Matrices

An m×n matrix is a rectangular array of complex or real numbers arranged in *m* rows and *n* columns:

$$A = [a_{ij}] = \begin{vmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{vmatrix}$$

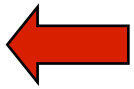
$$(i = 1,2,\dots,m; \quad j = 1,2,\dots,n)$$

Types, Operations, etc.

- Types: square, symmetric, diagonal, triangular (upper U or Lower L), tri-diagonal, banded, transpose, sparse, Hermithean, …
- Basic operations: A+B, A-B, AB (generally AB≠BA).
- Square matrices
  - Determinant: det(A)
  - Inverse matrix A^{-1}: $AA^{-1} = I$ (I is a unit matrix)
  - …

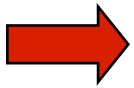
Applications

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
&\cdots\cdots\cdots\cdots \\
a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m,
\end{aligned}
$$



Linear systems of equations

Eigenvalue problem



$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= \lambda x_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= \lambda x_2 \\
&\cdots\cdots\cdots\cdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= \lambda x_n,
\end{aligned}
$$

Linear systems of equations

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m,
\end{aligned}
$$

- $m>n$ over determined system (data processing)
- $m=n$ square case (what we will do)
- $m<n$ under determined system

Linear systems in matrix notation

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{pmatrix}$$

The are four solution possibilities

1. A unique solution (consistent set of solutions)
2. No solution
3. The trivial solution $x_i = 0 \quad (i = 1, 2, \dots n)$
4. An infinite number of solutions

Two cases for right-hand coefficients

- right-hand coefficients $b_i \neq 0$

  Unique solution if the determinant det(A) ≠ 0

- right-hand coefficients $b_i = 0$

  Unique solution if the determinant det(A) = 0

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{pmatrix}$$

Two fundamental groups for solving (1)

Direct elimination methods – systematic procedures for obtaining solutions in a fixed number of operations

- ✓ Gauss elimination
- ✓ Gauss-Jordan elimination
- ✓ the matrix inverse method
- ✓ Doolittle LU factorization
- ✓ …

Direct elimination methods are generally used when one or more of the following conditions holds:

- ✓ the number of equations is small (<100)
- ✓ the most of the coefficients are nonzero
- ✓ the system is not diagonally dominant
- ✓ the system is ill-conditioned

Two fundamental groups for solving (2)

Iterative methods – obtain a solution asymptotically by an iterative procedure

- ✓ Jacobi iteration
- ✓ Gauss-Seidel iteration
- ✓ SOR – successive-over-iteration
- ✓ …

Iterative methods are used

- ✓ when the number of the equations is large and the most of the coefficients are zero (sparse matrix)

Iterative methods generally diverse unless thye system is diagonally dominant

Useful row operations

Useful properties for solving systems of liner equations

- ✓ any equation may be multiplied by a constant (scaling)
- ✓ the order of the equations may be interchanged (pivoting)
- ✓ any equation can be replaced by a weighted linear combination of that equation with any other equation (elimination)

Analytic solutions for n=2

$a_{11}x_1 + a_{12}x_2 = b_1$

$a_{21}x_1 + a_{22}x_2 = b_2$

expressing the first unknown x_1 from the first equation

$x_1 = (b_1 - a_{12}x_2)/a_{11}$

and substituting to the second equation we have a single equation with one unknown x_2.

