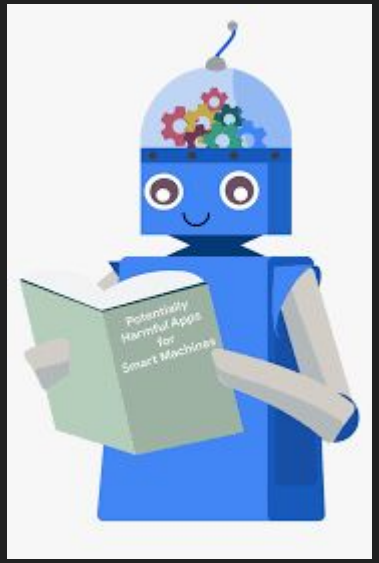
Supplemental Lecture 4

Raphael Baer-Way

Intro to Machine Learning

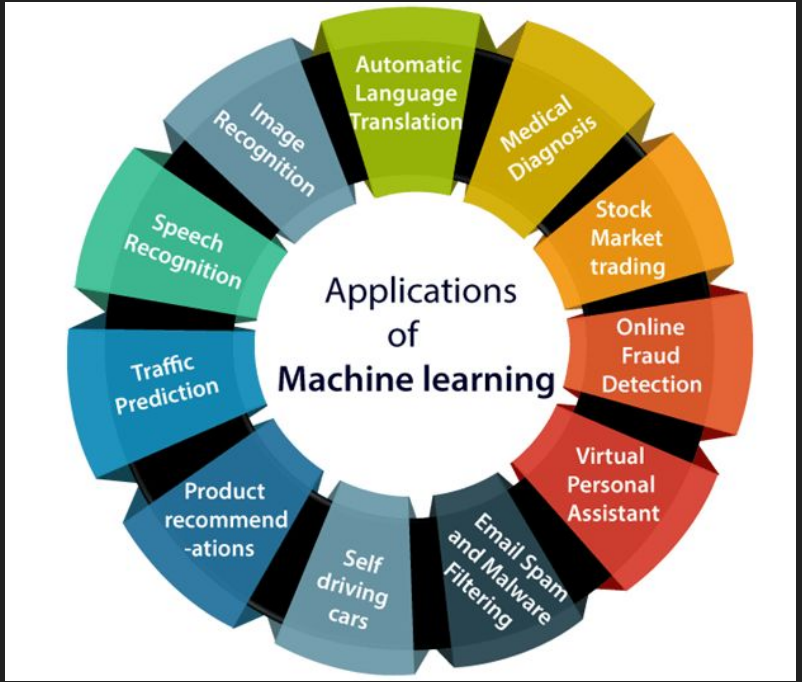
- Programming real-life applications is really hard!
  - Real life is not a perfect model
  - Too many factors to code

- What if we let the computer learn through examples training rather than explicitly code everything?
  - Learns through experience and reward



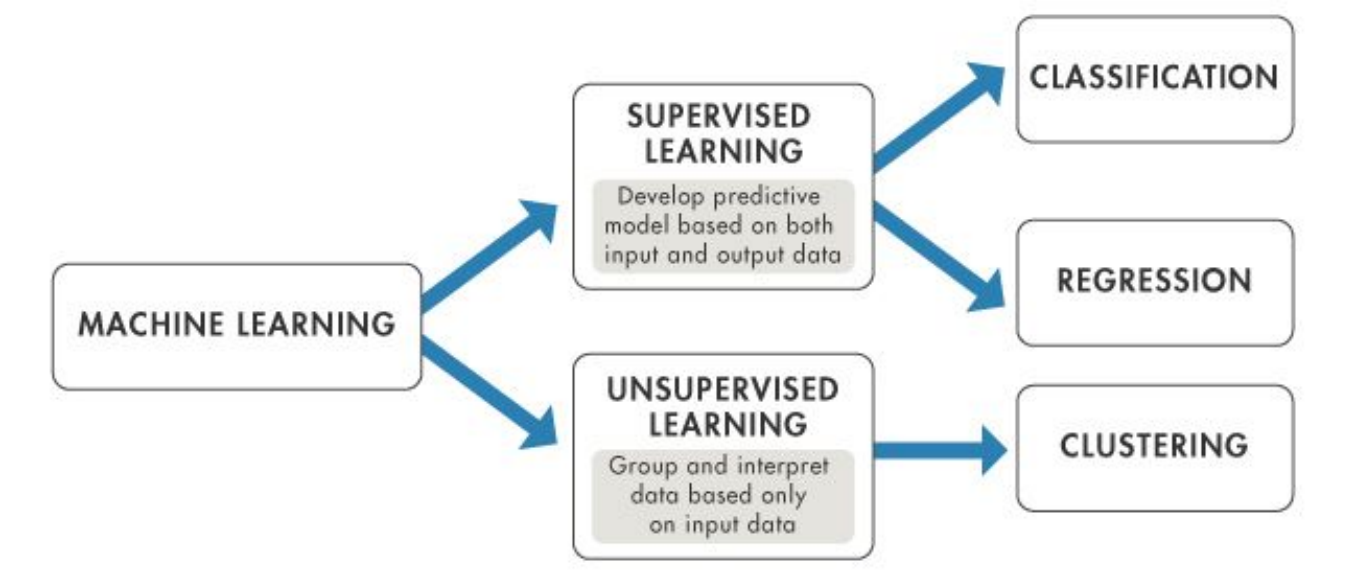
Machine Learning Applications

- Self-driving cars
- Netflix & Tiktok recommendations
- Siri & Alexa speech recognition
- Google Translate
- Medical diagnosis
- Face recognition



