```
% MATLAB implementation of the "Generalized Tridiagonal Decomposition"
%               Version 1.1, January 25, 2005
% Input:
%
% H = U*S*V' (singular value decomposition of H)
%       U and V orthonormal columns,
%       S diagonal matrix with positive diagonal entries
%       r desired diagonal of R
%        --length (r) = nnz (S)
%        --r multiplicatively majorized by diag (S)
%        --product r = product diag (S)
%
% Output:
%
% H = Q*R*P' (GTD based on r)
%       P and Q orthonormal columns
%       R upper triangular, R (i, i) = r (i)
%

function [Q, R, P] = gtd (U, S, V, r)

d = diag (S) ;
if any (d <= 0)
   error ('a diagonal element of S is <= 0') ;
end
K = length (d) ;

% Check if r satisfies the multiplicative majorization condition

cr = cumprod (flipud (sort (abs (r (:))))) ;
cd = cumprod (flipud (sort (d))) ;
if any (cr (1:K-1)./cd (1:K-1) - 1 > 10*K*eps) | ...
      abs (cr(K)-cd(K))/cd(K) > 10*K*eps
   error('r is not multiplicatively majorized by diag (S)') ;
end

P = V ;
Q = U ;
R = zeros (K) ;
for k = 1 : K-1
   rk = r (k) ;
   abs_rk = abs (rk) ;
   kp1 = k + 1 ;
   km1 = k - 1 ;
```

```
I = find (abs (d (k : K)) > abs_rk) ;
if ( isempty (I) )
   [x, p] = max (abs (d (k : K))) ;
   p = p + km1 ;
   d ([k p]) = d ([p k]) ;
else
   I = I + km1 ;
   [x, p] = min (abs (d (I))) ;
   p = I (p) ;
   d ([k p]) = d ([p k]) ;
   I = find (abs (d (kp1: K)) <= abs_rk) ;
   if ( isempty (I) )
      d ([p k]) = d ([k p]) ;
      [x, p] = min (abs (d (k : K))) ;
      p = p + km1 ;
      d ([p k]) = d ([k p]) ;
   else
      I = I + k ;
      [x, q] = max (abs (d (I))) ;
      q = I (q) ;
      delta2 = d (q) ;
   end
end
delta1 = d (k) ;
if ( isempty (I) ) % interchange columns P, Q, R
   R (1:km1, [k p]) = R (1:km1, [p k]) ;
   R (k, k) = rk ;
   P (:, [k p]) = P (:, [p k]) ;
   Q (:, [k p]) = Q (:, [p k]) ;
   t = rk'*delta1/(abs (rk)*abs (delta1)) ;
   Q (:, k) = Q (:, k)*t ;
   continue ;
end
d ([kp1 q]) = d ([q kp1]) ;
sq_delta1 = abs (delta1)^2 ;
sq_delta2 = abs (delta2)^2 ;
sq_rk = abs_rk^2 ;
denom = sq_delta1 - sq_delta2 ;
c = sqrt ((sq_rk - sq_delta2)/denom) ;
s = sqrt (1-c*c) ;
x = -s*c*rk*denom/sq_rk ;
y = delta2*((delta1*rk)/sq_rk) ;
G1 = [ c -s
     s  c ] ;
```

```
   G2 = [ c*delta1 -s*(delta2')
        s*delta2  c*(delta1') ] ;
   G2 = ( (rk')/sq_rk) * G2 ;
   if ( k > 1 )
      % permute the columns
      R (1:km1, [k p]) = R (1:km1, [p k]) ;
      R (1:km1, [kp1 q]) = R (1:km1, [q kp1]) ;

      % apply G1 to R
      R (1:km1, [k kp1]) = R (1:km1, [k kp1])*G1 ;
   end
   R (k, k) = rk ;
   R (k, kp1) = x ;
   d (kp1) = y ;

   % permute the columns
   P (:, [k p]) = P (:, [p k]) ;
   P (:, [kp1 q]) = P (:, [q kp1]) ;
   Q (:, [k p]) = Q (:, [p k]) ;
   Q (:, [kp1 q]) = Q (:, [q kp1]) ;

   % apply G1 to P
   P (:, [k kp1]) = P (:, [k kp1])*G1 ;

   % apply G2 to Q
   Q (:, [k kp1]) = Q (:, [k kp1])*G2 ;
end

R (K, K) = r (K) ;
if ( r (K) ~= 0. )
   Q (:, K) = Q (:, K)*d (K)/ r (K) ;
end
P = P (:, 1:K) ;
Q = Q (:, 1:K) ;
```