PHY407: Computational Physics
Fall, 2017

Instructor: Paul Kushner

TAs: Heather Fong, Alex Cabaj, Haruki Hirasawa

Lecture 4

- Correction to Lab 04
- Solving linear systems
- Eigenvalue systems
- Solving nonlinear systems
- Finding minima and maxima

Correction to Lab04

- Unfortunately, the original version of the lab contained a question, Q2b, that corresponded to a worked example in the book.
- This morning I posted a new version of the lab with a corrected version of the problem.
- The lab is now due at 5 p.m. on October 7 intsead of 5 p.m. on October 6.
- I apologize for the confusion.

Solving Linear Systems

- In linear algebra courses you learn to solve linear systems of the form

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{v}$$

using Gaussian elimination.

- This works pretty well in many cases. Let's do an example based on Newman's `gausselim.py`, for

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, v = \begin{pmatrix} 5 \\ 6 \end{pmatrix}$$

Solving Linear Systems

- In linear algebra courses you learn to solve linear systems of the form

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{v}$$

using Gaussian elimination.

- This works pretty well in many cases. Let's do an example based on Newman's `gausselim.py`, for

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, v = \begin{pmatrix} 5 \\ 6 \end{pmatrix} \Rightarrow x = \begin{pmatrix} -4.0 \\ 4.5 \end{pmatrix}$$

Newman's approach

```
from numpy import array,empty

example = input('Enter an example [1-3]: ')
example = int(example)

if example==1:
    A = array([[1,2],[3,4]],float)
    v = array([5,6], float)

N = len(v)

# Gaussian elimination

for m in range(N):

    # Divide by the diagonal element
    div = A[m,m]
    A[m,:] /= div
    v[m] /= div

    # Now subtract from the lower rows
    for i in range(m+1,N):
        mult = A[i,m]
        A[i,:] -= mult*A[m,:]
        v[i] -= mult*v[m]

# Backsubstitution
x = empty(N,float)
for m in range(N-1,-1,-1):
    x[m] = v[m]
    for i in range(m+1,N):
        x[m] -= A[m,i]*x[i]

print(x)
```

The Problem with Gaussian Elimination

- But it’s easy to come up with examples that don’t work so well.
- The example below is a valid system but the original code will “break”.

$$A = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix}, v = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

The Problem with Gaussian Elimination

- But it’s easy to come up with examples that don’t work so well.
- The example below is a valid system but the original code will “break”.

$$A = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix}, v = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \Rightarrow x \approx \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

The Problem with Gaussian Elimination

- Sample output from `gausselim-examples.py`

```
A is  [[  1.00000000e-20   1.00000000e+00] [
1.00000000e+00   1.00000000e+00]]
v is  [ 1.  0.]
x from linalg.solve is [-1.  1.]
but x from gaussian elimination is...[ 0.  1.]
```

$$A = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix}, v = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \Rightarrow x \approx \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

The Problem with Gaussian Elimination

- But it's easy to come up with examples that don't work so well.
- The example below is a valid system but the original code will "break".
- For any small number in the upper left hand corner, we'll tend to get inaccuracies.
- So we use pivoting (row swapping).

$$A = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix}, v = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \Rightarrow x \approx \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

Partial Pivoting

- Sample output

```
A is  [[  1.00000000e-20   1.00000000e+00] [
1.00000000e+00   1.00000000e+00]]
v is  [ 1.  0.]
x from linalg.solve is [-1.  1.]
and x from partial pivoting is...[-1.  1.]
```

$$A = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix}, v = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \Rightarrow x \approx \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

LU Decomposition

- Gaussian elimination can be written as a series of matrix multiplications on both sides resulting in an upper triangular matrix multiplying *x*,

$$L_N L_{N-1} \ldots L_0 A x = U x = \begin{pmatrix} + & + & + & + & + \\ 0 & + & + & + & + \\ 0 & 0 & + & + & + \\ 0 & 0 & 0 & + & + \\ 0 & 0 & 0 & 0 & + \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ \vdots \\ x_N \end{pmatrix} = L_N L_{N-1} \ldots L_0 v$$

which can be solved by back substitution.