Machine Learning

Two types of machine learning:



Machine Learning

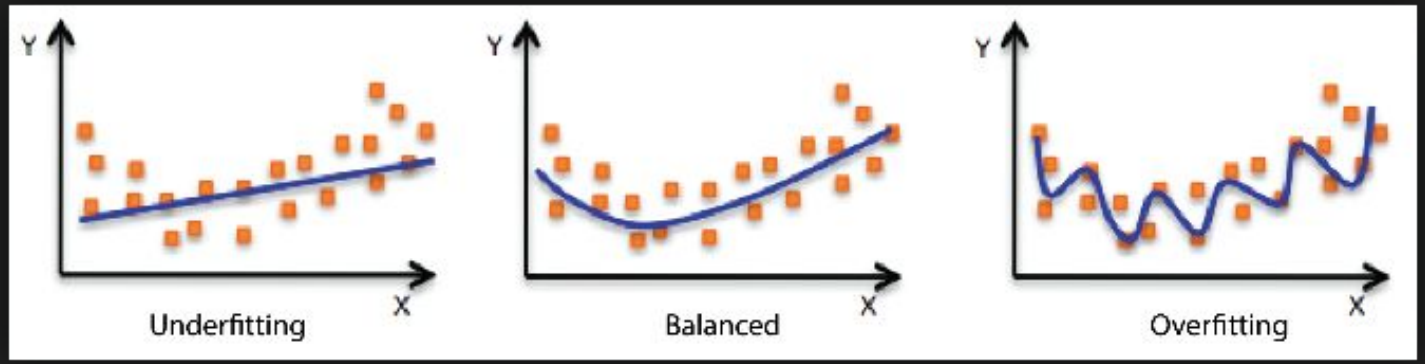
Two types of machine learning:

- Supervised: (input data and labels, output predicted labels)
  - Regression—fitting to given data (includes basic linear regression)!
  - Classification—is this image a person or a car?

- Unsupervised: (input data and no labels, output ex: machine creates common groups)
  - Clustering—Netflix classifying movies as “irreverent thrillers”

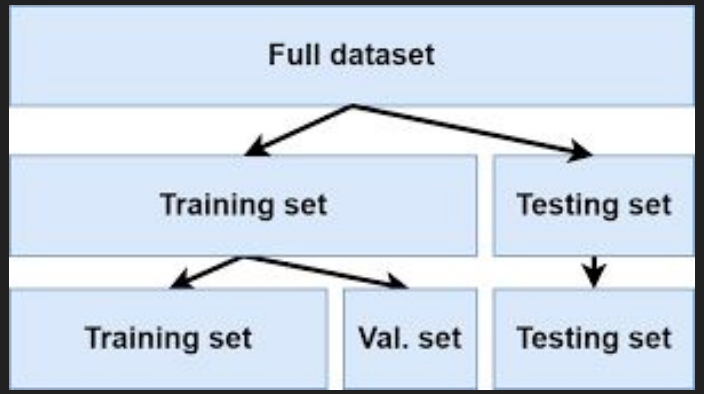
Regression

- Many different types of models—linear, polynomial, logistic
- Calculate parameters that minimize the loss given input data (x) and output labels (y)
  - Real data is never perfect! There will always be errors/noise
- Important to consider simplicity versus overfitting
  - Use regularization, cross-validation, different loss function



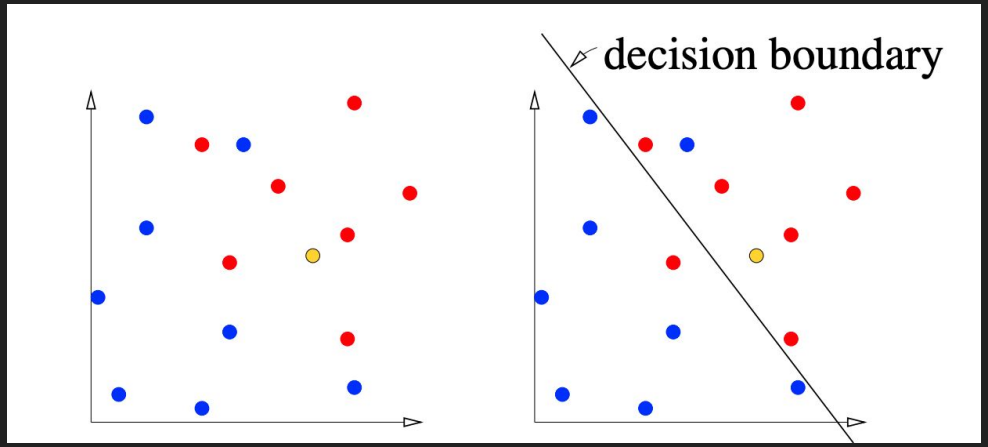
Using Data

- Split the data into a training set and a test set
  - Training set to train the model
  - Testing set to evaluate the accuracy of the model on never before seen data
- Very important not to use test set as training data!! (independent evaluation)
- More important to have high test accuracy than training accuracy



