Welcome to WOA7015 Advance Machine Learning Lab - Week 2

This code is generated for the purpose of WOA7015 module. The code is available in github https://github.com/shiernee/Advanced_ML

The Gaussian Distribution

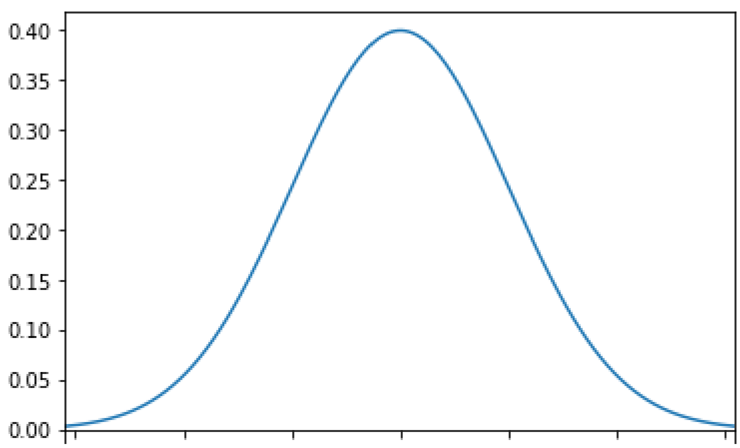
The p.d.f of random variable Z with a gaussian / normal distribution is shown below

$$p(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2}.$$

It is defined for all real values z, from $-\infty$ to ∞.

The distribution looks like this:

```
# import symbulate https://dlsun.github.io/symbulate/index.htm

!pip install -q symbulate
from symbulate import *


Normal().plo
```



Expected Value

The expected value of a standard normal random variable, $E[Z]$, is...

```
Normal().mean()
```

```
0.0
```

Variance

The variance of a standard normal random variable, $\text{Var}[Z]$, is...

```
Normal().va
```

```
1.0
```

The (General) Normal Distribution

The standard normal distribution is centered at 0 with a variance of 1. In general, we can

- scale the bell shape to be as wide as we want,
- shift the bell shape to be centered wherever we want.

If Z is standard normal, then

$$X = \mu + \sigma Z$$

is $\text{Normal}(\mu, \sigma)$. The parameter μ is the expected value, and the parameter σ is the standard deviation. (So σ^2 is the variance.)

Exercise 1

Generate a normal distribution with

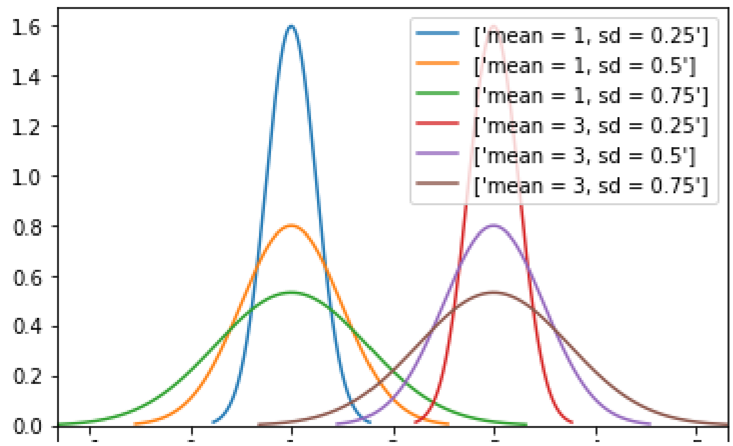
1. mean=1, stdev=0.25
2. mean=1, stdev=0.5
3. mean=1, stdev=0.75
4. mean=3, stdev=0.25
5. mean=3, stdev=0.5
6. mean=3, stdev=0.75

in the same plot with different colors with legends.

```
import matplotlib.pyplot as plt

legends = []
for mean in [1, 3]:
    for std in [0.25, 0.5, 0.75]:
        Normal(mean=mean, sd=std).plot()
        legends.append(["mean = {}, sd = {}".format(mean, std)])
```

```
plt.legend(legends)
```

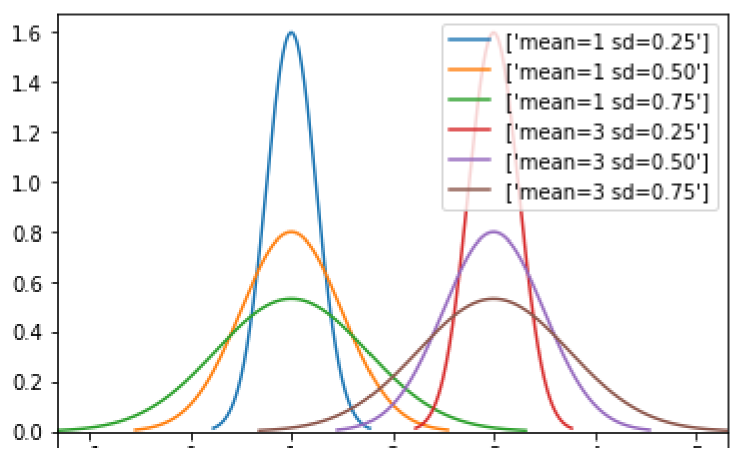
```
<matplotlib.legend.Legend at 0x7f40af59d250>
```



Exercise 1 Solution - Try yourself first.

显示代码

```
<matplotlib.legend.Legend at 0x7f40af59d810>
```



Probability

To calculate probabilities, we integrate the p.d.f. over the relevant region. For example,

$$P(Z \leq 1) = \int_{-\infty}^{1} \frac{1}{\sqrt{2\pi}} e^{-z^2/2}\, dz.$$

Unlike other continuous distributions we have studied, the p.d.f. $p(z)$ has no elementary antiderivative. That means that you will not be able to evaluate this integral by paper and pencil, using techniques you learned in calculus. It has to be evaluated numerically. Fortunately, you can do this easily in Symbulate.

