I. Basic decompositions:
QR factorization:
New implementation of a rank revealing QR – floating point issues and applications
A case study

Zlatko Drmač

Virginia Tech
Department of Mathematics

September 4. 2007
Introduction

LAPACK release 3.1

Setting the scene: Jacobi SVD

QR+Businger–Golub CP

Analysis

Perturbation estimates for the QRF
Introduction: goals

LAPACK release 3.1

Setting the scene: Jacobi SVD

QR+Businger–Golub CP

Analysis

Perturbation estimates for the QRF
Introduction: goals

We want to compute matrix singular values (SVD, PSVD, QSVD) and eigenvalues of positive definite matrices (pencils $HM - \lambda I$, $H - \lambda M$) to high relative accuracy, if such accuracy is warranted by the data. This means that for computed $\tilde{\lambda}_i \approx \lambda_i$

$$|\tilde{\lambda}_i - \lambda_i| \leq C\varepsilon|\lambda_i|$$

Two basic factorizations used in the eigenvalue and SVD methods are

- QR factorization, $A = Q \begin{pmatrix} R \\ O \end{pmatrix}$, or pivoted $AP = Q \begin{pmatrix} R \\ O \end{pmatrix}$;
- Cholesky factorization, $H = LL^*$, or pivoted $PHP = LL^*$.

Our goal is to understand the details of their floating–point computations. Both factorizations are well–known, with established perturbation estimates and numerical software is available (LINPACK, LAPACK). We start with the QRF.
Introduction

Introduction: goals

LAPACK release 3.1

Setting the scene: Jacobi SVD

QR+Businger–Golub CP

Analysis

Perturbation estimates for the QRF
http://www.netlib.org/lapack/lapack-3.1.0.changes

================
== LAPACK 3.1 ==
================

* LAPACK 3.1: What’s new
* Contributor list
* Developer list
* Thanks
* LAPACK subroutine interface policy
* More details

================
== LAPACK 3.1: What’s new ==
================
LAPACK 3.1: What’s new

5) New partial column norm updating strategy for QR factorization with pivoting.

Comments: This fixes a subtle numerical bug dating back to LINPACK that can give completely wrong results.
Changes:
M SRC/c,d,s,zgeqpf.f
M SRC/c,d,s,zlaqp2.f
M SRC/c,d,s,zlaqps.f
Reference:
1. Z. Drmač and Z. Bujanović, LAPACK Working Note 176, On the failure of rank revealing QR factorization software - a case study, June 2006. (submitted to ACM Transaction on Mathematical Software)
Introduction: goals

LAPACK release 3.1

Setting the scene: Jacobi SVD

QRCP as preconditioner

Businger–Golub pivoting

Jacobi++

Numerical Poka–Yoke

Kill the code

Surprise :(}

QR+Businger–Golub CP

Analysis

Perturbation estimates for the QRF
QRCP as preconditioner

\[
AP = Q \begin{pmatrix} R \\ 0 \end{pmatrix}; \quad A = \begin{pmatrix} \ast & \ast & \ast \\ 0 & \ast & \ast \\ \ast & 0 & \ast \end{pmatrix}, \quad R = \begin{pmatrix} \ast \end{pmatrix}
\]

\[
\text{SVD}(R) \leftrightarrow \text{SVD}(A), \quad \text{SVD}(R^T) \leftrightarrow \text{SVD}(A)
\]

- \( P^T A^T A P = R^T R \)
  - \( X = R \), diagonalize \( X^T X = R^T R \)
  - \( X \underbrace{J_1 J_2 \cdots J_\infty}_{V_x} = U_x \Sigma \implies R \equiv X = U_x \Sigma V_x^T \)

- \( AA^T = Q \begin{pmatrix} RR^T & 0 \\ 0 & 0 \end{pmatrix} Q^T \)
  - \( X = R^T \), diagonalize \( X^T X = RR^T \)
  - \( X \underbrace{J_1 J_2 \cdots J_\infty}_{V_x} = U_x \Sigma \implies R \equiv X^T = V_x \Sigma U_x^T \)
QRCP as preconditioner

\[ AP = Q \begin{pmatrix} R \\ 0 \end{pmatrix}; \quad A = \begin{pmatrix} \ast \\ \ast \\ \ast \end{pmatrix}, \quad R = \begin{pmatrix} \ast \end{pmatrix} \]

SVD(\(R\)) \leftrightarrow SVD(\(A\)), SVD(\(R^T\)) \leftrightarrow SVD(\(A\))

- \(P^T A^T A P = R^T R\)
  - \(X = R\), diagonalize \(X^T X = R^T R\);
  - \(X_{J_1 J_2 \cdots J_\infty} = U_x \Sigma \implies R \equiv X = U_x \Sigma V_x^T\)
  
- \(AA^T = Q \begin{pmatrix} RR^T & 0 \\ 0 & 0 \end{pmatrix} Q^T\)
  - \(X = R^T\), diagonalize \(X^T X = RR^T\);
  - \(X_{J_1 J_2 \cdots J_\infty} = U_x \Sigma \implies R \equiv X^T = V_x \Sigma U_x^T\)
Businger–Golub pivoting

\[ AP = Q \begin{pmatrix} R \\ 0 \end{pmatrix}, \quad R = \begin{pmatrix} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ 0 & 0 & \star & \bullet & \bullet & \bullet & \bullet \\ 0 & 0 & 0 & \bullet & \bullet & \bullet & \bullet \\ 0 & 0 & 0 & 0 & \bullet & \bullet & \bullet \\ 0 & 0 & 0 & 0 & 0 & \star \end{pmatrix} \]

\[ |R_{ii}| \geq \sqrt{\sum_{k=i}^{j} |R_{kj}|^2}, \text{ for all } 1 \leq i \leq j \leq n. \]

\[ |R_{11}| \geq |R_{22}| \geq \cdots \geq |R_{nn}| \]