LU Decomposition

- A related matrix *L* is lower triangular

$$L = \left(L_N L_{N-1} \dots L_0\right)^{-1} = L_0^{-1} \dots L_N^{-1} = \begin{pmatrix} + & 0 & 0 & 0 & 0 \\ + & + & 0 & 0 & 0 \\ + & + & + & 0 & 0 \\ + & + & + & + & 0 \\ + & + & + & + & + \end{pmatrix}$$

and

$$LU = A, \text{ so } LUx = v, \text{ or } Ux = L^{-1}v$$

- Combine with pivoting, we get a robust and efficient method to solve the system for any *v*.

LU Decomposition

- In Lab04, we use numpy.linalg.solve, which does the LU decomposition for us.
- But we could use scipy.linalg.lu to save the pivoting and LU information for multiple solves (if we have multiple *v*).
- Note also that we can use the decomposition to solve for the *inverse* of *A*, although we don't use it in this lab. Try `numpy.linalg.inv`

LU Decomposition

- When do you think the LU decomposition approach won’t work very well?

LU Decomposition

- When do you think the LU decomposition approach won't work very well?

When A is singular, or close to singular. E.g.

```
A=array([[1,2],[2,4+1e-16]],float)
v = array([3,5],float)
solve(A,v) #returns error
```

Eigenvalues and Eigenvectors

- The other standard system to solve is

$$Av = \lambda v \text{ or } \mathrm{AV=VD}$$

Eigenvalues and Eigenvectors

- The other standard system to solve is

$$Av = \lambda v \text{ or } \mathrm{AV=VD}$$

Real Symmetric or Hermitian Matrix

Eigenvector

Eigenvalue

Orthonormal matrix

Diagonalized, with eigenvalues on the diagonal

Eigenvalues and Eigenvectors

- The other standard system to solve is

$$Av = \lambda v \text{ or } \mathrm{AV=VD}$$

Real Symmetric or Hermitian Matrix

Eigenvector

Eigenvalue

Orthonormal matrix

Diagonalized, with eigenvalues on the diagonal

- We use built in functions to carry out the *QR* algorithm, described in the next slide.

QR Algorithm

- Use Gramm Schmidt orthogonalization to transform the columns of *A* to an orthogonal basis and form columns into an orthonormal matrix *Q*. Then $A=QR$, where $R=Q^TA$ is upper triangular.
- Now perform iterations of the following form:

$$RQ = Q^T AQ = A' = Q'R'$$

$$R'Q' = Q'^T Q^T AQQ' = A'' = Q''R''$$

- Eventually, this iteration converges to an output Q^kR^k that is diagonal with the eigenvalues on the diagonal. The eigenvectors are the columns of $V=QQ'Q''\ldots$

Simple application

```
from numpy.linalg import eigh
eigh(array([[2,1],[1,2]]))
(array([ 1.,  3.]),
array([[-0.70710678,  0.70710678],
[ 0.70710678,  0.70710678]]))
```

Simple application

```
from numpy.linalg import eigh
eigh(array([[2,1],[1,2]]))
(array([ 1.,  3.]),
array([[-0.70710678,  0.70710678],
[ 0.70710678,  0.70710678]]))
```

Matrix A

Eigenvalues

Orthonormal axes at $\pm\pi/4$

Question

```
from numpy.linalg import eigh
eigh(array([[2,3],[1,2]]))
```

Matrix A

What happens next?

