AE567

Statistical Inference, Estimation, and Learning

Project 2:

Part 1: Bayesian Inference and Decisions

Deniz Dost

UM ID: 82337583

1. Introduction

The problem statement is given with a set of initial data from a sensor on a flying robot as shown below in Table 1. The sensor outputs are given according to the x and y coordinates. The output basically shows the strength of the scent to localize a buried object. The field in consideration is two-dimensional $(X, Y) : [-1,1] \times [-1,1]$.

X	Y	Sensor
0.1	0.05	3.39382006
-0.9	0.3	3.2073034
0.2	0.4	3.39965035
0.8	-0.3	3.68810201
-0.6	0.3	2.96941623
0.3	-0.2	2.99495501
0.5	-0.84	3.94274928
-0.5	0.85	2.7968011
-0.01	-0.76	3.34929734
-0.9	-0.9	3.91296165

Table 1: Initial Sensor Data

The initial set of data points can be seen in three dimension in Figure 1.

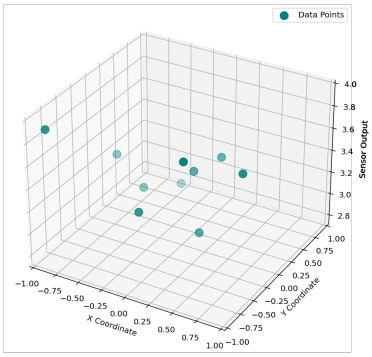

Figure 1: Initial Set of Data Points

2. GPR Setting

In order to solve this problem, an algorithm for non-parametric Gaussian Process Regression (GPR) is used. This algorithm basically uses the training data (training inputs and training outputs), such as the data provided in Table 1, and then computes the array of mean and the covariance matrix at prediction points. The computation for the mean is done according to the following formula:

$$E[f(x)\,|\,d] = \mu(x,y) + k(x,\bar{x},y,\bar{y})(k(x,\bar{x},y,\bar{y}) + \sigma^2 I)^{-1}(\bar{z} - \mu(\bar{x},\bar{y}))$$

And the computation for covariance is done according to the following formula:

$$Cov(f(x), f(x)\,|\,d) = k(x,x,y,y) - k(x,\bar{x},y,\bar{y})(k(\bar{x},\bar{x},\bar{y},\bar{y}) + \sigma^2 I)^{-1}(k(\bar{x},x,\bar{y},y))$$

where;

- d is the data, $(\bar{x},\bar{y})$ as the x and y coordinates of the sensor data and $\bar{z}$ as the sensor output value,
- (x,y) is the prediction point coordinates
- σ^2 is the noise variance,
- μ is the prior mean function, which is calculated from the data set for this problem setting by simply averaging the given sensor data, and
- $k(x,y)$ is the covariance kernel function which outputs a covariance matrix.

The covariance kernel is a function between two arbitrary points which computes the covariance. Here, the squared exponential kernel is used which is a simple algorithm to work with two dimensional problem like in this case with only two parameters to be tuned. The squared exponential kernel is formulated as below for the one dimensional case:

$$k(x,x';\tau,l) = \tau\ e^{(-0.5\frac{(x-x')^2}{l^2})}$$

This can be converted into the two dimensional case for this problem statement as below:

$$k(x,x',y,y';\tau,l) = k(x,x')k(y,y')$$

where

$$k(x, x'; \tau_1, l) = \tau_1 \, e^{(-0.5\frac{(x-x')^2}{l^2})}$$

$$k(y, y'; \tau_2, l) = \tau_2 \, e^{(-0.5\frac{(y-y')^2}{l^2})}, \quad where \ \tau_2 = 1$$

Inference is performed directly for the predictions. Nonparametric GPR does not require a basis function. Nonparametric GPR is used in this problem since the data provided is only a small amount and the computational cost associated with it, offline cost of the number of data points and online cost of inversion which is its third power, would not be much.

For this problem, the field is discretized by $n = 50$, where $X = linspace([-1 : n : 1])$ and $Y = linspace([-1 : n : 1])$, to create prediction points. τ, l, and σ^2 are chosen to be 0.3, 0.2, 10^{-8} respectively starting with the numbers used in examples and modifying them by hand to a point.

3. Initial GPR Results

When the GPR function is run with the data provided in the previous section, the result is computed as shown in Figure 2 with the sensor data shown in black dots. A surface graph as shown in Figure 3 can be obtained to qualitatively observe the trends in the mean (upper surface) and variance (lower surface) in a better way.

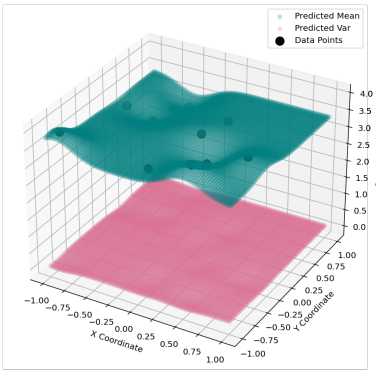

Figure 2: Mean and Variance at Prediction Points with Initial Data Set

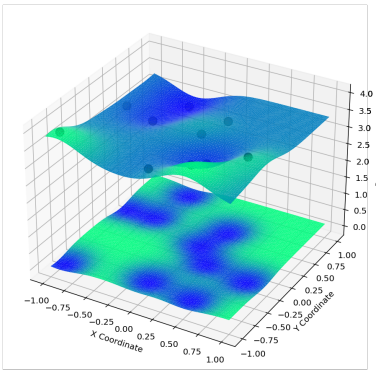

Figure 3: Mean and Variance Surfaces at Prediction Points with Initial Data Set

There are clearly some ups and downs in both mean and variance at the prediction points. It would be preferred to examine further sensor data at some of these. However, only looking at the mean and variance may not be enough on this decision. A further analysis can be done by combining them. In this case, computing the standard deviation from the variation and a look at the mean ± 2(standard deviation) value as shown in Figure 4 could be useful. This way, the information both from the mean, on how strong the scent would be, and the variance, how certain/spread the output is, can be used simultaneously. Again, a surface plot would be useful to understand the trend as shown in Figure 5. Here, it is observed that some peaks in the mean are between a smaller space encapsulated by the lower and upper surfaces, and some are in larger spaces.

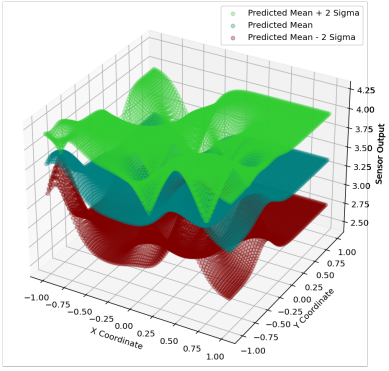

Figure 4: Mean ± 2 Standard Deviation at Prediction Points with Initial Data Set

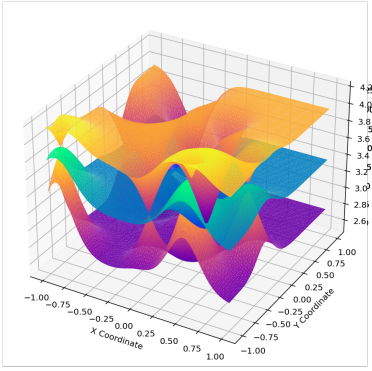

Figure 5: Mean ± 2 Standard Deviation Surfaces at Prediction Points with Initial Data Set

With the mean values at prediction points, there is the expectation of scent strength from the sensor. A look at the standard deviation alongside with mean and mean ± 2(standard deviation) can be seen in Figure 6 and a clearer surface graph with standard deviation at the very bottom in Figure 7. This way, the peaks of mean can be compared with its standard deviation to make an educated guess on the further sensor readings to be acquired. Low standard deviation areas are the certain areas. High standard deviation areas are the spread areas, where if a peak exists around would mean that there is uncertainty on that area and the scent could also be coming from there.

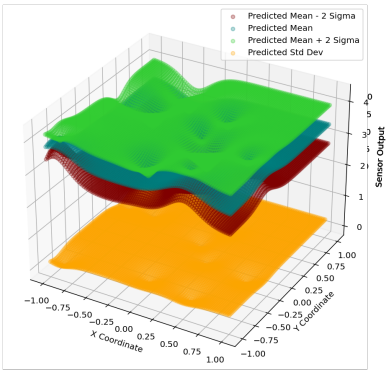

Figure 6: Mean ± 2 Standard Deviation and Standard Deviation at Prediction Points with Initial Data Set

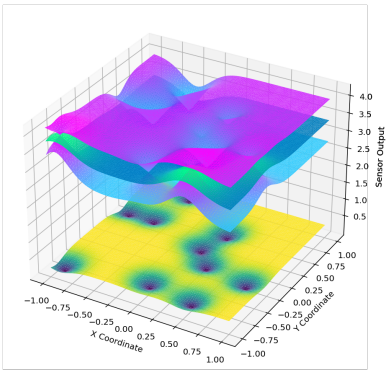

Figure 7: Mean ± 2 Standard Deviation and Standard Deviation Surfaces at Prediction Points with Initial Data Set

4. Hyperparameter Selection

The presented results above are obtained through initial τ, l, and σ^2 values. These are the hyper parameters for this problem statement since they are the inputs of the function for inference algorithms used, namely the kernel and GPR. These parameters can be tuned, both intuitively by hand and algorithmically.

One of the ways to tune the hyper parameters algorithmically is by maximizing the marginal likelihood. This method can be used in low data regime, which is also present in this problem statement since only 10 sensor readings are provided. The expression to be maximized is as shown below:

$$log(\bar{y}\,|\,\bar{x},\theta) = -0.5(\bar{y}-\mu(\bar{x}))^T(k(\bar{x},\bar{x})+\sigma^2 I)^{-1}(\bar{y}-\mu(\bar{x})) - 0.5log\,|\,k(\bar{x},\bar{x})+\sigma^2 I\,| - 0.5nlog(2\pi)$$

where $\bar{x}$ is the sensor rating coordinates (for this problem setting $(\bar{x},\bar{y})$) and $\bar{y}$ is the sensor outputs at these corresponding coordinates.

So, a likelihood function is created to compute the likelihood to be maximized. Then, the python built in fiction for minimization is used to minimize the negative of this likelihood. That way, an optimization for the hyperparameters are made. The output for the hyperparameters is 3.78974722e-01 2.64125695e-01 5.54443024e-09 respectively for τ, l, and σ^2 for the initial set of data.

The results obtained when the GPR is done with the optimized hyper parameters are shown in Figure 8, and as surfaces in Figure 9. The trends are similar since the optimized values do not differ drastically from the initial values.

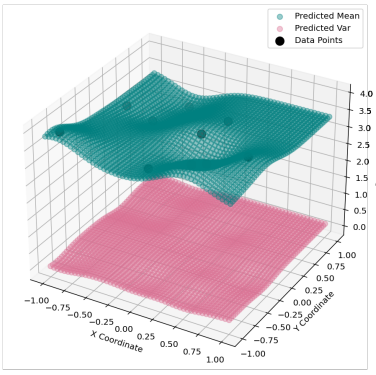

Figure 8: Mean and Variance at Prediction Points with Initial Data Set with Optimized Hyperparameters

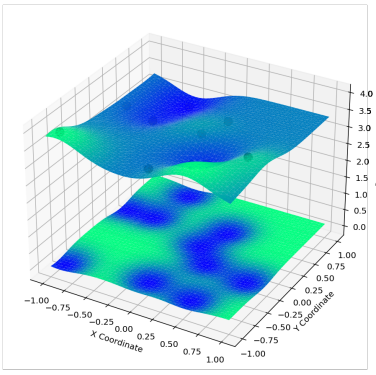

Figure 9: Mean and Variance Surfaces at Prediction Points with Initial Data Set with Optimized Hyperparameters

In order to take both the mean and variance into account, Figure 10 and Figure 11 are constructed as below. The figures presented can be used to decide on points to further get readings from at each level of this project.

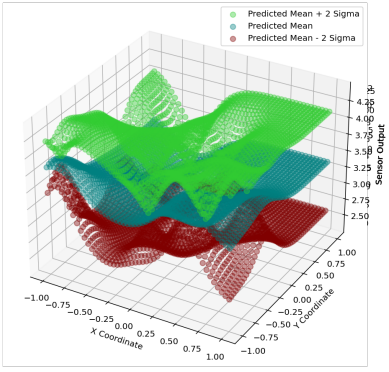

Figure 10: Mean ± 2 Standard Deviation and Mean at Prediction Points with Initial Data Set with Optimized Hyperparameters

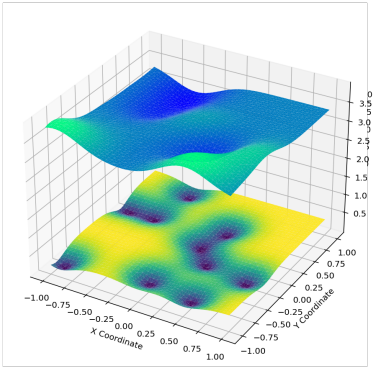

Figure 11: Mean and Standard Deviation Surfaces at Prediction Points with Initial Data Set with Optimized Hyperparameters

The points to get more data points can be selected from the graph by eye too. However, it is also useful to use an algorithm to find these points.