★ may not be rank revealing
QRCP as preconditioner

\[
AP = Q \begin{pmatrix} R \\ 0 \end{pmatrix}; \quad P^T A^T A P = R^T R
\]
QRCP as preconditioner

\[ AP = Q \begin{pmatrix} R \\ 0 \end{pmatrix}; \quad AA^T = Q \begin{pmatrix} R R^T & 0 \\ 0 & 0 \end{pmatrix} Q^T \]
QRCP as preconditioner

\[ R^T P_1 = Q_1 R_1; \quad R^T R = Q_1 R_1 R_1^T Q_1^T \]
QR as preconditioner

spy(P1)
**QRCP as preconditioner**

Let \( AP = Q \begin{pmatrix} R \\ 0 \end{pmatrix} \). Let

\[
A_c = A \cdot \text{diag}(\frac{1}{\|A(:, 1)\|_2}, \ldots, \frac{1}{\|A(:, n)\|_2})
\]

In the same way, define \( R_c \). Let

\[
R_r = \text{diag}(\frac{1}{\|R(1, :)\|_2}, \ldots, \frac{1}{\|R(n, :)\|_2}) \cdot R.
\]

\[
R_r = \begin{pmatrix} \rightarrow & \rightarrow & \rightarrow \\ \downarrow & \downarrow & \downarrow \\ 0 & \rightarrow & \rightarrow \\ 0 & 0 & \rightarrow \end{pmatrix}, \quad R_c = \begin{pmatrix} 0 & \downarrow & \downarrow & \downarrow \\ 0 & \downarrow & \downarrow & \downarrow \\ 0 & 0 & \downarrow \end{pmatrix}
\]

Let \( A = \text{Hilbert}(100) \). Then \( \kappa_2(A_c) = \kappa_2(R_c) \approx 2.26 \cdot 10^{19} \), BUT \( \kappa_2(R_r) \approx 48.31 \).

Repeat with \( A \rightarrow R^T \) with \( P = I \) to get new \( \kappa_2(R_r) \approx 3.22 \).

No need for SGEQP3. Use faster SGEQRF. Window pivoting possible in the second factorization.
Jacobi ++ (Drmač, Veselić 06.)

• $$(\Pi A)P = Q \begin{bmatrix} R \\ 0 \end{bmatrix}; \rho = \text{rank}(R) (A = D_1 BD_2)$$

• $$R(1 : \rho, 1 : n)^T = Q_1 \begin{bmatrix} R_1 \\ 0 \end{bmatrix};$$

• $$X = R_1^T = \begin{bmatrix} \text{■} & \text{0} \\ \text{0} & \text{■} \end{bmatrix}; X^T X - \xi I \text{ quasi-definite}$$

• $$X_\infty \equiv U_x \Sigma = X \langle J_1 J_2 \cdots J_\infty \rangle$$

• $$V_x = R_1^{-T}(X_\infty)$$

• $$U = \Pi^T Q \begin{bmatrix} U_x & 0 \\ 0 & I_{m-\rho} \end{bmatrix}; V = PQ_1 \begin{bmatrix} V_x & 0 \\ 0 & I_{n-\rho} \end{bmatrix}$$

• if $$\rho = n$$, $$Q_1 V_x = R^{-1} X_\infty$$
Jacobi ++ (Drmač, Veselić 06.)

- \((\Pi A)P = Q \begin{pmatrix} R \\ 0 \end{pmatrix} ; \rho = \text{rank}(R) (A = D_1 B D_2)\)
  - \(R(1 : \rho, 1 : n)^T = Q_1 \begin{pmatrix} R_1 \\ 0 \end{pmatrix} ; \)
  - \(X = R_1^T = \begin{pmatrix} \text{■} & 0 \\ \text{■} & \text{■} \end{pmatrix} ; X^T X - \xi I \text{ quasi–definite} \)
  - \(X_\infty \equiv U_x \Sigma = X\langle J_1 J_2 \cdots J_\infty \rangle \)
  - \(V_x = R_1^{-T}(X_\infty) \)
  - \(U = \Pi^T Q \begin{pmatrix} U_x & 0 \\ 0 & I_{m-\rho} \end{pmatrix} ; V = PQ_1 \begin{pmatrix} V_x & 0 \\ 0 & I_{n-\rho} \end{pmatrix} \)
  - if \(\rho = n, Q_1 V_x = R^{-1} X_\infty \)
Jacobi ++ (Drmač, Veselić 06.)

\[(\Pi A)P = Q \begin{pmatrix} R \\ 0 \end{pmatrix}; \rho = \text{rank}(R) (A = D_1BD_2)\]

- \(R(1 : \rho, 1 : n)^T = Q_1 \begin{pmatrix} R_1 \\ 0 \end{pmatrix};\)
- \(X = R_1^T = \begin{pmatrix} \text{\#} & 0 \\ \text{\#} \end{pmatrix}; X^TX - \xi I \text{ quasi--definite}\)
- \(X_\infty \equiv U_x\Sigma = X \langle J_1 J_2 \cdots J_\infty \rangle_{V_x}\)
- \(V_x = R_1^{-T}(X_\infty)\)
- \(U = \Pi^T Q \begin{pmatrix} U_x & 0 \\ 0 & I_{m-\rho} \end{pmatrix}; V = PQ_1 \begin{pmatrix} V_x & 0 \\ 0 & I_{n-\rho} \end{pmatrix}\)
- if \(\rho = n\), \(Q_1 V_x = R^{-1}X_\infty\)
Jacobi ++ (Drmač, Veselić 06.)

- $(\Pi A) P = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$; $\rho = \text{rank}(R)$ ($A = D_1 B D_2$)

- $R(1 : \rho, 1 : n)^T = Q_1 \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$;

- $X = R_1^T = \begin{pmatrix} \mathbf{v}^T & 0 \\ \mathbf{w}^T & \mathbf{x} \end{pmatrix}$; $X^T X - \xi I$ quasi–definite

- $X_\infty \equiv U_x \Sigma = X \langle J_1, J_2, \ldots, J_\infty \rangle_{V_x}$

- $V_x = R_1^{-T}(X_\infty)$

- $U = \Pi^T Q \begin{pmatrix} U_x & 0 \\ 0 & I_{m-\rho} \end{pmatrix}$; $V = P Q_1 \begin{pmatrix} V_x & 0 \\ 0 & I_{n-\rho} \end{pmatrix}$

- if $\rho = n$, $Q_1 V_x = R^{-1} X_\infty$
Jacobi ++ (Drmač, Veselić 06.)

