_j^{(k)}\|_2(1+\epsilon_j^{(k)})$. If $\tilde{\omega}_j^{(k)}$ is obtained by computing the norm of $\tilde{\mathbf{z}}_j^{(k)}$ explicitly, then $|\epsilon_j^{(k)}|$ is at most a small multiple of $\mathfrak{e}$. We need to know how $\epsilon_j^{(k)}$ propagates through repeated applications of the updating formula (6). To this end, consider floating point version of (5):

$$\begin{pmatrix} \tilde{\beta}_j^{(k+1)} \\ \tilde{\mathbf{z}}_j^{(k+1)} \end{pmatrix} = \hat{H}_k \hat{\mathbf{z}}_j^{(k)}, \text{ where } \hat{\mathbf{z}}_j^{(k)} = \tilde{\mathbf{z}}_j^{(k)} + \delta\tilde{\mathbf{z}}_j^{(k)} \text{ is backward perturbed } \tilde{\mathbf{z}}_j^{(k)}, \tag{10}$$

and $\hat{H}_k$ is exactly orthogonal, close to the actually used numerically orthogonal $\tilde{H}_k$. The backward perturbation $\delta\tilde{\mathbf{z}}_j^{(k)}$ is small, and $\|\hat{\mathbf{z}}_j^{(k)}\|_2 = \|\tilde{\mathbf{z}}_j^{(k)}\|_2(1+\lambda_j^{(k)})$, where $|\lambda_j^{(k)}|$ is bounded by a small multiple of the roundoff $\mathfrak{e}$, depending on the implementation details (e.g. simple or aggregated transformations). Note that in this situation the analog of (6) reads

$$\|\tilde{\mathbf{z}}_j^{(k+1)}\|_2 = \sqrt{\|\hat{\mathbf{z}}_j^{(k)}\|_2^2 - (\tilde{\beta}_j^{(k+1)})^2}. \tag{11}$$

Remark 3.3 The transformation (10) can produce $\tilde{\mathbf{z}}_j^{(k+1)}$ with $\dfrac{\|\tilde{\mathbf{z}}_j^{(k+1)}\|_2}{\tilde{\omega}_j^{(k+1)}}$ much smaller than $\mathfrak{e}$. Such a sharp drop may go undetected. Having null vector may go undetected, and it can happen that zero column $\tilde{z}_j^{(k+1)}$ can be taken as pivot because it appeared of larger norm than non–zero columns. We have encountered such catastrophic miss–pivoting.

Example 3.1 To illustrate how this strategy can fail, take (in MATLAB)

$$\begin{pmatrix} \tilde{\beta}_j^{(k+1)} \\ \tilde{\mathbf{z}}_j^{(k+1)} \end{pmatrix} = \begin{pmatrix} 1 \ominus 11 \odot \mathfrak{e} \\ \mathfrak{s} \\ \mathfrak{s} \\ \mathfrak{s} \\ \mathfrak{s} \end{pmatrix}, |\mathfrak{s}| \le \mathfrak{e}, \text{ and let } \tilde{\omega}_j^{(k)} = \tilde{\nu}_j = 1.$$

$\mathfrak{t} = \max\{1 \ominus (\tilde{\beta}_j^{(k+1)} \oslash \tilde{\omega}_j^{(k)}) {*}{*}2, 0\} = 4.8849813083506689\mathrm{e}{-}015$, and the value of the control parameter $0.05 \odot \mathfrak{t} \odot (\tilde{\omega}_j^{(k)} \oslash \tilde{\nu}_j){*}{*}2$ is $2.4424906541753 45e{-}016$, which is slightly larger than $\mathfrak{e}$. The computed value of $\tilde{\omega}_j^{(k+1)}$ is 6.989264130329236e−008, and the true norm of $\tilde{\mathbf{z}}_j^{(k+1)}$ is $2\mathfrak{e} \approx 4.44e{-}16$. To the pivoting device, $\tilde{\mathbf{z}}_j^{(k+1)}$ will appear as roughly 10^8 times bigger than it actually is. Further, in the next step, the value of $\mathfrak{t}$ will be computed as one, the safety check will allow updating formula, $\tilde{\omega}_j^{(k+2)} = \tilde{\omega}_j^{(k+1)}$. From this point on, the partial column norms in the j–th column will never again be refreshed by explicit norm computation. Note that e.g. in the case $\mathfrak{s} = 0$, even the zero vector can mistakenly be taken for pivot. (We have encountered such cases using SGEQPF from LAPACK.)

Proposition 3.2 *Let $\tilde{t}_j^{(k)} = \max\{1 \ominus (\tilde{\beta}_j^{(k+1)} \oslash \tilde{\omega}_j^{(k)}){*}{*}2, 0\}$. Then $\tilde{t}_j^{(k)} \in \{0\} \bigcup [\mathfrak{e}, 1] \bigcap \mathfrak{F}$. (Depending on the way the compiler and the optimizer use long registers, $\mathfrak{e}$ could be replaced by a smaller value.) As a consequence, if $\|\tilde{\mathbf{z}}_j^{(k+1)}\|_2$ is computed by $\tilde{\omega}_j^{(k+1)} = sqrt(\tilde{t}_j^{(k)}) \odot \tilde{\omega}_j^{(k)}$ (see (6)), the sharpest drop in the partial column norm that can be observed is of order $\sqrt{\mathfrak{e}}$, which is the order of magnitude of smallest nonzero value of $sqrt(\tilde{t}_j^{(k)})$.*

Proposition 3.3 *Let $\tilde{\omega}_j^{(k)} = \|\tilde{\mathbf{z}}_j^{(k)}\|_2(1+\epsilon_j^{(k)})$, and let $\tilde{\omega}_j^{(k+1)}$ be computed as in Proposition 3.2, with $\tilde{t}_j^{(k)} > 0$. If $\tilde{\mathbf{z}}_j^{(k+1)} \neq \mathbf{0}$, then $\tilde{\omega}_j^{(k+1)} = \|\tilde{\mathbf{z}}_j^{(k+1)}\|_2(1+\epsilon_j^{(k+1)})$ with*

$$\begin{aligned} 1+\epsilon_j^{(k+1)} &= (1+\epsilon_j^{(k)})(1+\alpha_j^{(k)})\sqrt{1-\left[\frac{(\tilde{\beta}_j^{(k+1)})^2}{\|\hat{\mathbf{z}}_j^{(k)}\|_2^2-(\tilde{\beta}_j^{(k+1)})^2}\right]\sigma_j^{(k)}}, \quad \textit{where} \\ 1+\alpha_j^{(k)} &= \frac{\sqrt{1+e_3}(1+e_4)(1+e_5)}{1+\lambda_j^{(k)}}, \quad \sigma_j^{(k)} = \frac{(1+\lambda_j^{(k)})^2}{(1+\epsilon_j^{(k)})^2}(1+e_1)^2(1+e_2)-1 \end{aligned}$$

and $\max_i |e_i| \leq \mathfrak{e}$. *If $\tilde{\mathbf{z}}_j^{(k+1)} = \mathbf{0}$ and $\tilde{t}_j^{(k)} > 0$, then $\tilde{\omega}_j^{(k+1)}$ will be computed as*

$$\tilde{\omega}_j^{(k+1)} = \|\mathbf{z}_j^{(k)}\|_2 \frac{1+\epsilon_j^{(k)}}{1+\lambda_j^{(k)}}\sqrt{1-\frac{(1+\lambda_j^{(k)})^2}{(1+\epsilon_j^{(k)})^2}(1+e_1)^2(1+e_2)}\sqrt{1+e_3}(1+e_4)(1+e_5).$$

This is the lowest nonzero value that can be computed in this update.

