Notes for 2016-03-11

Woah, We're Halfway There

Last time, we showed that the QR iteration maps upper Hessenberg matrices to upper Hessenberg matrices, and this fact allows us to do one QR sweep in $O(n^2)$ time. We did not, however, discuss how to get to upper Hessenberg form. This lecture is devoted to that reduction, and to other "halfway there" forms:

$A = QHQ^T$ where Q is orthogonal and H upper Hessenberg
$A = QTQ^T$ where Q is orthogonal, T symmetric tridiagonal, and A symmetric
$A = UBV^T$ where U and V are orthogonal and B is upper bidiagonal

reduction of Unlike the SVD or various eigendecompositions, these forms can be computed directly. And in many cases, as we will discuss, they are just as useful!

Hessenberg via Householder

Let's start with an over-ambitious goal[1]: can we directly compute the Schur form by applying Householder transformations on the left and right of a matrix? We already secretly know that the answer is no, but trying and failing will set us up neatly for computing a Hessenberg form — and for understanding why the Hessenberg form is in fact a natural target.

How might we go about trying to compute a Schur form directly? A natural first step is to apply an orthogonal transformation from the left in order to introduce zeros in the first column, and then apply the same transformation on the right. Let's try this for the 4×4 matrix case (we'll mark with a star each nonzero element that is updated by the previous transformation):

$$\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \mapsto \begin{bmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix} \mapsto \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix}.$$

[1]Ah, but a man's reach should exceed his grasp...

Oops! Applying the transformation from the right scrambles the columns, and undoes the zeros we started just introduced. The problem is that we got too greedy. What if instead of trying to zero out everything below the diagonal, we instead zero out everything below the first subdiagonal? Then we affect rows 2 through n, and a subsequent transform affecting columns 2 through n does not kill the zeros we introduced:

$$\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \mapsto \begin{bmatrix} \times & \times & \times & \times \\ * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix} \mapsto \begin{bmatrix} \times & * & * & * \\ \times & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix}.$$

A second step to introduce zeros in the second column finishes the reduction to Hessenberg form:

$$\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \mapsto \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & * & * & * \\ 0 & 0 & * & * \end{bmatrix} \mapsto \begin{bmatrix} \times & \times & * & * \\ \times & \times & * & * \\ 0 & \times & * & * \\ 0 & 0 & * & * \end{bmatrix}.$$

Let's make this concrete:

```
% [H] = hess_house(A)
%
% Reduce square matrix A to upper Hessenberg form

function A = hess_house(A)

  [n,~] = size(A);
  for j = 1:n-2
    u = reflector(A(j+1:end,j));
    A(j+1:n,j:n) = A(j+1:n,j:n) - 2*u*(u'*A(j+1:n,j:n));
    A(:,j+1:n) = A(:,j+1:n) - 2*(A(:,j+1:n)*u)*u';
    A(j+2:n,j) = 0;
  end
```

Terrific Tridiagonals and Busy Bidiagonals

When A is a symmetric matrix, the corresponding Hessenberg form is symmetric as well. This means that all the entries below the first subdiagonal are zero (by upper Hessenberg structure), and so are all the entries above the first superdiagonal (by symmetry).

What about bidiagonalization? In bidiagonalization, we again interleave transformations on the left and right, but now we allow ourselves to use different transformations. The key is that we want to keep introducing new zero elements through these transformations without destroying zeros we already created. In the 4×4 case, the first two steps look like

$$\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \mapsto \begin{bmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix} \mapsto \begin{bmatrix} \times & * & 0 & 0 \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix}$$

The code is

```
% [H] = bidiag_house(A)
%
% Reduce square matrix A to bidiagonal form

function A = bidiag_house(A)

  [n,~] = size(A);
  for j = 1:n-1

    % Introduce zeros in column j (below diagonal)
    u = reflector(A(j:end,j));
    A(j:n,j:n) = A(j:n,j:n) - 2*u*(u'*A(j:n,j:n));

    % Introduce zeros in row j (right of superdiagonal)
    u = reflector(A(j,j+1:end)');
    A(j:n,j+1:n) = A(j:n,j+1:n) - 2*(A(j:n,j+1:n)*u)*u';

  end
```

Using the Factorizations

Let's look concretely at two cases where these factorizations are useful. The first is in computing *transfer functions* that occur in control theory, where we have the form

$$T(s) = c^T(sI - A)^{-1}b + d$$

for various values of s. Naively, it looks like we might have to spend $O(n^3)$ per value of s where we want the transfer function; but we can instead use the Hessenberg reduction $A = QHQ^T$ to get

$$T(s) = (c^TQ)(sI - H)^{-1}(Q^Tb) + d$$

Solving a Hessenberg linear system like $(sI - H)$ can be done in $O(n^2)$ time rather than $O(n^3)$ time (why?); in fact, this is one of the structures that MATLAB's sparse solver checks for.

A second application again deals with parameter-dependent systems, but in a different setting. Suppose $A = UBV^T$ is a bidiagonal reduction of A, and we are interested in computing the Tikhonov-regularized solution for different values of the regularization parameter. How can we do this efficiently? Note that

$$\left\| \begin{bmatrix} A \\ \lambda I \end{bmatrix} x - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\| = \left\| \begin{bmatrix} B \\ \lambda I \end{bmatrix} (V^T x) - \begin{bmatrix} U^T b \\ 0 \end{bmatrix} \right\|$$

The latter equations can be reduced back to bidiagonal form in $O(n)$ time (left as an exercise for the student!). Therefore, after the initial bidiagonal reduction, we can solve the least squares problem for each new value of the regularization parameter λ in $O(n)$ additional work.

Enter Arnoldi

When we discussed QR factorization, we started with Gram-Schmidt and then moved to the Householder-based method. In talking about reduction to Hessenberg form, we started with the Householder approach — but there is something akin to Gram-Schmidt as well. We can read the basic idea by looking at the columns of the Hessenberg matrix equation $AQ = QH$:

$$Aq_j = \sum_{k=1}^{j} q_j h_{jk} + h_{j,j+1} q_{j+1}.$$

Rearranging, we have

$$q_{j+1} = \frac{1}{h_{j,j+1}} \left(Aq_j - \sum_{k=1}^{j} q_k h_{kj} \right)$$

where $h_{kj} = q_k^T A q_j$. That is, the coefficients H can be exactly derived from using Gram-Schmidt orthgonalization to orthonormalize Aq_j against $\{q_1, \ldots, q_j\}$ for each successive j!

The beauty about using the Arnoldi procedure to reduce a matrix to upper Hessenberg form is two fold: we can readily apply the method to sparse matrices; and we can *stop early!*. This latter fact is true of the Householder-based methods as well. But in the case of the Arnoldi procedure, we will end up making use of partial Hessenberg reduction to turn Arnoldi into a method of approximating linear system solutions (GMRES) and a method of approximating eigenpairs for large systems (the Arnoldi method).

When A is symmetric, the Arnoldi procedure becomes the *Lanczos* procedure. Note that in this case, we *tridiagonalize* the original matrix, which means at each step we need (in exact arithmetic) to orthogonalize each successive Aq_j against only two other vectors. This rather remarkable fact allows us to parley the Lanczos iteration into an eigenvalue iteration as well as two truly remarkable iterations for solving linear systems: MINRES and the famous method of conjugate gradients (CG). We will return to these methods in about a week.