PY410 / 505
Computational Physics 1

Salvatore Rappoccio
Next up: Linear algebra

- Covered in Chapter 4 of Garcia and Chapter 2 of Numerical Recipes

- Huge number of applications!
  - Complex circuit diagrams
  - Coupled oscillators
  - General solution of fitting arbitrary curves

- We’ll learn how to:
  - Solve linear equations
  - Compute eigenvalues
  - Apply these to various applications
You should all be familiar with the basics of linear algebra:

- Vectors
- Matrices
- Solving matrix equations
- Gaussian (or Gauss-Jordan) elimination
  
  ```latex
  \begin{bmatrix}
  \cos 90^\circ & \sin 90^\circ \\
  -\sin 90^\circ & \cos 90^\circ 
  \end{bmatrix}
  \begin{bmatrix}
  a_1 \\
  a_2 
  \end{bmatrix}
  = \begin{bmatrix}
  0 \\
  0 
  \end{bmatrix}
  ```

- We’ll go over the computational issues here

http://en.wikipedia.org/wiki/Gaussian_elimination

http://xkcd.com/184/
• Recall Cramer’s rule:

• If we have the determinant of a matrix $A$:

\[
|A| = \sum_{i=1}^{n} (-1)^{i+j} A_{ij} |R_{ij}|,
\]

• where $R_{ij}$ is the “residual matrix” or “minor” of $A$, removing row $i$ and column $j$

• Then the inverse of the matrix is:

\[
A_{ij}^{-1} = (-1)^{i+j} \frac{|R_{ji}|}{|A|}.
\]

• This is a recursive rule
Linear Algebra

• Computing this “brute force” way is okay for small \( n \), but problematic for large \( n \) (\( >10 \))
• Scales as \( n \) factorial (\( n! \))

• To see this, consider an expansion of the determinant:

\[
|A| = \sum_{P} (-1)^{P} A_{1p_1} A_{2p_2} \cdots A_{np_n} ,
\]

• Here, \( P \) runs over the \( n! \) permutations of the indices
• \( 20! = 2.43\times10^{18} \) .... yipes! (which is “yipes, factorial”)

• OK, well, scratch that idea.
  – What else ya got, Sal?
• Medium sized matrices (n \approx 10-10^3)
  – Gaussian elimination, LU-decomposition, and Householder method

• Larger matrices (n > 10^3)
  – Storage becomes a problem
  – Cannot practically do this for arbitrary matrices
  – However, most matrices are “sparse” with lots of zeroes in practical applications
  – Can, however, store and solve these fairly well
Linear Algebra

• Linear algebra is the raison d’être for numpy.
  – Let’s just use it.

• Most of the scipy algorithms for linear algebra are from LAPACK. (Linear Algebra Package)
  • http://www.netlib.org/lapack/

• Other options:
  – BLAS (Basic Linear Algebra Subprograms)
    • http://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms
  – LAPACK
  – BOOST Basic Linear Algebra Library
    • http://www.boost.org/doc/libs/1_54_0/libs/numeric/ublas/doc/index.htm
  – matlab (the “mat” in “matlab” stands for “matrix”)
    • http://www.mathworks.com/products/matlab/
Linear Algebra

• Game plan:
  – Use numpy software.
  – Go over algorithms that are used internally
  – Check into use cases
Linear Algebra

• So if you recall the Gaussian Elimination, we have a matrix equation:

\[ Ax = b, \]

• Or, written out for the case of \( n=3 \):

\[
\begin{pmatrix}
  a_{00} & a_{01} & a_{02} \\
  a_{10} & a_{11} & a_{12} \\
  a_{20} & a_{21} & a_{22}
\end{pmatrix}
\begin{pmatrix}
  x_0 \\
  x_1 \\
  x_2
\end{pmatrix}
= 
\begin{pmatrix}
  b_0 \\
  b_1 \\
  b_2
\end{pmatrix},
\]

\[
\begin{pmatrix}
  a_{00}x_0 + a_{01}x_1 + a_{02}x_2 \\
  a_{10}x_0 + a_{11}x_1 + a_{12}x_2 \\
  a_{20}x_0 + a_{21}x_1 + a_{22}x_2
\end{pmatrix}
= 
\begin{pmatrix}
  b_0 \\
  b_1 \\
  b_2
\end{pmatrix}.
\]
Linear Algebra