For example, $P(Z \leq 1)$ is just the c.d.f. evaluated at 1. The c.d.f. of the standard normal distribution is often represented by $\Phi(z)$. So we need to calculate $\Phi(1)$.

```
Normal().cdf
```

```
0.8413447460685429
```

Exercise 2:

How would you calculate $P(-2 < Z < 2)$?

```
# YOUR CODE HERE
cdf = Normal().cdf([2]) - Normal().cdf([-2])

print(cdf)
```

```
[0.95449974]
```

Solution 2 - Try yourself first

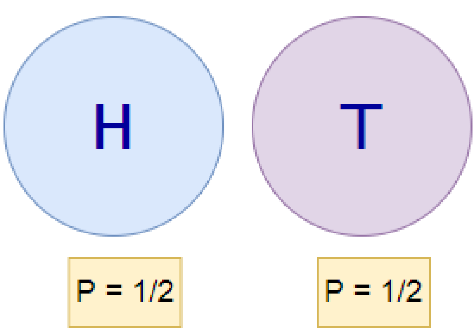
显示代码

```
array([0.95449974])
```

Monte Carlo Approximation

Example 1: Coin Flip Example

The probability of head for a fair coin is 1/2. Monte-Carlo method to simulate the coin-flipping iteratively 5000 times to find out why the probability of a head or tail is always 1/2.



```
# import require libraries
import random
import numpy as np
import matplotlib.pyplot as plt
```

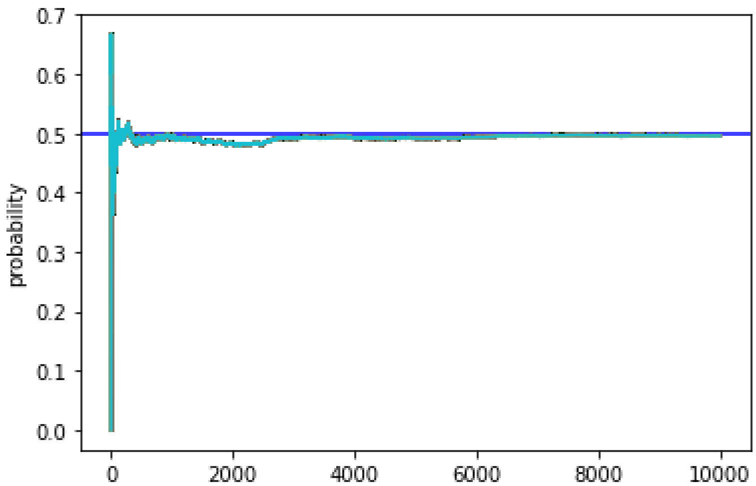
```
# coin flip function:
# 0 --> Head
# 1 --> Tail

def coin_flip():
    return random.randint(0, 1)

# check the output of coin_flip
for i in range(10):
    print('iteration' + str(i) + '--> ' + str(coin_flip()))
```

```
iteration0--> 0
iteration1--> 1
iteration2--> 0
iteration3--> 1
iteration4--> 1
iteration5--> 0
iteration6--> 1
iteration7--> 0
iteration8--> 0
iteration9--> 0
```

```
# Monte Carlo Simulation
list1= []

def monte_carlo(n):
    results = 0
    plt.axhline(y=0.5, color='blue', linestyle='-')

    for i in range(n):
        flip_result = coin_flip()
        results = results + flip_result

        # calculate probabibility valuue
        prob_value = results / (i+1)

        # append probability to list1
        list1.append(prob_value)

        # plot results
        plt.xlabel('iteration')
        plt.ylabel('probability')
        plt.plot(list1)
```

```
    return results / n

# call monte carlo functioiin
answer = monte_carlo(10000)
print('final value of probability: ', answer)
```

```
final value of probability:  0.4957
```



Example 2: Estimating Pi from Circle and Square

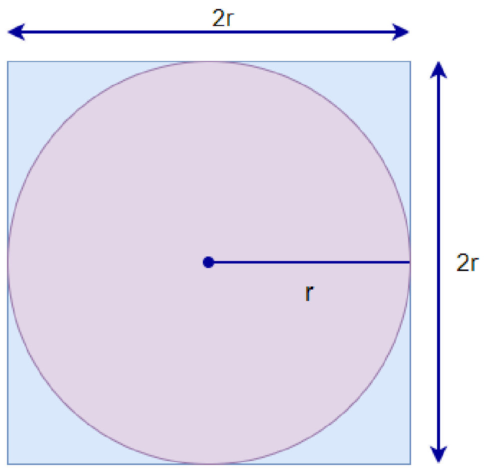
To estimate the value of Pi, we can use the area of circle and square.