Gaussian elimination

- Since there is no such an operator as elimination neither in C++ nor Fortran we should translate this procedure to an appropriate numerical method for solving systems of linear equations.
- Numerical method = Gaussian elimination

Gaussian elimination for n=3

$$
\begin{aligned}
a_{11}x_1 &+ a_{12}x_2 &+ a_{13}x_3 &= b_1 \\
a_{21}x_1 &+ a_{22}x_2 &+ a_{23}x_3 &= b_2 \\
a_{31}x_1 &+ a_{32}x_2 &+ a_{33}x_3 &= b_3,
\end{aligned}
$$

Let subtract the first equation multiplied by the coefficient a_{21}/a_{11} from the second one, and multiplied by the coefficient a_{31}/a_{11} from the third equation.

$$
\begin{aligned}
a_{11}x_1 &+ a_{12}x_2 &+ a_{13}x_3 &= b_1 \\
0 &+ a'_{22}x_2 &+ a'_{23}x_3 &= b'_2 \\
0 &+ a'_{32}x_2 &+ a'_{33}x_3 &= b'_3,
\end{aligned}
$$

$$
\begin{aligned}
a'_{ij} &= a_{ij} - a_{1j}a_{i1}/a_{11} \\
b'_i &= b_i - b_1 a_{i1}/a_{11},
\end{aligned}
$$

Step 2:

Repeating the same procedure to the last of two equations gives

$$\begin{aligned} a_{11}x_1 &+ a_{12}x_2 &+ a_{13}x_3 &= b_1 \\ 0 &+ a'_{22}x_2 &+ a'_{23}x_3 &= b'_2 \\ 0 &+ 0 &+ a''_{33}x_3 &= b''_3, \end{aligned}$$

where

$$a''_{ij} = a'_{ij} - a'_{2j}a'_{i2}/a'_{22}$$

$$b''_i = b'_i - b'_2 a'_{i2}/a'_{22}$$

Step 3:

Doing back substitution we will find x_2 and then x_1.

This direct method to find solutions for a system of linear equations by the successive elimination is known as Gaussian elimination.

$$\begin{aligned} a_{11}x_1 &+ a_{12}x_2 &+ a_{13}x_3 &= b_1 \\ 0 &+ a'_{22}x_2 &+ a'_{23}x_3 &= b'_2 \\ 0 &+ 0 &+ a''_{33}x_3 &= b''_3, \end{aligned}$$

Problems!

- ⇨ zero diagonal elements
- ⇨ round-off errors
- ⇨ ill-conditioned systems
- ⇨ computational time

Zero diagonal elements

- The problem may be solved by interchanging the rows of the system, pushing zero elements to off the diagonal. This is the *partial pivoting* procedure.
- Moreover, reordering the system in a way when $a_{11}>a_{22}>a_{33}>...>a_{nn}$ would increase efficiency of the method. This is the issue of the *complete pivoting*.

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m,
\end{aligned}
$$

Round-off errors

- For the each new elimination the Gaussian method utilizes results from the previous eliminations. This procedure accumulates the round-off errors.
- Thus, for large systems you may get wrong numerical solution by doing everything right.
- It is highly recommended to check solutions by direct substitution of x_1, x_2, ..., x_n into the original system of linear equations.

Round-off errors 2.

- How can we reduce the round-off errors?
- Usually, complete pivoting may be very efficient.
- Scaling, multiplication of the i-th equation by a constant c_i, may also help in improving accuracy.

ill-conditioned systems

- a small change in coefficients will produce large changes in the result.
- In particularly, this situation occurs when the determinant for **A** is close to zero.
- The solution may be very unstable, regarding the way you are solving the system

Computer time

- As the number of equations in the system increases, the computation time grows nonlinearly.
- Systems with hundreds, even thousands, equations are common in physics. And you may face a problem of waiting for weeks, if not years, to get an output from your computer.
- Sometimes iterative methods can help to increase speed, but generally they are less accurate.

The most powerful method!

- Using standard libraries!!!
- Specifically, LAPACK library is a very large Linear Algebra Package with hundreds programs.
- However, you have to be careful selecting a program that is right for your specific system of linear equations.

Eigenvalue problem

Structure calculations for quantum systems
(atomic, molecular, nuclear, solid state systems)

Ax = λx

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = \lambda x_1$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = \lambda x_2$$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

$$a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = \lambda x_n ,$$

coefficients λ (eigenvalues) are unknown!

