Lecture 11

QR iterations for eigenvalues

See 5.4-5.7 of the text

History

- Two-stage approach doesn't work:
  1. compute the coefficients of the characteristic polynomial which can be done in $O(n^3)$ or $O(n^4)$
  2. compute the zeros of the characteristic polynomial.
- Disaster - the zeros of a polynomial are sensitive to tiny changes in the coefficients.
- The replacement of those n^2 entries by the n coefficients of the characteristic polynomial is too great a condensation of the data.

Basic QR iterations

- Given $A \in \mathbb{R}_{n\times n}$

$$\begin{aligned} A^{(0)} &= A \\ \text{for } k &= 1, 2, \dots \\ & \quad Q^{(k)}R^{(k)} = A^{(k-1)} \\ & \quad A^{(k)} = R^{(k)}Q^{(k)} \end{aligned}$$

- Orthogonal similarity

$$A^{(k)} = R^{(k)}Q^{(k)} = (Q^{(k)})^T A^{(k-1)} Q^{(k)}, \quad k = 0, 1, \dots$$

  Thus

$$A^{(k)} = (Q^{(0)}Q^{(1)} \cdots Q^{(k)})^T A (Q^{(0)}Q^{(1)} \cdots Q^{(k)})$$

- Eigenvalues are preserved in the whole process.

- Some variant: $A^{(0)} = (Q^{(0)})^T A Q^{(0)}$ where $Q^{(0)}$ is some orthogonal matrix.

- Convergence theorem for basic QR iterations

  Theorem 1. Let $A \in GL_n(\mathbb{R})$ such that the moduli of the eigenvalues $\lambda_1, \ldots, \lambda_n$ of A are distinct, that is,

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n| \ (> 0). \tag{1}$$

  Let $A = Y^{-1}DY$ where

$$D := \mathrm{diag}\,(\lambda_1, \cdots, \lambda_n).$$

  Suppose $Y = LU$, where L is unit lower triangular and U is upper triangular.

  1. The strictly lower triangular part of $A^{(k)}$ converges to zero in $O(t^k)$.
  2. The diagonal part of $A^{(k)}$ converges to D in $O(t^k)$,

  where

$$t := \max\left\{\left|\frac{\lambda_2}{\lambda_1}\right|, \cdots, \left|\frac{\lambda_n}{\lambda_{n-1}}\right|\right\} < 1,$$

- The upper triangular entries may fail to converge, in contrast to what the book says on p.206.

- Under the assumption of the above theorem, all eigenvalues are real.

- `basicqr`: Basic QR iteration

```
function [T,Q,R]=basicqr(A,niter)
T=A;
for i=1:niter,
    [Q,R]=mod_grams(T);
    T=R*Q;
end
return
```

- Inefficient because

  1. Each step costs $2n^3$ flops by MGS. The matrix multiplication that follows costs $\approx 2n^3$ flops.

  2. slow convergence in general.

- Read p.203-207

Basic QR iteration starting from Hessenberg

- Generate the real Schur decomposition $T = Q^T AQ$ of A given in Program 30.
- If $A = QR$ is nonsingular Hessenberg, so is RQ.
- Reduce A in Hessenberg form. Costs $O(n^3)$.
- To acheieve max efficiency and stability, use Givens rotations to carry out QR factorization in Program 31
- Each QR step costs $O(n^2)$ flops.
- Program 30 `hessqr`: Hessenberg-QR method

```
function [T,Q,R]=hessqr(A,niter)
n=max(size(A));
[T,Qhess]=houshess(A);
for j=1:niter
     [Q,R,c,s]= qrgivens(T);
    T=R;
    for k=1:n-1,
        T=gacol(T,c(k),s(k),1,k+1,k,k+1);
    end
end
return
```

- Program 31 `qrgivens`: QR factorization with Givens rotations

```
function [Q,R,c,s]= qrgivens(H)
[m,n]=size(H);
for k=1:n-1
     [c(k),s(k)]=givcos(H(k,k),H(k+1,k));
    H=garow(H,c(k),s(k),k,k+1,k,n);
```

```
end
R=H; Q=prodgiv(c,s,n);
return
```

```
function Q=prodgiv(c,s,n)
n1=n-1; n2=n-2;
Q=eye(n); Q(n1,n1)=c(n1); Q(n,n)=c(n1);
Q(n1,n)=s(n1); Q(n,n1)=-s(n1);
for k=n2:-1:1,
   k1=k+1; Q(k,k)=c(k); Q(k1,k)=-s(k);
   q=Q(k1,k1:n); Q(k,k1:n)=s(k)*q;
   Q(k1,k1:n)=c(k)*q;
end
return
```

- cost of `prodgiv` is $n^2 - 2$ flops without explicity forming the Givens matrices.