• We take linear combinations of the rows to convert the matrix into “reduced row echelon form”:
  – Multiply first equation by $a(10)/a(00)$ and subtract from second:
    \[
    a_{10}x_0 + a_{11}x_1 + a_{12}x_2 - (a_{10}/a_{00})(a_{00}x_0 + a_{01}x_1 + a_{02}x_2) = b_1 - (a_{10}/a_{00})b_0.
    \]
  – Then $x_0$ is eliminated from the second equation:
    \[
    (a_{11} - a_{10}a_{01}/a_{00})x_1 + (a_{12} - a_{10}a_{02}/a_{00})x_2 = b_1 - (a_{10}/a_{00})b_0,
    \]
  – which can be written as:
    \[
    a'_1x_1 + a'_2x_2 = b'_1.
    \]
  – Repeat until you run out of rows
  – Example:

\[
\begin{bmatrix}
1 & 3 & 1 & 9 \\
1 & 1 & -1 & 1 \\
3 & 11 & 5 & 35 \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
1 & 3 & 1 & 9 \\
0 & -2 & -2 & -8 \\
0 & 2 & 2 & 8 \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
1 & 3 & 1 & 9 \\
0 & -2 & -2 & -8 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
1 & 0 & -2 & -3 \\
0 & 1 & 1 & 4 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}
\]
Linear Algebra

• Can also think about this in terms of an “augmented” matrix where you add the vector \( b \) as another column:

\[
\begin{pmatrix}
 a_{00} & a_{01} & a_{02} & b_0 \\
 a_{10} & a_{11} & a_{12} & b_1 \\
 a_{20} & a_{21} & a_{22} & b_2 \\
\end{pmatrix}
\
\]

• You eliminate each row iteratively, reducing the dimension by one each time:

\[
\begin{pmatrix}
 a_{00} \\
 a_{10} - a_{00}a_{10} \\
 a_{20} - a_{00}a_{20} \\
\end{pmatrix}
\begin{pmatrix}
 a_{01} \\
 a_{11} - a_{01}a_{10} \\
 a_{21} - a_{01}a_{20} \\
\end{pmatrix}
\begin{pmatrix}
 a_{02} \\
 a_{12} - a_{02}a_{10} \\
 a_{22} - a_{02}a_{20} \\
\end{pmatrix}
\begin{pmatrix}
 b_0 \\
 b_1 - b_0a_{10}/a_{00} \\
 b_2 - b_0a_{20}/a_{00} \\
\end{pmatrix}
\]

\[
\equiv \begin{pmatrix}
 a'_{00} & a'_{01} & a'_{02} & b'_0 \\
 0 & a'_{11} & a'_{12} & b'_1 \\
 0 & a'_{21} & a'_{22} & b'_2 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
 a''_{00} \\
 a''_{10} - a'_{11}a'_{21}/a'_{11} \\
 0 \\
\end{pmatrix}
\begin{pmatrix}
 a''_{01} \\
 a''_{11} \\
 0 \\
\end{pmatrix}
\begin{pmatrix}
 a''_{02} \\
 a''_{12} \\
 0 \\
\end{pmatrix}
\begin{pmatrix}
 b''_0 \\
 b'_1 \\
 b''_2 \\
\end{pmatrix}
\]

\[
\equiv \begin{pmatrix}
 a''_{00} & a''_{01} & a''_{02} & b''_0 \\
 0 & a''_{11} & a''_{12} & b''_1 \\
 0 & 0 & a''_{22} & b''_2 \\
\end{pmatrix}
\]
To solve for the actual equation, we then use “back substitution”, starting at the last row:

- For instance

\[ x_2 = \frac{b''}{a''_{22}}. \]

- Then we use the second equation

\[ x_1 = \frac{1}{a''_{11}} [b'' - a''_{12}x_2]. \]

- And finally

\[ x_0 = \frac{1}{a''_{00}} [b'' - a''_{01}x_1 - a''_{02}x_2]. \]
• Gaussian elimination is is $O(n^3)$ operations
  – $n^2$ matrix elements, and $O(n)$ row operations on each

• Back-substitution is $O(n^2)$ operations

• So the total is $O(n^3) + O(n^2) \sim O(n^3)$ for large $n$

• Much better than $O(n!) \gg O(n^3)$

• Easy to extend to arbitrarily large $n$’s
  – But, as we mentioned earlier, storage becomes a problem
• You can see one problem already
• If any of the rows have a zero as the first element, then you divide by zero and get an exception

• So, need to make sure this doesn’t happen by “partial pivoting”:
  – Before performing the ith row operation, search for the element $a_{ki}$ (k=i,...,n-1) with the largest magnitude
  – If k != i, interchange rows i and k of the augmented matrix, and interchange $x_i \leftrightarrow x_k$
  – Perform as usual
• Will not fail for small $a_i$’s, also stable to roundoff errors

• Note: in “full” pivoting you swap rows AND columns
Linear Algebra

• Another variation is Gauss-Jordan elimination
  – Does not require the backsubstitution step
  – Replaces $A$ by the inverse “in place”, avoiding memory copy

• So, we have another augmented equation (again in $n=3$):

$$
\begin{pmatrix}
  a_{00} & a_{01} & a_{02} \\
  a_{10} & a_{11} & a_{12} \\
  a_{20} & a_{21} & a_{22}
\end{pmatrix}
\begin{pmatrix}
  x_0 \\
  x_1 \\
  x_2
\end{pmatrix}
= 
\begin{pmatrix}
  b_0 & 1 & 0 & 0 \\
  b_1 & 0 & 1 & 0 \\
  b_2 & 0 & 0 & 1
\end{pmatrix}
$$

• This is a way of stating:

$$Ax = b, \quad \text{and} \quad AA^{-1} = I.$$
Linear Algebra

• Algorithm:
  – Start with zeroth row, divide by \( a(00) \) and subtract
  – Subtract all zeroth column entries (as in Gaussian elimination)
  – For row \( i=1,...,n-1 \), divide by diagonal element \( a(ii) \) and eliminate elements in column \( i \) other than the diagonal by subtracting \( a(ij) \) times the \( i \)th row elements from the \( j \)th row
    • Gaussian elimination: subtracts only those below diagonal
    • Gauss-Jordan: subtracts ALL elements where \( i \neq j \)
  – Simultaneously perform on the augmented \((b \ 1)\) matrix on the RHS
Linear Algebra

- In the end we get:

\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
b_0 \\
b_1 \\
b_2
\end{pmatrix}
\begin{pmatrix}
a_{00}^{-1} & a_{01}^{-1} & a_{02}^{-1} \\
a_{10}^{-1} & a_{11}^{-1} & a_{12}^{-1} \\
a_{20}^{-1} & a_{21}^{-1} & a_{22}^{-1}
\end{pmatrix}
= 
\begin{pmatrix}
x_0 \\
x_1 \\
x_2
\end{pmatrix}
\begin{pmatrix}
a_{00}^{-1} & a_{01}^{-1} & a_{02}^{-1} \\
a_{10}^{-1} & a_{11}^{-1} & a_{12}^{-1} \\
a_{20}^{-1} & a_{21}^{-1} & a_{22}^{-1}
\end{pmatrix}
\]

- A is replaced by a unit matrix
- b is reduced to input vector x
- Unit matrix on RHS is replace by inverse $A^{-1}$