We see that the critical *condition number* for this update is the value

$$\hat{\kappa}_j^{(k)} = \frac{(\tilde{\beta}_j^{(k+1)})^2}{\|\hat{\mathbf{z}}_j^{(k)}\|_2^2-(\tilde{\beta}_j^{(k+1)})^2} = \frac{\frac{(\tilde{\beta}_j^{(k+1)})^2}{\|\hat{\mathbf{z}}_j^{(k)}\|_2^2}}{1-\frac{(\tilde{\beta}_j^{(k+1)})^2}{\|\hat{\mathbf{z}}_j^{(k)}\|_2^2}} \equiv \frac{1-\hat{t}_j^{(k)}}{\hat{t}_j^{(k)}}, \quad \hat{t}_j^{(k)} = 1-\frac{(\tilde{\beta}_j^{(k+1)})^2}{\|\hat{\mathbf{z}}_j^{(k)}\|_2^2}.$$

(Note that $\hat{\kappa}_j^{(k)} \leq 1$ for $\hat{t}_j^{(k)} \geq 1/2$.) Since $\hat{t}_j^{(k)}$ is not accessible, its role is taken by the computed $\tilde{t}_j^{(k)}$. It is easily checked that

$$\hat{t}_j^{(k)} = \frac{1}{1+\sigma_j^{(k)}}\left(\frac{\tilde{t}_j^{(k)}}{1+e_3}+\sigma_j^{(k)}\right). \tag{12}$$

From the numerical experiments we know that the occurrences of failure are not easily found and that slightest change of rounding errors decides between success and failure. It seems that rounding errors can conspire to bring down the updating strategy.

Example 3.2 Here is one realistic scenario: Let $\tilde{\omega}_j^{(k)} = \tilde{\nu}_j$ be computed by explicit norm computation, thus $|\epsilon_j^{(k)}| \leq O(n)\mathfrak{e}$. Then $|\sigma_j^{(k)}| \leq O(n)\mathfrak{e}$ as well; take for instance $\sigma_j^{(k)} \approx -30\mathfrak{e}$. Now assume that $\tilde{t}_j^{(k)} = -(1+e_3)\sigma_j^{(k)}(1-O(\mathfrak{e})) \approx 30\mathfrak{e}$, which is the value above the threshold and updating formula will be used. But, $\hat{t}_j^{(k)} = O(\mathfrak{e})\sigma_j^{(k)}/(1+\sigma_j^{(k)}) \approx O(\mathfrak{e}^2)$, and $|\epsilon_j^{(k+1)}|$ can be as big as $O(1/\sqrt{\mathfrak{e}})$. Note that the failure is caused by a severe underestimate of the actual condition number, and that $\sigma_j^{(k)}$ had to be negative to make this scenario possible. ⊠

This example indicates that the threshold tolerance for $\tilde{t}_j^{(k)}$ should be lifted to the level where we can guarantee satisfactory lower bound for $\hat{t}_j^{(k)}$. (We should keep in mind that condition number is also computed quantity, and it has its own condition number.) To unroll the recurrence from Proposition 3.3 after s consecutive updates is rather tedious. Note that

$$\begin{aligned}\sigma_j^{(k)} &= -2\epsilon_j^{(k)} + 2\lambda_j^{(k)} + O(\mathfrak{e}) + \cdots higher \cdots order \cdots terms \cdots \doteq -2\epsilon_j^{(k)} \\ \epsilon_j^{(k+1)} &= \frac{\epsilon_j^{(k)}}{\hat{t}_j^{(k)}} + \frac{\lambda_j^{(k)}}{\hat{t}_j^{(k)}} + \alpha_j^{(k)} + \cdots higher \cdots order \cdots terms \cdots \doteq \frac{\epsilon_j^{(k)}}{\hat{t}_j^{(k)}},\end{aligned}$$

where $\doteq$ indicates dependence on the dominant, potentially largest term (in modulus). (In a simplified model with $\lambda_j^{(k)}$ and all e_i's equal to zero, we have $\sigma_j^{(k)} = -2\epsilon_j^{(k)}$ and $\epsilon_j^{(k+1)} = \dfrac{\epsilon_j^{(k)}}{\hat{t}_j^{(k)}}$.)

Let us take in (7) $tol = \sqrt{\mathfrak{e}}$.

Then, starting with $\tilde{\omega}_j^{(k)} = \tilde{\nu}_j$, $|\epsilon_j^{(k)}| \leq O(n)\mathfrak{e}$, $\tilde{t}_j^{(k)} > \sqrt{\mathfrak{e}}(1+O(\mathfrak{e}))$ implies

$$|\sigma_j^{(k)}| \leq O(n)\mathfrak{e}, \quad \frac{|\sigma_j^{(k)}|}{\tilde{t}_j^{(k)}} \leq O(n)\sqrt{\mathfrak{e}},$$

and we see (using (12)) that $\hat{t}_j^{(k)} \gtrapprox \tilde{t}_j^{(k)}(1 - O(n)\sqrt{\mathfrak{e}}) \geq O(\sqrt{\mathfrak{e}})$, which in turn guarantees sufficiently small $\epsilon_j^{(k+1)} \doteq \epsilon_j^{(k)}/\hat{t}_j^{(k)}$. Thus, the first update after explicit norm computation is safe. (Cf. Example 3.2.)