5. Data Point Selection

In order to find the data points to further investigate, three different things can be considered, such as the mean, mean ± 2(standard deviation), and variance at predicted points. The peaks for each data set can be found. Mean peaks exhibit the expected high sensor data which is how strong the scent is. The variance peaks are where the model is having high variance which would mean that the expected scent at that prediction point is more vague. So, these areas would require more data points to inspect. And the mean ± 2(standard deviation) would represent a combination of these two. An algorithm which follows the following method is run to find the coordinates of the peaks at each of the aforementioned as shown below:

- Create a pad of zeros at the perimeter of the matrix to avoid undefined evaluations at the corners and sides
- For all y coordinates check every x coordinate
- Evaluate the value (mean, mean ± 2(standard deviation), and variation)
- See if its bigger than all of its eight neighbours

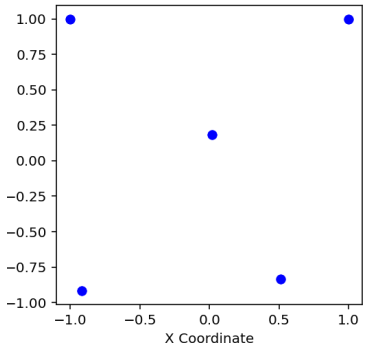

Figure 12: Peak Point Coordinates for Mean

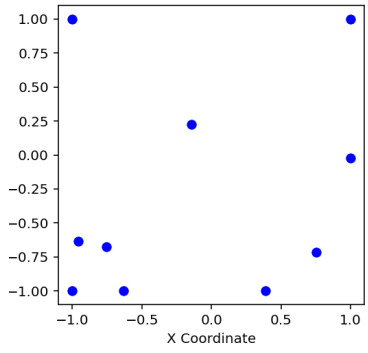

Figure 13: Peak Point Coordinates for Mean ± 2(standard deviation)

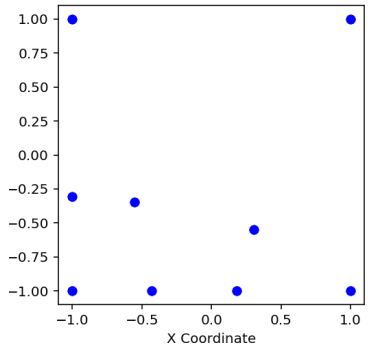

Figure 14: Peak Point Coordinates for Variance

The coordinates obtained for the three set can be seen in Figures 12-14. The values are examined at these points. The observation can also be done visually from the Figures 9-11. Points shown in Figure 13 were of interest since they show the behavior

of both mean and spread. Two points were selected from that, and the other two were selected form the mean and variance considering the visual evaluation of these two at the same time. This way, a set of coordinates to require sensor readings are obtained.

See Appendix 1 for the code of this part for the analysis with the initial data set.

6. Analysis with the First Data Set

The initial data set is now expanded with the additional data requested according to the previous section. So, the first data set is as shown below in Table 2 with the additional data shown at the bottom:

X	Y	Sensor
0.1	0.05	3.39382006
-0.9	0.3	3.2073034
0.2	0.4	3.39965035
0.8	-0.3	3.68810201
-0.6	0.3	2.96941623
0.3	-0.2	2.99495501
0.5	-0.84	3.94274928
-0.5	0.85	2.7968011
-0.01	-0.76	3.34929734
-0.9	-0.9	3.91296165
0.51	-0.84	3.96070842
0.75	-0.7	3.91571481
-0.75	-0.7	3.57317869
0.18	-1	3.80625401

Table 2: First Set of Sensor Data

The first set of data points can be seen in three dimension in Figure 15:

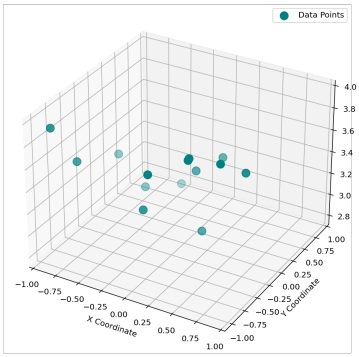

Figure 15: First Set of Sensor Data

In this part onward, only the analysis with the optimized hyper parameters are included. When the marginal likelihood is minimized, the output for the hyperparameters follows as 3.91039125e-01 3.58599428e-01 7.27023913e-05 respectively for τ, l, and σ^2. When these parameters are used to obtain results from the GPR algorithm, the output is as shown in Figure 16, and in Figure 17 as mean surface at the top and variance surface at the bottom. Mean surface looks to have sharper peaks in a sense while the main trend seems similar. Also, variance is flatter in more area.

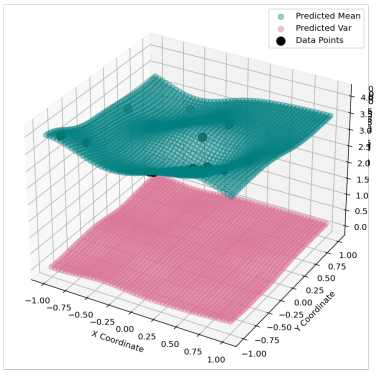

Figure 16: Mean and Variance at Prediction Points with the First Data Set with Optimized Hyperparameters

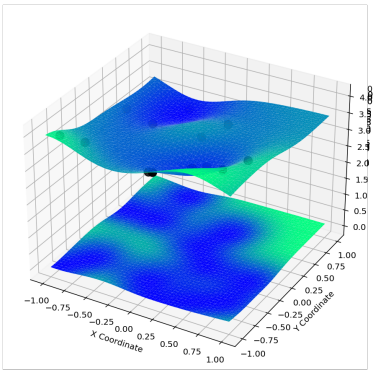

Figure 17: Mean and Variance at Prediction Points with the First Data Set with Optimized Hyperparameters

This makes sense since the data points have increased, which decrease the variance at these coordinates as shown in Figure 19. It is also observed from Figure 18, in a closer look at the mean, that the data points fit on the predicted mean surface. There seems to be another peak emerging behind the most obvious peak as a difference from the initial data set results.

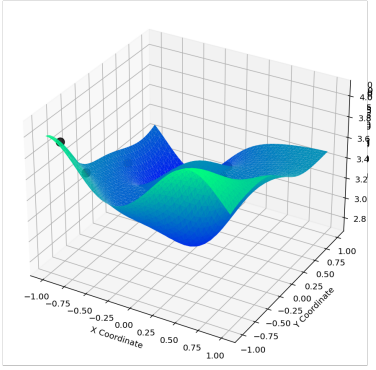

Figure 18: Mean at Prediction Points with the First Data Set with Optimized Hyperparameters

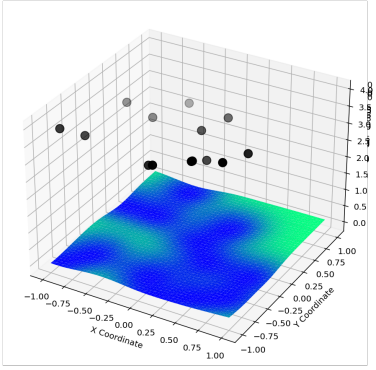

Figure 19: Variance at Prediction Points with the First Data Set with Optimized Hyperparameters

As before, mean ± 2(standard deviation) evaluations can be seen in Figure 20 as well as the standard deviation (at the bottom) and the mean (on the top) surfaces in Figure 21. The standard deviation trend is intuitively same with the variance trend as before. These plots are used to decide on sampling points as described in the previous sections.

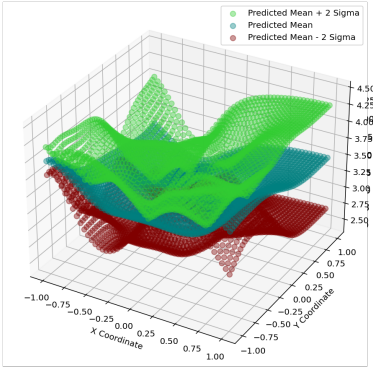

Figure 20: Mean ± 2 Standard Deviation and Mean at Prediction Points with First Data Set with Optimized Hyperparameters

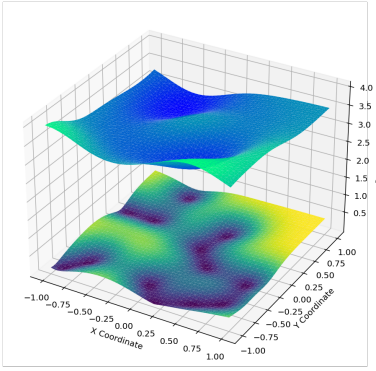

Figure 21: Mean and Standard Deviation Surfaces at Prediction Points with the First Data Set with Optimized Hyperparameters

When the peaks at each of these sets are evaluated, the following figures, Figure 22-24 are obtained.

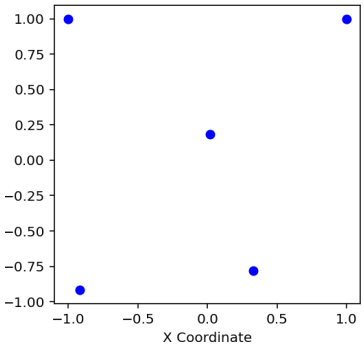

Figure 22: Peak Point Coordinates for Mean

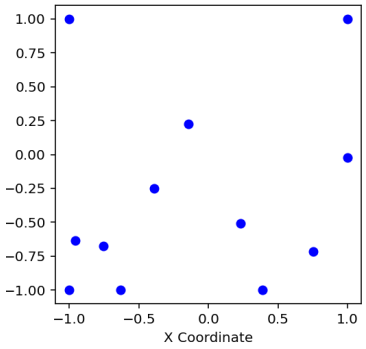

Figure 23: Peak Point Coordinates for Mean ± 2(standard deviation)

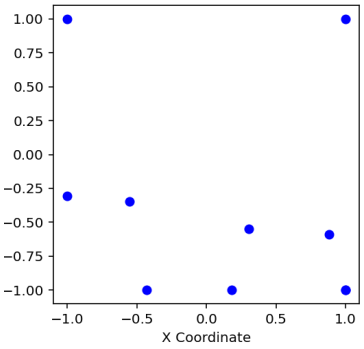

Figure 24: Peak Point Coordinates for Variance

These points are evaluated and their combination is also taken into account when requiring new data points as described previously.

See Appendix 2 for the code of this part for the analysis with the first data set.

7. Analysis with the Second Data Set

The first data set is now expanded with the additional data requested according to the previous section. So, the second data set is as shown below in Table 3 with the additional data shown at the bottom:

X	Y	Sensor
0.1	0.05	3.39382006
-0.9	0.3	3.2073034
0.2	0.4	3.39965035
0.8	-0.3	3.68810201
-0.6	0.3	2.96941623
0.3	-0.2	2.99495501
0.5	-0.84	3.94274928
-0.5	0.85	2.7968011
-0.01	-0.76	3.34929734
-0.9	-0.9	3.91296165
0.51	-0.84	3.96070842
0.75	-0.7	3.91571481
-0.75	-0.7	3.57317869
0.18	-1	3.80625401
0.33	-0.78	3.60667046
0.88	-0.59	3.71839899
0.23	-0.51	4.36210715
-0.39	-0.25	3.45358599

Table 3: Second Set of Sensor Data

The second set of data points can be seen in three dimension in Figure 25:

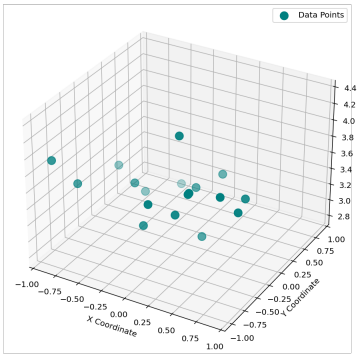

Figure 25: Second Set of Sensor Data

The output for the hyperparameters is 3.94739573e-01 1.57105366e-01 1.07775013e-08 respectively for τ, l, and σ^2. When these parameters are used to obtain results from the GPR algorithm, the output is as shown in Figure 26, and in Figure 27 as mean surface at the top and variance surface at the bottom. There is an obvious different peak in the top plot of Figure 27. The variance is also low on sensor data points.

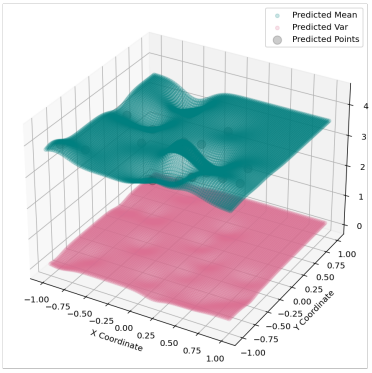

Figure 26: Mean and Variance at Prediction Points with the Second Data Set with Optimized Hyperparameters

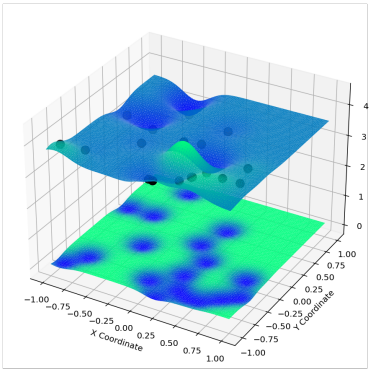

Figure 27: Mean and Variance Surfaces at Prediction Points with the Second Data Set with Optimized Hyperparameters

This makes sense since the data points have increased in that area, which decrease the variance at these coordinates as shown in Figure 29. It is also observed from Figure 28,

in a closer look at the mean, that the data points fit on the predicted mean surface. The peak emerging behind the previous peak as a difference from the first data set results is observed qualitatively.

