On the failure of rank revealing QR factorization software – a case study

ZLATKO DRMAČ and ZVONIMIR BUJANOVIĆ
Department of Mathematics, University of Zagreb, 10000 Zagreb, Croatia.

This paper 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, the software implementation of the factorization can catastrophically fail to produce properly structured triangular factor, thus leading to potentially severe underestimate of a matrix's numerical rank. The 30 year old problem, dating back to LINPACK, has (undetectedly) badly affected many computational routines and software packages, as well as the study of rank revealing QR factorizations. We combine computer experiments and numerical analysis to isolate, analyze and fix the problem. Our modification of the current LAPACK xGEQP3 routine is already included in the LAPACK 3.1.0 release. The modified routine is numerically more robust and with a negligible overhead. We also provide a new, equally efficient and provably numerically safe, partial column norm updating strategy.

Categories and Subject Descriptors: G1 [**Numerical Analysis**]: Numerical Linear Algebra—*Error Analysis*; G4 [**Mathematical Software**]: —*Reliability*; *Robustness*

General Terms: Algorithms, Reliability, Theory

Additional Key Words and Phrases: Rank revealing QR factorization, pivoting,

1. INTRODUCTION

During the implementation and testing of a new Jacobi–type SVD algorithm [Drmač and Veselić 2007a; 2007b], 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 a 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 was used in our SVD software. It implements a BLAS 3 version [Quintana-Orti et al. 1998] of the Businger–Golub [1965] pivot strategy which, for $A \in \mathbb{R}^{m\times n}$, computes

Authors' address: Department of Mathematics, University of Zagreb, Bijenička 30, 10000 Zagreb, Croatia. This work is supported by the Croatian Ministry of Science, Education and Sport under grants 0037120 (*Numerical Analysis and Matrix Theory*) and 0372783–2750 (*Spectral decompositions–numerical methods and applications*).

permutation matrix P, orthonormal Q and upper triangular matrix R such that

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

Our detailed experimental investigation and numerical analysis have shown that the LINPACK and LAPACK software implementations of (1) may exhibit serious instabilities, which may lead to numerical catastrophes in engineering applications. For instance, the numerical rank of a matrix can be severely underestimated. We have tracked the problem down to inaccurately updated partial column norms needed to implement (1). More precisely, we have found a numerical bug in the safety switch which is used to protect the updating formula from massive cancellations. The flaw is so subtle that wrong pivoting can be provoked or cured simply by changing the compiler options, or even by writing certain auxiliary variables to the screen. This sensitivity indicates computation near a singularity. In other words, the condition number of the updating formula is mistakenly underestimated.

We give detailed description of the problem and provide two solutions. Our first modification, adopted by LAPACK 3.1.0, is a quick fix for the current LAPACK code. It uses a better safety switch and it does not affect the input/output specifications of the current LAPACK's routines. We also propose a new provably numerically safe partial column norm updating scheme, with an overhead of n extra locations in the workspace. In both cases, the run–time overhead is negligible.

This issue has implications to the backward (or mixed) stability. It is well known that the QR factorization is backward stable in floating point arithmetic. Therefore, in a 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 backward stability statement we insist on the structure: $\tilde{R}$ must be upper triangular and $\tilde{Q}$ must be numerically orthonormal. Although the factorization is computed with *pivoting*, which should impose certain *structure* on the triangular factor, the structure of $\tilde{R}$ is never mentioned in backward error analysis. It certainly cannot be taken for 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 not considered to be the responsibility of the backward error analysis. Thus, strictly speaking, if the structure of $\tilde{R}$ is not guaranteed (at least in the mixed stability sense), the computation of (1) is not backward (nor mixed) stable. Our new scheme is provably stable implementation of the pivoted QR factorization (1).

The material is organized as follows. In §2 we give several examples which illustrate how the 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. These examples should convince the reader that the problem is serious. An experimental diagnostics is presented in §3. Section 4 offers an analysis of the erratic behavior and proposes modifications of the current LAPACK code. In §5, 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, §6 recalls the importance of pivoting from the numerical point of view. We discuss how pivoting contributes to more accurate factorization, and how it can be used as a preconditioning technique.

2. EXAMPLES OF SOFTWARE FAILURE

To make our case, we first give several examples of software failure. In particular, we show that both the monotonicity of the diagonal and the diagonal dominance can be destroyed in common software implementations of (1). We also illustrate how this failure causes breakdowns of more complex routines, such as SVD computation or least squares solvers.

The matrix which first exposed the weakness of the code was the famous Kahan [1966] 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 numerical deficiency of $\mathfrak{K}$ is exactly one [Zha 1997]. 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 [Powell and Reid 1969] leaves $\mathfrak{K}$ unchanged. It is also well known that rounding errors during the computation may provoke permutation different from the identity, and the computed $\tilde{R} \neq \mathfrak{K}$ could be rank–revealing in the sense that $|\tilde{R}_{nn}|$ correctly estimates the magnitude of the minimal singular value $\sigma_{\min}(\mathfrak{K})$ of $\mathfrak{K}$.

Our first examples were generated using the LAPACK xGEQP3 and xGEQPF procedures under a GNU FORTRAN compiler. Later on, the same problem is found in other software packages (MATLAB, SciLab, Octave, Intel Fortran compiler). In this presentation we use examples generated in MATLAB 6.5. under MS Windows.

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 $\max_{i=1:n-1} \mu_i/|R_{ii}| \leq 1$.

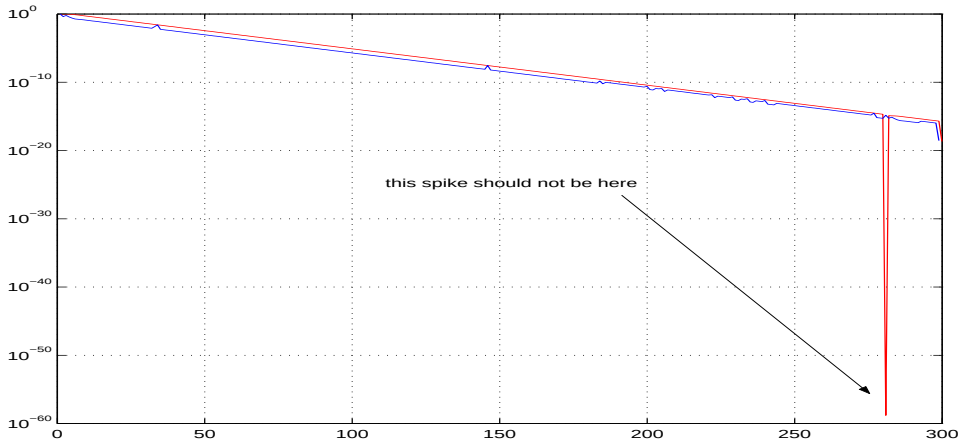

Fig. 1. *The values of* $|R_{ii}|$ *(red line) and* μ_i *(blue line) for the matrix* $\mathfrak{K}(300, c)$ *in Example 2.1. Here* $\max_{i=1:n-1} \mu_i/|R_{ii}| \approx 9.2734 \cdot 10^{43}$ *(and it should be at most* 1*).*

EXAMPLE 2.1. Take $n = 300$ and $c = 4.664999999999993 \cdot 10^{-1}$, define $\mathfrak{A} = \mathfrak{K}(n, c)$ and compute $[Q, R, P] = qr(\mathfrak{A})$.[1] 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 $|R_{281,281}/R_{280,280}| < 10^{-44}$. Having the property (1) of R in mind, one assumes 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 column permutation, 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 the index 281. 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 Figure 2.

