Reflections, Rotations and QR Factorization

QR Factorization figures in *Least-Squares* problems and *Singular-Value Decompositions* among other things numerical. These notes explain some reflections and rotations that do it, and offer MATLAB implementations; in its notation, $x' :=$ (complex conjugate transpose of x) .

Householder Reflections

A *Householder Reflection* is $W = I - w·w' = W' = W^{-1}$ for any column w satisfying $w'·w = 2$. If $y = W·x$ then $y'·y = x'·x$, and $y'·x = x'·y$ is real even if x, y and w are complex. W is a reflection because $W·w = -w$ and $W·p = p$ whenever $w'·p = 0$. Numerical analysts name this reflection after Alston S. Householder because he introduced it to them in the mid 1950s as part of an improved way to solve *Least-Squares* problems.

Given columns x and $e := [1, 0, 0, …, 0]'$ so that $e'·x = x_1$, we seek column w so that $w'·w = 2$ and $W := I - w·w'$ reflects x to $W·x = e·ß$ for some scalar $ß$. Its $|ß| = ||x|| :=\sqrt{}(x'·x)$ and $x'·W·x = x'·e·ß = x_1'·ß$ must be real, so $ß = ±||x||·x_1/|x_1|$ if $x_1 \neq 0$.

Construction: Set $Ñ := ||x||$, $ß := ±Ñ·x_1/|x_1|$, $d := x - e·ß$, and $w := d/\sqrt{}(d'·d/2)$ unless $d = o$, in which case set $w := o$. But all bets are off if UNDERFLOW degrades $x'·x$.

Proof: Let $p := x + e·ß$ so that $p'·d = 0 = p'·w$. Then $W·d = -d$ and $W·p = p$, whereupon $2W·x = W·(p+d) = p-d = 2e·ß$ as desired.

How is the sign $±$ in $ß$ chosen? The simplest way maximizes $Ω^2 := d'·d/2 = Ñ·(Ñ - (±)|x_1|)$ by setting $ß := -Ñ·x_1/|x_1|$, as we'll see. Of course, any $±$ sign works when $x_1 = 0$.

Detailed Construction: Let $v := x - e·x_1$, so that $e'·v = v_1 = 0$, and let $µ := v'·v > 0$, so that $Ñ := \sqrt{}(x'·x) = \sqrt{}(µ + |x_1|^2)$. Next set $ç := x_1/|x_1| = \mathrm{sign}(x_1)$ except that we reset $ç := 1$ if $x_1 = 0$. Next we choose $ß := ±ç·Ñ$. Numerical stability requires two cases to be distinguished:

If $ß = -ç·Ñ$ set $d := x - e·ß = x + e·ç·Ñ$ by copying x to d and then resetting
$$d_1 := x_1 - ß = ç·(|x_1| + Ñ) .$$

If $ß = +ç·Ñ$ set $d := x - e·ß = x - e·ç·Ñ$ by copying x to d and then resetting
$$d_1 := x_1 + ß = -ç·µ/(|x_1| + Ñ) .$$

Next $Ω := \sqrt{}\big((|d_1|^2 + µ)/2 \big) = \sqrt{}\big(|d_1|·Ñ \big)$ and $w := d/Ω$. Return $[w, ß] = \mathrm{hshldrw}(x)$.

QR Factorization:
Given an m-by-n matrix F with no fewer rows than columns (so $m \ge n$), we wish to factorize $F = Q\cdot R$, with $Q'\cdot Q = I$ and R upper-triangular, by using Householder reflections thus: $W_n\cdot\ldots\cdot W_2\cdot W_1\cdot F = \begin{bmatrix} R \\ O \end{bmatrix}$ in which each reflection $W_j = W_j' = W_j^{-1}$ is constructed to annihilate all subdiagonal elements in column j of $F_{j-1} := W_{j-1}\cdot\ldots\cdot W_2\cdot W_1\cdot F$. Then $Q := W_1\cdot W_2\cdot\ldots W_n\cdot \begin{bmatrix} I \\ O \end{bmatrix}$. Each $W_j = I - w_j\cdot w_j'$ has $w_j'\cdot w_j = 2$ (or 0) and no nonzero element in w_j above row j . Each $F_j = W_j\cdot F_{j-1}$ has the same first j–1 rows as F_{j-1} and no nonzero subdiagonal elements in its first j columns. Each $Q_j := W_j\cdot W_{j+1}\cdot\ldots W_n\cdot \begin{bmatrix} I \\ O \end{bmatrix}$ has ones on the diagonal and zeros elsewhere in its first j–1 rows and columns, so $Q_j = W_j\cdot Q_{j+1}$ is obtained by altering only rows and columns of Q_{j+1} with indices no less than j .

