# General Instructions for Students

1. This programming assignment focuses on understanding the fundamental concepts of regression.
2. Your notebook must be able to execute completely using the "Run All" command. Any runtime errors in the submitted notebook will result in a score of zero.
3. Ensure all plots are properly labeled. Use supporting plots to explain your results wherever necessary.
4. Report your observations in a markdown cell below the code for each section.
5. The use of Chat-GPT is permitted.
6. Plotting data and predictions is highly encouraged for debugging purposes. However, remove any debugging or exploratory code before submission.
7. Submit your notebook with the filename format: Team_{TeamNumber}.ipynb. The notebook must run seamlessly; otherwise, it will not be evaluated.
8. Submissions must be compiled into a single zip file, which should include the .ipynb notebook file.
9. If you encounter difficulties copying and pasting cells (especially markdown cells) from the provided PDF, you may create your own markdown cells that closely resemble those in the PDF.
10. Ensure a thorough understanding of your solutions, as you will be required to explain your approach to the problems and observations, during your case study presentations. You do not need to submit the PowerPoint presentation, only the notebook. The presentation will occur later, giving you additional preparation time after submitting the notebook.
11. The case study is divided into six parts, each following the marking scheme:
    - **Understanding Error Surfaces: (5 marks)** This section involves generating noisy data based on a height-weight relationship, fitting a linear model using least squares, and analyzing the error surface to estimate and compare model parameters.
    - **Understanding Model Order and Overfitting: (5 marks)** Explore the effects of model complexity and regularization on overfitting by fitting polynomial models of varying degrees, analyzing errors, and examining the impact of regularization on model parameters.
    - **Understanding the Choice of Kernel: (5 marks)** Investigate the effectiveness of different kernels (polynomial, Gaussian, and sigmoidal) in fitting data generated from sinusoidal, triangular, and Gaussian functions, analyzing the fit quality across various model orders.
    - **Understanding Training Parameters: (5 marks)** Learn the use of stochastic gradient descent for weight updates in kernel-based models, examining the impact of step size and batch size on convergence speed.
    - **Understanding Bias-Variance Trade-off: (10 marks)** Generate multiple datasets of noisy sinusoidal data, fit high-order regression models, and analyze the bias-variance trade-off.
    - **Exploring Maximum a Posteriori (MAP) Estimation: (10 marks)** Fit a high-order regression model to noisy sinusoidal data using Bayesian methods, sample from the posterior distribution to analyze curve fits, and evaluate the predictive distribution of targets.

## Part - (1) : Understanding Error Surfaces

Assume that the relationship between human height (in inches) and weight (in pounds) is given by

$$w(h) = 3.86h - 100.42$$

(a) Generate 25 data points based on this relationship. Introduce noise to these data points using a Gaussian distribution with zero mean and a variance of 20, simulating a noisy sensor. Plot these data points on a scatter plot.

(b) Using the noisy data from part (a), we need to find the best-fit line that describes the relationship between height and weight. The line can be represented as $\hat{t} = y(x, w) = w_0 + w_1 x$, where $w_0$ and $w_1$ are the parameters we want to estimate. Use the least squares method to find these parameters. Create and plot a graph showing the error surface $J(w_0, w_1)$, which represents how the error changes with different values of $w_0$ and $w_1$. Identify the point on this surface where the error is minimized.

(c) Calculate the parameters $w_0$ and $w_1$ using the least squares formula:

$$w_{opt} = (X^T X)^{-1} X^T t$$

. Compare these calculated values with the original values you were aiming for.

(d) Write down all your observations and findings from the tasks above.

In [ ]:

```python
# Understanding the error surface
#All imports
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import collections

#########################################
#Generate meaningfull data
#########################################



#########################################
#Plot scatter plot of data
#########################################



#########################################
#Weight estimation through error surface, i.e., empirically locate the minima of error surface
#########################################
#Sample a bunch of w's around w_opt and compute the associated error



#Compute the error
def Error(w,t,x): #inputs : 1)weight 2)data i.e (t,x)
    #Estimate the target

    #Compute and return the error
    return error


#Plot 3D error surface and the corresponding contour plots
#Error surface is a function of w0 and w1



#Locate the minima of the error surface



#########################################
#Least squares approach to estimate the weights
#########################################
#Complete the below linear regression function
def LinearRegression(x,t): #inputs : 1)input data i.e (x). 2)target i.e (t)

    return w_opt


#Estimate optimal weight's using "LinearRegression" function



#Estimate the targets using the input x and the estimated weights



#Plot the estimated line on top of the above scatter plot



#########################################
#Compare the estimated weight's using least squares approach with the error surface approach
```

**Report your observations**

1.

2.

3.

