EPS 109 Review Session

Jizhou Wu, Chirag Manghani

12/05/2023

Course Evaluation

Please fill the course evaluations at https://course-evaluations.berkeley.edu/

We are aiming for >70% . Upon reaching the 70% mark we will bump up your grade for the last assignment by 2 points

What did we learn this semester?

- Statistics (Average, standard deviation, correlation)
- Python Basics- Loops, list v.s. numpy array, numpy functions
- Python Visualization Libraries and Functions
- Mandelbrot/Julia Set
- Diffusion Limited Aggregation(DLA) and random walk
- Finding root of equation: (Bisection, Scant, Newton)
- Solving Partial Differential Equation (e.g. Heat Equation)
- Solving Ordinary Differential Equation (e.g. Kepler Orbit)
- Fourier Transform and Fourier Series (Audio Files)
- Image Processing (RGB, Labeling ,Quantization)

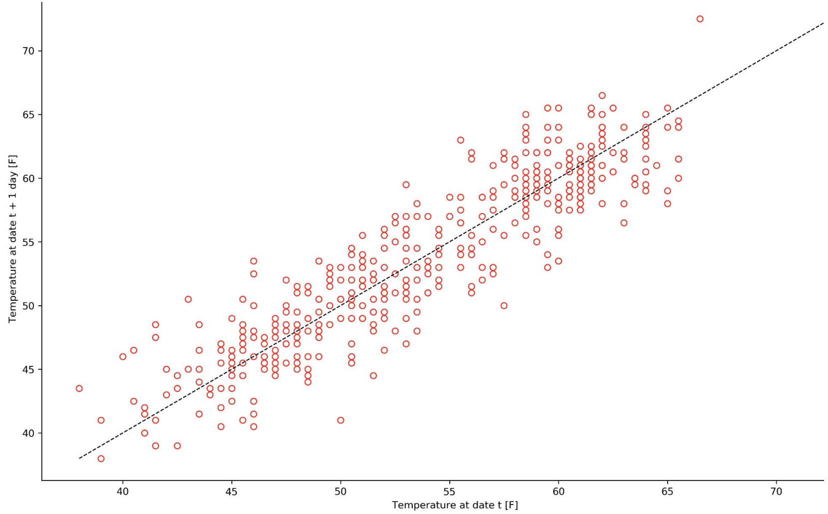
Statistics

- Average or Mean $$\bar{x} = \frac{\sum_{i=0}^{N} x_i}{N}$$
- Standard Deviation of all measurement: $$\sigma = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N}$$
- Error bar of a sample (Standard Deviation in Numpy): $$\epsilon = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N - 1}$$

Statistics

- Pearson Correlation Coefficient:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2} \times \sqrt{\sum_{j=1}^{n}(y_j - \bar{y})^2}}$$



- A statistical measure of correlation between X and Y
- r=1 : positive linear correlation; r=0: no correlation: r=-1: negative linear correlation

Python list vs Numpy array

- Python list: l = [1,2,3,4,5]
- Easy to add new element: l.append(6)
- Initialize: l = [0] *5 OR l = [0 for i in range(5)]
    - L = [[0]* 5 for i in range(5)] **//2D list** L = [[0]*5]*5 **//will not work**

Numpy arrays can accommodate a lot more numbers, easy to manipulate

- Numpy array: arr = np.array([1,2,3,4,5])
- **Initialize**: np.zeros((51,51)), np.ones((51,51)) **// 2D arrays**
    - np.linspace(start, stop, n)
    - np.arange(start, stop, step)
- **Math commands** np.cos(), np.sin(), np.var(), np.std()

Manipulating Arrays

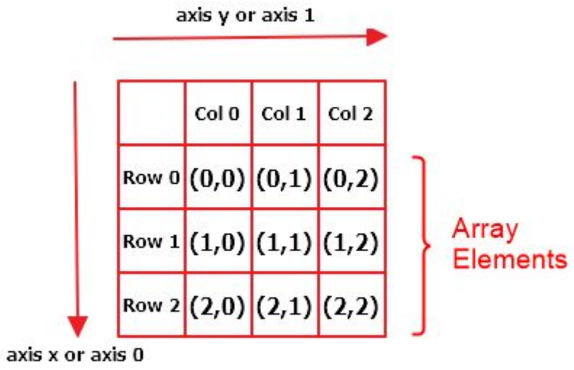
- Slicing: arr[50:100] (subarray containing element from arr[50] to arr[99])

  Arr[50:100: 10] (elements skipping 10 elements in between)

For 2 Dimensional Arrays

- for i in range(arr.shape[0]):
- for j in range(arr.shape[1]):

  ...



