Non-Negative Diagonals and High Performance on Low-Profile Matrices from Householder QR

LAPACK Working Note 203

James W. Demmel Mark Hoemmen Yozo Hida
E. Jason Riedy

May 30, 2008

Abstract

The Householder reflections used in LAPACK's QR factorization leave positive and negative real entries along R's diagonal. This is sufficient for most applications of QR factorizations, but a few require that R have a non-negative diagonal. This note provides a new Householder generation routine to produce a non-negative diagonal. Additionally, we find that scanning for trailing zeros in the generated reflections leads to large performance improvements when applying reflections with many trailing zeros. Factoring low-profile matrices, those with non-zero entries mostly near the diagonal (*e.g.* band matrices), now requires far fewer operations. For example, QR factorization of matrices with profile width b that are stored densely in an $n \times n$ matrix improves from $O(n^3)$ to $O(n^2 + nb^2)$.

1 Introduction

The Householder reflections used in LAPACK's QR factorization leave positive and negative real entries along R's diagonal. This is sufficient for most applications of QR factorizations, but a few require that R have a non-negative diagonal. G. W. Stewart [1980] notes such a restriction when using QR to generate a random, uniformly distributed (by the Haar measure) unitary operator Q. Another example comes from continuing a Krylov subspace in Householder GMRES [Homer F. Walker, 1988]. The iteration must start with a reflection in the same direction as when the iteration was paused. Generating all reflections in a known direction (along the positive real axis) ensures the iteration always can be restarted.

This note provides a new Householder generation routine to produce a non-negative diagonal. Maintaining compatibility with existing code *inside* LAPACK requires adding a new routine rather than modifying the existing generators x`LARFG`. The multi-shift Hessenberg QR routines [Karen Braman, et al., 2002,?] directly generate and apply small reflections to predict when the subdiagonal entries are negligible. We decided to introduce a new routine rather than rework the deflation prediction code. There are no changes to any existing programming interface.

LAPACK's routines for applying reflections (in versions 3.1.1 and prior) handle columns that already are zero below the diagonal and real on the diagonal specially. The routines do not apply the reflection at all when there is no work to do. However, the reflection still is applied if the diagonal has a non-zero imaginary part, even if the trailing vector is entirely zero. The complex x`GEQRF` routines in these releases of LAPACK apply $O(n^3)$ operations to factor matrices that already are upper triangular. If we were to follow the same path and reflect the entire, mostly zero column to switch the diagonal's sign, almost all upper-triangular matrices would require $O(n^3)$ work to produce a trivial factorization.

Instead, we modify the routines to scan for trailing zeros inside the reflection and in the matrix to which the reflection is applied. This modification not only prevents wasting work during trivial factorizations, it also reduces the time to factor low-profile matrices. For example, factoring a matrix of bandwidth b in dense format requires $O(n^2 + nb^2)$ computation when trailing zeros are not applied. While still greater than the $O(nb^2)$ work required when the matrix is stored in a band format, the new cost is far less than the full, dense $O(n^3)$ and requires no user-level changes. This change also reduces the cost of factoring upper-triangular matrices to $O(n^2)$. The $O(n^2)$ component accounts for scanning for the zeros, and the $O(nb^2)$ component accounts for the floating-point work in the updates.

Scanning for trailing zeros in both Householder vectors and the target matrix also improves performance on low-profile matrices, *e.g.* a matrix with a narrow profile except for one large, dense block. Similar techniques apply to LU factorization, and could improve performance for low-profile matrices both asymptotically and by enabling BLAS-3 operations without copying. We leave those investigations to future work.

It is worth noting that scanning for trailing zeros during two-sided reductions provides no *asymptotic* benefit when an output or intermediate value is dense. For example, reducing a band matrix to Hessenberg form produces a matrix with a dense upper triangle, and reduction to tridiagonal form produces dense orthogonal matrices. It remains to be seen how scanning affects the constant factor when an implementation is applied to practical matrices.

We derive the routine to compute the necessary Householder reflections accurately in Section 2. Section 3 demonstrates that checking for short Householder reflections improves QR performance on band matrices to $O(n^2 + nb^2)$. And Section 4 describes which routines in LAPACK are affected by our changes.

