Week 10–11: Friday, Oct 30 and Monday, Nov 2

Orthogonal iteration revisited

Last time, we described a generalization of the power methods to compute invariant subspaces. That is, starting from some initial subspace $\mathcal{V}^{(0)}$, we defined the sequence

$$\mathcal{V}^{(k+1)} = A\mathcal{V}^{(k)} = A^{k+1}\mathcal{V}^{(0)}.$$

Under some assumptions, the spaces $\mathcal{V}^{(k)}$ asymptotically converge to an invariant subspace of A. In order to actually implement this iteration, though, we need a concrete representation of each of the subspaces in terms of a basis. In the case of the power method, we normalized our vector at each step to have unit length. The natural generalization in the case of subspace iteration is to normalize our bases to be orthonormal at each step. If the columns of $V^{(k)}$ form an orthonormal basis for $\mathcal{V}^{(k)}$, then the columns of $AV^{(k)}$ form an orthonormal basis for $\mathcal{V}^{(k+1)}$; and we can compute an orthonormal basis $V^{(k+1)}$ for $\mathcal{V}^{(k+1)}$ by an economy QR decomposition:

$$V^{(k+1)}R^{(k+1)} = AV^{(k)}.$$

This *orthogonal iteration* gives us a sequence of orthonormal bases $V^{(k+1)}$ for the spaces $\mathcal{V}^{(k)}$.

We also mentioned in the last lecture that orthogonal iteration has the marvelous property that it runs subspace iteration *for a sequence of nested subspaces*. Using MATLAB notation, we have that for any l,

$$V^{(k+1)}(:, 1:l)R^{(k+1)}(1:l, 1:l) = AV^{(k)}(:, 1:l).$$

So by running orthogonal iteration on an m-dimensional subspace, we magically *also* run orthogonal iteration on an l-dimensional subspaces for each $l \leq m$. Recall that the Schur factorization

$$AU = UT$$

involves an orthonormal basis U such that for any l, $U(:, 1:l)$ spans an l-dimensional invariant subspace (this is from the triangularity of T). So we might hope that if we ran orthogonal iteration on *all* of A, we would

eventually converge to the U matrix in the Schur factorization. That is, starting from $\underline{Q}^{(0)} = I$, we iterate

$$\underline{Q}^{(k+1)} R^{(k+1)} = A\underline{Q}^{(k)} \tag{1}$$

in the hopes that the columns of $\underline{Q}^{(k)}$, since they span nested bases undergoing subspace iteration, will converge to the unitary factor U.

Now, consider the first two steps of this iteration:

$$\begin{aligned} \underline{Q}^{(1)} R^{(1)} &= A \\ \underline{Q}^{(2)} R^{(2)} &= A\underline{Q}^{(1)} \end{aligned}$$

If we multiply the second equation on the right by $R^{(1)}$, we have

$$\underline{Q}^{(2)} R^{(2)} R^{(1)} = A\underline{Q}^{(1)} R^{(1)} = A^2. \tag{2}$$

Similarly, if we multiply $\underline{Q}^{(3)} R^{(3)} = A\underline{Q}^{(3)}$ by $R^{(2)} R^{(1)}$, we have

$$\underline{Q}^{(3)} R^{(3)} R^{(2)} R^{(1)} = A\underline{Q}^{(2)} R^{(2)} R^{(1)} = A^3,$$

where the last equality comes from (2). We can keep going in this fashion, and if we define the upper triangular matrix $\underline{R}^{(k)} = R^{(k)} R^{(k-1)} \dots R^{(1)}$, we have

$$\underline{Q}^{(k)} \underline{R}^{(k)} = A^k.$$

That is, $\underline{Q}^{(k)}$ is precisely the unitary factor in a QR decomposition of A^k. This fact may be unsurprising if we consider that we derived this orthogonal iteration from the power method.

From orthogonal iteration to the QR iteration

Suppose we have the Schur form $AU = UT$, or equivalently $T = U^* AU$. Now, the matrices $A^{(k)} = (\underline{Q}^{(k)})^* A (\underline{Q}^{(k)})$ are each unitarily similar to A, and if the matrices $\underline{Q}^{(k)}$ converge to U, then they $A^{(k)}$ must converge to T. More generally, if the first span of the first l columns of $\underline{Q}^{(k)}$ converges to an invariant subspace basis of A, then $A^{(k)}$ should at least converge to a block upper triangular matrix with zeros below the leading $l \times l$ block.