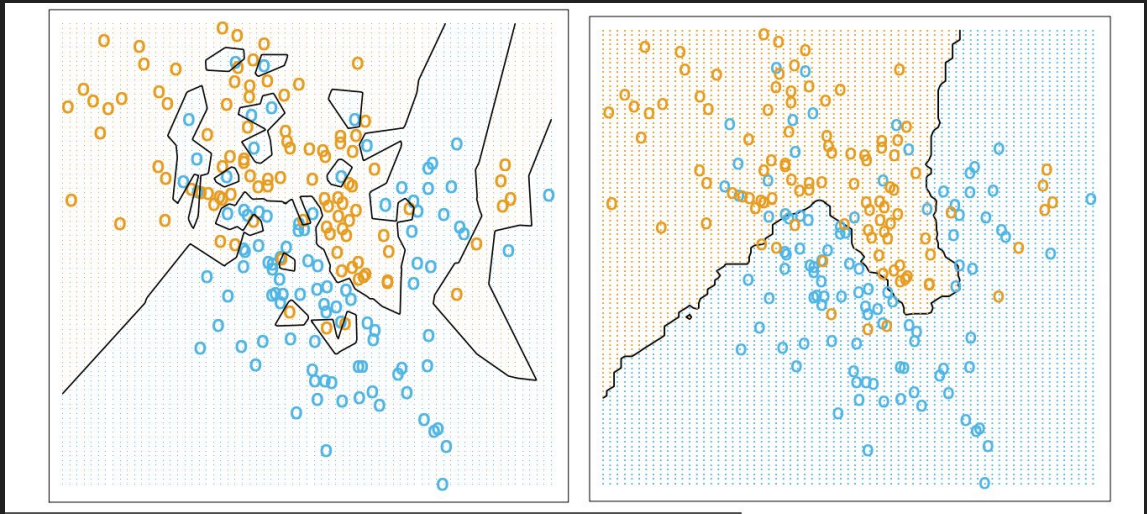
Classification

- Given many data points with different class labels
- You are then asked to predict/classify new data points
- Define a decision boundary to split the data



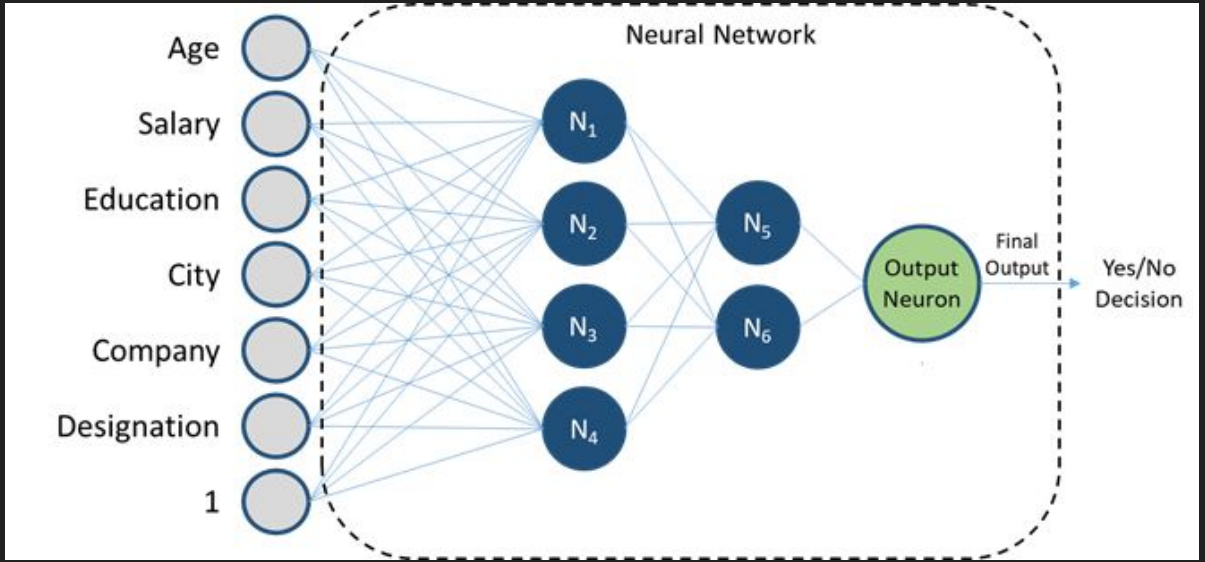
Classification

- Often data is not easily separable
- Need tradeoffs to balance test accuracy
- Left has smaller training error, but right is more general to test data



Neural Networks

- Use layers of linear and nonlinear functions to model any function
- Multiple layers each with interconnected nodes
- The final weights may have no physical meaning and just approximate the function