## Part - (2) : Understanding model order and overfitting

(a). Generate 20 data points from $t_n = \sin(2\pi x_n) + e_n$, where $x_n \in [0,1]$ and $e_n \sim N(0, 0.1)$ . Partition these data points into two subsets: a training set and a testing set, each comprising 10 points.

(b). Fit an $M^{th}$ degree polynomial to the training data using least squares approach, i.e.,

$$\hat{t_n} = w_0 + w_1 x + \ldots + w_m x^m + \ldots + w_M x^M$$

Predict the target values in both the training and testing datasets utilizing the estimated parameter vector $w$. Plot the root mean squared error for with each dataset, for M=0,1,...,9. Explain your findings.

(c) Expand the size of the training dataset to 100 points, and repeat (b).

(d) Introduce a $l_2$ regularization term to the objective function in (b) and repeat (b) and (c). Examine the impact of Lagrange multiplier λ on the root mean squared error of the training and testing datasets

(e) Alter the function in (a) to $t_n = 5 + \sin(2\pi x_n) + e_n$ to investigate the impact of regularizing the bias coefficient $w_0$.

(f) Document all your observations.

In [ ]:

```
#Understanding model order and overfitting
########################################
#Generate 20 data points
########################################



#Obtain train and test splits



########################################
#Fitting Mth degree polynomial using least squares approach
########################################
#Complete the function
def PolynomialFit(X_train,Y_train,M,lamda): #(training data, trining targets, Model order, Regularization coefficient)
    #Transform the data using polynomial kernel

    #Find Pseudo inverse solution

    #return the weight vector
    return w_opt

#Complete the function
def PolynomialPred(w_est,X_train,X_test): #(weight,training data, testing data, training targets, testing targets)
    #Estimate the targets for both training and testing data


    #Return training and testing predictions
    return TrainError,TestError

#Complete the function
def PolynomialPred_Error(w_est,X_train,X_test,Y_train,Y_test): #(weight,training data, testing data, training targets, testing targets)
    #Estimate the targets for both training and testing data


    #Return training and testing error
    return TrainError,TestError
```

```
#Iterate through range of M values
M_range=list(range(10))
TrError = []
TeError = []
for M in M_range:
    #Fit Mth order polynomial i.e estimate optimal w. Use the function "PolynomialFit"


    #Predict training and testing targets


    #Predict errors on both training and testing data using estimated w. Use the function "PolynomialPred_Error"


    #Store them for plotting


#Plot training and testing estimates alogwith the original targets



#Plot training error vs polynomial order, and testing error vs polynomial order



########################################
#Increase the size of training data set to 100 points and repeat the experiments
########################################





########################################
#Effect of regularization
########################################
#Consider a set of lambda's. For example: lamdas = [0, 1e-7 , 1e-4, 1e-2, 1]
#Repeat the experiments, i.e., plot the prediction and error in predictions with respect to model order. Contrast these results with those obtained without regularization.





########################################
#Effect of bias regularization
########################################
#Modify the function i.e include bias
#Generate data



#Estimate the polynomial with and without regularization constraint



#Compare the two estimated polynomials and report the observations
```

**Report your observations**

1.

2.

3.

## Part - (3) : Understanding the choice of kernel

(a). Generate 100 data points from $t_n = \sin(2\pi x_n) + e_n$, where $x_n \in [0, 1]$ and $e_n \sim N(0, 0.1)$. Divide these data points into two sets: a training set and a testing set, each containing 50 points. Fit an $M^{th}$ degree polynomial using polynomial,Gaussian and sigmoidal kernels, and evaluate the goodness of fit for each kernel across different model orders M

(b). Repeat the process described in part (a), but modify the target function as follows:

$$t_n = \begin{cases} \text{sinusoid} + e_n, & \text{where } x \in [0, 1) \\ \text{triangle} + e_n, & \text{where } x \in [1, 2) \\ \text{Gaussian} + e_n, & \text{where } x \in [2, 3) \end{cases}$$

Clearly discuss your observations/results for each of the three kernels.

(c). Summarize and report all your observations from parts (a) and (b), highlighting the differences in performance and fit quality for each kernel.