Detailed Construction in MATLAB:
Start with $F_0 := F$. For $j = 1, 2, \ldots, n$ in turn get $[\overline{w}_j, ß_j] :=$ hshldrw(F_j(j:m, j)) as above, and store $\overline{w}_j$ in place of F_j(j:m, j) ; then if $j < n$ overwrite $F_j(j{:}m, j{+}1{:}n) - \overline{w}_j\cdot(\overline{w}_j'\cdot F_j(j{:}m, j{+}1{:}n))$ onto F_j(j:m, j+1:n) to get F_{j+1}(j:m, j+1:n) .

Next, R := Diag([$ß_1, ß_2, \ldots, ß_n$]) + triu(F_n(1:n, 1:n), 1) .

Finally, set $G_{n+1} := F_n$ and, for $j = n, n-1, \ldots, 1$ in turn, extract $\overline{w}_j$ from G_{j+1}(j:m, j) , overwrite column $[o_{j-1}; 1; o_{m-j}]$ onto G_{j+1}(:, j) , and then onto G_{j+1}(j:m, j:n) overwrite $G_j(j{:}m, j{:}n) := G_{j+1}(j{:}m, j{:}n) - \overline{w}_j\cdot(\overline{w}_j'\cdot G_{j+1}(j{:}m, j{:}n))$. Then $Q := G_1$.

Return [Q, R] = hshldrqr(F) .

Numerical experiments indicate that MATLAB uses the same method to get [Q, R] = qr(F, 0) .

QR Factorization by Givens Rotations

A Givens Rotation is $Q := \begin{bmatrix} c & s \\ -s' & c \end{bmatrix}$ so chosen that a 2-vector $v = \begin{bmatrix} x \\ y \end{bmatrix}$ is rotated to $Q\cdot v = \begin{bmatrix} r \\ 0 \end{bmatrix}$ wherein $|r|^2 = v'\cdot v$, so $c^2 + s'\cdot s = 1$ when (by convention) we choose $c \ge 0$. Here v' is the complex conjugate transpose of v , and s' is the complex conjugate of s . The rotation is named after Wallace Givens who introduced this rotation to numerical analysts in the 1950s while he was working at Argonne National Labs near Chicago. The rotation is encoded in one complex number $t := (y/x)'$ from which are derived $c := 1/\sqrt{(1 + t'\cdot t)}$, $s := c\cdot t$ and $r := x/c$. In the special case that $t = \infty$ (presumably because $x = 0$), we set $c := 0$, $s := 1$ and $r := y$. In any event, note that $Q^{-1} = Q'$. Return [c, s, t, r] = givenst(x, y) .

Bottom-Up QR Factorization:
Given an m-by-n matrix F with no fewer rows than columns (so $m \ge n$), we wish to factorize $F = Q \cdot R$, with $Q' \cdot Q = I$ and R upper-triangular, by using Givens rotations thus:

For $1 \le i \le m-1$ and $1 \le j \le n$ let Q_{ij} be the Givens rotation that acts upon an m-by-n matrix Z to overwrite $Q_{ij} \cdot \begin{bmatrix} z_{i,j} \\ z_{i+1,j} \end{bmatrix} = \begin{bmatrix} r_{i,j} \\ 0 \end{bmatrix}$ onto $\begin{bmatrix} z_{i,j} \\ z_{i+1,j} \end{bmatrix}$. We shall premultiply F by a sequence of rotations Q_{ij} in this order (from right to left):

for j = 1 up to n in turn { for i = m–1 down to j in turn { premultiply by Q_{ij} }}.