- Find the maximum/minimum?
  - Initialize a max/min variable
  - Iterate over each element in the matrix and remember to update the current max/min

Plots & Visualizations

Matplotlib: `x` and `y` are sequences of values.

Function	Description
`plt.plot(x, y)`	Creates a line plot of `x` against `y`
`plt.scatter(x, y)`	Creates a scatter plot of `x` against `y`
`plt.hist(x, bins=None)`	Creates a histogram of `x`; `bins` can be an integer or a sequence
`plt.bar(x, height)`	Creates a bar plot of categories `x` and corresponding heights `height`

Matplotlib cheat sheet: https://matplotlib.org/cheatsheets/cheatsheets.pdf

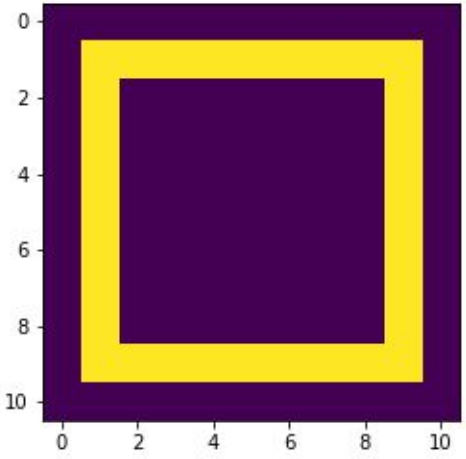
Pixel Graphics

- Assign values to numpy 2-dimensional array

```
n = 11
data = np.zeros((n,n))

data[1:10, 1:10:8] = 1
data[1:10:8, 1:10] = 1

plt.imshow(data, interpolation='nearest')
plt.show()
```

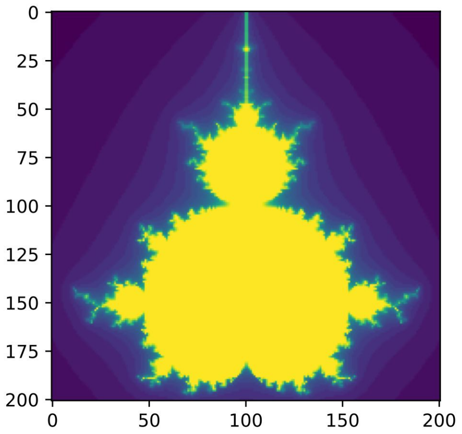


Fractals: Mandelbrot Set

$$z_{n+1} = z_n^2 + C$$

- $z_0 = 0$,Iterate enough steps, then determine convergence

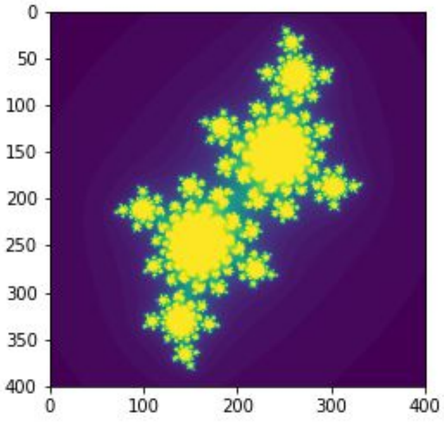
- A set of all points C



Fractals: Julia Set

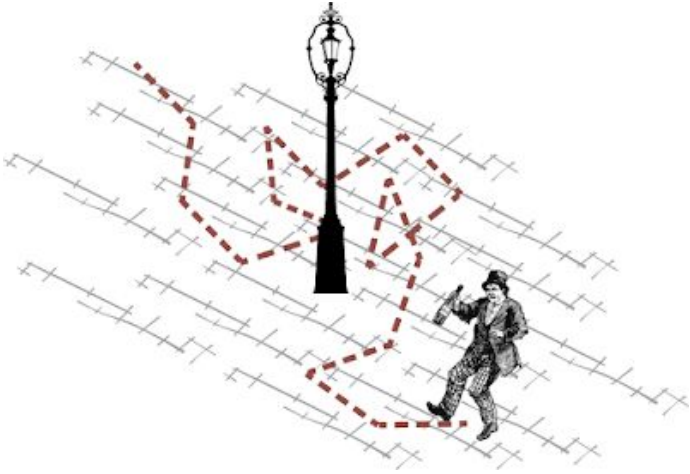
$$z_{n+1} = z_n^2 + C$$

- C is fixed to some complex number
- Start with different z, Iterate enough steps, then determine convergence
- A set of initial value z
- Julia set is a subset of Mandelbrot Set

