Updating the QR factorization and the least squares problem

Sven Hammarling and Craig Lucas

November 2008

MIMS EPrint: **2008.111**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

ISSN 1749-9097

Updating the QR Factorization and the Least Squares Problem

Sven Hammarling* Craig Lucas†

November 12, 2008

Abstract

In this paper we treat the problem of updating the QR factorization, with applications to the least squares problem. Algorithms are presented that compute the factorization $\widetilde{A} = \widetilde{Q}\widetilde{R}$ where $\widetilde{A}$ is the matrix $A = QR$ after it has had a number of rows or columns added or deleted. This is achieved by updating the factors Q and R, and we show this can be much faster than computing the factorization of $\widetilde{A}$ from scratch. We consider algorithms that exploit the Level 3 BLAS where possible and place no restriction on the dimensions of A or the number of rows and columns added or deleted. For some of our algorithms we present Fortran 77 LAPACK-style code and show the backward error of our updated factors is comparable to the error bounds of the QR factorization of $\widetilde{A}$.

1 Introduction

1.1 The QR Factorization

For $A \in \mathbb{R}^{m\times n}$ the QR factorization is given by

$$A = QR, \tag{1.1}$$

where $Q \in \mathbb{R}^{m\times m}$ is orthogonal and $R \in \mathbb{R}^{m\times n}$ is upper trapezoidal.

*Numerical Algorithms Group, Wilkinson House, Jordan Hill Road, Oxford. OX2 8DR (`sven@nag.co.uk`, `http://www.nag.co.uk/about/shammarling.asp`)

†Department of Mathematics, University of Manchester, M13 9PL, England. (`clucas@ma.man.ac.uk`, `http://www.ma.man.ac.uk/~clucas`)

1.1.1 Computing the QR Factorization

We compute the QR factorization of $A \in \mathbb{R}^{m\times n}$, $m \geq n$ and of full rank, by applying an orthogonal transformation matrix Q^T so that

$$Q^T A = R,$$

and Q is the product of orthogonal matrices chosen to transform A to be the upper triangular matrix R.

One method uses *Givens matrices* to introduce zeros below the diagonal one element at a time. A Givens matrix, $G(i,j) \in \mathbb{R}^{m\times m}$, is of the form

$$G(i,j) = \begin{array}{cc} \begin{array}{ccccc} & i & & j & \end{array} & \\ \begin{bmatrix} I & & & & \\ & c & & s & \\ & & I & & \\ & -s & & c & \\ & & & & I \end{bmatrix} & \begin{array}{c} \\ i \\ \\ j \\ \\ \end{array} \end{array}$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$ for some θ, and is therefore orthogonal.

For $x \in \mathbb{R}^n$, if we set

$$c = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}, \quad s = \frac{-x_j}{\sqrt{x_i^2 + x_j^2}},$$

then for $G(i,j)^T x = y$,

$$y_k = \begin{cases} cx_i - sx_j & k = i, \\ 0 & k = j, \\ x_k & k \neq i, j, \end{cases}$$

so only the ith and jth elements are affected. We can compute c and s by the following algorithm.

Algorithm 1.1 *This function returns scalars c and s such that* $\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} d \\ 0 \end{bmatrix}$, *where $a, b,$ and d are scalars, and $s^2 + c^2 = 1$.*

function $[c, s] =$ **givens**(a, b)
if $b = 0$
  $c = 1$
  $s = 0$
else

if abs(b) $\geq$ abs(a)
  $t = -a/b$
  $s = 1/\sqrt{1+t^2}$
  $c = st$
else
  $t = -b/a$
  $c = 1/\sqrt{1+t^2}$
  $s = ct$
end
end

Here the computation of c and s has been rearranged to avoid possible overflow.

Now to transform A to an upper trapezoidal matrix we require a Givens matrix for each subdiagonal element of A, and apply each one in a suitable order such as

$$G(n,n+1)^T \dots G(m-1,m)^T G(1,2)^T \dots G(m-1,m)^T A = Q^T A = R,$$

which is a QR factorization.

The matrix Q need not be formed explicitly. It is possible to encode c and s in a single scalar (see [9, Sec. 5.1.11]), which can then be stored in the eliminated a_{ij}.

The primary use of Givens matrices is to eliminate particular elements in a matrix. A more efficient approach for a QR factorization is to use *Householder matrices* which introduce zeros in all the subdiagonal elements of a column simultaneously.

Householder matrices, $H \in \mathbb{R}^{n\times n}$, are of the form

$$H = I - \tau v v^T, \quad \tau = \frac{2}{v^T v},$$

where the *Householder vector*, $v \in \mathbb{R}^n$, is nonzero. It is easy to see that H is symmetric and orthogonal. If y and z are distinct vectors such that $\|y\|_2 = \|z\|_2$ then there exists an H such that

$$Hy = z.$$

We can determine a Householder vector such that

$$H \begin{bmatrix} \alpha \\ x \end{bmatrix} = \begin{bmatrix} \beta \\ 0 \end{bmatrix},$$

where $x \in \mathbb{R}^{n-1}$, and α and β are scalars. By setting

$$v = \begin{bmatrix} \alpha \\ x \end{bmatrix} \pm \left\| \begin{bmatrix} \alpha \\ x \end{bmatrix} \right\|_2 e_1. \tag{1.2}$$

We then have

$$H \begin{bmatrix} \alpha \\ x \end{bmatrix} = \mp \left\| \begin{bmatrix} \alpha \\ x \end{bmatrix} \right\|_2 e_1.$$

In we choose the sign in (1.2) to be negative then β is positive. However, if $[\,\alpha \quad x^T\,]$ is close to a positive multiple of e_1, then this can give large cancellation error. So we use the formula [14]

$$v_1 = \alpha - \|\,[\,\alpha \quad x^T\,]\,\|_2 = \frac{\alpha^2 - \|\,[\,\alpha \quad x^T\,]\,\|_2^2}{\alpha + \|\,[\,\alpha \quad x^T\,]\,\|_2} = \frac{-\|x\|_2^2}{\alpha + \|\,[\,\alpha \quad x^T\,]\,\|_2}$$

to avoid this in the case when $\alpha > 0$. This is adopted in the following algorithm.

Algorithm 1.2 *For $\alpha \in R$ and $x \in \mathbb{R}^{n-1}$ this function returns a vector $v \in \mathbb{R}^{n-1}$ and a scalar τ such that $\tilde{v} = \begin{bmatrix} 1 \\ v \end{bmatrix}$ is a Householder vector, scaled so $\tilde{v}(1) = 1$ and $H = \left[I - \tau \begin{bmatrix} 1 \\ v \end{bmatrix} [\,1 \quad v^T\,] \right]$ is orthogonal, with $H \begin{bmatrix} \alpha \\ x \end{bmatrix} = \begin{bmatrix} \beta \\ 0 \end{bmatrix}$, where $\beta \in \mathbb{R}$.*

function $[v, \tau] =$ **householder**(α, x)
$s = \|x\|_2^2$
$v = x$
if $s = 0$
    $\tau = 0$
else
    $t = \sqrt{\alpha^2 + s}$
    % choose sign of v
    if $\alpha \le 0$
        $v\_one = \alpha - t$
    else
        $v\_one = -s/(\alpha + t)$
    end
    $\tau = 2v\_one^2/(s + v\_one^2)$
    $v = v/v\_one$
end

Here we have normalized v so $v_1 = 1$ and the *essential* part of the Householder vector, $v(2{:}\,n)$, can be stored in x.

Thus if we apply n Householder matrices, H_j, to introduce zeros in the subdiagonal columns one by one, we have the QR factorization

$$H_n \dots H_1 A = Q^T A = R,$$

and the H_j are such that their vectors v_j are of the form

$$\begin{aligned} v_j(1{:}\, j-1) &= 0, \\ v_j(j) &= 1, \\ v_j(j+1{:}\, m) :& \quad \text{as } v \text{ in Algorithm 1.2.} \end{aligned}$$

The algorithm requires $2n^2(m - n/3)$ flops. The essential part of the Householder vectors can be stored in the subdiagonal, and we refer to Q being in *factored form*.

If Q is to be formed explicitly we can do so with the *backward accumulation* method by computing

$$(H_1 \dots (H_{n-2}(H_{n-1}H_n))).$$

This exploits the fact that the leading $(j-1)$-by-$(j-1)$ part of H_j is the identity.

1.1.2 The Blocked QR Factorization

We can derive a blocked Householder QR factorization by using the following relationship [18]. We can write the product of p Householder matrices, $H_i = I - \tau_i v_i v_i^T$, as

$$H_1 H_2 \dots H_p = I - VTV^T,$$

where

$$V = [\, v_1 \quad v_2 \quad \dots \quad v_p \,] = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix},$$

and $V_1 \in \mathbb{R}^{p \times p}$ is lower triangular with $T = T_p$ upper triangular and defined recursively as

$$T_1 = \tau_1, \qquad T_i = \begin{bmatrix} T_{i-1} & -\tau_i T_{i-1} V(:, 1{:}\, i-1)^T v_i \\ 0 & \tau_i \end{bmatrix}, \quad i = 2{:}\, p.$$

With this representation of the Householder vectors we can derive a blocked algorithm. At each step we factor p columns of A, for some block size p. We can then apply $I - VT^TV^T$ to update the trailing matrix.

1.2 The Least Squares Problem

The linear system,

$$Ax = b,$$

where $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$ and $b \in \mathbb{R}^n$ is *overdetermined* if $m \geq n$. We can solve the *least squares problem*

$$\min_x \|Ax - b\|_2,$$

with A having full rank. We then have with the QR factorization $A = QR$ and with $d = Q^T b$,

$$\begin{aligned} \|Ax - b\|_2^2 &= \|Q^T A - Q^T b\|_2^2 \\ &= \|Rx - d\|_2^2 \\ &= \left\| \begin{bmatrix} R_1 \\ 0 \end{bmatrix} x - \begin{bmatrix} f \\ g \end{bmatrix} \right\|_2^2 \\ &= \|R_1 x - f\|_2^2 + \|g\|_2^2, . \end{aligned}$$

where $R_1 \in \mathbb{R}^{n\times n}$ is upper triangular and $f \in \mathbb{R}^n$. The minimum 2-norm solution is then found by solving $R_1 x = f$. The quantity $\|g\|_2$ is the *residual* and is zero in the case $m = n$.

Each row of the matrix A can be said to hold *observations* of the *variables* x_i, $i = 1\colon n$. An example of this is the data fitting problem. Consider the function

$$g(t_i) \approx b_i, \quad i = 1\colon m.$$

The value b_i has been observed at time t_i. We wish to find the function g that approximates the value b_i. In least squares fitting we restrict ourselves to functions of the form

$$g(t) = x_1 g_1(t) + x_2 g_2(t) + \cdots + x_n g_n(t),$$

where the functions $g_i(t)$ we call basis functions, and the coefficients x_i are to be determined. We find the coefficients by solving the least squares problem with

$$A = \begin{bmatrix} g_1(t_1) & g_2(t_1) & \cdots & g_n(t_1) \\ g_1(t_2) & g_2(t_2) & \cdots & g_n(t_2) \\ \vdots & \vdots & & \vdots \\ g_1(t_m) & g_2(t_m) & \cdots & g_n(t_m) \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}.$$

Now, it may be required to *update* the least squares solution in the case where one or more observations (rows of A) are added or deleted. For instance we could have a *sliding window* where for each new observation recorded the oldest one is deleted. The observations for a particular time period may be found to be faulty, thus a block of rows of A would need to be deleted. Also, variables (columns of A) may be added or omitted to compare the different solutions. Updating after rows and columns have been deleted is also known as *downdating*.

To solve these updated least squares problems efficiently we have the problem of updating the QR factorization efficiently, that is we wish to find $\widetilde{A} = \widetilde{Q}\widetilde{R}$, where $\widetilde{A}$ is the updated A, without recomputing the factorization from scratch. We assume that $\widetilde{A}$ has full rank. We also need to compute $\tilde{d}$ such that

$$\|\widetilde{A}x - \tilde{b}\| = \|\widetilde{R}x - \tilde{d}\|,$$

where $\tilde{b}$ is the updated b corresponding to $\widetilde{A}$ and $\tilde{d} = \widetilde{Q}^T\tilde{b}$.

2 Updating Algorithms

In this section we will examine all the cases where observations and variables are added to or deleted from the least squares problem. We derive algorithms for updating the solution of the least squares problem by updating the QR factorization of $\widetilde{A}$, in the case $m \geq n$. For completeness we have also included discussion and algorithms for updating the QR factorization only when $m < n$. In all cases we give algorithms for computing $\widetilde{Q}$ should it be required. We will assume that A and $\widetilde{A}$ have full rank.

Where possible we derive blocked algorithms to exploit the Level 3 BLAS and existing Level 3 LAPACK routines. We include LAPACK style Fortran 77 code for updating the QR factorization in the cases of adding and deleting blocks of columns.

For clarity the sines and cosines for Givens matrices and the Householder vectors are stored in separate vectors and matrices, but could be stored in the elements they eliminate. Wherever possible new data overwrites original data. All unnecessary computations have been avoided, unless otherwise stated.

We give floating point operation counts for our algorithms and compare them to the counts for the Householder QR factorization of $\widetilde{A}$.

Some of the material is based on material in [1] and [9].

2.1 Deleting Rows

2.1.1 Deleting One Row

If we wish to update the least squares problem in the case of *deleting* an *observation* we have the problem of updating the QR factorization of A having deleted the kth row, a_k^T. We can write

$$\widetilde{A} = \begin{bmatrix} A(1\colon k-1, 1\colon n) \\ A(k+1\colon m, 1\colon n) \end{bmatrix}$$

and we interpret $A(1\colon 0, 1\colon n)$ and $A(m+1\colon m, 1\colon n)$ as empty rows. We define a permutation matrix P such that

$$PA = \begin{bmatrix} a_k^T \\ A(1\colon k-1, 1\colon n) \\ A(k+1\colon m, 1\colon n) \end{bmatrix} = \begin{bmatrix} a_k^T \\ \widetilde{A} \end{bmatrix} = PQR,$$

and if q^T is the first row of PQ then we can zero $q(2\colon m)$ with $m-1$ Givens matrices, $G(i,j) \in \mathbb{R}^{m\times m}$, so that

$$G(1,2)^T \ldots G(m-1,m)^T q = \alpha e_1, \quad |\alpha| = 1, \tag{2.1}$$

since the Givens matrices are orthogonal. And we also have

$$G(1,2)^T \ldots G(m-1,m)^T R = \begin{bmatrix} v^T \\ \widetilde{R} \end{bmatrix},$$

which is upper Hessenberg, so $\widetilde{R}$ is upper trapezoidal.

So we have finally

$$\begin{aligned} PA = \begin{bmatrix} a^T \\ \widetilde{A} \end{bmatrix} &= (PQG(m-1,m)\ldots G(1,2))(G(1,2)^T \ldots G(m-1,m)^T R) \\ &= \begin{bmatrix} \alpha & 0 \\ 0 & \widetilde{Q} \end{bmatrix} \begin{bmatrix} v^T \\ \widetilde{R} \end{bmatrix}, \end{aligned}$$

and

$$\widetilde{A} = \widetilde{Q}\widetilde{R}.$$

Note that the zero column below α is forced by orthogonality. Also note the choice of a sequence of Givens matrices over one Householder matrix. If we were to use a Householder matrix then the transformed R would be full, as H is full, and not upper Hessenberg. We update b by computing

$$G(1,2)^T \ldots G(m-1,m)^T Q^T Pb = \begin{bmatrix} \nu \\ \tilde{d} \end{bmatrix}.$$

This gives the following algorithm.

Algorithm 2.1 *Given $A = QR \in \mathbb{R}^{m\times n}$, with $m \geq n$, this algorithm computes $\widetilde{Q}^T\widetilde{A} = \widetilde{R} \in \mathbb{R}^{(m-1)\times n}$ where $\widetilde{R}$ is upper trapezoidal, $\widetilde{Q}$ is orthogonal and $\widetilde{A}$ is A with the kth row deleted, $1 \leq k \leq m$, and $\tilde{d}$ such that $\|\widetilde{A}x - \tilde{b}\|_2 = \|\widetilde{R}x - \tilde{d}\|_2$, where $\tilde{b}$ is b with the kth element deleted. The residual, $\|\tilde{d}(n+1\colon m-1)\|_2$, is also computed.*

$q^T = Q(k, 1\colon m)$
if $k \neq 1$
  % Permute b
  $b(2\colon k) = b(1\colon k-1)$
end
$d = Q^T b$
for $j = m-1\colon -1\colon 1$
  $[c(j), s(j)] = \mathbf{givens}(q(j), q(j+1))$
  % Update q
  $q(j) = c(j)q(j) - s(j)q(j+1)$
  % Update R if there is a nonzero row
  if $j \leq n$

$$R(j\colon j+1, j\colon n) = \begin{bmatrix} c(j) & s(j) \\ -s(j) & c(j) \end{bmatrix}^T R(j\colon j+1, j\colon n)$$

  end
  % Update d

$$d(j\colon j+1) = \begin{bmatrix} c(j) & s(j) \\ -s(j) & c(j) \end{bmatrix}^T d(j\colon j+1)$$

end
$\widetilde{R} = R(2\colon m, 1\colon n)$
$\tilde{d} = d(2\colon m)$
% Compute the residual
$resid = \|\tilde{d}(n+1\colon m-1)\|_2$

Computing $\widetilde{R}$ requires $3n^2$ flops, versus $2n^2(m - n/3)$ for the Householder QR factorization of $\widetilde{A}$. If $\widetilde{Q}$ is required, it can be computed with the following algorithm.

Algorithm 2.2 *Given vectors c and s from Algorithm 2.1 this algorithm forms an orthogonal matrix $\widetilde{Q} \in \mathbb{R}^{(m-1)\times(m-1)}$ such that $\widetilde{A} = \widetilde{Q}\widetilde{R}$, where $\widetilde{A}$ is the matrix $A = QR$ with the kth row deleted.*

if $k \neq 1$
  % Permute Q
  $Q(2\colon k, 1\colon m) = Q(1\colon k-1, 1\colon m)$
end
for $j = m-1\colon -1\colon 2$

$$Q(2\colon m, j\colon j+1) = Q(2\colon m, j\colon j+1) \begin{bmatrix} c(j) & s(j) \\ -s(j) & c(j) \end{bmatrix}$$

end

% Do not need to update 1st column of Q
$Q(2\colon m, 2) = s(1)Q(2\colon m, 1) + c(1)Q(2\colon m, 2)$
$\widetilde{Q} = Q(2\colon m, 2\colon m)$

2.1.2 Deleting a Block of Rows

If a block of p observations is to be deleted from our least squares problem, equivalent to deleting the p rows $A(k\colon k+p-1, 1\colon n)$ from A, we would like to find an analogous method to Algorithm 2.1 that uses Householder matrices, such that if H is a product of p Householder matrices then

$$PA = \begin{bmatrix} A(k\colon k+p-1, 1\colon n) \\ \widetilde{A} \end{bmatrix} = (PQH)(HR) = \begin{bmatrix} I & 0 \\ 0 & \widetilde{Q} \end{bmatrix} \begin{bmatrix} V \\ \widetilde{R} \end{bmatrix}.$$

However as noted in the single row case, HR is full and H is chosen to introduce zeros in Q not R. Thus in order to compute $\widetilde{A} = \widetilde{Q}\widetilde{R}$ and $\tilde{d}$, we need the equivalent of p steps of Algorithm 2.1 and Algorithm 2.2, since Givens matrices only affect two rows of the matrix they are multiplying, and so we have the following algorithm.