Since each Q_{ij} affects only rows i and i+1 of columns j to n of the product, we may store t_{ij} in place of the product’s zero element in position (i+1, j) since it will not figure in subsequent premultiplications. After the last premultiplication we find R in the product’s first n rows and columns after ignoring the subdiagonal elements that hold t_{ij}s . Then these are used to construct Q as a product of inverse rotations Q_{ij}' premultiplying $\begin{bmatrix} I \\ O \end{bmatrix}$ in this reverse order (from right to left):

for j = n down to 1 in turn { for i = j up to m–1 in turn { premultiply by Q_{ij}' }}.

Each premultiplication by Q_{ij}' affects only rows i and i+1 of columns j to n of the product after t_{ij} was extracted from location (i+1, j) and replaced by 0 .

Return [Q, R] = gvnsupqr(F) .

This is not the only way to use Givens rotations for QR factorizations. Another is …

Top-Down QR Factorization:
Given an m-by-n matrix F with no fewer rows than columns (so $m \ge n$), we wish to factorize $F = Q \cdot R$, with $Q' \cdot Q = I$ and R upper-triangular, by using Givens rotations thus:

For $1 \le j \le n$ and $j+1 \le i \le m$ let Q_{ij} be the Givens rotation that acts upon an m-by-n matrix Z to overwrite $Q_{ij} \cdot \begin{bmatrix} z_{j,j} \\ z_{i,j} \end{bmatrix} = \begin{bmatrix} r_{j,j} \\ 0 \end{bmatrix}$ onto $\begin{bmatrix} z_{j,j} \\ z_{i,j} \end{bmatrix}$. We shall premultiply F by a sequence of rotations Q_{ij} in this order (from right to left):

for j = 1 up to n in turn { for i = j+1 up to m in turn { premultiply by Q_{ij} }}.

Since each Q_{ij} affects only rows i and j of columns j to n of the product, we may store t_{ij} in place of the product’s zero element in position (i, j) since it will not figure in subsequent premultiplications. After the last premultiplication we find R in the product's first n rows and columns after ignoring the subdiagonal elements that hold t_{ij}s . Then these are used to construct Q as a product of inverse rotations Q_{ij}' premultiplying $\begin{bmatrix} I \\ O \end{bmatrix}$ in this reverse order (from right to left):

for j = n down to 1 in turn { for i = m down to j+1 in turn { premultiply by Q_{ij}' }}.

Each premultiplication by Q_{ij}' affects only rows i and j of columns j to n of the product after t_{ij} was extracted from location (i, j) and replaced by 0 .

Return [Q, R] = gvnsdnqr(F) .

MATLAB appears to use Householder reflections to get its [Q, R] = qr(F, 0) . There is no reason to expect any two of the three different [Q, R] factorizations to agree though they must be related in the absence of roundoff: $R_1 \cdot R_2^{-1} = Q_1' \cdot Q_2$ must be a diagonal unitary matrix.