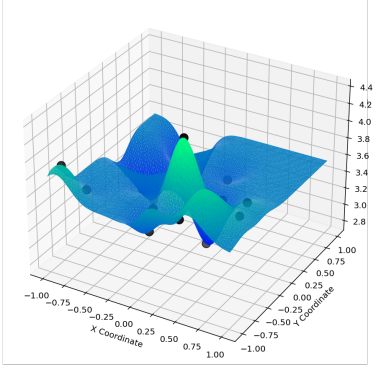

Figure 28: Mean at Prediction Points with the Second Data Set with Optimized Hyperparameters

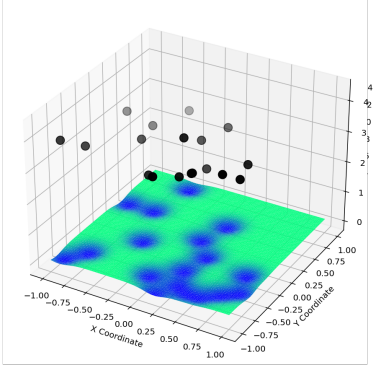

Figure 29: Variance at Prediction Points with the Second Data Set with Optimized Hyperparameters

As before, mean ± 2(standard deviation) evaluations can be seen in Figure 30 as well as the standard deviation (at the bottom) and the mean (on the top) surfaces in Figure 31. The standard deviation trend is intuitively same with the variance trend as before. And the standard deviation at the new peak is low. These plots are used to decide on sampling points as described in the previous sections.

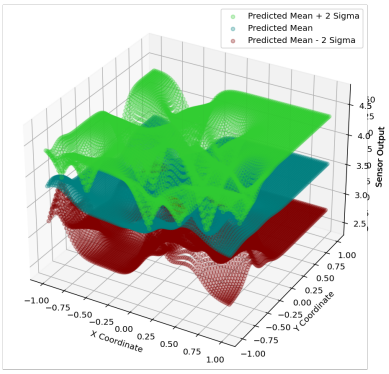

Figure 30: Mean ± 2 Standard Deviation and Mean at Prediction Points with the Second Data Set with Optimized Hyperparameters

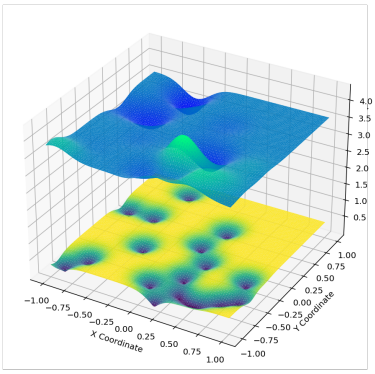

Figure 31: Mean and Standard Deviation Surfaces at Prediction Points with the Second Data Set with Optimized Hyperparameters

When the peaks at each of these sets are evaluated, the following figures, Figure 32-34 are obtained.

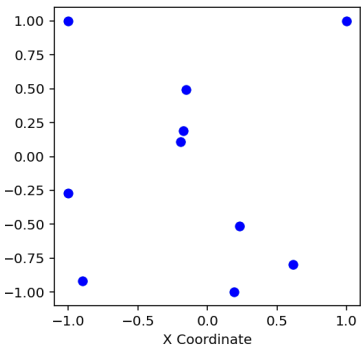

Figure 32: Peak Point Coordinates for Mean

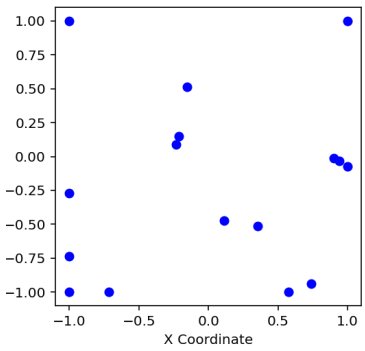

Figure 33: Peak Point Coordinates for Mean ± 2(standard deviation)

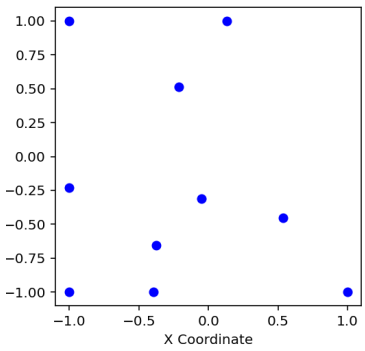

Figure 34: Peak Point Coordinates for Variance

The first data point is obtained by examining the mean surface where the last one was obtained by examining the variance. The other two is from the mean ± 2(standard deviation) examination. Also, their relationships, when is one high/low where the other is high/low is also considered when making this decision.

See Appendix 3 for the code of this part for the analysis with the second data set.

8. Analysis with the Third Data Set

The third data set is as shown below in Table 4 with the additional data shown at the bottom:

X	Y	Sensor
0.1	0.05	3.39382006
-0.9	0.3	3.2073034
0.2	0.4	3.39965035
0.8	-0.3	3.68810201
-0.6	0.3	2.96941623
0.3	-0.2	2.99495501
0.5	-0.84	3.94274928
-0.5	0.85	2.7968011
-0.01	-0.76	3.34929734
-0.9	-0.9	3.91296165
0.51	-0.84	3.96070842
0.75	-0.7	3.91571481
-0.75	-0.7	3.57317869
0.18	-1	3.80625401
0.33	-0.78	3.60667046
0.88	-0.59	3.71839899
0.23	-0.51	4.36210715
-0.39	-0.25	3.45358599
0.62	-0.8	4.0757880
0.35	-0.5	4.18858322
-0.15	0.5	4.82464266
-0.05	-0.45	4.05570494

Table 4: Third Set of Sensor Data

The third set of data points can be seen in three dimension in Figure 35:

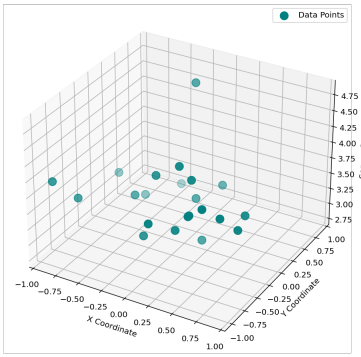

Figure 35: Third Set of Sensor Data

The output for the hyperparameters is 5.03440238e-01 1.97190426e-01 6.65356895e-09 respectively for τ, l, and σ^2. When these parameters are used to obtain results from the GPR algorithm, the output is as shown in Figure 36, and in Figure 37 as mean surface at the top and variance surface at the bottom. There is another peak in the top plot of Figure 37, which reaches the highest value so far. The variance is also flatter on on general compared to the previous.

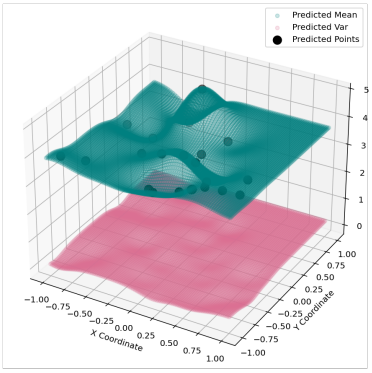

Figure 36: Mean and Variance at Prediction Points with the Third Data Set with Optimized Hyperparameters

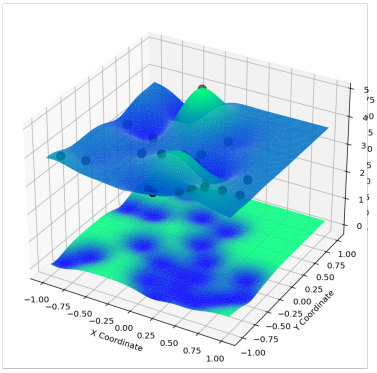

Figure 37: Mean and Variance Surfaces at Prediction Points with the Third Data Set with Optimized Hyperparameters

This makes sense since the data points have increased, which decrease the variance around these coordinates as shown in Figure 39. It is also observed from Figure 38, in

a closer look at the mean, that the data points fit on the predicted mean surface. The new peak emerging is obvious by qualitative examination since it has the highest value so far. Variance also seems to be low in that area in Figure 39.

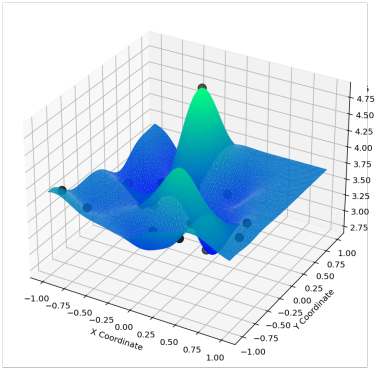

Figure 38: Mean at Prediction Points with the Third Data Set with Optimized Hyperparameters

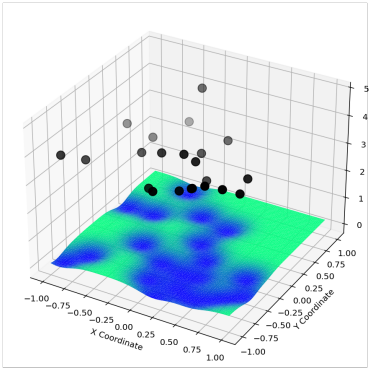

Figure 39: Variance at Prediction Points with the Third Data Set with Optimized Hyperparameters

As before, mean ± 2(standard deviation) evaluations can be seen in Figure 40 as well as the standard deviation (at the bottom) and the mean (on the top) surfaces in Figure 41. The standard deviation trend is intuitively same with the variance trend as before. And the standard deviation at the new peak is low.

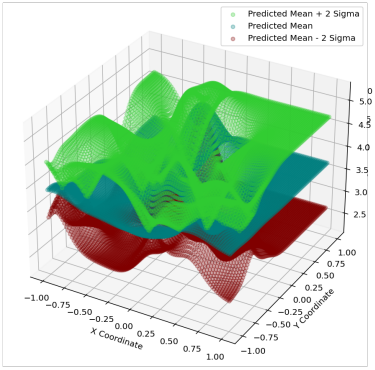

Figure 40: Mean ± 2 Standard Deviation and Mean at Prediction Points with the Third Data Set with Optimized Hyperparameters

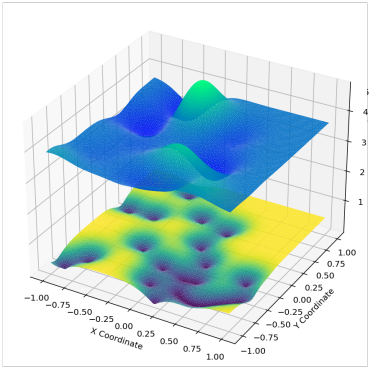

Figure 41: Mean and Standard Deviation Surfaces at Prediction Points with the Third Data Set with Optimized Hyperparameters

When the peaks at each of these sets are evaluated, the following figures, Figure 42-44 are obtained. This time, the low point of variance is also examined, and it is observed that it is actually low at the highest peak, from Figure 45. This makes sense since there is sensor data from that part of the area.

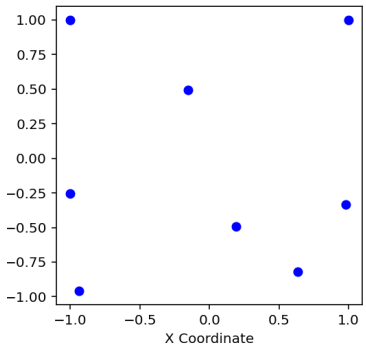

Figure 42: Peak Point Coordinates for Mean

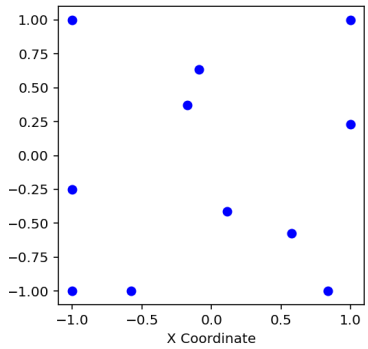

Figure 43: Peak Point Coordinates for Mean ± 2(standard deviation)

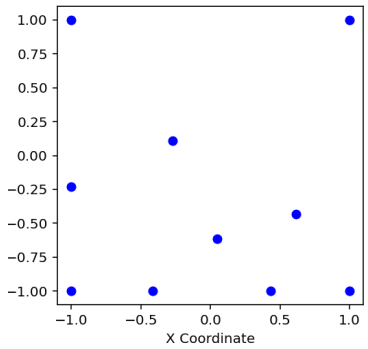

Figure 44: Peak Point Coordinates for Variance

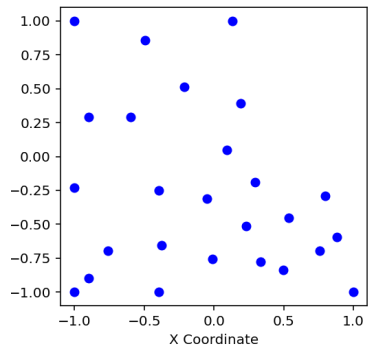

Figure 44: Low Point Coordinates for Variance

The possible location can be interpreted as somewhere around (-0.151515,0.494949) with the expected value of 4.824232 as the scent strength reading. This is commented on by examining the graphs above. Figure 42 depicts the peaks for the mean at the prediction point. Therefore, a high value would indicate high scent. Figure 45 depicts the lowest values of variances for the prediction points. The point with the high mean corresponds to a point with low variance which would indicate that the model is more

certain in that area. The variance at that point is 1.78808797e-04, which is one of the lowest in the set.

