Lecture 7: Finding eigenvalues

Sourendu Gupta

TIFR Graduate School

Computational Physics 1
March 3, 2010

Outline

Quick recap

- The eigenvalues of a square matrix, A, are the solutions of its **characteristic polynomial** $p(z) = |zI - A|$. The set of eigenvalues $\sigma(A)$ is called the **spectrum** of A. Eigenvalues can also be defined as the singular points of the **resolvent** $\rho(z) = |(zI - A)^{-1}|$. If λ is an eigenvalue and $\mathbf{v}$ is a vector such that $A\mathbf{v} = \lambda\mathbf{v}$, then $\mathbf{v}$ is called a (right) eigenvector. If $\mathbf{v}^\dagger A = \lambda\mathbf{v}^\dagger$ then $\mathbf{v}$ is called a left eigenvector.
- The **Hermitean conjugate** of a matrix, A, denoted $A^\dagger$, is defined as follows $\left(A^\dagger\right)_{ij} = A^*_{ji}$. If A is real, then the Hermitean conjugate is the **transpose**. The eigenvalues of $A^\dagger$ are complex conjugates of the eigenvalues of A. The left eigenvectors of A are the right eigenvectors of $A^\dagger$ and vice versa.
- A **normal matrix** commutes with its Hermitean conjugate. The right eigenvectors of every normal matrix are Hermitean conjugates of the left eigenvectors [Wilkinson].

Some warm-up problems

Problem 1: Hermitean matrices ($A^\dagger = A$), anti-Hermitean matrices ($A^\dagger = -A$), unitary matrices ($A^\dagger = A^{-1}$) and anti-unitary matrices ($A^\dagger = -A^{-1}$) are normal. Exhibit a simple matrix such that $A^\dagger = A^6$ but any smaller power of A is not equal to $A^\dagger$. Is any matrix of the form $A^\dagger = f(A)$ normal? Are there any restrictions on the function f? Does this exhaust the class of normal matrices?

Problem 2: Write fast routines for copying one vector into another, adding or subtracting two vectors and taking the 2-norm of any vector. Count the complexity of each algorithm and check that the speed of your program is in agreement with your count. These will be building blocks of your later exercises, so get them to work in nanoseconds (since all modern CPUs work on multi GHz clocks, this is practical).

Problem 3: Speed up your implementation of the Gauss elimination routine to work in nanoseconds. This routine will be used as a basic building block later, so tuning its performance is important.

Bounding eigenvalues

Gershgorin's theorem: If A is a square matrix, then $\sigma(A)$ lies in the union of disks in the complex plane

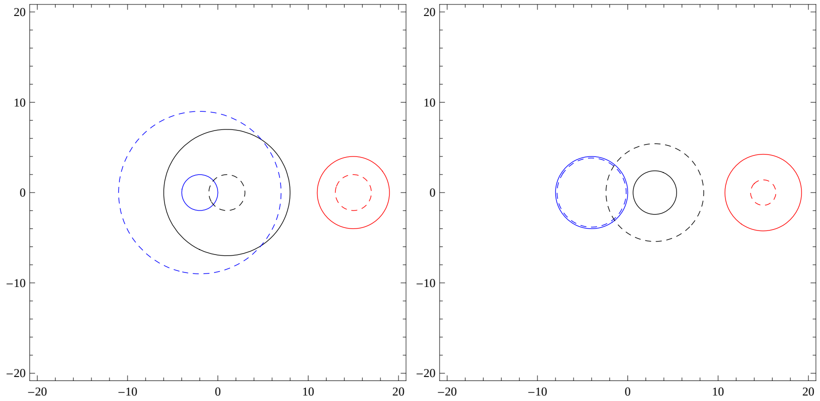
$$\bigcup_i \left\{ z \;\middle|\; |z - A_{ii}| \le \sum_{j\ne i} |A_{ij}| \right\}.$$