- Can also implement pivoting in this algorithm to make sure we have numeric stability
Another variation is “LU decomposition” (“lower-upper”)
Reduce A to an L*U product:

\[
\begin{pmatrix}
a_{00} & a_{01} & a_{02} \\
a_{10} & a_{11} & a_{12} \\
a_{20} & a_{21} & a_{22}
\end{pmatrix}
= \begin{pmatrix}
\alpha_{00} & 0 & 0 \\
\alpha_{10} & \alpha_{11} & 0 \\
\alpha_{20} & \alpha_{21} & \alpha_{22}
\end{pmatrix}
\begin{pmatrix}
\beta_{00} & \beta_{01} & \beta_{02} \\
\beta_{11} & \beta_{12} \\
0 & 0 & \beta_{22}
\end{pmatrix}
\]

Then solve the problem in two steps :
– Solve \( Ly = b \) with forward substitution:

\[
y_0 = \frac{b_0}{\alpha_{00}} , \quad y_i = \frac{1}{\alpha_{ii}} \left[ b_i - \sum_{j=0}^{i-1} \alpha_{ij} y_j \right] , \quad i = 1, 2, \ldots, n - 1 .
\]

– Solve \( Ux = y \), since \( Ax = L(Ux) = Ly = b \), with backward substitution :

\[
x_{n-1} = \frac{y_{n-1}}{\beta_{n-1,n-1}} , \quad x_i = \frac{1}{\beta_{ii}} \left[ y_i - \sum_{j=i+1}^{n-1} \beta_{ij} x_j \right] , \quad i = n - 2, \ldots, 0 .
\]
Factoring the matrix $A = LU$ can be done with “Crout’s Algorithm”:

- Set $\alpha(iii) = 1$ for $i=0,...,n-1$
- for each $j = 0,...,n-1$:
  - for $i = 0,...,j$
    - compute
      \[
      \beta_{ij} = a_{ij} - \sum_{k=0}^{i-1} \alpha_{ik}\beta_{kj}.
      \]
  - for $j = j+1,...,n-1$
    - compute
      \[
      \alpha_{ij} = \frac{1}{\beta_{jj}} \left[ a_{ij} - \sum_{k=0}^{j-1} \alpha_{ik}\beta_{jk} \right].
      \]

- Replaces $A$ by $LU$ “in place”:

\[
\begin{pmatrix}
a_{00} & a_{01} & a_{02} \\
a_{10} & a_{11} & a_{12} \\
a_{20} & a_{21} & a_{22}
\end{pmatrix} \rightarrow \begin{pmatrix}
\beta_{00} & \beta_{01} & \beta_{02} \\
\alpha_{10} & \beta_{11} & \beta_{12} \\
\alpha_{20} & \alpha_{21} & \beta_{22}
\end{pmatrix}
\]

The RH matrix is NOT a matrix! It is a storage unit in the computer!
Linear Algebra

• Simple special case: Tridiagonal matrices

• If you have a problem such as:

\[
M = \begin{pmatrix}
2 - c & -1 & 0 & \cdots & 0 & 0 \\
-1 & 2 - c & -1 & \cdots & 0 & 0 \\
0 & -1 & 2 - c & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & 2 - c & -1 \\
0 & 0 & 0 & \cdots & -1 & 2 - c
\end{pmatrix},
\]

• This matrix is sparse!
  – Can solve in $O(n)$ operations

• Equations are:

\[
M_i^- u_{i-1} + M_i^0 u_i + M_i^+ u_{i+1} = b_i, \quad i = 1, \ldots, n-1.
\]

• Can show that (recursively):

\[
\alpha_{i-1} = -\frac{M_i^-}{M_i^0 + \alpha_i M_i^+}, \quad \beta_{i-1} = \frac{b_i - \beta_i M_i^+}{M_i^0 + \alpha_i M_i^+}.
\]
Linear Algebra

- Can start from the right boundary value for $i=n-2,\ldots,0$

$$u_n = \alpha_{n-1} u_{n-1} + \beta_{n-1} \quad \text{if} \quad \alpha_{n-1} = 0, \quad \beta_{n-1} = u_n,$$

