Week 11: Friday, Nov 6

Logistics

1. Two course advertisements:
   (a) CS 5220: Applications of Parallel Computers (Bindel)
   (b) Math 6140: Differential Games, Optimal Control, Front Propagation, and Dynamic Programming (Vladimirsky)

Double-shift QR steps

Last time, we discussed the Wilkinson strategy of choosing as a shift one of the roots of the trailing 2-by-2 submatrix of $A^{(k)}$ (the one closest to the final entry). We also noted that if we want to convert to *real* Schur form, the Wilkinson shift has the distinct disadvantage that it might launch us into the complex plane. The Francis shift strategy is to simultaneously apply a complex conjugate pair of shifts, essentially computing two steps together:

$$\begin{aligned} Q^{(k)}R^{(k)} &= (A^{(k-1)} - \sigma_k I)(A^{(k-1)} - \bar{\sigma}_k I) \\ &= (A^{(k-1)})^2 - 2\Re(\sigma_k)A^{(k-1)} + |\sigma_k|^2 I \\ A^{(k)} &= (Q^{(k)})^* A^{(k-1)} (Q^{(k)}). \end{aligned}$$

When the Wilkinson shift is real, we let σ_k be the same as the Wilkinson shift; when the Wilkinson strategy leads to a conjugate pair of possible shifts, we use both, maintaining efficiency by doing the steps *implicitly*. Let's now make this implicit magic a little more explicit by building code for an implicit double-shift QR step.

Our first step will be to construct the polynomial associated with the Francis double-shift. In the case where the trailing 2-by-2 submatrix (or 2-by-2 block Rayleigh quotient, if one prefers) has a complex pair of eigenvalues, we just use its characteristic polynomial. Otherwise, we use the polynomial associated with two steps with a Wilkinson shift.

```
function [b,c] = lec29qrpoly(H)
  % Compute b, c s.t. z^2 + b*z + c = (z-sigma)(z-conj(sigma))
  % where sigma is the Wilkinson double shift
```

```
% Get shifts via trailing submatrix
HH    = H(end-1:end,end-1:end);
trHH  = HH(1,1)+HH(2,2);
detHH = HH(1,1)*HH(2,2)-HH(1,2)*HH(2,1);

if trHH^2 > 4*detHH   % Real eigenvalues

  % Use the one closer to H(n,n)
  lHH(1) = (trHH + sqrt(trHH^2-4*detHH))/2;
  lHH(2) = (trHH - sqrt(trHH^2-4*detHH))/2;
  if abs(lHH(1)-H(end,end)) < abs(lHH(2)-H(end,end))
    lHH(2) = lHH(1);
  else
    lHH(1) = lHH(2);
  end

  % z^2 + bz + c = (z-sigma_1)(z-sigma_2)
  b = -lHH(1)-lHH(2);
  c =  lHH(1)*lHH(2);

else

  % In the complex case, we want the char poly for HH
  b = -trHH;
  c = detHH;

end
```

The code `lec29qrpoly` gives us coefficients b_k and c_k for a quadratic function $s_k(z) = z^2 + b_k z + c_k$. We now want to compute

$$\begin{aligned} Q^{(k)} R^{(k)} &= s_k(A^{(k-1)}) = (A^{(k-1)})^2 + b_k A^{(k-1)} + c_k I \\ A^{(k)} &= (Q^{(k)})^* A^{(k-1)} (Q^{(k)}). \end{aligned}$$

The trick is to realize that all the iterates $A^{(k)}$ are Hessenberg, and the Hessenberg form for a matrix is usually unique (up to signs). Therefore, we