Consider the next, second, update. Let $\tilde{t}_j^{(k+1)}(\tilde{\omega}_j^{(k+1)}/\tilde{\nu}_j)^2 > \sqrt{\mathfrak{e}}(1+O(\mathfrak{e}))$. Hence, $\tilde{t}_j^{(k+1)}\tilde{t}_j^{(k)} > \sqrt{\mathfrak{e}}(1+O(\mathfrak{e}))$. Since the dominant part in $\epsilon_j^{(k+2)}$ is $\epsilon_j^{(k)}/(\hat{t}_j^{(k)}\hat{t}_j^{(k+1)})$, the key question is how small can be the product $\hat{t}_j^{(k)}\hat{t}_j^{(k+1)}$, given the fact that $\tilde{t}_j^{(k)}\tilde{t}_j^{(k+1)} > \sqrt{\mathfrak{e}}(1+O(\mathfrak{e}))$. Note that $\tilde{t}_j^{(k+1)} > (\sqrt{\mathfrak{e}}/\tilde{t}_j^{(k)})(1+O(\mathfrak{e}))$, and that by (12)

$$\hat{t}_j^{(k+1)}\hat{t}_j^{(k)} = \frac{\tilde{t}_j^{(k+1)}\tilde{t}_j^{(k)}}{(1+\sigma_j^{(k+1)})(1+\sigma_j^{(k)})}\left[\frac{1}{1+f_3} + \frac{\sigma_j^{(k+1)}}{\tilde{t}_j^{(k+1)}}\right]\left[\frac{1}{1+e_3} + \frac{\sigma_j^{(k)}}{\tilde{t}_j^{(k)}}\right],$$

where $\sigma_j^{(k+1)} \doteq -2\epsilon_j^{(k+1)} \doteq -2\epsilon_j^{(k)}/\hat{t}_j^{(k)}$, and $|f_3| \leq \mathfrak{e}$. It follows that

$$\frac{\sigma_j^{(k+1)}}{\tilde{t}_j^{(k+1)}} \doteq -2\frac{\epsilon_j^{(k+1)}}{\tilde{t}_j^{(k+1)}} \doteq -2\frac{\epsilon_j^{(k)}}{\hat{t}_j^{(k)}\tilde{t}_j^{(k+1)}} \doteq -2\frac{\epsilon_j^{(k)}}{\tilde{t}_j^{(k)}\tilde{t}_j^{(k+1)}}(1+O(n)\sqrt{\mathfrak{e}})$$

and thus $\hat{t}_j^{(k+1)}\hat{t}_j^{(k)} \approx \tilde{t}_j^{(k+1)}\tilde{t}_j^{(k)}$. Hence, $\epsilon_j^{(k+2)} \doteq \epsilon_j^{(k)}/(\tilde{t}_j^{(k+1)}\tilde{t}_j^{(k)}) \approx O(n)\sqrt{\mathfrak{e}}$. Note how important it is that the upper bound on $|\sigma_j^{(k+1)}|$ does not reach the lower bound on $\tilde{t}_j^{(k+1)}$ in the case of negative $\sigma_j^{(k+1)}$.

In general case, $\tilde{t}_j^{(k+s)}(\tilde{\omega}_j^{(k+s)}/\tilde{\nu}_j)^2 > \sqrt{\mathfrak{e}}(1+O(\mathfrak{e}))$, that is $\prod_{i=0}^{s}\tilde{t}_k^{(k+i)} > \sqrt{\mathfrak{e}}(1+O(s)\mathfrak{e})$. From previous updates we have

$$\sigma_j^{(k+s)} \doteq -2\epsilon_j^{(k+s)} \doteq -2\frac{\epsilon_j^{(k)}}{\prod_{i=0}^{s-1}\hat{t}_j^{(k+i)}}, \quad \frac{\sigma_j^{(k+s)}}{\tilde{t}_j^{(k+s)}} \doteq -2\frac{\epsilon_j^{(k)}}{\tilde{t}_j^{(k+s)}\prod_{i=0}^{s-1}\hat{t}_j^{(k+i)}}.$$

```
$t = |\tilde{\beta}_j^{(k+1)}/\tilde{\omega}_j^{(k)}|$ ;  $t = \max\{0, (1-t)\cdot(1+t)\}$
$t_2 = t \cdot (\tilde{\omega}_j^{(k)}/\tilde{\nu}_j)^2$
if ( $t_2 \leq \sqrt{\mathfrak{e}}$ ) then
    push $\tilde{z}_j^{(k+1)}$ to stack of unresolved columns
else
    $\tilde{\omega}_j^{(k+1)} = \tilde{\omega}_j^{(k)}\sqrt{t}$
end if
```

Table 3: Modification of the LINPACK/LAPACK column norm update.

If $\prod_{i=0}^{s-1}\hat{t}_j^{(k+i)} \approx \prod_{i=0}^{s-1}\tilde{t}_j^{(k+i)}$ (or at least the two products are of the same order of magnitude), then by (12) $\hat{t}_j^{(k+s)} \approx \tilde{t}_j^{(k+s)}$, and $\tilde{\omega}_j^{(k+s+1)} = \sqrt{\tilde{t}_j^{(k+s)}} \odot \tilde{\omega}_j^{(k+s)}$ is sufficiently accurate.

In the previous considerations we have not included possible accumulation of errors, and have neglected some 'less dangerous' terms. However, it is shown that in the current LAPACK code the switch between explicit norm update and scalar updating formula is not safe and that it is a possible source of numerical catastrophes in engineering applications. It is also shown that the updating strategy with higher threshold value ($\sqrt{\mathfrak{e}}$) can survive several updates. In fact, such a modified updating strategy, shown in Table 3, has successfully passed all our tests, performed in single precision with $tol = \sqrt{\mathfrak{e}} \approx 2.44e-4$. (It has failed with slightly smaller threshold, $tol = 1.0e-4$.) In Appendix A. we give the critical part of the new routine, to show that it takes only a minor modification of the current LAPACK code to make it much more reliable. Numerical experiments in §4 show that the modified code is equally efficient.

3.2.2 An alternative formula

Since accurate partial column norms are crucial for the success of pivoting, it is desirable to have updating formula with controlled forward error for all partial column norms of the computed floating point matrices. The goal is to have provably safe implementation of the Businger–Golub column pivoting.