- and then solve from the left boundary value for $i=1,\ldots,n-1$

$$u_{i+1} = \alpha_i u_i + \beta_i,$$

- So we “sweep twice” and obtain $O(n)$ operations
• Examples:
  – Polynomial fits (again)
  – Circuit diagrams
  – Boundary value problems
Recall: General fitting of curves

• This is a matrix equation, so we define the “design matrix”:

\[
A_{ij} = \frac{Y_j(x_i)}{\sigma_i}
\]

\[
A = \begin{bmatrix}
Y_1(x_1)/\sigma_1 & Y_2(x_1)/\sigma_1 & \cdots \\
Y_1(x_2)/\sigma_2 & Y_2(x_2)/\sigma_2 & \cdots \\
\vdots & \vdots & \ddots
\end{bmatrix}
\]

• Then our chi2 minimization becomes:

\[
(A^T A) \tilde{a} = A^T \tilde{b}
\]

• So:

\[
\tilde{a} = (A^T A)^{-1} A^T \tilde{b}
\]
If we define the “correlation matrix”:

\[ C = (A^T A)^{-1} \]

Then the uncertainty on \( a_j \) is:

\[ \sigma_{a_j} = \sqrt{C_{jj}} \]
As a first example, let’s look at polynomial fits

\[ y = \sum_{k=0}^{m-1} a_k x^k . \]

Slight generalization of the linear fit we did previously

General solution is to minimize the chi2 :

\[ \chi^2(\vec{a}) \equiv \sum_{i=0}^{n-1} \left( \frac{y_i - y(x; \vec{a})}{\sigma_i} \right)^2 \]

In this case :

\[ \chi^2(\vec{a}) = \sum_{i=0}^{n-1} \left( \frac{y_i - \sum_{j=0}^{M} a_j x^j}{\sigma_i} \right)^2 \]
• Our design matrix is therefore:

\[ A_{ij} = \frac{x_i^j}{\sigma_i} \]

• Caveat: This oftentimes is ill-formed, so don’t go too crazy here. Typically we do quadratic, cubic, quartic, but above that it strains credibility.
Polynomial fits

• Can finally generalize our formalism to arbitrary functional fits

• Example : quadratic fit for our CO2 data!
Linear Algebra

• Example:
  – Kirchoff’s Law for a Wheatstone bridge:

This is a matrix equation:

\[ \begin{pmatrix} R_1 + R_v & R_2 & R_v \\ R_1 + R_a & \ -R_a & -R_3 \\ R_x + R_a & -R_2 - R_x - R_a & R_x \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} v \\ 0 \\ 0 \end{pmatrix} \]
Try to “zero” the potential at $V_G$:

$$V_G = \left( \frac{R_x}{R_3 + R_x} - \frac{R_2}{R_1 + R_2} \right) V_s$$
Example: boundary value problems

Consider:

\[ \frac{d^2 u}{dx^2} = -\frac{\pi^2}{4} (u + 1), \]

with Dirichlet boundary conditions \( u(0) = 0 \) and \( u(1) = 1 \)

We can discretize this:

\[ \frac{2u_i - u_{i+1} - u_{i-1}}{h^2} = \frac{\pi^2}{4} (u_i + 1), \quad i = 1, \ldots, N - 1 \]

This is therefore a sparse matrix equation with \( c = h^2 \pi^2/4 \):

\[
M = \begin{pmatrix}
2 - c & -1 & 0 & \cdots & 0 & 0 \\
-1 & 2 - c & -1 & \cdots & 0 & 0 \\
0 & -1 & 2 - c & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & 2 - c & -1 \\
0 & 0 & 0 & \cdots & -1 & 2 - c
\end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix}
u_1 \\
u_2 \\
u_3 \\
\vdots \\
u_{N-2} \\
u_{N-1}
\end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix}
u_0 + c \\
c \\
c \\
\vdots \\
c \\
u_N + c
\end{pmatrix},
\]
Hands on!
• Today: Eigenvalues and eigenvectors, more hands on