compute the first Householder transformation W in a QR factorization of $s_k(A^{(k)}$ explicitly. The first column of $Q^{(k)}$ is the same as the first column of W. The remaining columns of $Q^{(k)}$ can be determined by the requirement that $A^{(k)}$ is in Hessenberg form. We compute them implicitly by applying the usual Hessenberg reduction algorithm to $B = WA^{(k-1)}W$, taking advantage of the fact that B has special structure to do $O(n^2)$ work. Each step of the reduction moves a "bulge" down the diagonal by one.

```
function [H] = lec29qrstep(H)
  % Implicit QR step using a Francis double shift
  % (there should really be some re-scalings for floating point)

  % Compute double-shift poly and initial column of H^2 + b*H + c*I
  [b,c]   = lec29qrpoly(H);
  C1      = H(1:3,1:2)*H(1:2,1);
  C1(1:2) = C1(1:2) + b*H(1:2,1);
  C1(1)   = C1(1)   + c;

  % Apply a similarity associated with the first step of QR on C
  v        = house(C1);
  H(1:3,:) = H(1:3,:)-2*v*(v'*H(1:3,:));
  H(:,1:3) = H(:,1:3)-(H(:,1:3)*(2*v))*v';

  % Do "bulge chasing" to return to Hessenberg form
  % (compare to lec27hess).
  %
  n = length(H);
  for j = 1:n-2
    k = min(j+3,n);

    % -- Find W = I-2vv' to put zeros below H(j+1,j), H := WHW'
    v          = house(H(j+1:k,j));
    H(j+1:k,:) = H(j+1:k,:)-2*v*(v'*H(j+1:k,:));
    H(:,j+1:k) = H(:,j+1:k)-(H(:,j+1:k)*(2*v))*v';
    H(k,j)     = 0;

  end
```

In the LAPACK codes, the Francis double-shift strategy is mixed with some "exceptional shifts" that occur every few iterations. These exceptional shifts serve to keep the algorithm from getting stuck in certain pathological situations (e.g. a cyclic permutation matrix).

Deflation

A sequence of implicit doubly-shifted QR steps with the Francis shift will usually give us rapid convergence of a trailing 1-by-1 or 2-by-2 submatrix to a block of a Schur factorization. As this happens, the trailing row (or two rows) becomes very close to zero. When the values in these rows are close enough to zero, we *deflate* by setting them equal to zero. This corresponds to a small perturbation to the original problem.

The following code converts a Hessenberg matrix to a block upper triangular matrix with 1-by-1 and 2-by-2 blocks. To reduce this matrix further to real Schur form, we would need to make an additional pass to further reduce any 2-by-2 block with real eigenvalues into a pair of 1-by-1 blocks.

```
function [H] = lec29qr(H)

  n = length(H);
  tol = norm(H,'fro') * 1e-8;
  k = 0;
  while n > 2
    if abs(H(n,n-1)) < tol
      fprintf('At step %d: Deflated 1-by-1 block\n', k);
      H(n,n-1) = 0;
      n = n-1;
    elseif abs(H(n-1,n-2)) < tol
      fprintf('At step %d: Deflated 2-by-2 block\n', k);
      H(n-1,n-2) = 0;
      n = n-2;
    else
      H(1:n,1:n) = lec29qrstep(H(1:n,1:n));
      k = k+1;
    end
  end
```

More careful deflation criteria are usually used in practice; see the book. This criterion at least corresponds to small normwise perturbations to the original problem, but it may result in less accurate estimates of small eigenvalues than we could obtain with a more aggressive criterion.

Stability of the method

Each step of the implicitly double-shifted QR iteration changes the matrix only with orthogonal transformations (which are perfectly conditioned) or deflations. Hence, the QR iteration is backward stable. However, this is *not* the same as saying that the method is forward stable! For forward stability, the conditioning of the eigenvalues is critical, and multiple (or nearly multiple) eigenvalues of multiplicity m usually inherit an $O(\epsilon^{1/m})$ error, as we saw in our earlier discussion of sensitivity.

The intermediate computations in the QR code as given above are prone to scaling problems, and so the basic QR codes in LAPACK (`dlahqr`) uses a more careful construction of a scaled copy of the first Householder transformation.

The state of the art

The current state of the art in QR iterations is the LAPACK code `dgehqr` written by Ralph Byers, which is based on an award-winning set of papers by Braman, Byers, and Mathias. This code uses the following general strategy:

1. Run the basic QR iteration to find the eigenvalues of a trailing $b \times b$ submatrix. Apply the transformations to the whole matrix, resulting in a "spike" to the left of the triangularized portion.

2. Look for converged eigenvalues in the trailing submatrix by analyzing the "spike" to find small elements. Deflate any eigenvalues found (and there may be several). This is called *aggressive early deflation*.

3. Use several of the remaining eigenvalues from the Rayleigh quotient block as a sequence of successive shifts. These can be run simultaneously by chasing a sequence of closely-spaced bulges down the main diagonal. The similarity transformations associated are applied in a blocky way to get good cache performance.