$$\frac{Area\ Circle}{Area\ Square} = \frac{\pi * r^2}{2r * 2r}$$

$$\frac{Area\ Circle}{Area\ Square} = \frac{\pi}{4}$$

π value can be estimate using the following formula

$$\pi = 4 * \frac{Area\ Circle}{Area\ Square}$$



Assuming r = 0.5

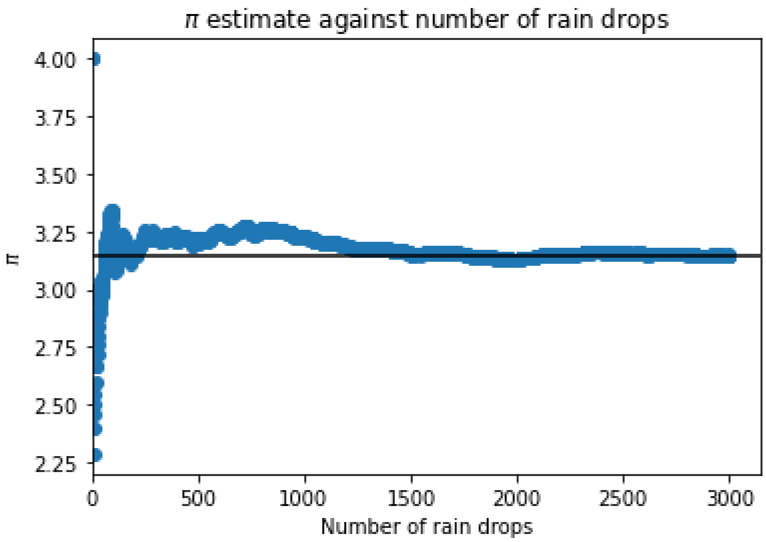
length_of_field = 2r = 1.0

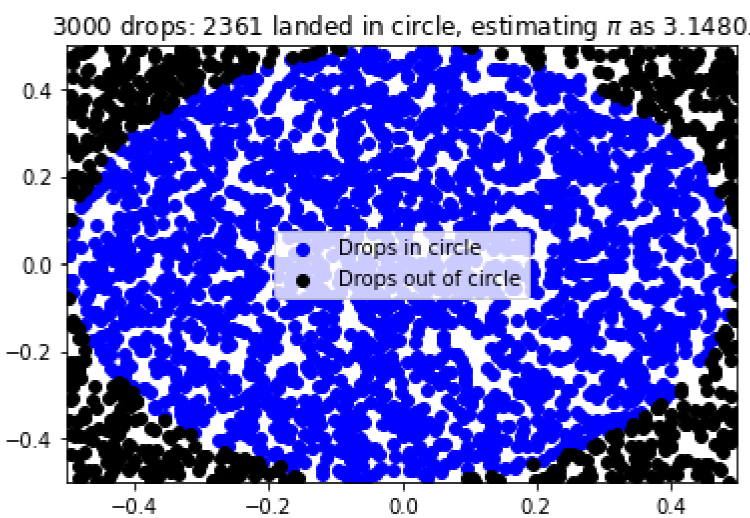
```
import turtle
from random import random
import matplotlib.pyplot as plt
import math


# simulate raindrop
# return x and y coordinates of raindrop

def rain_drop(length_of_field=1):
        """
        Simulate a random rain drop
        """
        return [(.5 - random()) * length_of_field, (.5 - random()) * length_of_field]



# check if raindrop fall in circle by using circle formula

def is_point_in_circle(point, length_of_field=1):
        """
        Return True if point is in inscribed circle
        Use circle formula --> x^2 + y^2 <= r^2
        """
        return (point[0]) ** 2 + (point[1]) ** 2 <= (length_of_field / 2) ** 2


def plot_rain_drops(drops_in_circle, drops_out_of_circle, length_of_field=1, format='pdf'):
        """ Function to draw rain drops """
        number_of_drops_in_circle = len(drops_in_circle)
        number_of_drops_out_of_circle = len(drops_out_of_circle)
        number_of_drops = number_of_drops_in_circle + number_of_drops_out_of_circle
```

```
    plt.figure()
    plt.xlim(-length_of_field / 2, length_of_field / 2)
    plt.ylim(-length_of_field / 2, length_of_field / 2)
    plt.scatter([e[0] for e in drops_in_circle], [e[1] for e in drops_in_circle], color='
    plt.scatter([e[0] for e in drops_out_of_circle], [e[1] for e in drops_out_of_circle],
    plt.legend(loc="center")
    plt.title("%s drops: %s landed in circle, estimating $\pi$ as %.4f." % (number_of_d
    plt.savefig("%s_drops.%s" % (number_of_drops, format))


# simulate raindrop
# return total number of raindrop in circle and in square

def rain(number_of_drops=1000, length_of_field=1, plot=True, format='pdf', dynamic=False):
    """
    Function to make rain drops.
    """
    number_of_drops_in_circle = 0
    drops_in_circle = []
    drops_out_of_circle = []
    pi_estimate = []
    for k in range(number_of_drops):
        d = (rain_drop(length_of_field))
        if is_point_in_circle(d, length_of_field):
            drops_in_circle.append(d)
            number_of_drops_in_circle += 1
        else:
            drops_out_of_circle.append(d)
        if dynamic:  # The dynamic option if set to True will plot every new dr
            print("Plotting drop number: %s" % (k + 1))
            plot_rain_drops(drops_in_circle, drops_out_of_circle, length_of_field, forma
        pi_estimate.append(4 * number_of_drops_in_circle / (k + 1))  # This updates
    # Plot the pi estimates
    plt.figure()
    plt.scatter(range(1, number_of_drops + 1), pi_estimate)
    max_x = plt.xlim()[1]
    plt.hlines(math.pi, 0, max_x, color='black')
    plt.xlim(0, max_x)
    plt.title("$\pi$ estimate against number of rain drops")
    plt.xlabel("Number of rain drops")
    plt.ylabel("$\pi$")
    # plt.savefig("Pi_estimate_for_%s_drops_thrown.pdf" % number_of_drops)

    if plot and not dynamic:
        # If the plot option is passed and matplotlib is installed this plots
        # the final set of drops
        plot_rain_drops(drops_in_circle, drops_out_of_circle, length_of_field, format)

    return [number_of_drops_in_circle, number_of_drops]
```

```
# call the function
number_of_drops = 3000
r = rain(number_of_drops, plot=True, format='png', dynamic=False)

print("----------------------")
print("%s drops" % number_of_drops)
print("pi estimated as: %s " % (4 * r[0] / r[1]))
print("----------------------")
```

```
----------------------
3000 drops
pi estimated as: 3.148
----------------------
```





Now try increasing number_of_drops and check the value of π.

At what value of number_of_drops does the π value approaches 3.14? Write down your answer below.

```
# write your answer here.

### Accoding to the graph, the pi value approaches 3.14 when the value of
### number_of_drops is around 1000, and it gets steady along the line of 3.14
### when the value of number_of_drops is around 2000
```

Let's go back to power point - slide 11

Multivariate Gaussian Distribution

For two continuous random variables, plot type density uses the simulated (x, y) to estimate the joint probability density function and plot it.