Algorithm 2.3 *Given $A = QR \in \mathbb{R}^{m\times n}$, with $m \geq n$, this algorithm computes $\widetilde{Q}^T\widetilde{A} = \widetilde{R} \in \mathbb{R}^{(m-p)\times n}$ where $\widetilde{R}$ is upper trapezoidal, $\widetilde{Q}$ is orthogonal and $\widetilde{A}$ is A with the kth to $(k+p-1)$st rows deleted, $1 \leq k \leq m-p+1$, $1 \leq p < m$, and $\tilde{d}$ such that $\|\widetilde{A}x - \tilde{b}\|_2 = \|\widetilde{R}x - \tilde{d}\|_2$, where $\tilde{b}$ is b with the kth to $(k+p-1)$st elements deleted. The residual, $\|\tilde{d}(n+1\colon m-p)\|_2$, is also computed.*

$W = Q(k\colon k+p-1, 1\colon m)$
if $k \neq 1$
    % Permute b
    $b(p+1\colon k+p-1) = b(1\colon k-1)$
end
$d = Q^T b$
for $i = 1\colon p$
    for $j = m-1\colon -1\colon i$
        $[C(i,j), S(i,j)] =$ **givens**$(W(i,j), W(i,j+1))$
        % Update W
        $W(i,j) = W(i,j)C(i,j) - W(i,j+1)S(i,j)$
        $W(i+1\colon p, j\colon j+1) = W(i+1\colon p, j\colon j+1)\begin{bmatrix} C(i,j) & S(i,j) \\ -S(i,j) & C(i,j) \end{bmatrix}$
        % Update R if there is a nonzero row
        if $j \leq n+i-1$

$$R(j\colon j+1, j-i+1\colon n) = \begin{bmatrix} C(i,j) & S(i,j) \\ -S(i,j) & C(i,j) \end{bmatrix}^T R(j\colon j+1, j-i+1\colon n)$$

end

% Update d

$$d(j\colon j+1) = \begin{bmatrix} C(i,j) & S(i,j) \\ -S(i,j) & C(i,j) \end{bmatrix}^T d(j\colon j+1)$$

end

end

$\widetilde{R} = R(p+1\colon m, 1\colon n)$

$\tilde{d} = d(p+1\colon m)$

% Compute the residual

$resid = \|\tilde{d}(n+1\colon m-p)\|_2$

Computing $\widetilde{R}$ requires $3n^2p + p^2(m/3 - p)$ flops, versus $2n^2(m-p-n/3)$ for the Householder QR factorization of $\widetilde{A}$. Note the following algorithm to compute $\widetilde{Q}$ is more economical than calling Algorithm 2.2 p times, saving $3mp^2$ flops by not updating the first p rows of $\widetilde{Q}$.

Algorithm 2.4 *Given matrices C and S from Algorithm 2.3 this algorithm forms an orthogonal matrix $\widetilde{Q} \in \mathbb{R}^{(m-p)\times(m-p)}$ such that $\widetilde{A} = \widetilde{Q}\widetilde{R}$, where $\widetilde{A}$ is the matrix $A = QR$ with the kth to $(k+p-1)$st rows deleted.*

if $k \neq 1$

% Permute Q

$Q(p+1\colon k+p-1, 1\colon m) = Q(1\colon k-1, 1\colon m)$

end

for $i = 1\colon p$

for $j = m-1\colon -1\colon i+1$

$$Q(p+1\colon m, j\colon j+1) = Q(p+1\colon m, j\colon j+1) \begin{bmatrix} C(i,j) & S(i,j) \\ -S(i,j) & C(i,j) \end{bmatrix}$$

end

end

% Do not need to update columns $1\colon p$ of Q

$Q(p+1\colon m, i+1) = S(i,i)Q(p+1\colon m, i) + C(i,i)Q(p+1\colon m, i+1)$

$\widetilde{Q} = Q(p+1\colon m, p+1\colon m)$

2.1.3 Updating the QR Factorization for any m and n

The relevant parts of Algorithm 2.3 and Algorithm 2.4 could be used to update the QR factorization of $\widetilde{A}$ in the case when $m < n$ without any alteration.

2.2 Alternative Methods for Deleting Rows

2.2.1 Hyperbolic Transformations

If we have the QR factorization $A = QR \in \mathbb{R}^{m\times n}$, then

$$A^T A = R^T Q^T Q R = R^T R,$$

which is a Cholesky factorization of $A^T A$. And if we define a permutation matrix P such that

$$PA = \begin{bmatrix} a_k^T \\ A(1\colon k-1, 1\colon n) \\ A(k+1\colon m, 1\colon n) \end{bmatrix} = \begin{bmatrix} a^T \\ \widetilde{A} \end{bmatrix} = PQR,$$

where a_k^T is the kth row of A, then we have

$$R^T Q^T P^T P Q R = R^T R = A^T A = \begin{bmatrix} a_k^T \\ \widetilde{A} \end{bmatrix}^T \begin{bmatrix} a_k^T \\ \widetilde{A} \end{bmatrix}. \tag{2.2}$$

Thus if we find $\widetilde{R}$, such that $\widetilde{R}^T \widetilde{R} = \widetilde{A}^T \widetilde{A}$, then we have computed $\widetilde{R}$ for $\widetilde{A}$ being A with the kth row deleted. This can be achieved with *hyperbolic transformations.*

We define $W \in \mathbb{R}^{m\times m}$ as *pseudo-orthogonal* with respect to the *signature matrix*

$$J = \mathrm{diag}(\pm 1) \in \mathbb{R}^{m\times m}$$

if

$$W^T J W = J.$$

If we transform a matrix with W we say that this is a hyperbolic transformation.

Now from (2.2) we have

$$\begin{aligned} \widetilde{A}^T \widetilde{A} &= A^T A - a_k a_k^T \\ &= R^T R - a_k a_k^T \\ &= \begin{bmatrix} R^T & a_k \end{bmatrix} \begin{bmatrix} I_n & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} R \\ a_k^T \end{bmatrix}, \end{aligned}$$

with the signature matrix

$$J = \begin{bmatrix} I_n & 0 \\ 0 & -1 \end{bmatrix}. \tag{2.3}$$

And suppose there is a $W \in \mathbb{R}^{(n+1)\times(n+1)}$ such that $W^T J W = J$ with the property

$$W \begin{bmatrix} R \\ a_k^T \end{bmatrix} = \begin{bmatrix} \widetilde{R} \\ 0 \end{bmatrix}$$

is upper trapezoidal. It follows that

$$\begin{aligned}\widetilde{A}^T\widetilde{A} &= [\,R^T \quad a_k\,]\,W^TJW\begin{bmatrix} R \\ a_k^T \end{bmatrix} \\ &= [\,\widetilde{R}^T \quad 0\,]\,J\begin{bmatrix} \widetilde{R} \\ 0 \end{bmatrix} \\ &= \widetilde{R}^T\widetilde{R},\end{aligned}$$

which is the Cholesky factorization we seek.

We construct the hyperbolic transformation matrix, W, by a product of *hyperbolic rotations*, $W(i, n+1) \in \mathbb{R}^{(n+1)\times(n+1)}$, which are of the form

$$W(i,n+1) = \begin{array}{c} \begin{array}{cccc} \quad & i & \quad & n+1 \end{array} \\ \begin{bmatrix} I & & & \\ & c & & -s \\ & & I & \\ & -s & & c \end{bmatrix} \end{array} \begin{array}{c} \\ i \\ \\ n+1 \end{array}$$

where $c = \cosh(\theta)$ and $s = \sinh(\theta)$ for some θ and $c^2 - s^2 = 1$. $W(i, n+1)^T JW(i, n+1) = J$, where J is given in (2.3).

$W(i, n+1)x$ only transforms the ith and $(n+1)$st elements. To solve the 2×2 problem

$$\begin{bmatrix} c & -s \\ -s & c \end{bmatrix}\begin{bmatrix} x_i \\ x_{n+1} \end{bmatrix} = \begin{bmatrix} y \\ 0 \end{bmatrix},$$

we note that $cx_{n+1} = sx_i$ and there is no solution for $x_i = x_{n+1} \neq 0$. If $x_i \neq x_{n+1}$ then we can compute c and s with the following algorithm.

Algorithm 2.5 *This algorithm generates scalars c and s such that* $\begin{bmatrix} c & -s \\ -s & c \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y \\ 0 \end{bmatrix}$ *where x_1, x_2 and y are scalars and $c^2 - s^2 = 1$, if a solution exists.*

```
if x_2 = 0
    s = 0
    c = 1
else
    if |x_2| < |x_1|
        t = x_2/x_1
        c = 1/sqrt(1 - t^2)
        s = ct
    else
```

no solution exists

end

end

Note the norm of the rotation gets large as x_1 gets close to x_2.

We thus generate n hyperbolic transformations such that

$$W(n, n+1) \ldots W(2, n+1) W(1, n+1) \begin{bmatrix} R \\ a_k^T \end{bmatrix} = \begin{bmatrix} \widetilde{R} \\ 0 \end{bmatrix}.$$

It turns out that all the $W(i, n+1)$ can be found if A has full rank [1].

2.2.2 Chamber's Algorithm

A method due to Chambers [5] mixes a hyperbolic and Givens rotation. If we have our usual Givens transformation on the vector x

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

then the transformed x_i, $\tilde{x}_i$, are

$$\begin{aligned} \tilde{x}_1 &= cx_1 - sx_2, \\ \tilde{x}_2 &= sx_1 + cx_2, \end{aligned} \tag{2.4}$$

with

$$c = \frac{x_1}{\sqrt{x_1^2 + x_2^2}}, \quad s = \frac{-x_2}{\sqrt{x_1^2 + x_2^2}}.$$

Now suppose we know $\tilde{x}_1$ and want to recreate the vector x, then rearranging (2.4) we have

$$\begin{aligned} x_1 &= (sx_2 + \tilde{x}_1)/c, \\ \tilde{x}_2 &= sx_1 + cx_2, \end{aligned}$$

with

$$c = \frac{\sqrt{\tilde{x}_1^2 - x_2^2}}{\tilde{x}_1}, \quad s = \frac{-x_2}{\tilde{x}_1}.$$

Thus we can recreate the steps that would have updated $\widetilde{R}$ had we *added* a_k^T to $\widetilde{A}$, instead of deleting it from A. At the ith step, for $i = 1{:}\, n+1$, with

$$\begin{aligned} x_1 &= R(i,i), \\ \tilde{x}_1 &= \widetilde{R}(i,i), \\ x_2 &= a_k^{(i-1)}(i), \end{aligned}$$

we compute, for $j = i\colon n+1$:

$$\begin{aligned} \widetilde{R}(i,j) &= (R(i,i) + s a_k(j))/c, \\ a_k^{(i)}(j) &= sR(i,j) + c a_k^{(i-1)}(j). \end{aligned}$$

2.2.3 Saunders' Algorithm

If Q is not available then Saunders' algorithm [17] offers an alternative to Algorithm 2.1. The first row of

$$PA = \begin{bmatrix} a_k^T \\ \widetilde{A} \end{bmatrix} = PQ \begin{bmatrix} R_1 \\ 0 \end{bmatrix},$$

can be written

$$a_k^T = q^T \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = [\, q_1^T q_2^T \,] \begin{bmatrix} R_1 \\ 0 \end{bmatrix},$$

where $q_1 \in \mathbb{R}^n$. We compute q_1 by solving

$$R_1^T q_1 = a_k,$$

and since $\|q\|_2 = 1$ we have

$$\eta = \|q_2\|_2 = (1 - \|q_1\|_2^2)^{1/2}.$$

Then we have, with the same Givens matrices in (2.1),

$$G(n+1, n+2)^T \ldots G(m-1, m)^T \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} q_1 \\ \pm\eta \\ 0 \end{bmatrix},$$

which would not effect R. So we need only compute

$$G(1,2)^T \ldots G(n, n+1)^T \begin{bmatrix} q_1 \\ \eta \end{bmatrix} = \alpha e_1, \quad |\alpha| = 1,$$

and update R by

$$G(1,2)^T \ldots G(n, n+1)^T R = \begin{bmatrix} v^T \\ \widetilde{R} \end{bmatrix}.$$

This algorithm is implemented in LINPACK's `xCHDD`.

2.2.4 Stability Issues

Stewart [20] shows that hyperbolic transformations are not backward stable. However, Chamber's and Saunder's algorithms are *relationally stable* [3], [19], that is if W represents the product of all the transformations then

$$W^T R = \begin{bmatrix} v^T \\ \widetilde{R} \end{bmatrix} + E,$$

where

$$\|E\| \leq c_n u \|R\|,$$

and c_n is a constant that depends on n.

Saunder's algorithm can fail for certain data, see [2].

2.2.5 Block Downdating

Hyperbolic transformations have been generalized by Rader and Steinhardt as *hyperbolic Householder transformations* [15].

Alternatives are discussed in Elden and Park [8], and the references contained within, including a generalization of Saunders Algorithm. See also Bojanczyk, Higham and Patel [4] and Olskanskyj, Lebak and Bojanczyk [13].

2.3 Adding Rows

2.3.1 Adding One Row

If we wish to *add* an *observation* to our least squares problem then we need to add a row, $u^T \in \mathbb{R}^n$, in the kth position, $k = 1\colon m+1$, of $A = QR \in \mathbb{R}^{m \times n}$, $m \geq n$. We can then write

$$\widetilde{A} = \begin{bmatrix} A(1\colon k-1, 1\colon n) \\ u^T \\ A(k\colon m, 1\colon n) \end{bmatrix}$$

and we can define a permutation matrix, P, such that

$$P\widetilde{A} = \begin{bmatrix} A \\ u^T \end{bmatrix},$$

and then

$$\begin{bmatrix} Q^T & 0 \\ 0 & 1 \end{bmatrix} P\widetilde{A} = \begin{bmatrix} R \\ u^T \end{bmatrix}. \tag{2.5}$$

For example, with $m = 8$ and $n = 6$ the right-hand side of (2.5) looks like:

$$\begin{bmatrix} + & + & + & + & + & + \\ 0 & + & + & + & + & + \\ 0 & 0 & + & + & + & + \\ 0 & 0 & 0 & + & + & + \\ 0 & 0 & 0 & 0 & + & + \\ 0 & 0 & 0 & 0 & 0 & + \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \ominus & \ominus & \ominus & \ominus & \ominus & \ominus \end{bmatrix},$$

with the nonzero elements of R represented with a + and the elements to be eliminated are shown with a $\ominus$.

Thus to find $\widetilde{A} = \widetilde{Q}\widetilde{R}$, we can define n Givens matrices, $G(i,j) \in \mathbb{R}^{m+1 \times m+1}$, to eliminate $u^T$to give

$$G(n, m+1)^T \ldots G(1, m+1)^T \begin{bmatrix} R \\ u^T \end{bmatrix} = \widetilde{R},$$

so we have

$$\widetilde{A} = \left(P^T \begin{bmatrix} Q & 0 \\ 0 & 1 \end{bmatrix} G(1, m+1) \ldots G(n, m+1) \right) \widetilde{R} = \widetilde{Q}\widetilde{R}.$$

and to update b we compute

$$G(n, m+1)^T \ldots G(1, m+1)^T \begin{bmatrix} Q^T b \\ \mu \end{bmatrix} = \tilde{d},$$

where μ is the element inserted into b corresponding to u^T. This gives the following algorithm.

Algorithm 2.6 *Given $A = QR \in \mathbb{R}^{m \times n}$, with $m \geq n$, this algorithm computes $\widetilde{Q}^T\widetilde{A} = \widetilde{R} \in \mathbb{R}^{(m+1)\times n}$ where $\widetilde{R}$ is upper trapezoidal, $\widetilde{Q}$ is orthogonal and $\widetilde{A}$ is A with a row, $u^T \in \mathbb{R}^n$, inserted in the kth position, $1 \leq k \leq m+1$, and $\tilde{d}$ such that $\|\widetilde{A}x - \tilde{b}\|_2 = \|\widetilde{R}x - \tilde{d}\|_2$, where $\tilde{b}$ is b with a scalar μ inserted in the kth position. The residual, $\|\tilde{d}(n+1\colon m+1)\|_2$, is also computed.*

$d = Q^T b$
for $j = 1\colon n$
  $[c(j), s(j)] = \textbf{givens}(R(j,j), u(j))$
  $R(j,j) = c(j)R(j,j) - s(j)u(j)$

% Update jth row of R and u
$t1 = R(j, j+1\colon n)$
$t2 = u(j+1\colon n)$
$R(j, j+1\colon n) = c(j)t1 - s(j)t2$
$u(j+1\colon n) = s(j)t1 + c(j)t2$
% Update jth row of d and μ
$t1 = d(j)$
$t2 = \mu$
$d(j) = c(j)t1 - s(j)t2$
$\mu = s(j)t1 + c(j)t2$
end
$\widetilde{R} = \begin{bmatrix} R \\ 0 \end{bmatrix}$
$\tilde{d} = \begin{bmatrix} d \\ \mu \end{bmatrix}$
% Compute the residual
$resid = \|\tilde{d}(n+1\colon m+1)\|_2$

Computing $\widetilde{R}$ requires $3n^2$ flops, versus $2n^2(m - n/3)$ for the Householder QR factorization of $\widetilde{A}$. If $\widetilde{Q}$ is required, it can be computed with the following algorithm.

