Q4

Part 1

The confusion matrix for SVM via cubic B-spline(implemented in Matlab) are shown as follows.

```
>> Q4P1

table =

     5     0     0     0
     0     8     0     1
     0     1     1     2
     0     1     0    11
```

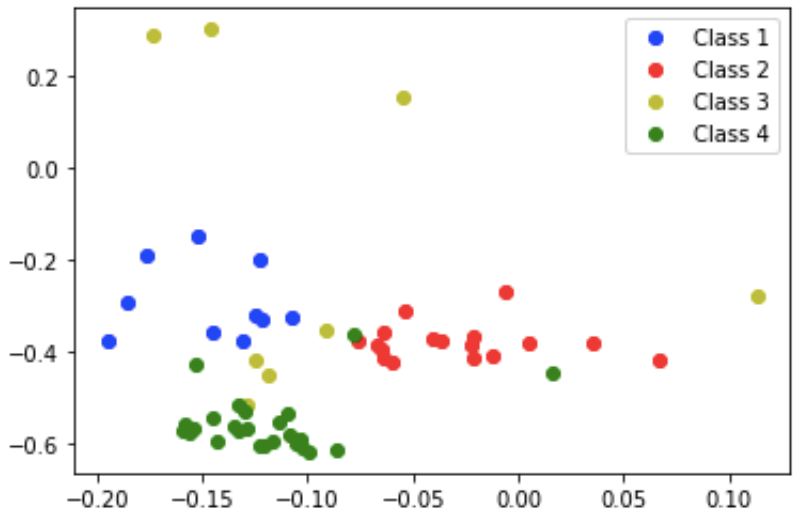
The accuracy of SVM via cubic B-spline with Matlab is 0.83.

The confusion matrix and accuracy for cubic B-spline(implemented in Python) are shown as follows.

```
Confusion Matrix - cubic B-spline
          Predicted 1  Predicted 2  Predicted 3  Predicted 4
Class 1             0            4            0            2
Class 2             0            6            0            0
Class 3             0            4            0            0
Class 4             0            1            0           13
Accuracy:0.6333333333333333
```

Part 2

First off, I used FPCA to extract the top two FPC scores and visualize the data. The plot and the confusion matrix of this classifier(SVM with the first top FPC scores) are shown as follows.




```
Confusion Matrix - FPCA
          Predicted 1  Predicted 2  Predicted 3  Predicted 4
Class 1             4            0            0            0
Class 2             0            9            0            0
Class 3             0            0            1            2
Class 4             0            1            0           13
Accuracy:0.9
```

The accuracies and the confusion matrices of the four classifiers are as follows. Please scroll down to view the code and output, or check out the code file uploaded separately.

Using the 2 harmonic-

```
Confusion Matrix - FPCA
          Predicted 1  Predicted 2  Predicted 3  Predicted 4
Class 1             0            0            0            4
Class 2             0            8            1            0
Class 3             0            1            0            2
Class 4             0            2            0           12
Accuracy:0.6666666666666666
```

Using the 5 harmonic-

```
Confusion Matrix - FPCA
          Predicted 1  Predicted 2  Predicted 3  Predicted 4
Class 1             1            1            0            2
Class 2             3            4            0            2
Class 3             2            0            0            1
Class 4             0            3            0           11
Accuracy:0.5333333333333333
```

Using the 8 harmonic-

```
Confusion Matrix - FPCA
          Predicted 1  Predicted 2  Predicted 3  Predicted 4
Class 1             0            4            0            0
Class 2             0            5            0            4
Class 3             0            2            0            1
Class 4             0            4            0           10
Accuracy:0.5
```

Using the 10 harmonic-

```
Confusion Matrix - FPCA
          Predicted 1  Predicted 2  Predicted 3  Predicted 4
Class 1             3            0            1            0
Class 2             1            4            0            4
Class 3             1            1            1            0
Class 4             0            1            1           12
Accuracy:0.6666666666666666
```