Other form

Regrouping terms gives

$$
\begin{aligned}
(a_{11}-\lambda)x_1 + \quad a_{12}x_2 \quad + \cdots + \quad a_{1n}x_n \quad &= 0 \\
a_{21}x_1 \quad + (a_{22}-\lambda)x_2 + \cdots + \quad a_{2n}x_n \quad &= 0 \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
a_{n1}x_1 \quad + \quad a_{n2}x_2 \quad + \cdots + (a_{nn}-\lambda)x_n &= 0.
\end{aligned}
$$

Looks like a system of linear equations

For the each eigenvalue there is a unique set of solutions called eigenvectors

One more form

Introducing a unit matrix I, which is

$$I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

then $(\mathbf{A} - \lambda\mathbf{I}) \cdot \mathbf{x} = 0.$

Finding λ

Solutions for the system $(\mathbf{A} - \lambda\mathbf{I})\cdot\mathbf{x} = 0.$
exists if and only if the determinant of the matrix is zero

$$\det|\mathbf{A} - \lambda\mathbf{I}| = 0.$$

For a n×n matrix the equation above would give a polynomial in λ of degree n

$$c_n\lambda^n + c_{n-1}\lambda^{n-1} + \cdots + c_1\lambda + c_0 = 0.$$

The coefficients c are determined through the matrix elements a_{ij} by the definition for the matrix determinant. This polynomial equation is known as the characteristic equation of the matrix **A**. Roots of this equation would give the required eigenvalues.

But, this method is NOT practical unless n is very small

Eigenvalues

It has been proved that it is not possible to calculate the roots of an arbitrary nth-degree polynomial exactly in a finite number of steps, for n >5

The eigenvalue problem stands apart from other problems in computational linear algebra.

All methods for finding matrix eigenvalues are iterative.

Most methods for finding all eigenvalues and eigenvectors of a matrix are based on the fact that the transformation

$$A \rightarrow Q^{-1}AQ$$

does not change the eigenvalues of A.

Thus "transformation" methods attempt to find matrices Q such that $Q^{-1}AQ$ has a form that makes eigenvalue extraction trivial (the eigenvalues can be read off the diagonals of triangular and diagonal matrices) or at least easier

The direct power method

The method:

1. Consider $\mathbf{Ax} = \lambda \mathbf{x}$ and assume a trial vector (solution)$\mathbf{x}^{(0)}$ with one of the components equal to 1.
2. Perform the multiplication $\mathbf{y}^{(1)} = \mathbf{A}\mathbf{x}^{(0)}$
3. Use this $\mathbf{y}^{(1)} = \mathbf{A}\mathbf{x}^{(0)}$ to get the next approximation by scaling $\mathbf{y}^{(1)} = \lambda^{(1)}\mathbf{x}^{(1)}$ so that one of the components of $\mathbf{x}^{(1)}$ is 1.
4. Repeat steps (2) and (3) with $\mathbf{x} = \mathbf{x}^{(1)}$ until you see the convergence.

At convergence, the value λ is the largest (in absolute value) eigenvalue of $\mathbf{Ax}$ and the vector $\mathbf{x}$ is the corresponding eigenvector.

The general procedure is $\mathbf{A}\mathbf{x}^{(n)} = \mathbf{y}^{(n+1)} = \lambda^{(n+1)}\mathbf{x}^{(n+1)}$

Example of power method

$$\mathbf{A} = \begin{bmatrix} 8 & -2 & -2 \\ -2 & 4 & -2 \\ -2 & -2 & 13 \end{bmatrix}$$

Assume

$$\mathbf{x}^{(0)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \text{then} \quad \mathbf{y}^{(1)} = \mathbf{A}\mathbf{x}^{(0)} = \begin{bmatrix} 8 & -2 & -2 \\ -2 & 4 & -2 \\ -2 & -2 & 13 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 9 \end{bmatrix}$$

Select the scale factor $\lambda^{(1)} = 9$ so

$$\begin{bmatrix} 4 \\ 0 \\ 9 \end{bmatrix} = 9 \begin{bmatrix} 0.444 \\ 0 \\ 1 \end{bmatrix} = 9\,\mathbf{x}^{(1)}$$