Algorithm 2.7 *Given vectors c and s from Algorithm 2.6 this algorithm forms an orthogonal matrix $\widetilde{Q} \in \mathbb{R}^{(m+1)\times(m+1)}$ such that $\widetilde{A} = \widetilde{Q}\widetilde{R}$, where $\widetilde{A}$ is the matrix $A = QR$ with a row added in the kth position.*

Set $\widetilde{Q} = \begin{bmatrix} Q & 0 \\ 0 & 1 \end{bmatrix}$
if $k \neq m+1$
% Permute Q
$Q = \begin{bmatrix} Q(1\colon k-1, 1\colon n) \\ Q(m+1, 1\colon n) \\ Q(k\colon m, 1\colon n) \end{bmatrix}$
end
for $j = 1\colon n$
$t1 = \widetilde{Q}(1\colon m+1, j)$
$t2 = \widetilde{Q}(1\colon m+1, m+1)$
$\widetilde{Q}(1\colon m+1, j) = c(j)t1 - s(j)t2$
$\widetilde{Q}(1\colon m+1, m+1) = s(j)t1 + c(j)t2$
end

2.3.2 Adding a Block of Rows

To add a block of p observations to our least squares problem we add a block of p rows, $U \in \mathbb{R}^{(p\times n)}$, in the kth to $(k+p-1)$st positions, $k = 1{:}\,m+1$, of $A = QR \in \mathbb{R}^{m\times n}$, $m \geq n$, we can then write

$$\widetilde{A} = \begin{bmatrix} A(1{:}\,k-1, 1{:}\,n) \\ U \\ A(k{:}\,m, 1{:}\,n) \end{bmatrix}$$

and we can define a permutation matrix, P, such that

$$P\widetilde{A} = \begin{bmatrix} A \\ U \end{bmatrix},$$

and

$$\begin{bmatrix} Q^T & 0 \\ 0 & I_p \end{bmatrix} P\widetilde{A} = \begin{bmatrix} R \\ U \end{bmatrix}. \tag{2.6}$$

For example, with $m = 8$, $n = 6$ and $p = 3$ the right-hand side of Equation (2.6) looks like:

$$\begin{bmatrix} + & + & + & + & + & + \\ 0 & + & + & + & + & + \\ 0 & 0 & + & + & + & + \\ 0 & 0 & 0 & + & + & + \\ 0 & 0 & 0 & 0 & + & + \\ 0 & 0 & 0 & 0 & 0 & + \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \ominus & \ominus & \ominus & \ominus & \ominus & \ominus \\ \ominus & \ominus & \ominus & \ominus & \ominus & \ominus \\ \ominus & \ominus & \ominus & \ominus & \ominus & \ominus \end{bmatrix},$$

with the nonzero elements of R represented with a $+$ and the elements to be eliminated are shown with a $\ominus$.

Thus to find $\widetilde{A} = \widetilde{Q}\widetilde{R}$, we can define n Householder matrices to eliminate U to give

$$H_n \dots H_1 \begin{bmatrix} R \\ U \end{bmatrix} = \widetilde{R},$$

so we have

$$\widetilde{A} = \left(P^T \begin{bmatrix} Q & 0 \\ 0 & I_p \end{bmatrix} H_1 \dots H_n \right) \widetilde{R} = \widetilde{Q}\widetilde{R}.$$

The Householder matrix, $H_j \in \mathbb{R}^{(m+p)\times(m+p)}$, will zero the jth column of U. Its associated Householder vector, $v_j \in \mathbb{R}^{(m+p)}$, is such that

$$\left.\begin{array}{rcl} v_j(1\colon j-1) & = & 0, \\ v_j(j) & = & 1, \\ v_j(j+1\colon m) & = & 0, \\ v_j(m+1\colon m+p) & = & x/(r_{jj} - \|\,[\,r_{jj} \quad x^T\,]\,\|_2), \text{ where } x = U(1\colon p, j). \end{array}\right\} \quad (2.7)$$

So the H_j have the following structure

$$H_j = \begin{bmatrix} I & & & \\ & h_{jj} & & [\,h_{j,m+1} \quad \ldots \quad h_{j,m+p}\,] \\ & & I & \\ & \begin{bmatrix} h_{m+1,j} \\ \vdots \\ h_{m+p,j} \end{bmatrix} & & \begin{bmatrix} h_{m+1,m+1} & \ldots & h_{m+1,m+p} \\ \vdots & & \vdots \\ h_{m+p,m+1} & \ldots & h_{m+p,m+p} \end{bmatrix} \end{bmatrix}.$$

Then to update b we compute

$$H_n \ldots H_1 \begin{bmatrix} Q^T b \\ e \end{bmatrix} = \tilde{d},$$

where e is such that $Ux = e$. This gives the following algorithm.

Algorithm 2.8 *Given $A = QR \in \mathbb{R}^{m\times n}$, with $m \geq n$, this algorithm computes $\widetilde{Q}^T \widetilde{A} = \widetilde{R} \in \mathbb{R}^{(m+p)\times n}$ where $\widetilde{R}$ is upper trapezoidal, $\widetilde{Q}$ is orthogonal and $\widetilde{A}$ is A with a block of rows, $U \in \mathbb{R}^{p\times n}$, inserted in the kth to $(k+p-1)$st positions, $1 \leq k \leq m+1$, $p \geq 1$, and $\tilde{d}$ such that $\|\widetilde{A}x - \tilde{b}\|_2 = \|\widetilde{R}x - \tilde{d}\|_2$, where $\tilde{b}$ is b with the vector e inserted in the kth to $(k+p-1)$st positions. The residual, $\|\tilde{d}(n+1\colon m+p)\|_2$, is also computed.*

$d = Q^T b$
for $j = 1\colon n$
    $[V(1\colon p, j), \tau(j)] =$ **householder**$(R(j,j), U(1\colon p, j))$
    % Remember old jth row of R
    $R_j = R(j, j+1\colon n)$
    % Update jth row of R
    $R(j, j\colon n) = (1-\tau(j))R(j, j\colon n) - \tau(j)V(1\colon p, j)^T U(1\colon p, j\colon n)$
    % Update trailing part if U
    if $j < n$

$$U(1\colon p, j+1\colon n) = U(1\colon p, j+1\colon n) - \tau(j)V(1\colon p, j)R_j$$
$$-\tau(j)V(1\colon p, j)(V(1\colon p, j)^T U(1\colon p, j+1\colon n))$$

end

% Remember old jth element of d

$d_j = R(j)$

% Update jth element of d

$d(j) = (1 - \tau(j))d(j) - \tau(j)V(1\colon p, j)^T e(1\colon p)$

% Update e

$$e(1\colon p) = e(1\colon p) - \tau(j)V(1\colon p, j)d_j$$
$$-\tau(j)V(1\colon p, j)(V(1\colon p, j)^T e(1\colon p))$$

end

$$\widetilde{R} = \begin{bmatrix} R \\ 0 \end{bmatrix}$$

$$\tilde{d} = \begin{bmatrix} d \\ e \end{bmatrix}$$

% Compute the residual

$resid = \|\tilde{d}(n+1\colon m+p)\|_2$

Computing $\widetilde{R}$ requires $2n^2p$ flops, versus $2n^2(m+p-n/3)$ for the Householder QR factorization of $\widetilde{A}$. If $\widetilde{Q}$ is required, it can be computed with the following algorithm.

Algorithm 2.9 *Given the matrix V and vector τ from Algorithm 2.8 this algorithm forms an orthogonal matrix $\widetilde{Q} \in \mathbb{R}^{(m+p)\times(m+p)}$ such that $\widetilde{A} = \widetilde{Q}\widetilde{R}$, where $\widetilde{A}$ is the matrix $A = QR$ with a block of rows inserted in the kth to $(k+p-1)$st positions.*

Set $\widetilde{Q} = \begin{bmatrix} Q & 0 \\ 0 & I \end{bmatrix}$

if $k \neq m+1$

% Permute Q

$$\widetilde{Q} = \begin{bmatrix} \widetilde{Q}(1\colon k-1, 1\colon m+p) \\ \widetilde{Q}(m+1\colon m+p, 1\colon m+p) \\ \widetilde{Q}(k\colon m, 1\colon m+p) \end{bmatrix}$$

end

for $j = 1\colon n$

% Remember jth column of $\widetilde{Q}$

$\widetilde{Q}_k = \widetilde{Q}(1\colon m+p, j)$

% Update jth column

$$Q(1\colon m+p, j) = Q(1\colon m+p, j)(1 - \tau(j)) -$$
$$\widetilde{Q}(1\colon m+p, m+1\colon m+p)\tau(j)V(1\colon p, j)$$

$$
\begin{aligned}
&\% \text{ Update } m+1{:}p \text{ columns of } \widetilde{Q}\\
&\widetilde{Q}(1{:}m+p, m+1{:}m+p) = \widetilde{Q}(1{:}m+p, m+1{:}m+p)\\
&\qquad\qquad -\tau(j)\widetilde{Q}_k V(1{:}p, j)^T\\
&\qquad\qquad -\tau(j)(\widetilde{Q}(1{:}m+p, m+1{:}m+p)V(1{:}p, j))V(1{:}p, j)^T
\end{aligned}
$$

end

This algorithm could be made more economical by noting that at the jth stage, for $i > m$, $\tilde{q}_{ij} = 0$, and avoiding some unnecessary multiplications by zero. Also $\widetilde{Q}(m+1{:}m+p, 1{:}m) = 0$ and $\widetilde{Q}(m+1{:}m+p, m+1{:}m+p) = I_p$, prior to the permutation.

Algorithm 2.8 can be improved by exploiting the Level 3 BLAS by using the representation of the product of n_b Householder matrices, H_i, as

$$
H_1 H_2 \dots H_{n_b} = I - VTV^T, \tag{2.8}
$$

where

$$
V = [\, v_1 \quad v_2 \quad \dots \quad v_{n_b} \,] = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix},
$$

and $V_1 \in \mathbb{R}^{n_b \times n_b}$ is lower triangular and T is upper triangular. We can write

$$
Q^T \widetilde{A} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \\ 0 & 0 & 0 \\ U_{11} & U_{12} & U_{13} \end{bmatrix}, \tag{2.9}
$$

where the R_{ii} are upper triangular with $R_{11} \in \mathbb{R}^{r \times r}$ and $R_{22} \in \mathbb{R}^{n_b \times n_b}$, and after we have updated the first r columns, then the transformed right-hand side of (2.9) looks like:

$$
\begin{bmatrix} R_{11}^{(r)} & R_{12}^{(r)} & R_{13}^{(r)} \\ 0 & R_{22}^{(r)} & R_{23}^{(r)} \\ 0 & 0 & R_{33}^{(r)} \\ 0 & 0 & 0 \\ 0 & U_{12}^{(r)} & U_{13}^{(r)} \end{bmatrix}.
$$

Now we eliminate the first column of $U_{12}^{(r)}$ and instead of updating the trailing parts of R and U we update only the trailing parts of $U_{12}^{(r)}$ and the $(r+1)$st row of $R_{22}^{(r)}$, which are the only elements affected in this middle block column, and continue in this way until U_{12} has been eliminated. We can then employ the representation (2.8) to apply n_b Householder matrices to update the last block column in one go. We have, by the definition of the Householder vectors in (2.7)

$$
V = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}, \quad V_1 = I_{n_b}, \quad V_2 = \begin{bmatrix} 0 \\ \overline{V}_2 \end{bmatrix},
$$

where $\overline{V}_2 \in \mathbb{R}^{p\times n_b}$ hold the essential part of the Householder vectors for the current block column. Then

$$[\, I - VT^TV^T \,]^T \begin{bmatrix} R_{23} \\ R_{33} \\ 0 \\ U_{13} \end{bmatrix} = \begin{bmatrix} I_{m+p-r} - \begin{bmatrix} I_{n_b} \\ 0 \\ 0 \\ \overline{V}_2 \end{bmatrix} T^T [\, I_{n_b} \quad 0 \quad 0 \quad \overline{V}_2^T \,] \end{bmatrix} \begin{bmatrix} R_{23} \\ R_{33} \\ 0 \\ U_{13} \end{bmatrix}$$

$$= \begin{bmatrix} (I_{n_b} - T^T)R_{23} - T^T\overline{V}_2^T U_{13} \\ R_{33} \\ 0 \\ -\overline{V}_2 T^T R_{23} + (I - \overline{V}_2 T^T \overline{V}_2^T) U_{13} \end{bmatrix},$$

This approach leads to a blocked algorithm, where at the kth stage we factorize $[\, R_{22}^T \quad 0 \quad U_{12}^T \,]^T$, where $R_{22} \in \mathbb{R}^{n_b\times n_b}$ and $U_{12} \in \mathbb{R}^{p\times n_b}$, then update $R_{23} \in \mathbb{R}^{n_b\times(n-kn_b)}$ and $U_{13}^T \in \mathbb{R}^{p\times(n-kn_b)}$ as above. And to update $Q^Tb = d$ we compute

$$\begin{bmatrix} d(1\colon(k-1)n_b) \\ (I_{n_b} - T^T)d((k-1)n_b + 1\colon kn_b) - T^T\overline{V}_2^T e \\ d(kn_b + 1\colon m) \\ -\overline{V}_2 T^T d((k-1)n_b + 1\colon kn_b) + (I - \overline{V}_2 T^T \overline{V}_2^T)e \end{bmatrix} = \begin{bmatrix} d \\ e \end{bmatrix}.$$

Algorithm 2.10 *Given $A = QR \in \mathbb{R}^{m\times n}$, with $m \geq n$, this algorithm computes $\widetilde{Q}^T\widetilde{A} = \widetilde{R} \in \mathbb{R}^{(m+p)\times n}$ where $\widetilde{R}$ is upper trapezoidal, $\widetilde{Q}$ is orthogonal and $\widetilde{A}$ is A with a block of rows, $U \in \mathbb{R}^{p\times n}$, inserted in the kth to $(k+p-1)$st positions, $1 \leq k \leq m+1$, $p \geq 1$, and $\tilde{d}$ such that $\|\widetilde{A}x - \tilde{b}\|_2 = \|\widetilde{R}x - \tilde{d}\|_2$, where $\tilde{b}$ is b with the vector e inserted in the kth to $(k+p-1)$st positions. The residual, $\|\tilde{d}(n+1\colon m+p)\|_2$, is also computed. This is a Level 3 BLAS algorithm with block size n_b.*

$d = Q^Tb$
for $k = 1\colon n_b\colon n$
  % Check for the last column block
  $jb = \min(n_b, n-k+1)$
  Factorize current block with Algorithm 2.8 where
  V is $V(1\colon p, k\colon k+jb-1)$
  % If we are not in last block column build T
  % and update trailing matrix
  if $k + jb \leq n$
    for $j = k\colon k+jb-1$
      % Build T

```
            if $j = k$
                $T(1,1) = \tau(j)$
            else
                $T(1\colon j-k, j-k+1) = -\tau(j)T(1\colon j-k, 1\colon j-k)$
                                $*V(1\colon p, k\colon j-1)^T V(1\colon p, j)$
                $T(j-k+1, j-k+1) = \tau(j)$
            end
        end
        % Compute products we use more than once
        $T_V = T^T V(1\colon p, k\colon k+jb-1)^T$
        $T_e = T_V e$
        $T_U = T_V U(1\colon p,, k+jb\colon n)$
        % Remember old $d$ and $e$
        $d_k = d(k\colon k+jb-1)$
        $e_k = e$
        % Update $d$ and $e$
        $d(k\colon k+jb-1) = d_k - T^T d_k - T_e$
        $e = -V(1\colon p, k\colon k+jb-1)T^T d_k + e_k$
                        $-V(1\colon p, k\colon k+jb-1)T_e$
        % Remember old trailing parts of $R$ and $U$
        $R_k = R(k\colon k+jb-1, k+jb\colon n)$
        $U_k = U(1\colon p, k+jb\colon n)$
        % Update trailing parts of $R$ and $U$
        $R(k\colon k+jb-1, k+jb\colon n) = R_k - T^T R_k - T_U$
        $U(1\colon p, k+jb\colon n) = -V(1\colon p, k\colon k+jb-1)T^T R_k + U_k$
                        $-V(1\colon p, k\colon k+jb-1)T_U$
    end
end
$\widetilde{R} = \begin{bmatrix} R \\ 0 \end{bmatrix}$
$\tilde{d} = \begin{bmatrix} d \\ e \end{bmatrix}$
% Compute the residual
$resid = \|\tilde{d}(n+1\colon m+p)\|_2$
```

We could apply the same approach to improve Algorithm 2.9.

2.3.3 Updating the QR Factorization for any m and n

In the case where $m < n$ after m steps of Algorithm 2.8 we have

$$\widetilde{A} = P^T \begin{bmatrix} Q^T & 0 \\ 0 & I_p \end{bmatrix} H_1 \dots H_n \begin{bmatrix} R_{11} & R_{22} \\ 0 & V \end{bmatrix},$$

where R_{11} is upper triangular and V is the transformed $U(1\colon p, m+1\colon n)$. Thus if we compute the QR factorization $V = Q_V R_V$, we than have

$$\widetilde{A} = \left(P^T \begin{bmatrix} Q^T & 0 \\ 0 & I_p \end{bmatrix} H_1 \dots H_n \begin{bmatrix} I_m & 0 \\ 0 & Q_V^T \end{bmatrix} \right) \widetilde{R} = \widetilde{Q}\widetilde{R}.$$

This gives us the following algorithms to update the QR factorization for any m and n.

Algorithm 2.11 *Given $A = QR \in \mathbb{R}^{m\times n}$ this algorithm computes $\widetilde{Q}^T\widetilde{A} = \widetilde{R} \in \mathbb{R}^{(m+p)\times n}$ where $\widetilde{R}$ is upper trapezoidal, $\widetilde{Q}$ is orthogonal and $\widetilde{A}$ is A with a block of rows, $U \in \mathbb{R}^{p\times n}$, inserted in the kth to $(k+p-1)$st positions, $1 \le k \le m+1$, $p \ge 1$.*

> $lim = \min(m, n)$
> for $j = 1\colon lim$
> $[V(1\colon p, j), \tau(j)] = \textbf{householder}(R(j,j), U(1\colon p, j))$
>   % Remember old jth row of R
> $R_k = R(j, j+1\colon n)$
>   % Update jth row of R
> $R(j, j\colon n) = (1-\tau(j))R(j, j\colon n) - \tau(j)V(1\colon p, j)^T U(1\colon p, j\colon n)$
>   % Update trailing part if U
>   if $j < n$
> $U(1\colon p, j+1\colon n) = U(1\colon p, j+1\colon n) - \tau(j)V(1\colon p, j)R_k$
> $-\tau(j)V(1\colon p, j)(V(1\colon p, j)^T U(1\colon p, j+1\colon n))$
>   end
> end
> $\widetilde{R} = \begin{bmatrix} R \\ 0 \end{bmatrix}$
> if $m < n$
>   Perform the QR factorization $U(:, m+1\colon n) = Q_U R_U$
> $\widetilde{R}(m+1\colon m+p, m+1\colon n) = R_U$
> end

This algorithm could also be improved by using the representation (2.8) to include a Level 3 BLAS part. If $\widetilde{Q}$ is required it can be computed with the following algorithm.

