Math 405: Numerical Methods for Differential Equations 2015 W1
Topics 11: Matrix Eigenvalues and the Symmetric QR Algorithm

References: Trefethen & Bau textbook.

Eigenvalue problem: given a matrix A, find (a/several/all) *eigenvalue(s)* λ and and corresponding *eigenvector(s)* v such that:

$$Av = \lambda v.$$

Iterative Methods: methods such as LU or QR factorizations are *direct*: they compute a certain number of operations and then finish with "the answer". But eigenvalue calculations in general cannot be direct because they are equivalent to finding roots of the characteristic polynomials: for degree greater than 5, there does not exist a finite sequence of arithmetic operations for a solution. They instead must be iterative:

- construct a sequence;
- truncate that sequence "after convergence";
- typically concerned with fast convergence rate (rather than operation count).

Notations: for $x \in \mathbb{R}^n$, we take norm $\|x\| = \sqrt{x^{\mathrm{T}}x}$ to be Euclidean length of x. In iterative methods, x_k usually means the vector x at the kth iteration (rather than kth entry of vector x). Some sources use x^k or $x^{(k)}$ instead.

Power Iteration: a simple method for calculating a single (largest) eigenvalue of a square matrix A (and its associated eigenvector). For arbitrary $y \in \mathbb{R}^n$, set $x_0 = y/\|y\|$ to calculate an initial vector, and then for $k = 0, 1, \ldots$

Compute $y_k = Ax_k$
and set $x_{k+1} = y_k/\|y_k\|$.

This is the **Power Method** or **Iteration**, and computes unit vectors in the direction of $x_0, Ax_0, A^2x_0, A^3x_0, \ldots, A^kx_0$.

Suppose that A is diagonalizable so that there is a basis of eigenvectors of A:

$$\{v_1, v_2, \ldots, v_n\}$$

with $Av_i = \lambda_i v_i$ and $\|v_i\| = 1$, $i = 1, 2, \ldots, n$, and assume that

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|.$$

Then we can write

$$x_0 = \sum_{i=1}^{n} \alpha_i v_i$$

for some $\alpha_i \in \mathbb{R}$, $i = 1, 2, \ldots, n$, so

$$A^k x_0 = A^k \sum_{i=1}^{n} \alpha_i v_i = \sum_{i=1}^{n} \alpha_i A^k v_i.$$

However, since $Av_i = \lambda_i v_i \implies A^2 v_i = A(Av_i) = \lambda_i A v_i = \lambda_i^2 v_i$, inductively $A^k v_i = \lambda_i^k v_i$. So

$$A^k x_0 = \sum_{i=1}^{n} \alpha_i \lambda_i^k v_i = \lambda_1^k \left[\alpha_1 v_1 + \sum_{i=2}^{n} \alpha_i \left(\frac{\lambda_i}{\lambda_1} \right)^k v_i \right].$$

Since $(\lambda_i/\lambda_1)^k \to 0$ as $k \to \infty$, $A^k x_0$ tends to look like $\lambda_1^k \alpha_1 v_1$ as k gets large. The result is that by normalizing to be a unit vector

$$\frac{A^k x_0}{\|A^k x_0\|} \to \pm v_1 \quad \text{and} \quad \frac{\|A^k x_0\|}{\|A^{k-1} x_0\|} \approx \left| \frac{\lambda_1^k \alpha_1}{\lambda_1^{k-1} \alpha_1} \right| = |\lambda_1|$$

as $k \to \infty$, and the sign of λ_1 is identified by looking at, e.g., $(A^k x_0)_1/(A^{k-1} x_0)_1$.

Essentially the same argument works when we normalize at each step: the Power Iteration may be seen to compute $y_k = \beta_k A^k x_0$ for some β_k. Then, from the above,

$$x_{k+1} = \frac{y_k}{\|y_k\|} = \frac{\beta_k}{|\beta_k|} \cdot \frac{A^k x_0}{\|A^k x_0\|} \to \pm v_1.$$

Similarly, $y_{k-1} = \beta_{k-1} A^{k-1} x_0$ for some β_{k-1}. Thus

$$x_k = \frac{\beta_{k-1}}{|\beta_{k-1}|} \cdot \frac{A^{k-1} x_0}{\|A^{k-1} x_0\|} \quad \text{and hence} \quad y_k = A x_k = \frac{\beta_{k-1}}{|\beta_{k-1}|} \cdot \frac{A^k x_0}{\|A^{k-1} x_0\|}.$$

Therefore, as above,

$$\|y_k\| = \frac{\|A^k x_0\|}{\|A^{k-1} x_0\|} \approx |\lambda_1|,$$

and the sign of λ_1 may be identified by looking at, e.g., $(x_{k+1})_1/(x_k)_1$.