Next step:

$$\begin{bmatrix} 8 & -2 & -2 \\ -2 & 4 & -2 \\ -2 & -2 & 13 \end{bmatrix} \begin{bmatrix} 0.444 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.555 \\ -2.888 \\ 12.111 \end{bmatrix} = 12.111 \begin{bmatrix} 0.128 \\ -0.238 \\ 1 \end{bmatrix} = \lambda^{(2)}\mathbf{x}^{(2)}$$

After 30 iterations

$$\lambda = 13.8705, \qquad \mathbf{x} = \begin{bmatrix} -0.2917 \\ -0.1435 \\ 1.0000 \end{bmatrix}$$

Convergence is slow.

The basis of the power method

Assume that $\mathbf{A}$ is $n \times n$ nonsingular matrix with n linearly independent eigenvectors $\mathbf{x_1}, \mathbf{x_2}..., \mathbf{x_n}$. Assume further that the eigenvalues are arranged such that $|\lambda_1| > |\lambda_2| > |\lambda_3| .... > |\lambda_n|$.

Since vectors $\mathbf{x}_i$ are linearly independent any arbitrary vector $\mathbf{x}$ can be expressed as linear combination

$$\mathbf{x} = c_1\mathbf{x_1} + c_2\mathbf{x_2} + ... + c_n\mathbf{x_n} = \sum_{i=1}^{n} c_i\mathbf{x_i}$$

multiplying both sides by $\mathbf{A}$, $\mathbf{AA} = \mathbf{A^2}$, $\mathbf{A^3}$, ... and using $\mathbf{Ax_i} = \lambda_i\mathbf{x_i}$ one gets

$$\mathbf{Ax} = \sum_{i=1}^{n} c_i\mathbf{Ax_i} = \sum_{i=1}^{n} c_i\lambda_i\mathbf{x_i} = \mathbf{y^{(1)}}$$

$$\mathbf{A^2x} = \mathbf{Ay^{(1)}} = \sum_{i=1}^{n} c_i\mathbf{Ax_i} = \sum_{i=1}^{n} c_i\lambda_i\mathbf{x_i} = \mathbf{y^{(2)}}$$

.

.

$$\mathbf{A^kx} = \mathbf{Ay^{(k-1)}} = \sum_{i=1}^{n} c_i\lambda_i^{k-1}\mathbf{Ax_i} = \sum_{i=1}^{n} c_i\lambda_i^{k}\mathbf{x_i} = \mathbf{y^{(k)}} \qquad (1)$$

Factoring out the largest eigenvalue we get

$$\mathbf{A^k x} = \lambda_1^k \sum_{i=1}^{n} c_i \left(\frac{\lambda_i}{\lambda_1}\right)^k \mathbf{x_i} = \mathbf{y^{(k)}}$$

Since $\frac{\lambda_i}{\lambda_1} < 1$ for $i = 2, 3, \ldots, n$ we get $\left(\frac{\lambda_i}{\lambda_1}\right)^k \to 0$ as $k \to \infty$ so the Eq. (1) turns to

$$\mathbf{y^{(k)}} = \mathbf{A^k x} = \lambda_1^k c_1 \mathbf{x_1} \overset{k\to\infty}{\to} \begin{cases} 0 & \text{if } |\lambda_1| < 1 \\ \infty & \text{if } |\lambda_1| > 1 \end{cases}$$

$\Rightarrow$ we need some rescaling

$$(y_1^{(k)})^{\text{new}} = \frac{1|}{\lambda_1} y_1^{(k)}$$

Now $\mathbf{y^{(k)}}$ converges as $k \to \infty$.
The convergence rate is proportional to $\frac{\lambda_{\max}}{\lambda_{\text{next-to-max}}}$.