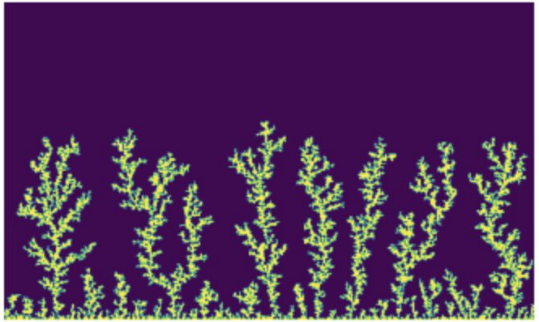


Random Walk

- Generate a random number r between (0, 1) and determine the walking direction
- Four directions with same probability
- Trajectory of wandering drunk



DLA

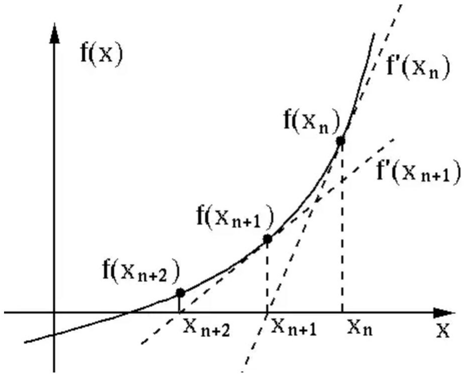


- Diffusion Limited Aggregation (DLA)
- Generate a random number r between (0, 1) and determine the walking direction
- Stick together if in contact with other particles or the wall

Finding Root

- Bisection
- Scant
- Newton: (Approximate $f(x) = f(x_n) + f'(x_n)(x - x_n)$)

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



From :https://www.quora.com/What-is-Newtons-method

Partial Differential Equation

- Heat Equation: $$\frac{\partial T}{\partial t} = k\frac{\partial^2 T}{\partial x^2}$$
- Discretize: $$\frac{\partial T}{\partial t} = \frac{T(x, t + \Delta t) - T(x, t)}{\Delta t} \qquad \frac{\partial T}{\partial x} = \frac{T(x + \Delta x, t) - T(x, t)}{\Delta x}$$

$$\frac{\partial^2 T}{\partial x^2} = \frac{\frac{\partial T}{\partial x}(T, x) - \frac{\partial T}{\partial x}(T, x - \Delta x)}{\Delta x} = \frac{T(x + \Delta x, t) - 2T(x, t) + T(x - \Delta x, t)}{\Delta x^2}$$

Dimensionless Constant

Iteration Formula: $$T(x,t+\Delta t) = \frac{\Delta t\ k}{\Delta x^2}\left[T(x+\Delta x,t) - 2T(x,t) + T(x-\Delta x,t)\right] + T(x,t)$$

Translation Into Code: `Tnew[i] = eta * (T[i+1] - 2*T[i] + T[i-1]) + T[i];`

Stable Solution? eta < 0.5 (1D); eta < 0.25 (2D); eta < 1/6 (3D);

$$\eta = \frac{\Delta t\ k}{\Delta x^2}$$

Convert the units!

Case I. Stationary Solution

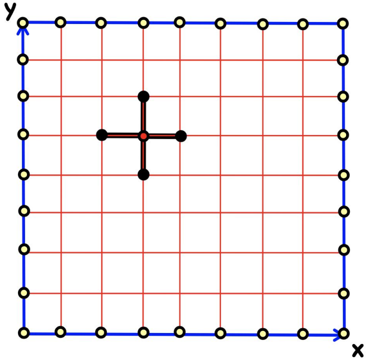
- $$\frac{\partial T}{\partial t} = 0$$

- $$\frac{\partial^2 T}{\partial x^2} = 0$$

- $T[i+1] - 2T[i] + T[i-1] = 0$

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

$$\frac{T(x+\Delta x,y) - 2T(x,y) + T(x-\Delta x,y)}{\Delta x^2} + \frac{T(x,y+\Delta y) - 2T(x,y) + T(x,y-\Delta y)}{\Delta y^2} = 0$$