Example: These matrices have the same characteristic equation:

$$A = \begin{pmatrix} 15 & -3 & 1 \\ 1 & -2 & 1 \\ 1 & 6 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 15 & -2\sqrt{2} & -\sqrt{2} \\ 0 & -4 & -4 \\ \sqrt{2} & 1 & 3 \end{pmatrix}.$$

The eigenvalues of A lie in the union of three disks $|z - 15| \le 4$, $|z + 2| \le 2$ and $|z - 1| \le 7$. The disk centered on $z = 15$ is entirely disjoint from the other two and must contain one eigenvalue. The eigenvalues of B are in the union of the disjoint disks $|z - 15| \le 3\sqrt{2}$, $|z + 4| \le 4$ and $|z - 3| \le 1 + \sqrt{2}$. Although $\sigma(A^T) = \sigma(A)$, the Gershgorin disks of A and A^T may be different.

Gershgorin disks for A and B



The full lines are the disks constructed from rows, and the dashed lines from columns (disks for A on the left, B on the right).

A problem

Problem 4: Construct orthogonal transformations of the matrix A of the previous problem using the orthogonal matrices

$$U(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix},$$

in the form $M(\phi) = U^T(\phi) M U(\phi)$. Are there any values of ϕ for which the Gershgorin disks are disjoint?

Problem 5: Use `Mathematica` to draw the Gershgorin disks for any given ϕ. Using `Mathematica` functions construct an animation of the figure as ϕ changes from 0 to 2π.

Problem 6: Convert your `Mathematica` code for animation of Gershgorin disks into a `Mathematica` program which works on any one-parameter family of matrices of any dimension.

Condition number of an eigenvalue

Let A be a square matrix with a non-degenerate eigenvalue λ. Let $\mathbf{x}$ be the normalized right eigenvector ($A\mathbf{x} = \lambda\mathbf{x}$) and $\mathbf{y}^\dagger$ the normalized left eigenvector ($\mathbf{y}^\dagger A = \lambda \mathbf{y}^\dagger$). The **condition number** of λ, $K(\lambda) = \mathbf{y}^\dagger\mathbf{x}$, determines how much it changes when A is perturbed slightly [Golub]. Let A be perturbed to $A + \epsilon B$ where $||B||_2 = 1$. Let the perturbed eigenvalue and eigenvectors be $\lambda(\epsilon)$ and $\mathbf{x}(\epsilon)$, where $(A + \epsilon B)\mathbf{x}(\epsilon) = \lambda(\epsilon)\mathbf{x}(\epsilon)$. Taking the derivative with respect to ϵ as $\epsilon \to 0$, we obtain

$$(A - \lambda)\mathbf{x}' + B\mathbf{x} = \lambda'\mathbf{x}.$$

Taking a dot product with $\mathbf{y}^\dagger$ kills the first term and gives

$$\lambda' = \frac{\mathbf{y}^\dagger B\mathbf{x}}{\mathbf{y}^\dagger x} \le \frac{1}{K(\lambda)}.$$

For a normal matrix $K(\lambda) = 1$, so one can find eigenvalues without great numerical sensitivity. For non-normal matrices $K(\lambda)$ can be small so the eigenvalues may be sensitive to numerical errors.

Outline

Maximum eigenvalue of a matrix

If A is a non-deficient square matrix whose maximum eigenvalue is unique, then it can be obtained by the algorithm on the right. Its output is the eigenvalue λ and the eigenvector $\mathbf{v}$. The successive estimators of λ should converge exponentially to the correct answer. If the matrix dimension is N then step 4 takes $\mathcal{O}(N^2)$ steps and every other step is $\mathcal{O}(N)$.

```
1 v is a random unit vector
2 $\mathbf{r} = \mathbf{v}$
3 until $||\mathbf{r}||_2 < \epsilon$ do
4         $\mathbf{w} = A\mathbf{v}$
5         $\mathbf{w} = \mathbf{w}/||w||_2$
6         $\mathbf{r} = \mathbf{w} - \mathbf{v}$
7         $\lambda = \mathbf{v}^\dagger \mathbf{w}$
8         $\mathbf{v} = \mathbf{w}$
9 end do
```

Problem 7: By writing $\mathbf{v}$ as a linear combination of eigenvectors, prove that the algorithm works under the conditions specified and find the rate of convergence. Complete the computation of the complexity. Write a program and check that its performance tallies with your count of the complexity.