Set $\xi_j^{(k)} = \|\mathbf{x}_j^{(k)}\|_2$, and immediately note that $\xi_j^{(k+1)} = \sqrt{(\xi_j^{(k)})^2 + (\beta_j^{(k+1)})^2}$, and that

$$\omega_j^{(k)} = \sqrt{\|\mathbf{a}_j^{(k)}\|_2^2 - (\xi_j^{(k)})^2}. \tag{13}$$

Proposition 3.4 *The formula (13) can be used to compute $\tilde{\omega}_j^{(k)}$ with controlled forward error. It involves substraction of two quantities always known to guaranteed high relative accuracy.*

Proof: First note that $\alpha_j^{(k)} \equiv \|\mathbf{a}_j^{(k)}\|_2 = \alpha_j^{(1)} \equiv \|\mathbf{a}_j^{(1)}\|_2$ for all j, k, and thus

$$\omega_j^{(k)} = \sqrt{\|\mathbf{a}_j^{(k)}\|_2^2 - (\xi_j^{(k)})^2} = \sqrt{(\alpha_j^{(1)})^2 - (\xi_j^{(k)})^2}. \tag{14}$$

Backward stability (Theorem 5.1) implies that $\|\tilde{\mathbf{a}}_j^{(k)}\|_2 = \alpha_j^{(1)}(1+\theta_j^{(k)})$, where upper bound on $|\theta_j^{(k)}|$ depends on the details of the algorithm. In Givens QR factorization $|\theta_j^{(k)}| \leq O(m+n)\mathfrak{e}$. For the Householder algorithm, $|\theta_j^{(k)}| \leq O(mk)\mathfrak{e}$. (This means that with $\mathfrak{e} \approx 10^{-16}$ and a million–by–million matrix we can have three accurate digits in all norms even with the most pessimistic case of straightforward computation.)

The computed column norms of the initial A satisfy $\tilde{\alpha}_j^{(1)} = \alpha_j^{(1)}(1 + O(m)\mathfrak{e})$, and thus

$$\tilde{\alpha}_j^{(1)} = \|\tilde{\mathbf{a}}_j^{(k)}\|_2 \frac{1 + O(m)\mathfrak{e}}{1 + \theta_j^{(k)}}, \quad 1 \le j \le n, \text{ and all } k.$$

Thus, once we have computed the column norms of A (the $\tilde{\alpha}_j^{(1)}$'s), we have accurate approximations of the norms of all columns of all computed $\tilde{A}^{(k)}$'s. Further, at any moment, $\tilde{\xi}_j^{(k)}$ can be available to high relative accuracy, $\|\tilde{\mathbf{x}}_j^{(k)}\|_2 = \tilde{\xi}_j^{(k)}(1 + \zeta_j^{(k)})$, $|\zeta_j^{(k)}| \le O(k\mathfrak{e})$.

Rewriting (13) to avoid underflow and overflow gives the following proposal for partial norm computation:

$$\tilde{\omega}_j^{(k)} = \tilde{\alpha}_j^{(1)} \odot \mathit{sqrt}(\mathit{max}(1 \ominus (\tilde{\xi}_j^{(k)} \oslash \tilde{\alpha}_j^{(1)})^2, 0)). \tag{15}$$

To analyze (15), we need to know how accurately we can compute $\tilde{\omega}_j^{(k)}$, and how accurately the exactly computed $\hat{\omega}_j^{(k)} = \tilde{\alpha}_j^{(1)}\sqrt{1 - (\tilde{\xi}_j^{(k)}/\tilde{\alpha}_j^{(1)})^2}$ approximates $\|\tilde{\mathbf{z}}_j^{(k)}\|_2$. It is straightforward to show that

$$\hat{\omega}_j^{(k)} = \|\tilde{\mathbf{z}}_j^{(k)}\|_2 \frac{1+O(m)\mathfrak{e}}{1+\theta_j^{(k)}}\sqrt{1 - \frac{e_j^{(k)}\|\tilde{\mathbf{x}}_j^{(k)}\|_2^2}{\|\tilde{\mathbf{a}}_j^{(k)}\|_2^2 - \|\tilde{\mathbf{x}}_j^{(k)}\|_2^2}}, \text{ where } e_j^{(k)} = \frac{(1+\theta_j^{(k)})^2}{(1+\zeta_j^{(k)})^2(1+O(m)\mathfrak{e})^2} - 1, \tag{16}$$

$$\text{and under the assumption } \|\tilde{\mathbf{a}}_j^{(k)}\|_2 \ne \|\tilde{\mathbf{x}}_j^{(k)}\|_2; \tag{17}$$

$$\tilde{\omega}_j^{(k)} = \hat{\omega}_j^{(k)}\sqrt{1+\epsilon_3}(1+\epsilon_4)(1+\epsilon_5)\sqrt{1 - \frac{f_j^{(k)}(\tilde{\xi}_j^{(k)})^2}{(\tilde{\alpha}_j^{(1)})^2 - (\tilde{\xi}_j^{(k)})^2}}, \text{ where} \tag{18}$$

$$f_j^{(k)} = (1+\epsilon_1)^2(1+\epsilon_2) - 1, \quad \max_{i=1:5}|\epsilon_i| \le \mathfrak{e}, \tag{19}$$

$$\text{and under the assumption that } \tilde{\alpha}_j^{(1)} \ne \tilde{\xi}_j^{(k)}. \tag{20}$$

Finally, note that the inverse relative distances (condition numbers for this computation)

$$\tilde{\chi}_j^{(k)} = \frac{(\tilde{\xi}_j^{(k)})^2}{(\tilde{\alpha}_j^{(1)})^2 - (\tilde{\xi}_j^{(k)})^2}, \quad \chi_j^{(k)} = \frac{\|\tilde{\mathbf{x}}_j^{(k)}\|_2^2}{\|\tilde{\mathbf{a}}_j^{(k)}\|_2^2 - \|\tilde{\mathbf{x}}_j^{(k)}\|_2^2}$$

can safely be compared against given tolerance. Only $\tilde{\chi}_j^{(k)}$ is accessible, and it will be below given tolerance tol if $\tilde{\xi}_j^{(k)}/\tilde{\alpha}_j^{(1)} < \sqrt{tol/(1+tol)}$. In that case $\chi_j^{(k)} < tol(1+\varsigma)/(1-\varsigma\cdot tol) \approx tol$, where $\|\tilde{\mathbf{x}}_j^{(k)}\|_2/\|\tilde{\mathbf{a}}_j^{(k)}\|_2 = (\tilde{\xi}_j^{(k)}/\tilde{\alpha}_j^{(1)})(1+\varsigma)$, and $|\varsigma|$ is bounded by roundoff $\mathfrak{e}$ times a modest polynomial in m. Hence, we can always correctly predict and thus avoid conditions for catastrophic cancelations.