Example. Assume

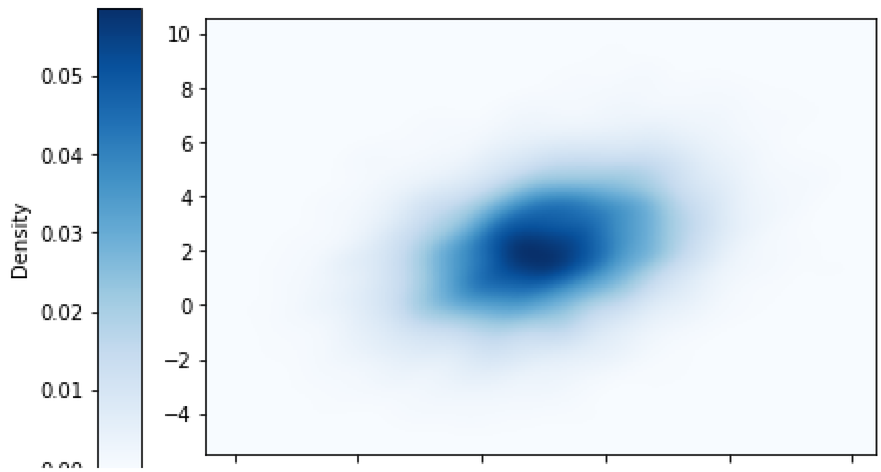
mean of X = 1, mean of Y = 2
variance of X = 2, variance of Y = 4
covariance of xy and yx = 1

```
mu = [1, 2]
Sigma = [[2, 1],
         [1, 4]]

X, Y = RV(MultivariateNormal(mean = mu, cov = Sigma))
Z = X + Y


# understand each output

x = X.sim(10000)
y = Y.sim(10000)
z = Z.sim(10000)
print('X mean:', x.mean())
print('Y mean:', y.mean())
print('Z mean:', z.mean())
print('X variance:', x.sd()**2)
print('Y variance:', y.sd()**2)
print('Z variance:', z.sd()**2)
```

```
X mean: 1.0215924793316042
Y mean: 1.983257519864951
Z mean: 2.990867823165647
X variance: 2.032573629369374
```

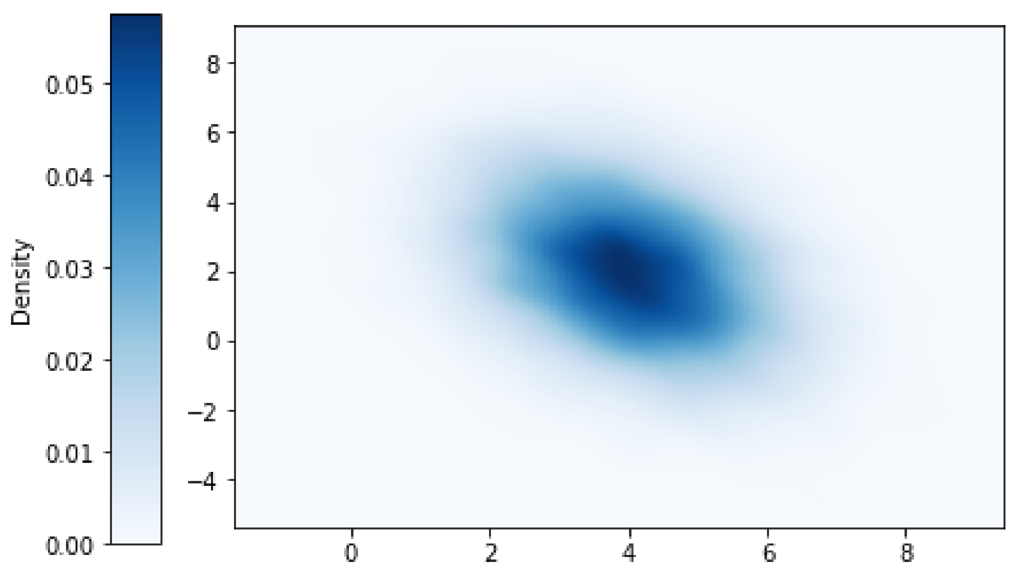
```
Y variance: 4.005193486624156
```

```
(X & Y).sim(10000).plot(type="densit
```

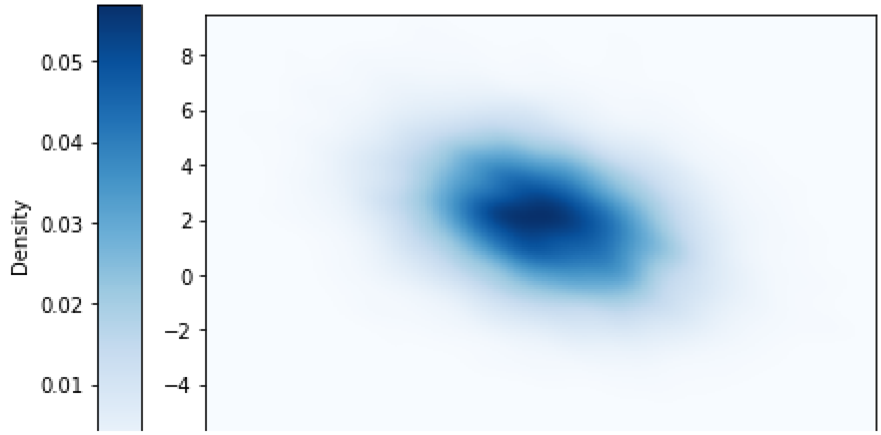


Exercise 3

Generate the Multivariate Gaussian as shown below given variance of X = 2, variance of Y = 4

