EE5907/EE5027: CA1

Deadline: October 4, 2023 at 5pm

Read carefully: This is an individual assignment. Academic integrity must be stricly followed. Copying from other's code/text or from anyone or any source is not allowed. Exchanging codes is not allowed. Software will be used to detect any form of source code plagiarism. Your submitted code must be grouped/separated into the same parts in the instructions. In your submission, you must provide us with all necessary libraries. You are not allowed to use any toolboxes (if you are not sure, you should ask). If you use separate files in your submission, you must zip them to one file. The submission must not be in separate files in different times (it must be submitted together). The deadline is a strict deadline, so please prepare and plan early and carefully. **Late submission will be deducted 10 points (out of 100) for every 24 hours. Zero marks for late submissions of more than 5 days.**

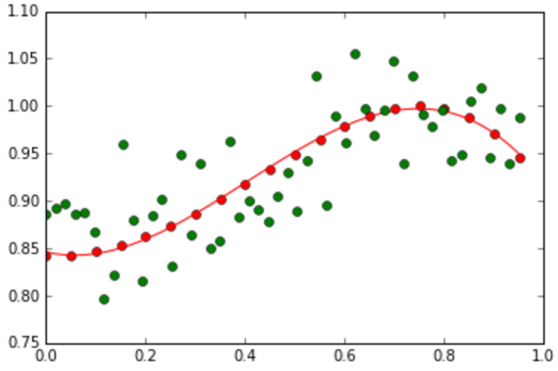
Programming language: Python version 3.
Submission: Jupyter Notebook (any other format won't be accepted). In your Jupyter notebook file, you must compile the codes and show the results there in the file.

Frequently Asked Questions (FAQs)

Part 1: MAP

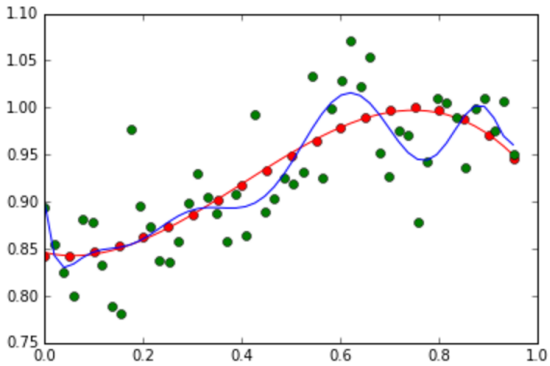
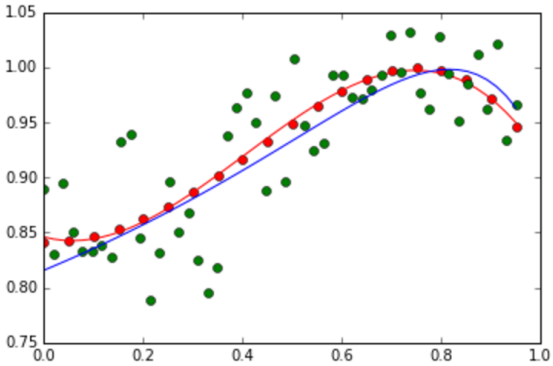
Write a program in python to implement the MAP (or regularization) for polynomial curve fitting problem. Follow the instructions below:

1. Generate 50 2D-data points using the following function: $y = \sin(x^2 + 1)$
2. Add Gaussian random noise to the data
3. Show the original curve line and the noisy data.



The red line is the original curve based on the equation. The green dots are the noisy data.

4. Fit the generated noisy data using the MAP as discussed in class.
5. Compute and display the total absolute error value (between the predicted and the correct ones) of using the computed w.
6. Display the estimated values of w
7. Experiment with your code by changing M and alpha (the coefficient of the regularization/prior term) to various values, and then show the plots. On each the plot, you must show the values of M and alpha.