Finding Roots of Nonlinear Equations

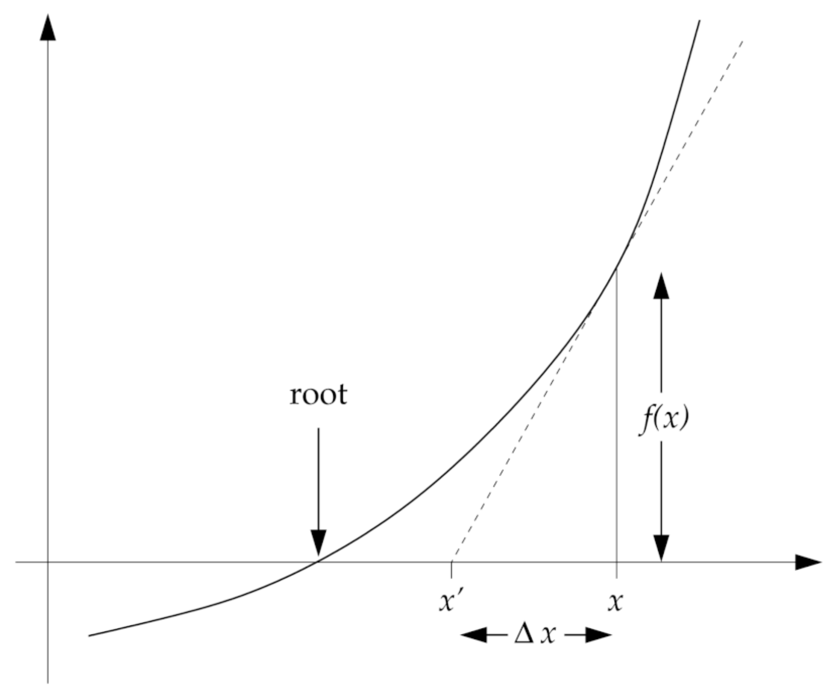
- Newman discusses several examples of this: relaxation, Newton's method, bisection, etc.
- *Relaxation* can be understood using graphs. Let's look at Example 6.3, which solves the Ising model of ferromagnetism. The problem requires solving

$$x = \tanh(ax), a = const.$$

- See the script `relaxation_demo.py`. We successively replace x by tanh(ax).

Other methods

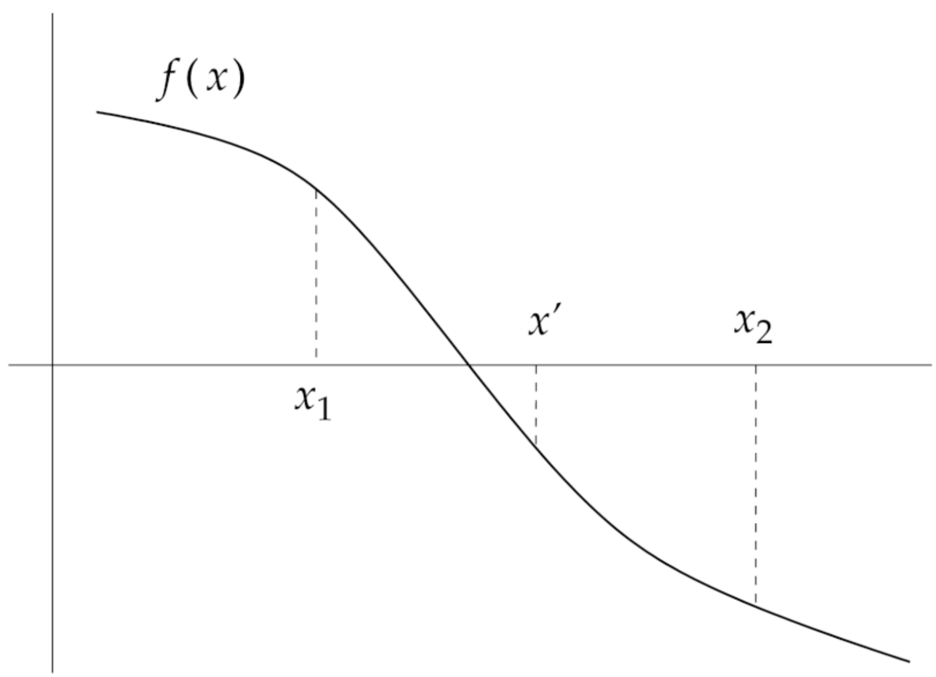
- *Newton's method* solves for the roots of equations using information about the tangent to obtain the roots of f(x)=0.
  - The algorithm is to successively replace x by x-f(x)/f'(x).
  - See `newton_demo.py`



- Secant method: f'(x) found using forward difference.

Other methods

- *Bisection* brackets the roots on either side of f(x)=0 and uses the midpoint as a subsequent bracket.



Convergence Characteristics

- To start with, keep iterating until the desired number of digits doesn’t change.

Convergence Characteristics

- To start with, keep iterating until the desired number of digits doesn't change.
- This tabulates convergence characteristics.

Method	Convergence Test	Formula
Relaxation	Taylor expansion, assuming proximity to root	$\varepsilon = \dfrac{x - x'}{1 - 1/f'(x)}$
Newton	Taylor expansion about solution of f(x)=0. Very fast convergence.	$\varepsilon = x - x'$ $\varepsilon = O\left(\varepsilon_0^{2^N}\right)$
Binary search	Error decreases by a factor of two each iteration	$\varepsilon = \Delta / 2^N$

Convergence Characteristics

- To start with, keep iterating until the desired number of digits doesn't change.
- This tabulates convergence characteristics.