In [ ]:

```
#Understanding the choice of kernel
########################################
#Generate 100 data points
########################################




#Obtian train and test splits
#Take even samples for training and odd samples for testing




#Function to estimate the parameters
def KernelRegressionFit(X_train,Y_train,kernelType,M,lamda): #(training data, training targets, type of kernel, regularization coefficient)
    #kernelType : {Polynomial,Gaussian,Sigmoid}
    #print(X_train.shape)
    #X_train = np.reshape(X_train, (1,-1))
    #Y_train = np.reshape(Y_train, (1,-1))
    X = []
    if kernelType=='polynomial':
      #Use polynomial kernel to transform the data

    if kernelType=='gaussian':
      #Use Gaussian kernel to transform the data

    if kernelType=='sigmoidal':
      #Use Sigmoid kernel to transform the data

    #Estimate weights using Pseudo iverse solution


    #Return the estimated weights
    return w_opt

#Function to compute the training and testing errors from the current weight estimates
def KernelRegressionPred_Error(w_est,X_train,Y_train,X_test,Y_test,kernelType):
    #kernelType : {Polynomial,Gaussian,Sigmoid}
    X_tr = []
    X_te = []
    #X_train = np.reshape(X_train, (1,-1))
    #Y_train = np.reshape(Y_train, (1,-1))
    #X_test = np.reshape(X_test, (1,-1))
    #Y_test = np.reshape(Y_test, (1,-1))
    M = len(w_est)-1
    if kernelType=='polynomial':
      #Use polynomial kernel to transform the data

    if kernelType=='gaussian':
      #Use Gaussian kernel to transform the data

    if kernelType=='sigmoidal':
      #Use Sigmoid kernel to transform the data
    #Estimate training and testing targets


    #Compute and return the training and testing errors
    return TrainError,TestError


#Iterate through range of M values
M_range=list(range(10))

polynomial_tr_error = []
polynomial_te_error = []
gaussian_tr_error = []
gaussian_te_error = []
sigmoid_tr_error = []
sigmoid_te_error = []

for M in M_range:
    #Fit Mth order polynomial using three kernels i.e {Polynomial,Gaussian,Sigmoid}



    #Predict training and testing targets using estimated w


    #Predict errors on both training and testing data using estimated w



    #Store them for plotting


#Plot the predicted training and testing targets alongside the original targets for various model orders and all three different kernels.




#Plot training error vs polynomial order and testing error vs polynomial order for all the three different kernels




########################################
#Repeat the experiments by changing target function
########################################
```

**Report your observations**

1.

2.

3.

## Part - (4) : Understanding training Parameters

(a). Repeat 3(a) and 3(b) using stochastic gradient descent for weight update.Study the effect of step size η on convergence of the weights, and compare them to those obtained using closed form expressions in 3. Plot the mse as a function of iterations.

- Step Size (η): Investigate how different values of the step size (learning rate) affect the convergence of the weights. The step size is a crucial hyperparameter in SGD that determines the magnitude of weight updates. Experiment with various step sizes to observe their impact on the speed and stability of convergence.
- Comparison with Closed Form Solutions: Compare the weights obtained using SGD with those derived from closed-form expressions in Part 3. Closed-form solutions provide exact answers, while SGD offers iterative approximations. Analyze the differences in convergence behavior and final weight values.
- Plotting MSE: Plot the mean squared error (MSE) as a function of iterations to visualize the convergence process.

(b). Study the effect of batch size on the speed of convergence

- Batch Size Variations: Experiment with different batch sizes, such as small (mini-batch), medium, and large (full-batch), to observe their impact on convergence speed. S
- Convergence Speed: Analyze how batch size influences the number of iterations required for the model to converge to an optimal solution.

(c). Report all your observations. Also suggest on how to choose step size and batch size for a new problem

In [ ]:
```
def ErrorPred(w_est,X_train,Y_train,X_test,Y_test,kernelType): #(estimated weight, training data, training targets, testing data, testing targets, type of the kernel )
    #kernelType : {Polynomial,Gaussian,Sigmoid}
    X_tr = []
    X_te = []
    M = len(w_est)-1
    if kernelType=='polynomial':
        #Use polynomial kernel to transform the data


    if kernelType=='gaussian':
        #Use Gaussian kernel to transofrm the data


    if kernelType=='sigmoidal':
        #Use sigmoidal kernel to transform the data


    #Compute and return the train and test errors
    return TrainError, TestError


def Training(X_train,Y_train,X_test, Y_test, kernelType,M,Epochs,BatchSize,stepSize):
    #(training data, training targets, testing data, testing targets, type of the kernel, order of the mode, Number of epochs, Batch size, Step size)
    #kernelType : {Polynomial,Gaussian,Sigmoid}
    #Initialize the weights


    #Initialize the necessary variables
    #Iterate through epochs

    epochs = range(Epochs)
    for epoch in epochs:
        #Compute the train and test errors using the current weights
        tr_err, te_err = ErrorPred(weights, X_train, Y_train, X_test, Y_test, kernelType)
        #Store training and testing errors for plotting

        #Shuffle the data

        #Iterate through the batches
        for batch in range(batches):
            #Initialize the necessary variables
            #Get a batch of data

            #Iterate through the data points of obtained batch
            for n in range(len(data_b)):
                #Obtain kernel representation
                X_tr = []
                if kernelType=='polynomial':

                if kernelType=='gaussian':

                if kernelType=='sigmoidal':

                #Compute the gradient of weight's

                #Compute the running mean of the weights gradients for the batch update

            #Update the weights using mean gradient, consider using reasonable stepSize
```

```python
    #Plot training and testing error across the epochs


    #Return the estimated weights
    return weights

def Pred(w_est,X_train,X_test,kernelType): #(estimated weights, training data, testing data, type of the kernel )
    #kernelType : {Polynomial,Gaussian,Sigmoid}
    #Initialize the required variables

    #Obtain kernel representations
    if kernelType=='polynomial':

    if kernelType=='gaussian':

    if kernelType=='sigmoidal':

    #Compute and return the training and testing target estimates
    return Y_tr_error,Y_te_error

def Pred_Error(w_est,X_train,Y_train,X_test,Y_test,kernelType): #(estimated weights, training data, training targets, testing data, testing targets, type of the kernel )
    #kernelType : {Polynomial,Gaussian,Sigmoid}
    #Initialize the required variables

    #Obtain kernel representations
    if kernelType=='polynomial':

    if kernelType=='gaussian':

    if kernelType=='sigmoidal':

    #Compute and return the training and testing errors
    return Y_tr_error,Y_te_error


##################################################
#Repeat 3(a) and 3(b) using stochastic gradient descent for weight update.
##################################################




##################################################
#Repeat 3(a) and 3(b) using stochastic gradient descent for weight update.
##################################################




########################################
#Study the effect of stepSize on the convergence of weights ( plot required results )
########################################




########################################
#Study the effect of batchsize on the speed of convergence ( plot required results )
########################################
```

