2019-10-30

1 Inverse iteration and the QR method

When we discussed the power method, we found that we could improve convergence by a spectral transformation that mapped the eigenvalue we wanted to something with large magnitude (preferably much larger than the other eigenvalues). This was the *shift-invert* strategy. We already know there is a connection leading from the power method to orthogonal iteration to the QR method, which we can summarize with a small number of formulas. Let us see if we can follow the same path to uncover a connection from inverse iteration (the power method with A^{-1}, a special case of shift-invert in which the shift is zero) to QR. If we call the orthogonal factors in orthogonal iteration $\underline{Q}^{(k)}$ ($\underline{Q}^{(0)} = I$) and the iterates in QR iteration $A^{(k)}$, we have

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

$$A^{(k)} = (\underline{Q}^{(k)})^* A (\underline{Q}^{(k)}). \tag{2}$$

In particular, note that because $R^{(k)}$ are upper triangular,

$$A^k e_1 = (\underline{Q}^{(k)} e_1) r_{11}^{(k)};$$

that is, the first column of $\underline{Q}^{(k)}$ corresponds to the kth step of power iteration starting at e_1. What happens when we consider negative powers of A? Inverting (1), we find

$$A^{-k} = (\underline{R}^{(k)})^{-1} (\underline{Q}^{(k)})^*$$

The matrix $\tilde{R}^{(k)} = (\underline{R}^{(k)})^{-1}$ is again upper triangular; and if we look carefully, we can see in this fact another power iteration:

$$e_n^* A^{-k} = e_n^* \tilde{R}^{(k)} (\underline{Q}^{(k)})^* = \tilde{r}_{nn}^{(k)} (\underline{Q}^{(k)} e_n)^*.$$

That is, the last column of $\underline{Q}^{(k)}$ corresponds to a power iteration converging to a *row* eigenvector of A^{-1}.

2 Shifting gears

The connection from inverse iteration to orthogonal iteration (and thus to QR iteration) gives us a way to incorporate the shift-invert strategy into QR

iteration: simply run QR on the matrix $A - \sigma I$, and the (n, n) entry of the iterates (which corresponds to a Rayleigh quotient with an increasingly-good approximate row eigenvector) should start to converge to $\lambda - \sigma$, where λ is the eigenvalue nearest σ. Put differently, we can run the iteration:

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

If we choose a good shift, then the lower right corner entry of $A^{(k)}$ should converge to the eigenvalue closest to σ in fairly short order, and the rest of the elements in the last row should converge to zero.

The shift-invert power iteration converges fastest when we choose a shift that is close to the eigenvalue that we want. We can do even better if we choose a shift *adaptively*, which was the basis for running Rayleigh quotient iteration. The same idea is the basis for the *shifted QR iteration*:

$$Q^{(k)} R^{(k)} = A^{(k-1)} - \sigma_k I \tag{3}$$

$$A^{(k)} = R^{(k)} Q^{(k)} + \sigma_k I. \tag{4}$$

This iteration is equivalent to computing

$$\begin{aligned} \underline{Q}^{(k)} \underline{R}^{(k)} &= \prod_{j=1}^{n} (A - \sigma_j I) \\ A^{(k)} &= (\underline{Q}^{(k)})^* A (\underline{Q}^{(k)}) \\ \underline{Q}^{(k)} &= Q^{(k)} Q^{(k-1)} \dots Q^{(1)}. \end{aligned}$$

What should we use for the shift parameters σ_k? A natural choice is to use $\sigma_k = e_n^* A^{(k-1)} e_n$, which is the same as $\sigma_k = (\underline{Q}^{(k)} e_n)^* A (\underline{Q}^{(k)} e_n)$, the Rayleigh quotient based on the last column of $\underline{Q}^{(k)}$. This simple shifted QR iteration is equivalent to running Rayleigh iteration starting from an initial vector of e_n, which we noted before is locally quadratically convergent.