If $\tilde{\xi}_j^{(k)}$ and $\tilde{\alpha}_j^{(1)}$ are too close, then $\tilde{\omega}_j^{(k)}$ is evaluated by explicit norm computation of $\tilde{\mathbf{z}}_j^{(k)}$. Further, in that case the updating formula is reset: $\tilde{\alpha}_j^{(1)} = \tilde{\omega}_j^{(k)}$, and $\tilde{\xi}_j^{(k)}$ is set to zero. ⊠

Assuming familiarity with the xGEQP3 code, we give in Table 4 partial column norm update strategy, which uses both global (14) and local (6) formulas. Note that we allow at most one use

```
$\tilde{\xi}_j^{(k+1)} = \max\{\tilde{\xi}_j^{(k)}, \beta_j^{(k+1)}\}\sqrt{1 + (\min\{\tilde{\xi}_j^{(k)}, \beta_j^{(k+1)}\}/\max\{\tilde{\xi}_j^{(k)}, \beta_j^{(k+1)}\})^2}$
$t_0 = \tilde{\xi}_j^{(k+1)}/\tilde{\alpha}_j^{(1)}$
**if** ( $t_0 < \gamma_1$ ) **then**
    $\tilde{\omega}_j^{(k+1)} = \tilde{\alpha}_j^{(1)}\sqrt{(1-t_0)(1+t_0)}$
**else**
    $t_1 = |\beta_j^{(k+1)}/\tilde{\omega}_j^{(k)}|$ ;  $t_2 = \max\{0, (1-t_1)(1+t_1)\}$
    **if** ( $(t_2 < \gamma_2)$ **and** $(\tilde{\omega}_j^{(k)} > 0)$) **then**
        $\tilde{\omega}_j^{(k+1)} = -\tilde{\omega}_j^{(k)}\sqrt{t_2}$
    **else**
        push $\tilde{z}_j^{(k+1)}$ to stack of unresolved columns
    **end if**
**end if**
```

Table 4: Critical part in the column norm update.

of the formula (6) per Householder reflector per column, and that control counter is obtained by flipping the sign of $\tilde{\omega}_j^{(k+1)}$. The parameters γ_1, γ_2 can be taken e.g. around $1 - \sqrt{\varepsilon}$ (for *tol* $\approx 1/\sqrt{\varepsilon}$), with necessary fine–tuning. (Setting $\gamma_2 = 0$ switches off updating by the local formula.)

4 New software for Businger–Golub pivoting

We now present our new code for the Businger–Golub QR factorization – a modification of the xGEQP3 subroutine, with new norm updating strategies as outlined above. The goal is *strongly* backward stable numerical software (with column–wise small backward error *and properly structured computed upper triangular factor.*), which can be immediately used in our SVD code [8], [9], and which we will recommend as a replacement for the current LAPACK software.

For this report, we have tested preliminary codes SGEQP3A (Table 3) and SGEQP3Z (Table 4).

A collection of 1792 1000×800 test matrices is generated following [9, §3.3.1]. The matrices are divided into eight groups, each groups containing 224 matrices of the form $A = A_c D$ with fixed $\kappa_2(A_c) = 10^i$ for the i–th group. The diagonal scaling can have arbitrarily high condition number, we have taken up to 10^{12}. The whole collection is enumerated so that the values $\kappa_2(A_c)$ form nondecreasing sequence. The first 224 matrices have $\kappa_2(A_c) = 10$, the next 224 have $\kappa_2(A_c) = 10^2$ and so on. Around the index 900, $\kappa_2(A_c)$ reaches $1/\sqrt{\varepsilon}$.

In Figure 7 and Figure 8 we show the timing results of one test run with our first versions of the code, SGEQP3A (Table 3) and SGEQP3Z (Table 4), respectively. In SGEQP3Z we used only the global update formula, i.e. $\gamma_2 = 0$. With local formula added to updating strategy, the performance of SGEQP3Z can be improved by one or two percent.

It is clear that, depending on the matrix, there is a drop in the performance because more often explicit computation of the column norms interferes with block strategy. From the numerical point of view, we find this few percent increase in run time a negligible price for the

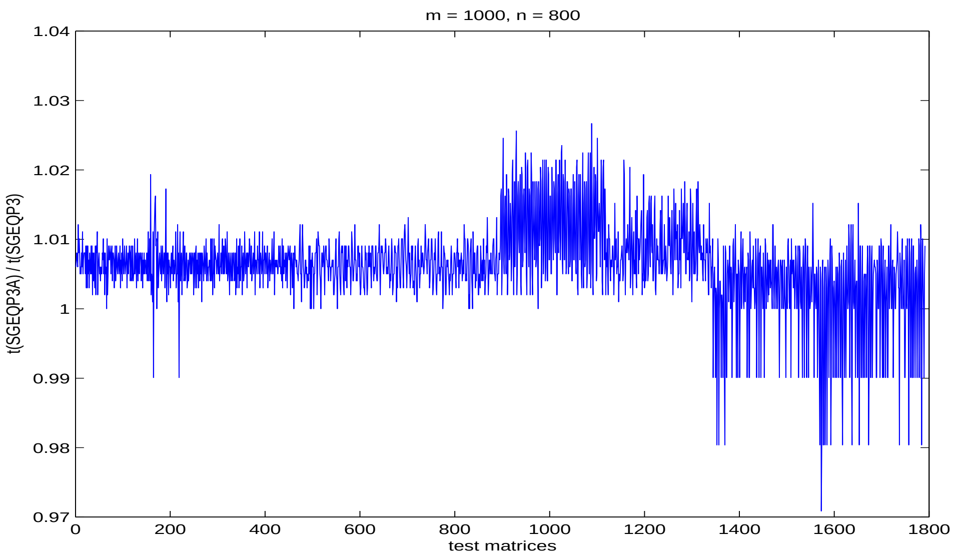

Figure 7: *The relative timings*: time(SGEQP3A) / time(SGEQP3).

provable numerical reliability of the software.

5 Importance of pivoting

5.1 Perturbation theory

In the forward error analysis of the QR factorization, one usually ignores the pivoting. For the backward error analysis of a particular algorithm, it is usually said that pivoting is equivalent to initial permutation of matrix columns, followed by the factorization without pivoting. Then, in order to simplify the notation, permutation matrix is replaced with identity, and the subject of the ensuing forward analysis is factorization without pivoting.

An exception where pivoting enters the backward analysis explicitly and substantially is the complete pivoting of Powell and Reid [13], for an analysis (which assumes that column pivoting is executed properly) see Cox and Higham [4] and Higham [11]. For perturbation bounds in the QR factorization, we refer the reader to [15], [17], [16].

Our goal in this section is to show that pivoting can play important role in the forward error analysis (perturbation theory) of the QR factorization. To specify the kind of perturbations of interest, we recall backward stability result for the Householder and Givens QR factorization algorithms. (For more details see [5], [10].)