• See Chapter 11 of Numerical Recipes

• In this case, even they recommend using packaged software for eigenvalues and eigenvectors, but let’s get the general gist here
Example: normal modes of a harmonic oscillator between \( n \) objects

\[
\mathcal{L} = \frac{1}{2} \sum_{j,k}^n M_{jk} \ddot{q}_j \dot{q}_k - \frac{1}{2} \sum_{j,k}^n A_{jk} q_j q_k ,
\]

“Normal mode” is the mode of the system where all of the coordinates oscillate with some frequency \( \omega \):

\[
q_j(t) = x_j e^{i\omega t} , \quad \sum_{j=1}^n [A_{jk} - M_{jk}\omega^2] x_j = 0 , \quad (A - M\omega^2)x = 0 .
\]

Homogeneous matrix equation has solutions for \( \omega \) for which the determinant is zero:

\[
\det \begin{vmatrix} A - M\omega^2 \end{vmatrix} = 0 .
\]
This is an instance of a general class of eigenvalue and eigenvector problems. So, if \( A \) is an \( n \times n \) matrix, \( x \) is a column vector (the “right” eigenvector), \( \lambda \) is a number (the “eigenvalue”), and:

\[
Ax = \lambda x,
\]

Then the equation is satisfied when:

\[
|A - \lambda I| = \sum_{k=0}^{n} a_k \lambda^k = 0,
\]

This is an \( n \)-th degree polynomial in \( \lambda \) that depends on the matrix elements. \( N \)-degree polynomial \( \implies \) \( n \) different eigenvalues, so we need to solve for them.
Eigenvalues and Eigenvectors

• Example:

\[
A = \begin{pmatrix}
1 & 1 \\
1 & 2
\end{pmatrix}
\]

• The eigenvalue equation and eigenvalues are:

\[
A = \begin{pmatrix}
1 - \lambda & 1 \\
1 & 2 - \lambda
\end{pmatrix}
\]

\[(1-\lambda)(2-\lambda) - 1 = \lambda^2 - 3\lambda + 1 = 0 \quad \Rightarrow \quad \lambda_0 = \frac{3 + \sqrt{5}}{2}, \quad \lambda_1 = \frac{3 - \sqrt{5}}{2}.\]

• We solve for the eigenvectors:

\[
\begin{pmatrix}
1 & 1 \\
2 & 1
\end{pmatrix}
\begin{pmatrix}
x_0 \\
x_1
\end{pmatrix} = (1 \pm \sqrt{2})
\begin{pmatrix}
x_0 \\
x_1
\end{pmatrix}
\]

\[
x^{(0)} = \begin{pmatrix}
\frac{1}{\sqrt{3}} \\
\frac{2}{\sqrt{3}}
\end{pmatrix}, \quad x^{(1)} = \begin{pmatrix}
\frac{1}{\sqrt{3}} \\
\frac{2}{\sqrt{3}}
\end{pmatrix}.
\]
Eigenvalues and Eigenvectors

• Also consider:

\[
A = \begin{pmatrix}
1 & 1 \\
-1 & 1
\end{pmatrix}.
\]

• So we have:

\[
(1 - \lambda)^2 + 1 = \lambda^2 - 2\lambda + 2 = 0
\]

• And thus the eigenvalues are complex:

\[
\lambda_0 = 1 + i, \quad \lambda_1 = 1 - i, \quad \text{where} \quad i = \sqrt{-1}.
\]
“Hey Sal! I know what we can do!” you say. “We can just find the roots of the characteristic polynomial with our root-finding methods!”

“Excellent idea in principle, my clever students,” I say. “But unfortunately, it’s hard in practice.”

– Need to know roughly where the roots are ahead of time
– You don’t get the eigenvectors this way either

So, we’ll look at ways to do both!
First, define “left” eigenvectors:

\[ yA = \lambda y , \]

In this case, \( y \) is a row vector.