3 Double trouble

The simple shift strategy we described in the previous section gives *local* quadratic convergence, but it is not *globally* convergent. As a particularly pesky example, consider what happens if we want to compute a complex

conjugate pair of eigenvalues of a real matrix. With our simple shifting strategy, the QR iteration never produce a complex iterate, a complex shift, or a complex eigenvalue. The best we can hope for is that our initial shift is closer to both eigenvalues in the conjugate pair than it is to anything else in the spectrum; in this case, we will most likely find that the last two columns of $\underline{Q}^{(k)}$ are converging to a basis for an *invariant row subspace* of A, and the corresponding eigenvalues are the eigenvalues of the trailing 2-by-2 sub-block.

Fortunately, we know how to compute the eigenvalues of a 2-by-2 matrix! This suggests the following shift strategy: let σ_k be one of the eigenvalues of $A^{(k)}(n-1:n, n-1:n)$. Because this 2-by-2 problem can have complex roots even when the matrix is real, this shift strategy allows the possibility that we could converge to complex eigenvalues. On the other hand, if our original matrix is real, perhaps we would like to consider the *real* Schur form, in which U is a real matrix and T is block diagonal with 1-by-1 and 2-by-2 diagonal blocks that correspond, respectively, to real and complex eigenvalues. If we shift with *both* roots of $A^{(k)}(n-1:n, n-1:n)$, equivalent to computing

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

There is one catch here: even if we started with $A^{(0)}$ in Hessenberg form, it is unclear how to do this double-shift step in $O(n^2)$ time!

The following fact will prove our salvation: if we Q and V are both orthogonal matrices and $Q^T AQ$ and $V^T AV$ are both (unreduced) Hessenberg[1]) and the first column of Q is the same as the first column of V, then all successive columns of Q are unit scalar multiples of the corresponding columns of V. This is the *implicit Q theorem*. Practically, it means that we can do any sort of shifted QR step we would like in the following way:

1. Apply as a similarity any transformations in the QR decomposition that affect the leading submatrix (1-by-1 or 2-by-2).

2. Restore the resulting matrix to Hessenberg form without further transformations to the leading submatrix.

In the first step, we effectively compute the first column of Q; in the second step, we effectively compute the remaining columns. Certainly we compute

[1] An unreduced Hessenberg matrix has no zeros on the first subdiagonal.

some transformation with the right leading column; and the implicit Q theorem tells us that any such transformation is basically the one we would have computed with an ordinary QR step.

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.

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

  % 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 `francis_poly` 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 $B = W A^{(k-1)} W$, 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.

```
% [H] = hessqr_francis(H)
%
% Compute a (double) implicit Hessenberg QR step with Francis shift.
% Compare to hessqr_basic.
%
function [H] = hessqr_francis(H)
  % Implicit QR step using a Francis double shift
  % (there should really be some re-scalings for floating point)

```

```
10    % Compute double-shift poly and initial column of H^2 + b*H + c*I
11    [b,c]  = francis_poly(H);
12    C1     = H(1:3,1:2)*H(1:2,1);
13    C1(1:2) = C1(1:2) + b*H(1:2,1);
14    C1(1)  = C1(1)  + c;
15
16    % Apply a similarity associated with the first step of QR on C
17    v       = house(C1);
18    H(1:3,:) = H(1:3,:)-2*v*(v'*H(1:3,:));
19    H(:,1:3) = H(:,1:3)-(H(:,1:3)*(2*v))*v';
20
21    % Do "bulge chasing" to return to Hessenberg form
22    n = length(H);
23    for j = 1:n-2
24      k = min(j+3,n);
25
26      % -- Find W = I-2vv' to put zeros below H(j+1,j), H := WHW'
27      v        = house(H(j+1:k,j));
28      H(j+1:k,:) = H(j+1:k,:)-2*v*(v'*H(j+1:k,:));
29      H(:,j+1:k) = H(:,j+1:k)-(H(:,j+1:k)*(2*v))*v';
30      H(k,j)    = 0;
31
32    end
33
34  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).

4 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.

```
% [H] = hessqr(H)
%
% Toy implementation of Hessenberg QR iteration with Francis double
% shift strategy and deflation.
%
function [H] = hessqr(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) = hessqr_francis(H(1:n,1:n));
      k = k+1;
    end
  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.

5 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.

6 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.