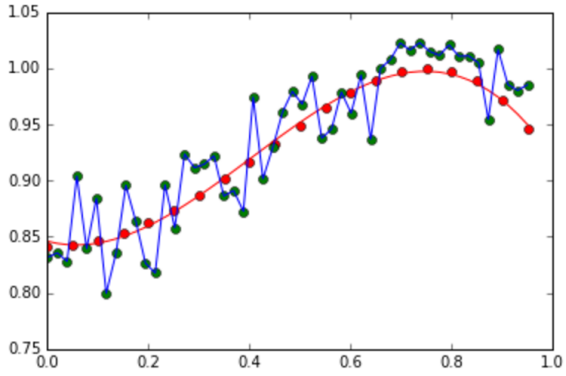
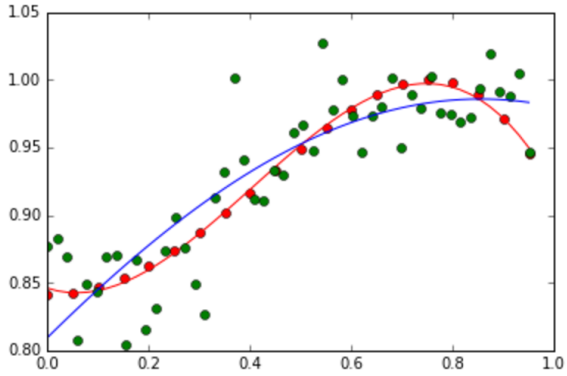
Left: An example of the overfitting problem when M = 10. Right: An example of how the regularization term reduces the overfitting problem (M =10, alpha = 0.4).

8. From the experiment in #7, discuss how M and alpha influence on the fitting accuracy.

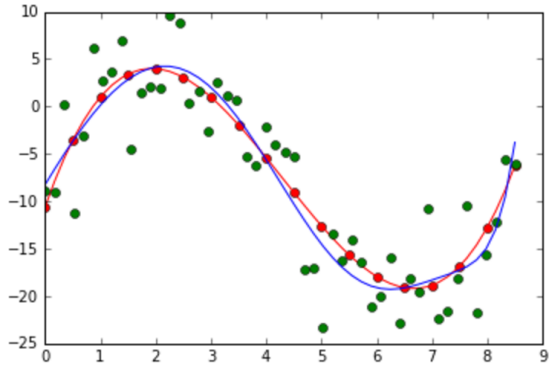
Part 2: BASIS FUNCTION

Write a program in python to implement the MLE that employs basis functions to solve the polynomial curve fitting problem. Follow the instructions below:

1. Generate 50 2D-data points using the following function: $y = \sin(x^2 + 1)$
2. Add Gaussian random noise to the data
3. Fit the generated noisy data using the MLE that employs the **Gaussian basis functions** as discussed in class.
4. Show your results for different values of M that generate overfitting and underfitting curves.

5. Change the basis functions to the **sigmoid basis functions**, and show the results for different values of M that generate overfitting and underfitting curves.
6. Change the original curve function to $y = 0.4345x^3 - 5.607x^2 + 16.78x - 10.61$, and use the sigmoid basis function to estimate the best curve fitting from the noisy data.

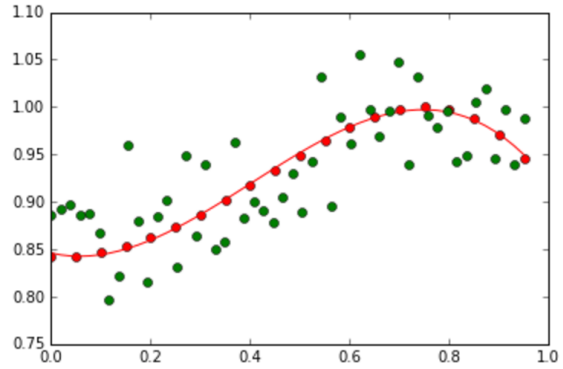


7. Experiment with different parameters of the Gaussian and sigmoid basis functions, and then show the plot. Also, discuss the advantages of these basis functions over polynomial functions.

Part 3: FULL BAYESIAN + PREDICTIVE DISTRIBUTION

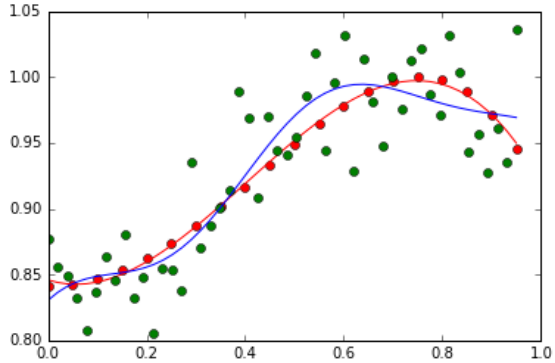
Write a program in python to implement the full Bayesian inference on Gaussian variables for curve fitting problem. Follow the instructions below:

1. Generate 50 2D-data points using the following function: $y = \sin(x^2 + 1)$. Add Gaussian random noise to the data. Show the original curve line and the noisy data.