In [131]:

```
#import modules

from scipy.ndimage import gaussian_filter
from matplotlib import cm
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
from scipy.interpolate import BSpline
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
import matplotlib.pyplot as plt
%matplotlib inline

#helper function

def BSplineBasis(x: np.array, knots: np.array, degree: int) -> np.arra
y:
    '''Return B-Spline basis. Python equivalent to bs in R or the spma
k/spval combination in MATLAB.
    This function acts like the R command bs(x,knots=knots,degree=degr
ee, intercept=False)
    Arguments:
        x: Points to evaluate spline on, sorted increasing
        knots: Spline knots, sorted increasing
        degree: Spline degree.
    Returns:
        B: Array of shape (x.shape[0], len(knots)+degree+1).
    Note that a spline has len(knots)+degree coefficients. However, be
cause the intercept is missing
```

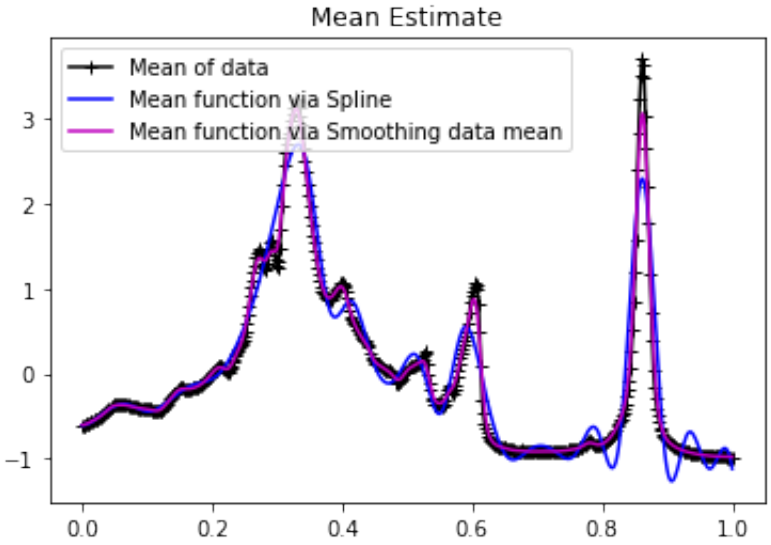
```
    you will need to remove the last 2 columns. It's being kept this w
ay to retain compatibility with
    both the matlab spmak way and how R's bs works.

    If K = length(knots) (includes boundary knots)
    Mapping this to R's bs: (Props to Nate Bartlett )
    bs(x,knots,degree,intercept=T)[,2:K+degree] is same as BSplineBasi
s(x,knots,degree)[:,:-2]
    BF = bs(x,knots,degree,intercept=F) drops the first column so BF[,
1:K+degree] == BSplineBasis(x,knots,degree)[:,:-2]
    '''
    nKnots = knots.shape[0]
    lo = min(x[0], knots[0])
    hi = max(x[-1], knots[-1])
    augmented_knots = np.append(
        np.append([lo]*degree, knots), [hi]*degree)
    DOF = nKnots + degree +1 # DOF = K+M, M = degree+1
    spline = BSpline(augmented_knots, np.eye(DOF),
                     degree, extrapolate=False)
    B = spline(x)
    return B
```

In [130]:
```
#part 1
data = pd.read_csv('Question4.csv',header=None)
y_true = np.array(data.iloc[:,-1]) #of shape(60)
X = data.iloc[:,:-1].to_numpy() #of shape(60,570)
x = np.linspace(0,1,570)
knots = np.linspace(0, 1, 70)
B = BSplineBasis(x, knots, degree=3)[:,:-2]
Bcoef = np.linalg.lstsq(B, X.T)[0].T #shape(60,72) #Bcoef is the extra
cted features of 72 dims
indices = np.arange(60)
X_trg, X_tst, Y_trg, Y_tst, i_trg, i_tst = train_test_split(
    X, y_true, indices, train_size=0.5, random_state=111) #111
clf = SVC(gamma='auto',kernel='sigmoid')
clf.fit(Bcoef[i_trg,:],Y_trg)
pred = clf.predict(Bcoef[i_tst,:])
conf = confusion_matrix(Y_tst, pred)
conf = pd.DataFrame(conf, index=['Class 1', 'Class 2', 'Class 3', 'Cla
ss 4'], columns=[
                    'Predicted 1', 'Predicted 2', 'Predicted 3', 'Pred
icted 4'])
print('Confusion Matrix - cubic B-spline\n', conf)
print(f'Accuracy:{clf.score(Bcoef[i_tst,:],Y_tst)}')
```