Algorithm 2.12 *Given the matrices V and Q_U and vector τ from Algorithm 2.11 this algorithm forms an orthogonal matrix $\widetilde{Q} \in \mathbb{R}^{(m+p)\times(m+p)}$ such that $\widetilde{A} = \widetilde{Q}\widetilde{R}$, where $\widetilde{A}$ is the matrix $A = QR$ with a block of rows inserted in the kth to $(k+p-1)$st positions.*

Set $\widetilde{Q} = \begin{bmatrix} Q & 0 \\ 0 & I \end{bmatrix}$
if $k \neq m+1$
  % Permute Q
  $\widetilde{Q} = \begin{bmatrix} \widetilde{Q}(1{:}\,k-1, 1{:}\,m+p) \\ \widetilde{Q}(m+1{:}\,m+p, 1{:}\,m+p) \\ \widetilde{Q}(k{:}\,m, 1{:}\,m+p) \end{bmatrix}$
end
$lim = \min(m, n)$
for $j = 1{:}\,lim$
  % Remember jth column of $\widetilde{Q}$
  $\widetilde{Q}_k = \widetilde{Q}(1{:}\,m+p, j)$
  % Update jth column
  $Q(1{:}\,m+p, j) = Q(1{:}\,m+p, j)(1 - \tau(j))$
    $-\widetilde{Q}(1{:}\,m+p, m+1{:}\,m+p)\tau(j)V(1{:}\,p, j)$
  % Update $m+1{:}\,p$ columns of $\widetilde{Q}$
  $\widetilde{Q}(1{:}\,m+p, m+1{:}\,m+p) = \widetilde{Q}(1{:}\,m+p, m+1{:}\,m+p)$
    $-\tau(j)\widetilde{Q}_k V(1{:}\,p, j)^T$
    $-\tau(j)(\widetilde{Q}(1{:}\,m+p, m+1{:}\,m+p)V(1{:}\,p, j))V(1{:}\,p, j)^T$
end
if $m < n$
  $Q(1{:}\,m+p, m+1{:}\,m+p) = Q(1{:}\,m+p, m+1{:}\,m+p)Q_U$
end

2.4 Deleting Columns

2.4.1 Deleting One Column

If we wish to *delete* a *variable* from our least squares problem then we have the problem of updating the QR factorization of A where we delete the kth column, $k \neq n$, from A, we can write

$$\widetilde{A} = [\, A(1{:}\,m, 1{:}\,k-1) \quad A(1{:}\,m, k+1{:}\,n) \,]$$

then

$$Q^T\widetilde{A} = [\, R(1{:}\,m, 1{:}\,k-1) \quad R(1{:}\,m, k+1{:}\,n) \,]. \tag{2.10}$$

For example, with $m = 8$, $n = 6$ and $k = 3$ the right-hand side of Equation (2.10) looks like:

$$\begin{bmatrix} + & + & + & + & + \\ 0 & + & + & + & + \\ 0 & 0 & + & + & + \\ 0 & 0 & \ominus & + & + \\ 0 & 0 & 0 & \ominus & + \\ 0 & 0 & 0 & 0 & \ominus \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

with the nonzero elements to remain represented with a $+$ and the elements to be eliminated are shown with a $\ominus$.

Thus we can define $n-k$ Givens matrices, $G(i,j) \in \mathbb{R}^{m\times m}$, to eliminate the subdiagonal elements of $Q^T\widetilde{A}$ to give

$$(G(n,n+1)^T \ldots G(k,k+1)^T Q^T)\widetilde{A} = \widetilde{Q}^T\widetilde{A} = \widetilde{R},$$

where $\widetilde{R} \in \mathbb{R}^{m\times(n-1)}$ is upper trapezoidal and $\widetilde{Q} \in \mathbb{R}^{m\times m}$ is orthogonal, and to update b we compute

$$G(n,n+1)^T \ldots G(k,k+1)^T Q^T b = \tilde{d}.$$

This gives the following algorithm.

Algorithm 2.13 *Given $A = QR \in \mathbb{R}^{m\times n}$, with $m \geq n$, this algorithm computes $\widetilde{Q}^T\widetilde{A} = \widetilde{R} \in \mathbb{R}^{m\times(n-1)}$ where $\widetilde{R}$ is upper trapezoidal, $\widetilde{Q}$ is orthogonal and $\widetilde{A}$ is A with the kth column deleted, $1 \leq k \leq n-1$, and $\tilde{d}$ such that $\|\widetilde{A}x - b\|_2 = \|\widetilde{R}x - \tilde{d}\|_2$. The residual, $\|\tilde{d}(n+1{:}\,m)\|_2$, is also computed.*

$\tilde{d} = Q^T b$
set $R(1{:}\,m, k{:}\,n-1) = R(1{:}\,m, k+1{:}\,n)$
for $j = k{:}\,n-1$
    $[c(j), s(j)] = \mathbf{givens}(R(j,j), R(j+1,j))$
    % Update R
    $R(j,j) = c(j)R(j,j) - s(j)R(j+1,j)$
    $R(j{:}\,j+1, j+1{:}\,n-1) = \begin{bmatrix} c(j) & s(j) \\ -s(j) & c(j) \end{bmatrix}^T R(j{:}\,j+1, j+1{:}\,n-1)$
    % Update $\tilde{d}$
    $\tilde{d}(j{:}\,j+1) = \begin{bmatrix} c(j) & s(j) \\ -s(j) & c(j) \end{bmatrix}^T \tilde{d}(j{:}\,j+1)$
end

$\widetilde{R} =$ upper triangular part of $R(1\colon m, 1\colon n-1)$
% Compute the residual
$resid = \|\tilde{d}(n+1\colon m)\|_2$

Computing $\widetilde{R}$ requires $n^2/2 - nk + k^2/2$ flops, versus $2n^2(m - n/3)$ for the Householder QR factorization of $\widetilde{A}$. If $\widetilde{Q}$ is required, it can be computed with the following algorithm.

Algorithm 2.14 *Given vectors c and s from Algorithm 2.13 this algorithm forms an orthogonal matrix $\widetilde{Q} \in \mathbb{R}^{m\times m}$ such that $\widetilde{A} = \widetilde{Q}\widetilde{R}$, where $\widetilde{A}$ is the matrix $A = QR$ with the kth column deleted.*

for $j = k\colon n-1$
$$Q(1\colon m, j\colon j+1) = Q(1\colon m, j\colon j+1)\begin{bmatrix} c(j) & s(j) \\ -s(j) & c(j) \end{bmatrix}$$
end
$\widetilde{Q} = Q$

In the case when $k = n$ then

$$\widetilde{A} = A(1\colon m, 1\colon k-1), \quad \widetilde{R} = R(1\colon m, 1\colon k-1), \quad \widetilde{Q} = Q, \text{ and } \tilde{d} = Q^T b,$$

and there is no computation to do.

2.4.2 Deleting a Block of Columns

To delete a block of p variables from our least squares problem we delete a block of p columns, from the kth column onwards, from A and we can write

$$\widetilde{A} = [\, A(1\colon m, 1\colon k-1) \quad A(1\colon m, k+p\colon n) \,]$$

then

$$Q^T\widetilde{A} = [\, R(1\colon m, 1\colon k-1) \quad R(1\colon m, k+p\colon n) \,]\,. \tag{2.11}$$

For example, with $m = 10$, $n = 8$, $k = 3$ and $p = 3$ the right-hand side of Equation (2.11) looks like:

$$\begin{bmatrix} + & + & + & + & + \\ 0 & + & + & + & + \\ 0 & 0 & + & + & + \\ 0 & 0 & \ominus & + & + \\ 0 & 0 & \ominus & \ominus & + \\ 0 & 0 & \ominus & \ominus & \ominus \\ 0 & 0 & 0 & \ominus & \ominus \\ 0 & 0 & 0 & 0 & \ominus \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

with the nonzero elements to remain represented with a $+$ and the elements to be eliminated are shown with a $\ominus$.

Thus we can define $n-p-k+1$ Householder matrices, $H_j \in \mathbb{R}^{m\times m}$, with associated Householder vectors, $v_j \in \mathbb{R}^{(p+1)}$ such that

$$\begin{aligned} v_j(1\colon j-1) &= 0, \\ v_j(j) &= 1, \\ v_j(j+1\colon j+p) &= x/((\widetilde{Q}^T\widetilde{A})_{jj} - \|\,[\,(\widetilde{Q}^T\widetilde{A})_{jj} \quad x^T\,]\,\|_2), \\ &\ \text{where } x = Q^T\widetilde{A}(j+1\colon j+p, j), \\ v_j(j+p+1\colon m) &= 0. \end{aligned}$$

The H_j have the following structure

$$H_j = \begin{bmatrix} I & & \\ & \begin{bmatrix} h_{j,j} & \dots & h_{j,j+p} \\ \vdots & & \vdots \\ h_{j+p,j} & \dots & h_{j+p,j+p} \end{bmatrix} & \\ & & I \end{bmatrix},$$

and can be used to eliminate the subdiagonal of $Q^T\widetilde{A}$ to give

$$(H_{n-p}\dots H_k Q^T)\widetilde{A} = \widetilde{Q}^T\widetilde{A} = \widetilde{R},$$

where $\widetilde{R} \in \mathbb{R}^{m\times(n-p)}$ is upper trapezoidal and $\widetilde{Q} \in \mathbb{R}^{m\times m}$ is orthogonal, and we update b by computing

$$H_{n-p}\dots H_k Q^T b = \tilde{d}.$$

This gives the following algorithm.

Algorithm 2.15 *Given $A = QR \in \mathbb{R}^{m\times n}$, with $m \geq n$, this algorithm computes $\widetilde{Q}^T\widetilde{A} = \widetilde{R} \in \mathbb{R}^{m\times(n-p)}$ where $\widetilde{R}$ is upper trapezoidal, $\widetilde{Q}$ is orthogonal and $\widetilde{A}$ is A with the kth to $(k+p-1)$st columns deleted, $1 \leq k \leq n-p$, $1 \leq p < n$, and $\tilde{d}$ such that $\|\widetilde{A}x - b\|_2 = \|\widetilde{R}x - \tilde{d}\|_2$. The residual, $\|\tilde{d}(n+1{:}\,m)\|_2$, is also computed.*

$$
\begin{array}{l}
\tilde{d} = Q^T b \\
\text{set } R(1{:}\,m, k{:}\,n-p) = R(1{:}\,m, k+p{:}\,n) \\
\text{for } j = k{:}\,n-p \\
\quad [V(1{:}\,p, j), \tau(j)] = \textbf{householder}(R(j,j), R(j+1{:}\,j+p, j)) \\
\quad \%\ \text{Update } R \\
\quad R(j,j) = R(j,j) - \tau(j)R(j,j) - \tau(j)V(1{:}\,p,j)^T R(j+1{:}\,j+p, j) \\
\quad \text{if } j < n-p \\
\qquad R(j{:}\,j+p, j+1{:}\,n-p) = R(j{:}\,j+p, j+1{:}\,n-p) \\
\qquad -\tau(j)\begin{bmatrix} 1 \\ V(1{:}\,p,j) \end{bmatrix}\left([\,1 \quad V(1{:}\,p,j)^T\,]\, R(j{:}\,j+p, j+1{:}\,n-p)\right) \\
\quad \text{end} \\
\quad \%\ \text{Update } \tilde{d} \\
\quad \tilde{d}(j{:}\,j+p) = \tilde{d}(j{:}\,j+p, j+1) \\
\qquad\qquad\qquad -\tau(j)\begin{bmatrix} 1 \\ V(1{:}\,p,j) \end{bmatrix}\left([\,1 \quad V(1{:}\,p,j)^T\,]\, \tilde{d}(j{:}\,j+p)\right) \\
\text{end} \\
\widetilde{R} = \text{ upper triangular part of } R(1{:}\,m, 1{:}\,n-p) \\
\%\ \text{Compute the residual} \\
resid = \|\tilde{d}(n+1{:}\,m)\|_2
\end{array}
$$

Computing $\widetilde{R}$ requires $4(np(n/2-p-k) + p^2(p/2+k) + pk^2$ flops, versus $2(n-p)^2(m-(n-p)/3)$ for the Householder QR factorization of $\widetilde{A}$. If $\widetilde{Q}$ is required, it can be computed with the following algorithm.

Algorithm 2.16 *Given the matrix V and vector τ from Algorithm 2.15 this algorithm forms an orthogonal matrix $\widetilde{Q} \in \mathbb{R}^{m\times m}$ such that $\widetilde{A} = \widetilde{Q}\widetilde{R}$, where $\widetilde{A}$ is the matrix $A = QR$ with the kth to $(k+p-1)$st columns deleted.*

$$
\begin{array}{l}
\text{for } j = k{:}\,n-p \\
Q(1{:}\,m, j{:}\,j+p) = Q(1{:}\,m, j{:}\,j+p) \\
\qquad\qquad -\tau(j)\Big(Q(1{:}\,m, j{:}\,j+p)\begin{bmatrix} 1 \\ V(1{:}\,p,j) \end{bmatrix}\Big)\,[\,1 \quad V(1{:}\,p,j)^T\,] \\
\text{end} \\
\widetilde{Q} = Q
\end{array}
$$

In the case when $k = n - p + 1$ then

$$\widetilde{A} = A(1{:}\,m, 1{:}\,k-1), \quad \widetilde{R} = R(1{:}\,m, 1{:}\,k-1), \quad \widetilde{Q} = Q, \text{ and } \tilde{d} = Q^T b,$$

and there is no computation to do.

2.4.3 Updating the QR Factorization for any m and n

In the case when $m < n$ we need to:

- Increase the number of steps, we introduce lim, the last column to be updated.
- Determine the last index of the Householder vectors, which cannot exceed m.

This gives the following algorithms to update the QR factorization for any m and n.

Algorithm 2.17 *Given $A = QR \in \mathbb{R}^{m\times n}$ this algorithm computes $\widetilde{Q}^T\widetilde{A} = \widetilde{R} \in \mathbb{R}^{m\times(n-p)}$ where $\widetilde{R}$ is upper trapezoidal, $\widetilde{Q}$ is orthogonal and $\widetilde{A}$ is A with the kth to $(k+p-1)$st columns deleted,* $1 \le k \le \min(m-1, n-p)$, $1 \le p < n$.

set $R(1{:}\,m, k{:}\,n-p) = R(1{:}\,m, k+p{:}\,n)$
$lim = \min(m-1, n-p)$
for $j = k{:}\,lim$
$last = \min(j+p, m)$
  $[V(1{:}\,last-j, j), \tau(j)] = \textbf{householder}(R(j,j), R(j+1{:}\,last, j))$
  % Update R
  $R(j,j) = R(j,j) - \tau(j)R(j,j) - \tau(j)V(1{:}\,last-j, j)^T R(j+1{:}\,last, j)$
  if $j < n - p$
    $R(j{:}\,last, j+1{:}\,n-p) = R(j{:}\,last, j+1{:}\,n-p)$
      $-\tau(j)\begin{bmatrix} 1 \\ V(1{:}\,last-j, j)\end{bmatrix}$
      $*\ ([1 \quad V(1{:}\,last-j, j)^T]\, R(j{:}\,last, j+1{:}\,n-p))$
  end
end
$\widetilde{R} =$ upper triangular part of $R(1{:}\,m, 1{:}\,n-p)$

If $\widetilde{Q}$ is required, it can be computed with the following algorithm.

Algorithm 2.18 *Given the matrix V and vector τ from Algorithm 2.17 this algorithm forms an orthogonal matrix $\widetilde{Q} \in \mathbb{R}^{m\times m}$ such that $\widetilde{A} = \widetilde{Q}\widetilde{R}$, where $\widetilde{A}$ is the matrix $A = QR$ with the kth to $(k+p-1)$st columns deleted.*

$$
\begin{aligned}
&\mathrm{lim} = \min(m-1, n-p)\\
&\text{for } j = k\colon lim\\
&last = \min(j+p, m)\\
&Q(1\colon m, j\colon last) = Q(1\colon m, j\colon last)\\
&\quad -\tau(j)\Big(Q(1\colon m, j\colon last)\begin{bmatrix} 1 \\ V(1\colon last-j, j)\end{bmatrix}\Big)\,[\,1 \quad V(1\colon last-j, j)^T\,]\\
&\text{end}\\
&\widetilde{Q} = Q
\end{aligned}
$$

In the case when $k > \min(m-1, n-p)$ then either $k = n-p+1$ or $m < n$ and the deleted columns are in R_{12} where

$$Q^T\widetilde{A} = [\,R_{11} \quad R_{12}\,],$$

with R_{12} full. There is no computation to do in either case.

See Appendix 6.5 for Fortran codes `delcols.f` and `delcolsq.f` for updating R and Q respectively.

2.5 Adding Columns

2.5.1 Adding One Column

If we wish to *add* a *variable* to our least squares problem, we have the problem of updating $A = QR$ after adding a column, $u \in \mathbb{R}^m$, in the kth position, $1 \le k \le n+1$, of $A = QR$, we can then write

$$\widetilde{A} = [\,A(1\colon m, 1\colon k-1) \quad u \quad A(1\colon m, k\colon n)\,]$$

then

$$Q^T\widetilde{A} = [\,R(1\colon m, 1\colon k-1) \quad v \quad R(1\colon m, k\colon n)\,], \tag{2.12}$$

where $v = Q^T u$. For example, with $m = 8$, $n = 6$ and $k = 4$ the right-hand side of Equation (2.12) looks like:

$$\begin{bmatrix} + & + & + & + & + & + & + \\ 0 & + & + & + & + & + & + \\ 0 & 0 & + & + & + & + & + \\ 0 & 0 & 0 & + & + & + & + \\ 0 & 0 & 0 & \ominus & \oplus & + & + \\ 0 & 0 & 0 & \ominus & 0 & \oplus & + \\ 0 & 0 & 0 & \ominus & 0 & 0 & \oplus \\ 0 & 0 & 0 & \ominus & 0 & 0 & 0 \end{bmatrix},$$

with the nonzero elements to remain represented with a $+$, the elements to be eliminated are $\ominus$ and the zero elements that can be filled in are shown with a $\oplus$.

Thus we can define $m-k$ Givens matrices, $G(i,j) \in \mathbb{R}^{m\times m}$, to eliminate $v(k+1\colon m)$. We then have

$$(G(k,k+1)^T \dots G(m-1,m)^T Q^T)\widetilde{A} = \widetilde{Q}^T \widetilde{A} = \widetilde{R},$$

where $\widetilde{R} \in \mathbb{R}^{m\times(n+1)}$ is upper trapezoidal and $\widetilde{Q} \in \mathbb{R}^{m\times m}$ is orthogonal. We then update b by computing

$$G(k,k+1)^T \dots G(m-1,m)^T Q^T b = \tilde{d}.$$

This gives the following algorithm.