See Appendix 4 for the code of this part for the analysis with the third data set.

9. Appendices

Appendix 1:

```
#### APPENDIX 1 #####

%matplotlib inline
%matplotlib widget
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import math
from scipy.optimize import minimize

# Input Values

xySensor  = np.array([[0.1,-0.9,0.2,0.8,-0.6,0.3,0.5,-0.5,-0.01,-0.9],
                   [0.05,0.3,0.4,-0.3,0.3,-0.2,-0.84,0.85,-0.76,-0.9]])
#xSensor   = xySensor[0,:]
#ySensor   = xySensor[1,:]
outSensor                                                                                    =
np.array([3.39382006,3.2073034,3.39965035,3.68810201,2.96941623,2.99495501,3.94274928,2.7968011,3.34929
734,3.91296165])

discretization = 50
xPred =  np.linspace(-1, 1, discretization)
yPred =  np.linspace(-1, 1, discretization)
xPredMesh, yPredMesh = np.meshgrid(xPred,yPred)
xyPred = (np.array([xPredMesh.flatten(),yPredMesh.flatten()]))

MEAN = np.mean(outSensor)

noise = 1e-8
tau = 0.3
l = 0.2

# Plot Data Points

plt.figure(num='Data Points',figsize=(10,8))
fig1 = plt.axes(projection='3d')
fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='teal',label="Data Points",s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig1.set_zlabel("Sensor Output")
plt.legend()
plt.show()

# Function: Kernel

def sqexp(x, xp, tauin, lin):
    """Squared exponential kernel (1 dimensional)

    Inputs
    ------
    x : (N), array of multiple inputs
    xp: float

    Returns
```

```
    -------
    cov (N,) -- Covariance between each input at *x* and the function values at *x*
    """
    cov = tauin**2 * np.exp(-1/2 * (x - xp)**2 / lin**2)
    return cov

def sqexp2d(x, xp, tauin, lin):
    """Squared exponential kernel (2 dimensional) """
    xX   = x[0,:]
    xY   = x[1,:]
    xpX  = xp[0]
    xpY  = xp[1]
    tauX = tauin
    tauY = 1 # tauY=1 bc I only want one tau to tune
    sqexpX = sqexp(xX, xpX, tauX, lin)
    sqexpY = sqexp(xY, xpY, tauY, lin)
    cov = sqexpX*sqexpY
    return cov

# Function: Covariance

def build_covariance(x, xp,tauin, lin, kern):
    """Build a covariance matrix

    Inputs
    -------
    x: (N) array of inputs
    xp: (M) array of inptus
    kern: a function mapping inputs to covariance

    Outputs
    -------
    cov: (N, M) covariance matrix
    """
    out = np.zeros((x.shape[1], xp.shape[1]))
    for jj in range(xp.shape[1]):
        out[:, jj] = kern(x, xp[:,jj], tauin, lin)
    return out

# Function: GPR

def gpr(xtrain, ytrain, xpred, noise_var, mean_func, kernel):
    """Gaussian process regression Algorithm

    Inputs
    -------
    xtrain: (N, ) training inputs
    ytrain: (N, ) training outputs
    xpred:  (M, ) locations at which to make predictions
    noise_var: (N, ) noise at every training output
    mean_func: function to compute the prior mean
    kernel: covariance kernel

    Returns
    -------
    pred_mean : (M, ) predicted mean at prediction points
    pred_cov : (M, M) predicted covariance at the prediction points
    --
    """
    #xtrain  = np.transpose(xtrain)
    cov = build_covariance(xtrain, xtrain, tau, l, kernel)
    u, s, v = np.linalg.svd(cov)
    sqrtcov = np.dot(u, np.sqrt(np.diag(s)))
```

```
    noiseArray=(np.ones(np.size(xtrain[1,:])))*noise_var

    # pseudoinverse is better conditioned
    invcov = np.linalg.pinv(cov + np.diag(noiseArray))

    vec_pred = build_covariance(xpred, xtrain, tau, l, kernel)

    pred_mean = MEAN + np.dot(vec_pred, np.dot(invcov, ytrain - MEAN))
    cov_predict_pre = build_covariance(xpred, xpred, tau, l, kernel)
    cov_predict_up = np.dot(vec_pred, np.dot(invcov, vec_pred.T))
    pred_cov = cov_predict_pre - cov_predict_up
    #print(cov_predict_pre)

    return pred_mean, pred_cov

# GPR Results

meanPred, covPred = gpr(xySensor, outSensor, xyPred, noise, MEAN, sqexp2d)

xPredMeshF, yPredMeshF = xPredMesh.flatten(),yPredMesh.flatten()
plt.figure(num='mean and cov',figsize=(10,8))
fig1 = plt.axes(projection='3d')
fig1.plot_trisurf(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label='Predicted Mean')
fig1.plot_trisurf(xPredMeshF,yPredMeshF,np.diag(covPred),cmap='winter',label='Predicted Cov')
fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Data Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig1.set_zlabel("Sensor Output")
plt.show()

# GPR Results

meanPred, covPred = gpr(xySensor, outSensor, xyPred, noise, MEAN, sqexp2d)

xPredMeshF, yPredMeshF = xPredMesh.flatten(),yPredMesh.flatten()
plt.figure(num='mean and cov',figsize=(10,8))
fig1 = plt.axes(projection='3d')
fig1.scatter3D(xPredMeshF,yPredMeshF,meanPred,c='teal',label='Predicted Mean', alpha=0.2)
fig1.scatter3D(xPredMeshF,yPredMeshF,np.diag(covPred),c='palevioletred',label='Predicted Var', alpha=0.2)
fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Data Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig1.set_zlabel("Sensor Output")
plt.legend()
plt.show()

# Function: ± 2 Sigma Pred Computation

def twoSigma(inputMean, inputCov):
    varPred    = np.array(np.diag(inputCov))
    stdDevPred = np.sqrt(varPred)
    P2SigmaPred = inputMean + (2*stdDevPred)
    N2SigmaPred = inputMean - (2*stdDevPred)
    plt.figure(num='Pred Mean  ± 2 Sigma Pred Computation',figsize=(10,8))
    ax = plt.axes(projection='3d')
       ax.scatter3D(xPredMeshF,yPredMeshF,P2SigmaPred,c='limegreen',label='Predicted  Mean  +  2  Sigma',  alpha=0.4,
s=40)
    ax.scatter3D(xPredMeshF,yPredMeshF,meanPred,c='teal',label='Predicted Mean', alpha=0.4, s=40)
```

```
    ax.scatter3D(xPredMeshF,yPredMeshF,N2SigmaPred,c='maroon',label='Predicted Mean - 2 Sigma', alpha=0.4, s=40)
    locs, labels = plt.xticks()  # Get the current locations and labels.
    plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
    plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
    plt.xlabel("X Coordinate")
    plt.ylabel("Y Coordinate")
    ax.set_zlabel("Sensor Output")
    plt.legend()
    plt.show(ax)
    return(stdDevPred,P2SigmaPred,N2SigmaPred)

stdDevMeanPred, P2SigmaMeanPred, N2SigmaMeanPred = twoSigma(meanPred, covPred)

# Function: ± 2 Sigma Pred Computation

def twoSigma2(inputMean, inputCov):
    varPred    = np.array(np.diag(inputCov))
    stdDevPred = np.sqrt(varPred)
    P2SigmaPred = inputMean + (2*stdDevPred)
    N2SigmaPred = inputMean - (2*stdDevPred)
    plt.figure(num='Pred Mean  ± 2 Sigma Pred Computation',figsize=(10,8))
    ax = plt.axes(projection='3d')
    ax.plot_trisurf(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label='Predicted Mean')
    ax.plot_trisurf(xPredMeshF,yPredMeshF,P2SigmaPred,cmap='plasma',label='Predicted Mean + 2 Sigma')
    ax.plot_trisurf(xPredMeshF,yPredMeshF,N2SigmaPred,cmap='plasma',label='Predicted Mean - 2 Sigma')
    locs, labels = plt.xticks()  # Get the current locations and labels.
    plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
    plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
    plt.xlabel("X Coordinate")
    plt.ylabel("Y Coordinate")
    ax.set_zlabel("Sensor Output")
    plt.show(ax)
    return(stdDevPred,P2SigmaPred,N2SigmaPred)



# ± 2 Sigma Pred Computation
varPred,stdDevMeanPred, P2SigmaMeanPred, N2SigmaMeanPred = twoSigma2(meanPred, covPred)