The inverse power method

This method finds the largest (in magnitude) eigenvalue of the inverse matrix $\mathbf{A}^{-1}$ which is the smallest eigenvalue of the original matrix $\mathbf{A}$.

$$\mathbf{Ax} = \lambda\mathbf{x} \Rightarrow \mathbf{A^{-1}Ax} = \mathbf{Ix} = \mathbf{x} = \lambda\mathbf{A^{-1}x} \Rightarrow \mathbf{A^{-1}x} = \lambda^{-1}\mathbf{x}$$

Note that the eigenvectors of $\mathbf{A}^{-1}$ are the same as eigenvectors of $\mathbf{A}$.
The method works well if the smallest eigenvalue is distinct. In practice, the LU method is used to solve the inverse eigenproblem instead of calculation of $\mathbf{A^{-1}}$.
The procedure:

1. Solve for $\mathbf{L}$ and $\mathbf{U}$ ($\mathbf{A} = \mathbf{LU}$) by Doolittle method.
2. Assume $\mathbf{x}^{(0)}$ with one component being 1.
3. Solve for $\mathbf{x}'$ by forward substitution $\mathbf{Lx}' = \mathbf{x}^{(0)}$.
4. Solve for $\mathbf{y}^{(1)}$ by backward substitution $\mathbf{Uy}^{(1)} = \mathbf{x}'$.
5. Scale $\mathbf{y}^{(1)}$ so that the first component is 1 $\mathbf{y}^{(1)} = \lambda^{(1)}_{\text{inverse}}\mathbf{x}^{(1)}$.
6. Repeat steps 3 to 5 with $\mathbf{x}^{(1)}$.

At convergence $\lambda = \frac{1}{\lambda^{k\to\infty}_{\text{inverse}}}$ and $\mathbf{x}^{k\to\infty}$ is the eigenvector.

$$\mathbf{Lx}' = \mathbf{x}^{(k)}, \qquad \mathbf{Uy}^{(k+1)} = \mathbf{x}', \qquad \mathbf{y}^{(k+1)} = \lambda^{(k+1)}_{\text{inv}} x^{(k+1)}$$

$$\mathbf{A} = \begin{bmatrix} 8 & -2 & -2 \\ -2 & 4 & -2 \\ -2 & -2 & 13 \end{bmatrix}$$

Assume

$$\mathbf{x}^{(0)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \text{then} \quad \mathbf{y}^{(1)} = \mathbf{A}\mathbf{x}^{(0)} = \begin{bmatrix} 8 & -2 & -2 \\ -2 & 4 & -2 \\ -2 & -2 & 13 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 9 \end{bmatrix}$$

Solving for $\mathbf{L}$ and $\mathbf{U}$ gives (see Doolittle method)

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & -0 \\ -\frac{1}{4} & 1 & 0 \\ -\frac{1}{4} & -\frac{5}{7} & 1 \end{bmatrix}, \qquad \mathbf{U} = \begin{bmatrix} 8 & -2 & -2 \\ 0 & \frac{7}{2} & -\frac{5}{2} \\ 0 & 0 & \frac{75}{7} \end{bmatrix}$$

Solve for $\mathbf{x}'$ by $\mathbf{L}\mathbf{x}' = \mathbf{x}^{(0)}$

$$\begin{bmatrix} 1 & 0 & -0 \\ -\frac{1}{4} & 1 & 0 \\ -\frac{1}{4} & -\frac{5}{7} & 1 \end{bmatrix} \begin{bmatrix} x_1' \\ x_2' \\ x_3' \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \Rightarrow \mathbf{x}' = \begin{bmatrix} 1 \\ 5/4 \\ 15/7 \end{bmatrix}$$

Solve for $\mathbf{y^{(1)}}$ by $\mathbf{Lx' = x^{(0)}}$

$$\begin{bmatrix} 8 & -2 & -2 \\ 0 & \frac{7}{2} & -\frac{5}{2} \\ 0 & 0 & \frac{75}{7} \end{bmatrix} \begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \\ y_3^{(1)} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \Rightarrow \mathrm{y}^{1)} = \begin{bmatrix} 0.3 \\ 0.5 \\ 0.2 \end{bmatrix}$$