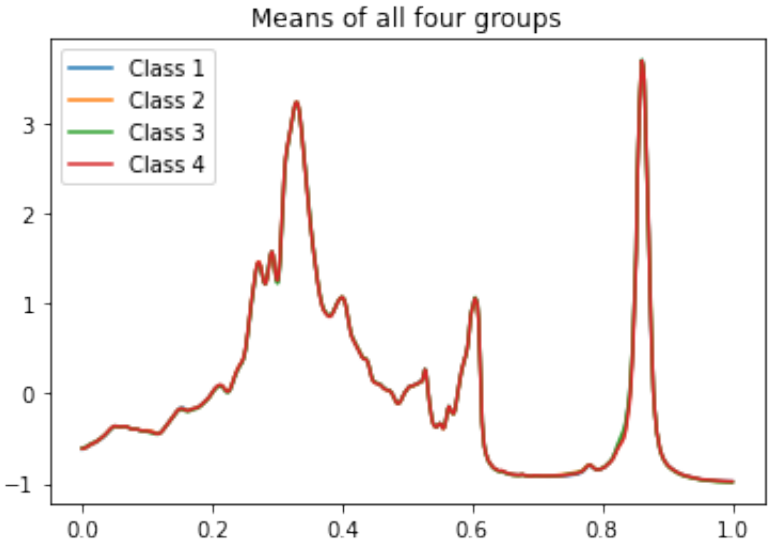
```
Confusion Matrix - cubic B-spline
          Predicted 1  Predicted 2  Predicted 3  Predicted 4
Class 1             0            4            0            2
Class 2             0            6            0            0
Class 3             0            4            0            0
Class 4             0            1            0           13
Accuracy:0.6333333333333333

/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:8: Futu
reWarning: `rcond` parameter will change to the default of machine p
recision times ``max(M, N)`` where M and N are the input matrix dime
nsions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
```

In [66]:
```
#part 2
#fpca for feature extraction

data = pd.read_csv('Question4.csv',header=None)
y_true = np.array(data.iloc[:,-1]) #of shape(60)
X = data.iloc[:,:-1].to_numpy() #of shape(60,570)
indices = np.arange(60)
g1 = X[y_true==1,:]
g2 = X[y_true==2,:]
g3 = X[y_true==3,:]
g4 = X[y_true==4,:]
m1= g1.mean(0)
m2= g2.mean(0)
m3= g3.mean(0)
m4= g4.mean(0)
X_trg, X_tst, Y_trg, Y_tst, i_trg, i_tst = train_test_split(
    X, y_true, indices, train_size=0.5, random_state=123)

x = np.linspace(0,1,570) #explanatory variables x
knots = np.linspace(0, 1, 30)
B = BSplineBasis(x, knots, degree=3)[:,:-2] #basis matrix B of shape(5
70,10)

#step 1
#estimate mean function via one dimensional kernel regression
B_stacked = np.tile(B.T, 60).T
X_stacked = X.ravel()
beta = np.linalg.lstsq(B_stacked, X_stacked)[0]
mu_hat = B.dot(beta)  # Mean function via B-Spline #of shape(570,1)
mu_hat2 = gaussian_filter(X.mean(0),3) #3 is the standard deviation fo
r gaussian kernel

plt.plot(x, X.mean(0), 'k+-', label='Mean of data')
plt.plot(x, mu_hat, 'b', label='Mean function via Spline')
plt.plot(x, mu_hat2, 'm', label='Mean function via Smoothing data mean
')
plt.legend()
plt.title("Mean Estimate")
plt.show()
```