- \((\Pi A)P = Q \begin{pmatrix} R \\ 0 \end{pmatrix} \); \(\rho = \text{rank}(R) \) \((A = D_1 BD_2)\)
- \(R(1 : \rho, 1 : n)^T = Q_1 \begin{pmatrix} R_1 \\ 0 \end{pmatrix} \);
- \(X = R_1^T = \begin{pmatrix} \text{\textcolor{red}{\textbullet}} & 0 \\ \text{\textcolor{yellow}{\textbullet}} & \text{\textcolor{pink}{\textbullet}} \end{pmatrix} \); \(X^T X - \xi I\) quasi–definite
  - \(X_\infty \equiv U_x \Sigma = X \langle J_1 J_2 \cdots J_\infty \rangle_{V_x} \)
  - \(V_x = R_{1}^{-T}(X_\infty)\)
- \(U = \Pi^T Q \begin{pmatrix} U_x & 0 \\ 0 & I_{m-\rho} \end{pmatrix} \); \(V = PQ_1 \begin{pmatrix} V_x & 0 \\ 0 & I_{n-\rho} \end{pmatrix} \)
- if \(\rho = n\), \(Q_1 V_x = R^{-1} X_\infty\)
Jacobi ++ (Drmač, Veselić 06.)

- \((\prod A)P = Q \begin{pmatrix} R \\ 0 \end{pmatrix} ; \rho = \text{rank}(R) (A = D_1BD_2)\)

- \(R(1 : \rho, 1 : n)^T = Q_1 \begin{pmatrix} R_1 \\ 0 \end{pmatrix} ;\)

- \(X = R_1^T = \begin{pmatrix} \text{\ding{51}} & 0 \\ \text{\ding{52}} & \text{\ding{53}} \end{pmatrix} ; X^T X - \xi I \text{ quasi–definite}\)

- \(X_\infty \equiv U_\infty \Sigma = X \langle J_1 J_2 \cdots J_\infty \rangle \)

- \(V_X = R_1^{-T} (X_\infty)\)

- \(U = \prod^T Q \begin{pmatrix} U_\infty & 0 \\ 0 & I_{m-\rho} \end{pmatrix} ; V = P Q_1 \begin{pmatrix} V_X & 0 \\ 0 & I_{n-\rho} \end{pmatrix}\)

- if \(\rho = n, Q_1 V_X = R^{-1} X_\infty\)
Jacobi ++ (Drmač, Veselić 06.)

- \((\prod A)P = Q \begin{pmatrix} R \\ 0 \end{pmatrix} ; \rho = \text{rank}(R) (A = D_1 BD_2)\)

- \(R(1 : \rho, 1 : n)^T = Q_1 \begin{pmatrix} R_1 \\ 0 \end{pmatrix} ;\)

- \(X = R_1^T = \begin{pmatrix} \text{[ ]} & 0 \\ \text{[ ]} & \text{[ ]} \end{pmatrix} ; X^T X - \xi I \text{ quasi-definite}\)

- \(X_\infty \equiv U_x \Sigma = X \langle J_1 J_2 \cdots J_\infty \rangle _{V_x}\)

- \(V_x = R_1^{-T}(X_\infty)\)

- \(U = \prod^T Q \begin{pmatrix} U_x \\ 0 \\ I_{m-\rho} \end{pmatrix} ; V = PQ_1 \begin{pmatrix} V_x \\ 0 \\ I_{n-\rho} \end{pmatrix} \)

- if \(\rho = n, Q_1 V_x = R^{-1} X_\infty \)
Jacobi ++ (Drmač, Veselić 06.)

- \((\Pi A)P = Q \begin{pmatrix} R \\ 0 \end{pmatrix} ; \rho = \text{rank}(R) (A = D_1BD_2)\)

- \(R(1 : \rho, 1 : n)^T = Q_1 \begin{pmatrix} R_1 \\ 0 \end{pmatrix} ; \)

- \(X = R_1^T = \begin{pmatrix} \text{■} & \text{0} \\ \text{□} & \text{□} \end{pmatrix} ; X^T X - \xi I \text{ quasi–definite} \)

- \(X_\infty \equiv U_x \Sigma = X \langle J_1 J_2 \cdots J_{\infty} \rangle_{V_x} \)

- \(V_x = R_{1}^{-T}(X_\infty) \)

- \(U = \Pi^T Q \begin{pmatrix} U_x & 0 \\ 0 & I_{m-\rho} \end{pmatrix} ; V = PQ_1 \begin{pmatrix} V_x & 0 \\ 0 & I_{n-\rho} \end{pmatrix} \)

- \(\text{if } \rho = n, Q_1 V_x = R^{-1} X_\infty \)
Jacobi ++ (Drmač, Veselić 06.)

- \((\Pi A)P = Q \begin{pmatrix} R \\ 0 \end{pmatrix} \); \(\rho = \text{rank}(R)\) \((A = D_1BD_2)\)
- \(R(1 : \rho, 1 : n)^T = Q_1 \begin{pmatrix} R_1 \\ 0 \end{pmatrix} \);
- \(X = R_1^T = \begin{pmatrix} \text{black} & 0 \\ \text{yellow} & \text{pink} \end{pmatrix} \); \(X^T X - \xi I\) quasi–definite
  - \(X_\infty \equiv U_x \Sigma = X \langle J_1J_2 \cdots J_\infty \rangle_{V_x}\)
- \(V_x = R_1^{-T}(X_\infty)\)
- \(U = \Pi^T Q \begin{pmatrix} U_x & 0 \\ 0 & I_{m-\rho} \end{pmatrix} \); \(V = P Q_1 \begin{pmatrix} V_x & 0 \\ 0 & I_{n-\rho} \end{pmatrix} \)
- if \(\rho = n\), \(Q_1 V_x = R^{-1} X_\infty\)
Numerical Poka–Yoke devices

In our SVD code we use **numerical poka–yoke devices**: 

- Useful data is collected with negligible overhead: sizes of the pivots, sizes of the rotation angles, patterns of skipped Jacobi rotations, etc.