Inverse iteration: improving approximate eigenvalues

For a square matrix A if an isolated eigenvalue is known to have value approximately z, then inverse iteration refines the estimate of that eigenvalue. The process should converge exponentially. The cost of the process is dominated by the inversion of the matrix, *i.e.*, the solution of the linear system $(A - z)\mathbf{w} = \mathbf{v}$. Efficient programming of the function for Gauss elimination will be essential to this routine, since that function is called repeatedly inside this.

```
1  $\mathbf{v}$ is a random unit vector
2  $\mathbf{r} = \mathbf{v}$
3  until $||\mathbf{r}||_2 < \epsilon$ do
4          $\mathbf{w} = (A - z)^{-1}\mathbf{v}$
5          $\mathbf{w} = \mathbf{w}/||w||_2$
6          $\mathbf{r} = \mathbf{w} - \mathbf{v}$
7          $\mathbf{v} = \mathbf{w}$
8  end do
9  $\lambda = \mathbf{w}^\dagger A\mathbf{w}$
```

Outline

Eigenvalues of symmetric matrices

Real symmetric matrices, A, have real eigenvalues. They can be diagonalized by orthogonal transformations, *i.e.*, it is possible to find an orthogonal matrix O such that $O^{-1}AO$ is diagonal. The columns of O are eigenvectors of A.

Example: Consider the real symmetric matrix

$$A = \begin{pmatrix} 15 & 1 & 1 \\ 1 & -2 & 6 \\ 1 & 6 & 1 \end{pmatrix}.$$

The Gershgorin disks are $|z-15| \leq 2$, $|z+2| \leq 7$ and $|z-1| \leq 7$. The last two disks overlap. Since all the eigenvalues are real, it should be possible to transform A so that the disks are non-overlapping.

Jacobi's method consists of building successive orthogonal transformations which shrink the Gershorgin disks by monotonically reducing the sum of the absolute values of the off-diagonal elements:

$$J(A) = \sum |A_{ij}|^2.$$

The 2×2 case

Consider the transformation

$$\begin{pmatrix} b_{11} & 0 \\ 0 & b_{22} \end{pmatrix} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

where $c = \cos\phi$ and $s = \sin\phi$. A straightforward computation shows that this implies $\tan 2\phi = 2a_{12}/(a_{11} - a22) = 1/\tau$. One derives from this that

$$\cos\phi = \frac{1}{\sqrt{1+t^2}}, \quad \sin\phi = \frac{t}{\sqrt{1+t^2}}, \quad \text{where} \quad t = \tau \pm \sqrt{1+\tau^2}.$$

For stability one uses the smaller of the two solutions. These are called Jacobi or Givens rotations.
One can apply this to any 2×2 block of a larger matrix. Usually one chooses the block which includes the largest off-diagonal component and reduces it to zero.

The Jacobi rotation

This is the Jacobi rotation element obtained for $1 \leq i < j \leq N$ where A is a real symmetric $N \times N$ matrix.

```
1 If $A_{ij} = 0$ then $c = 1$ and $s = 0$; exit
2 $\tau = (A_{ii} - A_{jj})/(2A_{ij})$
3 **If** $\tau \geq 0$ **then**
4         $t = 1/(\tau + \sqrt{1 + \tau^2})$
5 **else**
6         $t = -1/(-\tau + \sqrt{1 + \tau^2})$
7 **end**
8 $c = 1/\sqrt{1 + t^2}$, $s = tc$.
```

This algorithm takes about 10 flops and two square roots.