2 Computing Householder Reflections for a Non-Negative Diagonal

A Householder reflection is a unitary transformation $H \in \mathbb{C}^{n \times n}$ that maps a complex vector $b \in \mathbb{C}^n$ to $p \in \mathbb{C}^n$ by $Hb = p$, "reflecting" the vectors so they project onto the same point in the space orthogonal to the span of $v = b - p$, with $\|b\|_2 = \|p\|_2$. Following Dirk Laurie [1997], $H = (I - P) - wP$ for the projection matrix $P = \frac{vv^*}{v^*v}$ and parameter $w = \overline{v^*b}/v^*b$. In our notation, I is the $n \times n$ identity, and v^* is the Hermitian transpose of v. For real vectors b and p, $w = 1$ and $H = I - 2P$.

The LAPACK routines x`LARFG` generate the transpose of H as reflectors in a form that is packed into the lower triangle during QR factorizations. These routines essentially accept an n-long vector $[\alpha; x]$, where α is a scalar and x holds the other $n-1$ entries. The leading scalar is treated separately for storage optimizations within LAPACK. We use the GNU Octave [John W. Eaton, 2002] dialect of MATLAB®'s language [The MathWorks, Inc., 2007] for indexing and building matrices and vectors, so $[\alpha; x]$ stacks the scalar α above the vector x.

The LAPACK reflections take the form $H^* = I - \tau[1; y][1; y]^*$, producing a scalar τ and overwriting the vector with $[\beta; y]$ such that $H^*[\alpha; x] = [\beta; 0]$. This different form allows the transformation to produce the diagonal entry β in the QR factorization while storing the transformation vector y below the diagonal. The $\min\{m, n\}$ scalars τ generated in an $m \times n$ factorization are stored in a separate array.

Requiring that $[\alpha; x]$ is reflected to $[\beta; 0]$ only constrains the magnitude $|\beta| = \|[\alpha; x]\|_2$ and not the direction or sign of β. The LAPACK code for x`LARFG` in versions 3.1.1 and prior constrain β to be real. As discussed above, some applications would find $\beta \geq 0$ to be useful. Computing the reflection accurately while producing a real $\beta \geq 0$ requires some care when $|\operatorname{Re}\alpha| \approx \beta$.

To map from Laurie's formulations to LAPACK's,

$$\begin{aligned} \tau &= (1 + \overline{w})\frac{\overline{v(1)}v(1)}{v^*v}, \text{ and} \\ y &= \frac{v(2:n)}{v(1)}. \end{aligned}$$

Substituting the LAPACK parameters and simplifying gives

$$\begin{aligned} \tau &= \frac{\beta - \alpha}{\beta}, \text{ and} \\ y &= \frac{x}{\alpha - \beta}. \end{aligned}$$

There are two complications in computing these quantities: limited precision when $|\operatorname{Re}\alpha| \approx \beta$ and limited range when computing $\|[\alpha; x]\|_2$. LAPACK's x`NRM2`, x`LAPY2`, and x`LAPY3` routines scale intermediates to avoid unnecessary overflows in $\|[\alpha; x]\|_2$. The reflection routines attempt to detect underflow and rescale the data; see W. Kahan [1981] for details. Hence, we wish to handle computing $|\operatorname{Re}\alpha| \approx \beta$ without introducing *new* overflow problems. We assume that complex division is computed accurately and without extraneous overflows [Douglas M. Priest, 2004].

The Householder reflection H^* preserves length, so the real scalar $\beta = \sigma\|[\alpha; x]\|_2$ with $\sigma = \pm 1$. Common folklore dictates choosing $\sigma = -\operatorname{sign}\alpha$. Then $\alpha - \beta$ becomes an addition of like-signed quantities and is performed to high relative accuracy. With $\sigma = \operatorname{sign}\alpha$, $\alpha - \beta \approx 0$ when $x \approx 0$, and the computation can lose all accuracy relative to the input values through cancellation. LAPACK's x`LARFG` chooses $\sigma = -\operatorname{sign}\alpha$, producing β values of either sign. Listing 1 provides an Octave translation of LAPACK's x`LARFG` for reference. We assume **norm** (x, 2) emulates

Listing 1: Octave translation of LAPACK's xLARFG routines.

```
function [tau, beta, y] = xlarfg (alpha, x)
  xnorm = norm (x, 2);
  if xnorm == 0 && imag (alpha) == 0,
    tau = zero; beta = alpha; y = x; return;
  endif
  beta = -sign (real (alpha)) * norm([alpha; x], 2);
  [k, beta, alpha, x, xnorm] = possibly_rescale (beta, alpha, x, xnorm);
  tau = (beta - alpha) / beta;
  y = x/(alpha - beta);
  if k > 0, beta = beta * 2**k; endif
endfunction
```

LAPACK's routines and computes carefully to avoid overflow. We hide details of rescaling to avoid underflows in the routine possibly_rescale in Listing 3.