To show the subtlety of the problem, take $\tilde{c} = c * (1 + eps) = 4.664999999999994 \cdot$

[1]The results of these experiments depend on the machine and on the way one generates $\mathfrak{K}$ because just 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.

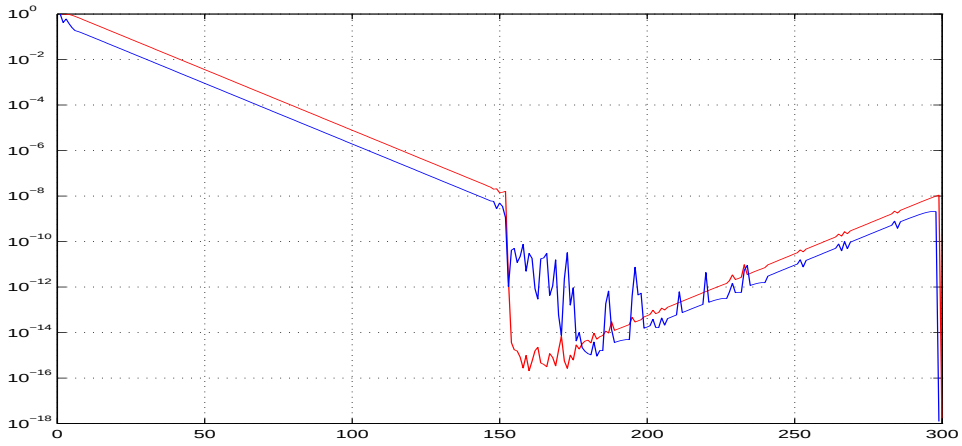

Fig. 2. *The values of* $|R_{ii}|$ *(red line) and* μ_i *(blue line) for the row and column permuted matrix* $\mathfrak{K}(300, c)$ *in Example 2.1. Here* $\max_{i=1:n-1} \mu_i/|R_{ii}| \approx 2.7330 \cdot 10^5$.

10^{-1} (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. A comparison of the diagonal entries of the computed factors R and $\tilde{R}$ is given in (2). Note that the computation is in double precision (MATLAB), so underflow is not an issue here.

i	$\lvert R_{ii}\rvert$	$\lvert \tilde{R}_{ii}\rvert$
⋮	⋮	
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	1.510608517753438e-059	$1.756268120293820e-015$
282	$1.191304454419907e-015$	$1.553456382058059e-015$
283	$1.173764088030493e-015$	$1.374065100352209e-015$
⋮	⋮	⋮
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 3). 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

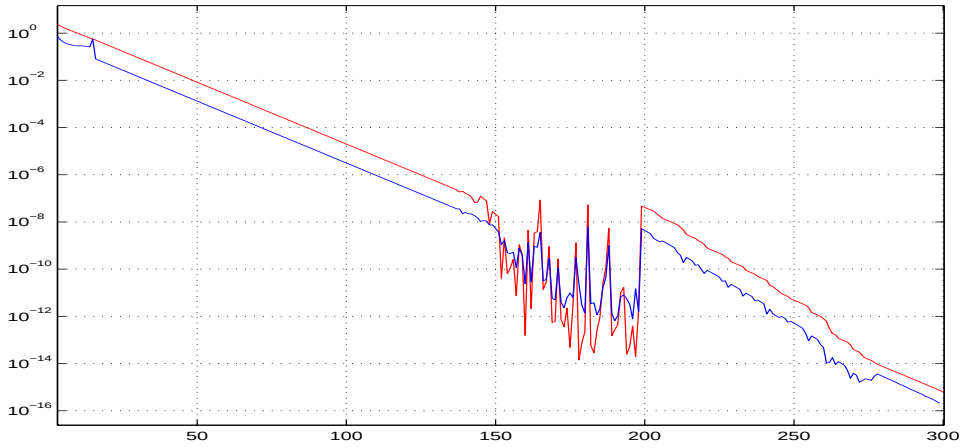

Fig. 3. *The values of* $|R_{ii}|$ *(red line) and* μ_i *(blue line),* $i = 1:300$ *for the matrix* $\mathfrak{M} = \mathfrak{B} + \mathfrak{B}^T$, $\mathfrak{B} = \mathfrak{K}(300, 0.4630)$. *Here* $\max_{i=1:n-1} \mu_i/|R_{ii}| \approx 1.6633 \cdot 10^3$.

obtain $\max_{i=1:n-1} \mu_i/|R_{ii}| \approx 4.1037 \cdot 10^9$. (We will use this matrix in Example 2.4.)

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 failing to preserve the pivot ordering impacts these solvers' reliability. We give a 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 a new Jacobi type SVD algorithm [Drmač and Veselić 2007a; 2007b]. To illustrate, we first describe a simplified variant based on the accelerated Jacobi algorithm [Veselić and Hari 1989].

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 as in (1). Instead of implicit diagonalization of $\mathsf{M} \equiv R^T R = P^T(A^T A)P$, the 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 $\|R_r^{-1}\|_2$, where R_r is obtained by scaling the rows of R to unit Euclidean length. The value $\|R_r^{-1}\|_2$ is expected to be moderate for any A, and it holds that $\|R_r^{-1}\|_2 \leq n^{3/2} \min_{D=diag} \kappa_2(AD)$, where $\kappa_2(\cdot)$ is the spectral condition number. (See §6.2.)

If the computed $\tilde{R} \approx R$ violates (1), as in §2.1, then the value of $\|\tilde{R}_r^{-1}\|_2$ may even overflow. This means that preconditioning completely fails – instead of reducing, it drastically increases the condition number. An occurrence of this situation triggers safety switches in the Jacobi SVD algorithm [Drmač and Veselić 2007a; 2007b].