Method	Convergence Test	Formula	
Relaxation	Taylor expansion, assuming proximity to root	$\varepsilon = \dfrac{x - x'}{1 - 1/f'(x)}$	Easy but not reliable.
Newton	Taylor expansion about solution of f(x)=0. Very fast convergence.	$\varepsilon = x - x'$ $\varepsilon = O\left(\varepsilon_0^{2^N}\right)$	Fast go-to method
Binary search	Error decreases by a factor of two each iteration	$\varepsilon = \Delta / 2^N$	Good to know about.

Convergence Characteristics

- For all methods, bear in mind the possibility of multiple roots and be prepared to test carefully.

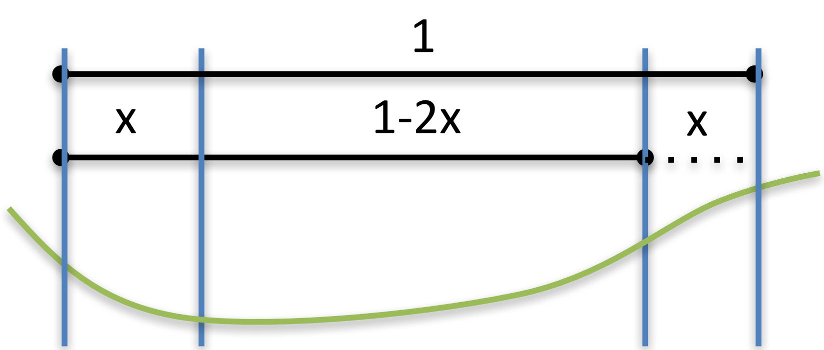
Method	Convergence Test	Formula
Relaxation	Taylor expansion, assuming proximity to root	$\varepsilon = \dfrac{x - x'}{1 - 1/f'(x)}$
Newton	Taylor expansion about solution of f(x)=0. Very fast convergence.	$\varepsilon = x - x'$ $\varepsilon = O\left(\varepsilon_0^{2^N}\right)$
Binary search	Error decreases by a factor of two each iteration	$\varepsilon = \Delta / 2^N$

Finding Maxima and Minima

- Like finding the roots of the derivative of a function.

Finding Maxima and Minima

- Like finding the roots of the derivative of a function.
- Golden ratio search is analogous to binary search: we want to shrink the interval bracketing extremum consistently each step
- Find x and z satisfying:

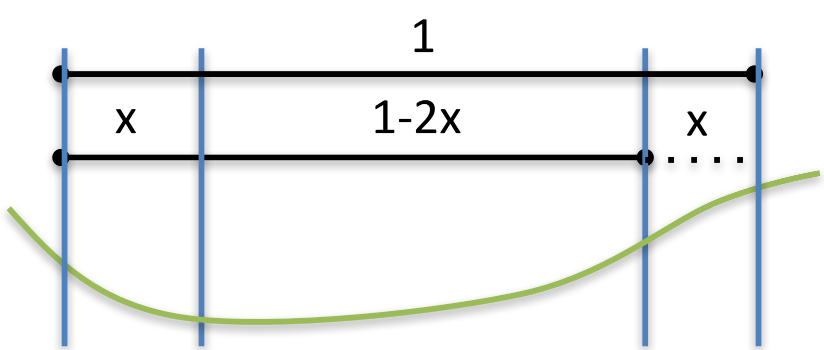


$$z = \frac{1}{1-x} = \frac{1-x}{x}$$

$$\Rightarrow z = \frac{1+\sqrt{5}}{2} = 1.618\ldots,\ x = 0.382\ldots$$

Finding Maxima and Minima

- Like finding the roots of the derivative of a function.
- Golden ratio search is analogous to binary search: we want to shrink the interval bracketing extremum consistently each step
- Find x and z satisfying:



$$z = \frac{1}{1-x} = \frac{1-x}{x}$$

$$\Rightarrow z = \frac{1+\sqrt{5}}{2} = 1.618\ldots,\ x = 0.382\ldots$$

z is the *golden ratio*