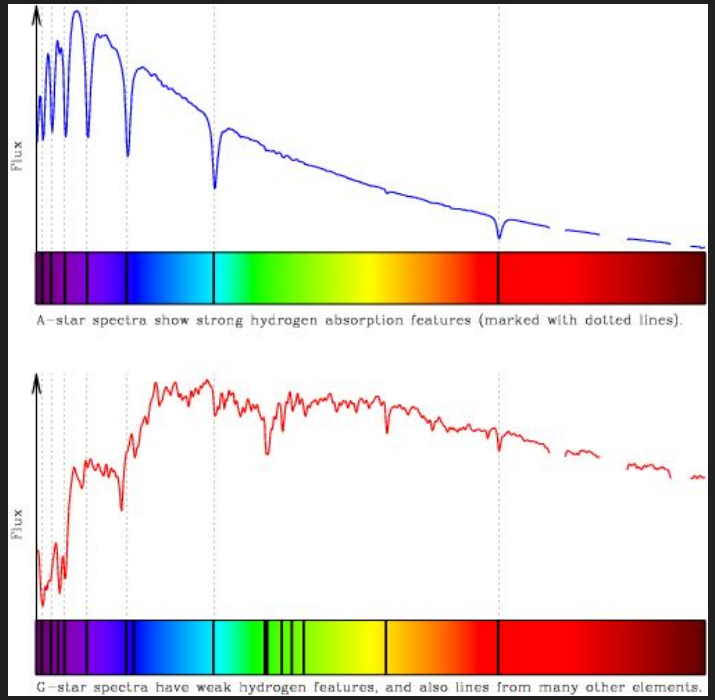


Machine Learning in Astronomy

- Astronomers use machine learning to model and classify data
- Examples:
  - Inferring stellar parameters from spectra (regression)
  - Detecting supernova (classification)
  - Spatial clustering of dark matter halos (clustering)