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

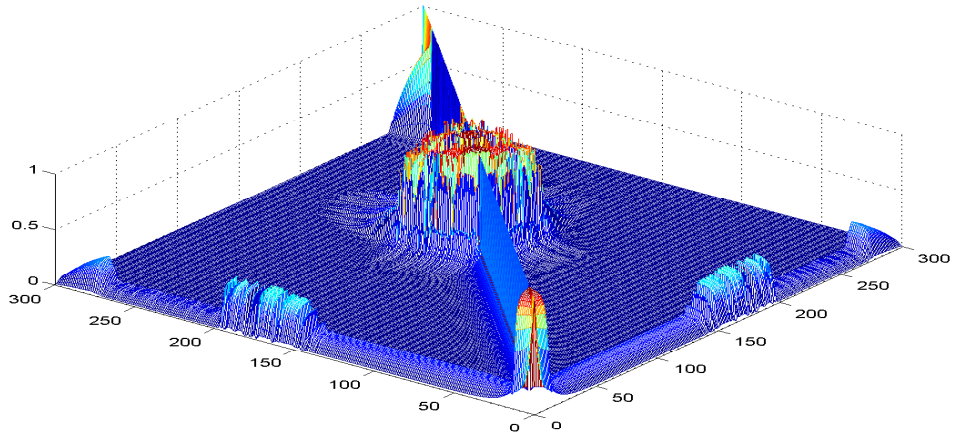

Fig. 4. *The matrix* W_s*, where* $\tilde{R}$ *is the matrix used in Figure 3. The ill–conditioned "tower"* $\mathsf{W}_s(150\!:\!200, 150\!:\!200)$ *is a result of wrong pivot choices.*

as in Figure 4, where $\mathsf{W}_s(150\!:\!200, 150\!:\!200)$ 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 this 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. 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 a 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 *backslash* operator to solve $\min_x \|Ax - d\|_2$.

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

```
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 the 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. The computed 305–th singular value of A, $\sigma_{305}(A) \approx 2.0470 \cdot 10^{-8}$ is sufficiently accurate to conclude, using the Eckart–Young–Mirsky theorem, that the distance to the closest matrix of rank at most 304 is more than $2 \cdot 10^{-8}$.

To understand what caused this warning, we compute `[Q,R,P] = qr(A)` and analyze the structure of R. See Figure 5. It is clear where 304 comes from – here

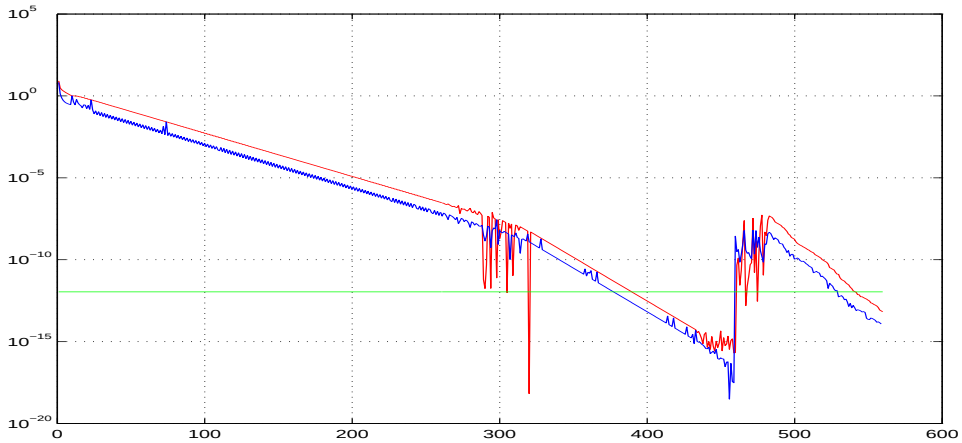

Fig. 5. *The values of* $|R_{ii}|$ *(red line) and* μ_i *(blue line),* $i = 1 : 559$ *for the matrix* A *in Example 2.4. The green line marks the tolerance* $1.0994 \cdot 10^{-12}$ *used to determine numerical rank.*

$|R(305, 305)| \approx 9.7560 \cdot 10^{-13}$ is the first diagonal absolute value below the threshold $1.0994 \cdot 10^{-12}$. Then, R is assumed to have block partition

$$R = \begin{pmatrix} R_{[11]} & R_{[12]} \\ 0 & R_{[22]} \end{pmatrix}, \text{ where } R_{[11]} = R(1 : 304, 1 : 304), \quad \|R_{[22]}\|_F \leq 1.5610 \cdot 10^{-11}.$$

Setting $R_{[22]}$ to zero implicitly defines a rank 304 matrix $A + \delta A$ with $\|\delta A\|_F \leq 1.5610 \cdot 10^{-11}$. On the other hand, if we just count the number of $|R(i, i)|$'s above $1.0994 \cdot 10^{-12}$ we obtain exactly 466.

EXAMPLE 2.5. More sophisticated rank revealing factorizations [Chandrasekaran and Ipsen 1994], [Bischof and Quintana-Orti 1998] 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. From the source code of TOMS ♯ 782 one can conclude that erratic behavior can be expected in the initial phase (see xGEQPC.F, xGEQPW.F, xGEQPB.F). Our numerical experiments with xGEQPX have shown that it can catastrophically fail, and that a modification of the code is necessary.

3. EXPERIMENTAL DIAGNOSIS OF THE PROBLEM

Our first approach is experimental diagnosis. 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, and that we must go into the details of a concrete software implementation. We first focus to xGEQPF, because: *(i)* the code is simpler than its BLAS 3 implementation xGEQP3; *(ii)* it has less numerical uncertainties than xGEQP3. (The same problem occurs in xQRDC from LINPACK. It should be stressed that xGEQP3 and xGEQPF are not numerically equivalent.) To remove the unknown factor of ma-