We can think of $A^{(k)}$ as being defined by the relation

$$A\underline{Q}^{(k)} = \underline{Q}^{(k)} A^{(k)}.$$

But we have seen the expression $A\underline{Q}^{(k)}$ before, in the iteration equation (1). So we really have

$$\underline{Q}^{(k+1)} R^{(k+1)} = \underline{Q}^{(k)} A^{(k)}.$$

Now, suppose we multiply the equation on the left by $(\underline{Q}^{(k)})^*$ and define $Q^{(k+1)} = (\underline{Q}^{(k)})^* \underline{Q}^{(k+1)}$. Then

$$Q^{(k+1)} R^{(k+1)} = A^{(k)}.$$

So we can compute $Q^{(k+1)}$ and $R^{(k+1)}$ by a QR factorization of $A^{(k)}$, and from there compute $\underline{Q}^{(k+1)} = Q^{(k+1)} \underline{Q}^{(k)}$. What has this to do with anything? Let us consider what happens when we compute $A^{(k+1)}$ using these formulas:

$$\begin{aligned}
A^{(k+1)} &= (\underline{Q}^{(k+1)})^* A (\underline{Q}^{(k+1)}) \\
&= (Q^{(k+1)})^* (\underline{Q}^{(k)})^* A (\underline{Q}^{(k)}) (Q^{(k+1)}) \\
&= (Q^{(k+1)})^* A^{(k)} (Q^{(k+1)}) \\
&= R^{(k+1)} Q^{(k+1)}.
\end{aligned}$$

Putting things together, we have the remarkable *QR iteration* that goes from $A^{(k)}$ to $A^{(k+1)}$:

$$\begin{aligned}
Q^{(k+1)} R^{(k+1)} &= A^{(k)} \\
A^{(k+1)} &= R^{(k+1)} Q^{(k+1)}.
\end{aligned}$$

Under some restrictions on A, the matrices $A^{(k)}$ will ultimately converge to the triangular Schur factor of A. But we have two problems:

1. Each step of the QR iteration requires a QR factorization, which is an $O(n^3)$ operation. This is rather expensive, and even in the happy case where we might be able to get each eigenvalue with a constant number of steps, $O(n)$ total steps at a cost of $O(n^3)$ each gives us an $O(n^4)$ algorithm. Given that everything else we have done so far costs only $O(n^3)$, an $O(n^4)$ cost for eigenvalue computation seems excessive.

2. Like the power iteration upon which it is based, the basic iteration converges linearly, and the rate of convergence is related to the ratios of the moduli of eigenvalues. Convergence is slow when there are eigenvalues of nearly the same modulus, and nonexistent when there are eigenvalues with the same modulus.

We now describe how to overcome these two difficulties.

Hessenberg matrices and QR steps in $O(n^2)$

A matrix H is said to be *upper Hessenberg* if it has nonzeros only in the upper triangle and the first subdiagonal. For example, the nonzero structure of a 5-by-5 Hessenberg matrix is

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}.$$

For any square matrix A, we can find a unitarily similar Hessenberg matrix $H = Q^* A Q$ by the following algorithm (see for comparison the Householder QR code in lecture 18):

```
function [H,Q] = lec27hess(A)
  % Compute the Hessenberg decomposition H = Q'*A*Q using
  % Householder transformations.

  n = length(A);
  Q = eye(n);      % Orthogonal transform so far
  H = A;           % Transformed matrix so far

  for j = 1:n-2

    % -- Find W = I-2vv' to put zeros below H(j+1,j)
    u    = H(j+1:end,j);
    u(1) = u(1) + sign(u(1))*norm(u);
    v    = u/norm(u);
```

```
    % -- H := WHW', Q := QW
    H(j+1:end,:) = H(j+1:end,:)-2*v*(v'*H(j+1:end,:));
    H(:,j+1:end) = H(:,j+1:end)-(H(:,j+1:end)*(2*v))*v';
    Q(:,j+1:end) = Q(:,j+1:end)-(Q(:,j+1:end)*(2*v))*v';

  end
```

A Hessenberg matrix H is very nearly upper triangular, and so the Householder QR routine is very economical. The Householder reflection computed in order to introduce a zero in the $(j+1, j)$ entry needs only to operate on rows j and $j+1$. Therefore, we have

$$Q^* H = W_{n-1} W_{n-2} \dots W_1 H = R,$$

where W_j is a Householder reflection that operates only on rows j and $j+1$. Computing R costs $O(n^2)$ time, since each W_j only affects two rows ($O(n)$ data). Now, note that

$$RQ = R(W_1 W_2 \dots W_{n-1});$$

that is, RQ is computed by an operation that first mixes the first two columns, then the second two columns, and so on. The only subdiagonal entries that can be introduced in this process lie on the first subdiagonal, and so RQ is again a Hessenberg matrix. Therefore, one step of QR iteration on a Hessenberg matrix results in another Hessenberg matrix, and a Hessenberg QR step can be performed in $O(n^2)$ time.

Putting these ideas in concrete form, we have the following code

```
function H = lec27hessqr(H)
  % Basic Hessenberg QR step via Householder transformations.

  n = length(H);
  V = zeros(2,n-1);

  % Compute the QR factorization
  for j = 1:n-1

    % -- Find W_j = I-2vv' to put zero into H(j+1,j)
    u      = H(j:j+1,j);
```

```
  u(1)   = u(1) + sign(u(1))*norm(u);
  v      = u/norm(u);
  V(:,j) = v;

  % -- H := W_j H
  H(j:j+1,:) = H(j:j+1,:)-2*v*(v'*H(j:j+1,:));

end

% Compute RQ
for j = 1:n-1

  % -- H := WHW', Q := QW
  v = V(:,j);
  H(:,j:j+1) = H(:,j:j+1)-(H(:,j:j+1)*(2*v))*v';

end
```