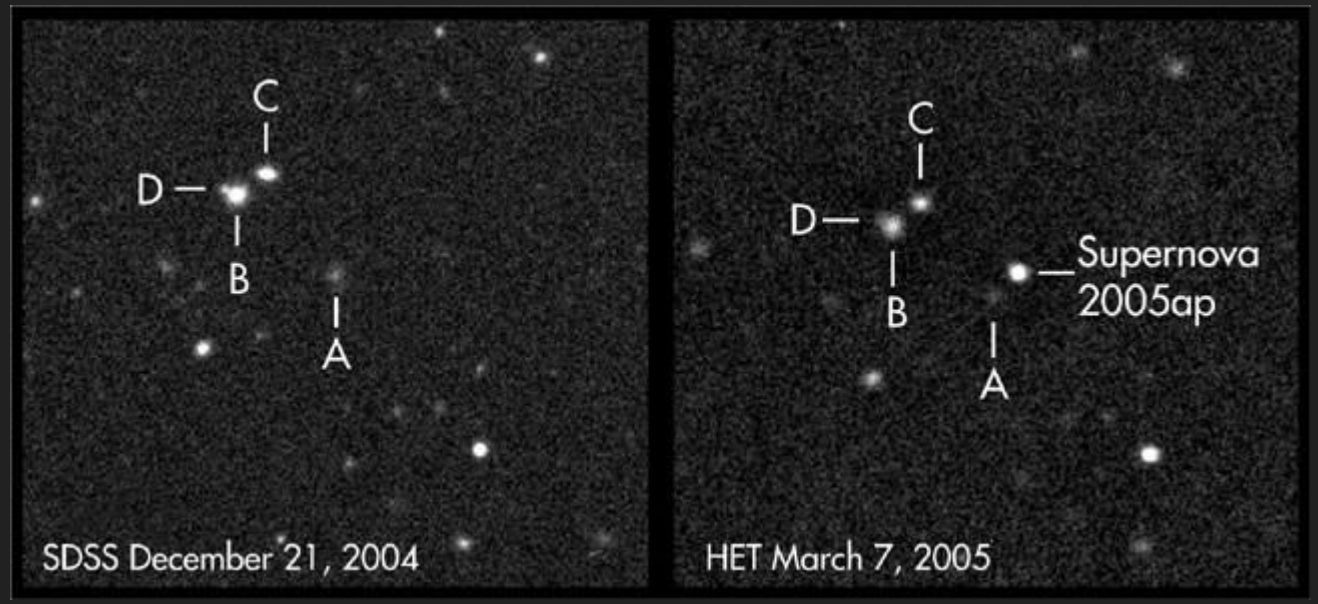
Machine Learning in Astronomy

Inferring stellar parameters from spectra



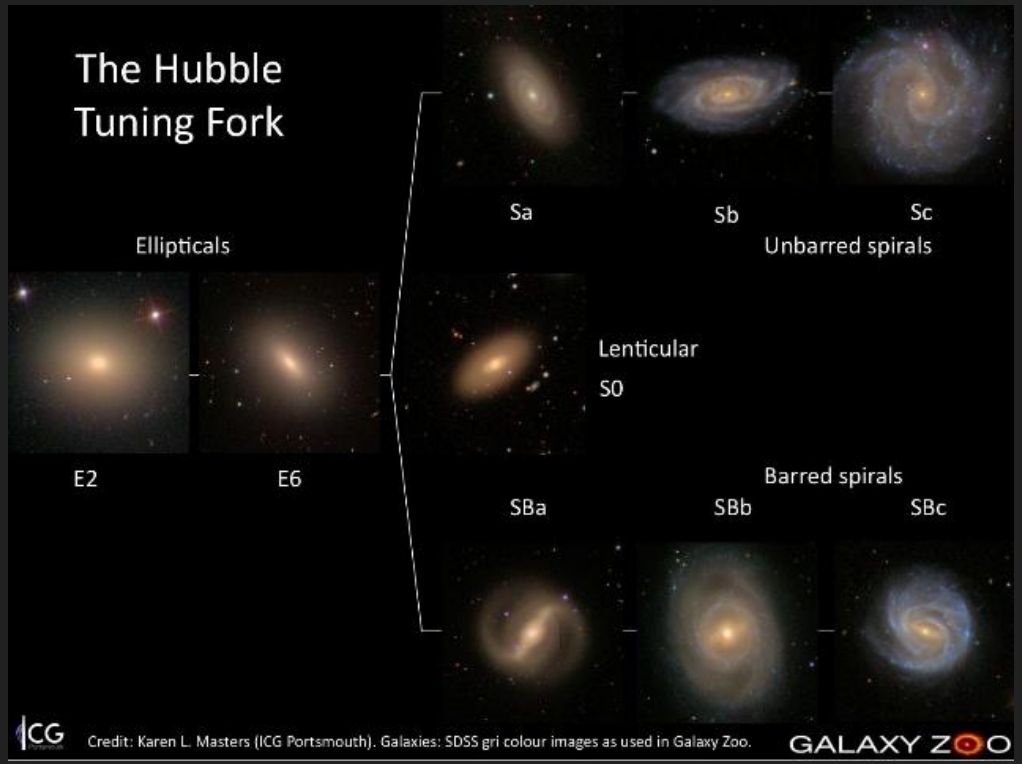
Machine Learning in Astronomy

Detecting supernova



Machine Learning in Astronomy

Galaxy type classification



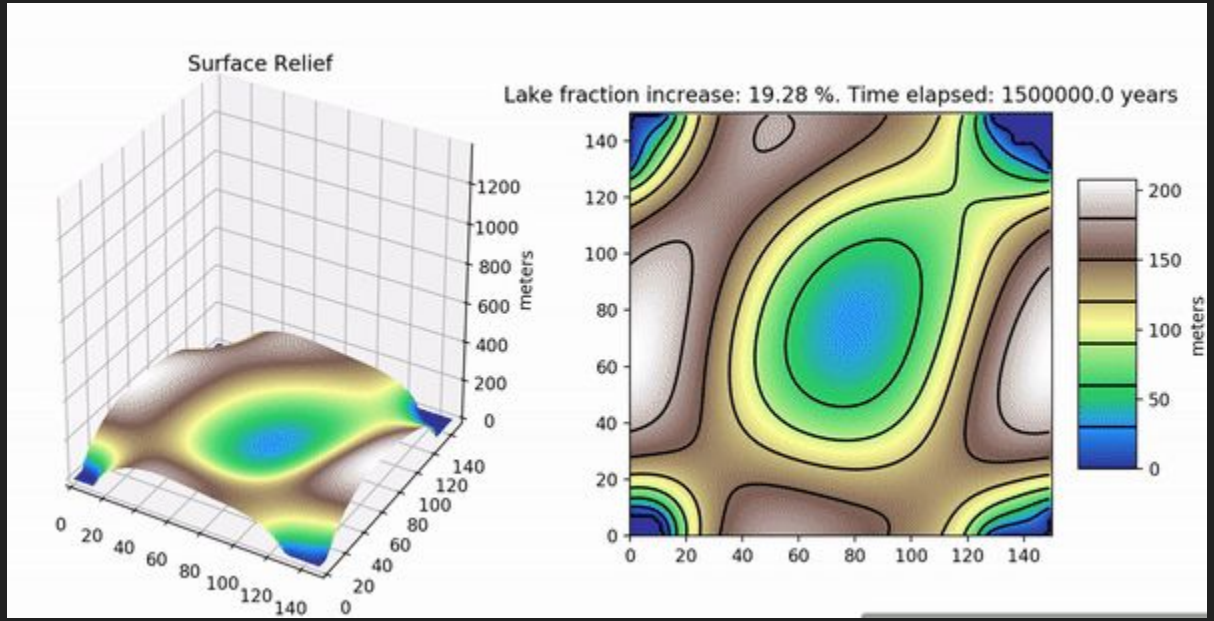
Helpful Links

- https://towardsdatascience.com/5-beginner-friendly-steps-to-learn-machine-learning-and-data-science-with-python-bf69e211ade5
- Check out past lectures and assignments from courses like Data8 (Intro to data science), Data100 (More advanced data science techniques), CS188 (Artificial intelligence), CS189(Machine learning)
  - https://inst.eecs.berkeley.edu/classes-eecs.html