# STD Pred Plot
plt.figure(num='Pred Mean STD',figsize=(10,8))
fig3 = plt.axes(projection='3d')
fig3.plot_trisurf(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label='Predicted Mean')
fig3.plot_trisurf(xPredMeshF,yPredMeshF,P2SigmaMeanPred,cmap='cool',label='Predicted Mean + 2 Sigma')
fig3.plot_trisurf(xPredMeshF,yPredMeshF,N2SigmaMeanPred,cmap='cool',label='Predicted Mean - 2 Sigma')
fig3.plot_trisurf(xPredMeshF, yPredMeshF, stdDevMeanPred,cmap='viridis',label="Predicted Points")
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig3.set_zlabel("Sensor Output")
plt.show()

# STD Pred Plot

plt.figure(num='Pred Mean STD',figsize=(10,8))
fig3 = plt.axes(projection='3d')
fig3.scatter3D(xPredMeshF,yPredMeshF,N2SigmaMeanPred,c='maroon',label='Predicted Mean - 2 Sigma', alpha=0.3)
fig3.scatter3D(xPredMeshF,yPredMeshF,meanPred,c='teal',label='Predicted Mean', alpha=0.3)
fig3.scatter3D(xPredMeshF,yPredMeshF,P2SigmaMeanPred,c='limegreen',label='Predicted Mean + 2 Sigma', alpha=0.3)
fig3.scatter3D(xPredMeshF, yPredMeshF, stdDevMeanPred,c='orange',label='Predicted Std Dev', alpha=0.3)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
```

```
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig3.set_zlabel("Sensor Output")
plt.legend()
plt.show()

# Function: HyperParameters

def likelihood(x):
    tau_l, l_l, noise_l = x
    noiseArray=(np.ones(np.size(xySensor[0,:])))*noise_l
    cov = build_covariance(xySensor, xySensor, tau_l, l_l, sqexp2d)
    invcov = np.linalg.pinv(cov + np.diag(noiseArray))
    likelihoodOut = 0.5 * np.dot((outSensor.T-MEAN), np.dot(invcov, (outSensor-MEAN)))\
          + 0.5 * np.log(np.linalg.det(cov + np.diag(noiseArray)))\
          + 0.5 * np.size(xySensor[1,:]) * np.log(2*np.pi)

    return likelihoodOut

# HyperParameters Results

noise = 1e-8
tau = 0.3
l = 0.2

xInit=np.array([tau, l, noise])
bounds = [(1e-15,None),(1e-15,None),(1e-15,None)]
minOut = minimize(likelihood, xInit, bounds=bounds, method='nelder-mead')
minOutX = np.array(minOut.x)
tau = minOutX[0]
l = minOutX[1]
noise = minOutX[2]
meanPred, covPred = gpr(xySensor, outSensor, xyPred, noise, MEAN, sqexp2d)
fig = plt.figure('Hyperparameter Pred Mean',figsize=(10,8))
fig4 = plt.axes(projection='3d')
fig4.plot_trisurf(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label="Predicted Mean")
fig4.plot_trisurf(xPredMeshF,yPredMeshF,np.diag(covPred),cmap='winter',label="Predicted Cov")
fig4.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Data Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig4.set_zlabel("Sensor Output")
plt.show()

plt.figure(num='Hyperparameter Pred Mean',figsize=(10,8))
fig4 = plt.axes(projection='3d')
fig4.scatter3D(xPredMeshF,yPredMeshF,meanPred,c='teal',label='Predicted Mean', alpha=0.4, s=40)
fig4.scatter3D(xPredMeshF,yPredMeshF,np.diag(covPred),c='palevioletred',label='Predicted Var', alpha=0.4, s=40)
fig4.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Data Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig3.set_zlabel("Sensor Output")
plt.legend()
plt.show()

print(minOutX)

# Hyperparameter ± 2 Sigma Pred Computation
```

```
stdDevMeanPred, P2SigmaMeanPred, N2SigmaMeanPred = twoSigma(meanPred, covPred)

varPred = np.array(np.diag(covPred))

# Hyperparameter STD Pred Plot

plt.figure(num='Pred Mean STD',figsize=(10,8))
fig5 = plt.axes(projection='3d')
#fig5.plot_trisurf(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label='Predicted Mean')
#fig3.plot_trisurf(xPredMeshF,yPredMeshF,P2SigmaMeanPred,cmap='cool',label='Predicted Mean + 2 Sigma')
#fig3.plot_trisurf(xPredMeshF,yPredMeshF,N2SigmaMeanPred,cmap='cool',label='Predicted Mean - 2 Sigma')
fig5.plot_trisurf(xPredMeshF, yPredMeshF, stdDevMeanPred,cmap='viridis',label="Predicted Points")
#fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig5.set_zlabel("Sensor Output")
plt.show()

# Function : Finding indices of the peaks of a matrix

def findPeaks(inputMatrix):
   paddedMatrix = np.pad(inputMatrix,(1),'constant',constant_values=0)
   length1      = len(inputMatrix[0])
   length2      = len(inputMatrix[1])
   peakInd      = np.zeros((length1,length2))
   for xi in range(0,length1):
      for yi in range(0,length2):
         if paddedMatrix[xi+1,yi+1] > paddedMatrix[   xi , yi   ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[   xi , yi+1 ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[   xi , yi+2 ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+1 , yi   ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+1 , yi+2 ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+2 , yi   ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+2 , yi+1 ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+2 , yi+2 ]:
             peakInd[xi,yi] = 1

   return peakInd

# PredMatrix Peak Indices

meanPredMatrix       = meanPred.reshape(discretization,discretization)
meanPredPeakIndCoeff = findPeaks(meanPredMatrix)
meanPredPeak         = meanPredMatrix[meanPredPeakIndCoeff!=0]
print(meanPredPeak)

meanPredPeakIndX = xPredMesh * meanPredPeakIndCoeff
meanPredPeakIndX = meanPredPeakIndX[meanPredPeakIndX!=0]

meanPredPeakIndY = yPredMesh * meanPredPeakIndCoeff
meanPredPeakIndY = meanPredPeakIndY[meanPredPeakIndY!=0]

meanPredPeakInd  = np.array([meanPredPeakIndX,meanPredPeakIndY])
print(meanPredPeakInd)

fig6= plt.figure('PredMatrix Indices',figsize=(4,4))
plt.plot(meanPredPeakIndX,meanPredPeakIndY,'o',c='b')
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.show()
```

```
# PredMatrix + 2 Sigma Peak Indices

meanPredP2Sigma = P2SigmaMeanPred
meanPredP2SigmaMatrix       = meanPredP2Sigma.reshape(discretization,discretization)
meanPredP2SigmaPeakIndCoeff = findPeaks(meanPredP2SigmaMatrix)
meanPredP2SigmaPeak         = meanPredP2SigmaMatrix[meanPredP2SigmaPeakIndCoeff!=0]
print(meanPredP2SigmaPeak)

meanPredP2SigmaPeakIndX = xPredMesh * meanPredP2SigmaPeakIndCoeff
meanPredP2SigmaPeakIndX = meanPredP2SigmaPeakIndX[meanPredP2SigmaPeakIndX!=0]

meanPredP2SigmaPeakIndY = yPredMesh * meanPredP2SigmaPeakIndCoeff
meanPredP2SigmaPeakIndY = meanPredP2SigmaPeakIndY[meanPredP2SigmaPeakIndY!=0]

meanPredP2SigmaPeakInd  = np.array([meanPredP2SigmaPeakIndX,meanPredP2SigmaPeakIndY])
print(meanPredP2SigmaPeakInd)

fig7= plt.figure('PredMatrix + 2 Sigma Peak Indices',figsize=(4,4))
plt.plot(meanPredP2SigmaPeakIndX,meanPredP2SigmaPeakIndY,'o',c='b')
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.show()

# VarMatrix Peak Indices

#varPred = varPred
varPredMatrix       = varPred.reshape(discretization,discretization)
varPredPeakIndCoeff = findPeaks(varPredMatrix)
varPredPeak         = varPredMatrix[varPredPeakIndCoeff!=0]
print(varPredPeak)

varPredPeakIndX = xPredMesh * varPredPeakIndCoeff
varPredPeakIndX = varPredPeakIndX[varPredPeakIndX!=0]

varPredPeakIndY = yPredMesh * varPredPeakIndCoeff
varPredPeakIndY = varPredPeakIndY[varPredPeakIndY!=0]

varPredPeakInd  = np.array([varPredPeakIndX,varPredPeakIndY])
print(varPredPeakInd)


fig8= plt.figure('VarMatrix Peak Indices',figsize=(4,4))
plt.plot(varPredPeakIndX,varPredPeakIndY,'o',c='b')
plt.ylabel("Y Coordinate")
plt.xlabel("X Coordinate")

plt.show()
```