```
1.                  DO 30 J = I+1, N
2.                      IF ( WORK( J ).NE.ZERO ) THEN
3.                          TEMP = ONE - ( ABS( A( I, J ) ) / WORK( J ) )**2
4.                          TEMP = MAX( TEMP, ZERO )
5.                          TEMP2 = ONE + 0.05*TEMP*( WORK( J ) / WORK( N+J ) )**2
6.                          IF( TEMP2.EQ.ONE ) THEN
7.                              IF( M-I.GT.0 ) THEN
8.                                  WORK( J ) = SNRM2( M-I, A( I+1, J ), 1 )
9.                                  WORK( N+J ) = WORK( J )
10.                             ELSE
11.                                 WORK( J ) = ZERO
12.                                 WORK( N+J ) = ZERO
13.                             END IF
14.                         ELSE
15.                             WORK( J ) = WORK( J )*SQRT( TEMP )
16.                         END IF
17.                     END IF
18.     30          CONTINUE
```

Table I. Critical part in the column norm update in SGEQPF.F. The input matrix A is M×N.

chine optimized libraries we use BLAS and LAPACK compiled from the source code from the NETLIB repository. (The first bad cases were discovered while using BLAS from the Intel's MKL library.) Our machine is Intel Pentium 4 based HP X2100 Workstation running under MS Windows/Suse Linux.[2]

In Table I we display a part of SGEQPF.F which is critical for column norms of submatrices in the factorization process. This well known updating strategy is the same as in the LINPACK procedure SQRDC. At the beginning of the I–th step of the factorization, for each column index J, WORK(J) contains the norm of A(I:M,J). On exit from the I–step, WORK(J) contains the norm of A(I+1:M,J) which is computed explicitly (line 8. or 11.) or by the updating formula (lines 3. and 15.), depending on the safety switch (lines 5. and 6.). Explicitly computed norms are kept in WORK(N+J) and used to estimate cancellations in the J–th column at later stages. Initially, WORK(J)=WORK(N+J) contains the norm of the J–th column of the input matrix.

What follows is a brief description of our course of action after facing this problem. The first attempt to resolve the problem is the obvious one – enforce explicit norm computation by SNRM2 in all cases. That is not satisfactory – we still do not know the 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 (lines 5. and 6.).

We returned the code to its original version in order to study the switching mechanism. An old fashion debugging practice called 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 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? But, we have encountered the same problem with the Intel Fortran compiler.

Since finding a bug in a program which returns a correct result is difficult, we

[2] The same problems were discovered using Athlon and Pentium Xeon processors.

removed the WRITE statements and restored the 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, we got a clue what might be going on.

We focus on the possibility that the optimizer keeps the variable TEMP2 in a long register (80 bit, 64 bit mantissa) which can change some equivalence relations we are used to take for granted. Note that the test "IF (TEMP2 .EQ. ONE) THEN" in line 6. was meant to be an equivalent way of asking

IF (0.05*TEMP*(WORK(J) / WORK(N+J))**2 .LT. EPS) THEN (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 from using 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 requires popping it from the stack. The FORTRAN code compiled with `-O -ffloat-store` has successfully factorized all our bad examples.

	g77 -O -S	g77 -O -ffloat-store -S
IF(TEMP2 .EQ. ONE) THEN ... ⟺		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 II. *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 a numerical program running correctly (and probably more slowly) is simply wrong. Extra steps would be required from the processor to prevent it from using extra precision in numerical software!

A 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 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. So, we use relation (3) instead of the line 6. in Table I.[3] (For xGEQP3, one has to go to xLAQPS.F and

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

xLAQP2.F to find the critical IF statements and make this change.) As a result of this change, the code compiled with `-O` runs fine in all our previous cases. The issue of improper implicit use of machine epsilon seems to be resolved.

Unfortunately, this is not the end of the story. If we turn the optimizer off, or if we have the optimizer on, but with `-O -ffloat-store`, the problems reappear, but with some new bad matrices. So, in this case the optimizer works in favor of the accuracy. Again, assembler code reveals that the key point is keeping TEMP in a long register. It was 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 causes substantially different results? How this looks like if one uses software debugging tools together with,[4] or without the optimizer? What if the problem is somewhere else, and everything we have observed thus far are just its various manifestations?

To identify the source of the problem and to develop reliable implementation of the factorization (1) we need numerical analysis as debugging tool.

4. UPDATING STRATEGY – NUMERICAL ANALYSIS

The experimentally observed erratic behavior of the LAPACK's xGEQP3 and xGEQPF routines is best understood through numerical analysis of the updating formula and its condition number. The analysis in §4.2 reveals that the sharpest decrease of the partial column norm $(\mathrm{WORK(J)}_{new}/\mathrm{WORK(J)}_{old})$ that can be detected by the current code is approximately the square root of the roundoff unit. That conclusion leads to a more appropriate threshold value in the updating formula, and the modified software computes correctly structured triangular factor in all cases mentioned in the previous section. However, we were not able to prove that this new threshold will successfully endure many updates of a particular column. For a provably reliable software, in §4.3 we introduce and analyze a new updating strategy with guaranteed accuracy of the column norms.

4.1 Preliminaries

Consider the k–th elimination step. Let $A^{(1)} = A = (\mathbf{a}_1, \ldots, \mathbf{a}_n) \in \mathbb{R}^{m\times n}$ and let

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

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