Theorem 5.1 *Let $A\tilde{P} \approx \tilde{Q}\tilde{R}$ be the computed QR factorization with column pivoting. ($\tilde{P}$ is permutation matrix obtained by applications of prescribed rule of a concrete pivoting.) Then there exist backward perturbation δA and an orthonormal $\hat{Q}$ such that $(A+\delta A)\tilde{P} = \hat{Q}\tilde{R}$, where $\|\hat{Q}-\tilde{Q}\|_F \leq \eta_Q$, and*

$$\|\delta A(:,i)\|_2 \leq \eta_A \|A(:,i)\|_2, \quad 1 \leq i \leq n.$$

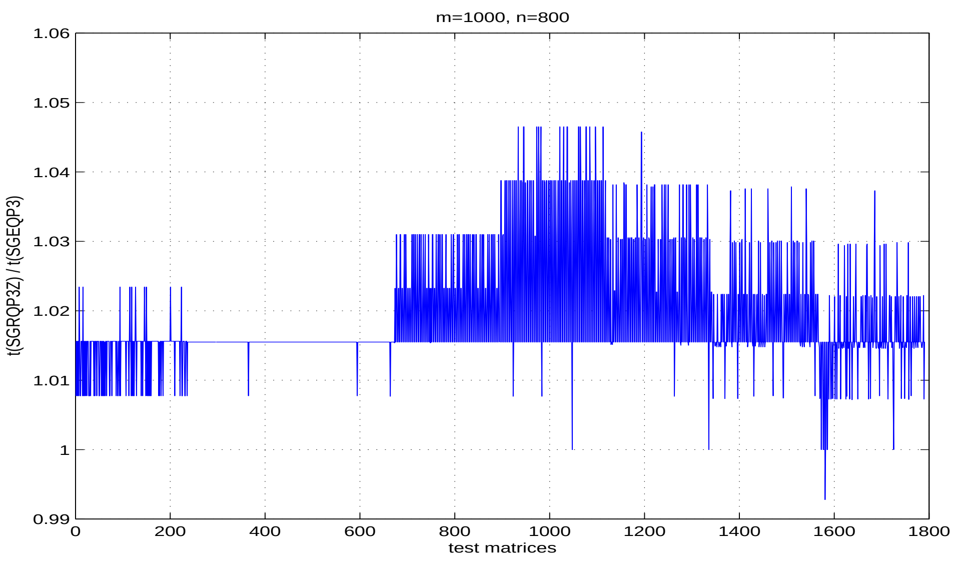

Figure 8: *The relative timings*: time(SGEQP3Z) / time(SGEQP3).

The error bounds η_A, η_Q depend on the implementation details, and both are bounded by a low degree polynomial times the machine precision $\mathfrak{e}$. For the Givens rotation based computation $\eta_A, \eta_Q \approx O(m+n)\mathfrak{e}$. For Householder reflection based algorithm $\eta_A, \eta_Q \approx O(mn)\mathfrak{e}$.

Under the perturbation from the previous theorem, the QR factorization $(A+\delta A)\tilde{P} = \hat{Q}\tilde{R}$ must be compared with the exact factorization $A\tilde{P} = QR$, but with the intended and actually computed structures of $\tilde{R}$ taken into considerations.
For the sake of simplicity, let us assume that both A and $\tilde{A} = A + \delta A$ have full column rank.

Note that we implicitly deal with the Cholesky factorizations $\tilde{H} = \tilde{R}^T\tilde{R}$ and $H = R^TR = \tilde{H} + \delta\tilde{H}$. Following the ideas from [7], we write $R^TR = \tilde{R}^T(I+E)\tilde{R}$, with

$$E \equiv \tilde{R}^{-T}\delta\tilde{H}\tilde{R}^{-1} = -\hat{Q}^T\delta A\tilde{P}\tilde{R}^{-1} - \tilde{R}^{-T}\tilde{P}^T\delta A^T\hat{Q} + \tilde{R}^{-T}\tilde{P}^T\delta A^T\delta A\tilde{P}\tilde{R}^{-1}.$$

The matrix E expresses the size of $\delta\tilde{H}$ relative to $\tilde{H} = \tilde{R}^T\tilde{R}$. To bound E, first define $\tilde{R}_c$ to be the matrix obtained from $\tilde{R}$ by scaling its columns to have unit Euclidean norm. Then note that $E = \hat{Q}^TF + F^T\hat{Q} + F^TF$, $F = -\delta A\tilde{P}\tilde{R}^{-1}$, together with Theorem 5.1, implies

$$\|E\|_F \leq 2\|(\hat{Q}\hat{Q}^T)F\|_F + \|F\|_2\|F\|_F, \quad \|F\|_F \leq \frac{\sqrt{n}\eta_A}{1-\eta_A}\|\tilde{R}_c^{-1}\|_2. \tag{21}$$

Theorem 5.2 *Let $(A+\delta A)\tilde{P} = \hat{Q}\tilde{R}$ be the factorization from Theorem 5.1, where the permutation matrix $\tilde{P}$ is determined following the Businger–Golub column pivoting. Let $A\tilde{P} = QR$ be the exact QR factorization of $A\tilde{P}$, and let $R = \tilde{R} + \delta\tilde{R}$. Further, let*

$$|\tilde{R}_{ii}| \geq \tilde{\rho}_i\sqrt{\sum_{k=1}^{j}|\tilde{R}_{kj}|^2}, \quad 1 \leq i \leq j \leq n. \tag{22}$$

If the pivoting has functioned properly, with correct choices of pivot columns, then $\tilde{\rho}_i \geq 1$ for all i. Then for all $i = 1, \ldots, n$

$$\begin{aligned} \|\delta\tilde{R}(:,i)\|_2 &\leq \|\Gamma(:,1:i)\|_2 \|\tilde{R}(:,i)\|_2, \;\; \delta\tilde{R}_{ii} = \Gamma_{ii}\tilde{R}_{ii}, & (23) \\ \|\delta\tilde{R}(i,:)\|_\infty &\leq \frac{1}{\tilde{\rho}_i}\|\Gamma(i,:)\|_2 \|\tilde{R}(i,:)\|_\infty, & (24) \end{aligned}$$

where $\delta\tilde{R} = \Gamma\tilde{R}$, and the matrix $\Gamma = R\tilde{R}^{-1} - I$ is bounded as follows:

- *If $\|E\|_F < \frac{1}{2}$, then $\|\Gamma\|_F \leq \dfrac{\sqrt{2}\|E\|_F}{1+\sqrt{1-2\|E\|_F}}$.*
- *Let $\chi_n = 1/2 + \lceil \log_2 n \rceil$. If $\|E\|_2 \leq \dfrac{1}{4\chi_n^2}$, then $\|\Gamma\|_2 \leq \dfrac{2\chi_n\|E\|_2}{1+\sqrt{1-4\chi_n^2\|E\|_2}}$.*
- *It always holds that $\|\Gamma\|_F \leq \sqrt{8n + 2\sqrt{n}}\|E\|_F$.*

In all cases, the norm of E is bounded using (21). Further, $\hat{Q} - Q = Q\Gamma - F$.

