Week 7: Wednesday, Oct 7

Householder QR

Householder transformations are simple orthogonal transformations corresponding to reflection through a plane. Reflection across the plane orthogonal to a unit normal vector v can be expressed in matrix form as

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

At the end of last lecture, we drew a picture to show how we could construct a reflector that maps a given vector x to be a scalar multiple of another given vector y. In particular, if we take $u = x - s\|x\|e_1$ where $s = \pm 1$ and $v = u/\|u\|$ then

$$Hx = \left(I - 2\frac{uu^T}{u^Tu}\right) x = s\|x\|e_1.$$

Let us first verify that this works:

$$\begin{aligned}
u^Tx &= (x - s\|x\|e_1)^Tx \\
&= \|x\|^2 - sx_1\|x\| \\
u^Tu &= (x - s\|x\|e_1)^T(x - s\|x\|e_1) \\
&= \|x\|^2 - 2sx_1\|x\| + \|x\|^2\|e_1\|^2 \\
&= 2(\|x\|^2 - sx_1\|x\|) = 2u^Tx \\
Hx &= x - 2u\frac{u^Tx}{u^Tu} = x - u = s\|x\|e_1.
\end{aligned}$$

As a byproduct of this calculation, note that we have

$$u^Tu = -2s\|x\|u_1,$$

where $u_1 = x_1 - s\|x\|$; and if we define $w = u/u_1$, we have

$$H = I - 2\frac{ww^T}{w^Tw} = I + \frac{su_1}{\|x\|}ww^T = I - \tau ww^T,$$

where $\tau = -su_1/\|x\|$.

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:

```
function [Q,R] = lec18hqr1(A)
  % Compute the QR decomposition of an m-by-n matrix A using
  % Householder transformations.

  [m,n] = size(A);
  Q = eye(m);      % Orthogonal transform so far
  R = A;           % Transformed matrix so far

  for j = 1:n

    % -- Find H = I-tau*w*w' to put zeros below R(j,j)
    normx = norm(R(j:end,j));
    s     = -sign(R(j,j));
    u1    = R(j,j) - s*normx;
    w     = R(j:end,j)/u1;
    w(1)  = 1;
    tau   = -s*u1/normx;

    % -- R := HR, Q := QH
    R(j:end,:) = R(j:end,:)-(tau*w)*(w'*R(j:end,:));
    Q(:,j:end) = Q(:,j:end)-(Q(:,j:end)*w)*(tau*w)';

  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:

```
function [A,tau] = lec18hqr2(A)
  % Compute the QR decomposition of an m-by-n matrix A using
  % Householder transformations, re-using the storage of A
  % for the Q and R factors.

  [m,n] = size(A);
  tau = zeros(n,1);
```

```
  for j = 1:n

    % -- Find H = I-tau*w*w' to put zeros below A(j,j)
    normx        = norm(A(j:end,j));
    s            = -sign(A(j,j));
    u1           = A(j,j) - s*normx;
    w            = A(j:end,j)/u1;
    w(1)         = 1;
    A(j+1:end,j) = w(2:end);       % Save trailing part of w
    A(j,j)       = s*normx;        % Diagonal element of R
    tau(j)       = -s*u1/normx;

    % -- R := HR
    A(j:end,j+1:end) = A(j:end,j+1:end)-...
                       (tau(j)*w)*(w'*A(j:end,j+1:end));

  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 = lec18applyQ(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 = lec18applyQT(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
```

Conditioning and sensitivity

Recall that we were motivated to look at the QR factorization because of the connection to linear least squares. We noted that if $A = QR$ then

$$\|Ax - b\|_2^2 = \|Rx - Q^T b\|_2^2,$$

and if we write

$$R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$$

where R_1 is a square upper triangular matrix, then we minimize $\|Rx - \tilde{b}\|$ ($\tilde{b} = Q^T b$) precisely by solving the triangular linear system $R_1 x = b_1$. Because $\kappa(R) = \kappa(A)$ and $\kappa(A^T A) = \kappa(A)^2$, we expect the linear system involved in this QR-based method to be much less sensitive than the linear system that appears in the normal equations. But this only tells us about the sensitivity of some linear system of equations that is *related* to our original least squares problem. What about the sensitivity of the solution to the actual least squares problem with respect to perturbations in the coefficient matrix and the right hand side?

To derive a sensitivity estimate, let us consider the relation between infinitesimal perturbations to A, x, and b in the normal equations $A^T Ax = A^T b$:

$$(\delta A^T A + A^T \delta A)x + A^T A\, \delta x = \delta A^T b + A^T \delta b.$$

We can rearrange this to get

$$\delta x = (A^T A)^{-1} A^T (\delta b - \delta A\, x) + (A^T A)^{-1} \delta A^T (b - Ax).$$

The first piece of δx involves an instance of $(A^T A)^{-1}$ and A^T together, and we might reasonably (and correctly) expect it to contribute a term involving $\kappa(A)$ in our final relative sensitivity bound. The second piece, on the other hand, still involves $(A^T A)^{-1}$ without the mitigating effects of A, and we might reasonably (and again correctly) guess that this leads to a term involving $\kappa(A)^2$. We will spell out the details in the next lecture.