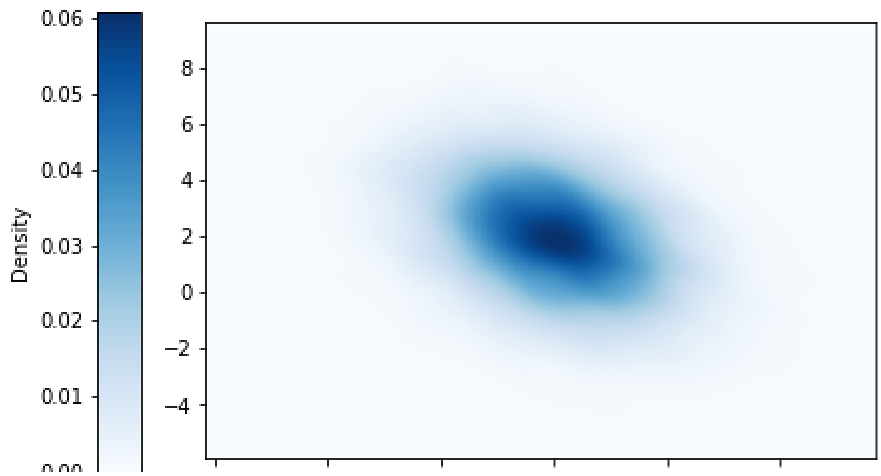



```
# your code here
mu = [4, 2]
Sigma = [[2, -1],
         [-1, 4]]

X, Y = RV(MultivariateNormal(mean = mu, cov = Sigma))
(X & Y).sim(10000).plot(type="density")
```



Solution - Try yourself first

显示代码



Let's go back to power point - slide 20

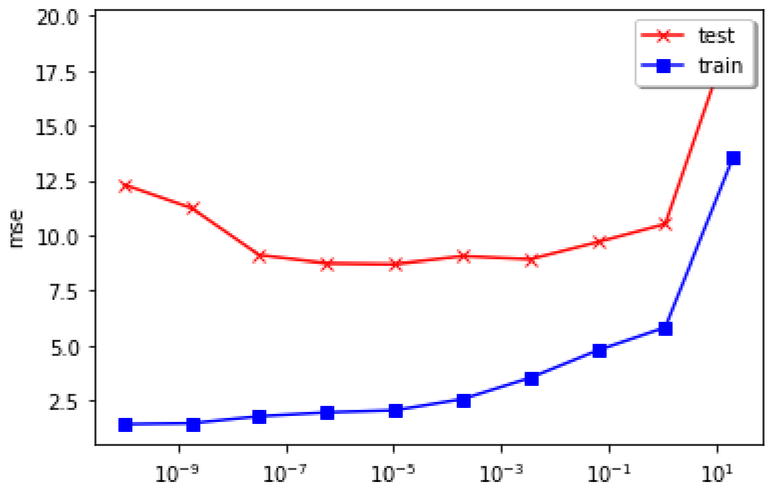
Regularization

Here we examine how regularizer in Ridge regression help in reducing overfitting.

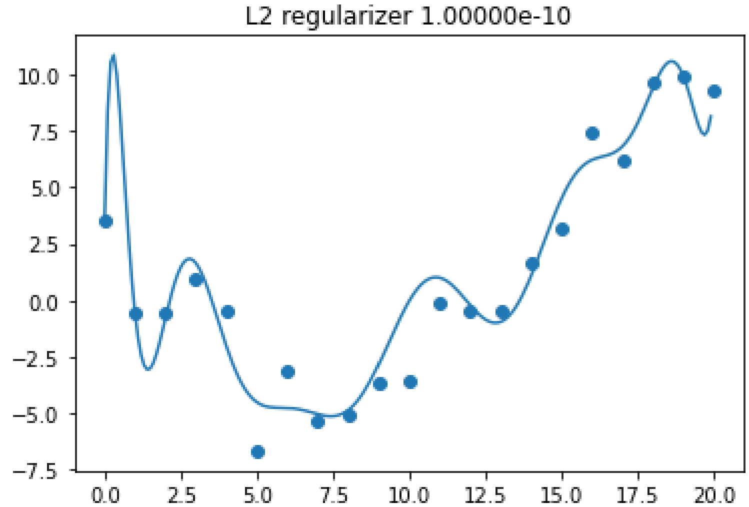
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error as mse
```

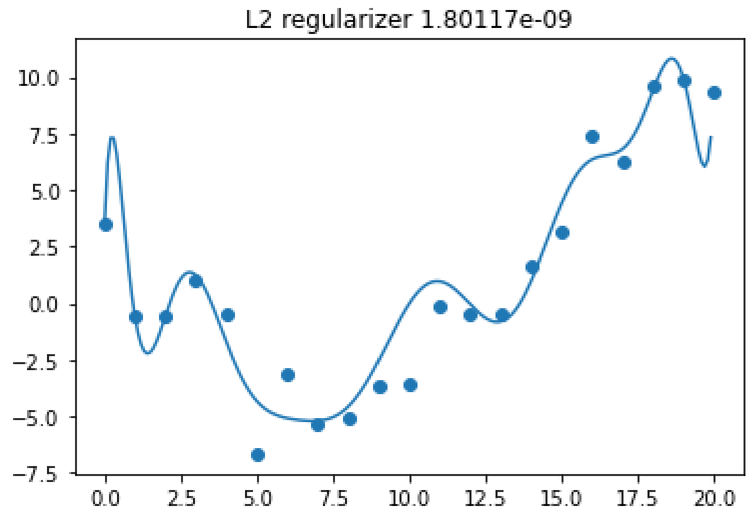
```
# generate 1d regression data
def make_1dregression_data(n=21):
        np.random.seed(0)
        xtrain = np.linspace(0.0, 20, n)
        xtest = np.arange(0.0, 20, 0.1)
        sigma2 = 4
        w = np.array([-1.5, 1/9.])
        fun = lambda x: w[0]*x + w[1]*np.square(x)
        ytrain = fun(xtrain) + np.random.normal(0, 1, xtrain.shape) * \
                np.sqrt(sigma2)
        ytest= fun(xtest) + np.random.normal(0, 1, xtest.shape) * \
                np.sqrt(sigma2)
        return xtrain, ytrain, xtest, ytest