[4] Some GNU CC compilers have the debugging option `-g` by default with the `-O2` optimization.

H_k be Householder reflector such that

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

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 sharp norm reduction.

The following proposition shows that sharp norm reduction is related to the condition number of A_c, where A_c denotes the matrix obtained from A by scaling its nonzero columns to have unit Euclidean length. (cf. §6.2). In other words, cancellation indicates ill–conditioning, and the rank–revealing pivoting is at risk exactly when its performance is most needed.

PROPOSITION 4.1. *If* $\mathrm{rank}(A) = n$, *then for each* j, k, $\|A_c^\dagger\|_2 \geq \|\mathbf{a}_j^{(1)}\|_2 / \|\mathbf{z}_j^{(k)}\|_2$. *The matrix* A *is rank deficient if and only if* $\mathbf{z}_j^{(k)} = \mathbf{0}$ *for some* j, k.

For the proof note that $\|A_c^\dagger\|_2$ is independent of the permutation of the columns of A, and that any $\mathbf{z}_j^{(k)}$ can become pivot column by a suitable permutation.

REMARK 4.1. Even if all $\omega_j^{(k)}$ are computed by xNRM2, we cannot guarantee that the computed $\tilde{R}$ satisfies (1). Instead, we can only say that $|\tilde{R}_{ii}| \geq \tilde{\rho}_i \sqrt{\sum_{k=1}^{j} |\tilde{R}_{kj}|^2}$, $1 \leq i \leq j \leq n$, with parameters $\tilde{\rho}_i$ close to one. ⊠

For the purpose of error analysis, computed quantities will be denoted by tildes, e.g. $\tilde{\omega}_j^{(k)}$ is the computed floating point value of $\omega_j^{(k)}$. We use hats to denote perturbed quantities created in backward error analysis. Basic operations $+$, $-$, $\cdot$, $\div$, $\sqrt{\cdot}$ 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 $\mathfrak{F}$.

4.2 LAPACK (LINPACK) updating strategy

To compute $\omega_j^{(k+1)}$ we go back to (4) and (5). Orthogonality of H_k implies that in (5) the norm of $\mathbf{z}_j^{(k)}$ equals $\omega_j^{(k)} = \sqrt{(\beta_j^{(k+1)})^2 + \|\mathbf{z}_j^{(k+1)}\|_2^2}$, and thus

$$\omega_j^{(k+1)} = \sqrt{(\omega_j^{(k)})^2 - (\beta_j^{(k+1)})^2} = \omega_j^{(k)} \sqrt{1 - \left(\frac{\beta_j^{(k+1)}}{\omega_j^{(k)}} \right)^2}. \tag{6}$$

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

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 $\tilde{\omega}_j^{(k+1)}$ is computed explicitly as vector norm. In that case, the value of $\tilde{\omega}_j^{(k+1)}$ is also stored in the variable $\tilde{\nu}_j$, $\tilde{\nu}_j = \tilde{\omega}_j^{(k+1)}$. Thus, at any moment in the algorithm, $\tilde{\nu}_j$ contains the last explicitly computed partial column norm in

the j–th column. In Table I, WORK(N+J) contains $\tilde{\nu}_j$. Initially, the $\tilde{\nu}_j$'s are the computed column norms of A.

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

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

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

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

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

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

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 $\tilde{\omega}_j^{(k+1)}/\tilde{\nu}_j$ 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, $\tilde{\omega}_j^{(k+1)} = sqrt(\tilde{t}_j^{(k)}) \odot \tilde{\omega}_j^{(k)}$. 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 should we 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 $\|\tilde{\mathbf{z}}_j^{(k+1)}\|_2 = \sqrt{\|\hat{\mathbf{z}}_j^{(k)}\|_2^2 - (\tilde{\beta}_j^{(k+1)})^2}$.

REMARK 4.3. The transformation (10) can produce $\tilde{\mathbf{z}}_j^{(k+1)}$ with $\|\tilde{\mathbf{z}}_j^{(k+1)}\|_2/\tilde{\omega}_j^{(k+1)}$ 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 4.1. To illustrate how this strategy can fail, take (in MATLAB)

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

$\mathfrak{t} = \max\{1 \ominus (\tilde{\beta}_j^{(k+1)} \oslash \tilde{\omega}_j^{(k)}) ** 2, 0\} = 22\mathfrak{e}$, and the value of the control parameter is $0.05 \odot \mathfrak{t} \odot (\tilde{\omega}_j^{(k)} \oslash \tilde{\nu}_j) ** 2 = 1.1\mathfrak{e} > \mathfrak{e}$. The computed value of $\tilde{\omega}_j^{(k+1)}$ is $\sqrt{22\mathfrak{e}} \approx 6.99 \cdot 10^{-8}$, and the true norm of $\tilde{\mathbf{z}}_j^{(k+1)}$ is at most $2\mathfrak{e} \approx 4.44 \cdot 10^{-16}$. 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 in the case $\mathfrak{s} = 0$, even the zero vector can mistakenly be taken for pivot.

The following proposition summarizes the above considerations, and stresses the fact that in the LAPACK's updating formula the sharpest observable norm reduction factor is approximately $\sqrt{\mathfrak{e}}$.

PROPOSITION 4.2. *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* $sqrt(\tilde{t}_j^{(k)})$.

Next, we identify relevant condition number of one step of the updating formula (6). To anticipate the outcome, consider $f(x) = \sqrt{1-x^2}$, and its relative change

$$\frac{f(x+\delta x) - f(x)}{f(x)} \approx \frac{\delta x}{x} \frac{x f'(x)}{f(x)} = \frac{\delta x}{x} \frac{-x^2}{1-x^2}.$$

Computing $f(x)$ for x close to one is obviously sensitive, and the following proposition gives formal error analysis.

PROPOSITION 4.3. *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 4.2, with* $\tilde{t}_j^{(k)} > 0$. *Further, let in (10)* $\|\hat{\mathbf{z}}_j^{(k)}\|_2 = \|\tilde{\mathbf{z}}_j^{(k)}\|_2(1+\lambda_j^{(k)})$.

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*

$$1+\epsilon_j^{(k+1)} = (1+\epsilon_j^{(k)})(1+\alpha_j^{(k)})\sqrt{1-\left[\frac{(\tilde{\beta}_j^{(k+1)})^2}{\|\hat{\mathbf{z}}_j^{(k)}\|_2^2-(\tilde{\beta}_j^{(k+1)})^2}\right]\sigma_j^{(k)}}, \quad \textit{where}$$

$$1+\alpha_j^{(k)} = \frac{\sqrt{1+e_3}(1+e_4)(1+e_5)}{1+\lambda_j^{(k)}}, \quad \sigma_j^{(k)} = \frac{(1+\lambda_j^{(k)})^2}{(1+\epsilon_j^{(k)})^2}(1+e_1)^2(1+e_2)-1$$

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*

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

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

$$\hat{t}_j^{(k)} = \frac{1}{1+\sigma_j^{(k)}}\left(\frac{\tilde{t}_j^{(k)}}{1+e_3}+\sigma_j^{(k)}\right), \quad \tilde{t}_j^{(k)} = \max\{1 \ominus (\tilde{\beta}_j^{(k+1)} \oslash \tilde{\omega}_j^{(k)})**2, 0\}. \tag{11}$$

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. The rounding errors can conspire to bring down the updating strategy.

EXAMPLE 4.2. 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. ⊠