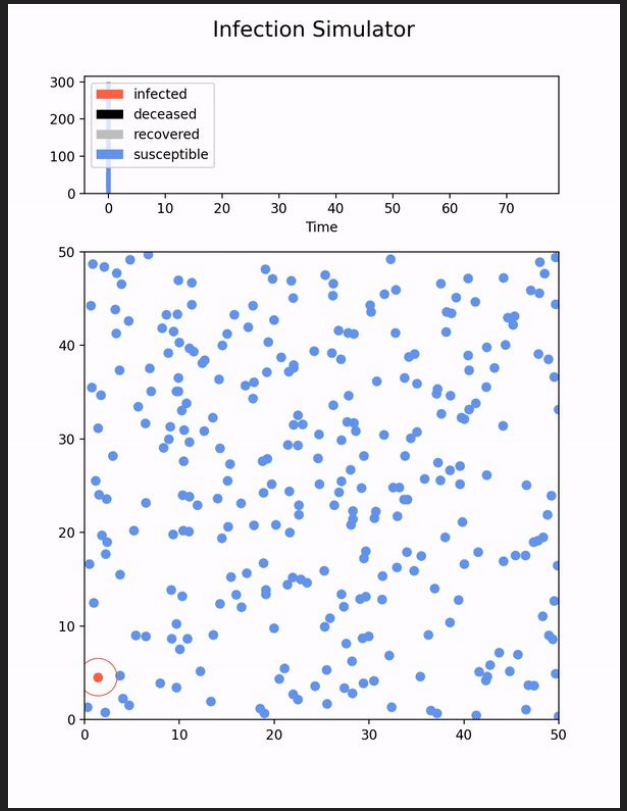
Intro to Animation Examples

Saeed Mohanna's

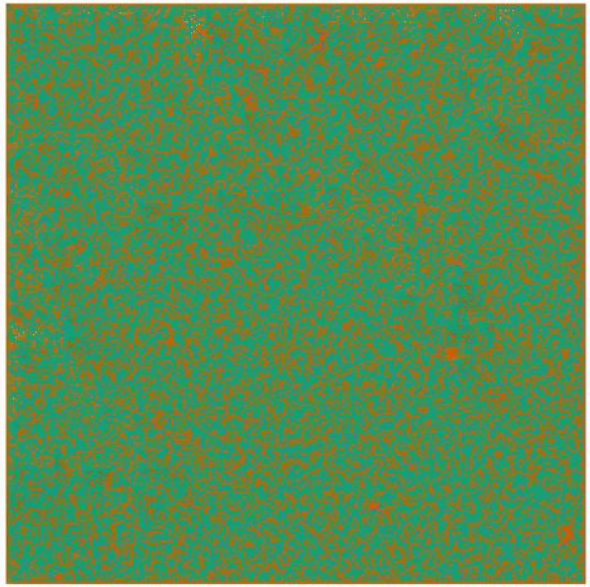
"Erosion o Crater Lake, Oregon"

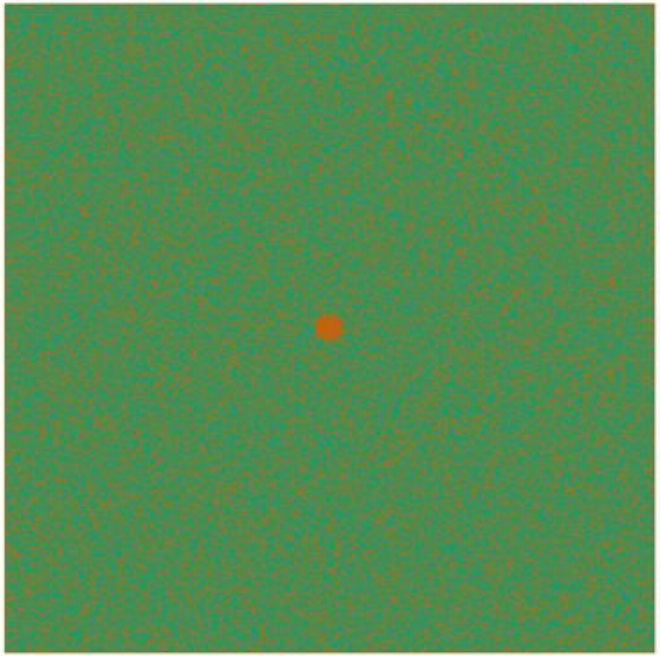


Intro to Animation Examples



Forest Fire Example





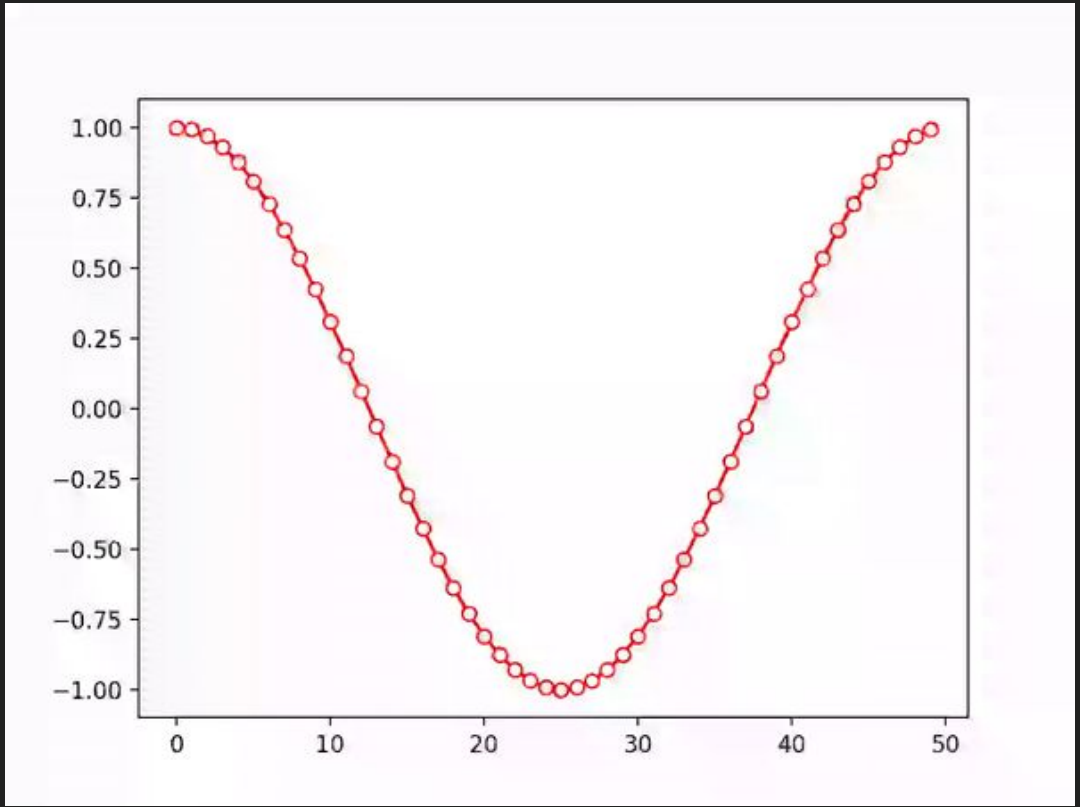
There are lots of different ways to animate!