Case I. Stationary Solution

- Jacobi : Make a new array and update the value based on old array Mostly used method
- 1D: T_new[i] = (T[i-1] + T[i+1]) / 2
- 2D: T_new[i,j] = (T[i-1,j] + T[i+1,j] + T[i,j-1] + T[i,j+1]) / 4

- Gauss-Seidel : Update array value with the updated value in this loop
- 1D: T[i] = (T[i-1] + T[i+1]) / 2
- 2D: T[i,j] = (T[i-1,j] + T[i+1,j] + T[i,j-1] + T[i,j+1]) / 4

Case II. Time dependent PDE

- $$\frac{T(x, t+\Delta t) - T(x,t)}{\Delta t} = k\frac{T(x+\Delta x, t) - 2T(x,t) + T(x-\Delta x, t)}{\Delta x^2}$$

- $$T(x, t+\Delta t = T(x,t) + \underbrace{\frac{k\Delta t}{\Delta x^2}}_{\eta}[T(x+\Delta x, t) - 2T(x,t) + T(x-\Delta x, t)]$$

- Stable solution: $\eta < \frac{1}{2}$ (1D) ; $\eta < \frac{1}{4}$ (2D)

Keep in mind:

Convert index(i,j,n) to real time and distance

$$t = n\Delta t$$
$$x = i\Delta x$$
$$y = j\Delta y$$

- Stable solution: (1D) ; (2D)

Ordinary Differential Equation(ODE)

- $\frac{dy}{dt} = f(y, t)$
- Discretize: $\frac{y(t_n + h) - y(t_n)}{h} = f(y, t_n)$

y=y+h*f(y)

- Euler: $\boxed{\begin{aligned} y_{n+1} &= y_n + h\ f(y_n, t_n) \\ t_{n+1} &= t_n + h \end{aligned}}$ $y_n = y(t_n)$ and $t_n = t_0 + nh$

Runge Kutta Methods

Runge Kutta method

$$\boxed{\vec{y}_{n+1} = \vec{y}_n + \frac{h}{6}\left[\vec{F}_1 + 2\vec{F}_2 + 2\vec{F}_3 + \vec{F}_4\right]}$$

$$\begin{aligned}
\vec{F}_1 &= \vec{f}(t_n, \vec{y}_n) \\
\vec{F}_2 &= \vec{f}(t_n + \tfrac{h}{2}, \vec{y}_n + \tfrac{h}{2}\vec{F}_1) \\
\vec{F}_3 &= \vec{f}(t_n + \tfrac{h}{2}, \vec{y}_n + \tfrac{h}{2}\vec{F}_2) \\
\vec{F}_4 &= \vec{f}(t_n + h, \vec{y}_n + h\vec{F}_3)
\end{aligned}$$

This approach reduce the 4th error to O(h^4).

It yields a tremendous gain in efficiency.

➔ **demonstration in homework.**

F1=f(y)

F2=f(y+F1*h/2)

F3=f(y+F2*h/2)

F4=f(y+F3*h)

y=y+(F1+2*F2+f*F3+F4)*h/6

Planet Orbit

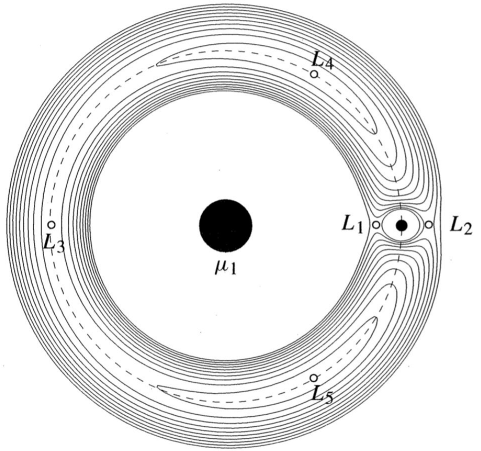
- Conserved Quantity in Planet Motion:
- Energy
- Angular Momentum(conserved for all central force system)
- Runge Lenz Vector