Rescaling

$$\mathrm{y}^{1)} = 0.3 \times \begin{bmatrix} 0.3 \\ 0.5 \\ 0.2 \end{bmatrix} \quad \lambda_{\text{inverse}}^{(1)} = 0.3 \quad \Rightarrow \quad \mathbf{x}^{(1)} = \begin{bmatrix} 1.0 \\ 1.667 \\ 0.667 \end{bmatrix}$$

The final solution is

$$\lambda_3 = \frac{1}{\lambda_{\text{inv}}} = \frac{1}{0.3985} = 2.5090 \quad \text{with} \quad \mathbf{x_3} = \begin{bmatrix} 1.0 \\ 2.1457 \\ 0.5997 \end{bmatrix}$$

The shifted power method

The eigenvalues of matrix $\mathbf{A}$ can be shifted by a scalar s' by subtracting $s\mathbf{I}\mathbf{x} = s\mathbf{x}$ from both sides of $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$:

$$(\mathbf{A} - s\mathbf{I})\mathbf{x} = \lambda\mathbf{x} - s\mathbf{x} \quad \Leftrightarrow \quad \mathbf{A}_{\text{shifted}}\mathbf{x} = \lambda_{\text{shifted}}\mathbf{x}$$

Shifting the eigenvalues of a matrix may be used to find intermediate eigenvalues and/or accelerate convergence. Note that the shifting does not change the eigenvectors:

- Shifting to find the opposite extreme eigenvalues.
  Example: let the solutions for eigenvalues be $\lambda = \{1, 2, 4, 8\}$. Here 8 is the largest and 1 is the smallest eigenvalue. Solving for the largest eigenvalue by direct power method gives $\lambda_{\text{max}} = 8$. Sifting by $s = 8$ yields $\lambda = \{-7, -6, -4, 0\}$. Solving again for the largest eigenvalue gives $\lambda_{\text{max}} = -7$ (with the eigenvector corresponding to $\lambda = 1$ for non-shifted $\mathbf{A}$).
- Shifting to find intermediate eigenvalues.
  Consider the same example $\lambda = \{1, 2, 4, 8\}$. After finding the largest and the smallest eigenvalues λ_{max} and λ_{min}. Let us guess an intermediate $\lambda_{\text{guess}} = 5$ and use $s = \lambda_{\text{guess}} = 5$ to obtain $\mathbf{A}_{\text{shifted}}$. Now the λ's are $\{-4, -3, -1, 3\}$ and solving for the smallest eigenvalue will give us the eigenvector corresponding to $\lambda = 4$ in the original problem.

Meanwhile in physics

- In physics, we often deal with either symmetric $a_{ij}=a_{ji}$ or Hermithean $a_{ij}=a_{ji}^*$ matrices
  (a* stands for complex conjugate elements).
- It is important o know that all the eigenvalues for these matrices are real.
- Symmetric matrix eigenvalue problems are MUCH easier to solve
  # we can avoid complex arithmetic
  # they are generally solved using algorithms especially designed for symmetric matrices

The Jacobi method

The Jacobi method for the symmetric eigenvalue problem is no longer considered state-of-the-art (there are other methods that are somewhat faster).

However, it has the advantage that it is simple to program and to analyze, and it is no less stable or robust than the more sophisticated methods.

It is guaranteed to find all the eigenvalues and eigenvectors of a symmetric matrix in a reasonable amount of time.