Appendix 2:

```
#### APPENDIX 2 #####

%matplotlib inline
%matplotlib widget
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import math
from scipy.optimize import minimize
```

```
# Input Values

xySensor  = np.array([[ 0.1,-0.9,0.2, 0.8,-0.6, 0.3,  0.5,-0.5,-0.01,-0.9,
                        5.10000000000000089e-01,  7.50000000000000000e-01,  -7.50000000000000000e-01,
1.79999999999999933e-01],
               [0.05, 0.3,0.4,-0.3, 0.3,-0.2,-0.84,0.85,-0.76,-0.9,
                      -8.39999999999999689e-01, -6.99999999999999556e-01,  -6.99999999999999556e-01,
-1.00000000000000000e+00]])
#xSensor   = xySensor[0,:]
#ySensor   = xySensor[1,:]
#xySensor  = np.transpose(xySensor)
outSensor                                                                                                    =
np.array([3.39382006,3.2073034,3.39965035,3.68810201,2.96941623,2.99495501,3.94274928,2.7968011,3.34929
734,3.91296165,
                        3.960708425707508251e+00, 3.915714809492213622e+00, 3.573178696741371230e+00,
3.806254007495223135e+00])

discretization = 50
xPred =  np.linspace(-1, 1, discretization)
yPred =  np.linspace(-1, 1, discretization)
xPredMesh, yPredMesh = np.meshgrid(xPred,yPred)
xyPred = (np.array([xPredMesh.flatten(),yPredMesh.flatten()]))


MEAN = np.mean(outSensor)
noise = 1e-8
tau = 0.3
l = 0.2

plt.figure(num='Data Points',figsize=(10,8))
fig1 = plt.axes(projection='3d')
fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='teal',label="Data Points",s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig1.set_zlabel("Sensor Output")
plt.legend()
plt.show()

# Function: Kernel

def sqexp(x, xp, tauin, lin):
    """Squared exponential kernel (1 dimensional)

    Inputs
    ------
    x : (N), array of multiple inputs
    xp: float

    Returns
    -------
    cov (N,) -- Covariance between each input at *x* and the function values at *x*
    """
    cov = tauin**2 * np.exp(-1/2 * (x - xp)**2 / lin**2)
    return cov

def sqexp2d(x, xp, tauin, lin):
    """Squared exponential kernel (2 dimensional) """
    xX   = x[0,:]
    xY   = x[1,:]
    xpX  = xp[0]
    xpY  = xp[1]
```

```
    tauX = tauin
    tauY = 1 # tauY=1 bc I only want one tau to tune
    sqexpX = sqexp(xX, xpX, tauX, lin)
    sqexpY = sqexp(xY, xpY, tauY, lin)
    cov = sqexpX*sqexpY
    return cov

# Function: Covariance

def build_covariance(x, xp,tauin, lin, kern):
    """Build a covariance matrix

    Inputs
    -------
    x: (N) array of inputs
    xp: (M) array of inptus
    kern: a function mapping inputs to covariance

    Outputs
    -------
    cov: (N, M) covariance matrix
    """
    out = np.zeros((x.shape[1], xp.shape[1]))
    for jj in range(xp.shape[1]):
        out[:, jj] = kern(x, xp[:,jj], tauin, lin)
    return out

# Function: GPR

def gpr(xtrain, ytrain, xpred, noise_var, mean_func, kernel):
    """Gaussian process regression Algorithm

    Inputs
    -------
    xtrain: (N, ) training inputs
    ytrain: (N, ) training outputs
    xpred:  (M, ) locations at which to make predictions
    noise_var: (N, ) noise at every training output
    mean_func: function to compute the prior mean
    kernel: covariance kernel

    Returns
    -------
    pred_mean : (M, ) predicted mean at prediction points
    pred_cov : (M, M) predicted covariance at the prediction points
    --
    """
    #xtrain  = np.transpose(xtrain)
    cov = build_covariance(xtrain, xtrain, tau, l, kernel)
    u, s, v = np.linalg.svd(cov)
    sqrtcov = np.dot(u, np.sqrt(np.diag(s)))

    noiseArray=(np.ones(np.size(xtrain[1,:])))*noise_var

    # pseudoinverse is better conditioned
    invcov = np.linalg.pinv(cov + np.diag(noiseArray))

    vec_pred = build_covariance(xpred, xtrain, tau, l, kernel)

    pred_mean = MEAN + np.dot(vec_pred, np.dot(invcov, ytrain - MEAN))
    cov_predict_pre = build_covariance(xpred, xpred, tau, l, kernel)
    cov_predict_up = np.dot(vec_pred, np.dot(invcov, vec_pred.T))
    pred_cov = cov_predict_pre - cov_predict_up
    #print(cov_predict_pre)
```

```
    return pred_mean, pred_cov

# GPR Results

meanPred, covPred = gpr(xySensor, outSensor, xyPred, noise, MEAN, sqexp2d)

xPredMeshF, yPredMeshF = xPredMesh.flatten(),yPredMesh.flatten()
plt.figure(num='mean and cov',figsize=(10,8))
fig1 = plt.axes(projection='3d')
fig1.scatter3D(xPredMeshF,yPredMeshF,meanPred,c='teal',label='Predicted Mean', alpha=0.2)
#fig1.scatter3D(xPredMeshF,yPredMeshF,np.diag(covPred),c='palevioletred',label='Predicted Var', alpha=0.2)
fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Data Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig1.set_zlabel("Sensor Output")
plt.legend()
plt.show()

# GPR Results

meanPred, covPred = gpr(xySensor, outSensor, xyPred, noise, MEAN, sqexp2d)

xPredMeshF, yPredMeshF = xPredMesh.flatten(),yPredMesh.flatten()
plt.figure(num='mean and cov',figsize=(10,8))
fig1 = plt.axes(projection='3d')
fig1.plot_trisurf(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label='Predicted Mean')
fig1.plot_trisurf(xPredMeshF,yPredMeshF,np.diag(covPred),cmap='winter',label='Predicted Cov')
fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig1.set_zlabel("Sensor Output")
plt.show()

# Function: ± 2 Sigma Pred Computation

def twoSigma(inputMean, inputCov):
    varPred    = np.array(np.diag(inputCov))
    stdDevPred = np.sqrt(varPred)
    P2SigmaPred = inputMean + (2*stdDevPred)
    N2SigmaPred = inputMean - (2*stdDevPred)

    plt.figure(num='Pred Mean  ± 2 Sigma Pred Computation',figsize=(10,8))
    ax = plt.axes(projection='3d')
       ax.scatter3D(xPredMeshF,yPredMeshF,P2SigmaPred,c='limegreen',label='Predicted  Mean  +  2  Sigma',  alpha=0.4,
s=40)
    ax.scatter3D(xPredMeshF,yPredMeshF,meanPred,c='teal',label='Predicted Mean', alpha=0.4, s=40)
    ax.scatter3D(xPredMeshF,yPredMeshF,N2SigmaPred,c='maroon',label='Predicted Mean - 2 Sigma', alpha=0.4, s=40)
    #ax.plot_trisurf(xPredMeshF, yPredMeshF, stdDevPred,cmap='jet',label="Predicted Points")
    #ax.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100)
    locs, labels = plt.xticks()  # Get the current locations and labels.
    plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
    plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
    plt.xlabel("X Coordinate")
    plt.ylabel("Y Coordinate")
    ax.set_zlabel("Sensor Output")
    plt.legend()
    plt.show(ax)
```

```
    return(varPred,stdDevPred,P2SigmaPred,N2SigmaPred)


# ± 2 Sigma Pred Computation
varMeanPred,stdDevMeanPred, P2SigmaMeanPred, N2SigmaMeanPred = twoSigma(meanPred, covPred)

# STD Pred Plot

plt.figure(num='Pred Mean STD',figsize=(10,8))
fig3 = plt.axes(projection='3d')
fig3.scatter3D(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label='Predicted Mean')
fig3.scatter3D(xPredMeshF,yPredMeshF,P2SigmaMeanPred,cmap='cool',label='Predicted Mean + 2 Sigma')
fig3.scatter3D(xPredMeshF,yPredMeshF,N2SigmaMeanPred,cmap='cool',label='Predicted Mean - 2 Sigma')
fig3.scatter3D(xPredMeshF, yPredMeshF, stdDevMeanPred,cmap='magma',label="Predicted Points")
#fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.legend()
plt.show()

# Function: HyperParameters

def likelihood(x):
    tau_l, l_l, noise_l = x
    noiseArray=(np.ones(np.size(xySensor[0,:])))*noise_l
    cov = build_covariance(xySensor, xySensor, tau_l, l_l, sqexp2d)
    invcov = np.linalg.pinv(cov + np.diag(noiseArray))
    likelihoodOut = 0.5 * np.dot((outSensor.T-MEAN), np.dot(invcov, (outSensor-MEAN)))\
          + 0.5 * np.log(np.linalg.det(cov + np.diag(noiseArray)))\
          + 0.5 * np.size(xySensor[1,:]) * np.log(2*np.pi)

    return likelihoodOut

# HyperParameters Results

noise = 1e-8
tau = 0.3
l = 0.2

xInit=np.array([tau, l, noise])
bounds = [(1e-15,None),(1e-15,None),(1e-15,None)]
minOut = minimize(likelihood, xInit, bounds=bounds, method='nelder-mead')
minOutX = np.array(minOut.x)
tau = minOutX[0]
l = minOutX[1]
noise = minOutX[2]
meanPred, covPred = gpr(xySensor, outSensor, xyPred, noise, MEAN, sqexp2d)

fig = plt.figure('Hyperparameter Pred Mean',figsize=(10,8))
fig4 = plt.axes(projection='3d')
fig4.plot_trisurf(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label="Predicted Mean")
fig4.plot_trisurf(xPredMeshF,yPredMeshF,np.diag(covPred),cmap='winter',label="Predicted Cov")
fig4.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
#plt.legend()
plt.show()
```

```
plt.figure(num='Hyperparameter Pred Mean',figsize=(10,8))
fig4 = plt.axes(projection='3d')
fig4.scatter3D(xPredMeshF,yPredMeshF,meanPred,c='teal',label='Predicted Mean', alpha=0.4, s=40)
fig4.scatter3D(xPredMeshF,yPredMeshF,np.diag(covPred),c='palevioletred',label='Predicted Var', alpha=0.4, s=40)
fig4.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Data Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig3.set_zlabel("Sensor Output")
plt.legend()
plt.show()

print(minOutX)

# Hyperparameter ± 2 Sigma Pred Computation
varPred, stdDevMeanPred, P2SigmaMeanPred, N2SigmaMeanPred = twoSigma(meanPred, covPred)

# Hyperparameter STD Pred Plot

plt.figure(num='Pred Mean STD',figsize=(10,8))
fig5 = plt.axes(projection='3d')
fig5.plot_trisurf(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label='Predicted Mean')
#fig3.plot_trisurf(xPredMeshF,yPredMeshF,P2SigmaMeanPred,cmap='cool',label='Predicted Mean + 2 Sigma')
#fig3.plot_trisurf(xPredMeshF,yPredMeshF,N2SigmaMeanPred,cmap='cool',label='Predicted Mean - 2 Sigma')
fig5.plot_trisurf(xPredMeshF, yPredMeshF, stdDevMeanPred,cmap='viridis',label="Predicted Points")
#fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
#plt.legend()
plt.show()

# Function : Finding indices of the peaks of a matrix

def findPeaks(inputMatrix):
   paddedMatrix = np.pad(inputMatrix,(1),'constant',constant_values=0)
   length1      = len(inputMatrix[0])
   length2      = len(inputMatrix[1])
   peakInd      = np.zeros((length1,length2))
   for xi in range(0,length1):
      for yi in range(0,length2):
         if paddedMatrix[xi+1,yi+1] > paddedMatrix[   xi , yi   ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[   xi , yi+1 ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[   xi , yi+2 ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+1 , yi   ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+1 , yi+2 ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+2 , yi   ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+2 , yi+1 ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+2 , yi+2 ]:

             peakInd[xi,yi] = 1

   return peakInd

# PredMatrix Peak Indices

meanPredMatrix       = meanPred.reshape(discretization,discretization)
meanPredPeakIndCoeff = findPeaks(meanPredMatrix)
meanPredPeak         = meanPredMatrix[meanPredPeakIndCoeff!=0]
```

```
print(meanPredPeak)

meanPredPeakIndX = xPredMesh * meanPredPeakIndCoeff
meanPredPeakIndX = meanPredPeakIndX[meanPredPeakIndX!=0]

meanPredPeakIndY = yPredMesh * meanPredPeakIndCoeff
meanPredPeakIndY = meanPredPeakIndY[meanPredPeakIndY!=0]

meanPredPeakInd  = np.array([meanPredPeakIndX,meanPredPeakIndY])
print(meanPredPeakInd)

fig6= plt.figure('PredMatrix Indices',figsize=(4,4))
plt.plot(meanPredPeakIndX,meanPredPeakIndY,'o',c='b')
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.show()

# PredMatrix + 2 Sigma Peak Indices

meanPredP2Sigma = P2SigmaMeanPred
meanPredP2SigmaMatrix       = meanPredP2Sigma.reshape(discretization,discretization)
meanPredP2SigmaPeakIndCoeff = findPeaks(meanPredP2SigmaMatrix)
meanPredP2SigmaPeak         = meanPredP2SigmaMatrix[meanPredP2SigmaPeakIndCoeff!=0]
print(meanPredP2SigmaPeak)

meanPredP2SigmaPeakIndX = xPredMesh * meanPredP2SigmaPeakIndCoeff
meanPredP2SigmaPeakIndX = meanPredP2SigmaPeakIndX[meanPredP2SigmaPeakIndX!=0]

meanPredP2SigmaPeakIndY = yPredMesh * meanPredP2SigmaPeakIndCoeff
meanPredP2SigmaPeakIndY = meanPredP2SigmaPeakIndY[meanPredP2SigmaPeakIndY!=0]

meanPredP2SigmaPeakInd  = np.array([meanPredP2SigmaPeakIndX,meanPredP2SigmaPeakIndY])
print(meanPredP2SigmaPeakInd)

fig7= plt.figure('PredMatrix + 2 Sigma Peak Indices',figsize=(4,4))
plt.plot(meanPredP2SigmaPeakIndX,meanPredP2SigmaPeakIndY,'o',c='b')
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.show()

# VarMatrix Peak Indices

varPred = varPred
varPredMatrix       = varPred.reshape(discretization,discretization)
varPredPeakIndCoeff = findPeaks(varPredMatrix)
varPredPeak         = varPredMatrix[varPredPeakIndCoeff!=0]
print(varPredPeak)

varPredPeakIndX = xPredMesh * varPredPeakIndCoeff
varPredPeakIndX = varPredPeakIndX[varPredPeakIndX!=0]

varPredPeakIndY = yPredMesh * varPredPeakIndCoeff
varPredPeakIndY = varPredPeakIndY[varPredPeakIndY!=0]

varPredPeakInd  = np.array([varPredPeakIndX,varPredPeakIndY])
print(varPredPeakInd)


fig8= plt.figure('VarMatrix Peak Indices',figsize=(4,4))
plt.plot(varPredPeakIndX,varPredPeakIndY,'o',c='b')
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.show()
```

