2020-02-19

Orthogonal transformations and Gram-Schmidt

We saw in the last lecture that a natural decomposition for thinking about least squares problems is the QR decomposition

$$A = QR,$$

where Q is an $m \times m$ orthogonal matrix and R is an $m \times n$ upper triangular matrix. Equivalently, we can write the "economy" version of the decomposition, $A = QR$ with an $m \times n$ matrix Q and an $n \times n$ upper triangular R, where the columns of Q form an orthonormal basis for the range space of A. Using this decomposition, we can solve the least squares problem via the triangular system

$$Rx = Q^T b.$$

The *Gram-Schmidt* procedure is usually the first method people learn to convert some existing basis (columns of A) into an orthonormal basis (columns of Q). For each column of A, the procedure subtracts off any components in the direction of the previous columns, and then scales the remainder to be unit length. In MATLAB, Gram-Schmidt looks something like this:

```
function [Q] = orth_cgs(A)

  [m,n] = size(A);
  Q = zeros(m,n);
  for j = 1:n
    v = A(:,j);                 % Take the jth original basis vector
    v = v-Q(:,1:j-1)*(Q(:,1:j-1)'*v); % Make it orthogonal to q_i, i = 1:j-1
    v = v/norm(v);              % Normalize what remains
    Q(:,j) = v;                 % Append the result to the basis
  end
```

Where does R appear in this algorithm? It appears thus:

```
function [Q,R] = orth_cgs(A)

  [m,n] = size(A);
  Q = zeros(m,n);
  for j = 1:n
    v = A(:,j);         % Take the jth original basis vector
```

```
7     rp = Q(:,1:j-1)'*v; % Project v onto previous basis vectors
8     v = v-Q(:,1:j-1)*rp; % Make it orthogonal to q_i, i = 1:j-1
9     R(1:j-1,j) = rp;   % Update R with multipliers
10    R(j,j) = norm(v);  % Get the normalizing factor
11    v = v/R(j,j);      % Normalize what remains
12    Q(:,j) = v;        % Append the result to the basis
13  end
```

That is, R accumulates the multipliers that we computed from the Gram-Schmidt procedure. This idea that the multipliers in an algorithm can be thought of as entries in a matrix should be familiar, since we encountered it before when we looked at Gaussian elimination.

Householder transformations

The Gram-Schmidt orthogonalization procedure is not generally recommended for numerical use. Suppose we write $A = [a_1 \ldots a_m]$ and $Q = [q_1 \ldots q_m]$. The essential problem is that if $r_{jj} \ll \|a_j\|_2$, then cancellation can destroy the accuracy of the computed q_j; and in particular, the computed q_j may not be particularly orthogonal to the previous q_j. Actually, loss of orthogonality can build up even if the diagonal elements of R are not exceptionally small. This is Not Good, and while we have some tricks to mitigate the problem, we need a different approach if we want the problem to go away.

Recall that one way of expressing the Gaussian elimination algorithm is in terms of Gauss transformations that serve to introduce zeros into the lower triangle of a matrix. *Householder* transformations are orthogonal transformations (reflections) that can be used to similar effect. Reflection across the plane orthogonal to a unit normal vector v can be expressed in matrix form as

$$H = I - 2vv^T.$$

Now suppose we are given a vector x and we want to find a reflection that transforms x into a direction parallel to some unit vector y. The right reflection is through a hyperplane that bisects the angle between x and y (see Figure 1), which we can construct by taking the hyperplane normal to

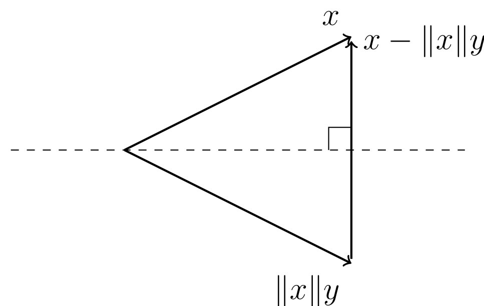

Figure 1: Construction of a reflector to transform x into $\|x\|y$, $\|y\| = 1$.

$x - \|x\|y$. That is, letting $u = x - \|x\|y$ and $v = u/\|u\|$, we have

$$\begin{aligned}(I - 2vv^T)x &= x - 2\frac{(x + \|x\|y)(x^T x + \|x\|x^T y)}{\|x\|^2 + 2x^T y\|x\| + \|x\|^2\|y\|^2} \\ &= x - (x - \|x\|y) \\ &= \|x\|y.\end{aligned}$$

If we use $y = \pm e_1$, we can get a reflection that zeros out all but the first element of the vector x. So with appropriate choices of reflections, we can take a matrix A and zero out all of the subdiagonal elements of the first column.