- Based on the collected data, and using theory, driver routine obtains various condition number(s) estimates and uses them in the decision process/control switches (some switches with memory). In this way, certain ill–conditioned (causing loss of accuracy) or unefficient (with respect to convergence) operations can be predicted and avoided.

- On exit, the driver returns the trace of the actual branch in the algorithm, taken in a particular run. Useful for post mortem analysis.
During the final testing of our code, we performed a 'kill the code' stress test.

- Large scale testing
  - full range, $\sigma_i \in (\text{small}, \text{BIG})$
  - two levels of accuracy
    - $|\tilde{\sigma}_i - \sigma_i| \leq K \cdot \epsilon \cdot \|A\|_2$ for all $i$
    - $|\tilde{\sigma}_i - \sigma_i| \leq C \cdot \epsilon \cdot \sigma_i$ for all $i$
  - theoretical bounds attained; all test passed

Then we started looking for trouble.

And we found trouble. For some test matrices, the traces of the runs were completely unexpected and the results were not as predicted by the theory. This prompted detailed debugging and the problem is traced to an unexpected result of the xGEQP3 routine, which performs the QR factorization with the Businger–Golub column pivoting.
Kill the code

During the final testing of our code, we performed a ’kill the code’ stress test.

- Large scale testing
  - full range, $\sigma_i \in (\text{small}, \text{BIG})$
  - two levels of accuracy
    - $|\tilde{\sigma}_i - \sigma_i| \leq K \cdot \epsilon \cdot \|A\|_2$ for all $i$
    - $|\tilde{\sigma}_i - \sigma_i| \leq C \cdot \epsilon \cdot \sigma_i$ for all $i$
  - theoretical bounds attained; all test passed

- Then we started looking for trouble.

And we found trouble. For some test matrices, the traces of the runs were completely unexpected and the results were not as predicted by the theory. This prompted detailed debugging and the problem is traced to an unexpected result of the xGEQP3 routine, which performs the QR factorization with the Businger–Golub column pivoting.
Kill the code

During the final testing of our code, we performed a 'kill the code' stress test.

- Large scale testing
  - full range, $\sigma_i \in (\text{small, BIG})$
  - two levels of accuracy
    - $|\tilde{\sigma}_i - \sigma_i| \leq K \cdot \epsilon \cdot \|A\|_2$ for all $i$
    - $|\tilde{\sigma}_i - \sigma_i| \leq C \cdot \epsilon \cdot \sigma_i$ for all $i$
  - theoretical bounds attained; all test passed
- Then we started looking for trouble.
  And we found trouble. For some test matrices, the traces of the runs were completely unexpected and the results were not as predicted by the theory. This prompted detailed debugging and the problem is traced to an unexpected result of the xGEQP3 routine, which performs the QR factorization with the Businger–Golub column pivoting.
Kill the code

During the final testing of our code, we performed a ‘kill the code’ stress test.

- Large scale testing
  - full range, $\sigma_i \in (\text{small}, \text{BIG})$
  - two levels of accuracy
    - $|\tilde{\sigma}_i - \sigma_i| \leq K \cdot \epsilon \cdot \|A\|_2$ for all $i$
    - $|\tilde{\sigma}_i - \sigma_i| \leq C \cdot \epsilon \cdot \sigma_i$ for all $i$
- theoretical bounds attained; all test passed
- Then we started looking for trouble.

And we found trouble. For some test matrices, the traces of the runs were completely unexpected and the results were not as predicted by the theory. This prompted detailed debugging and the problem is traced to an unexpected result of the xGEQP3 routine, which performs the QR factorization with the Businger–Golub column pivoting.
\[ \xi = \text{Kahan}(n, c); \quad A = \xi + \xi^T. \quad RR^T. \quad \text{The ill–conditioned} \]

"tower" \text{is a result of wrong pivot choices.}
Introduction

LAPACK release 3.1

Setting the scene: Jacobi SVD
QRCP as preconditioner
Businger–Golub pivoting
Jacobi++
Numerical Poka–Y oke
Kill the code
Surprise :(

QR+Businger–Golub CP

Analysis
Perturbation estimates for the QRF

\[ \mathcal{K} = \text{Kahan}(n, c); \ A = \mathcal{K} + \mathcal{K}^T. \ RR^T. \]
\[ \mathcal{K} = \text{Kahan}(n, c \cdot (1 + \epsilon)); A = \mathcal{K} + \mathcal{K}^T \cdot RR^T. \]
Introduction

LAPACK release 3.1

Setting the scene: Jacobi SVD

QRCP software – case study

Zlatko Drmač

Examples

Consequences

Analysis

Perturbation estimates for the QRF

Businger–Golub pivoting

\[ AP = Q \begin{pmatrix} R \\ 0 \end{pmatrix}, \quad R = \begin{pmatrix} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ 0 & 0 & \bullet & \bullet & \bullet & \bullet & \bullet \\ 0 & 0 & 0 & \bullet & \bullet & \bullet & \bullet \\ 0 & 0 & 0 & 0 & \bullet & \bullet & \bullet \\ 0 & 0 & 0 & 0 & 0 & \bullet & \bullet \end{pmatrix} \]

\[ |R_{ij}| \geq \sqrt{\sum_{k=i}^{j} |R_{kj}|^2}, \quad \text{for all } 1 \leq i \leq j \leq n. \]

\[ |R_{11}| \geq |R_{22}| \geq \cdots \geq |R_{nn}| \]

★ may not be rank revealing but it must be guaranteed by the software (xGEQPF, xGEQPF3)
Ap = Q \left( \begin{array}{c} \mathbf{R} \\ \mathbf{0} \end{array} \right), \quad \mathbf{R} = \begin{pmatrix}
\mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1}
\end{pmatrix}