Appendix 3:

```
#### APPENDIX 3 #####

%matplotlib inline
%matplotlib widget
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
#from matplotlib import cm
import math
from scipy.optimize import minimize

# Input Values

xySensor  = np.array([[ 0.1,-0.9,0.2, 0.8,-0.6, 0.3,  0.5,-0.5,-0.01,-0.9,
                       5.10000000000000089e-01,  7.50000000000000000e-01,  -7.50000000000000000e-01,
1.79999999999999933e-01,
                        3.33333299999999833e-01, 8.78787879999999657e-01, 2.32323229999999917e-01,
-3.93939390000000004e-01],
               [0.05, 0.3,0.4,-0.3, 0.3,-0.2,-0.84,0.85,-0.76,-0.9,
                      -8.39999999999999689e-01, -6.99999999999999556e-01,  -6.99999999999999556e-01,
-1.00000000000000000e+00,
                      -7.77777779999999742e-01, -5.95959600000000338e-01, -5.15151519999999739e-01,
-2.52525249999999789e-01]])
outSensor                                                                                                    =
np.array([3.39382006,3.2073034,3.39965035,3.68810201,2.96941623,2.99495501,3.94274928,2.7968011,3.34929
734,3.91296165,
                        3.960708425707508251e+00, 3.915714809492213622e+00, 3.573178696741371230e+00,
3.806254007495223135e+00,
                        3.606670456681302372e+00, 3.718398992818654936e+00, 4.362107148258433043e+00,
3.453585989860598726e+00])

discretization = 100
xPred =  np.linspace(-1, 1, discretization)
yPred =  np.linspace(-1, 1, discretization)
xPredMesh, yPredMesh = np.meshgrid(xPred,yPred)
xyPred = (np.array([xPredMesh.flatten(),yPredMesh.flatten()]))


MEAN = np.mean(outSensor)
noise = 1e-8
tau = 0.3
l = 0.2

plt.figure(num='Data Points',figsize=(10,8))
fig1 = plt.axes(projection='3d')
fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='teal',label="Data Points",s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig1.set_zlabel("Sensor Output")
plt.legend()
plt.show()

# Function: Kernel
```

```
def sqexp(x, xp, tauin, lin):
    """Squared exponential kernel (1 dimensional)

    Inputs
    ------
    x : (N), array of multiple inputs
    xp: float

    Returns
    -------
    cov (N,) -- Covariance between each input at *x* and the function values at *x*
    """
    cov = tauin**2 * np.exp(-1/2 * (x - xp)**2 / lin**2)
    return cov

def sqexp2d(x, xp, tauin, lin):
    """Squared exponential kernel (2 dimensional) """
    xX   = x[0,:]
    xY   = x[1,:]
    xpX  = xp[0]
    xpY  = xp[1]
    tauX = tauin
    tauY = 1 # tauY=1 bc I only want one tau to tune
    sqexpX = sqexp(xX, xpX, tauX, lin)
    sqexpY = sqexp(xY, xpY, tauY, lin)
    cov = sqexpX*sqexpY
    return cov

# Function: Covariance

def build_covariance(x, xp,tauin, lin, kern):
    """Build a covariance matrix

    Inputs
    -------
    x: (N) array of inputs
    xp: (M) array of inptus
    kern: a function mapping inputs to covariance

    Outputs
    -------
    cov: (N, M) covariance matrix
    """
    out = np.zeros((x.shape[1], xp.shape[1]))
    for jj in range(xp.shape[1]):
        out[:, jj] = kern(x, xp[:,jj], tauin, lin)
    return out

# Function: GPR

def gpr(xtrain, ytrain, xpred, noise_var, mean_func, kernel):
    """Gaussian process regression Algorithm

    Inputs
    -------
    xtrain: (N, ) training inputs
    ytrain: (N, ) training outputs
    xpred:  (M, ) locations at which to make predictions
    noise_var: (N, ) noise at every training output
    mean_func: function to compute the prior mean
    kernel: covariance kernel

    Returns
    -------
```

```
    pred_mean : (M, ) predicted mean at prediction points
    pred_cov : (M, M) predicted covariance at the prediction points
    --
    """
    #xtrain  = np.transpose(xtrain)
    cov = build_covariance(xtrain, xtrain, tau, l, kernel)
    u, s, v = np.linalg.svd(cov)
    sqrtcov = np.dot(u, np.sqrt(np.diag(s)))

    noiseArray=(np.ones(np.size(xtrain[1,:])))*noise_var

    # pseudoinverse is better conditioned
    invcov = np.linalg.pinv(cov + np.diag(noiseArray))

    vec_pred = build_covariance(xpred, xtrain, tau, l, kernel)

    pred_mean = MEAN + np.dot(vec_pred, np.dot(invcov, ytrain - MEAN))
    cov_predict_pre = build_covariance(xpred, xpred, tau, l, kernel)
    cov_predict_up = np.dot(vec_pred, np.dot(invcov, vec_pred.T))
    pred_cov = cov_predict_pre - cov_predict_up
    #print(cov_predict_pre)

    return pred_mean, pred_cov

# GPR Results

meanPred, covPred = gpr(xySensor, outSensor, xyPred, noise, MEAN, sqexp2d)

xPredMeshF, yPredMeshF = xPredMesh.flatten(),yPredMesh.flatten()
plt.figure(num='mean and cov',figsize=(10,8))
fig1 = plt.axes(projection='3d')
fig1.scatter3D(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label='Predicted Mean')
fig1.scatter3D(xPredMeshF,yPredMeshF,np.diag(covPred),cmap='winter',label='Predicted Cov')
fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig1.set_zlabel("Sensor Output")
plt.legend()
plt.show()

# GPR Results

xPredMeshF, yPredMeshF = xPredMesh.flatten(),yPredMesh.flatten()
plt.figure(num='mean and cov',figsize=(10,8))
fig1 = plt.axes(projection='3d')
fig1.plot_trisurf(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label='Predicted Mean')
fig1.plot_trisurf(xPredMeshF,yPredMeshF,np.diag(covPred),cmap='winter',label='Predicted Cov')
fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig1.set_zlabel("Sensor Output")
plt.show()

# Function: ± 2 Sigma Pred Computation

def twoSigma(inputMean, inputCov):
    varPred    = np.array(np.diag(inputCov))
    stdDevPred = np.sqrt(varPred)
```

```
    P2SigmaPred = inputMean + (2*stdDevPred)
    N2SigmaPred = inputMean - (2*stdDevPred)

    plt.figure(num='Pred Mean  ± 2 Sigma Pred Computation',figsize=(10,8))
    ax = plt.axes(projection='3d')
    ax.scatter3D(xPredMeshF,yPredMeshF,P2SigmaPred,c='limegreen',label='Predicted Mean + 2 Sigma', alpha=.3)
    ax.scatter3D(xPredMeshF,yPredMeshF,meanPred,c='teal',label='Predicted Mean', alpha=.3)
    ax.scatter3D(xPredMeshF,yPredMeshF,N2SigmaPred,c='maroon',label='Predicted Mean - 2 Sigma', alpha=.3)
    locs, labels = plt.xticks()  # Get the current locations and labels.
    plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
    plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
    plt.xlabel("X Coordinate")
    plt.ylabel("Y Coordinate")
    ax.set_zlabel("Sensor Output")
    plt.legend()
    plt.show(ax)

    return(varPred,stdDevPred,P2SigmaPred,N2SigmaPred)


# ± 2 Sigma Pred Computation
varMeanPred,stdDevMeanPred, P2SigmaMeanPred, N2SigmaMeanPred = twoSigma(meanPred, covPred)

# STD Pred Plot

plt.figure(num='Pred Mean STD',figsize=(10,8))
fig3 = plt.axes(projection='3d')
fig3.plot_trisurf(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label='Predicted Mean', alpha=1)
fig3.plot_trisurf(xPredMeshF,yPredMeshF,P2SigmaMeanPred,cmap='cool',label='Predicted Mean + 2 Sigma', alpha=1)
fig3.plot_trisurf(xPredMeshF,yPredMeshF,N2SigmaMeanPred,cmap='cool',label='Predicted Mean - 2 Sigma', alpha=1)
fig3.plot_trisurf(xPredMeshF, yPredMeshF, stdDevMeanPred,cmap='magma',label='Predicted Points', alpha=1)
#fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.show()

# Function: HyperParameters

def likelihood(x):
    tau_l, l_l, noise_l = x
    noiseArray=(np.ones(np.size(xySensor[0,:])))*noise_l
    cov = build_covariance(xySensor, xySensor, tau_l, l_l, sqexp2d)
    invcov = np.linalg.pinv(cov + np.diag(noiseArray))
    likelihoodOut = 0.5 * np.dot((outSensor.T-MEAN), np.dot(invcov, (outSensor-MEAN)))\
                + 0.5 * np.log(np.linalg.det(cov + np.diag(noiseArray)))\
                + 0.5 * np.size(xySensor[1,:]) * np.log(2*np.pi)

    return likelihoodOut

# HyperParameters Results

noise = 1e-8
tau = 0.3
l = 0.2

xInit=np.array([tau, l, noise])
bounds = [(1e-15,None),(1e-15,None),(1e-15,None)]
minOut = minimize(likelihood, xInit, bounds=bounds, method='nelder-mead')
minOutX = np.array(minOut.x)
tau = minOutX[0]
l = minOutX[1]
```

```
noise = minOutX[2]
meanPred, covPred = gpr(xySensor, outSensor, xyPred, noise, MEAN, sqexp2d)
fig = plt.figure('Hyperparameter Pred Mean',figsize=(10,8))
fig4 = plt.axes(projection='3d')
fig4.scatter3D(xPredMeshF,yPredMeshF,meanPred,c='teal',label="Predicted Mean", alpha=0.2)
#fig4.scatter3D(xPredMeshF,yPredMeshF,np.diag(covPred),c='palevioletred',label="Predicted Var", alpha=0.2)
fig4.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100, alpha=0.2)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.legend()
plt.show()

print(minOutX)

# HyperParameters Results

fig = plt.figure('Hyperparameter Pred Mean',figsize=(10,8))
fig4 = plt.axes(projection='3d')
fig4.plot_trisurf(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label="Predicted Mean")
#fig4.plot_trisurf(xPredMeshF,yPredMeshF,np.diag(covPred),cmap='winter',label="Predicted Cov")
fig4.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
#plt.legend()
plt.show()

# Hyperparameter ± 2 Sigma Pred Computation
varPred, stdDevMeanPred, P2SigmaMeanPred, N2SigmaMeanPred = twoSigma(meanPred, covPred)


# Hyperparameter STD Pred Plot

plt.figure(num='Pred Mean STD',figsize=(10,8))
fig5 = plt.axes(projection='3d')
fig5.plot_trisurf(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label='Predicted Mean')
#fig3.plot_trisurf(xPredMeshF,yPredMeshF,P2SigmaMeanPred,cmap='cool',label='Predicted Mean + 2 Sigma')
#fig3.plot_trisurf(xPredMeshF,yPredMeshF,N2SigmaMeanPred,cmap='cool',label='Predicted Mean - 2 Sigma')
fig5.plot_trisurf(xPredMeshF, yPredMeshF, stdDevMeanPred,cmap='viridis',label='Std Dev')
#fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.show()

# Hyperparameter STD Pred Plot

plt.figure(num='Pred Mean STD',figsize=(10,8))
fig5 = plt.axes(projection='3d')
fig5.scatter3D(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label='Predicted Mean')
#fig3.plot_trisurf(xPredMeshF,yPredMeshF,P2SigmaMeanPred,cmap='cool',label='Predicted Mean + 2 Sigma')
#fig3.plot_trisurf(xPredMeshF,yPredMeshF,N2SigmaMeanPred,cmap='cool',label='Predicted Mean - 2 Sigma')
fig5.scatter3D(xPredMeshF, yPredMeshF, stdDevMeanPred,cmap='magma',label='Std Dev')
#fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
```

```
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.legend()
plt.show()

# Function : Finding indices of the peaks of a matrix

def findPeaks(inputMatrix):
    paddedMatrix = np.pad(inputMatrix,(1),'constant',constant_values=0)
    length1      = len(inputMatrix[0])
    length2      = len(inputMatrix[1])
    peakInd      = np.zeros((length1,length2))
    for xi in range(0,length1):
        for yi in range(0,length2):
            if paddedMatrix[xi+1,yi+1] > paddedMatrix[   xi , yi   ] and \
               paddedMatrix[xi+1,yi+1] > paddedMatrix[   xi , yi+1 ] and \
               paddedMatrix[xi+1,yi+1] > paddedMatrix[   xi , yi+2 ] and \
               paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+1 , yi   ] and \
               paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+1 , yi+2 ] and \
               paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+2 , yi   ] and \
               paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+2 , yi+1 ] and \
               paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+2 , yi+2 ]:

                peakInd[xi,yi] = 1

    return peakInd

# PredMatrix Peak Indices

meanPredMatrix       = meanPred.reshape(discretization,discretization)
meanPredPeakIndCoeff = findPeaks(meanPredMatrix)
meanPredPeak         = meanPredMatrix[meanPredPeakIndCoeff!=0]
print(meanPredPeak)

meanPredPeakIndX = xPredMesh * meanPredPeakIndCoeff
meanPredPeakIndX = meanPredPeakIndX[meanPredPeakIndX!=0]

meanPredPeakIndY = yPredMesh * meanPredPeakIndCoeff
meanPredPeakIndY = meanPredPeakIndY[meanPredPeakIndY!=0]

meanPredPeakInd  = np.array([meanPredPeakIndX,meanPredPeakIndY])
print(meanPredPeakInd)

fig6= plt.figure('PredMatrix Indices',figsize=(4,4))
plt.plot(meanPredPeakIndX,meanPredPeakIndY,'o',c='b')
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.show()

meanPredMaxInd = meanPredMatrix*meanPredPeakIndCoeff
MeanPredMax = meanPredMatrix[meanPredMaxInd!=0]
print(meanPredMatrix[meanPredPeakIndCoeff!=0])

print(meanPredPeakIndCoeff)

# PredMatrix + 2 Sigma Peak Indices

meanPredP2Sigma = P2SigmaMeanPred
meanPredP2SigmaMatrix       = meanPredP2Sigma.reshape(discretization,discretization)
meanPredP2SigmaPeakIndCoeff = findPeaks(meanPredP2SigmaMatrix)
meanPredP2SigmaPeak         = meanPredP2SigmaMatrix[meanPredP2SigmaPeakIndCoeff!=0]
print(meanPredP2SigmaPeak)

meanPredP2SigmaPeakIndX = xPredMesh * meanPredP2SigmaPeakIndCoeff
```

```
meanPredP2SigmaPeakIndX = meanPredP2SigmaPeakIndX[meanPredP2SigmaPeakIndX!=0]

meanPredP2SigmaPeakIndY = yPredMesh * meanPredP2SigmaPeakIndCoeff
meanPredP2SigmaPeakIndY = meanPredP2SigmaPeakIndY[meanPredP2SigmaPeakIndY!=0]

meanPredP2SigmaPeakInd  = np.array([meanPredP2SigmaPeakIndX,meanPredP2SigmaPeakIndY])
print(meanPredP2SigmaPeakInd)

fig7= plt.figure('PredMatrix + 2 Sigma Peak Indices',figsize=(4,4))
plt.plot(meanPredP2SigmaPeakIndX,meanPredP2SigmaPeakIndY,'o',c='b')
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.show()

# VarMatrix Peak Indices

#varPred = varPred
varPredMatrix       = varPred.reshape(discretization,discretization)
varPredPeakIndCoeff = findPeaks(varPredMatrix)
varPredPeak         = varPredMatrix[varPredPeakIndCoeff!=0]
print(varPredPeak)

varPredPeakIndX = xPredMesh * varPredPeakIndCoeff
varPredPeakIndX = varPredPeakIndX[varPredPeakIndX!=0]

varPredPeakIndY = yPredMesh * varPredPeakIndCoeff
varPredPeakIndY = varPredPeakIndY[varPredPeakIndY!=0]

varPredPeakInd  = np.array([varPredPeakIndX,varPredPeakIndY])
print(varPredPeakInd)

fig8= plt.figure('VarMatrix Peak Indices',figsize=(4,4))
plt.plot(varPredPeakIndX,varPredPeakIndY,'o',c='b')
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.show()
```