# split data into train and test
xtrain, ytrain, xtest, ytest = make_1dregression_data(n=21)

#Rescaling data
scaler = MinMaxScaler(feature_range=(-1, 1))
Xtrain = scaler.fit_transform(xtrain.reshape(-1, 1))
Xtest = scaler.transform(xtest.reshape(-1, 1))



# fit Ridge model with different regularizer strength
deg = 14
alphas = np.logspace(-10, 1.3, 10)   # Regularization strength
nalphas = len(alphas)
mse_train = np.empty(nalphas)
mse_test = np.empty(nalphas)
ytest_pred_stored = dict()


for i, alpha in enumerate(alphas):
        model = Ridge(alpha=alpha, fit_intercept=False)
        poly_features = PolynomialFeatures(degree=deg, include_bias=False)
        Xtrain_poly = poly_features.fit_transform(Xtrain)
        model.fit(Xtrain_poly, ytrain)
        ytrain_pred = model.predict(Xtrain_poly)
        Xtest_poly = poly_features.transform(Xtest)
        ytest_pred = model.predict(Xtest_poly)
        mse_train[i] = mse(ytrain_pred, ytrain)
        mse_test[i] = mse(ytest_pred, ytest)
        ytest_pred_stored[alpha] = ytest_pred



# Plot MSE vs degree
fig, ax = plt.subplots()
```

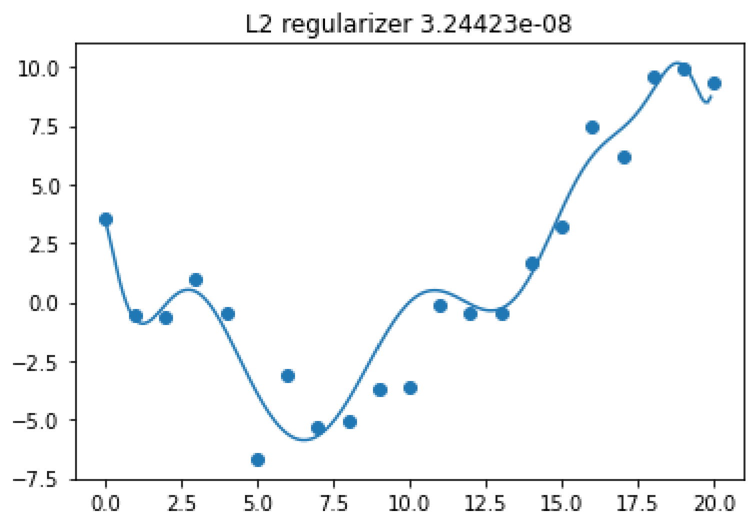
```
mask = [True]*nalphas
ax.plot(alphas[mask], mse_test[mask], color = 'r', marker = 'x',label='test')
ax.plot(alphas[mask], mse_train[mask], color='b', marker = 's', label='train')
ax.set_xscale('log')
ax.legend(loc='upper right', shadow=True)
plt.xlabel('L2 regularizer')
plt.ylabel('mse')
plt.show()
```

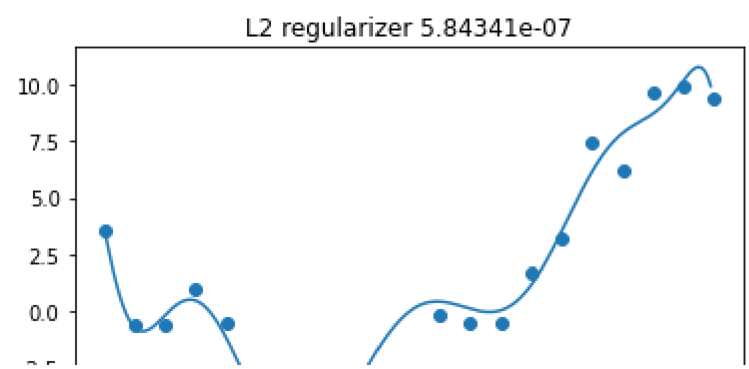


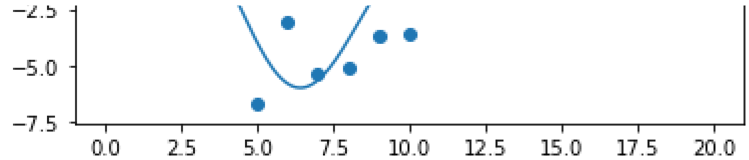