Note that $\beta = 0$ implies $\alpha = 0$ and $x = 0$. In this case we set $\tau = 0$ and $H^* = I$. We will revisit "shortened" transformations like $H^* = I$ in Section 3.

Beresford N. Parlett [1971] and others show that the subtractions can be expanded and canceled algebraically when necessary, maintaining the relative accuracy. To leave a positive entry on the diagonal, set $\sigma = 1$ and $\beta = \|[\alpha; x]\|_2$. To preserve accuracy in $\operatorname{Re}\alpha - \beta$ when $\operatorname{Re}\alpha > 0$, we use $\beta^2 = (\operatorname{Re}\alpha)^2 + (\operatorname{Im}\alpha)^2 + \|x\|_2^2$ in computing

$$\begin{aligned}\operatorname{Re}\alpha - \beta &= (\operatorname{Re}\alpha - \beta)\frac{\operatorname{Re}\alpha + \beta}{\operatorname{Re}\alpha + \beta} \\ &= \frac{(\operatorname{Re}\alpha)^2 - \beta^2}{\operatorname{Re}\alpha + \beta} \\ &= -\frac{(\operatorname{Im}\alpha)^2 + \|x\|_2^2}{\operatorname{Re}\alpha + \beta}.\end{aligned}$$

Let $\gamma = \operatorname{Re}\alpha + \beta$. Now $\gamma \geq \beta = \|[\alpha; x]\|_2 \geq \|x\|_2$ and similarly $\gamma \geq |\operatorname{Im}\alpha|$. So we can rearrange the expression for $\operatorname{Re}\alpha - \beta$ to avoid overflow during computation by computing

$$\delta = \operatorname{Re}\alpha - \beta = -\left(\operatorname{Im}\alpha \cdot \left(\frac{\operatorname{Im}\alpha}{\gamma}\right) + \|x\|_2 \cdot \left(\frac{\|x\|_2}{\gamma}\right)\right).$$

We produce a non-negative β when $\operatorname{sign}\operatorname{Re}\alpha > 0$ by computing τ and y as

$$\begin{aligned}\tau &= \frac{\beta - \alpha}{\beta} = -\frac{\delta + \hat{\imath}\cdot\operatorname{Im}\alpha}{\beta}, \text{ and} \\ y &= \frac{x}{\alpha - \beta} = \left(\frac{1}{\delta + \hat{\imath}\cdot\operatorname{Im}\alpha}\right)x.\end{aligned}$$

Both ${}^{\operatorname{Im}\alpha}/_{\gamma}$ and ${}^{\|x\|_2}/_{\gamma}$ are less than one, so $|\delta| \leq |\beta|$ and these computations introduce no new overflow possibilities into the existing routines.

Listing 2: Octave translation of the new xLARFP routines. The REAL Fortran implementation is simpler because **imag**(alpha) == 0 and **real**(alpha) == alpha.

```
function [tau, beta, y] = xlarfp (alpha, x)
  xnorm = norm (x, 2);
  if xnorm == 0 && imag (alpha) == 0,
    tau = zero; beta = alpha; y = x; return;
  endif
  beta = -sign (real (alpha)) * norm ([alpha; x], 2);
  [k, beta, alpha, x, xnorm] = possibly_rescale (beta, alpha, x, xnorm);
  if beta >= 0,
    tau = (beta - alpha) / beta;
    y = x/(alpha - beta);
  else
    beta = -beta;
    gamma = real (alpha) + beta;
    delta = -(imag (alpha) * (imag (alpha) / gamma) ...
             + xnorm * (xnorm / gamma));
    tau = -(delta + i*imag (alpha)) / beta;
    y = x / (delta + i*imag (alpha));
  endif
  if k > 0, beta = beta * 2**k; endif
endfunction
```

The arithmetic operations computing each component from a given α and β involve a constant number of additions of like-signed quantities, multiplications, and divisions. So long as the divisions are computed accurately without extraneous overflows [Douglas M. Priest, 2004] and rescaling has rendered underflow innocuous, both τ and y are computed accurately from the given α, β, and $\|x\|_2$.