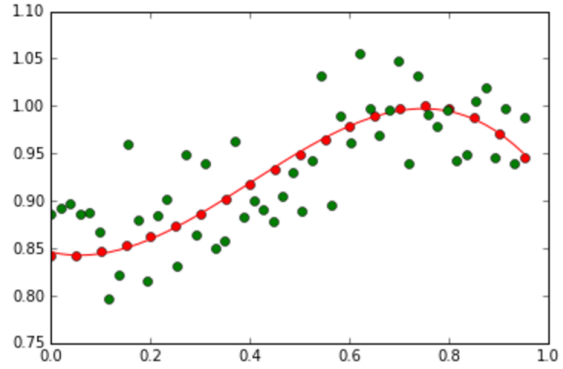


The red line is the original curve based on the equation. The green dots are the noisy data.

2. Compute w based on the full Bayesian inference (by using basis functions like discussed in class). Display the estimated values of w.
3. Experiment with your code by changing alpha and beta. Discuss the meaning of them with respect to the curve fitting results.
4. Show your best fitting, similar to:

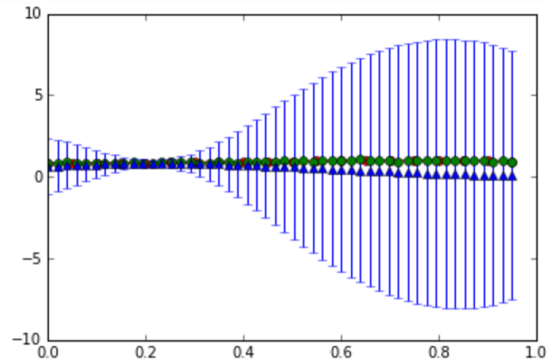


5. Explain how useful p(w|t) for the training and testing stages.
6. Generate 50 2D-data points using the following function: $y = \sin(x^2 + 1)$. Add Gaussian random noise to the data. Show the original curve line and the noisy data.

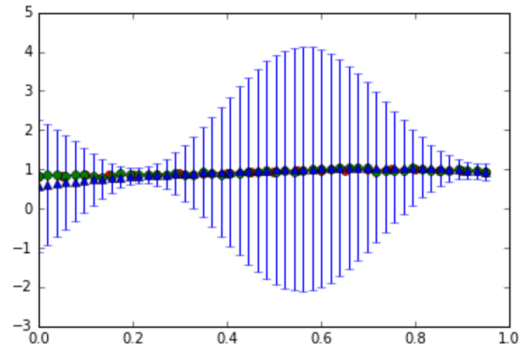


The red line is the original curve based on the equation. The green dots are the noisy data.

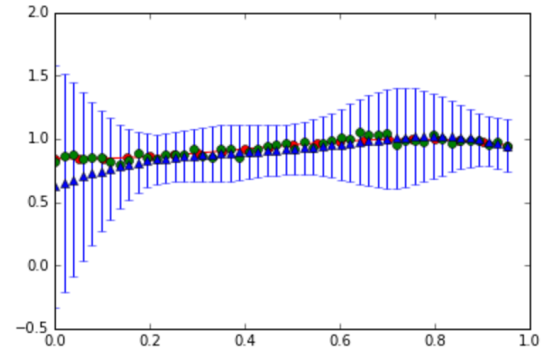
7. Compute the predictive distribution of every input data sequentially, where each input data is taken randomly from the noise data. Show your best prediction results for all 50 data one by one:



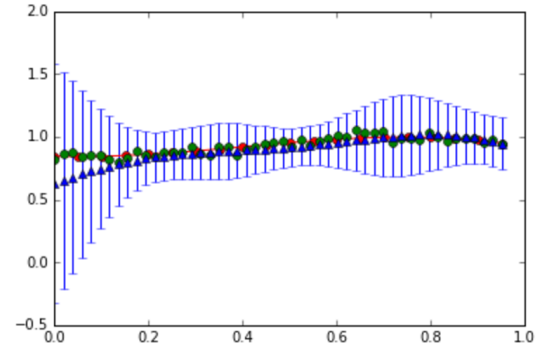
[11]
1



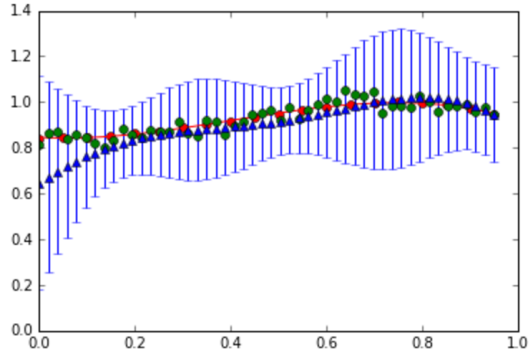
[11, 47]
2



[11, 47, 25]
3



[11, 47, 25, 26]
4

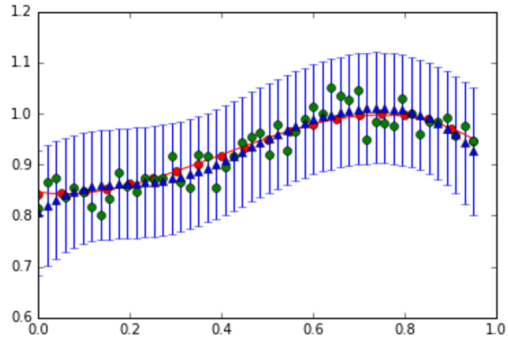


```
[11, 47, 25, 26, 7]
5
```

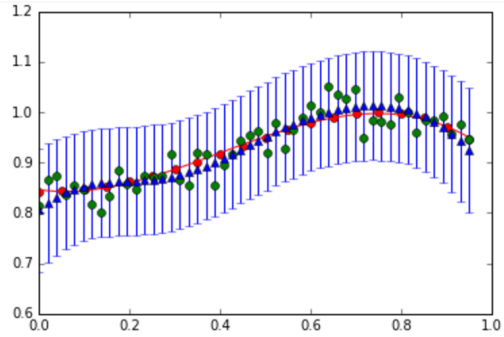
...

...

...



```
[11, 47, 25, 26, 7, 16, 5, 19, 17, 39, 11, 31, 3, 12, 21, 5, 5, 29, 1, 29, 4, 6, 3, 4, 14, 12, 0, 18, 9, 6, 19, 13,
 1, 13, 1, 4, 1, 2, 8, 7, 2, 2, 5, 2, 5, 1, 2, 0, 1]
49
```



```
[11, 47, 25, 26, 7, 16, 5, 19, 17, 39, 11, 31, 3, 12, 21, 5, 5, 29, 1, 29, 4, 6, 3, 4, 14, 12, 0, 18, 9, 6, 19, 13,
 1, 13, 1, 4, 1, 2, 8, 7, 2, 2, 5, 2, 5, 1, 2, 0, 1, 0]
50
finish!
```

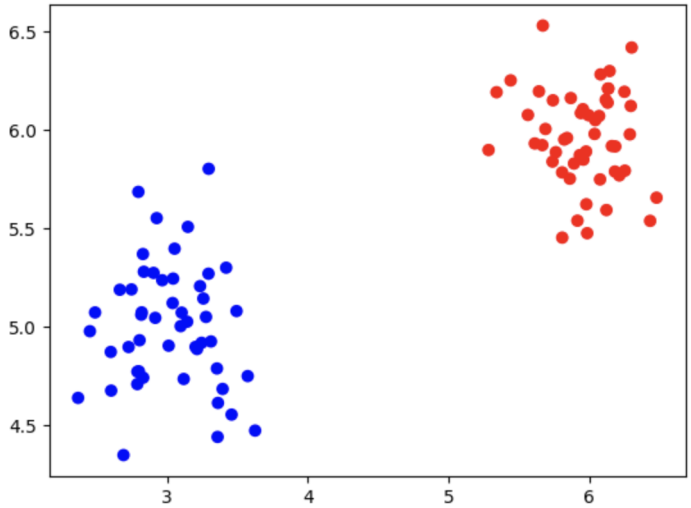
8. Explain why the predictive distribution is better than the original form of the full Bayesian inference.
9. Discuss the differences between p(t_new|t) and p(w|t).