Appendix 4:

```
#### APPENDIX 4 #####

%matplotlib inline
%matplotlib widget
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import math
from scipy.optimize import minimize

# Input Values

xySensor  = np.array([[ 0.1,-0.9,0.2, 0.8,-0.6, 0.3,  0.5,-0.5,-0.01,-0.9,
                       5.100000000000000089e-01,  7.500000000000000000e-01,  -7.500000000000000000e-01,
1.799999999999999933e-01,
                       3.333333299999999833e-01,  8.787878799999999657e-01,   2.323232299999999917e-01,
-3.939393900000000004e-01,
```

```
                      6.19999999999999956e-01,  3.49999999999999778e-01,  -1.49999999999999944e-01,
-5.00000000000000278e-02],
               [0.05, 0.3,0.4,-0.3, 0.3,-0.2,-0.84,0.85,-0.76,-0.9,
                      -8.39999999999999689e-01, -6.99999999999999556e-01,  -6.99999999999999556e-01,
-1.00000000000000000e+00,
                     -7.77777799999999742e-01, -5.95959600000000338e-01,  -5.15151519999999739e-01,
-2.52525249999999789e-01,
                      -8.00000000000000444e-01, -5.00000000000000000e-01,   5.00000000000000000e-01,
-4.50000000000000111e-01]])

outSensor                                                                                                  =
np.array([3.39382006,3.2073034,3.39965035,3.68810201,2.96941623,2.99495501,3.94274928,2.7968011,3.34929
734,3.91296165,
                       3.960708425707508251e+00, 3.915714809492213622e+00, 3.573178696741371230e+00,
3.806254007495223135e+00,
                       3.606670456681302372e+00, 3.718398992818654936e+00, 4.362107148258433043e+00,
3.453585989860598726e+00,
                       4.075788006170817823e+00, 4.188583218050431434e+00, 4.824642660411089246e+00,
4.055704943752908243e+00])

discretization = 100
xPred =  np.linspace(-1, 1, discretization)
yPred =  np.linspace(-1, 1, discretization)
xPredMesh, yPredMesh = np.meshgrid(xPred,yPred)
xyPred = (np.array([xPredMesh.flatten(),yPredMesh.flatten()]))


MEAN = np.mean(outSensor)
noise = 1e-8
tau = 0.3
l = 0.2

plt.figure(num='Data Points',figsize=(10,8))
fig1 = plt.axes(projection='3d')
fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='teal',label="Data Points",s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig1.set_zlabel("Sensor Output")
plt.legend()
plt.show()

# Function: Kernel

def sqexp(x, xp, tauin, lin):
    """Squared exponential kernel (1 dimensional)

    Inputs
    ------
    x : (N), array of multiple inputs
    xp: float

    Returns
    -------
    cov (N,) -- Covariance between each input at *x* and the function values at *x*
    """
    cov = tauin**2 * np.exp(-1/2 * (x - xp)**2 / lin**2)
    return cov

def sqexp2d(x, xp, tauin, lin):
    """Squared exponential kernel (2 dimensional) """
    xX   = x[0,:]
```

```
    xY   = x[1,:]
    xpX  = xp[0]
    xpY  = xp[1]
    tauX = tauin
    tauY = 1 # tauY=1 bc I only want one tau to tune
    sqexpX = sqexp(xX, xpX, tauX, lin)
    sqexpY = sqexp(xY, xpY, tauY, lin)
    cov = sqexpX*sqexpY
    return cov

# Function: Covariance

def build_covariance(x, xp,tauin, lin, kern):
    """Build a covariance matrix

    Inputs
    -------
    x: (N) array of inputs
    xp: (M) array of inptus
    kern: a function mapping inputs to covariance

    Outputs
    -------
    cov: (N, M) covariance matrix
    """
    out = np.zeros((x.shape[1], xp.shape[1]))
    for jj in range(xp.shape[1]):
        out[:, jj] = kern(x, xp[:,jj], tauin, lin)
    return out

# Function: GPR

def gpr(xtrain, ytrain, xpred, noise_var, mean_func, kernel):
    """Gaussian process regression Algorithm

    Inputs
    -------
    xtrain: (N, ) training inputs
    ytrain: (N, ) training outputs
    xpred:  (M, ) locations at which to make predictions
    noise_var: (N, ) noise at every training output
    mean_func: function to compute the prior mean
    kernel: covariance kernel

    Returns
    -------
    pred_mean : (M, ) predicted mean at prediction points
    pred_cov : (M, M) predicted covariance at the prediction points
    --
    """
    #xtrain  = np.transpose(xtrain)
    cov = build_covariance(xtrain, xtrain, tau, l, kernel)
    u, s, v = np.linalg.svd(cov)
    sqrtcov = np.dot(u, np.sqrt(np.diag(s)))

    noiseArray=(np.ones(np.size(xtrain[1,:])))*noise_var

    # pseudoinverse is better conditioned
    invcov = np.linalg.pinv(cov + np.diag(noiseArray))

    vec_pred = build_covariance(xpred, xtrain, tau, l, kernel)

    pred_mean = MEAN + np.dot(vec_pred, np.dot(invcov, ytrain - MEAN))
    cov_predict_pre = build_covariance(xpred, xpred, tau, l, kernel)
```

```
    cov_predict_up = np.dot(vec_pred, np.dot(invcov, vec_pred.T))
    pred_cov = cov_predict_pre - cov_predict_up
    #print(cov_predict_pre)

    return pred_mean, pred_cov

# GPR Results

meanPred, covPred = gpr(xySensor, outSensor, xyPred, noise, MEAN, sqexp2d)

xPredMeshF, yPredMeshF = xPredMesh.flatten(),yPredMesh.flatten()
plt.figure(num='mean and cov',figsize=(10,8))
fig1 = plt.axes(projection='3d')
fig1.scatter3D(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label='Predicted Mean')
fig1.scatter3D(xPredMeshF,yPredMeshF,np.diag(covPred),cmap='winter',label='Predicted Cov')
fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig1.set_zlabel("Sensor Output")
plt.legend()
plt.show()

# GPR Results

xPredMeshF, yPredMeshF = xPredMesh.flatten(),yPredMesh.flatten()
plt.figure(num='mean and cov',figsize=(10,8))
fig1 = plt.axes(projection='3d')
fig1.plot_trisurf(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label='Predicted Mean')
#fig1.plot_trisurf(xPredMeshF,yPredMeshF,np.diag(covPred),cmap='winter',label='Predicted Cov')
fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
fig1.set_zlabel("Sensor Output")
plt.show()

# Function: ± 2 Sigma Pred Computation

def twoSigma(inputMean, inputCov):
    varPred    = np.array(np.diag(inputCov))
    stdDevPred = np.sqrt(varPred)
    P2SigmaPred = inputMean + (2*stdDevPred)
    N2SigmaPred = inputMean - (2*stdDevPred)
    plt.figure(num='Pred Mean  ± 2 Sigma Pred Computation',figsize=(10,8))
    ax = plt.axes(projection='3d')
    ax.scatter3D(xPredMeshF,yPredMeshF,P2SigmaPred,c='limegreen',label='Predicted Mean + 2 Sigma', alpha=.3)
    ax.scatter3D(xPredMeshF,yPredMeshF,meanPred,c='teal',label='Predicted Mean', alpha=.3)
    ax.scatter3D(xPredMeshF,yPredMeshF,N2SigmaPred,c='maroon',label='Predicted Mean - 2 Sigma', alpha=.3)
    locs, labels = plt.xticks()  # Get the current locations and labels.
    plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
    plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
    plt.xlabel("X Coordinate")
    plt.ylabel("Y Coordinate")
    ax.set_zlabel("Sensor Output")
    plt.legend()
    plt.show(ax)

    return(varPred,stdDevPred,P2SigmaPred,N2SigmaPred)
```

```
# ± 2 Sigma Pred Computation
varMeanPred,stdDevMeanPred, P2SigmaMeanPred, N2SigmaMeanPred = twoSigma(meanPred, covPred)


# STD Pred Plot

plt.figure(num='Pred Mean STD',figsize=(10,8))
fig3 = plt.axes(projection='3d')
fig3.scatter3D(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label='Predicted Mean')
fig3.scatter3D(xPredMeshF,yPredMeshF,P2SigmaMeanPred,cmap='cool',label='Predicted Mean + 2 Sigma')
fig3.scatter3D(xPredMeshF,yPredMeshF,N2SigmaMeanPred,cmap='cool',label='Predicted Mean - 2 Sigma')
fig3.scatter3D(xPredMeshF, yPredMeshF, stdDevMeanPred,cmap='magma',label="Predicted Points")
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.legend()
plt.show()

# Function: HyperParameters

def likelihood(x):
    tau_l, l_l, noise_l = x
    noiseArray=(np.ones(np.size(xySensor[0,:])))*noise_l
    cov = build_covariance(xySensor, xySensor, tau_l, l_l, sqexp2d)
    invcov = np.linalg.pinv(cov + np.diag(noiseArray))
    likelihoodOut = 0.5 * np.dot((outSensor.T-MEAN), np.dot(invcov, (outSensor-MEAN)))\
          + 0.5 * np.log(np.linalg.det(cov + np.diag(noiseArray)))\
          + 0.5 * np.size(xySensor[1,:]) * np.log(2*np.pi)
    return likelihoodOut

# HyperParameters Results

noise = 1e-8
tau = 0.3
l = 0.2

xInit=np.array([tau, l, noise])
bounds = [(1e-15,None),(1e-15,None),(1e-15,None)]
minOut = minimize(likelihood, xInit, bounds=bounds, method='nelder-mead')
minOutX = np.array(minOut.x)
tau = minOutX[0]
l = minOutX[1]
noise = minOutX[2]
meanPred, covPred = gpr(xySensor, outSensor, xyPred, noise, MEAN, sqexp2d)

fig = plt.figure('Hyperparameter Pred Mean',figsize=(10,8))
fig4 = plt.axes(projection='3d')
fig4.scatter3D(xPredMeshF,yPredMeshF,meanPred,c='teal',label="Predicted Mean", alpha=0.2)
fig4.scatter3D(xPredMeshF,yPredMeshF,np.diag(covPred),c='palevioletred',label="Predicted Var", alpha=0.2)
fig4.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.legend()
plt.show()

print(minOutX)

# HyperParameters Results
```

```
fig = plt.figure('Hyperparameter Pred Mean',figsize=(10,8))
fig4 = plt.axes(projection='3d')
fig4.plot_trisurf(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label="Predicted Mean")
fig4.plot_trisurf(xPredMeshF,yPredMeshF,np.diag(covPred),cmap='winter',label="Predicted Cov")
fig4.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.show()

# Hyperparameter ± 2 Sigma Pred Computation
varPred, stdDevMeanPred, P2SigmaMeanPred, N2SigmaMeanPred = twoSigma(meanPred, covPred)

# STD Pred Plot

plt.figure(num='Pred Mean STD',figsize=(10,8))
fig3 = plt.axes(projection='3d')
fig3.plot_trisurf(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label='Predicted Mean', alpha=1)
fig3.plot_trisurf(xPredMeshF, yPredMeshF, stdDevMeanPred,cmap='viridis',label='Predicted Points', alpha=1)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.show()

# Hyperparameter STD Pred Plot

plt.figure(num='Pred Mean STD',figsize=(10,8))
fig5 = plt.axes(projection='3d')
fig5.scatter3D(xPredMeshF,yPredMeshF,meanPred,cmap='winter',label='Predicted Mean')
#fig3.plot_trisurf(xPredMeshF,yPredMeshF,P2SigmaMeanPred,cmap='cool',label='Predicted Mean + 2 Sigma')
#fig3.plot_trisurf(xPredMeshF,yPredMeshF,N2SigmaMeanPred,cmap='cool',label='Predicted Mean - 2 Sigma')
fig5.scatter3D(xPredMeshF, yPredMeshF, stdDevMeanPred,cmap='magma',label='Std Dev')
#fig1.scatter3D(xySensor[0,:], xySensor[1,:], outSensor, c='k',label="Predicted Points", s=100)
locs, labels = plt.xticks()  # Get the current locations and labels.
plt.xticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.yticks(np.arange(-1, 1.25, step=0.25))  # Set label locations.
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.legend()
plt.show()

# Function : Finding indices of the peaks of a matrix

def findPeaks(inputMatrix):
   paddedMatrix = np.pad(inputMatrix,(1),'constant',constant_values=0)
   length1      = len(inputMatrix[0])
   length2      = len(inputMatrix[1])
   peakInd      = np.zeros((length1,length2))
   for xi in range(0,length1):
      for yi in range(0,length2):
         if paddedMatrix[xi+1,yi+1] > paddedMatrix[   xi , yi   ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[   xi , yi+1 ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[   xi , yi+2 ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+1 , yi   ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+1 , yi+2 ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+2 , yi   ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+2 , yi+1 ] and \
            paddedMatrix[xi+1,yi+1] > paddedMatrix[ xi+2 , yi+2 ]:
             peakInd[xi,yi] = 1
```

```
    return peakInd

# PredMatrix Peak Indices

meanPredMatrix       = meanPred.reshape(discretization,discretization)
meanPredPeakIndCoeff = findPeaks(meanPredMatrix)
meanPredPeak         = meanPredMatrix[meanPredPeakIndCoeff!=0]
print(meanPredPeak)

meanPredPeakIndX = xPredMesh * meanPredPeakIndCoeff
meanPredPeakIndX = meanPredPeakIndX[meanPredPeakIndX!=0]

meanPredPeakIndY = yPredMesh * meanPredPeakIndCoeff
meanPredPeakIndY = meanPredPeakIndY[meanPredPeakIndY!=0]

meanPredPeakInd  = np.array([meanPredPeakIndX,meanPredPeakIndY])
print(meanPredPeakInd)

fig6= plt.figure('PredMatrix Indices',figsize=(4,4))
plt.plot(meanPredPeakIndX,meanPredPeakIndY,'o',c='b')
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.show()

meanPredMaxInd = meanPredMatrix*meanPredPeakIndCoeff
MeanPredMax = meanPredMatrix[meanPredMaxInd!=0]
print(meanPredMatrix[meanPredPeakIndCoeff!=0])
print(meanPredPeakIndX)
print(meanPredPeakIndY)

# PredMatrix + 2 Sigma Peak Indices

meanPredP2Sigma = P2SigmaMeanPred
meanPredP2SigmaMatrix       = meanPredP2Sigma.reshape(discretization,discretization)
meanPredP2SigmaPeakIndCoeff = findPeaks(meanPredP2SigmaMatrix)
meanPredP2SigmaPeak         = meanPredP2SigmaMatrix[meanPredP2SigmaPeakIndCoeff!=0]
print(meanPredP2SigmaPeak)

meanPredP2SigmaPeakIndX = xPredMesh * meanPredP2SigmaPeakIndCoeff
meanPredP2SigmaPeakIndX = meanPredP2SigmaPeakIndX[meanPredP2SigmaPeakIndX!=0]

meanPredP2SigmaPeakIndY = yPredMesh * meanPredP2SigmaPeakIndCoeff
meanPredP2SigmaPeakIndY = meanPredP2SigmaPeakIndY[meanPredP2SigmaPeakIndY!=0]

meanPredP2SigmaPeakInd  = np.array([meanPredP2SigmaPeakIndX,meanPredP2SigmaPeakIndY])
print(meanPredP2SigmaPeakInd)

fig7= plt.figure('PredMatrix + 2 Sigma Peak Indices',figsize=(4,4))
plt.plot(meanPredP2SigmaPeakIndX,meanPredP2SigmaPeakIndY,'o',c='b')
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.show()

# VarMatrix Peak Indices

#varPred = varPred
varPredMatrix       = varPred.reshape(discretization,discretization)
varPredPeakIndCoeff = findPeaks(varPredMatrix)
varPredPeak         = varPredMatrix[varPredPeakIndCoeff!=0]
print(varPredPeak)

varPredPeakIndX = xPredMesh * varPredPeakIndCoeff
varPredPeakIndX = varPredPeakIndX[varPredPeakIndX!=0]
```

```
varPredPeakIndY = yPredMesh * varPredPeakIndCoeff
varPredPeakIndY = varPredPeakIndY[varPredPeakIndY!=0]

varPredPeakInd  = np.array([varPredPeakIndX,varPredPeakIndY])
print(varPredPeakInd)


fig8= plt.figure('VarMatrix Peak Indices',figsize=(4,4))
plt.plot(varPredPeakIndX,varPredPeakIndY,'o',c='b')
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.show()
```