Now think about applying a sequence of Householder transformations to introduce subdiagonal zeros into A, just as we used a sequence of Gauss transformations to introduce subdiagonal zeros in Gaussian elimination. This leads us to the following algorithm to compute the QR decomposition:

```
1  function [Q,R] = hqr1(A)
2    % Compute the QR decomposition of an m-by-n matrix A using
3    % Householder transformations.
4
5    [m,n] = size(A);
6    Q = eye(m);     % Orthogonal transform so far
7    R = A;          % Transformed matrix so far
8
9    for j = 1:n
10
11     % -- Find H = I-tau*w*w' to put zeros below R(j,j)
12     normx = norm(R(j:end,j));
```

```
13      s    = -sign(R(j,j));
14      u1   = R(j,j) - s*normx;
15      w    = R(j:end,j)/u1;
16      w(1) = 1;
17      tau  = -s*u1/normx;
18
19      % -- R := HR, Q := QH
20      R(j:end,:) = R(j:end,:)-(tau*w)*(w'*R(j:end,:));
21      Q(:,j:end) = Q(:,j:end)-(Q(:,j:end)*w)*(tau*w)';
22
23    end
```

Note that there are two valid choices of u_1 at each step; we make the choice that avoids cancellation in the obvious version of the formula.

As with LU factorization, we can re-use the storage of A by recognizing that the number of nontrivial parameters in the vector w at each step is the same as the number of zeros produced by that transformation. This gives us the following:

```
1   function [A,tau] = hqr2(A)
2     % Compute the QR decomposition of an m-by-n matrix A using
3     % Householder transformations, re-using the storage of A
4     % for the Q and R factors.
5
6     [m,n] = size(A);
7     tau = zeros(n,1);
8
9     for j = 1:n
10
11      % -- Find H = I-tau*w*w' to put zeros below A(j,j)
12      normx      = norm(A(j:end,j));
13      s          = -sign(A(j,j));
14      u1         = A(j,j) - s*normx;
15      w          = A(j:end,j)/u1;
16      w(1)       = 1;
17      A(j+1:end,j) = w(2:end);   % Save trailing part of w
18      A(j,j)     = s*normx;      % Diagonal element of R
19      tau(j)     = -s*u1/normx;
20
21      % -- R := HR
22      A(j:end,j+1:end) = A(j:end,j+1:end)-...
23                         (tau(j)*w)*(w'*A(j:end,j+1:end));
24
25    end
```

If we ever need Q or Q^T explicitly, we can always form it from the compressed representation. We can also multiply by Q and Q^T implicitly:

```
function QX = applyQ(QR,tau,X)

  [m,n] = size(QR);
  QX = X;
  for j = n:-1:1
    w = [1; QR(j+1:end,j)];
    QX(j:end,:) = QX(j:end,:)-(tau(j)*w)*(w'*QX(j:end,:));
  end
```

```
function QTX = applyQT(QR,tau,X)

  [m,n] = size(QR);
  QTX = X;
  for j = 1:n
    w = [1; QR(j+1:end,j)];
    QTX(j:end,:) = QTX(j:end,:)-(tau(j)*w)*(w'*QTX(j:end,:));
  end
```

Givens rotations

Householder reflections are one of the standard orthogonal transformations used in numerical linear algebra. The other standard orthogonal transformation is a *Givens rotation*:

$$G = \begin{bmatrix} c & -s \\ s & c \end{bmatrix}.$$

where $c^2 + s^2 = 1$. Note that

$$G = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} cx - sy \\ sx + cy \end{bmatrix}$$

so if we choose

$$s = \frac{-y}{\sqrt{x^2 + y^2}}, \qquad c = \frac{x}{\sqrt{x^2 + y^2}}$$

then the Givens rotation introduces a zero in the second column. More generally, we can transform a vector in $\mathbb{R}^m$ into a vector parallel to e_1 by a sequence of $m - 1$ Givens rotations, where the first rotation moves the

last element to zero, the second rotation moves the second-to-last element to zero, and so forth.

For some applications, introducing zeros one by one is very attractive. In some places, you may see this phrased as a contrast between algorithms based on Householder reflections and those based on Givens rotations, but this is not quite right. Small Householder reflections can be used to introduce one zero at a time, too. Still, in the general usage, Givens rotations seem to be the more popular choice for this sort of local introduction of zeros.