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

\| R_{11} \| \geq | R_{22} | \geq \cdots \geq | R_{nn} |

★ may not be rank revealing but it must be guaranteed by the software (xGEQPF, xGEQP3)
Businger–Golub pivoting

\[ AP = Q \begin{pmatrix} R \end{pmatrix}, \quad R = \begin{pmatrix} \star & \star & \star & \star & \star & \star \\ 0 & \star & \star & \star & \star & \star \\ 0 & 0 & \bullet & \bullet & \bullet & \bullet \\ 0 & 0 & 0 & \bullet & \bullet & \bullet \\ 0 & 0 & 0 & 0 & \bullet & \bullet \\ 0 & 0 & 0 & 0 & 0 & \bullet \end{pmatrix} \]

\[ |R_{ii}| \geq \sqrt{\sum_{k=i}^{j} |R_{kj}|^2}, \quad \text{for all } 1 \leq i \leq j \leq n. \]

\[ |R_{11}| \geq |R_{22}| \geq \cdots \geq |R_{nn}| \]

\[ \star \] may not be rank revealing but it must be guaranteed by the software (xGEQPF, xGEQPF3)
Examples of failure of $\star$

$$|R_{ii}|, \max_{j \geq i} \sqrt{\sum_{k=i}^{j} |R_{kj}|^2}$$
Examples of failure of ⭐

\[ |R_{ii}|, \max_{j \geq i} \sqrt{\sum_{k=i}^{j} |R_{kj}|^2} \]
Examples of failure of ★

\[ |R_{ii}|, \max_{j \geq i} \sqrt{\sum_{k=i}^{j} |R_{kj}|^2} \]
Consequences

\[ \|Ax - d\|_2 \rightarrow \min; \quad x = A \backslash d \]

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

\[
\text{rank}(A, 1.0994e-12) \text{ returns 466}
\]
Consequences

Any routine based on \texttt{xQRDC} (LINPACK) or \texttt{xGEQPF}, \texttt{xGEQP3} (LAPACK) can catastrophically fail.

- \texttt{xGEQPX} (TOMS \# 782, rank revealing QRF)
- \texttt{xGELSX} and \texttt{xGELSY} in LAPACK ($\|Ax - b\|_2 \rightarrow \min$)
- \texttt{xGGSVP} in LAPACK (GSVD of $(A, B)$)

$$
U^T AQ = \begin{pmatrix}
0 & A_{12} & A_{13} \\
0 & 0 & A_{23} \\
0 & 0 & 0
\end{pmatrix}, \quad V^T BQ = \begin{pmatrix}
0 & 0 & B_{13} \\
0 & 0 & 0
\end{pmatrix}.
$$

- ... and many others ... long list. Need a new \texttt{xGEQP3}.
Consequences

Any routine based on xQRDC (LINPACK) or xGEQPF, xGEQP3 (LAPACK) can catastrophically fail.

- xGEQPX (TOMS # 782, rank revealing QRF)
- xGELSX and xGELSY in LAPACK ($\|Ax - b\|_2 \to \min$)
- xGGSVP in LAPACK (GSVD of $(A, B)$)

\[
U^T AQ = \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \\ 0 & 0 & 0 \end{pmatrix}, \quad V^T BQ = \begin{pmatrix} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{pmatrix}.
\]

- ... and many others ... long list. Need a new xGEQP3.
Introduction: goals

LAPACK release 3.1

Setting the scene: Jacobi SVD

QR+Businger–Golub CP

Analysis

An old fashioned debugging

Error analysis

New xGEQP3, XGEQPF

New updating formula

Perturbation estimates for the QRF
An old fashioned debugging

- Print out the SVD code and read it. :( 
- Read it again. :( Use colored pencils, draw flow diagrams, relate to the error analysis, check the theory, use test output. Conclusion: blame it to xGEQP3.
- Switch off optimized BLAS library (I used Intel’s MKL BLAS) and use only the code compiled from source codes. :)) All our bad examples turned into just ordinary runs, as expected. So, can we just blame it to the Intel’s MKL BLAS?
- We continue the stress test, without machine optimized BLAS. This was rather disappointing because we couldn’t use the efficiency of the MKL BLAS. And then, new bad examples (different from previous ones, but of the same type) appeared!
- Back to square one. Print out the xGEQP3 code (or the simpler xGEQP3F) and read it.
An old fashioned debugging

- Print out the SVD code and read it. :(  
- Read it again. :( Use colored pencils, draw flow diagrams, relate to the error analysis, check the theory, use test output. Conclusion: blame it to xGEQP3.  
- Switch off optimized BLAS library (I used Intel’s MKL BLAS) and use only the code compiled from source codes. :)) All our bad examples turned into just ordinary runs, as expected. So, can we just blame it to the Intel’s MKL BLAS?  
- We continue the stress test, without machine optimized BLAS. This was rather disappointing because we couldn’t use the efficiency of the MKL BLAS. And then, new bad examples (different from previous ones, but of the same type) appeared!  
- Back to square one. Print out the xGEQP3 code (or the simpler xGEQPF) and read it.
An old fashioned debugging

- Print out the SVD code and read it. :-(
- Read it again. :( Use colored pencils, draw flow diagrams, relate to the error analysis, check the theory, use test output. Conclusion: blame it to xGEQP3.

- Switch off optimized BLAS library (I used Intel’s MKL BLAS) and use only the code compiled from source codes. :)) All our bad examples turned into just ordinary runs, as expected. So, can we just blame it to the Intel’s MKL BLAS?

- We continue the stress test, without machine optimized BLAS. This was rather disappointing because we couldn’t use the efficiency of the MKL BLAS. And then, new bad examples (different from previous ones, but of the same type) appeared!