Hence the largest eigenvalue (and its eigenvector) can be found.

Note: it is possible for a chosen vector x_0 that $\alpha_1 = 0$, but rounding errors in the computation generally introduce a small component in v_1, so that in practice this is not a concern!

This simplified method for eigenvalue computation is the basis for effective methods (c.f., Rayleigh Quotient and the Arnoldi Algorithm as used in MATLAB's sparse `eigs` command).

For general dense matrices, we will look at a popular method known as the **QR Algorithm**. There are also Divide and Conquer algorithms (since late 1990's).

QR Algorithm We consider only the case where A is symmetric.

Recall: a symmetric matrix A is similar to B if there is a nonsingular matrix P for which $A = P^{-1}BP$. Similar matrices have the same eigenvalues, since if $A = P^{-1}BP$,

$$0 = \det(A - \lambda I) = \det(P^{-1}(B - \lambda I)P) = \det(P^{-1})\det(P)\det(B - \lambda I),$$

so $\det(A - \lambda I) = 0$ if, and only if, $\det(B - \lambda I) = 0$.

The basic **QR algorithm** is:

```
Set A_1 = A.
for k = 1, 2, ...
  form the QR factorization A_k = Q_k R_k
  and set A_{k+1} = R_k Q_k
end
```

Proposition. The symmetric matrices $A_1, A_2, \ldots, A_k, \ldots$ are all similar and thus have the same eigenvalues.

Proof. Since

$$A_{k+1} = R_k Q_k = (Q_k^{\mathrm{T}} Q_k) R_k Q_k = Q_k^{\mathrm{T}}(Q_k R_k) Q_k = Q_k^{\mathrm{T}} A_k Q_k = Q_k^{-1} A_k Q_k,$$

A_{k+1} is symmetric if A_k is, and is similar to A_k. □

At least when A has distinct eigenvalues, this basic QR algorithm can be shown to work (A_k converges to a diagonal matrix as $k \to \infty$, the diagonal entries of which are the eigenvalues). However, a really practical, fast algorithm is based on some refinements.

Reduction to tridiagonal form: the idea is to apply explicit similarity transformations $QAQ^{-1} = QAQ^{\mathrm{T}}$, with Q orthogonal, so that QAQ^{T} is tridiagonal.

Note: direct reduction to triangular form would reveal the eigenvalues, but is not possible. If

$$H(w)A = \begin{bmatrix} \times & \times & \cdots & \times \\ 0 & \times & \cdots & \times \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \times & \cdots & \times \end{bmatrix}$$

then $H(w)AH(w)^{\mathrm{T}}$ is generally full, i.e., all zeros created by pre-multiplication are destroyed by the post-multiplication. However, if

$$A = \begin{bmatrix} \gamma & u^{\mathrm{T}} \\ u & C \end{bmatrix}$$

(as $A = A^{\mathrm{T}}$) and

$$w = \begin{bmatrix} 0 \\ \hat{w} \end{bmatrix} \quad \text{where} \quad H(\hat{w})u = \begin{bmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

it follows that

$$H(w)A = \begin{bmatrix} \gamma & & u^{\mathrm{T}} & \\ \alpha & \times & \vdots & \times \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \times & \vdots & \times \end{bmatrix},$$

i.e., the u^{T} part of the first row of A is unchanged. However, then

$$H(w)AH(w)^{-1} = H(w)AH(w)^{\mathrm{T}} = H(w)AH(w) = \left[\begin{array}{c|c} \gamma & \alpha \;\; 0 \;\; \cdots \;\; 0 \\ \hline \alpha & \\ 0 & \\ \vdots & B \\ 0 & \end{array}\right],$$

where $B = H(\hat{w})CH^{\mathrm{T}}(\hat{w})$, as $u^{\mathrm{T}}H(\hat{w})^{\mathrm{T}} = (\alpha, \quad 0, \quad \cdots, \quad 0)$; note that $H(w)AH(w)^{\mathrm{T}}$ is symmetric as A is.

Now we inductively apply this to the smaller matrix B, as described for the QR factorization but using post- as well as pre-multiplications. The result of $n-2$ such Householder similarity transformations is the matrix

$$H(w_{n-2}) \cdots H(w_2)H(w)AH(w)H(w_2) \cdots H(w_{n-2}),$$

which is tridiagonal.