Part 4: CLASSIFICATION

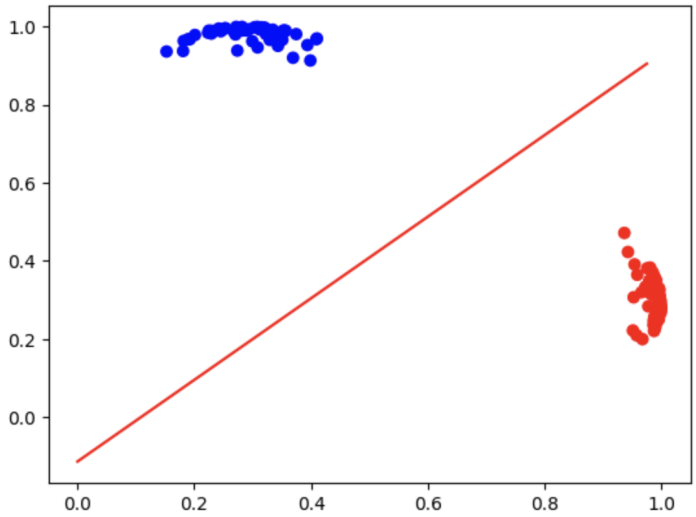
1. This is an example code to generate classification data of two classes

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


N1 = 50 # 100 or 50
N2 = 50
K = 2
sigma = 0.1

mean = (6,6)
cov = [[sigma, 0], [0, sigma]]
X1 = np.random.multivariate_normal(mean, cov, N1)
c1= ['red'] * len(X1)

mean = (3, 5)
cov = [[sigma, 0], [0, sigma]]
X2 = np.random.multivariate_normal(mean, cov, N2)
c2= ['blue'] * len(X2)

#outlier:
mean = (8, 6)
cov = [[sigma, 0], [0, sigma]]
X3 = np.random.multivariate_normal(mean, cov, int(np.floor(N2)))
c3= ['blue'] * len(X3)

#X2 = np.concatenate((X2,X3)) # open for n1=100 and close for n1=50
#c2 = np.concatenate((c2,c3)) # open for n1=100 and close for n1=50


X = np.concatenate((X1,X2))
color = np.concatenate((c1,c2))

T = []
for n in range(0,len(X)):
    if(n<len(X1)):
        T.append(0)
    if(n>=len(X1) and n<len(X1)+len(X2)):
        T.append(1)


plt.scatter(X[:, 0], X[:, 1],  marker='o', c=color)

plt.show()

```

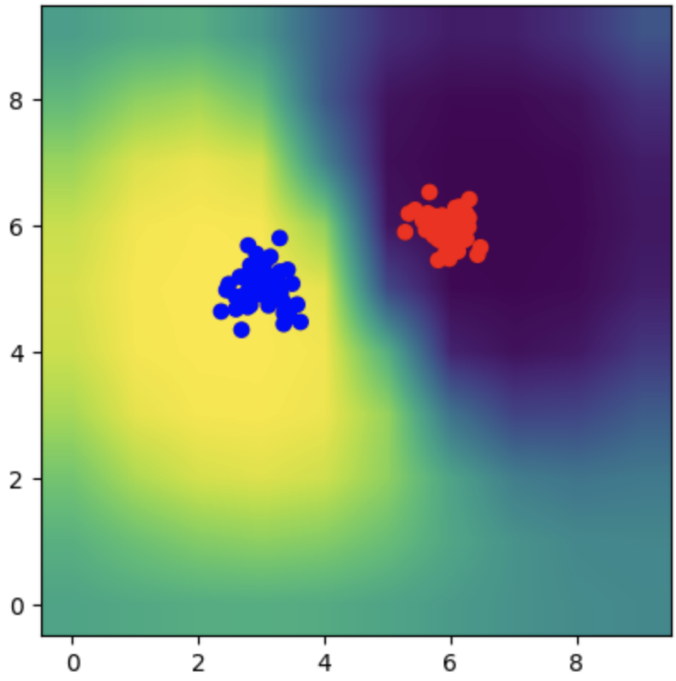
2. Based on the code above, generate data similar to:



3. Implement the MAP for classification, so that using the above data as training, we can generate a classifier in the basis function space (a space with higher dimensionality). Show the first and second bases similar to:



4. In the testing stage, use any other locations in the original space (in the original space with low dimensionality, in the range of 0 to 10) as the test data, and plot the predicted labels (whose values between 0 to 1) in colors similar to:



Submission

Submit both your code and results in an ipython-notebook format via Canvas by the deadline. Deadlines are strict. **Late submission will be deducted 10 points (out of 100) for every 24 hours. Zero marks for late submissions of more than 5 days.**