We should keep in mind that condition number is also computed quantity, and it has its own condition number. Further, the parameter $\tilde{t}_j^{(k)}$ itself depends on possibly inaccurate value $\tilde{\omega}_j^{(k)}$, with its own condition number. In a long run, a cumulative effect of cancellations is also to be expected, and it is not certain whether or not the *memorized* part in (7) will prevent inappropriate use of the updating formula. This all indicates that the threshold tolerance for $\tilde{t}_j^{(k)}$ should be lifted to the level where we can guarantee satisfactory lower bound for $\hat{t}_j^{(k)}$. Let us take in (7) $tol = \sqrt{\mathfrak{e}}$.

To unroll the recurrence from Proposition 4.3 after s consecutive updates is rather tedious. The sole purpose of this page is to show how hard it is to give any useful bound on error propagation through several updates by (6). Note that

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

where $\mathrel{\mathring{=}}$ 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 $\epsilon_j^{(k+1)} = \epsilon_j^{(k)}/\hat{t}_j^{(k)}$.

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

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

and we see (using (11)) 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 $\epsilon_j^{(k+1)} \mathrel{\mathring{=}} \epsilon_j^{(k)}/\hat{t}_j^{(k)}$. Thus, the first update after explicit norm computation is safe. (Cf. Example 4.2.)

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 (11)

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

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

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

and thus $\hat{t}_j^{(k+1)}\hat{t}_j^{(k)} \approx \tilde{t}_j^{(k+1)}\tilde{t}_j^{(k)}$. Hence, $\epsilon_j^{(k+2)} \mathrel{\mathring{=}} \epsilon_j^{(k)}/(\tilde{t}_j^{(k+1)}\tilde{t}_j^{(k)}) \approx O(n)\sqrt{\mathfrak{e}}$. 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

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

If $\prod_{i=0}^{s-1}\hat{t}_j^{(k+i)}$ and $\prod_{i=0}^{s-1}\tilde{t}_j^{(k+i)}$ are of the same order of magnitude, then by (11) $\hat{t}_j^{(k+s)} \approx \tilde{t}_j^{(k+s)}$, and $\tilde{\omega}_j^{(k+s+1)} = \sqrt{\tilde{t}_j^{(k+s)}} \odot \tilde{\omega}_j^{(k+s)}$ is sufficiently accurate.

$t = |\tilde{\beta}_j^{(k+1)}/\tilde{\omega}_j^{(k)}|$; $t = \max\{0, 1 - t^2\}$
$t_2 = t \cdot (\tilde{\omega}_j^{(k)}/\tilde{\nu}_j)^2$
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 III. Modification of the LINPACK/LAPACK column norm update.

These estimates are rather rough, messy and incomplete, but they indicate that the updating strategy with higher threshold value ($\sqrt{\mathfrak{e}}$) could survive several updates. Such a modified updating strategy, shown in Table III, has successfully passed all our tests, performed in single precision with $tol = \sqrt{\mathfrak{e}} \approx 2.44 \cdot 10^{-4}$. However, it has failed with slightly smaller threshold, $tol = 10^{-4}$.

4.3 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.

Note that in (4) we have $\|\mathbf{a}_j^{(k)}\|_2 = \|\mathbf{a}_j\|_2$ for all j, k. If we set $\xi_j^{(k)} = \|\mathbf{x}_j^{(k)}\|_2$, then $\xi_j^{(k+1)} = \sqrt{(\xi_j^{(k)})^2 + (\beta_j^{(k+1)})^2}$, and, using $\alpha_j = \|\mathbf{a}_j\|_2$,

$$\omega_j^{(k)} = \sqrt{\alpha_j^2 - (\xi_j^{(k)})^2}, \quad \omega_j^{(k+1)} = \sqrt{\alpha_j^2 - (\xi_j^{(k+1)})^2}. \tag{12}$$

PROPOSITION 4.4. *The formula (12) can be used to compute all $\tilde{\omega}_j^{(k)}$ with controlled forward error. It involves substraction of two quantities always known to guaranteed high relative accuracy.*

Proof: The backward stability of the QR factorization implies that $\|\tilde{\mathbf{a}}_j^{(k)}\|_2 = \alpha_j(1+\theta_j^{(k)})$, where the upper bound on $|\theta_j^{(k)}|$ depends on the details of the algorithm. In the 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 $\tilde{\alpha}_j = \alpha_j(1 + O(m)\mathfrak{e})$, and thus $\tilde{\alpha}_j = \|\tilde{\mathbf{a}}_j^{(k)}\|_2(1 + O(m)\mathfrak{e})/(1 + \theta_j^{(k)})$, for all j, k.

Hence, once we have computed the the $\tilde{\alpha}_j$'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, $\|\tilde{\mathbf{x}}_j^{(k)}\|_2 = \tilde{\xi}_j^{(k)}(1 + \zeta_j^{(k)})$, $|\zeta_j^{(k)}| \leq O(k\mathfrak{e})$.

Rewriting (12) to avoid underflow and overflow gives the following proposal for partial norm computation in k–th step:

$$\tilde{\omega}_j^{(k)} = \tilde{\alpha}_j^{(1)} \odot sqrt(max(1 \ominus (\tilde{\xi}_j^{(k)} \oslash \tilde{\alpha}_j^{(1)})^2, 0)). \tag{13}$$

To analyze (13), we need to know how accurately we can compute $\tilde{\omega}_j^{(k)}$, and how ac-

curately 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, under the assumption $\|\tilde{\mathbf{a}}_j^{(k)}\|_2 \neq \|\tilde{\mathbf{x}}_j^{(k)}\|_2$,

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

and that under the assumption $\tilde{\alpha}_j \neq \tilde{\xi}_j^{(k)}$ it holds

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

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

Finally, note that the condition numbers for this computation

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

can safely be compared against given tolerance. Only $\tilde{\chi}_j^{(k)}$ is accessible, and it will be below given tolerance tol ($\tilde{\chi}_j^{(k)} < tol$) if $\tilde{\xi}_j^{(k)}/\tilde{\alpha}_j < \sqrt{tol/(1+tol)}$. In that case $\chi_j^{(k)} < tol(1+\varsigma)/(1-\varsigma\cdot tol) \approx tol$, where $\|\tilde{\mathbf{x}}_j^{(k)}\|_2/\|\tilde{\mathbf{a}}_j^{(k)}\|_2 = (\tilde{\xi}_j^{(k)}/\tilde{\alpha}_j)(1+\varsigma)$, and $|\varsigma|$ is bounded by roundoff $\mathfrak{e}$ times a modest polynomial in m. The key point here is that we use original data ($\tilde{\alpha}_j$) and the values $\tilde{\xi}_j^{(k)}$ which are computed to high relative accuracy (without subtractions). Hence, we can always correctly predict and thus avoid conditions for catastrophic cancellations.

If $\tilde{\xi}_j^{(k)}$ and $\tilde{\alpha}_j$ are too close, then $\tilde{\omega}_j^{(k)}$ is computed explicitly as $\|\tilde{\mathbf{z}}_j^{(k)}\|_2$. In that case the updating formula is reset: $\tilde{\alpha}_j = \tilde{\omega}_j^{(k)}$, and $\tilde{\xi}_j^{(k)}$ is set to zero. ⊠