- Back to square one. Print out the xGEQP3 code (or the simpler xGEQPF) and read it.
An old fashioned debugging

- Print out the SVD code and read it. :(  
- Read it again. :( Use colored pencils, draw flow diagrams, relate to the error analysis, check the theory, use test output. **Conclusion: blame it to xGEQP3.**
- Switch off optimized BLAS library (I used Intel’s MKL BLAS) and use only the code compiled from source codes. :)) All our bad examples turned into just ordinary runs, as expected. So, can we just blame it to the Intel’s MKL BLAS?
- We continue the stress test, without machine optimized BLAS. This was rather disappointing because we couldn’t use the efficiency of the MKL BLAS. And then, new bad examples (different from previous ones, but of the same type) appeared!
- Back to square one. Print out the xGEQP3 code (or the simpler xGEQPF) and read it.
An old fashioned debugging

- Print out the SVD code and read it. :( 
- Read it again. :( Use colored pencils, draw flow diagrams, relate to the error analysis, check the theory, use test output. Conclusion: blame it to xGEQP3. 
- Switch off optimized BLAS library (I used Intel’s MKL BLAS) and use only the code compiled from source codes. :)) All our bad examples turned into just ordinary runs, as expected. So, can we just blame it to the Intel’s MKL BLAS? 
- We continue the stress test, without machine optimized BLAS. This was rather disappointing because we couldn’t use the efficiency of the MKL BLAS. And then, new bad examples (different from previous ones, but of the same type) appeared! 
- Back to square one. Print out the xGEQP3 code (or the simpler xGEQPF) and read it.
L(IN+A)PACK update

```
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
```

Expensive modification: SNRM2 in all cases. This also kills BLAS3 code.
L(IN+A)PACK update

\[ A^{(k)} \Pi_k = \begin{pmatrix} \cdot & \cdot & \odot & \cdot & \oplus & \cdot \\ \cdot & \cdot & \odot & \cdot & \oplus & \cdot \\ \boxtimes & \ast & \ast & \ast & \ast & \ast \\ \odot & \ast & \ast & \ast & \ast & \ast \\ \odot & \ast & \ast & \ast & \ast & \ast \\ \odot & \ast & \ast & \ast & \ast & \ast \end{pmatrix}, \quad a^{(k)}_j = \begin{pmatrix} \oplus \\ \oplus \\ \ast \\ \ast \\ \ast \end{pmatrix} = \begin{pmatrix} x^{(k)}_j \\ z^{(k)}_j \end{pmatrix} \]

(1)

\[ H_k z^{(k)}_k = \begin{pmatrix} R_{kk} \\ 0 \end{pmatrix}, \quad \begin{pmatrix} \beta^{(k+1)}_j \\ z^{(k+1)}_j \end{pmatrix}, \quad \omega^{(k)}_j = \|z^{(k)}_j\| = H_k z^{(k)}_j. \]

(2)

\[ \|z^{(k+1)}_j\| = \omega^{(k+1)}_j = \sqrt{\left(\omega^{(k)}_j\right)^2 - \left(\beta^{(k+1)}_j\right)^2}, \quad \text{provided that} \]

\[ \text{computed} \left( 1 - \left( \frac{\tilde{\beta}^{(k+1)}_j}{\tilde{\omega}^{(k)}_j} \right)^2 \right) \cdot \left( \frac{\tilde{\omega}^{(k)}_j}{\tilde{v}_j} \right)^2 > tol, \quad tol \approx 20\varepsilon, \]
L(IN+A)PACK update

\[ A^{(k)} \Pi_k = \begin{pmatrix} \cdots & \odot & \oplus & \cdots \\ \cdot & \odot & \oplus & \cdots \\ \bullet & \star & \star & \star \\ \odot & \star & \star & \star \\ \odot & \star & \star & \star \end{pmatrix}, \quad a_j^{(k)} = \begin{pmatrix} \oplus \\ \odot \\ \star \\ \star \\ \star \end{pmatrix} \equiv \begin{pmatrix} x_j^{(k)} \\ z_j^{(k)} \end{pmatrix} \quad (1) \]

\[ H_k z_k^{(k)} = \begin{pmatrix} R_{kk} \\ 0 \end{pmatrix}, \quad \beta_j^{(k+1)}, \quad \omega_j^{(k)} = \| z_j^{(k)} \| = H_k z_j^{(k)} \quad (2) \]

\[ \| z_j^{(k+1)} \| \equiv \omega_j^{(k+1)} = \sqrt{\left( \omega_j^{(k)} \right)^2 - \left( \beta_j^{(k+1)} \right)^2}, \quad \text{provided that} \]

\[ \text{computed} \left( 1 - \left( \frac{\tilde{\beta}_j^{(k+1)}}{\tilde{\omega}_j^{(k)}} \right)^2 \right) \cdot \left( \frac{\tilde{\omega}_j^{(k)}}{\tilde{v}_j} \right)^2 > tol, \quad tol \approx 20 \varepsilon, \]
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! How a WRITE statement can change the numerics?! A bug!!
Debugging: kill the optimizer

Delete the WRITE statement and start again, but with the optimizer switched OFF. Our bad examples turned into just fine ordinary runs. So, we blame it to the optimizer.

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 )" was meant to be an equivalent way of asking

\[
\text{IF ( 0.05*TEMP*( WORK( J ) / WORK( N+J ) )**2 .LT. EPS )}
\]

(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. A WRITE statement can help :)}
Debugging: -O -ffloat-store

Compiling with -O -ffloat-store makes it all right.

\[
\text{IF( TEMP2 .EQ. ONE ) THEN ...}
\]

<table>
<thead>
<tr>
<th>g77 -O -S</th>
<th>g77 -O -ffloat-store -S</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{flds -104(%ebp)} \text{fxch %st(1)} \text{jp L41 ...}</td>
<td>\text{fstps -24(%ebp)} \text{flds -24(%ebp)} \text{fxch %st(1)} \text{jp L41 ...}</td>
</tr>
</tbody>
</table>

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

But, this precludes extra precision and optimization! The machine is forced to do extra work in order to avoid using extra precision.
Debugging: use EPS explicitly