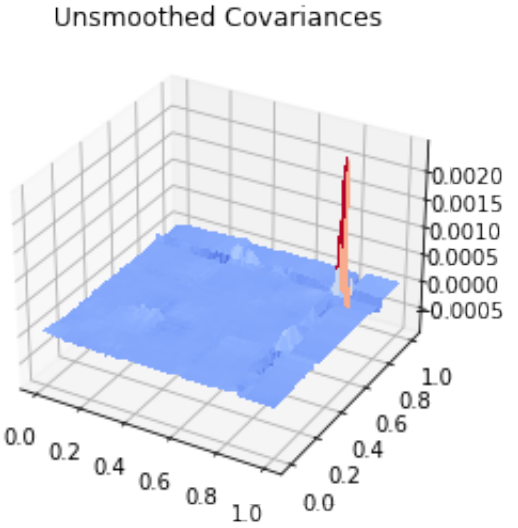
```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:27: Fut
ureWarning: `rcond` parameter will change to the default of machine
precision times ``max(M, N)`` where M and N are the input matrix dim
ensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
```

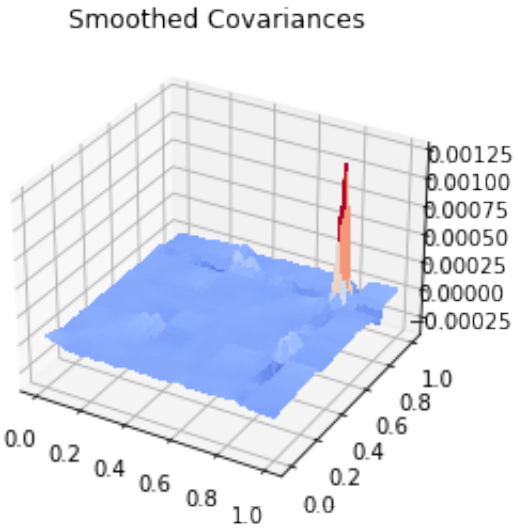


In [67]:
```
plt.plot(x, m1,  label='Class 1')
plt.plot(x, m2,  label='Class 2')
plt.plot(x, m3,  label='Class 3')
plt.plot(x, m4,  label='Class 4')
plt.legend()
plt.title("Means of all four groups")
plt.show()
```



In [75]:
```
#step 2
#estimate a smoothed functional covariance surface via two dimensional
kernel regression

diffs = X-mu_hat #of shape(60,570)
Cov = np.cov(diffs.T) #np.cov takes input with shape(num of variables,
num of observations)
grids = np.meshgrid(x, x)
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(grids[0], grids[1], Cov, cmap=cm.coolwarm,
                linewidth=0, antialiased=False)
plt.title('Unsmoothed Covariances')
plt.show()
# additional Cov smoothing:
Cov = gaussian_filter(Cov, sigma=7)
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(grids[0], grids[1], Cov, cmap=cm.coolwarm,
                linewidth=0, antialiased=False)
plt.title('Smoothed Covariances')
plt.show()
```

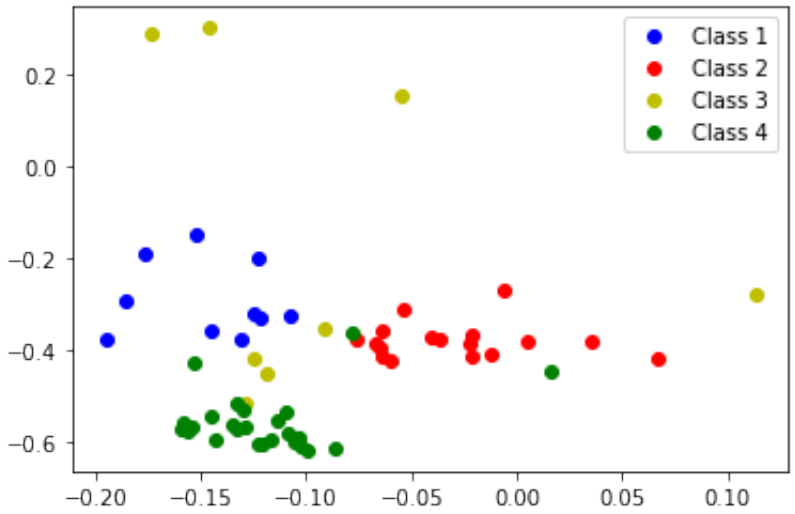

Unsmoothed Covariances
0.0020
0.0015
0.0010
0.0005
0.0000
−0.0005
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


Smoothed Covariances
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
−0.00025
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