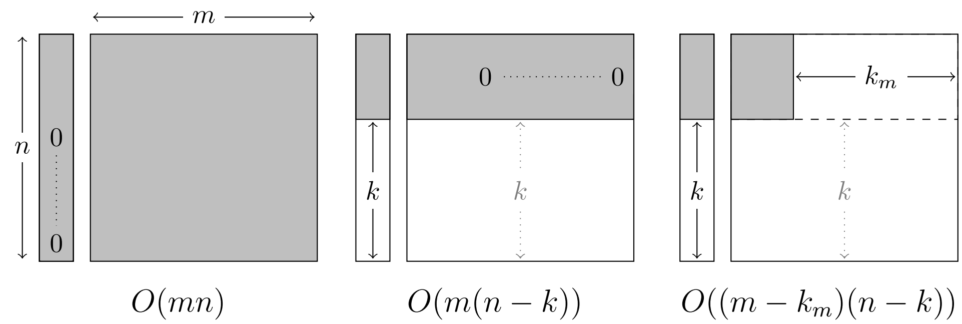
The more in-depth analysis of Beresford N. Parlett [1971] shows that the errors in computing $\|x\|_2$ and β dominate the total error for real reflections. If we assume that $\|x\|_2$ and β are computed satisfactorily after scaling, then we expect errors within a tiny constant factor of the existing xLARFG routines and the unchanged $\mathrm{Re}\,\alpha < 0$ branch. Assuming the computation time is dominated by computing $\|x\|_2$, the few additional operations necessary to maintain $\beta \geq 0$ do not affect performance.

Listing 2 provides Octave code equivalent to xLARFP, a new LAPACK routine for producing reflections with $\beta \geq 0$. The Fortran version passes all of LAPACK's tests in all types and precisions when used in QR, RQ, QL, and LQ factorizations.

3 Short Householder Reflections

Requiring that β be real and non-negative introduces an efficiency concern for inputs of the form $[\alpha; 0]$. These already have the correct form, so the vector $y = 0$. The xLARFG routines do not worry about the sign and always produce $\tau = 0$ and hence

Figure 1: Scanning the reflector for trailing zeros reduces the cost to apply it from $O(mn)$ to $O(m(n-k))$. Scanning the target matrix reduces the cost of applying the reflection to $O((m-k_m)(n-k))$.



$H = I$. The routines that apply reflections, *e.g.* xLARF, handle the $\tau = 0$ case specially and do not compute with y at all. Taking advantage of zeros beyond $\tau = 0$ provides significant performance benefits.

To produce β real and non-negative when $x = 0$, xLARFP also produces $y = 0$, but τ can lie anywhere on the unit circle centered at 1 when the input α is complex. Introducing a special case similar to $\tau = 0$ would require testing $|\tau - 1| = 1$ and carefully analyzing rounding errors in that test. Instead, we scan the vector y for the final non-zero entry, beginning with the bottom of the vector. This simple and accurate test may require $n-1$ comparisons with zero. Scanning for zeros not only handles the case where $|\tau - 1| = 1$ but also cases where $[\alpha; x]$ has trailing zeros even if not all of x is zero.

Detecting k zeros in the reflection saves $O(km)$ work when the reflection is applied to m vectors. Detecting k_m zeros in the upper $n-k$ slice of the vectors to be transformed saves additional work. Figure 1 graphically shows the reduction in cost. If the common case is that every entry of x or its target is $\neq 0$, then the tests waste only one comparison each and do not change overall performance.

The impact is more apparent in QR factorization. If a user has a square, $n \times n$ matrix with a profile of width b (*e.g.* a band matrix), factoring that low-profile matrix stored in a band storage structure should require $O(nb^2)$ computation. However, LAPACK does not implement a band QR factorization. Factoring the low-profile matrix using full storage requires $O(n^3)$ computation. Scanning for the final trailing non-zeros reduces this cost to $O(n^2 + nb^2)$ with no additional work for the user. The $O(n^2)$ component accounts for the comparisons, and the $O(nb^2)$ component accounts for the applying the reflections. If we scanned only the reflections for trailing zeros, the asymptotic cost of QR factorization on band matrices would be reduced only to $O(n^2 b)$ because all $O(n)$ columns would be transformed in the b-sized block of rows.