This implies:

\[ A^T y = \lambda y , \]

Since:

\[ |A^T - \lambda I| = |A - \lambda I| , \]

then the left and right eigenvalues are the same.

However the left and right eigenvectors are not necessarily the same.

Left eigenvector 0, though, is orthogonal to right eigenvector 1, etc.
If we consider a complex matrix $A$, we can define the Hermitian conjugate:

$$(A^\dagger)_{ij} = a_{ji}^*,$$

- This is also referred to as the “conjugate transpose”
- If the $a_{ij}$ are real, this is just the transpose

Define a “normal” matrix if the conjugate transpose commutes with the matrix:

$$N \cdot N^\dagger = N^\dagger \cdot N.$$

Note that normal matrices have the same left and right eigenvector sets.
Eigenvalues and Eigenvectors

• A Hermitian matrix (or self-adjoint matrix) is defined when

\[ H^\dagger = H , \]

• When the elements are real, this is called a “symmetric” matrix:

\[ S^T = S , \]

• These special matrices have:
  – n real eigenvalues
  – eigenvectors are orthogonal
  – the set of eigenvectors is “complete” (spans the n-dim space)
Eigenvalues and Eigenvectors

• Computation of eigenvalues/eigenvectors for symmetric or Hermitian matrices depends on a nice property

• If we consider the matrices formed by the right and left eigenvectors $X$ and $Y$:

$$X = \begin{pmatrix} \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} \\ \sqrt{\frac{2}{3}} & -\sqrt{\frac{2}{3}} \end{pmatrix}, \quad Y = \begin{pmatrix} \sqrt{\frac{2}{3}} & \sqrt{\frac{1}{3}} \\ \sqrt{\frac{2}{3}} & -\sqrt{\frac{1}{3}} \end{pmatrix} = \frac{2\sqrt{2}}{3} X^{-1}.$$

• In general:

$$Y \propto X^{-1}$$

• Can choose the normalization appropriately so that

$$Y = X^{-1}$$
Eigenvalues and Eigenvectors

• How does this help us?

\[ Y \cdot A \cdot X = X^{-1} \cdot A \cdot X \]
\[ = X^{-1} \cdot \lambda \cdot X \]
\[ = \lambda \]

• Can exploit this fact to compute the eigenvalues!
Eigenvalues and Eigenvectors

• This is a similarity transformation!

\[ A \rightarrow Z^{-1} \cdot A \cdot Z \]

• The interesting bit is that symmetry transformations leave the eigenvalues unchanged:

\[
\det |Z^{-1} \cdot A \cdot Z - \lambda \mathbf{1}| = \det |Z^{-1} \cdot (A - \lambda \mathbf{1}) \cdot Z|
\]

\[
= \det |Z| \det |A - \lambda \mathbf{1}| \det |Z^{-1}|
\]

\[
= \det |A - \lambda \mathbf{1}|
\]

• So, we can solve this easier problem instead
Eigenvalues and Eigenvectors

• Let $X$ be the matrix of eigenvectors
  – If the matrix $A$ is real and symmetric, then the eigenvectors are real and orthonormal, and:
    \[ X^{-1} = X^T \]

  – If the matrix $A$ is Hermitian, the matrix of eigenvectors is unitary:
    \[ X^{-1} = X^\dagger \]
Eigenvalues and Eigenvectors

• Generalized eigenvalue problem: solve
  $$A \cdot x = \lambda B \cdot x$$

• Strategy: Successively apply similarity transformations until the matrix is “almost” diagonal
  – Either diagonal, block diagonal, or tridiagonal and also easy to solve

• Nonsymmetric matrices will not, in general, have similarity matrices with real components
  – So, cannot use real similarity matrix
  – But! “Almost” can: will reduce to block-diagonal with two-by-two blocks replacing the complex eigenvalues

– Think

$$
\begin{pmatrix}
\cos \theta & \sin \theta \\
-\sin \theta & \cos \theta
\end{pmatrix}
$$
Eigenvalues and Eigenvectors