Applying the rotation

Problem 8: For a generic $N \times N$ matrix the above algorithm implicitly returns the orthogonal matrix

$$G(p,q,\theta) = \begin{pmatrix} 1 & \cdots & & & & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & & & & \cdots & 1 \end{pmatrix},$$

i.e., the identity matrix except in a 2×2 block. Similarity transformation by this matrix changes only two rows and two columns of the matrix A. This has complexity $\mathcal{O}(N)$. Compute the complexity and program this carefully to get the fastest possible implementation (the code should take less than $10N$ nanoseconds to execute).

The Jacobi algorithm

The complete algorithm works like this:

```
1 do as many sweeps as necessary
2     for each element above the diagonal
3         find the Jacobi rotation
4         apply the rotation
5     end for
6 end do
```

The inner loop is traversed $N(N-1)/2$ times and the effort at each step is $\mathcal{O}(N)$. Hence the CPU time required per sweep (*i.e.*, per traversal of the inner loop) is $\mathcal{O}(N^3)$. The number of sweeps usually does not change with N. There are various opinions about the ordering of the sweep. No matter what the ordering is, it is not going to be less than an $\mathcal{O}(N^2)$ process. It turns out to be most efficient to reduce the matrix to tridiagonal form and then use some other method.

A Householder transformation

Take a vector $\mathbf{w}$. The Householder matrix

$$P(\mathbf{w}) = 1 - 2\frac{\mathbf{w}\mathbf{w}^T}{||\mathbf{w}||^2},$$

where $||\mathbf{w}||$ denotes the 2-norm. The matrix is clearly Hermitean since $P^T = P$. Also, since $P^2 = 1 = P^\dagger P$, it is unitary.

Example: in 2 dimensions, let $\mathbf{x}^T = (1, 0)$ and $\mathbf{y}^T = (1, 1)/\sqrt{2}$. Then

$$P(\mathbf{x}) = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad P(\mathbf{y}) = -\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

For any vector $\mathbf{x}$, $P\mathbf{x} = \mathbf{x} - 2\mathbf{w}(\mathbf{w}^T\mathbf{x})/||\mathbf{w}||^2$. As a result, $\mathbf{w}^T P\mathbf{x} = -\mathbf{w}^T\mathbf{x}$. The Householder transformation flips the component of any vector in the direction of the chosen vector $\mathbf{w}$.

Example: For the above vectors and matrices

$$P(\mathbf{x})\mathbf{x} = -\mathbf{x},\ P(\mathbf{x})\mathbf{y} = \frac{1}{\sqrt{2}}\begin{pmatrix} -1 \\ 1 \end{pmatrix},\ P(\mathbf{y})\mathbf{x} = -\begin{pmatrix} 0 \\ -1 \end{pmatrix},\ P(\mathbf{y})\mathbf{y} = -\mathbf{y}.$$

Another Householder transformation

Define the unit vectors $\hat{e}_j$ to be the j-th column of the identity matrix. For a given vector $\mathbf{v}$, construct $\mathbf{w} = \mathbf{v} \pm ||\mathbf{v}||\hat{e}_j$. Now $\mathbf{w}^T\mathbf{w} = 2(||\mathbf{v}||^2 \pm ||\mathbf{v}||v_j)$ where $v_j = \mathbf{v}^T\hat{e}_j$. Then,

$$P(\mathbf{w})\mathbf{v} = \mathbf{v} - \mathbf{w}\frac{2(||\mathbf{v}||^2 \pm ||\mathbf{v}||v_j)}{2(||\mathbf{v}||^2 \pm ||\mathbf{v}||v_j)} = \mathbf{v} - \mathbf{w} = \mp||\mathbf{v}||\hat{e}_j.$$