The Jacobi method

The Jacobi method constructs a sequence of similarity transformations

$$A_{n+1} = Q_n^{-1} A_n Q_n \quad (A_0 = A)$$

where the Q-matrices are Givens rotation matrices of the form

$$Q_{ij} = \begin{array}{c} \\ \text{row } i \\ \\ \text{row } j \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} \text{column} & & \text{column} & & & & \\ i & & j & & & & \end{array} \\ \begin{bmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & 1 & & & & \\ & & & c & & -s & \\ & & & & 1 & & \\ & & & & & 1 & \\ & & & s & & c & \\ & & & & & & 1 \\ & & & & & & & 1 \\ & & & & & & & & 1 \end{bmatrix} \end{array} \qquad (c^2 + s^2 = 1)$$

for Givens rotational matrices

$$Q_{ij}^{-1} = Q_{ij}^T \qquad A_{n+1} = Q_{ij}^T A_n Q_{ij}$$

The Jacobi method

Multiplying a matrix A by Q_{ij}^T has the effect of replacing rows i and j by linear combinations of the original rows i and j.

Coefficients c and s can be chosen so that a zero will be introduced into the (j, i)th position of A.

However, to preserve the eigenvalues of A, we are forced to multiply by Q_{ij}, which has the effect of changing columns i and j,

… and the multiplication will normally cause the element just zeroed (a_{ji}) to become nonzero.

This is not a problem if we plan ahead; we simply have to choose c and s so that the zero is not created until the end—after remultiplication and postmultiplication.

The Jacobi method

$$\begin{bmatrix} & \vdots & & \vdots & \\ \cdots & b_{ii} & \cdots & b_{ij} & \cdots \\ & \vdots & & \vdots & \\ \cdots & b_{ji} & \cdots & b_{jj} & \cdots \\ & \vdots & & \vdots & \end{bmatrix} = B \equiv Q_{ij}^{\mathrm{T}} A Q_{ij}$$

$$= \begin{bmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & c & & & s & \\ & & & 1 & & & \\ & & & & 1 & & \\ & & -s & & & c & \\ & & & & & & 1 \\ & & & & & & & 1 \end{bmatrix} \begin{bmatrix} & \vdots & & \vdots & \\ \cdots & a_{ii} & \cdots & a_{ij} & \cdots \\ & \vdots & & \vdots & \\ \cdots & a_{ji} & \cdots & a_{jj} & \cdots \\ & \vdots & & \vdots & \end{bmatrix} \begin{bmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & c & & & -s & \\ & & & 1 & & & \\ & & & & 1 & & \\ & & s & & & c & \\ & & & & & & 1 \\ & & & & & & & 1 \end{bmatrix}$$

Although all elements in rows i and j and columns i and j may be modified by the transformation, we shall primarily be interested in the new values b_{ii}, b_{jj}, and (especially) $b_{ji} = b_{ij}$:

$$\begin{aligned} b_{ii} &= c^2 a_{ii} + s^2 a_{jj} + 2sca_{ji}, \\ b_{jj} &= s^2 a_{ii} + c^2 a_{jj} - 2sca_{ji}, \\ b_{ji} &= b_{ij} = cs(a_{jj} - a_{ii}) + (c^2 - s^2)a_{ji}. \end{aligned}$$

If we want $b_{ji} = b_{ij} = 0$, we need to choose c and s so that

$$\frac{c^2 - s^2}{cs} = \frac{a_{ii} - a_{jj}}{a_{ji}} \equiv 2\beta$$

(we can assume that $a_{ji} \neq 0$; otherwise no transformation is necessary). Substituting $c = (1 - s^2)^{1/2}$ into 3.2.4 gives

$$s^4 - s^2 + \frac{1}{4 + 4\beta^2} = 0$$

or, using the quadratic equation,

$$s^2 = \frac{1}{2} \pm \frac{\frac{1}{2}\beta}{(1 + \beta^2)^{1/2}}.$$

Thus the following values for s and c will satisfy 3.2.4 and $s^2 + c^2 = 1$:

$$s = \left(\frac{1}{2} - \frac{\frac{1}{2}\beta}{(1+\beta^2)^{1/2}}\right)^{1/2},$$

$$c = \left(\frac{1}{2} + \frac{\frac{1}{2}\beta}{(1+\beta^2)^{1/2}}\right)^{1/2}.$$

Thus by proper choice of s and c we can introduce a zero into any specified off-diagonal position, while preserving the eigenvalues (and the symmetry) of A. Now it would be nice if we could zero all the off-diagonal elements of A in succession, and after $N(N-1)/2$ transformations have a diagonal matrix that is similar to A; then we could read the eigenvalues off the diagonal.