• Can then repeatedly apply similarity transformations until we can solve the problem “easily”:

$$A \rightarrow P_1^{-1} \cdot A \cdot P_1 \rightarrow P_2^{-1} \cdot P_1^{-1} \cdot A \cdot P_1 \cdot P_2$$

$$\rightarrow P_3^{-1} \cdot P_2^{-1} \cdot P_1^{-1} \cdot A \cdot P_1 \cdot P_2 \cdot P_3 \rightarrow \text{etc.}$$

• If we end up in completely diagonal form, then the eigenvectors are just the columns of the total transformation:

$$X_R = P_1 \cdot P_2 \cdot P_3 \cdot \ldots$$

• Sometimes just want eigenvalues, in which case can stop when we reduce to triangular form

  – Then eigenvalues are the diagonal elements!
Eigenvalues and Eigenvectors

• One strategy we will look at:
  – Reduce to tridiagonal form by the Householder algorithm (Chapter 11 Section 2 of Numerical Recipes)
  – Solve the tridiagonal matrix problem using a QL algorithm with implicit shifts (Chapter 11 Section 3 of Numerical Recipes)
    • $Q = \text{orthogonal}$
    • $L = \text{lower triangular matrix}$

• These are basically “uninteresting” to go through the gory details, because they’re mostly just math tricks

• Instead, let’s just focus on an application
• Consider a linear triatomic molecule
  – Fowles and Cassiday, Chapter 11, Section 4
  – Taylor, Chapter 11, Section 6
• Approximate by masses on springs
Eigenvalues and Eigenvectors

• Lagrangian for the general case is:

\[
L = \frac{1}{2} m_1 \ddot{x}_1^2 + \frac{1}{2} m_2 \ddot{x}_2^2 + \frac{1}{2} m_3 \ddot{x}_3^2 - \frac{1}{2} \left[ k_{12} (x_1 - x_2)^2 + k_{23} (x_2 - x_3)^2 \right]
\]

• The equations of motion are

\[
\begin{align*}
m_1 \ddot{x}_1 &= \frac{d}{dt} \left( \frac{\partial L}{\partial \dot{x}_1} \right) = \frac{\partial L}{\partial x_1} = -k_{12} x_1 + k_{12} x_2 \\
m_2 \ddot{x}_2 &= \frac{d}{dt} \left( \frac{\partial L}{\partial \dot{x}_2} \right) = \frac{\partial L}{\partial x_2} = k_{12} x_1 - (k_{12} + k_{23}) x_2 + k_{23} x_3 \\
m_3 \ddot{x}_3 &= \frac{d}{dt} \left( \frac{\partial L}{\partial \dot{x}_3} \right) = \frac{\partial L}{\partial x_3} = -k_{23} x_3 + k_{23} x_2
\end{align*}
\]

• For our specific problem, we have

\[
m_1 = m_3, \quad m_2 = 2m_1, \quad k_{12} = k_{23} = K
\]
Eigenvalues and Eigenvectors

• Can be written as a matrix equation:

\[ M\ddot{q} = -Kq \]

• with normal modes of the form:

\[ q(t) = a \cos(\omega t - \delta) \]

• The normal mode frequencies are eigenvalues of the generalized eigenvalue equation:

\[ Ka = \omega^2 Ma \]
• The eigenvalues are therefore:

\[
\omega_1^2 = 0, \quad \omega_2^2 = \frac{K}{m}, \quad \omega_3^2 = \frac{K}{m} \left( 1 + \frac{2m}{M} \right)
\]

• And the eigenvectors are:

\[
\mathbf{a}_1 = a_{11} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad \mathbf{a}_2 = a_{12} \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}, \quad \mathbf{a}_3 = a_{13} \begin{pmatrix} 1 \\ -\frac{2m}{M} \\ 1 \end{pmatrix},
\]
Eigenvalues and Eigenvectors

- “triatomic” python notebook