Assuming familiarity with the xGEQP3 code, we give in Table IV partial column norm update strategy, which uses both the new (12) and the old (6) formulas. Note that we allow at most one use 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 $1-\sqrt{\mathfrak{e}}$ (for $tol \approx 1/\sqrt{\mathfrak{e}}$), with some fine–tuning for speed. (Setting $\gamma_2 = 0$ switches off updating by the old formula.)

5. NEW SOFTWARE FOR BUSINGER–GOLUB PIVOTING

We now present our new codes for the Businger–Golub QR factorization – modifications of the xGEQP3 subroutine, with the 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*).

We test preliminary codes SGEQP3A (Table III) and SGEQP3Z (Table IV).

A collection of 1792 1000×800 test matrices is generated following [Drmač and Veselić 2007b, §3.3.1]. The matrices are divided into eight groups, each group 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 $\kappa_2(A_c)$

$\tilde{\xi}_j^{(k+1)} = \max\{\tilde{\xi}_j^{(k)}, \beta_j^{(k+1)}\}\sqrt{1 + (\min\{\tilde{\xi}_j^{(k)}, \beta_j^{(k+1)}\}/\max\{\tilde{\xi}_j^{(k)}, \beta_j^{(k+1)}\})^2}$
$t_0 = \tilde{\xi}_j^{(k+1)}/\tilde{\alpha}_j^{(1)}$
if ($t_0 < \gamma_1$) **then**
  $\tilde{\omega}_j^{(k+1)} = \tilde{\alpha}_j^{(1)}\sqrt{1 - t_0^2}$
else
  $t_1 = |\beta_j^{(k+1)}/\tilde{\omega}_j^{(k)}|$; $t_2 = \max\{0, 1 - t_1^2\}$
  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 IV. Critical part in the column norm update.

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 $1/\sqrt{\mathfrak{e}}$.

In Figure 6 and Figure 7 we show the timing results of SGEQP3A and SGEQP3Z in one test run. In SGEQP3Z we used only the new updating formula, i.e. $\gamma_2 = 0$. With the old formula added to updating strategy, the performance of SGEQP3Z can be only slightly improved, which does not seem to be worth having more complicated code. Clearly, depending on the matrix, there can be a drop in the performance because explicit computations of the column norms interfere with block structure of the algorithm.

The xGEQP3A code is a simple modification of xGEQP3, and, since it does not require any change in the specifications, it has been included as replacement for xGEQP3 in LAPACK 3.1.0. The xGEQP3Z routine needs n extra locations in the workspace. From the numerical point of view, the few percent increase in run time and this increase of the workspace are a negligible price for the provable numerical reliability of the software. We stress that this reliability assumes properly implemented floating–point arithmetic and the BLAS library.

Remark 5.1. Much to our disappointment, during thorough testing we found examples of failure of our software. Both SGEQP3A and SGEQP3Z failed. This time, the problem was not in the updating formula, but in the explicit norm computation. Using old fashioned debugging we traced the problem to the BLAS 1 function SNRM2 in the Intel's MKL library. Namely, for $x = (x_1, \ldots, x_n)$ and $n > 24$, the norm $\|x\|_2$ can be computed completely wrong by SNRM2 if the result is smaller than the square root of the underflow threshold. Sapienti sat.

6. IMPORTANCE OF PIVOTING

In this section we show how pivoting plays an important role in the forward error analysis (perturbation theory) of the QR factorization. In particular, pivoting contributes to a more accurate and better conditioned upper triangular factor. Our main motivation is reliable implementation of the pivoted QR factorization in the new Jacobi SVD algorithm [Drmač and Veselić 2007a; 2007b], but the issue is also relevant for least squares solvers and other applications of the QR factorization.

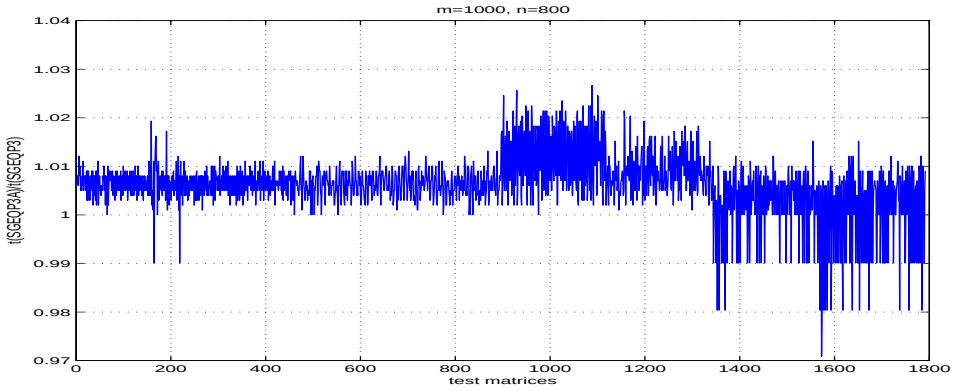

Fig. 6. *The relative timings*: time(SGEQP3A) / time(SGEQP3).

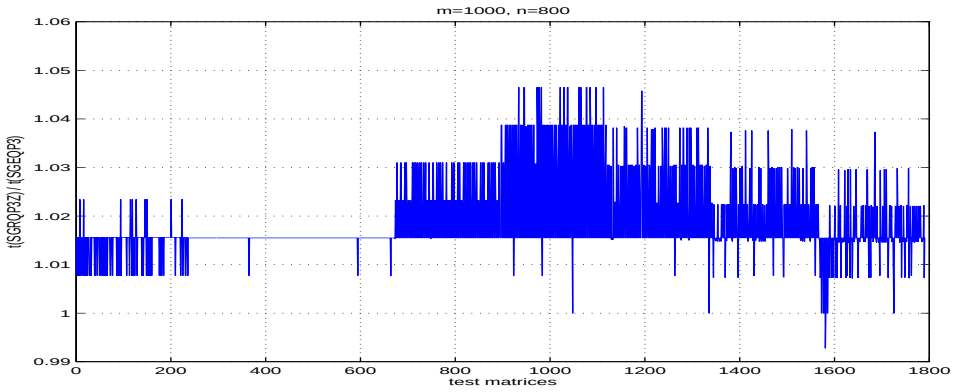

Fig. 7. *The relative timings*: time(SGEQP3Z) / time(SGEQP3).

6.1 Perturbation theory

In an 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. An exception is in [Cox and Higham 1998], where pivoting is the key for structured backward error of the QR factorization with the complete pivoting of Powell and Reid [1969]. Perturbation analysis of the factorization usually does not assume any pivoting, [Stewart 1977; 1993], [Sun 1991].

To specify the kind of perturbations of interest, we recall a backward stability result for the Householder and Givens QR factorization algorithms. (For more details see [Drmač 1994], [Higham 1996].)

THEOREM 6.1. *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

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

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 the 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 consideration.

Assume that both A and $\tilde{A} = A + \delta A$ have full column rank, and consider the Cholesky factorizations $\tilde{H} = \tilde{R}^T\tilde{R}$ and $H = R^TR = \tilde{H} + \delta\tilde{H}$. Following the ideas from [Drmač et al. 1994], we write $R^TR = \tilde{R}^T(I+E)\tilde{R}$, with

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

The matrix E expresses the size of $\delta\tilde{H}$ relative to $\tilde{H} = \tilde{R}^T\tilde{R}$, and F is a relative perturbation of $(A+\delta A)\tilde{P}$. If $\tilde{R}_c$ is the matrix obtained from $\tilde{R}$ by scaling its columns to have unit Euclidean norm, then Theorem 6.1 implies

$$\|E\|_F \leq 2\|(\hat{Q}\hat{Q}^T)F\|_F + \|F\|_2\|F\|_F, \;\; \|F\|_F \leq \frac{\sqrt{n}\eta_A}{1-\eta_A}\|\tilde{R}_c^{-1}\|_2. \tag{14}$$

Having the relative sizes (14) of the backward errors, we now estimate $R - \tilde{R}$.

THEOREM 6.2. *Let $(A+\delta A)\tilde{P} = \hat{Q}\tilde{R}$ be the factorization from Theorem 6.1, where the permutation matrix $\tilde{P}$ is computed following the Businger–Golub pivoting. Let $A\tilde{P} = QR$ be the exact QR factorization, and let $R = \tilde{R} + \delta\tilde{R}$. Further, let*

$$|\tilde{R}_{ii}| \geq \tilde{\rho}_i \sqrt{\sum_{k=1}^{j} |\tilde{R}_{kj}|^2}, \;\; 1 \leq i \leq j \leq n. \tag{15}$$

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

$$\|\delta\tilde{R}(:,i)\|_2 \;\leq\; \|\Gamma(:,1:i)\|_2\|\tilde{R}(:,i)\|_2, \;\; \delta\tilde{R}_{ii} = \Gamma_{ii}\tilde{R}_{ii}, \tag{16}$$

$$\|\delta\tilde{R}(i,:)\|_\infty \;\leq\; \frac{1}{\tilde{\rho}_i}\|\Gamma(i,:)\|_2\|\tilde{R}(i,:)\|_\infty, \tag{17}$$

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 (14). Further, $\hat{Q} - Q = Q\Gamma - F$.

Proof: Note that we can write $\tilde{R}^{-T}R^TR\tilde{R}^{-1} = I + E$, 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 (16) follows. To derive (17), note that