```
# Plot fitted functions
chosen_alphas = alphas[[0,5,8]]
for i, alpha in enumerate(alphas):
    fig, ax = plt.subplots()
    ax.scatter(xtrain, ytrain)
    ax.plot(xtest, ytest_pred_stored[alpha])
    plt.title('L2 regularizer {:0.5e}'.format(alpha))
    plt.show()
```


L2 regularizer 1.00000e-10
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

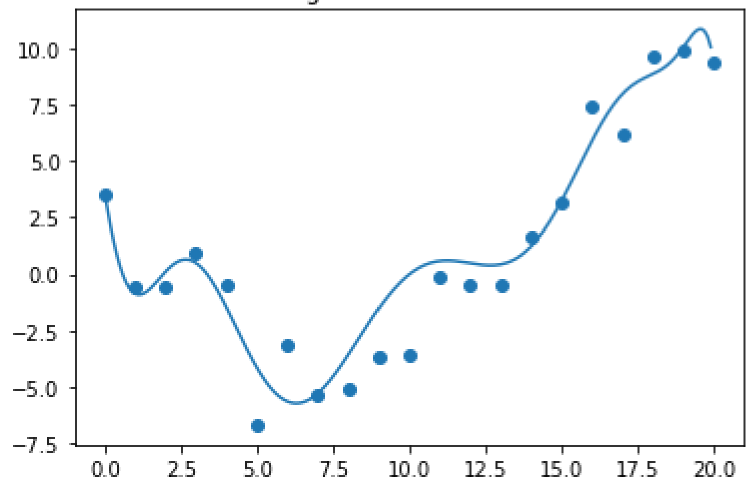

L2 regularizer 1.80117e-09
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


L2 regularizer 3.24423e-08
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

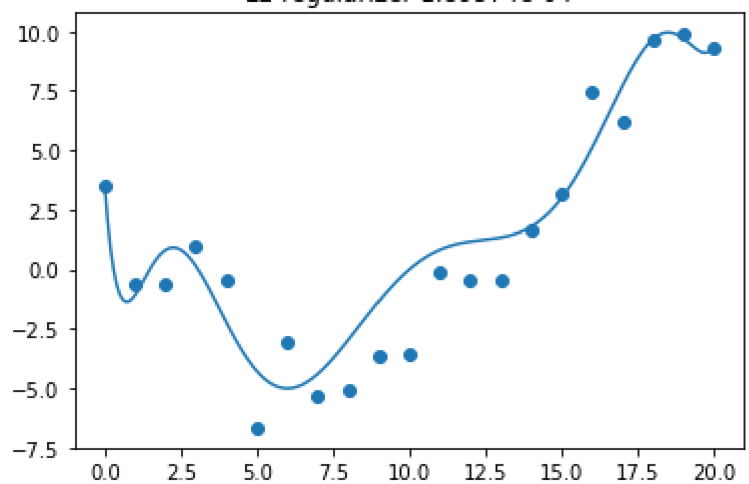

L2 regularizer 5.84341e-07
10.0
7.5
5.0
2.5
0.0



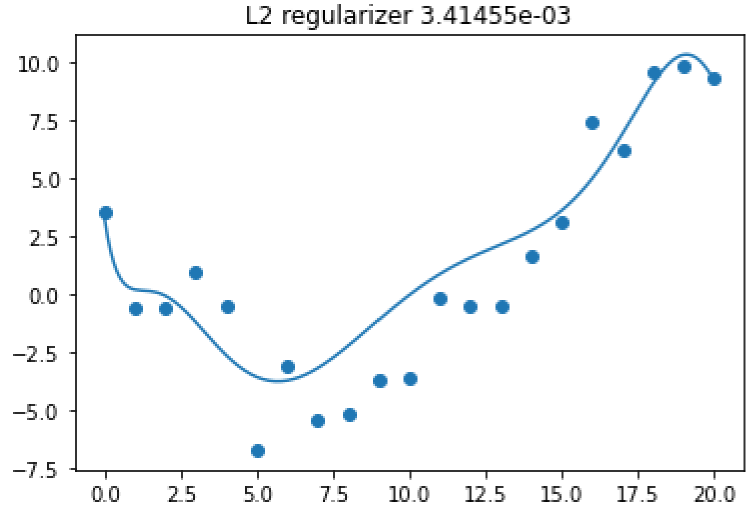
L2 regularizer 1.05250e-05



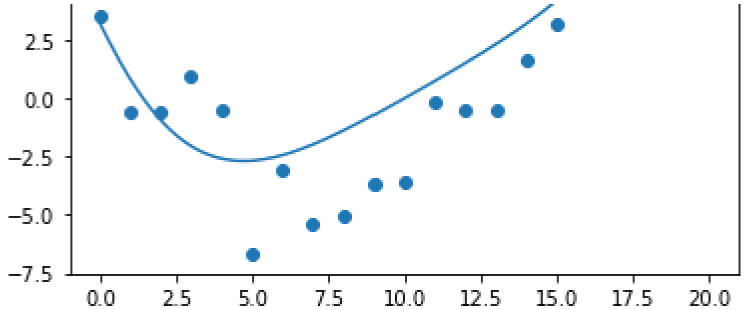
L2 regularizer 1.89574e-04

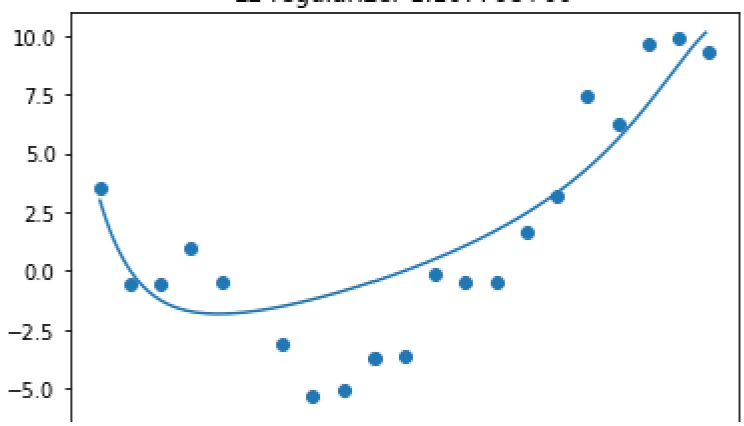


L2 regularizer 3.41455e-03



L2 regularizer 6.15020e-02





Exercise 4:

How do you choose what regularizer strength is optimal?? Explain your answer at the following cell.