The same technique works for the partitioned factorizations used in LAPACK's xGEQRF. Each block of columns is scanned for the last zero *row*, and each block of rows is scanned for the last zero *column*. More complicated schemes to split blocks according to zero locations are conceivable but likely introduce too much overhead to be efficient. Future work will investigate the performance on other structures stored in a dense matrix.

Figure 2 compares the times for band LU factorization, full QR factorization, and QR factorization that checks for short reflections when applied to double-precision, real matrices with bandwidth 40 and dimensions from $n = 200$ to 1500. We compare with band LU because LAPACK does not include a band QR factorization. The algorithms were run three times each and the minimum time was selected as representative of the best possible performance.

The least-squares lines on the log-log plot show three distinct slopes roughly corresponding to the algorithms' exponents. Full QR has a slope of 2.6, band LU has a slope of 1.1, and the new, short QR has a slope of 1.3. The slopes of 2.6 for full QR and 1.3 for short QR are smaller than the expected values of 3 and 2, respectively. We suspect that the differences reflect architectural, compilation, and implementation issues and not fundamental algorithmic characteristics; similar differences arose when examining eigenvalue algorithms in James W. Demmel, et al. [2007]. Our values above suffice to show that scanning for zeros improves dense QR performance on band matrices nearly to that of explicitly band LU.

The timing difference between QR factorization using x`LARFG` and using x`LARFP` is below the noise threshold of repeated measurements. Hence we claim no performance impact for ensuring a non-negative diagonal.

These timings are from an Intel® Core™2 Duo T6400 using LAPACK 3.1.1 and ATLAS 3.6.0. The processor's frequency was set to 2.13GHz. The processor has a 2 MiB cache, which corresponds to a 512×512 double precision matrix or 362×362 double-complex matrix. The tests were driven from Octave 3.0.1.

4 Impacts within LAPACK

Table 1 lists the suffixes for LAPACK routines that are directly or indirectly modified to use x`LARFP`. No existing programming interfaces are changed. Only the values returned are different, and then the changes still hold to the existing documentation. The LAPACK documentation for routines x`GEQRF` in versions 3.1.1 and prior make no promises about the diagonal. The complex routines x`LARFG` do document that they alter the diagonal to be real. The new reflections are only *adding* restrictions on the output; requiring a real, non-negative diagonal should not adversely affect users of x`GEQRF`. An alternative design is to add new routine names for x`GEQRF` variations that produce a non-negative diagonal. Every new routine name carries high long-term software maintenance and documentation costs, so we prefer modifying the existing factorizations. It is conceivable that some user has relied on the undocumented behavior that real versions of x`GEQRF` do not alter a triangular input matrix. Surprising such users is unfortunate, but we feel the known applications combined with the costs of creating new names for QR factorization routines outweigh the costs of possibly affecting unknown users.

The QR routine suffixes along with the routines x`GELS` listed in Table 2 could gain improved asymptotic performance on low-profile matrices. The $O(n^3)$ component of

Figure 2: Timing v. dimension for a double-precision matrix of bandwidth 20. Slopes of the least-squares lines: full QR, 2.6; short QR, 1.3; band LU, 1.1. The dashed red line displays the size where one full matrix fits in cache.

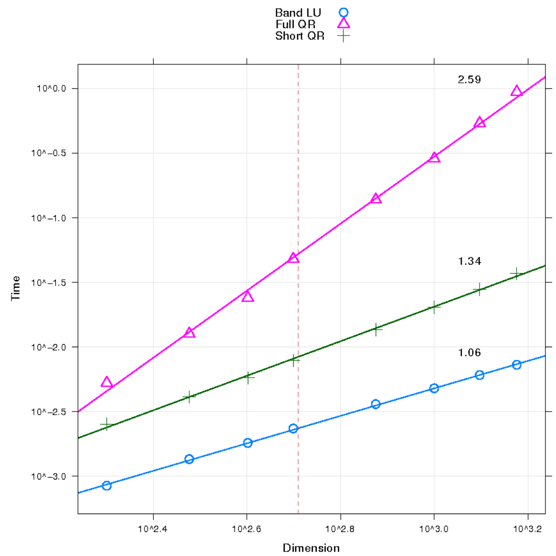


Table 1: Routine suffixes modified to use x`LARFP` and produce an R with a non-negative diagonal.

Routine suffix	Description
`LS`	QR-based least squares
`QRF, QLF, RQF, LQF`	QR decomposition
`QP3, QP2`	Column-pivoting QR decomposition

Table 2: Routine suffixes and structures that have been modified to asymptotic performance benefits on low-profile matrices by scanning for trailing zeros in reflections.