Proof: Note that we can write $\tilde{R}^{-T}R^T R\tilde{R}^{-1} = I + E$, which implies that $R\tilde{R}^{-1}$ is the Cholesky factor of $I + E$ and it can be written as $R\tilde{R}^{-1} = I + \Gamma$. Hence, $\delta\tilde{R} = \Gamma\tilde{R}$ and thus (23) follows. To derive (24), note that

$$\begin{aligned} \|\delta\tilde{R}(i,:)\|_\infty &= \max_{j=i:n}\left|\sum_{k=i}^{j}\Gamma_{ik}\tilde{R}_{kj}\right| \leq \max_{j=i:n}\|\Gamma(i,:)\|_2\sqrt{\sum_{k=i}^{j}|\tilde{R}_{kj}|^2} \\ &\leq \|\Gamma(i,:)\|_2\frac{1}{\tilde{\rho}_i}|\tilde{R}_{ii}| \leq \frac{1}{\tilde{\rho}_i}\|\Gamma(i,:)\|_2\|\tilde{R}(i,:)\|_\infty. \end{aligned}$$

It remains to estimate Γ, or equivalently, perturbation of the Cholesky factor of the identity under the perturbation $I \rightsquigarrow I + E$. We use the perturbation results from [7], and the proof is completed.

⊠

Remark 5.1 Perturbation estimates are derived relative to $\tilde{R}$ (instead to R) because (22) allows us to easily monitor its structure. The structure of exact triangular factor R of $A\tilde{P}$, with the computed permutation $\tilde{P}$, is not known.

Remark 5.2 Similarly as in [7], we note that the proof of Theorem 5.2 contains an element–wise bound: $\delta\tilde{R}_{ij} = \sum_{k=i}^{j}\Gamma_{ik}\tilde{R}_{kj}$ implies $|\delta\tilde{R}_{ij}| \leq \|\Gamma(i,:)\|_2\sqrt{\sum_{k=i}^{j}|\tilde{R}_{kj}|^2}$. Note in particular that $|\delta\tilde{R}_{ii}| \leq |\Gamma_{ii}||\tilde{R}_{ii}|$, which additionally stresses the importance of having dominant elements along the diagonal.

Remark 5.3 The value of $\|\tilde{R}_c^{-1}\|_2$ can be estimated by an $O(n^2)$ condition estimator and then Theorem 5.2 and (21) can be used in practice. The relevant condition number in this case is $\|\tilde{R}_c^{-1}\|_2$ and it properly reflects the column–wise structure of the perturbation (backward error). Recall that

$$\|\tilde{R}_c^{-1}\|_2 \leq \kappa_2(\tilde{R}_c) \equiv \frac{\sigma_{\max}(\tilde{R}_c)}{\sigma_{\min}(\tilde{R}_c)} \leq \sqrt{n} \min_{D=\mathrm{diag}} \kappa_2(\tilde{R}D) = \sqrt{n} \min_{D=\mathrm{diag}} \kappa_2(\tilde{A}D).$$

Example 5.1 Let $A \equiv I_2 R = \begin{pmatrix} \epsilon & \epsilon \\ 0 & \frac{1}{\epsilon} \end{pmatrix}$, and $\tilde{A} = A + \delta A = \begin{pmatrix} \epsilon & \epsilon \\ \epsilon^2 & \frac{1}{\epsilon} \end{pmatrix}$. Take ϵ so small that $1+\epsilon^2 \approx 1$ in the working precision. The QR factorization of $\tilde{A}$ reads

$$\tilde{A} = \frac{1}{\sqrt{1+\epsilon^2}} \begin{pmatrix} 1 & -\epsilon \\ \epsilon & 1 \end{pmatrix} \begin{pmatrix} \epsilon\sqrt{1+\epsilon^2} & \dfrac{1+\epsilon}{\sqrt{1+\epsilon^2}} \\ 0 & \dfrac{1}{\epsilon}\dfrac{1-\epsilon^3}{\sqrt{1+\epsilon^2}} \end{pmatrix} \approx \begin{pmatrix} 1 & -\epsilon \\ \epsilon & 1 \end{pmatrix} \begin{pmatrix} \epsilon & 1+\epsilon \\ 0 & \frac{1}{\epsilon} \end{pmatrix}.$$

The column–wise bound (23) holds, but the element R_{12} and the whole first row of the upper triangular factor are substantially changed. The cosine of the angle between the first two rows of R is $1/\sqrt{2}$, while the first two rows of $\tilde{R}$ are almost parallel.

On the other hand, if we pivot, the QR factorizations are

$$\begin{aligned}
\begin{pmatrix} \epsilon & \epsilon \\ 0 & \frac{1}{\epsilon} \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} &= \frac{1}{\sqrt{1+\epsilon^4}} \begin{pmatrix} \epsilon^2 & -1 \\ 1 & \epsilon^2 \end{pmatrix} \begin{pmatrix} \dfrac{\frac{1}{\epsilon}+\epsilon^3}{\sqrt{1+\epsilon^4}} & \dfrac{\epsilon^3}{\sqrt{1+\epsilon^4}} \\ 0 & \dfrac{-\epsilon}{\sqrt{1+\epsilon^4}} \end{pmatrix}, \\
&\approx \begin{pmatrix} \epsilon^2 & -1 \\ 1 & \epsilon^2 \end{pmatrix} \begin{pmatrix} \frac{1}{\epsilon} & \epsilon^3 \\ 0 & -\epsilon \end{pmatrix}; \\
\begin{pmatrix} \epsilon & \epsilon \\ \epsilon^2 & \frac{1}{\epsilon} \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} &= \frac{1}{\sqrt{1+\epsilon^4}} \begin{pmatrix} \epsilon^2 & -1 \\ 1 & \epsilon^2 \end{pmatrix} \begin{pmatrix} \dfrac{\frac{1}{\epsilon}+\epsilon^3}{\sqrt{1+\epsilon^4}} & \dfrac{\epsilon^3+\epsilon^2}{\sqrt{1+\epsilon^4}} \\ 0 & \dfrac{-\epsilon+\epsilon^4}{\sqrt{1+\epsilon^4}} \end{pmatrix} \\
&\approx \begin{pmatrix} \epsilon^2 & -1 \\ 1 & \epsilon^2 \end{pmatrix} \begin{pmatrix} \frac{1}{\epsilon} & \epsilon^3+\epsilon^2 \\ 0 & -\epsilon \end{pmatrix}.
\end{aligned}$$

In this case, the perturbation of R is column–wise (23) and row–wise ((24), Remark 5.2) small, although the relative change in R_{12} is arbitrarily bad.

5.2 Preconditioning

Let $AP = QR$ be the Businger–Golub QR factorization. Denote by A_c, R_c the matrices obtained from A, R, respectively, by scaling their columns to have unit Euclidean lengths, $A = A_c D_c$, $R = R_c D_c$, where $D_c = \mathrm{diag}(\|R(:,i)\|_2)$. From §5.1 it follows that $\kappa_2(R_c) = \kappa_2(A_c)$

is the relevant condition number for perturbations of the QR factorization under column–wise perturbations of A. Note that $\kappa_2(R_c) = \kappa_2(A_c)$ independent of the permutation P.