But we're going to focus on **matplotlib.animation.FFMpegWriter** (documentation can be found here)

matplotlib.animation.FuncAnimation is another common way to animate (documentation can be found here)

If you run into trouble installing ffmpeg, check out the Datahub (already installed): https://datahub.berkeley.edu/

A Moving Sinusoidal Wave (using FFMpeg)



A Moving Sine Wave (FFMpeg)

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.animation import FFMpegWriter
```

Import necessary libraries:

from matplotlib.animation import FFMpegWriter

```
metadata = dict(title='Sinusoidal Wave Animation', artist='Matplotlib')
writer = FFMpegWriter(fps=15, metadata=metadata)
fig = plt.figure()

with writer.saving(fig, 'sinusoidal_wave.mp4', dpi=200):
    nf = 100
    for it in range(nf):
        n = 50
        y = np.zeros(n)
        f = 2.0*np.pi/n
        for i in range(n):
            y[i] = np.cos(f*(i+it)) + np.sin(f*it)*np.cos(3*f*(i+it))
        plt.clf()
        plt.plot(y, 'ro-',mfc='w')
        plt.show()
        plt.draw()
        plt.pause(0.1) # On Macs 0.01 seems fine, on PCs use 0.05 or 0.1
        writer.grab_frame()
```

This is the normal code for plotting sine wave

A Moving Sine Wave (FFMpeg)

```
metadata = dict(title='Sinusoidal Wave Animation', artist='Matplotlib')
writer = FFMpegWriter(fps=15, metadata=metadata)
fig = plt.figure()
```

- Metadata is optional and just fills in file information such as the title, artist, comment, etc. -- note that the title isn't the name of the file
- Initialize the writer -- this is what saves each frame to the mp4 file
  - fps -- frame rate for your mp4
- Initialize the plot figure where the data will be plotted and saved

A Moving Sine Wave (FFMpeg)

```
with writer.saving(fig, 'sinusoidal_wave.mp4', dpi=200):
    nf = 100
    for it in range(nf):
        n = 50
        y = np.zeros(n)
        f = 2.0*np.pi/n
        for i in range(n):
            y[i] = np.cos(f*(i+it)) + np.sin(f*it)*np.cos(3*f*(i+it))
        plt.clf()
        plt.plot(y, 'ro-',mfc='w')
        plt.show()
        plt.draw()
        plt.pause(0.1) # On Macs 0.01 seems fine, on PCs use 0.05 or 0.1
        writer.grab_frame()
```

- with writer.saving() -- saves each paused frame of the figure into the mp4 file name -- the file name must end in '.mp4'!!
- plt.clf() -- clears the frame, plt.draw() -- draws the plot on the figure
- plt.pause() -- pauses the current frame for x seconds
- writer.grab_frame() -- actually saves the frame with the writer

A Moving Sine Wave (using FuncAnimation)

```
fig = plt.figure()
ax = plt.axes(xlim=(0, 3), ylim=(-3, 3))
line, = ax.plot([], [], lw=3)
```

- Initializing figure for our animation

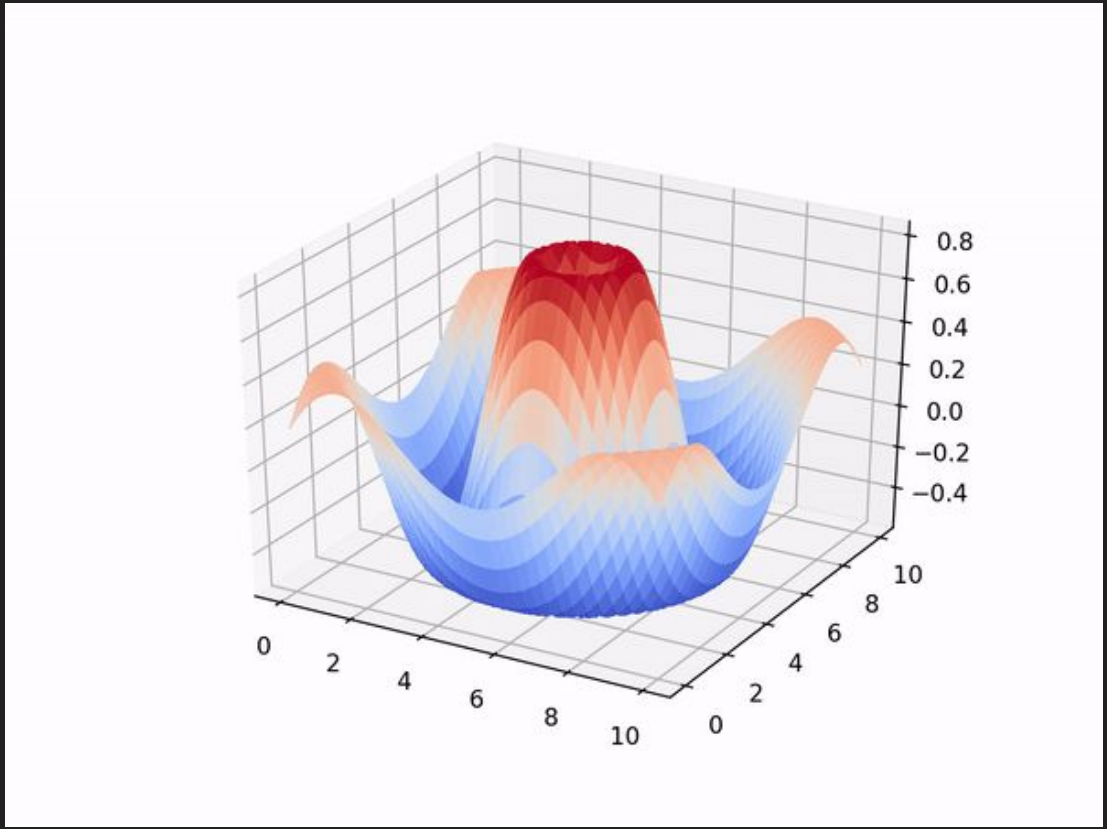
```
# Plots the background of each frame
def init():
    line.set_data([], [])
    return line,