This Householder transformation projects the chosen vector into the direction j without changing its norm.
Example: in 2 dimensions, let $\mathbf{x}^T = (1,0)$ and $\mathbf{y}^T = (1,1)/\sqrt{2}$. Set $\mathbf{w} = \mathbf{x} - \hat{e}_2$ and $\mathbf{v} = y - \hat{e}_2$. Then

$$P(\mathbf{w}) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad P(\mathbf{v}) = \frac{1}{\sqrt{2}}\begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}.$$

Clearly $P(\mathbf{w})\mathbf{x} = \hat{e}_2$ and $P(\mathbf{v})\mathbf{y} = \hat{e}_2$.
Note that the Householder matrices are not rotation matrices. They generate reflections in certain directions.

Transforming a matrix

At the k-th step of transforming a $N \times N$ matrix, A, take the Householder transformation to be the block diagonal form $P = 1_k \oplus P_{N-k}$. The notation means that the identity block is $k \times k$ and the Householder block is $(N-k) \times (N-k)$. The vector from which we build the Householder block is the last $N-k$ components of the k-th column of the matrix being reduced. A succession of such transformations reduces A to **tridiagonal** form.

If $\mathbf{w}$ is the vector defining the transformation, then let $\mathbf{r} = 2A\mathbf{w}/||\mathbf{w}||^2$, where A is to be transformed. The transformed matrix is

$$\begin{aligned} PAP &= \left(1 - 2\frac{\mathbf{w}\mathbf{w}^T}{||\mathbf{w}||^2}\right)(A - \mathbf{r}\mathbf{w}^T) = A - (\mathbf{r}\mathbf{w}^T + \mathbf{w}\mathbf{r}^T) + 2K\mathbf{w}\mathbf{w}^T \\ &= A - (\mathbf{y}\mathbf{w}^T + \mathbf{w}\mathbf{y}^T). \end{aligned}$$

We have used the notation $K = \mathbf{w}^T\mathbf{r}/||\mathbf{w}||^2$ and $\mathbf{y} = \mathbf{r} - K\mathbf{w}$. Implementing this formula optimizes the computation of the transform without requiring storage for P [Recipes].

An example

Tridiagonalize $A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{pmatrix}$, using $P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & * & * \\ 0 & * & * \end{pmatrix}$

P is built using the last two elements of the first column of A, *i.e.*, $\mathbf{v}^T = (0, 2, 3)$. Then using the definition $\mathbf{w} = \mathbf{v} - ||\mathbf{v}||\hat{e}_2$, we have

$$\mathbf{w} = \begin{pmatrix} 0 \\ 2-\sqrt{13} \\ 3 \end{pmatrix} \qquad \text{and} \qquad P(\mathbf{w}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2/\sqrt{13} & 3/\sqrt{13} \\ 0 & 3/\sqrt{13} & -2/\sqrt{13} \end{pmatrix}.$$

For such a simple matrix one can evaluate PAP directly. However, one also finds that $\mathbf{y}^T = (1.(2+\sqrt{13})/3, 1)$. Either way,

$$PAP = A - (\mathbf{y}\mathbf{w}^T + \mathbf{w}\mathbf{y}^T) = \begin{pmatrix} 1 & \sqrt{13} & 0 \\ \sqrt{13} & 10 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

Some problems

Problem 9: Write a `Mathematica` program to tridiagonalize 3×3 matrices using the Householder transformation used above. Use your earlier program to draw the Gershgorin circles of the original and transformed matrices.

Problem 10: Compute the complexity of the Householder algorithm. Time the program in `Numerical Recipes` and see whether this agrees with your computation of the complexity. Remember that the coefficients in the scaling of N have to be of the order of a couple of clock cycles (*i.e.*, nanoseconds). If they are not, then trace the reason for that and check whether it can be fixed.

Problem 11: Is it possible to diagonalize the tridiagonal form of the 3×3 matrices efficiently by Jacobi's method? If so, compute the complexity of this algorithm.