A way around this is to replace

\[
\text{IF ( TEMP2 .EQ. ONE )}
\]

with

\[
\text{IF ( 0.05*TEMP*( WORK( J ) / WORK( N+J ) )**2 .LT. EPS )}
\]

where \( \text{EPS} = \text{SLAMCH(’Epsilon’) } \) is the working precision. Try the stored bad examples with \(-o\) – it works in all cases. :) But, if we switch the optimizer off, or if we use \(-o -ffloat-store\) we quickly find new bad matrices. :( Assembler code reveals that the key point is in keeping \( \text{TEMP} = \text{ONE} - ( \text{ABS( A(I, J) ) / WORK( J ) )**2} \) in long register.
L(IN+A)PACK update

\[ A^{(k)} \Pi_k = \begin{pmatrix} \cdots & \odot & \odot & \oplus & \cdots \\ \odot & \cdots & \oplus & \cdots \\ \blacklozenge & \ast & \ast & \ast \\ \odot & \ast & \ast & \ast \\ \odot & \ast & \ast & \ast \end{pmatrix}, \quad a^{(k)}_j = \begin{pmatrix} \oplus \\ \oplus \\ \ast \\ \ast \\ \ast \end{pmatrix} \equiv \begin{pmatrix} x^{(k)}_j \\ z^{(k)}_j \end{pmatrix} \]  

(4)

\[ H_k z^{(k)}_j = \begin{pmatrix} R_{kk} \\ 0 \end{pmatrix}, \quad \begin{pmatrix} \beta^{(k+1)}_j \\ z^{(k+1)}_j \end{pmatrix}, \quad \omega^{(k)}_j = \| z^{(k)}_j \| = H_k z^{(k)}_j. \]  

(5)

\[ \| z^{(k+1)}_j \| \equiv \omega^{(k+1)}_j = \sqrt{(\omega^{(k)}_j)^2 - (\beta^{(k+1)}_j)^2}, \text{ provided that } \]  

\[ \text{computed} \left( 1 - \left( \frac{\beta^{(k+1)}_j}{\tilde{\omega}^{(k)}_j} \right)^2 \right) \cdot \left( \frac{\tilde{\omega}^{(k)}_j}{\tilde{\nu}_j} \right)^2 > tol, \quad tol \approx 20\varepsilon, \]
L(IN+A)PACK update

\[ A^{(k)} \Pi_k = \begin{pmatrix} \cdot & \cdot & \bigcirc & \cdot & \oplus & \cdot \\ \cdot & \bigcirc & \cdot & \oplus & \cdot \\ \mathbf{\blacksquare} & \mathbf{\bigstar} & \mathbf{\bigstar} & \mathbf{\bigstar} & \mathbf{\bigstar} \\ \mathbf{\bigcirc} & \mathbf{\bigstar} & \mathbf{\bigstar} & \mathbf{\bigstar} & \mathbf{\bigstar} \\ \mathbf{\bigcirc} & \mathbf{\bigstar} & \mathbf{\bigstar} & \mathbf{\bigstar} & \mathbf{\bigstar} \end{pmatrix} , \quad a_j^{(k)} = \begin{pmatrix} \oplus \\ \oplus \\ \bigstar \\ \bigstar \\ \bigstar \end{pmatrix} = \begin{pmatrix} x_j^{(k)} \\ z_j^{(k)} \end{pmatrix} \]

(4)

\[ H_k z_k^{(k)} = \begin{pmatrix} R_{kk} \\ 0 \end{pmatrix} , \quad \begin{pmatrix} \beta_j^{(k+1)} \\ z_j^{(k+1)} \end{pmatrix} , \quad \omega_j^{(k)} = \|z_j^{(k)}\| = H_k z_j^{(k)} . \]

(5)

\[ \|z_j^{(k+1)}\| \equiv \omega_j^{(k+1)} = \sqrt{(\omega_j^{(k)})^2 - (\beta_j^{(k+1)})^2} , \quad \text{provided that} \]

\[ \text{computed} \left( 1 - \left( \frac{\tilde{\beta}_j^{(k+1)}}{\tilde{\omega}_j^{(k)}} \right)^2 \right) \cdot \left( \frac{\tilde{\omega}_j^{(k)}}{\tilde{\nu}_j} \right)^2 > \text{tol} , \quad \text{tol} \approx 20\varepsilon , \]
Error analysis

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

Observations:
- Sharpest non-zero drop in (6) is \( \text{SRT}(\text{EPS}) = \sqrt{\varepsilon} \)
- If \( \text{rank}(A) = n \), \( \|A_c^\dagger\| \geq \|A(:,j)\|_2/\omega_j^{(k)} \) for all \( j, k \).
- \( \text{rank}(A) < n \) iff \( \omega_j^{(k)} = 0 \) for some \( j, k \).
- Relative condition number for \( f(x) = \sqrt{1 - x^2} \) is seen from
  \[ \frac{f(x + \delta x) - f(x)}{f(x)} \approx \frac{\delta x \cdot xf'(x)}{x \cdot f(x)} = \frac{\delta x}{x} \cdot \frac{-x^2}{1 - x^2} \]
- Condition number has its own condition number.
Error analysis: it hurts soo good

Analyze one step, no accumulated cancellations, \( \tilde{\omega}_j^{(k)} = \tilde{\nu}_j \).

Condition number of the update:

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

\( \hat{t}_j^{(k)} \) not accessible. Instead have computed TEMP = \( \tilde{\nu}_j^{(k)} \)

\[
\hat{t}_j^{(k)} = \frac{1}{1 + \sigma_j^{(k)}(\frac{\tilde{t}_j^{(k)}}{1 + \epsilon_3} + \sigma_j^{(k)}), \quad \tilde{t}_j^{(k)} = \max\{1 \oplus (\tilde{\beta}_j^{(k+1)} \odot \tilde{\omega}_j^{(k)}) \ast \ast 2,}
\]

Conspiracy theory: \( \tilde{t}_j^{(k)} \approx 30\varepsilon > tol \), and \( \hat{t}_j^{(k)} \approx O(\varepsilon^2) \ll tol \). Distance to singularity misscalculated! Failed in a single step. More updating steps, more problems.
Error analysis: it hurts soo good

Now, use $tol = \sqrt{\varepsilon} = \text{SQRT(EPS)} \approx 2.44 \cdot 10^{-4}$ and rewrite the LINPACK/LAPACK update into

$$ t = |\tilde{\beta}_j^{(k+1)}/\tilde{\omega}_j^{(k)}|; \quad t = \max\{0, 1 - t^2\} $$

$$ t_2 = t \cdot (\tilde{\omega}_j^{(k)}/\tilde{\nu}_j)^2 $$

if $(t_2 \leq \sqrt{\varepsilon})$ 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

In all tests it performed very well, with MKL BLAS $\pm 3 - 4\%$ timing difference compared to LAPACK’s SGEQP3.