Algorithm 2.19 *Given $A = QR \in \mathbb{R}^{m\times n}$, with $m \geq n$, this algorithm computes $\widetilde{Q}^T\widetilde{A} = \widetilde{R} \in \mathbb{R}^{m\times(n+1)}$ where $\widetilde{R}$ is upper trapezoidal, $\widetilde{Q}$ is orthogonal and $\widetilde{A}$ is A with a, $u \in \mathbb{R}^n$, column inserted in the kth position, $1 \leq k \leq n+1$, and $\tilde{d}$ such that $\|\widetilde{A}x - b\|_2 = \|\widetilde{R}x - \tilde{d}\|_2$. The residual, $\|\tilde{d}(n+1\colon m)\|_2$, is also computed.*

$u = Q^T u$
$\tilde{d} = Q^T b$
for $i = m\colon -1\colon k+1$
  $[c(i), s(i)] = \mathbf{givens}(u(i-1), u(i))$
  $u(i-1) = c(i)u(i-1) - s(i)\widetilde{R}(i)$
  % Update R if there is a nonzero row
  if $i \leq n+1$

$$R(i-1\colon i, i-1\colon n) = \begin{bmatrix} c(i) & s(i) \\ -s(i) & c(i) \end{bmatrix}^T R(i-1\colon i, i-1\colon n)$$

  end
  % Update R

$$\tilde{d}(i-1\colon i) = \begin{bmatrix} c(i) & s(i) \\ -s(i) & c(i) \end{bmatrix}^T \tilde{d}(i-1\colon i)$$

end

if $k = 1$

$\widetilde{R} =$ upper triangular part of $[\, u \quad R \,]$

else if $k = n+1$

$\widetilde{R} =$ upper triangular part of $[\, R \quad u \,]$

else

$\widetilde{R} =$ upper triangular part of $[\, R(1\colon m, 1\colon k-1) \quad u \quad R(1\colon m, k\colon n) \,]$

end

% Compute the residual

$resid = \|\tilde{d}(n+1\colon m)\|_2$

If $\widetilde{Q}$ is required, it can be computed with the following algorithm.

Algorithm 2.20 *Given the vectors c and s from Algorithm 2.19 this algorithm forms an orthogonal matrix* $\widetilde{Q} \in \mathbb{R}^{m\times m}$ *such that* $\widetilde{A} = \widetilde{Q}\widetilde{R}$*, where* $\widetilde{A}$ *is the matrix* $A = QR$ *with a column inserted in the* k*th position.*

for $i = m\colon -1\colon k+1$

$$Q(1\colon m, i-1\colon i) = Q(1\colon m, i-1\colon i) \begin{bmatrix} c(j) & s(j) \\ -s(j) & c(j) \end{bmatrix}$$

end

$\widetilde{Q} = Q$

2.5.2 Adding a Block of Columns

If we add p variables to our problem, that is add a block of p columns, $U \in \mathbb{R}^{m\times p}$, in the kth to $(k+p-1)$st positions of A we can write

$$\widetilde{A} = [\, A(1\colon m, 1\colon k-1) \quad U \quad A(1\colon m, k\colon n) \,]$$

then

$$Q^T\widetilde{A} = [\, R(1\colon m, 1\colon k-1) \quad V \quad R(1\colon m, k\colon n) \,],$$

where $V = Q^T U$. For example, with $m = 12$, $n = 6$, $k = 3$ and $p = 3$ the right-hand side of Equation (2.12) looks like:

$$\begin{bmatrix} + & + & + & + & + & + & + & + & + \\ 0 & + & + & + & + & + & + & + & + \\ 0 & 0 & + & + & + & + & + & + & + \\ 0 & 0 & \ominus & + & + & \oplus & + & + & + \\ 0 & 0 & \ominus & \ominus & + & \oplus & \oplus & + & + \\ 0 & 0 & \ominus & \ominus & \ominus & \oplus & \oplus & \oplus & + \\ 0 & 0 & \ominus & \ominus & \ominus & 0 & \oplus & \oplus & \oplus \\ 0 & 0 & \ominus & \ominus & \ominus & 0 & 0 & \oplus & \oplus \\ 0 & 0 & \ominus & \ominus & \ominus & 0 & 0 & 0 & 0 \\ 0 & 0 & \ominus & \ominus & \ominus & 0 & 0 & 0 & 0 \\ 0 & 0 & \ominus & \ominus & \ominus & 0 & 0 & 0 & 0 \\ 0 & 0 & \ominus & \ominus & \ominus & 0 & 0 & 0 & 0 \end{bmatrix},$$

with the nonzero elements to remain represented with a $+$, the elements to be eliminated are $\ominus$ and the zero elements that can be filled in are shown with a $\oplus$.

We would like an orthogonal matrix, W, such that

$$\begin{bmatrix} I & 0 \\ 0 & W^T \end{bmatrix} Q^T \widetilde{A} = \widetilde{R}, \quad W \in \mathbb{R}^{(m-k+1)\times(m-k+1)}.$$

If W were the product of Householder matrices, then $\widetilde{R}$ would be full. Thus we use Givens matrices and generalize Algorithm 2.19.

Algorithm 2.21 *Given $A = QR \in \mathbb{R}^{m\times n}$, with $m \geq n$, this algorithm computes $\widetilde{Q}^T \widetilde{A} = \widetilde{R} \in \mathbb{R}^{m\times(n+p)}$ where $\widetilde{R}$ is upper trapezoidal, $\widetilde{Q}$ is orthogonal and $\widetilde{A}$ is A with a block of columns, $U \in \mathbb{R}^{m\times p}$, inserted in the kth to $(k+p-1)$st position, $1 \leq k \leq n+1$, $p \geq 1$, and $\tilde{d}$ such that $\|\widetilde{A}x - b\|_2 = \|\widetilde{R}x - \tilde{d}\|_2$. The residual, $\|\tilde{d}(n+1\colon m)\|_2$, is also computed.*

$U = Q^T U$
$\tilde{d} = Q^T b$
for $j = 1\colon p$
  for $i = m\colon -1\colon k+j$
    $[C(i,j), S(i,j)] = \mathbf{givens}(U(i-1,j), U(i,j))$
    % Update U
    $U(i-1,j) = C(i,j)U(i-1,j) - S(i,j)U(i,j)$
    if $j < p$
      $U(i-1\colon i, j+1\colon p) =$

$$\begin{bmatrix} C(i,j) & S(i,j) \\ -S(i,j) & C(i,j) \end{bmatrix}^T U(i-1{:}\,i, j+1{:}\,p)$$

end

% Update R if there is a nonzero row

if $i \le n + j$

$R(i-1{:}\,i, i-j{:}\,n) =$

$$\begin{bmatrix} C(i,j) & S(i,j) \\ -S(i,j) & C(i,j) \end{bmatrix}^T R(i-1{:}\,i, i-j{:}\,n)$$

end

% Update $\tilde{d}$

$$\tilde{d}(i-1{:}\,i) = \begin{bmatrix} C(i,j) & S(i,j) \\ -S(i,j) & C(i,j) \end{bmatrix}^T \tilde{d}(i-1{:}\,i)$$

end

end

if $k = 1$

$\widetilde{R} =$ upper triangular part of $[\,U \quad R\,]$

else if $k = n + 1$

$\widetilde{R} =$ upper triangular part of $[\,R \quad U\,]$

else

$\widetilde{R} =$ upper triangular part of $[\,R(1{:}\,m, 1{:}\,k-1) \quad U \quad R(1{:}\,m, k{:}\,n)\,]$

end

% Compute the residual

$resid = \|\tilde{d}(n+1{:}\,m)\|_2$

Computing $\widetilde{R}$ requires $6(mp(n+p-m/2)-p^2(n/2-k/2-p/3)+kp(k/2-n))$ flops, versus $2(n+p)^2(m-(n+p)/3)$ for the Householder QR factorization of $\widetilde{A}$. If $\widetilde{Q}$ is required, it can be computed with the following algorithm.

Algorithm 2.22 *Given matrices C and S from Algorithm 2.21 this algorithm forms an orthogonal matrix $\widetilde{Q} \in \mathbb{R}^{m\times m}$ such that $\widetilde{A} = \widetilde{Q}\widetilde{R}$, where $\widetilde{A}$ is the matrix $A = QR$ with a block of columns inserted in the kth to $(k+p-1)$st positions.*

for $j = 1{:}\,p$

for $i = m{:}\,-1{:}\,k+j$

$$Q(1{:}\,m, i-1{:}\,i) = Q(1{:}\,m, i-1{:}\,i) \begin{bmatrix} C(i,j) & S(i,j) \\ -S(i,j) & C(i,j) \end{bmatrix}$$

end

end

$\widetilde{Q} = Q$

We can improve on this algorithm by including a Level 3 BLAS part by using a blocked QR factorization of part of $\widetilde{A}$ before we finish the elimination process with Givens matrices. That is, for our example:

$$\begin{bmatrix} + & + & + & + & + & + & + & + & + \\ 0 & + & + & + & + & + & + & + & + \\ 0 & 0 & + & + & + & + & + & + & + \\ 0 & 0 & \ominus & + & + & \oplus & + & + & + \\ 0 & 0 & \ominus & \ominus & + & \oplus & \oplus & + & + \\ 0 & 0 & \ominus & \ominus & \ominus & \oplus & \oplus & \oplus & + \\ 0 & 0 & \ominus & \ominus & \ominus & 0 & \oplus & \oplus & \oplus \\ 0 & 0 & \odot & \ominus & \ominus & 0 & 0 & \oplus & \oplus \\ 0 & 0 & \odot & \odot & \ominus & 0 & 0 & 0 & \oplus \\ 0 & 0 & \odot & \odot & \odot & 0 & 0 & 0 & 0 \\ 0 & 0 & \odot & \odot & \odot & 0 & 0 & 0 & 0 \\ 0 & 0 & \odot & \odot & \odot & 0 & 0 & 0 & 0 \end{bmatrix},$$

we eliminate the elements shown with a $\odot$ with a QR factorization of the bottom 6 by 3 block of V and the remainder of the elements can be eliminated with Givens matrices and are shown with a $\ominus$. The zero elements that can be filled in are shown with a $\oplus$ and the nonzero elements to remain represented with a $+$ as before.

For the case of $k \neq 1, n+1$ and $m > n+1$, we have

$$Q^T \widetilde{A} = \begin{bmatrix} R_{11} & V_{12} & R_{12} \\ 0 & V_{22} & R_{23} \\ 0 & V_{32} & 0 \end{bmatrix}$$

where $R_{11} \in \mathbb{R}^{(k-1)\times(k-1)}$ and $R_{23} \in \mathbb{R}^{(n-k+1)\times(n-k+1)}$ are upper triangular, then if V_{32} has the QR factorization $V_{32} = Q_V R_V \in \mathbb{R}^{(m-n)\times p}$ we have

$$\begin{bmatrix} I_n & 0 \\ 0 & Q_V^T \end{bmatrix} Q^T \widetilde{A} = \begin{bmatrix} R_{11} & V_{12} & R_{12} \\ 0 & V_{22} & R_{23} \\ 0 & R_V & 0 \end{bmatrix}.$$

We then eliminate the upper triangular part of R_V and the lower triangular part of V_{22} with Givens matrices which makes R_{23} full and the bottom right block upper trapezoidal. So we have finally

$$\begin{aligned} G(k+2p-2, k+2p-1)^T \quad & \ldots \quad G(k+p, k+p+1)^T G(k, k+1)^T \\ & \ldots \quad G(k+p-1, k+p)^T \begin{bmatrix} I_n & 0 \\ 0 & Q_V^T \end{bmatrix} Q^T \widetilde{A} = \widetilde{R} \end{aligned}$$

This gives the following algorithm.

Algorithm 2.23 *Given $A = QR \in \mathbb{R}^{m\times n}$, with $m \geq n$, this algorithm computes $\widetilde{Q}^T\widetilde{A} = \widetilde{R} \in \mathbb{R}^{m\times(n+p)}$ where $\widetilde{R}$ is upper trapezoidal, $\widetilde{Q}$ is orthogonal and $\widetilde{A}$ is A with a block of columns, $U \in \mathbb{R}^{m\times p}$, inserted in the kth to $(k+p-1)$st position, $1 \leq k \leq n+1$, $p \geq 1$, and $\tilde{d}$ such that $\|\widetilde{A}x - b\|_2 = \|\widetilde{R}x - \tilde{d}\|_2$. The residual, $\|\tilde{d}(n+1\colon m)\|_2$, is also computed. The algorithm incorporates a Level 3 QR factorization.*

$U = Q^T U$
$\tilde{d} = Q^T b$
if $m > n+1$
  % Factorize rows $n+1$ to m of U if there are more than 1,
  % with a Level 3 QR algorithm
  $U(n+1\colon m, 1\colon p) = Q_U R_U$
  % Update $\tilde{d}$
  $\tilde{d}(n+1\colon m) = Q_U^T \tilde{d}(n+1\colon m)$
end
if $k \leq n$
  % Zero out the rest with Givens
  for $j = 1\colon p$
    % First iteration updates one column
    $upfirst = n$
    for $i = n+j\colon -1\colon j+1$
      $[C(i,j), S(i,j)] = \textbf{givens}(U(i-1,j), U(i,j))$
      % Update U
      $U(i-1,j) = C(i,j)U(i-1,j) - S(i,j)U(i,j)$
      if $j < p$
        $U(i-1\colon i, j+1\colon p) =$
$$\begin{bmatrix} C(i,j) & S(i,j) \\ -S(i,j) & C(i,j) \end{bmatrix}^T U(i-1\colon i, j+1\colon p)$$
      end
      % Update R
        $R(i-1\colon i, upfirst\colon n) =$
$$\begin{bmatrix} C(i,j) & S(i,j) \\ -S(i,j) & C(i,j) \end{bmatrix}^T R(i-1\colon i, upfirst\colon n)$$
      % Update one more column next i step
      $upfirst = upfirst - 1$
      % Update $\tilde{d}$
$$\tilde{d}(i-1\colon i) = \begin{bmatrix} C(i,j) & S(i,j) \\ -S(i,j) & C(i,j) \end{bmatrix}^T \tilde{d}(i-1\colon i)$$
    end

    end
end
if $k = 1$
    $\widetilde{R} =$ upper triangular part of $[\,U \quad R\,]$
else if $k = n + 1$
    $\widetilde{R} =$ upper triangular part of $[\,R \quad U\,]$
else
    $\widetilde{R} =$ upper triangular part of $[\,R(1{:}\,m, 1{:}\,k-1) \quad U \quad R(1{:}\,m, k{:}\,n)\,]$
end
% Compute the residual
$resid = \|\tilde{d}(n+1{:}\,m)\|_2$

If $\widetilde{Q}$ is required, it can be computed with the following algorithm.

Algorithm 2.24 *Given matrices Q_U, C and S and the vector τ from Algorithm 2.23 this algorithm forms an orthogonal matrix $\widetilde{Q} \in \mathbb{R}^{m\times m}$ such that $\widetilde{A} = \widetilde{Q}\widetilde{R}$, where $\widetilde{A}$ is the matrix $A = QR$ with a block of columns inserted in the kth to $(k+p-1)$st positions.*

if $m > n + 1$
    $Q(1{:}\,m, 1{:}\,m-n) = Q(1{:}\,m, 1{:}\,m-n)Q_U$
end
if $k \le n$
    for $j = 1{:}\,p$
        for $i = n + j{:} -1{:}\,j + 1$
            $Q(1{:}\,m, i-1{:}\,i) = Q(1{:}\,m, i-1{:}\,i)\begin{bmatrix} C(i,j) & S(i,j) \\ -S(i,j) & C(i,j) \end{bmatrix}$
        end
    end
end
$\widetilde{Q} = Q$

2.5.3 Updating the QR Factorization for any m and n

In the case where $m > n$, to update only the QR factorization, then we need to consider the limits of the for loops and *upstart* to convert Algorithm 2.23.

- We introduce $jstop$ which is the last index in the outer for loop. There may be a situation where there are not elements to eliminate over the full width of U. For example $Q^T\widetilde{A}$, for $m=5$, $n=6$, $k=3$ and $p=3$, looks like:

$$\begin{bmatrix} + & + & + & + & + & + & + & + & + \\ 0 & + & + & + & + & + & + & + & + \\ 0 & 0 & + & + & + & + & + & + & + \\ 0 & 0 & \ominus & + & + & \oplus & + & + & + \\ 0 & 0 & \ominus & \ominus & + & \oplus & \oplus & + & + \end{bmatrix},$$

and there are no elements to eliminate in the last column of V.

- $istart$ is introduced as the first element in the jth column to be eliminated cannot exceed m.

- The first column to be updated for jth step may no longer be n, so $upfirst$ is set accordingly.

Note if $m \leq n+1$ and $k > n$ there is nothing to do and neither outer if block is entered.

Algorithm 2.25 *Given $A = QR \in \mathbb{R}^{m\times n}$, this algorithm computes $\widetilde{Q}^T\widetilde{A} = \widetilde{R} \in \mathbb{R}^{m\times(n+p)}$ where $\widetilde{R}$ is upper trapezoidal, $\widetilde{Q}$ is orthogonal and $\widetilde{A}$ is A with a block of columns, $U \in \mathbb{R}^{m\times p}$, inserted in the kth to $(k+p-1)$st position. The algorithm incorporates a Level 3 QR factorization.*

$U = Q^TU$
if $m > n+1$
    % Factorize rows $n+1$ to m of U if there are more than 1,
    % with a Level 3 QR algorithm
    $U(n+1{:}\, m, 1{:}\, p) = Q_U R_U$
end
if $k \leq n$
    % Zero out the rest with Givens, stop at the last column of
    % U or the last row if that is reached first
    $jstop = \min(p, m-k-2)$
    for $j = 1{:}\, jstop$
        % Start at first row to be eliminated in current column
        $istart = \min(n+j, m)$
        % Index of first nonzero column in update of R
        $upfirst = \max(istart - j - 1, 1)$
        for $i = istart{:}\, {-1}{:}\, j+1$
            $[C(i,j), S(i,j)] = \textbf{givens}(U(i-1,j), U(i,j))$

% Update U

$U(i-1,j) = C(i,j)U(i-1,j) - S(i,j)U(i,j)$

if $j < p$

% Update U

$U(i-1{:}\,i, j+1{:}\,p) =$

$$\begin{bmatrix} C(i,j) & S(i,j) \\ -S(i,j) & C(i,j) \end{bmatrix}^T U(i-1{:}\,i, j+1{:}\,p)$$

end

% Update R

$R(i-1{:}\,i, upfirst{:}\,n) =$

$$\begin{bmatrix} C(i,j) & S(i,j) \\ -S(i,j) & C(i,j) \end{bmatrix}^T R(i-1{:}\,i, upfirst{:}\,n)$$

% Update one more column next i step

$upfirst = upfirst - 1$

end

end

end

if $k = 1$

$\widetilde{R} =$ upper triangular part of $[\,U \quad R\,]$

else if $k = n+1$

$\widetilde{R} =$ upper triangular part of $[\,R \quad U\,]$

else

$\widetilde{R} =$ upper triangular part of $[\,R(1{:}\,m, 1{:}\,k-1) \quad U \quad R(1{:}\,m, k{:}\,n)\,]$

end

If $\widetilde{Q}$ is required, it can be computed with the following algorithm.