$$\|\delta\tilde{R}(i,:)\|_\infty = \max_{j=i:n}\left|\sum_{k=i}^{j}\Gamma_{ik}\tilde{R}_{kj}\right| \le \max_{j=i:n}\|\Gamma(i,:)\|_2\sqrt{\sum_{k=i}^{j}|\tilde{R}_{kj}|^2} \le \|\Gamma(i,:)\|_2\frac{1}{\tilde{\rho}_i}|\tilde{R}_{ii}|. \tag{18}$$

It remains to estimate Γ, or equivalently, the perturbation of the Cholesky factor of the identity under the perturbation $I \rightsquigarrow I + E$. We use the perturbation results from [Drmač et al. 1994], and the proof is completed. ⊠

Remark 6.1. Perturbation estimates are derived relative to $\tilde{R}$ (instead to R) because (15) 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 6.2. As in [Drmač et al. 1994], we note that the Theorem 6.2 contains an element–wise bound: $\delta\tilde{R}_{ij} = \sum_{k=i}^{j}\Gamma_{ik}\tilde{R}_{kj}$ implies $|\delta\tilde{R}_{ij}| \le \|\Gamma(i,:)\|_2\sqrt{\sum_{k=i}^{j}|\tilde{R}_{kj}|^2}$. Note in particular that $|\delta\tilde{R}_{ii}| \le |\Gamma_{ii}||\tilde{R}_{ii}|$, which additionally stresses the importance of having dominant elements along the diagonal.

Remark 6.3. The value of $\|\tilde{R}_c^{-1}\|_2$ can be estimated by an $O(n^2)$ condition estimator and then Theorem 6.2 and (14) 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

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

Example 6.1. Let $A \equiv I_2R = \begin{pmatrix}\epsilon & \epsilon\\ 0 & \frac{1}{\epsilon}\end{pmatrix}$, and $\tilde{A} = A + \delta A = \begin{pmatrix}\epsilon & \epsilon\\ \epsilon^{\mathbf{2}} & \frac{1}{\epsilon}\end{pmatrix}$. 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 (16) holds, but the element R_{12} and the whole first row of the upper triangular factor are substantially changed. Furthermore, 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

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

$$\approx \begin{pmatrix}\epsilon^2 & -1\\ 1 & \epsilon^2\end{pmatrix}\begin{pmatrix}\frac{1}{\epsilon} & \epsilon^3\\ 0 & -\epsilon\end{pmatrix};$$

$$\begin{pmatrix} \epsilon & \epsilon \\ \epsilon^2 & \frac{1}{\epsilon} \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \frac{1}{\sqrt{1+\epsilon^4}} \begin{pmatrix} \epsilon^2 & -1 \\ 1 & \epsilon^2 \end{pmatrix} \begin{pmatrix} \frac{1}{\epsilon} + \epsilon^3 & \epsilon^3 + \epsilon^2 \\ \frac{\epsilon}{\sqrt{1+\epsilon^4}} & \frac{}{\sqrt{1+\epsilon^4}} \\ 0 & \frac{-\epsilon + \epsilon^4}{\sqrt{1+\epsilon^4}} \end{pmatrix}$$

$$\approx \begin{pmatrix} \epsilon^2 & -1 \\ 1 & \epsilon^2 \end{pmatrix} \begin{pmatrix} \frac{1}{\epsilon} & \epsilon^3 + \epsilon^2 \\ 0 & -\epsilon \end{pmatrix}.$$

In this case, the perturbation of R is column–wise (16) and row–wise ((17), Remark 6.2) small, although the relative change in R_{12} is arbitrarily bad.

6.2 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 §6.1 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, the 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 ([Drmač 1994; 1999]) gives an estimate.

PROPOSITION 6.1. *Let $AP = QR$, where $|R_{ii}| \geq \sqrt{\sum_{k=i}^{j} |R_{kj}|^2}$ for all $1 \leq i \leq j \leq n$. Then $\| \, |R_r^{-1}| \, \|_2 \leq \sqrt{n} \| \, |R_c^{-1}| \, \|_2$ 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 $O(n)$ for any A.*

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 preconditioned Jacobi SVD algorithm [Drmač and Veselić 2007a; 2007b], and it is also relevant in least squares and linear system solvers. Just to illustrate, consider solving $Rx = b$ and $(R + \delta R)\tilde{x} = b$. The perturbed solution is $\tilde{x} = (I + R^{-1}\delta R)^{-1} x$, 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), and, at the same time, ensures that it is computed with row–wise small errors (cf. Theorem 6.2 and Example 6.1).

With our new implementation, these important properties of the Businger–Golub QR factorization can be used with confidence in numerical software.

ACKNOWLEDGMENTS

The authors thank Jim Demmel and Jason Riedy (Berkeley), Julien Langou (Denver), Nick Higham (Manchester) and Krešimir Veselić (Hagen) for many fruitful discussions. We are also indebted to the anonymous referees for their constructive criticism and substantial comments and suggestions.

REFERENCES

Bischof, C. H. and Quintana-Orti, G. 1998. Computing rank–revealing QR factorizations of dense matrices. *ACM Transactions on Mathematical Software 24,* 2, 226–253.

Businger, P. A. and Golub, G. H. 1965. Linear least squares solutions by Householder transformations. *Numerische Mathematik 7*, 269–276.

Chandrasekaran, S. and Ipsen, I. C. F. 1994. On rank–revealing factorizations. *SIAM Journal on Matrix Analisis and Applications 15,* 2, 592–622.

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

Drmač, Z. 1994. Computing the singular and the generalized singular values. Ph.D. thesis, Lehrgebiet Mathematische Physik, Fernuniversität Hagen.

Drmač, Z. 1999. A posteriori computation of the singular vectors in a preconditioned Jacobi SVD algorithm. *IMA Journal of Numerical Analysis 19*, 191–213.

Drmač, Z., Omladič, M., and Veselić, K. 1994. On the perturbation of the Cholesky factorization. *SIAM Journal on Matrix Analisis and Applications 15,* 4, 1319–1332.

Drmač, Z. and Veselić, K. 2007a. New fast and accurate Jacobi SVD algorithm: I. *SIAM Journal on Matrix Analisis and Applications*. LAPACK Working Note 169.

Drmač, Z. and Veselić, K. 2007b. New fast and accurate Jacobi SVD algorithm: II. *SIAM Journal on Matrix Analisis and Applications*. LAPACK Working Note 170.

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

Kahan, W. 1966. Numerical linear algebra. *Canadian Mathematical Bulletin 9,* 6, 757–801.

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

Quintana-Orti, G., Sun, X., and Bischof, C. H. 1998. A BLAS 3 version of the QR factorization with column pivoting. *SIAM Journal on Scientific Computing 19,* 5, 1486–1494.

Stewart, G. W. 1977. Perturbation bounds for the QR decomposition of a matrix. *SIAM Journal on Numerical Analysis 14,* 3, 509–518.

Stewart, G. W. 1993. On the perturbation of LU, Cholesky, and QR factorizations. *SIAM Journal on Matrix Analisis and Applications 14,* 4, 1141–1145.

Sun, J.-G. 1991. Perturbation bounds for the Cholesky and QR factorizations. *BIT 31*, 341–352.

Veselić, K. and Hari, V. 1989. A note on a one–sided Jacobi algorithm. *Numerische Mathematik 56*, 627–633.

Zha, H. 1997. Singular values of a classical matrix. *American Mathematical Monthly 104*, 172–173.

Received November 12 2006; revised June 2007; accepted ? 2007.