# Animation function
def animate(i):
    x = np.linspace(0, 3, 1000)
    y = 2 * np.sin(5 * np.pi * (x - 0.02 * i))  # try changing this function!
    line.set_data(x, y)
    return line,
```

- define functions for your background and how you want to animate

```
anim = animation.FuncAnimation(fig, animate, init_func=init, frames=200, interval=20, blit=True)
anim.save('sine_wave_animation.mp4', fps=30, extra_args=['-vcodec', 'libx264'])
```

2D Moving Sine Wave (using FFMpeg)



2D Moving Sine Wave (using FFMpeg)

```
# On Macs, use:    %matplotlib osx
# On PCs, use:    %matplotlib qt
%matplotlib osx

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.animation import FFMpegWriter
metadata = dict(title='Sinusoidal wave in 3D space', artist='Matplotlib')
writer = FFMpegWriter(fps=15, metadata=metadata,bitrate=200000)
fig = plt.figure(dpi=200)
```

- %matplotlib osx or %matplotlib qt allows a pop-up window to open to see the animation in real-time
- Import needed libraries including for 3D plots and colormap
- Initialize animation writer and plot figure

2D Moving Sine Wave (using FFMpeg)

```
n=151
L = 10.0
X = np.linspace(0, L, n)
Y = np.linspace(0, L, n)
X, Y = np.meshgrid(X, Y)
Z = np.zeros((n,n))
```

- Initialize the 3D meshgrid using np.linspace()
  - X, Y are the 2D axes and the Z axis is the vertical axis that plots the data values

2D Moving Sine Wave (using FFMpeg)

```
with writer.saving(fig, "wave2d.mp4", dpi=200):
    nf = 100
    for it in range(nf):
        if (it%10==0): print(it,end='')
        print('.',end='')

        f = 2.0*2.0*np.pi/n
        for i in range(n):
            for j in range(n):
                R = np.sqrt( (i-n/2)**2 + (j-n/2)**2 )
                Z[i,j] = np.sin(f*(R+it))*np.exp(-R/100.0)
        fig.clear()
        ax = fig.gca(projection='3d')
        ax.plot_surface(X, Y, Z, cmap=cm.coolwarm, antialiased=False)
        plt.draw()
        plt.pause(0.01)
        writer.grab_frame()
```

- Initialize ax with 3d projection and colormap
  - antialiased=False for faster plotting, though a bit jaggier
- Same format as the 1D wave earlier, except in this case the data is stored in Z, a 2D array instead of a 1D array (use a double for loop to update instead of one for loop)
- Also use ax.plot_surface() instead of plt.plot()

General Setup

```
import matplotlib
from matplotlib.animation import FFMpegWriter
metadata = dict(title='My first animation in 3D', artist='Matplotlib')
writer = FFMpegWriter(fps=15, metadata=metadata,bitrate=200000)
fig = plt.figure(dpi=200)
```

- Import FFMpegWriter and set up metadata, writer, and figure

```
with writer.saving(fig, "fig_name.mp4", dpi=200):
    for it in range(iterations):  # for each iteration it updates

        ## write code that updates your data

        plt.draw()
        plt.pause(0.01)  # runs the frame for 0.01 seconds
        writer.grab_frame()  # this saves the frame
```

- Create for loop that updates your data for the wanted number of iterations

** Be sure to include writer.grab_frame(), plt.pause() or nothing will be saved! **

Tips for Animating

- Basically just saving lots of frames, each is a normal plot (already learned this part!)
- Keep in mind boundary conditions: initialize those and check them at every loop
  - Ex: if one of the edges should always be a certain value
- For differential equations, anything that is dx/dt is increasing by delta x for every delta t step of the for loop
  - You can increment a for loop by a small delta t value like 0.00005
  - Ex: kinematic equations, heat diffusion, etc.
- Google is your friend!
- Check out the Animation Guide Jupyter Notebook on bCourses