Algorithm 2.26 *Given matrices Q_U, C and S and the vector τ from Algorithm 2.25 this algorithm forms an orthogonal matrix $\widetilde{Q} \in \mathbb{R}^{m\times m}$ such that $\widetilde{A} = \widetilde{Q}\widetilde{R}$, where $\widetilde{A}$ is the matrix $A = QR$ with a block of columns inserted in the kth to $(k+p-1)$st positions.*

if $m > n+1$

$Q(1{:}\,m, n+1{:}\,m) = Q(1{:}\,m, n+1{:}\,m)Q_U$

end

if $k \le n$

$jstop = \min(p, m-k-2)$

for $j = 1{:}\,jstop$

$istart = \min(n+j, m)$

for $i = istart{:} -1{:}\, j+1$

$$Q(1{:}\,m, i-1{:}\,i) = Q(1{:}\,m, i-1{:}\,i)\begin{bmatrix} C(i,j) & S(i,j) \\ -S(i,j) & C(i,j) \end{bmatrix}$$

end

end

end

$\widetilde{Q} = Q$

See Appendix 6.5 for Fortran codes `addcols.f` and `addcolsq.f` for updating R and Q respectively.

3 Error Analysis

It is well known that orthogonal transformations are stable. We have the following columnwise results [10], where

$$\tilde{\gamma}_k = \frac{cku}{1 - cku},$$

and u is the unit roundoff and c is a small integer constant.

Lemma 3.1 (Sequence of Givens Matrices) *If*

$$B = G_r \ldots G_1 A = Q^T A \in \mathbb{R}^{n \times n}$$

where G_i is a Givens matrix, then the computed matrix $\widehat{B}$ satisfies

$$Q^T A - \widehat{B} = \Delta B, \qquad \|\Delta b_j\|_2 \leq \tilde{\gamma}_r \|a_j\|_2, \quad j = 1{:}\,n. \quad \Box \tag{3.1}$$

Lemma 3.2 (Sequence of Householder Matrices) *If*

$$B = H_r \ldots H_1 A = Q^T A \in \mathbb{R}^{n \times n}$$

where H_i is a Householder matrix, then the computed matrix $\widehat{B}$ satisfies

$$Q^T A - \widehat{B} = \Delta B, \qquad \|\Delta b_j\|_2 \leq \tilde{\gamma}_{nr} \|a_j\|_2, \quad j = 1{:}\,n. \quad \Box \tag{3.2}$$

This result implies that Householder transformations are less accurate by a factor of n, but this is not observed in practice. We then have

Theorem 3.1 (Householder QR Factorization) *If*

$$R = Q^T A$$

where Q is a product of Householder matrices, then the computed factor $\widehat{R}$ satisfies

$$Q^T A - \widehat{R} = \Delta R, \qquad \|\Delta r_j\|_2 \;\leq \tilde{\gamma}_{mn} \|a_j\|_2, \quad j = 1\colon n. \quad \square$$

We now give results for computing the factor $\widetilde{R}$ by our algorithms.

3.1 Deleting Rows

We have from Section 2.1.2

$$G(1,2)^T \ldots G(m-1,m)^T \widehat{R} = \begin{bmatrix} v^T \\ \widetilde{R} \end{bmatrix},$$

and from (3.1) we have for the computed quantities $\widehat{\widetilde{R}}$ and $\hat{v}$

$$\widehat{\widetilde{R}} = \widetilde{R} + \Delta R, \qquad \|\Delta r_j\|_2 \leq \tilde{\gamma}_{mp} \left\| \begin{bmatrix} v^T(j) \\ \hat{r}(1\colon n, j) \end{bmatrix} \right\|_2, \quad j = 1\colon n.$$

Recall that the $G(i,j)$ are chosen to introduce zeros in Q.

3.2 Adding Rows

We have from Section 2.3.2

$$H_n \ldots H_1 \begin{bmatrix} \widehat{R} \\ U \end{bmatrix} = \widetilde{R}, \quad U \in \mathbb{R}^{p \times n},$$

and from (3.2) we have

$$\widehat{\widetilde{R}} = \widetilde{R} + \Delta R, \qquad \|\Delta r_j\|_2 \leq \tilde{\gamma}_{n(p+1)} \left\| \begin{bmatrix} \hat{r}_{jj} \\ U(:, j) \end{bmatrix} \right\|_2, \quad j = 1\colon n.$$

3.3 Deleting Columns

We have from Section 2.4.2

$$H_{n-p} \ldots H_k \, [\, \widehat{R}(:, 1\colon k-1) \quad \widehat{R}(:, k+p\colon n) \,] = \widetilde{R},$$

and from (3.2) we have

$$\widehat{\widetilde{R}} = \widetilde{R} + \begin{bmatrix} 0 \\ \Delta R \end{bmatrix}, \quad \Delta R \in \mathbb{R}^{(m-k+1)\times n}$$

$$\begin{aligned} \|\Delta r_j\|_2 &= 0, & j &= 1\colon k-1, \\ &\leq \tilde{\gamma}_{(n-k-p+1)(n-k+1)} \|\hat{r}(k\colon n, j)\|_2, & j &= k\colon n-p. \end{aligned}$$

3.4 Adding Columns

We have from Section 2.5.2

$$\begin{aligned} &G(k+2p-2, k+2p-1)\ldots \\ &G(k+p-1, k+p) \begin{bmatrix} I & 0 \\ 0 & Q_V^T \end{bmatrix} [\, \widehat{R}(:, 1\colon k-1) \quad \widehat{V} \quad \widehat{R}(:, k\colon n) \,] = \widetilde{R}, \end{aligned}$$

where $\widehat{V} = Q^T U \in \mathbb{R}^{m\times p}$ and from (3.1) and (3.2) we have

$$\widehat{\widetilde{R}} = \widetilde{R} + \begin{bmatrix} 0 \\ 0 \\ \Delta H \end{bmatrix} + \begin{bmatrix} 0 \\ \\ \Delta G \end{bmatrix}, \quad \Delta H \in \mathbb{R}^{(m-n)\times n}, \quad \Delta G \in \mathbb{R}^{(m-k+1)\times n}$$

$$\begin{aligned} \|\Delta H_j\|_2 &= 0, & k-1 &\geq j \geq k+p, \\ &\leq \tilde{\gamma}_{(n-k)p} \|\widehat{V}(n+1\colon m, j)\|_2, & j &= k\colon k+p-1, \end{aligned}$$

$$\begin{aligned} \|\Delta G_j\|_2 &= 0, & j &= 1\colon k-1, \\ &\leq \tilde{\gamma}_{(n-k)n} \left\| (Q^T\widehat{V})(k\colon m, j) + \begin{bmatrix} 0 \\ \Delta \hat{r}_j \end{bmatrix} \right\|_2, & j &= k\colon n+p. \end{aligned}$$

Given these results we expect the normwise backward error

$$\frac{\|\widetilde{A} - \widetilde{Q}\widetilde{R}\|_2}{\|\widetilde{A}\|_2},$$

when $\widetilde{Q}$ and $\widetilde{R}$ are computed with our algorithms to be close to that with $\widetilde{Q}$ and $\widetilde{R}$ computed directly from $\widetilde{A}$. We consider some examples in the next section.

4 Numerical Experiments

4.1 Speed Tests

In this section we test the speed of our double precision Fortran 77 codes, see Appendix 6.5, against LAPACK's `DGEQRF`, a Level 3 BLAS routine for computing the QR factorization of a matrix. The input matrix, in this case $\widetilde{A}$, is overwritten with $\widetilde{R}$, and $\widetilde{Q}$ is returned in factored form in the same way as our codes do.

The tests were performed on a 1400MHz AMD Athlon running Red Hat Linux version 6.2 with kernel 2.2.22. The unit roundoff $u \approx$ 1.1e-16.

We tested our code with

$$m = \{1000,\ 2000,\ 3000,\ 4000,\ 5000\}$$

and $n = 0.3m$, and the number of columns added or deleted was $p = 100$. We generated our test matrices by populating an array with random double precision numbers generated with the LAPACK auxiliary routine `DLARAN`. $A = QR$ was computed with `DGEQRF`, and $\widetilde{A}$ was formed appropriately.

We timed our codes acting on $Q^T\widetilde{A}$, the starting point for computing $\widetilde{R}$, and in the case of adding columns we included the computation of Q^TU in our timings, which we formed with the BLAS routine `DGEMM`. We also timed `DGEQRF` acting on only the part of $Q^T\widetilde{A}$ that needs to be updated, the nonzero part from row and column k onwards. Here we can construct $\widetilde{R}$ with this computation and the original R. Finally, we compute `DGEQRF` acting on $\widetilde{A}$. We aim to show our codes are faster than these alternatives. In all cases an average of three timings are given.

To test our code `DELCOLS` we first chose $k = 1$, the position of the first column deleted, where the maximum amount of work is required to update the factorization. We have

$$\widetilde{A} = A(1\colon m, p+1\colon n), \quad \text{and} \quad Q^T\widetilde{A} = R(1\colon m, p+1\colon n)$$

and timed:

- `DGEQRF` on $\widetilde{A}$.
- `DGEQRF` on $(Q^T\widetilde{A})(k\colon n, k\colon n-p)$ which computes the nonzero entries of $\widetilde{R}(k\colon m, p+1\colon n)$.
- `DELCOLS` on $Q^T\widetilde{A}$.

The results are given in Figure 1. Our code is clearly much faster than recomputing the factorization from scratch with `DGEQRF`, and for $n = 5000$ there is a speedup of 20. Our code is also faster than using `DGEQRF` on $(Q^T\widetilde{A})(k\colon n, k\colon n-p)$, where there is a maximum speedup of over 3.

We then tested for $k = n/2$ where much less work is required to perform the updating, we have

$$\begin{aligned} \widetilde{A} &= [\,A(1\colon m, 1\colon k-1) \quad A(1\colon m, k+p\colon n)\,], \quad \text{and} \\ Q^T\widetilde{A} &= [\,R(1\colon m, 1\colon k-1) \quad R(1\colon m, k+p\colon n)\,] \end{aligned}$$

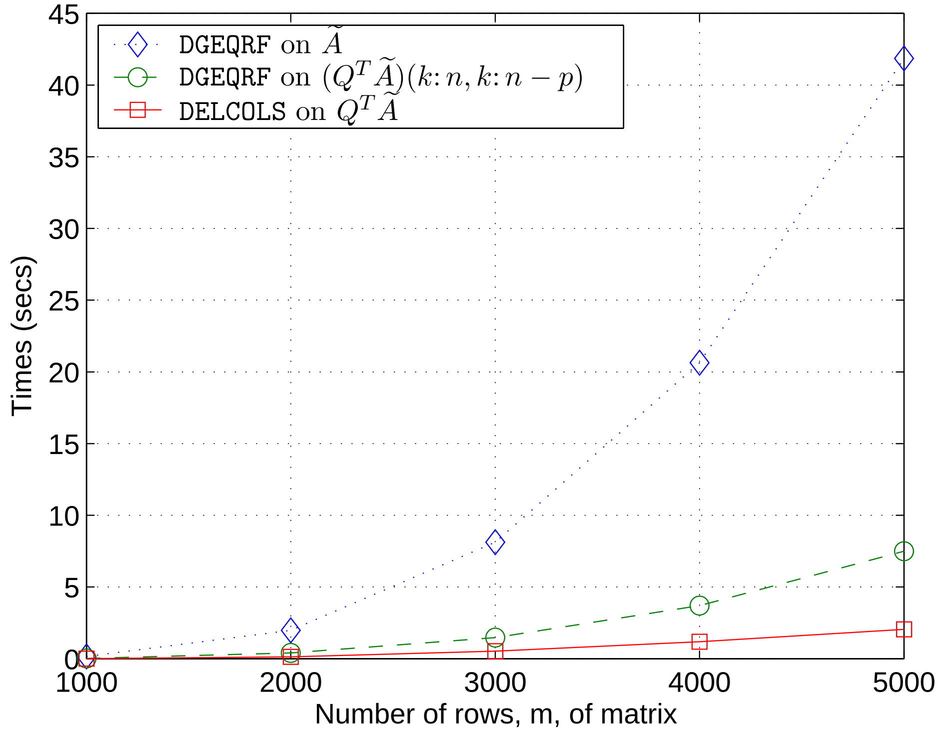

Figure 1: Comparison of speed for `DELCOLS` with $k = 1$ for different m.

and timed:

- `DGEQRF` on $(Q^T\widetilde{A})(k\colon n, k\colon n-p)$ which computes the nonzero entries of $\widetilde{R}(k\colon m, k\colon n-p)$.
- `DELCOLS` on $Q^T\widetilde{A}$.

The results are given in Figure 2. The timings for `DGEQFR` on $\widetilde{A}$ would, of course, be the same as for $k = 1$, giving a maximum speedup of over 100 in this case. We achieve a speedup of approximately 3 over using `DGEQRF` on $(Q^T\widetilde{A})(k\colon n, k\colon n-p)$.

We then considered the effect of varying p with `DELCOLS` for fixed $m = 3000$, $n = 1000$ and $k = 1$. As we delete more columns from A there are less columns to update, but more work is required for each one. We chose

$$p = \{100,\ 200,\ 300,\ 400,\ 500,\ 600\ 700,\ 800\}$$

and timed:

- `DGEQRF` on $\widetilde{A}$

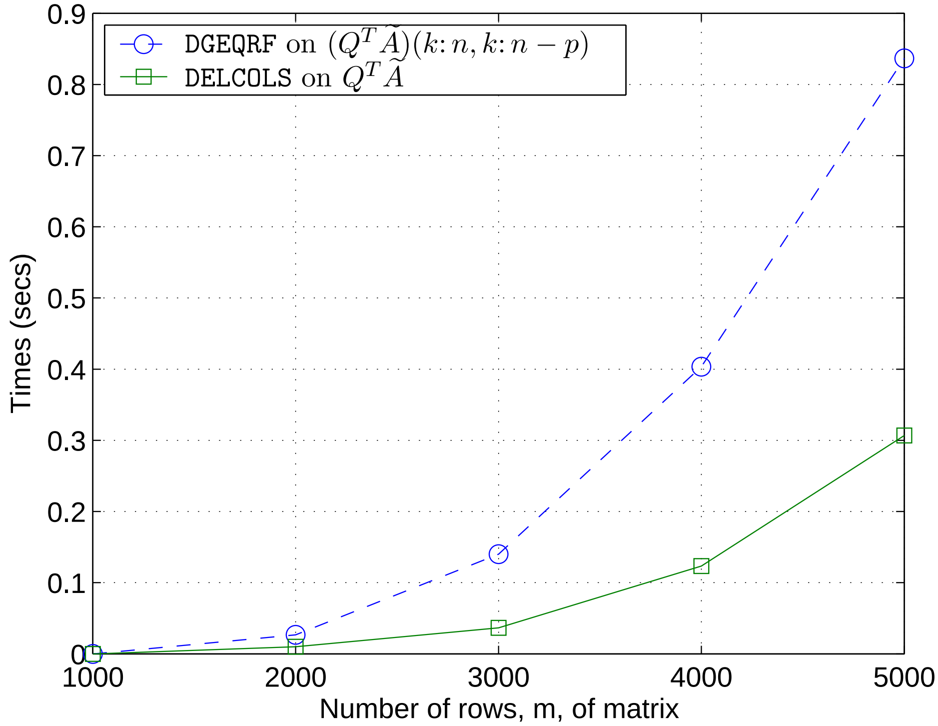

Figure 2: Comparison of speed for DELCOLS with $k = n/2$ for different m.

- DGEQRF on $(Q^T\widetilde{A})(k\colon n, k\colon n-p)$ which computes the nonzero entries of $\widetilde{R}(k\colon m, k\colon n-p)$.
- DELCOLS on $Q^T\widetilde{A}$.

The results are given in Figure 3. The timings for DELCOLS are relatively level and peak at $p = 300$, whereas the timings for the other codes obviously decrease with p. The speedup of our code decreases with p, and from $p = 300$ there is little difference between our code and DGEQRF on $(Q^T\widetilde{A})(k\colon n, k\colon n-p)$.

To test ADDCOLS we generated random matrices $A \in \mathbb{R}^{m\times n}$ and $U \in \mathbb{R}^{m\times p}$, and again use

$$m = \{1000,\ 2000,\ 3000,\ 4000,\ 5000\}$$

$n = 0.3m$, and $p = 100$. We first set $k = 1$ where maximum updating is required. We have

$$\widetilde{A} = [\,U \quad A\,], \quad \text{and} \quad Q^T\widetilde{A} = [\,Q^TU \quad R\,]$$

and timed:

- DGEQRF on $\widetilde{A}$.

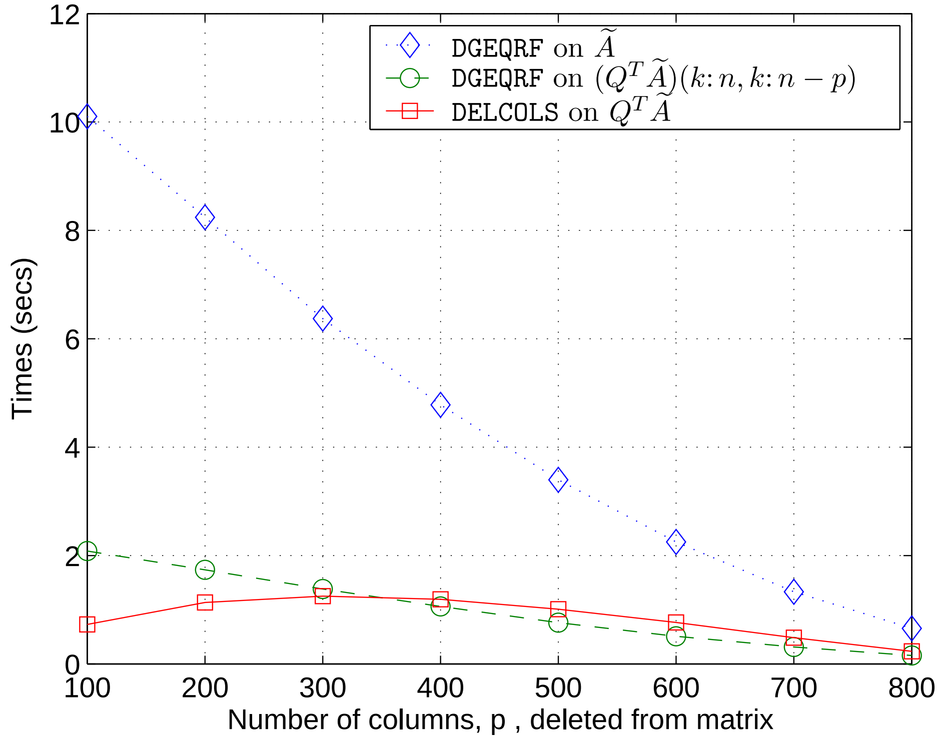

Figure 3: Comparison of speed for DELCOLS for different p.