But when we zero a new element of A, a previously zeroed element may become nonzero. Every time we knock out one off-diagonal element, others pop back up; so it might seem that our algorithm is useless.

Fortunately, although our transformed matrices never become quite diagonal, they do make steady progress toward that goal.

Theorem 3.2.1. *When the symmetric matrix A is transformed into $B = Q_{ij}^{\mathrm{T}} A Q_{ij}$, with Q_{ij} chosen so that $b_{ji} = 0$, the sum of the squares of the diagonal elements increases by $2a_{ji}^2$, while the sum of squares of the off-diagonal elements decreases by the same amount.*

Theorem 3.2.2. *If at each step of the Jacobi method the element to be zeroed satisfies $a_{ji}^2 \geq \frac{1}{2} e_k / [N(N-1)]$, then the convergence criterion*

$$\sum_{i \neq j} \sum a_{ij}^2 \leq \epsilon \sum_{i=1}^{N} \sum_{j=1}^{N} a_{ij}^2$$

will be satisfied after at most $L = N^2 \ln(1/\epsilon)$ iterations.

$$e_k = \sum_{i \neq j} \sum a_{ij}^2$$

Once we have eigenvalues λ

- Once we have eigenvalues, we may solve the system of linear equations $(\mathbf{A} - \lambda\mathbf{I}) \cdot \mathrm{x} = 0.$ to find a set of solutions $x = \{x_1, x_2, \ldots x_n\}$ for the each value of λ.
- These solutions are called eigenvectors.
- For Hermithean matrices, the eigenvectors corresponding to distinct eigenvalues are orthogonal.

Comments on the method above …

- ✓ In general, the scheme above for solving the eigenvalue problem looks very straightforward.
- ✓ However, this scheme is getting unstable as the size of the matrix increases.
- ✓ The standard libraries have many robust and stable computer programs for solving eigenvalue problem.
- ✓ In particularly, programs based on the Faddeev-Leverrier method are very popular and successful in atomic and molecular structure calculations.
- ✓ The Lanczos algorithm is a good choice for large and sparse matrices which are common in many-body problem.

Comments on Matrix Computing

Many scientific programming problems arise from the improper use of arrays on computers

- ⇨ Computers are finite: you can run out of memory or run very slowly when dealing with LARGE matrices
  for storing A(10000,10000) matrix -> 1 GB memory
- ⇨ Processing time: matrix operation, on average, require on the order of N^3 steps. Doubling the dimension leads to eightfold increase in processing time
- ⇨ Paging: when a program runs out of RAM (virtual memory on HDD). When a program is near the memory limit, even a slight increase in a matrix size may lead to an order of magnitude increase in running time.

Comments on Matrix Computing (cont.)

- Matrix storage: the computer stores matrices as a linear string of numbers. For a(2,2) matrix

```
Fortran: a(1,1) a(2,1) a(1,2) a(2,2)
C: a(0,0) a(0,1) a(1,0) a(1,1)
```

- Processing sizes to subprograms: you must watch that the sizes of your matrices do not exceed the bounds in the subprograms.

```
Main program
    dimension a(100)
subroutine One(a)
    dimension a(10)
    …
    a(300) = 8.0
```

Do not write your own matrix subroutines
unless you solve a simple problem.
Get them from a well established scientific library

Some libraries

- Short list: free
  - netlib (meta library/free)
  - slatec (free)
  - lapack (free)
  - lapack++ (free)

  commercial
  - nag ($$$)
  - imsl ($$$)
  - essl ($$$)
- Extended list:

  http://www.physics.odu.edu/~godunov/computing/lib_net.html