Routine suffix	Description
`LS`	QR-based least squares
`QRF, QLF, RQF, LQF`	QR decomposition

performance improves to $O(n^2 + nb^2)$ where b is the profile width, or bandwidth if the matrix is band. LAPACK does not contain band or low-profile versions of any of the QR or least-squares routines. Also, the likely $O(n^3)$ behavior of complex QR in LAPACK's x`GEQRF` on upper-triangular inputs is reduced to the more-expected $O(n^2)$.

The usual implementations of two-sided decompositions like reduction to tridiagonal form take a low-profile matrix and increase its profile linearly with the column number, either directly if the matrix is stored in a general structure (*e.g.* x`SYTRD`) or through a separate matrix for accumulating the transformations (*e.g.* x`SBTRD` when the Q matrix is requested). We expect these routines will see no significant performance changes from our changes. Other decompositions that maintain limited bandwidths like LU also could benefit from scanning for short transformations.

Table 3 lists all "external" routine suffixes affected directly or indirectly by the shortened Householder reflections. Routines with the second two characters `LA` are considered "internal" and are not listed. The entire LAPACK test suite, including tests for these routines, encounters no unexpected failures with our changes on two platforms (64-bit Intel® Core™2 Duo and Itanium® 2) and two BLAS implementations (reference and ATLAS [R. Clint Whaley, et al., 1997]).

The Hessenberg QR routines assume the x`LARFG` reflections when checking for deflation. The routines x`LAHQR` include an inlined, optimized reflection used when looking ahead for a negligible subdiagonal entry. Those routines cannot use the new x`LARFP` routines without further modification. However, we know of no applications for a non-negative subdiagonal in Hessenberg form, or of non-negative off-diagonals from other reductions. To minimize impact on user code, *only* routines listed in Table 1 use the new x`LARFP`.

Table 4 lists all the changed and new computational routines. All are "internal", although we expect there will be external users of the Householder generation and application routines. The other routines are simple helpers and should not be used outside of LAPACK. In particular, platform-specific tuned implementations may decide not to use the panel factorization routines or zero scanning routines and may not include those routines in their libraries.

Table 3: Routine suffixes affected by the shortened reflections and which *may* see performance changes on band and low-profile matrices. The boxed routines are known to be *incompatible* with xLARFP without other, internal changes.

Routine suffix	Description
LSD, LSE, LSS, LSY	least squares
GLM	Gauss-Markov linear model
QP3, QP2	Column-pivoting QR decomposition
SDD, SJA, SVD	SVD and generalized SVD decomposition
EV, EVR, EVX, EVD	Eigenvalue decomposition
GVD, GVX	Generalized eigenvalue decomposition
$\boxed{\texttt{ES, ESX, EQR}}$	Schur decomposition
SEN, EXC	Schur decomposition reordering
TRSNA, TGSNA	Eigenvalue condition estimation
BRD, BD2	Reduction to upper bidiagonal
HRD, HD2	Reduction to upper Hessenberg
QEZ	Hessenberg, triangular pair to eigens
RZF	Reduction to upper trapezoidal
TRD	Reduction to tridiagonal
SVP	Pre-processing for generalized SVD
GQR, GRQ, GQL, GLQ	Reflections to orthonormal/unitary matrix
G2R, GR2, G2L, GL2	Reflections to orthonormal/unitary matrix
GHR, GBR, GTR	Reflections to orthonormal/unitary matrix
MQR, MRQ, MQL, MLQ	Apply reflections to a matrix
M2R, MR2, M2L, ML2	Apply reflections to a matrix
MHR, MBR, MTR	Apply reflections to a matrix

Table 4: Routines created or directly modified.

Routine suffix	Description
LARFP	New generator: non-neg. diagonal
LARFG	Previous generator: scans for trailing zeros
LARF, LARFX, LARFB, LARFT	Applying reflections: scan for trailing zeros
LAQR2, LAQL2, LARQ2, LALQ2	QR panels: use xLARFP
LAQP2, LAQPS	Pivoting QR panels: use xLARFP
LATRZ	Trapezoidal panels: use xLARFP
ILAxLC, ILAxLR	New: find matrix's last non-zero column, row
ILAxLV	New: find vector's last non-zero entry

5 Summary

The code to generate reflections leaving a non-negative real is available as new LAPACK routines x`LARFP`. LAPACK's QR flavors pass all their tests when using x`LARFP`, and the entire test suite sees no unexpected failures with both x`LARFP` and the shortened reflections. Checking for trailing zeros in reflectors is included in the routines x`LARF`, x`LARFB`, and x`LARFT`. The modifications do not reduce the performance of QR factorization. Users with low-profile matrices see performance *gains* in Table 2's routines of at least a factor of n with no changes to their data layout.