- ADDCOLS on $Q^T\widetilde{A}$, including the computation of Q^TU with DGEMM.

The results are given in Figure 4. Here our code achieves a speedup of over 3 for $m = 5000$ over the complete factorization of $\widetilde{A}$.

We then tested for $k = n/2$, where less work is required to do the updating. We have

$$\begin{aligned} \widetilde{A} &= [\, A(1\colon m, 1\colon k-1) \quad U \quad A(1\colon m, k\colon n)\,], \quad \text{and} \\ Q^T\widetilde{A} &= [\, R(1\colon m, 1\colon k-1) \quad Q^TU \quad R(1\colon m, k\colon n)\,] \end{aligned}$$

and timed:

- DGEQRF on $\widetilde{A}$, as above.
- DGEQRF on $(Q^T\widetilde{A})(k\colon m, k\colon n{+}p)$ which computes $\widetilde{R}(k\colon m, k\colon n{+}p)$, including the computation of Q^TU for which we again use DGEMM.
- ADDCOLS on $Q^T\widetilde{A}$, including the computation of Q^TU.

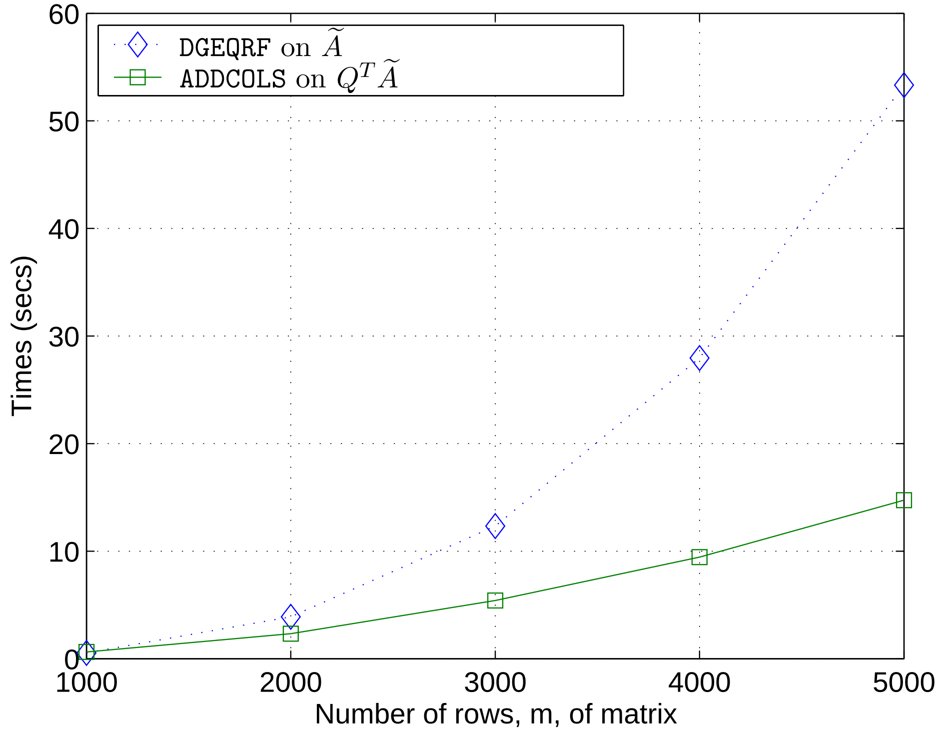

Figure 4: Comparison of speed for `ADDCOLS` with $k = 1$ for different m.

The results are given in Figure 5. Here we have a maximum speedup of over 4 with our code against `DGEQRF` on $\widetilde{A}$. We achieve a maximum speedup of approximately 2 against `DGEQRF` on $(Q^T\widetilde{A})(k\colon m, k\colon n+p)$.

We do not vary p as this increases the work for both our code and `DGEQRF` on $(Q^T\widetilde{A})(k\colon m, k\colon n+p)$ roughly equally.

4.2 Backward Error Tests

The tests here were performed on a 2545MHz AMD Pentium running a hybrid version of Red Hat Linux 8 and 9 with kernel 2.4.20.

Here we test our code for updating Q and R; `DELCOLS` and `DELCOLSQ` for deleting columns and `ADDCOLS` and `ADDCOLSQ` for adding columns. We did this in the following way:

- We form a random matrix

$$A^{(0)} = [\, A_1 \quad U \quad A_2 \,], \quad \|A_1\|_F, \|A_2\|_F, \|U\|_F \text{ of order } 100,$$

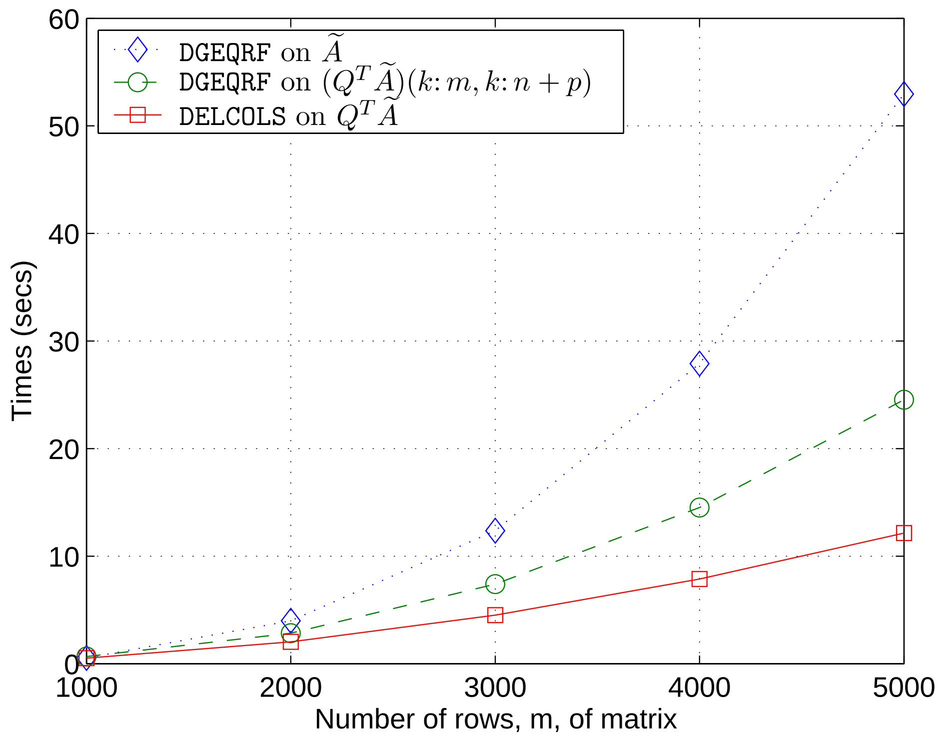

Figure 5: Comparison of speed for `ADDCOLS` with $k = n/2$ for different m.

where $A_1 \in \mathbb{R}^{m\times(k-1)}$, $A_2 \in \mathbb{R}^{m\times(n-k-p+1)}$, $U \in \mathbb{R}^{m\times p}$.

- We then form the QR factorization

$$A^{(0)} = Q^{(0)}R^{(0)} = Q^{(0)}\left[\, R_1 \quad R_U \quad R_2 \,\right],$$

where $R_1 \in \mathbb{R}^{m\times(k-1)}$, $R_2 \in \mathbb{R}^{m\times(n-k-p+1)}$, $R_U \in \mathbb{R}^{m\times p}$, using the LAPACK routines `DGEQRF` and `DORGQR`.

- Next, for

$$\widetilde{A} = \left[\, A_1 \quad A_2 \,\right],$$

we form

$$Q^{(0)^T}\widetilde{A} = \left[\, R_1 \quad R_2 \,\right],$$

and call `DELCOLS` and `DELCOLSQ` to update the QR factorization of $\widetilde{A}$, forming

$$\widetilde{A} = \widetilde{Q}\widetilde{R} = \left[\, \widetilde{R}_1 \quad \widetilde{R}_2 \,\right],$$

where $\widetilde{R}_1 \in \mathbb{R}^{m\times(k-1)}$, $\widetilde{R}_2 \in \mathbb{R}^{m\times(n-k-p+1)}$.

- We now compute the QR factorization of $A^{(0)}$ by updating $\widetilde{Q}$ and $\widetilde{R}$. We call `ADDCOLS` on

$$[\, \widetilde{R}_1 \quad \widetilde{Q}^T U \quad \widetilde{R}_2 \,]$$

to form $R^{(1)}$ and then call `ADDCOLSQ` to form $Q^{(1)}$, so we have, in exact arithmetic

$$A^{(0)} = Q^{(1)} R^{(1)}.$$

We then repeat this *rep* times and measure the normwise backward error

$$\frac{\|A^{(0)} - Q^{(rep)} R^{(rep)}\|_2}{\|A\|_2}.$$

We use every combination of the following set of parameters:

$$\begin{aligned} m &= 500 \\ n &= \{400,\ 500,\ 600\} \\ p &= \{50,\ 100,\ 150\} \\ k &= \{1,\ 51, \ldots,\ n-p+1\}. \end{aligned}$$

We then repeated the entire process, but with

$$\|U\|_F \text{ of order 1e+9.}$$

The results are given in Table 1 and Table 2. The error increases with the number of repeats which is expected. However, the value is not effected significantly by the value of $\|U\|_F$.

The smallest value of the error in every case was approximately of order $10u$. The worse case was still only of order $2 * rep * u$.

Table 1: Normwise backward error for $\|U\|_F$ order 100.

rep	5	50	500
Smallest error over all tests	1.146e-15	1.212e-15	1.223e-15
Largest error over all tests	5.031e-15	2.399e-14	1.252e-13

Table 2: Normwise backward error for $\|U\|_F$ order 1e+9.

rep	5	50	500
Smallest error over all tests	8.298e-16	9.309e-16	9.576e-16
Largest error over all tests	4.381e-15	2.055e-14	1.014e-13

5 Conclusions

The speed tests show that our updating algorithms are faster than computing the QR factorization from scratch or using the factorization to update columns k onward, the only columns needing updating.

Furthermore, the normwise backward error tests show that the errors are within the bound for computing the Householder QR factorization of $\widetilde{A}$. Thus, within the parameters of our experiments, the increase of speed is not at the detriment of accuracy.

We propose the double precision Fortran 77 codes `delcols.f`, `delcolsq.f`, `addcols.f` and `addcolsq.f`, and their single precision and complex equivalents, be included in LAPACK.

6 Software Available

Here we list some software that is available to update the QR factorization and least squares problem. An 'x' in a routine indicates more than one routine for different precisions or for real or complex data.

6.1 LINPACK

LINPACK [7] has three routines that update the least squares problem and the QR factorization.

- `xCHUD` updates the least squares problem when a row has been added in the $(m+1)$st position.
- `xCHDD` updates the least squares problem when a row has been deleted from the mth position, an implementation of Saunder's algorithm.

- `xCHEX` update the least squares problem when the rows of A have been permuted.

  In all cases the transformation matrices are represented by a vectors of sines and cosines, and $\widetilde{Q}$ is not constructed.

6.2 MATLAB

MATLAB [11] supply three routines for updating the QR factorization only.

- `qrdelete` updates when one row or column is deleted from any position.
- `qrinsert` updates when one row or column is added to any position.
- `qrupdate` returns the factorization of A after a rank one change, that is

$$\widetilde{A} = A + uv^T, \quad u \in \mathbb{R}^m, \; v \in \mathbb{R}^n.$$

  In all cases both $\widetilde{Q}$ and $\widetilde{R}$ are returned.

6.3 The NAG Library

The Mark 20 NAG Library [12] contains routines for updating two cases.

- `F06xPF` performs the factorization

$$\alpha uv^T + R_1 = \overline{Q}\widetilde{R}_1,$$

  where

$$R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}, \quad \widetilde{R} = \begin{bmatrix} \widetilde{R}_1 \\ 0 \end{bmatrix}, \quad R_1, \widetilde{R}_1 \in \mathbb{R}^{n \times n},$$

  and

$$\widetilde{Q} = Q\overline{Q}.$$

  $\overline{Q}$ is represented by vectors of sines and cosines.

- `F06xQF` performs the downdating problem

$$\begin{bmatrix} R_1 \\ \alpha v^T \end{bmatrix} = \overline{Q} \begin{bmatrix} \widetilde{R}_1 \\ 0 \end{bmatrix},$$

  where R, $\widetilde{R}$ and $\widetilde{Q}$ are as above.

6.4 Reichel and Gragg's Algorithms

Reichel and Gragg [16] provide several Fortran 77 implementations of the algorithms discussed in [6] for updating the QR factorization, returning both $\widetilde{Q}$ and $\widetilde{R}$. In all cases only $m \geq n$ is handled. The routines use BLAS like routines for matrix and vector operations written for optimal performance on the test machine used in [16]. No error results are given for the Fortran routines, although some results are given for the Algol implementations in [6].

- `DDELR` updates after one row is deleted; this algorithm varies from ours and uses a Gram-Schmidt re-orthogonalization process.
- `DINSR` updates when one row is added, and is similar to our algorithm.
- `DDELC` updates when one column is deleted, and is similar to our algorithm.
- `DINSC` updates after one column is added; this algorithm varies from ours and again uses a Gram-Schmidt re-orthogonalization process.
- `DRNK1` updates after a rank 1 modification to A.
- `DRRPM` updates when $\widetilde{A}$ is A with some if its columns permuted.

6.5 What's new in our algorithms

Our contribution is:

- We deal with adding/deleting block of p row/columns, and in two of the four cases we exploit the Level 3 BLAS. Also, the Level 2 code for deleting a block of rows is more efficient than calling the code for deleting one row p times.
- In the case of updating the QR factorization we place no restrictions on m and n.
- All our codes call existing BLAS and LAPACK routines.

In these Fortran files the dimension n refers to the number of columns in $\widetilde{A}$, and not A.

A delcols.f

```
      SUBROUTINE DELCOLS( M, N, A, LDA, K, P, TAU, WORK, INFO )
*
*     Craig Lucas, University of Manchester
*     March, 2004
*
*     .. Scalar Arguments ..
      INTEGER            INFO, K, LDA, M, N, P
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION   A( LDA, * ), TAU( * ), WORK( * )
*     ..
*
*  Purpose
*  =======
*
*  Given a real m by (n+p) matrix, B, and the QR factorization
*  B = Q_B * R_B, DELCOLS computes the QR factorization
*  C = Q * R where C is the matrix B with p columns deleted
*  from the kth column onwards.
*
*  The input to this routine is Q_B' * C
*
*  Arguments
*  =========
*
*  M       (input) INTEGER
*          The number of rows of the matrix C.  M >= 0.
*
*  N       (input) INTEGER
*          The number of columns of the matrix C.  N >= 0.
*
*  A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
*          On entry, the matrix Q_B' * C. The elements in columns
*          1:K-1 are not referenced.
*
*          On exit, the elements on and above the diagonal contain
*          the n by n upper triangular part of the matrix R. The
*          elements below the diagonal in columns k:n, together with
*          TAU represent the orthogonal matrix Q as a product of
```

```
*          elementary reflectors (see Further Details).
*
*  LDA     (input) INTEGER
*          The leading dimension of the array A.  LDA >= max(1,M).
*
*  K       (input) INTEGER
*          The position of the first column deleted from B.
*          0 < K <= N+P.
*
*  P       (input) INTEGER
*          The number of columns deleted from B.  P > 0.
*
*  TAU     (output) DOUBLE PRECISION array, dimension(N-K+1)
*          The scalar factors of the elementary reflectors
*          (see Further Details).
*
*  WORK    DOUBLE PRECISION array, dimension (P+1)
*          Work space.
*
*  INFO    (output) INTEGER
*          = 0: successful exit
*          < 0: if INFO = -I, the I-th argument had an illegal value.
*
*  Further Details
*  ===============
*
*  The matrix Q is represented as a product of Q_B and elementary
*  reflectors
*
*     Q = Q_B * H(k) * H(k+1) *...* H(last), last = min( m-1, n ).
*
*  Each H(j) has the form
*
*     H(j) = I - tau*v*v'
*
*  where tau is a real scalar, and v is a real vector with
*  v(1:j-1) = 0, v(j) = 1, v(j+1:j+lenh-1), lenh = min( p+1, m-j+1 ),
*  stored on exit in A(j+1:j+lenh-1,j) and v(j+lenh:m) = 0, tau is
*  stored in  TAU(j).
*
*  The matrix Q can be formed with DELCOLSQ
*
*  =====================================================================
*
*     .. Parameters ..
```

```
      DOUBLE PRECISION   ONE
      PARAMETER          ( ONE = 1.0D+0 )
*     ..
*     .. Local Scalars ..
      DOUBLE PRECISION   AJJ
      INTEGER            J, LAST, LENH
*     ..
*     .. External Subroutines ..
      EXTERNAL           DLARF, DLARFG, XERBLA
*     ..
*     .. Intrinsic Functions ..
      INTRINSIC          MAX, MIN
*     ..
*
*     Test the input parameters.
*
      INFO = 0
      IF( M.LT.0 ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
         INFO = -4
      ELSE IF( K.GT.N+P .OR. K.LE.0 ) THEN
         INFO = -5
      ELSE IF( P.LE.0 ) THEN
         INFO = -6
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'DELCOLS', -INFO )
         RETURN
      END IF
*
      LAST = MIN( M-1, N )
*
      DO 10 J = K, LAST
*
*        Generate elementary reflector H(J) to annihilate the nonzero
*        entries below A(J,J)
*
         LENH = MIN( P+1, M-J+1 )
         CALL DLARFG( LENH, A( J, J ), A( J+1, J ), 1, TAU( J-K+1 ) )
*
         IF( J.LT.N ) THEN
*
```

```
*           Apply H(J) to trailing matrix from left
*
            AJJ = A( J, J )
            A( J, J ) = ONE
            CALL DLARF( 'L', LENH, N-J, A( J, J ), 1, TAU( J-K+1 ),
     $                  A( J, J+1 ), LDA, WORK )
            A( J, J ) = AJJ
*
         END IF
*
   10 CONTINUE
*
      RETURN
*
*     End of DELCOLS
*
      END
```