```
# your answer here

### Judging from the graphs, the 10^-5 L2 regularizer, which is
### the 1.05250e-05 L2 regularizer is optimal. This is because
### that in the L2 regularizer / mse graph, the test error is
### at the lowest, and the training error is starting to raise
### rapidly with larger regularizer (to the right of the graph).
### And this could be the sign of uderfitting. And if with smaller
### regularizer (to the left of the graph), although the training
### error is very low, but the test error is rather high, and
### this could be the sign of overfitting, which is also shown
### clearly in the following L2 regularizer graphs. Hence, I think
### the 10^-5 L2 regularizer, which is the 1.05250e-05 L2
### regularizer is the optimal one.
```

```
#@title Solution
```

Solution

```
#Use cross validation - explain the process
```

Exercise 5 (10%)

1. Load Iris Data from sklearn. Use the following code to import iris data

```
from sklearn import datasets
iris = datasets.load_iris()
# iris.data = [(Sepal Length, Sepal Width, Petal Length, Petal Width)]
```

2. Use Sepal Length, Sepal Width, Petal Length as X to estimate Petal Width (Y)
3. What is the best regularizer value for Ridge regression model?

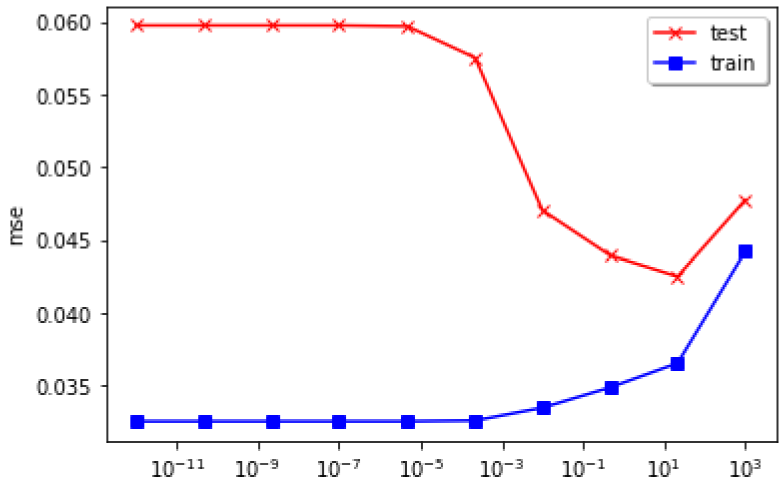
```
# Your code here

from sklearn import datasets
iris = datasets.load_iris()
#iris.data = [("Sepal Length", "Sepal Width", "Petal Length", "Petal Width")]
#iris.data = [("Sepal Length, Sepal Width, Petal Length, Petal Width")]
#iris.data = ["Sepal Length", "Sepal Width", "Petal Length", "Petal Width"]


import numpy as np
import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error as mse
from sklearn import model_selection


cv = model_selection.KFold(n_splits=5, shuffle=Tr


cv_length = int(1/5 * len(iris.data))
iris_train = iris.data[cv_length:, :]
iris_test = iris.data[:cv_length, :]
cvtrain = []
cvtest = []
for cvtrain_ind, cvtest_ind in cv.split(iris_train):
    cvtrain.append(iris_train[cvtrain_ind])
    cvtest.append(iris_train[cvtest_ind])
    #print(cvtrain_ind, cvtest_ind)

cvtrain = np.array(cvtrain)
cvtest = np.array(cvtest)


deg = 3
alphas = np.logspace(-12, 3, 10)   # Regularization strength
nalphas = len(alphas)
mse_train = np.zeros((nalphas))
```

```
mse_test = np.zeros((nalphas))
ytest_pred_stored = dict()

error_list = np.zeros(10)

for i, alpha in enumerate(alphas):
    errorsum = 0
    error_train = 0
    error_test = 0
    for j in range(5):
        Xtrain = cvtrain[j][:, :3]
        Ytrain = cvtrain[j][:, 3:]
        Xtest = cvtest[j][:, :3]
        Ytest = cvtest[j][:, 3:]

        model = Ridge(alpha=alpha, fit_intercept=False)
        poly_features = PolynomialFeatures(degree=deg, include_bias=False)
        Xtrain_poly = poly_features.fit_transform(Xtrain)
        model.fit(Xtrain_poly, Ytrain)
        ytrain_pred = model.predict(Xtrain_poly)
        Xtest_poly = poly_features.transform(Xtest)
        ytest_pred = model.predict(Xtest_poly)
        error_train += mse(ytrain_pred, Ytrain)
        error_test += mse(ytest_pred, Ytest)
        ytest_pred_stored[alpha] = ytest_pred

        errorsum += mse(ytest_pred, Ytest)

    error_list[i] = errorsum / 5
    mse_train[i] = error_train / 5
    mse_test[i] = error_test / 5

error_list = np.array(error_list)



minimum = np.argmin(error_list)
print("The alpha with minimum error is {} with error of: {}.".format(alphas[minimum], error_
error_list[minimum]
```

```
The alpha with minimum error is 21.544346900318867 with error of: 0.042511620260878756.
0.042511620260878756
```

```
# Plot MSE vs degree
fig, ax = plt.subplots()
mask = [True]*nalphas
ax.plot(alphas[mask], mse_test[mask], color = 'r', marker = 'x',label='test')
ax.plot(alphas[mask], mse_train[mask], color='b', marker = 's', label='train')
ax.set_xscale('log')
ax.legend(loc='upper right', shadow=True)
plt.xlabel('L2 regularizer')
```

```
plt.ylabel('mse')
plt.show()
```



Submission Instructions

Once you are finished, follow these steps:

Restart the kernel and re-run this notebook from beginning to end by going to Kernel > Restart Kernel and Run All Cells. If this process stops halfway through, that means there was an error. Correct the error and repeat Step 1 until the notebook runs from beginning to end. Double check that there is a number next to each code cell and that these numbers are in order. Then, submit your lab as follows:

Go to File > Print > Save as PDF. Double check that the entire notebook, from beginning to end, is in this PDF file. Make sure Solution for Exercise 5 are in for marks. Upload the PDF to Spectrum.

Acknowledgement

The works are inspired from

1. Normal Distribtion - https://colab.research.google.com/github/dlsun/Stat350F19/blob/master/Normal_Distribution.ipynb#scrollTo=4K2s06RQFP_1
2. Coin Flip Example - https://pub.towardsai.net/monte-carlo-simulation-an-in-depth-tutorial-with-python-bcf6eb7856c8
3. Estimating π from circle and square = https://www.youtube.com/watch?v=VJTFfIqO4TU

0 秒 完成时间：22:09