**Report your observations**

1.

2.

3.

## Part 5: Understanding bias-variance trade-off

(a). Generate L=100 datasets of noisy sinusoidal data, each having N=25 datapoints.

(b). For each dataset, fit a $M = 25^{\text{th}}$ order linear regression model incoporating 24 Gaussian basis functions and one bias parameter.

(c). Use regularized least squares, controlled by the parameter λ, to estimate the model parameters $w$.

(d). Illustrate the concept of bias and variance using these 100 different parameter fits.

(e) Chose three different regularization coefficeints (low,middle and high)

(f) For every regularization coefficient, produce two plots: one displaying 100 estimated curves, and the other showing the mean of the estimated curves alongside the original function.

(g) For three regularization coefficients, you should have a total of six plots, meaning two plots for each regularization.

(h) Using the six plots above, describe the bias-variance trade-off.

(i). Report all your observations

In [3]:

```
#Understanding the bias-variance trade-off
########################################
#Generate 100 data sets of noisy sinusoidal data
########################################




########################################
#Use regularized least squares to estimate w
########################################




########################################
#Illustrate the concept of Bias-Variance trade off
########################################
#1. Chose three different regularization coefficeints (low,middle and high)
#2. For every regularization coefficient, produce two plots: one displaying 100 estimated curves, and the other showing the mean of the estimated curves alongside the original function.
#3. For three regularization coefficients, you should have a total of six plots, meaning two plots for each regularization.
#4. Using the six plots above, describe the bias-variance trade-off.
```

**Report your observations**

1.

2.

3.

## Part 6: Exploring Maximum a Posteriori (MAP) Estimation

(a) Create 100 noisy data points based on a sinusoidal function. Fit a 20th-order linear regression model using Gaussian basis functions. Begin with a standard normal prior and update the posterior density statistics of the parameters through Bayesian sequential updates.

(b) Draw a parameter vector from the posterior distribution and generate a curve fit for this sample. Repeat this process multiple times, calculate the average of these curve fits, and compare it to the original sinusoidal function.

(c) Utilize the posterior distribution of the parameters to assess the predictive distribution of the target

$$p(t_0/x_0, X, t)$$

. Plot this distribution for varying numbers of training data points.

(d) Report all your observations.

In [3]:
```
#Understanding MAP estimate
########################################
#Generate 100 data sets of noisy sinusoidal data
# y = sin(2*pi*x) + e where x in (0,1) and e ~ N(0,0.1)
########################################



########################################
#Update the statistics of posterior density
########################################
#Initialie the parameters for standard normal prior



#Iterate through the data points and update the stats of posterior density, Such that PDF is maximized.




########################################
#Sample weight vector from posterior distribution. Estimate the curve, repeat the procedure for 100 times and get the avg fit
########################################





########################################
#Predictive distribution analysis
########################################
#Predictive distribution analysis through sampling
#Iterate through data points and sample weight vectors when partial data points are seen (10,25,50,75,100).





#Predictive distribution analysis through variance
#Iterate through data points and obtain necessary plots[Average fit plot and Plots of distribution at 10,25,50,75 and 100
```

**Report your observations**

1.

2.

3.