The QR factorization of a tridiagonal matrix can now easily be achieved with $n-1$ Givens rotations: if A is tridiagonal

$$\underbrace{J(n-1,n) \cdots J(2,3)J(1,2)}_{Q^{\mathrm{T}}}A = R, \quad \text{upper triangular.}$$

Precisely, R has a diagonal and 2 super-diagonals,

$$R = \begin{bmatrix} \times & \times & \times & 0 & 0 & 0 & \cdots & 0 \\ 0 & \times & \times & \times & 0 & 0 & \cdots & 0 \\ 0 & 0 & \times & \times & \times & 0 & \cdots & 0 \\ \vdots & \vdots & \diagdown & \diagdown & \diagdown & \diagdown & \diagdown & \vdots \\ 0 & 0 & 0 & 0 & \times & \times & \times & 0 \\ 0 & 0 & 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix}$$

(exercise: check!). In the QR algorithm, the next matrix in the sequence is RQ.

Lemma. In the QR algorithm applied to a symmetric tridiagonal matrix, the symmetry and tridiagonal form are preserved when Givens rotations are used.

Proof. We have already shown that if $A_k = QR$ is symmetric, then so is $A_{k+1} = RQ$. If $A_k = QR = J(1,2)^{\mathrm{T}}J(2,3)^{\mathrm{T}} \cdots J(n-1,n)^{\mathrm{T}}R$ is tridiagonal, then $A_{k+1} = RQ = RJ(1,2)^{\mathrm{T}}J(2,3)^{\mathrm{T}} \cdots J(n-1,n)^{\mathrm{T}}$. Recall that post-multiplication of a matrix by $J(i,i+1)^{\mathrm{T}}$

replaces columns i and $i+1$ by linear combinations of the pair of columns, while leaving columns $j = 1, 2, \ldots, i-1, i+2, \ldots, n$ alone. Thus, since R is upper triangular, the only subdiagonal entry in $RJ(1,2)^{\mathrm{T}}$ is in position $(2,1)$. Similarly, the only subdiagonal entries in $RJ(1,2)^{\mathrm{T}}J(2,3)^{\mathrm{T}} = (RJ(1,2)^{\mathrm{T}})J(2,3)^{\mathrm{T}}$ are in positions $(2,1)$ and $(3,2)$. Inductively, the only subdiagonal entries in

$$\begin{aligned} &RJ(1,2)^{\mathrm{T}}J(2,3)^{\mathrm{T}} \cdots J(i-2,i-1)^{\mathrm{T}}J(i-1,i)^{\mathrm{T}} \\ &\quad = (RJ(1,2)^{\mathrm{T}}J(2,3)^{\mathrm{T}} \cdots J(i-2,i-1)^{\mathrm{T}})J(i-1,i)^{\mathrm{T}} \end{aligned}$$

are in positions $(j, j-1)$, $j = 2, \ldots i$. So, the lower triangular part of A_{k+1} only has nonzeros on its first subdiagonal. However, then since A_{k+1} is symmetric, it must be tridiagonal. □

Using shifts. One further and final step in making an efficient algorithm is the use of **shifts**:

```
for k = 1, 2, ...
  form the QR factorization of A_k − μ_k I = Q_k R_k
  and set A_{k+1} = R_k Q_k + μ_k I
end
```

For any chosen sequence of values of $\mu_k \in \mathbb{R}$, $\{A_k\}_{k=1}^{\infty}$ are symmetric and tridiagonal if A_1 has these properties, and similar to A_1.

The simplest shift to use is $a_{n,n}$, which leads rapidly in almost all cases to

$$A_k = \left[\begin{array}{c|c} T_k & 0 \\ \hline 0^{\mathrm{T}} & \lambda \end{array}\right],$$

where T_k is $n-1$ by $n-1$ and tridiagonal, and λ is an eigenvalue of A_1. Inductively, once this form has been found, the QR algorithm with shift $a_{n-1,n-1}$ can be concentrated only on the $n-1$ by $n-1$ leading submatrix T_k. This process is called **deflation**.

The overall algorithm for calculating the eigenvalues of an n by n symmetric matrix:

```
reduce A to tridiagonal form by orthogonal
(Householder) similarity transformations.
for m = n, n−1, ... 2
  while a_{m−1,m} > tol
    [Q, R] = qr(A − a_{m,m} * I)
    A = R * Q + a_{m,m} * I
  end while
  record eigenvalue λ_m = a_{m,m}
  A ← leading m−1 by m−1 submatrix of A
end
record eigenvalue λ_1 = a_{1,1}
```