= =

MATLAB Programs

```
function  [F, R] = hshldrqr(F)
%  [Q, R] = hshldrqr(F)  uses  Householder Reflections  to
%  factorize  F = Q*R  so that  R  is upper-triangular and
%  Q  has orthonormal columns;  Q'*Q = I .  This works only
%  if  F  has no more columns than rows,  and if underflow
%  does not degrade  F'*F .  Uses  hshldrw.m .
[m, n] = size(F) ;
if (m < n),
    error(' F  has more columns than rows in  hshldrqr(F).'),  end
z = zeros(1, n) ;  w = zeros(m, 1) ;
for  j = 1:n
    [w, z(j)] = hshldrw(F(j:m, j)) ;
    F(j:m, j) = w ;
    if (j < n),
        F(j:m, j+1:n) = F(j:m, j+1:n) - w*(w'*F(j:m, j+1:n)) ;
  end,  end % ... j = 1:n
R = diag(z) + triu(F(1:n, 1:n), 1) ;
for  j = n:-1:1
    w = F(j:m, j) ;  F(:, j) = zeros(m,1) ;  F(j, j) = 1 ;
    F(j:m, j:n) = F(j:m, j:n) - w*(w'*F(j:m, j:n)) ;
  end % ... j = n:-1:1

% = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
```

```
function  [w, z] = hshldrw(x)
%  [w, z] = hshldrw(x)  yields  w  with  w'·w = 2  or  0 ,
%  so  W = I - w*w' = W' = W^-1  reflects the given column
%  x  to  W*x = [z; 0; 0; ...; 0]  with  |z| = norm(x) .
%  But all bets are off if  UNDERFLOW  degrades  x'*x .
w = x(:) ;  m = length(w) ;  x1 = w(1) ;  a1 = abs(x1) ;
if (m < 2),  w = 0 ;  z = x1 ;  return,  end
if (a1),  s = x1/a1 ;  else  s = 1 ;  end
vv = w(2:m)'*w(2:m) ;  ax = sqrt(a1*a1 + vv) ;
z = -s*ax ;  a1 = a1 + ax ;  w(1) = s*a1 ;
dd2 = a1*ax ;
if (dd2), w = w/sqrt(dd2) ;  end  %...  so  w'*w = 2  unless  w = o .

% = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =


function  [c, s, t, r] = givenst(x, y)
%  [c, s, t, r] = givenst(x, y)  satisfies  c >= 0 ,
%  c^2 + |s|^2 = 1 ,  r = c*x + s*y ,  t' = x/y = s'/c .
%  So  [c     s]* [x]  =  [r]   and  c = 1/sqrt(1 + t'*t)
%      [-s'   c]  [y]     [0]        s = c*t ,  r = x/c .
if (x ~= 0)
    t = conj(y/x) ;  u = sqrt(1 + t'*t) ;
    r = u*x ;  c = 1/u ;  s = c*t ;
  else
    t = inf ;  r = y ;  c = 0 ;  s = 1 ;
  end

% = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

function  [F, R] = gvnsupqr(F)
%  [Q, R] = gvnsupqr(F)  uses  Givens Rotations  to
%  factorize  F = Q*R  so that  R  is upper-triangular and
%  Q  has orthonormal columns;  Q'*Q = I .  This works only
%  if  F  has no more columns than rows,  and if underflow
%  does not degrade  F'*F .  Uses  givenst.m  bottom-up.
[m, n] = size(F) ;
if (m < n),
    error(' F  has more columns than rows in  gvnsupqr(F).'),  end
for  j = 1:n ,  for  i = m-1:-1:j
    [c, s, F(i+1, j), F(i, j)] = givenst(F(i, j), F(i+1, j)) ;
    if (j < n),
        F(i:i+1, j+1:n) = [c, s; -s', c]*F(i:i+1, j+1:n) ;  end
  end,  end % ... i = m-1:-1:j ,  j = 1:n
R = triu(F(1:n, 1:n)) ;
for  j = n:-1:1 ,  F(1:j, j) = zeros(j,1) ;  F(j,j) = 1 ;
    for i = j:m-1
        t = F(i+1, j) ;  F(i+1, j) = 0 ;  c = 1/sqrt(1 + t'*t) ;
        if (c~=0),  s = c*t ;  else  s = 1 ;  end
        F(i:i+1, j:n) = [c, -s; s', c]*F(i:i+1, j:n) ;
  end,  end % ... i = j:m-1,  j = n:-1:1

% = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
```

```
function  [F, R] = gvnsdnqr(F)
%  [Q, R] = gvnsdnqr(F)  uses  Givens Rotations  to
%  factorize  F = Q*R  so that  R  is upper-triangular and
%  Q  has orthonormal columns;  Q'*Q = I .  This works only
%  if  F  has no more columns than rows,  and if underflow
%  does not degrade  F'*F .  Uses  givenst.m  top-down.
[m, n] = size(F) ;
if (m < n),
    error(' F  has more columns than rows in  gvnsdnqr(F).'),  end
for  j = 1:n ,  for  i = j+1:m
    [c, s, F(i, j), F(j, j)] = givenst(F(j, j), F(i, j)) ;
    if (j < n),
        F([j,i], j+1:n) = [c, s; -s', c]*F([j,i], j+1:n) ;  end
  end,  end % ... i = j+1:m ,  j = 1:n
R = triu(F(1:n, 1:n)) ;
for  j = n:-1:1 ,  F(1:j, j) = zeros(j,1) ;  F(j,j) = 1 ;
    for i = m:-1:j+1
        t = F(i, j) ;  F(i, j) = 0 ;  c = 1/sqrt(1 + t'*t) ;
        if (c~=0),  s = c*t ;  else  s = 1 ;  end
        F([j,i], j:n) = [c, -s; s', c]*F([j,i], j:n) ;
  end,  end % ... i = m:-1:j+1,  j = n:-1:1

% = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
```