Problem 12: Extend Householder's method to general Hermitean matrices.

Tridiaogonal matrices

$$\begin{pmatrix}
* & * & & & & & & & \cdots \\
* & * & * & & & & & & \cdots \\
 & * & * & * & & & & & \cdots \\
 & & * & * & * & & & & \cdots \\
 & & & * & * & * & & & \cdots \\
 & & & & * & * & * & & \cdots \\
 & & & & & * & * & * & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots
\end{pmatrix}$$

Unreduced tridiagonal matrix

Tridiaogonal matrices

$$\begin{pmatrix} * & * & & & & & & & \cdots \\ * & * & * & & & & & & \cdots \\ & * & * & * & & & & & \cdots \\ & & * & * & 0 & & & & \cdots \\ & & & 0 & * & * & & & \cdots \\ & & & & * & * & * & & \cdots \\ & & & & & * & * & * & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots \end{pmatrix}$$

Unreduced tridiagonal matrix

The QR reduction

If $A = QR$ where A is a symmetric tridiagonal $n \times n$ matrix, Q is orthogonal and R is upper triangular, then $A' = RQ = Q^T AQ$. The **QR factorization** has the following properites—

1. If A is tridiagonal then Q has bandwidth 1 and R has bandwidth 2. As a result, A' is also symmetric and tridiagonal.
2. For any real x, if $A - x = QR$ is the QR factorization, then $A' = RQ + x$ is also tridiagonal. This is a **shifted QR factorization**.
3. If A is unreduced, then the first $n - 1$ columns of $A - x$ are independent whatever the value of x. Therefore if $x \in \sigma(A)$ and $A - x = QR$ then $R_{nn} = 0$. As a result, the last column of $A' = RQ + x$ is $x\hat{e}_n$. An exact shift reduces the matrix.
4. The QR factorization of A can be computed by a sequence of Jacobi (Givens) rotations.

The Givens rotation

This is the Givens rotation element which effects the transformation

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}^T \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} x \\ 0 \end{pmatrix}.$$

The following algorithm [Golub] has input (a, b) and output (c, s).

```
1 If $b = 0$ then
2       $c = 1;\ s = 0;$
3 else
4       If $|b| > |a|$ then
5             $t = -a/b;\ s = 1/\sqrt{1+t^2};\ c = st;$
6       else
7             $t = -b/a;\ c = 1/\sqrt{1+t^2};\ s = ct;$
8       end
9 end
```

This algorithm takes about 5 flops and one square root (compare with Jacobi).

The implicit shift algorithm

```
1  Until satisfied do
2      $d = (A_{n-1,n-1} - A_{nn})/2$
3      $\mu = A_{nn} - A^2_{n,n-1}/(d + \mathrm{Sign}(d)\sqrt{d^2 + A^2_{n,n-1}})$
4      $x = t_{11} - \mu;\ y = t_{21};$
5       for $k = 1 : n-1$ do
6          $(c, s) =$ givens$(x, y)$
7          $A = G(k, k+1, \theta)^T A G(k, k+1, \theta)$
8          $x = t_{k+1,k};\ y = t_{k+2,k};$
9       end
10 end
```

Example: in one of the `Mathematica` notebooks [Codes].

Outline

References and further reading

- The Algebraic Eigenvalue Problem, by J. H. Wilkinson, Clarendon Press, 1965. This is *the* book on linear algebra: with detailed theory, very good explanations, and, most of all, beautifully constructed examples.
- Matrix Computations, by G. H. Golub and C. F. van Loan, Johns Hopkins University Press, 1996. This is the recommended course book on algorithms in linear algebra. It is also the "industry standard".
- Numerical Recipes, by W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Cambridge University Press. Treat this as a well written manual for a ready source of building blocks.
- There are Mathematica notebooks associated with this lecture.