B delcolsq.f

```
      SUBROUTINE DELCOLSQ( M, N, A, LDA, Q, LDQ, K, P, TAU, WORK, INFO )
*
*     Craig Lucas, University of Manchester
*     March, 2004
*
*     .. Scalar Arguments ..
      INTEGER            INFO, K, LDA, LDQ, M, N, P
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION   A( LDA, * ), Q( LDQ, * ), TAU( * ), WORK( * )
*     ..
*
*  Purpose
*  =======
*
*  DELCOLSQ generates an m by m real matrix Q with orthogonal columns,
*  which is defined as the product of Q_B and elementary reflectors
*
*        Q = Q_B * H(k) * H(k+1) *...* H(last), last = min( m-1, n ) .
*
*  where the H(j) are as returned by DELCOLSQ, such that C = Q * R and
*  C is the matrix B = Q_B * R_B, with p columns deleted from the
*  kth column onwards.
*
*  Arguments
*  =========
*
*  M       (input) INTEGER
*          The number of rows of the matrix A.  M >= 0.
*
*  N       (input) INTEGER
*          The number of columns of the matrix A.  N >= 0.
*
*  A       (input) DOUBLE PRECISION array, dimension (LDA,N)
*          On entry, the elements below the diagonal in columns k:n
*          must contain the vector which defines the elementary
*          reflector H(J) as returned by DELCOLS.
*
*  LDA     (input) INTEGER
*          The leading dimension of the array A.  LDA >= max(1,M).
*
*  Q       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
*          On entry, the matrix Q_B.
```

```
*          On exit, the matrix Q.
*
*  LDQ     (input) INTEGER
*          The leading dimension of the array Q.  LDQ >= M.
*
*  K       (input) INTEGER
*          The position of the first column deleted from B.
*          0 < K <= N+P.
*
*  P       (input) INTEGER
*          The number of columns deleted from B.  P > 0.
*
*  TAU     (input) DOUBLE PRECISION array, dimension(N-K+1)
*          TAU(J) must contain the scalar factor of the elementary
*          reflector H(J), as returned by DELCOLS.
*
*  WORK    DOUBLE PRECISION array, dimension (P+1)
*          Work space.
*
*  INFO    (output) INTEGER
*          = 0: successful exit
*          < 0: if INFO = -I, the I-th argument had an illegal value.
*
*  =====================================================================
*
*     .. Parameters ..
      DOUBLE PRECISION   ONE
      PARAMETER          ( ONE = 1.0D+0 )
*     ..
*     .. Local Scalars ..
      DOUBLE PRECISION   AJJ
      INTEGER            J, LAST, LENH
*     ..
*     .. External Subroutines ..
      EXTERNAL           DLARF, XERBLA
*     ..
*     .. Intrinsic Functions ..
      INTRINSIC          MAX, MIN
*     ..
*
*     Test the input parameters.
*
      INFO = 0
      IF( M.LT.0 ) THEN
         INFO = -1
```

```
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
         INFO = -4
      ELSE IF( K.GT.N+P .OR. K.LE.0 ) THEN
         INFO = -5
      ELSE IF( P.LE.0 ) THEN
         INFO = -6
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'DELCOLSQ', -INFO )
         RETURN
      END IF
*
      LAST = MIN( M-1, N )
*
      DO 10 J = K, LAST
*
         LENH = MIN( P+1, M-J+1 )
*
*        Apply H(J) from right
*
         AJJ = A( J, J )
         A( J, J ) = ONE
*
         CALL DLARF( 'R', M, LENH, A( J, J ), 1, TAU( J-K+1 ),
     $               Q( 1, J ), LDQ, WORK )
*
         A( J, J ) = AJJ
*
   10 CONTINUE
*
      RETURN
*
*     End of DELCOLSQ
*
      END
```

C addcols.f

```
      SUBROUTINE ADDCOLS( M, N, A, LDA, K, P, TAU, WORK, LWORK, INFO )
*
*     Craig Lucas, University of Manchester
*     March, 2004
*
*     .. Scalar Arguments ..
      INTEGER            INFO, K, LDA, LWORK, M, N, P
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION   A( LDA, * ), TAU( * ), WORK( * )
*     ..
*
*  Purpose
*  =======
*
*  Given a real m by (n-p) matrix, B, and the QR factorization
*  B = Q_B * R_B, ADDCOLS computes the QR factorization
*  C = Q * R where C is the matrix B with p columns added
*  in the kth column onwards.
*
*  The input to this routine is Q_B' * C
*
*  Arguments
*  =========
*
*  M       (input) INTEGER
*          The number of rows of the matrix C.  M >= 0.
*
*  N       (input) INTEGER
*          The number of columns of the matrix C.  N >= 0.
*
*  A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
*          On entry, the matrix Q_B' * C. The elements in columns
*          1:K-1 are not referenced.
*
*          On exit, the elements on and above the diagonal contain
*          the n by n upper triangular part of the matrix R. The
*          elements below the diagonal in columns K:N, together with
*          TAU represent the orthogonal matrix Q as a product of
*          elementary reflectors and Givens rotations.
*          (see Further Details).
*
*  LDA     (input) INTEGER
```

```
*          The leading dimension of the array A.  LDA >= max(1,M).
*
*  K       (input) INTEGER
*          The position of the first column added to B.
*          0 < K <= N-P+1.
*
*  P       (input) INTEGER
*          The number of columns added to B.  P > 0.
*
*  TAU     (output) DOUBLE PRECISION array, dimension(P)
*          The scalar factors of the elementary reflectors
*          (see Further Details).
*
*  WORK    (workspace) DOUBLE PRECISION array, dimension ( LWORK )
*          Work space.
*
*  LWORK   (input) INTEGER
*          The dimension of the array WORK.  LWORK >= P.
*          For optimal performance LWORK >= P*NB, where NB is the
*          optimal block size.
*
*  INFO    (output) INTEGER
*          = 0: successful exit
*          < 0: if INFO = -I, the I-th argument had an illegal value.
*
*  Further Details
*  ===============
*
*  The matrix Q is represented as a product of Q_B, elementary
*  reflectors and Givens rotations
*
*     Q = Q_B * H(k) * H(k+1) *...* H(k+p-1) * G(k+p-1,k+p) *...
*         *G(k,k+1) * G(k+p,k+p+1) *...* G(k+2p-2,k+2p-1)
*
*  Each H(j) has the form
*
*     H(j) = I - tau*v*v'
*
*  where tau is a real scalar, and v is a real vector with
*  v(1:n-p-j+1) = 0, v(j) = 1, and v(j+1:m) stored on exit in
*  A(j+1:m,j), tau is stored in TAU(j).
*
*  Each G(i,j) has the form
*
*                 i-1  i
```

```
*              [ I          ]
*              [   c   -s   ] i-1
*     G(i,j) = [   s    c   ] i
*              [          I ]
*
*  and zero A(i,j), where c and s are encoded in scalar and
*  stored in A(i,j) and
*
*     IF A(i,j) = 1, c = 0, s = 1
*     ELSE IF | A(i,j) | < 1, s = A(i,j), c = sqrt(1-s**2)
*     ELSE c = 1 / A(i,j), s = sqrt(1-c**2)
*
*  The matrix Q can be formed with ADDCOLSQ
*
*  =====================================================================
*
*     .. Local Scalars ..
      DOUBLE PRECISION   C, S
      INTEGER            I, INC, ISTART, J, JSTOP, UPLEN
*     ..
*     .. External Subroutines ..
      EXTERNAL           DGEQRF, DLASR, DROT, DROTG, XERBLA
*     ..
*     .. Intrinsic Functions ..
      INTRINSIC          MAX, MIN
*     ..
*
*     Test the input parameters.
*
      INFO = 0
      IF( M.LT.0 ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
         INFO = -4
      ELSE IF( K.GT.N-P+1 .OR. K.LE.0 ) THEN
         INFO = -5
      ELSE IF( P.LE.0 ) THEN
         INFO = -6
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'ADDCOLS', -INFO )
         RETURN
      END IF
```

```
*
*     Do a QR factorization on rows below N-P, if there is more than one
*
      IF( M.GT.N-P+1 ) THEN
*
*        Level 3 QR factorization
*
         CALL DGEQRF( M-N+P, P, A( N-P+1, K ), LDA, TAU, WORK, LWORK,
     $                INFO )
*
      END IF
*
*     If K not equal to number of columns in B and not <= M-1 then
*     there is some elimination by Givens to do
*
      IF( K+P-1.NE.N .AND. K.LE.M-1) THEN
*
*        Zero out the rest with Givens
*        Allow for M < N
*
         JSTOP = MIN( P+K-1, M-1 )
         DO 20 J = K, JSTOP
*
*           Allow for M < N
*
            ISTART = MIN( N-P+J-K+1, M )
            UPLEN = N - K - P - ISTART + J + 1
*
            INC = ISTART - J
*
            DO 10 I = ISTART, J + 1, -1
*
*              Recall DROTG updates A( I-1, J ) and
*              stores C and S encoded as scalar in A( I, J )
*
               CALL DROTG( A( I-1, J ), A( I, J ), C, S )
               WORK( INC ) = C
               WORK( N+INC ) = S
*
*              Update nonzero rows of R
*              Do the next two line this way round because
*              A( I-1, N-UPLEN+1 ) gets updated
*
               A( I, N-UPLEN ) = -S*A( I-1, N-UPLEN )
               A( I-1, N-UPLEN ) = C*A( I-1, N-UPLEN )
```

```
*
               CALL DROT( UPLEN, A( I-1, N-UPLEN+1 ), LDA,
     $                    A( I, N-UPLEN+1 ), LDA, C, S )
*
               UPLEN = UPLEN + 1
               INC = INC - 1
*
   10       CONTINUE
*
*           Update inserted columns in one go
*           Max number of rotations is N-1, we've allowed N
*
            IF( J.LT.P+K-1 ) THEN
*

               CALL DLASR( 'L', 'V', 'B', ISTART-J+1, K+P-1-J,
     $                     WORK( 1 ), WORK( N+1 ), A( J, J+1 ), LDA )
*
            END IF
*
   20    CONTINUE
*
      END IF
      RETURN
*
*     End of ADDCOLS
*
      END
```

D addcolsq.f

```
      SUBROUTINE ADDCOLSQ( M, N, A, LDA, Q, LDQ, K, P, TAU, WORK, INFO)
*
*     Craig Lucas, University of Manchester
*     March, 2004
*
*     .. Scalar Arguments ..
      INTEGER            INFO, K, LDA, LDQ, M, N, P
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION   A( LDA, * ), Q( LDQ, * ), TAU( * ), WORK( * )
*     ..
*
*  Purpose
*  =======
*
*  ADDCOLSQ generates an m by m real matrix Q with orthogonal columns,
*  which is defined as the product of Q_B, elementary  reflectors and
*  Givens rotations
*
*     Q = Q_B * H(k) * H(k+1) *...* H(k+p-1) * G(k+p-1,k+p) *...
*         *G(k,k+1) * G(k+p,k+p+1) *...* G(k+2p-2,k+2p-1)
*
*  where the H(j) and G(i,j) are as returned by ADDCOLS, such that
*  C = Q * R and C is the matrix B = Q_B * R_B, with p columns added
*  from the kth column onwards.
*
*  Arguments
*  =========
*
*  M       (input) INTEGER
*          The number of rows of the matrix A.  M >= 0.
*
*  N       (input) INTEGER
*          The number of columns of the matrix A.  N >= 0.
*
*  A       (input) DOUBLE PRECISION array, dimension (LDA,N)
*          On entry, the elements below the diagonal in columns
*          K:K+P-1 (if M > M-P+1) must contain the vector which defines
*          the elementary reflector H(J). The elements above these
*          vectors and below the diagonal store the scalars such that
*          the Givens rotations can be constructed, as returned by
*          ADDCOLS.
*
```

```
*  LDA     (input) INTEGER
*          The leading dimension of the array A.  LDA >= max(1,M).
*
*  Q       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
*          On entry, the matrix Q_B.
*          On exit, the matrix Q.
*
*  LDQ     (input) INTEGER
*          The leading dimension of the array Q.  LDQ >= M.
*
*  K       (input) INTEGER
*          The postion of first column added to B.
*          0 < K <= N-P+1.
*
*  P       (input) INTEGER
*          The number columns added.  P > 0.
*
*  TAU     (output) DOUBLE PRECISION array, dimension(N-K+1)
*          The scalar factors of the elementary reflectors.
*
*  WORK    (workspace) DOUBLE PRECISION array, dimension (2*N)
*          Work space.
*
*  INFO    (output) INTEGER
*          = 0: successful exit
*          < 0: if INFO = -I, the I-th argument had an illegal value
*
*  =====================================================================
*
*     .. Parameters ..
      DOUBLE PRECISION   ONE, ZERO
      PARAMETER          ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*     ..
*     .. Local Scalars ..
      DOUBLE PRECISION   DTEMP
      INTEGER            COL, I, INC, ISTART, J, JSTOP
*     ..
*     .. External Subroutines ..
      EXTERNAL           DLARF, DLASR, XERBLA
*     ..
*     .. Intrinsic Functions ..
      INTRINSIC          ABS, MAX, MIN, SQRT
*
*     Test the input parameters.
*
```

```
      INFO = 0
      IF( M.LT.0 ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
         INFO = -4
      ELSE IF( K.GT.N-P+1 .OR. K.LE.0 ) THEN
         INFO = -5
      ELSE IF( P.LE.0 ) THEN
         INFO = -6
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'ADDCLQ', -INFO )
         RETURN
      END IF
*
*     We did a QR factorization on rows below N-P+1
*
      IF( M.GT.N-P+1 ) THEN
*
         COL = N - P + 1
         DO 10 J = K, K + P - 1
*
            DTEMP = A( COL, J )
            A( COL, J ) = ONE
*
*           If  N+P > M-N we have only factored the first M-N columns.
*
            IF( M-COL+1.LE.0 )
     $         GO TO 10
            CALL DLARF( 'R', M, M-COL+1, A( COL, J ), 1, TAU( J-K+1 ),
     $                  Q( 1, COL ), LDQ, WORK )
*
            A( COL, J ) = DTEMP
            COL = COL + 1
*
   10    CONTINUE
      END IF
*
*     If K not equal to number of columns in B then there was
*     some elimination by Givens
*
      IF( K+P-1.LT.N .AND. K.LE.M-1 ) THEN
*
```

```
*        Allow for M < N, i.e DO P wide unless hit the bottom first
*
         JSTOP = MIN( P+K-1, M-1 )
         DO 30 J = K, JSTOP
*
            ISTART = MIN( N-P+J-K+1, M )
            INC = ISTART - J
*
*           Compute vectors of C and S for rotations
*
            DO 20 I = ISTART, J + 1, -1
*
               IF( A( I, J ).EQ.ONE ) THEN
                  WORK( INC ) = ZERO
                  WORK( N+INC ) = ONE
               ELSE IF( ABS( A( I, J ) ).LT.ONE ) THEN
                  WORK( N+INC ) = A( I, J )
                  WORK( INC ) = SQRT( ( 1-A( I, J )**2 ) )
               ELSE
                  WORK( INC ) = ONE / A( I, J )
                  WORK( N+INC ) = SQRT( ( 1-WORK( INC )**2 ) )
               END IF
               INC = INC - 1
   20       CONTINUE
*
*           Apply rotations to the Jth column from the right
*
            CALL DLASR( 'R', 'V', 'b', M, ISTART-I+1, WORK( 1 ),
     $                  WORK( N+1 ), Q( 1, I ), LDQ )
*
   30    CONTINUE
*
      END IF
      RETURN
*
*     End of ADDCOLS
*
      END
```

References

[1] Å. Björck. *Numerical Methods for Least Squares Problems.* SIAM, Philadelphia, PA, USA, 1996.

[2] Å. Björck, L. Eldén, and H. Park. Accurate downdating of least squares solutions. *SIAM J. Matrix Anal. Appl.*, 15:549–568, 1994.

[3] A. W. Bojanczyk, R. P. Brent, P. Van Dooren, and F. R. De Hoog. A note on downdating the Cholesky factorization. *SIAM J. Sci. Stat. Comput.*, 8 (3):210–221, 1987.

[4] Adam Bojanczyk, Nicholas J. Higham, and Harikrishna Patel. Solving the indefinite least squares problem by hyperbolic QR factorization. *SIAM J. Matrix Anal. Appl.*, 24(4):914–931, 2003.

[5] J. M. Chambers. Regression updating. *Journal of the American Statistical Association*, 66(336):744–748, 1971.

[6] J. W. Daniel, W. B. Gragg, L. Kaufman, and G. W. Stewart. Reorthogonalization and stable algorithms for updating the Gram–Schmidt QR factorization. *Mathematics of Computation*, 30(136):772–795, 1976.

[7] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart. *LINPACK Users' Guide.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1979.

[8] L. Eldén and H. Park. Block downdating of least squares solutions. *SIAM J. Matrix Anal. Appl.*, 15:1018–1034, 1994.

[9] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Third edition, The Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[10] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms.* Second edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.

[11] MATLAB, version 6.5. The Mathworks Inc, Natick, MA, USA.

[12] NAG Fortran Library Manual, Mark 20. Numerical Algorithms Group, Oxford, UK.

[13] S. J. Olszanskyj, J. M. Lebak, and A. W. Bojanczyk. Rank-k modificaton methods for recursive least squares problems. *Numerical Algorithms*, 7:325–354, 1994.

[14] B. N. Parlett. Analysis of algorithms for reflections in bisectors. *SIAM Review*, 13:197–208, 1971.

[15] C. M. Rader and A. O. Steinhardt. Hyperbolic Householder transforms. *SIAM J. Matrix Anal. Appl.*, 9(2):269–290, 1988.

[16] L. Reichel and W. B. Gragg. Algorithm 686: FORTRAN subroutines for updating the QR decomposition. *ACM Trans. Math. Soft.*, 16:369–377, 1990.

[17] M. A. Saunders. Large-scale linear programming using the cholesky factorization. Technical Report CS252, Computer Science Department, Stanford University, CA, 1972.

[18] Robert Schreiber and Charles van Loan. A storage-efficient wy representation for products of householder transformations. *SIAM J. Sci. Stat. Comput.*, 10(1):53–57, 1989.

[19] G. W. Stewart. The effects of rounding error on an algorithm for downdating a Cholesky factorization. *Journal of the Institute of Mathematics and its Applications*, 23(2):203–213, 1979.

[20] G. W. Stewart. On the stability of sequential updates and downdates. *IEEE Trans. Signal Processing*, 43(11):2642–2648, 1995.