Adopted in LAPACK 3.1. as new xGEQP3, XGEQPF.

Error analysis tedious, and not conclusive. Cannot prove reliability in the standard model of the IEEE arithmetic. It can fail if $tol$ is reduced from $2.44 \cdot 10^{-4}$ to $10^{-4}$. 
NEW update

\[ A^{(k)} \Pi_k = \begin{pmatrix} \cdot & \cdot & \circ & \cdot & \oplus & \cdot \\ \cdot & \circ & \cdot & \oplus & \cdot \\ \blacklozenge & \ast & \ast & \ast & \ast \\ \circ & \ast & \ast & \ast & \ast \\ \circ & \ast & \ast & \ast & \ast \\ \circ & \ast & \ast & \ast & \ast \end{pmatrix}, \quad a_j^{(k)} = \begin{pmatrix} \oplus \\ \oplus \\ \ast \\ \ast \\ \ast \\ \ast \end{pmatrix} \equiv \begin{pmatrix} x_j^{(k)} \\ z_j^{(k)} \end{pmatrix} \quad (7) \]

\[ H_k z_k^{(k)} = \begin{pmatrix} R_{kk} \\ 0 \end{pmatrix}, \quad \left( \beta_j^{(k+1)} \right) \left( z_j^{(k+1)} \right), \quad \omega_j^{(k)} = \| z_j^{(k)} \| = H_k z_j^{(k)}. \quad (8) \]

\[ \| a_j^{(k)} \| = \alpha_j^{(k)} = \alpha_j^{(0)}; \quad \xi_j^{(k+1)} = \sqrt{(\xi_j^{(k)})^2 + (\beta_j^{(k+1)})^2} \]

\[ \| z_j^{(k+1)} \| \equiv \omega_j^{(k+1)} = \sqrt{(\alpha_j^{(0)})^2 - (\xi_j^{(k+1)})^2} \]
New updating strategy

\[
\tilde{\xi}^{(k+1)}_j = \max \{ \tilde{\xi}^{(k)}_j, \beta_j^{(k+1)} \} \sqrt{1 + \left( \frac{\min \{ \tilde{\xi}^{(k)}_j, \beta_j^{(k+1)} \}}{\max \{ \tilde{\xi}^{(k)}_j, \beta_j^{(k+1)} \}} \right)^2}
\]

\[
t_0 = \frac{\tilde{\xi}^{(k+1)}_j}{\tilde{\alpha}_j^{(1)}} \quad \% \text{ here } \tilde{\xi}^{(k+1)}_j \text{ forward stable.}
\]

if ( \( t_0 < \gamma_1 \)) then \% here \( \gamma_1 \approx 1 - \sqrt{\epsilon} \)

\[
\tilde{\omega}^{(k+1)}_j = \tilde{\alpha}_j^{(1)} \sqrt{1 - t_0^2}
\]

else

\[
t_1 = |\beta_j^{(k+1)} / \tilde{\omega}_j^{(k)}| ; \quad t_2 = \max \{ 0, 1 - t_1^2 \}
\]

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
New updating strategy

Repeated forward stable updates in the past of a partial column and only one risky operation make safe implementation and its analysis straightforward. The new code is comparable in speed with previous implementations. It requires $n$ extra locations in the workspace – an acceptable overhead for provable reliability. The code has passed all tests.

Then it happened again :(o. The disappointment was huge – both versions of the code failed on one example. But this time, one routine had theoretical proof of its reliability.

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.
New updating strategy

Repeated forward stable updates in the past of a partial column and only one risky operation make safe implementation and its analysis straightforward. The new code is comparable in speed with previous implementations. It requires $n$ extra locations in the workspace – an acceptable overhead for provable reliability. The code has passed all tests.

Then it happened again :(o. The disappointment was huge – both versions of the code failed on one example. But this time, one routine had theoretical proof of its reliability.

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.
New updating strategy

Repeated forward stable updates in the past of a partial column and only one risky operation make safe implementation and its analysis straightforward. The new code is comparable in speed with previous implementations. It requires $n$ extra locations in the workspace – an acceptable overhead for provable reliability. The code has passed all tests.

Then it happened again :(o. The disappointment was huge – both versions of the code failed on one example. But this time, one routine had theoretical proof of its reliability.

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.
Z.D. and Zvonimir Bujanović, Zagreb 2006
New code xGEQP3Z tested.

- Provably delivers Businger–Golub structured $R$ (up to small roundoff)
- Same efficiency as xGEQP3 (up to ± 3 – 4 %)
- Makes our Jacobi SVD code more robust
- Makes many other solvers more robust and can prevent catastrophes in mission critical applications
- Used as replacement for xGEQP3 in LAPACK 3.1.

NEW software

Z.D. and Zvonimir Bujanović, Zagreb 2006
New code xGEQP3Z tested.

- Provably delivers Businger–Golub structured $R$ (up to small roundoff)
- Same efficiency as xGEQP3 (up to $\pm 3 - 4\%$)
- Makes our Jacobi SVD code more robust
- Makes many other solvers more robust and can prevent catastrophes in mission critical applications
- Used as replacement for xGEQP3 in LAPACK 3.1.

Introduction: goals

LAPACK release 3.1

Setting the scene: Jacobi SVD

QR+Businger–Golub CP

Analysis

Perturbation estimates for the QRF
Perturbation in the QRF

Think of $\|Ax - b\| \rightarrow \min$, and $A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$, $x = R^{-1} c$, $c = Q^T b$. With computed QR, $(R + \delta R)\tilde{x} = \tilde{c}$,

$$\tilde{x} = (I + R^{-1} \delta R)^{-1} x$$

Write $R = DR_r$, $D$ diagonal, $R_r$ unit rows.

$$R^{-1} \delta R = R_r^{-1} (D_r^{-1} \delta R).$$