- givcos

```
function [c,s]=givcos(xi, xk)
if (xk==0), c=1; s=0; else,
   if abs(xk) > abs(xi)
      t=-xi/xk;
      s=1/sqrt(1+t^2);
      c=s*t;
   else
      t=-xk/xi;
      c=1/sqrt(1+t^2);
      s=c*t;
   end
end
return
```

- garow

```
function [M]=garow(M,c,s,i,k,j1,j2)
for j=j1:j2
    t1=M(i,j);
    t2=M(k,j);
    M(i,j)=c*t1-s*t2;
    M(k,j)=s*t1+c*t2;
end
return
```

- To summarize: Householder Hessenberg turn A into a Hessenberg matrix. Then apply QR iterations.

- MATLAB command `[V,D]=eig(A)` reduces the matrix to Hessenberg form and then performs approximately $2n$ implicit double QR iterations to obtain the eigenvalues. It then computes a complete set to eigenvectors.

- Problem: How about real matrices which will likely to have complex conjugate pair of eigenvalues and thus the assumption in the theorem is not satisfied.

- Example 5.9

$$A = \begin{bmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{bmatrix}$$

  whose eigenvalues are 65, ± 21.28 and ± 13.13.

Stability and Accuracy

- The QR iteration is backward stable

$$\hat{T} = Q^T(A + \delta A)Q, \quad \|\delta A\|_2 \approx u\|A\|_2$$

  where $\hat{T}$ is the computed matrices.

- The combination with Hessenberg reduction is also backward stable.

- MATLAB's command `eig(A)`

- MATLAB's command `roots` uses `eig(A)` to find the zeros of a polynomial.

- Read p.214

Single shift (Rayleigh quotient shift)–accelerate the convergence of QR iterations

- Eigenvalues of $A - \mu I$ are $\lambda_1 - \mu, \ldots, \lambda_n - \mu$.
- Introduce shift to accelerate the convergence:
$$\begin{aligned} A^{(k-1)} - \mu^{(k)} I &= Q^{(k)} R^{(k)} \\ A^{(k)} &= R^{(k)} Q^{(k)} + \mu^{(k)} I \end{aligned}$$
- $$A^{(k)} = (Q^{(k)})^T A^{(k-1)} Q^{(k)} = (Q^{(1)} \cdots Q^{(k)})^T A (Q^{(1)} \cdots Q^{(k)})$$
- Rayleigh quotient shift
$$\mu^{(k)} = \frac{(q_n^{(k)})^T A q_n^{(k)}}{(q_n^{(k)})^T q_n^{(k)}} = (q_n^{(k)})^T A q_n^{(k)}$$
where $q_n^{(k)}$ is the last column of $Q^{(1)} \cdots Q^{(k)}$.
- In fact
$$\mu(k) = a_{nn}^{(k)}$$
the (n, n) entry of $A^{(k)}$.
- The convergence of $a_{n,n-1}^{(k)} \to 0$ is quadratic in the sense
$$|a_{n,n-1}^{(k+1)}| / \|A\|_2 = O(\eta_k^2)$$
where $|a_{n,n-1}^{(k)}| / \|A\|_2 = \eta_k < 1$ for some k.
- In practice $a_{n,n-1}^{(k)}$ is set to zero if
$$|a_{n,n-1}^{(k)}| \le \epsilon(|a_{n-1,n-1}^{(k)}| + |a_{n,n}^{(k)}|), \qquad k \ge 0$$
for a prescribed ϵ of the order of u.

Implementation

- Program 36 `qrshift`: QR iteration with single shift

  toll = tolerance ϵ, itmax = max admissible number of iterations.

```
function [T,iter]=qrshift(A,toll,itmax)
n=max(size(A));
iter=0;
[T,Q]=houshess(A);
for k=n:-1:2
  I=eye(k);
  while abs(T(k,k-1)) > toll*(abs(T(k,k))+abs(T(k-1,k-1)))
     iter=iter+1;
     if (iter > itmax),
        return
     end
     mu=T(k,k);
     [Q,R,c,s]=qrgivens(T(1:k,1:k)-mu*I);
     T(1:k,1:k)=R*Q+mu*I;
  end
  T(k,k-1)=0;
end
return
```

- Read p.218-221

- The Rayleigh quotient shifting strategy doesn't always work:

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

- Shifts complicate the convergence analysis.

- No one has been able to prove that the QR iterations with some specific evidently successful shifting strategy always converges.