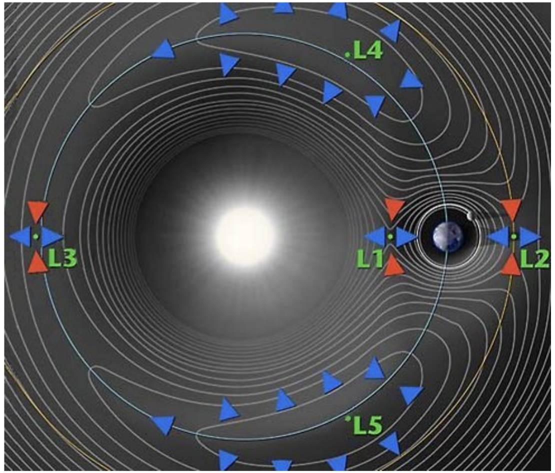
Lagrangian Points

- 5 stationary points

- 3 force reached balance

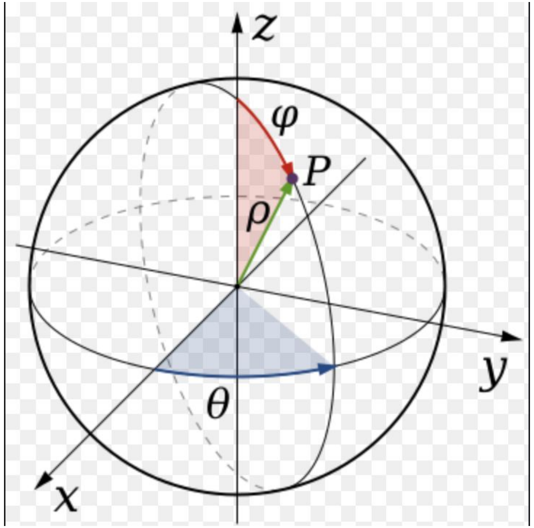
- Gravity of Sun
- Gravity of Planet
- Centrifugal Force





Distance on sphere

- Latitude: $\Delta l_x = R cos(\theta) \Delta \phi$
- Longitude: $\Delta l_y = R \Delta \theta$
- Distance : $d = \sqrt{\Delta l_x^2 + \Delta l_y^2}$



Fourier Transform

- Definition: (expand function f(x) in terms of infinite number of exponential plane waves functions)

$$e^{ikx}$$

- Fourier Amplitude derived from Fourier Transform: $F(k) = \int_{-\infty}^{+\infty} f(x)e^{-ikx}\,dx$

- Inverse: $f(x) = \int_{-\infty}^{+\infty} \frac{dk}{2\pi} F(k)e^{ikx}$

Fourier Series

- In reality, impossible to tackle infinite frequencies
- Fourier Series(Assumption of Periodicity (0,L)) :
- $f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n cos(n\frac{2\pi}{L}x) + \sum_{n=1}^{\infty} b_n sin(n\frac{2\pi}{L}x)$
- $a_n = \frac{2}{L}\int_0^L f(x)cos(n\frac{2\pi}{L}x)\,dx$
- $b_n = \frac{2}{L}\int_0^L f(x)sin(n\frac{2\pi}{L}x)\,dx$

Why?

Please calculate

the following integrals:

$$\int_0^L cos(nx)cos(mx)\,dx$$

$$\int_0^L cos(nx)sin(mx)\,dx$$

$$\int_0^L sin(nx)sin(mx)\,dx$$

Discrete Fourier Transform

- $fsin(k) = \sum_{j=0}^{n-1} y[j]sin(2\pi jk/n)$
- $fcos(k) = \sum_{j=0}^{n-1} y[j]cos(2\pi jk/n)$
- $f^2(k) = fsin^2(k) + fcos^2(k)$

Fast Fourier Transform

- import scipy
- scipy.fft(x)
- Much faster than manual implementation using “**for loop**”

Image Processing

- Gray image: 2-dimension array
- np.zeros([nx,ny],dtype=np.uint8)
- (data type integer: 0-255)

- RGB image: 3-dimension array , another axis for color
- np.zeros([nx,ny],3,dtype=np.uint8)

True-color Image

- Can also be stored in a *double* array with values ranging from 0 to 1
- For a pixel (i,j):
  - The red component is stored in

    `RGB[i,j,1]`
  - The green component is stored in

    `RGB[i,j,2]`
  - The blue component is stored in

    `RGB[i,j,3]`
  - The RGB triplet is stored in

    `RGB[i,j,:]`

The Color Planes of a Truecolor Image


Good luck with
your finals!