However, permutation P can substantially change another condition number related to R. Let $R = D_r R_r$, where D_r is the diagonal matrix of the Euclidean lengths of the rows of R. Preconditioning property of the Businger–Golub QR factorization is here understood in the light of the fact that $\kappa_2(R_r)$ can be much smaller and never much bigger that $\kappa_2(A_c)$. The following Proposition ([5], [6]) gives an estimate.

Proposition 5.1 *Let R be a nonsingular upper triangular factor in the factorization (1). Then*

$$\| \, |R_r{}^{-1}| \, \|_2 \leq \sqrt{n} + \max_{i<j} \frac{(D_r)_{jj}}{(D_c)_{ii}} \cdot \| \, |R_c{}^{-1} - \mathrm{diag}(R_c{}^{-1})| \, \|_2 \qquad (25)$$

$$\| \, |R_r{}^{-1}| \, \|_2 \leq \sqrt{n} \| \, |R_c{}^{-1}| \, \|_2, \qquad (26)$$

where the matrix absolute value is defined element–wise. Moreover, $\|R_r^{-1}\|_2$ is bounded by $O(2^n)$, independent of A. With exception of rare pathological cases, $\|R_r^{-1}\|_2$ is below n for any A.

The proof is based on the fact that for $i \leq j$

$$(R_r^{-1})_{ij} = (R_c^{-1})_{ij} \frac{(D_r)_{jj}}{(D_c)_{ii}} \leq (R_c^{-1})_{ij} \sqrt{n-j+1} \frac{|R_{jj}|}{|R_{ii}|} \leq (R_c^{-1})_{ij} \sqrt{n-j+1},$$

where both the dominance and non–increasing order of the diagonal elements contribute to the inequalities.

Now, $\kappa_2(A) = \kappa_2(R)$, $\kappa_2(A_c) = \kappa_2(R_c)$, but it is possible that $\kappa_2(R_r) \ll \kappa_2(A_c) \leq \sqrt{n}\kappa_2(A)$. Since $\|R_r^{-1}\|_2 \leq n\|R_c^{-1}\|_2$, $\kappa_2(R_r)$ is in the worst case only $n^{3/2}$ times larger than $\kappa_2(A_c)$. In fact, the stronger the scaling D_c (meaning that A has very differently scaled columns and thus larger condition number), the Businger–Golub QR factorization will compute R with smaller $\kappa_2(R_r)$. This feature is one of the important ingredients in the preconditioning Jacobi SVD algorithm [8], [9].

The fact that column pivoting produces small $\kappa_2(R_r)$ is in accordance with the row–wise forward error in case of perturbations (Theorem 5.2). Just to illustrate, consider solving $Rx = b$ and $(R+\delta R)\tilde{x} = b$. The perturbed solution is $\tilde{x} = (I + R^{-1}\delta R)x$, where $R^{-1}\delta R = R_r^{-1}(D_r^{-1}\delta R)$. Pivoting moves the triangular factor away from ill–conditioning (with respect to row–wise small perturbations).

6 Acknowledgments

The authors thank Jim Demmel, Jason Riedy (UC Berkeley) and Nick Higham (Manchester) for constructive comments.

References

[1] C. H. Bischof and G. Quintana-Orti. Computing rank–revealing QR factorizations of dense matrices. Argonne Preprint MCS–P559–0196, Argonne National Laboratory, 1990.

[2] P. A. Businger and G. H. Golub. Linear least squares solutions by Householder transformations. *Numer. Math.*, 7:269–276, 1965.

[3] Sh. Chandrasekaran and I. C. F. Ipsen. On rank–revealing factorizations. *SIAM J. Matrix Anal. Appl.*, 15(2):592–622, 1994.

[4] A. J. Cox and N. J. Higham. Stability of Householder QR factorization for weighted least squares problems. In D. F. Griffiths, D. J. Higham, and G. A. Watson, editors, *Numerical Analysis 1997, Proceedings of the 17th Dundee Biennial Conference*, volume 380 of *Pitman Research Notes in Mathematics*, pages 57–73. Addison Wesley Longman, Harlow, Essex, UK, 1998.

[5] Z. Drmač. *Computing the Singular and the Generalized Singular Values.* PhD thesis, Lehrgebiet Mathematische Physik, Fernuniversität Hagen, 1994.

[6] Z. Drmač. A posteriori computation of the singular vectors in a preconditioned Jacobi SVD algorithm. *IMA J. Numer. Anal.*, 19:191–213, 1999.

[7] Z. Drmač, M. Omladič, and K. Veselić. On the perturbation of the Cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 15(4):1319–1332, 1994.

[8] Z. Drmač and K. Veselić. New fast and accurate Jacobi SVD algorithm: I. Technical report, Department of Mathematics, University of Zagreb, Croatia, June 2005. LAPACK Working Note 169.

[9] Z. Drmač and K. Veselić. New fast and accurate Jacobi SVD algorithm: II. Technical report, Department of Mathematics, University of Zagreb, Croatia, June 2005. LAPACK Working Note 170.

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

[11] N. J. Higham. QR factorization with complete pivoting and accurate computation of the SVD. *Lin. Alg. Appl.*, 309:153–174, 2000.

[12] W. Kahan. Numerical linear algebra. *Canadian Mathematical Bulletin*, 9(6):757–801, 1966.

[13] M. J. D. Powell and J. K. Reid. On applying Householder transformations to linear least squares problems. In *Information Processing 68, Proc. International Federation of Information Processing Congress, Edinburgh, 1968*, pages 122–126. North Holland, Amsterdam, 1969.

[14] G. Quintana-Orti and E. S. Quintana-Orti. Guaranteeing termination of Chandrasekaran and Ipsen's algorithm for computing rank–revealing QR factorizations. Argonne Preprint MCS–P564–0196, Argonne National Laboratory, 1990.

[15] G. W. Stewart. Perturbation bounds for the QR decomposition of a matrix. *SIAM J. Numer. Anal.*, 14(3):509–518, 1977.

[16] G. W. Stewart. On the perturbation of LU, Cholesky, and QR factorizations. *SIAM J. Matrix Anal. Appl.*, 14(4):1141–1145, 1993.

[17] Ji-Guang Sun. Perturbation bounds for the Cholesky and QR factorizations. *BIT*, 31:341–352, 1991.

[18] K. Veselić and V. Hari. A note on a one–sided Jacobi algorithm. *Numer. Math.*, 56:627–633, 1989.