Similar shortening tricks could be applied to Gauss transforms in LU factorization. The column already is searched for a pivot; the same search could detect the final non-zero. LAPACK already includes a band LU factorization, but scanning for zeros can improve performance substantially for matrices that do not have perfect band structure. Achieving band- or sparse-like performance with user-friendly dense data structures is a promising direction for future work.

Computing a reflection to maintain a non-negative diagonal does not affect ScaLAPACK's parallel `P`x`LARFG` significantly. Communicating the location of the last non-zero in a reflector could be bundled with broadcasting τ without introducing new messages. We have not made these modifications, however. The communication-avoiding QR factorization in [James W. Demmel, et al., 2008] also avoids needing any significant modifications to produce a non-negative, real diagonal. Only the final QR factorization at the top of the reduction tree need worry about using the new x`LARFP`. Applying reflections is handled locally within each step, so there is no additional communication when taking advantage of any discovered structure.

Acknowledgments

Alan Edelman posed the initial request for LAPACK's QR factorization to return a real, non-negative diagonal in R. Sven Hammarling identified the performance problem of having undetected zero reflector vectors y.

References

Karen Braman, Ralph Byers, and Roy Mathias, *The multishift QR algorithm. part I: Maintaining well-focused shifts and level 3 performance*, SIAM Journal on Matrix Analysis and Applications, 23 (2002), pp. 929–947.

——, *The multishift QR algorithm. part II: Aggressive early deflation*, SIAM Journal on Matrix Analysis and Applications, 23 (2002), pp. 948–973.

James W. Demmel, Laura Grigori, Mark Frederick Hoemmen, and Julien Langou, *Communication-avoiding parallel and sequential QR factorizations*, Tech. Report UCB/EECS-2008-74, EECS Department, University of California, Berkeley, May 2008.

James W. Demmel, Osni A. Marques, Beresford N. Parlett, and Christof Vmel, *Performance and accuracy of LAPACK's symmetric tridiagonal eigensolvers*, Tech. Report 183, LAPACK Working Note, Apr. 2007.

John W. Eaton, *GNU Octave Manual*, Network Theory Limited, 2002.

W. Kahan, *Why do we need a floating-point arithmetic standard?*, technical report, University of California, Berkeley, CA, USA, Feb. 1981.

Dirk Laurie, *Complex analogue of Householder reflections: Summary.* In NA Digest 97 #22, May 1997.

Beresford N. Parlett, *Analysis of algorithms for reflections in bisectors*, SIAM Review, 13 (1971), pp. 197–208.

Douglas M. Priest, *Efficient scaling for complex division*, ACM Transactions on Mathematical Software, 30 (2004), pp. 389–401.

G. W. Stewart, *The efficient generation of random orthogonal matrices with an application to condition estimators*, SIAM Journal on Numerical Analysis, 17 (1980), pp. 403–409.

The MathWorks, Inc., Matlab^{TM}, 2007.

Homer F. Walker, *Implementation of the GMRES method using Householder transformations*, SIAM Journal on Scientific and Statistical Computing, 9 (1988), pp. 152–163.

R. Clint Whaley and Jack J. Dongarra, *Automatically tuned linear algebra software*, tech. report, University of Tennessee, Knoxville, Jan. 1997.

A Scaling

Listing 3: Rescaling to avoid unnecessary underflows, see W. Kahan [1981] for details.

```
function [k, beta, alpha, x] = possibly_rescale (beta, alpha, x)
  global safmin; global rsafmn;
  if isempty (safmin), safmin = realmin/eps; rsafmn = 1/safmin; endif
  k = 0;
  if abs (beta) >= safmin, return; endif
  while abs (beta) < safmin,
    x *= rsafmn;
    beta *= rsafmn;
    alpha *= rsafmn;
    k += 1;
  endwhile
  xnorm = norm(x, 2);
  beta = norm([alpha; x], 2);
  k *= log2 (safmin);
endfunction
```