In [76]:
```
#step 3
#solve eigen-functions

l, psi = np.linalg.eigh(Cov) #eigen-values and eigen-functions in asce
nding order
PC2 = psi[:, -2]
PC5 = psi[:, -5]
PC8 = psi[:, -8]
PC10 = psi[:, -10]
PCs = np.column_stack([PC2,PC5,PC8,PC10])
FPC_scores = diffs.dot(PCs) #principal components of the original data
matrix

clf = SVC()
clf.fit(FPC_scores[i_trg, :], Y_trg)
pred = clf.predict(FPC_scores[i_tst, :])
conf = confusion_matrix(Y_tst, pred)
conf = pd.DataFrame(conf, index=['Class 1', 'Class 2', 'Class 3', 'Cla
ss 4'], columns=[
                    'Predicted 1', 'Predicted 2', 'Predicted 3', 'Pred
icted 4'])
print('Confusion Matrix - FPCA\n', conf)
print(f'Accuracy:{clf.score(FPC_scores[i_tst, :],Y_tst)}')
```

```
Confusion Matrix - FPCA
          Predicted 1  Predicted 2  Predicted 3  Predicted 4
Class 1            0            0            0            4
Class 2            0            9            0            0
Class 3            0            1            0            2
Class 4            0            1            0           13
Accuracy:0.7333333333333333
```

In [77]:
```
PCs = psi[:, -2:]
FPC_scores = diffs.dot(PCs) #principal components of the original data matrix
plt.plot(FPC_scores[y_true==1,0],FPC_scores[y_true==1,1],'bo',label='Class 1')
plt.plot(FPC_scores[y_true==2,0],FPC_scores[y_true==2,1],'ro',label='Class 2')
plt.plot(FPC_scores[y_true==3,0],FPC_scores[y_true==3,1],'yo',label='Class 3')
plt.plot(FPC_scores[y_true==4,0],FPC_scores[y_true==4,1],'go',label='Class 4')
plt.legend()
plt.show()
clf = SVC()
clf.fit(FPC_scores[i_trg,:], Y_trg)
pred = clf.predict(FPC_scores[i_tst,:])
conf = confusion_matrix(Y_tst, pred)
conf = pd.DataFrame(conf, index=['Class 1', 'Class 2', 'Class 3', 'Class 4'], columns=[
                    'Predicted 1', 'Predicted 2', 'Predicted 3', 'Predicted 4'])
print('Confusion Matrix - FPCA\n', conf)
print(f'Accuracy:{clf.score(FPC_scores[i_tst, :],Y_tst)}')
```




```
Confusion Matrix - FPCA
          Predicted 1  Predicted 2  Predicted 3  Predicted 4
Class 1             4            0            0            0
Class 2             0            9            0            0
Class 3             0            0            1            2
Class 4             0            1            0           13
Accuracy:0.9
```

In [78]:
```
l, psi = np.linalg.eigh(Cov) #eigen-values and eigen-functions in ascending order
PC2 = psi[:, -2]
FPC_scores = diffs.dot(PC2).reshape(60,1) #principal components of the original data matrix
clf = SVC(gamma='scale',kernel='rbf')
clf.fit(FPC_scores[i_trg, :], Y_trg)
pred = clf.predict(FPC_scores[i_tst, :])
conf = confusion_matrix(Y_tst, pred)
conf = pd.DataFrame(conf, index=['Class 1', 'Class 2', 'Class 3', 'Class 4'], columns=[
                    'Predicted 1', 'Predicted 2', 'Predicted 3', 'Predicted 4'])
print('Confusion Matrix - FPCA\n', conf)
print(f'Accuracy:{clf.score(FPC_scores[i_tst, :],Y_tst)}')
```

```
Confusion Matrix - FPCA
          Predicted 1  Predicted 2  Predicted 3  Predicted 4
Class 1            0            0            0            4
Class 2            0            8            1            0
Class 3            0            1            0            2
Class 4            0            2            0           12
Accuracy:0.6666666666666666
```

In [79]:
```
l, psi = np.linalg.eigh(Cov) #eigen-values and eigen-functions in ascending order
PC5 = psi[:, -5]
FPC_scores = diffs.dot(PC5).reshape(60,1) #principal components of the original data matrix
clf = SVC()
clf.fit(FPC_scores[i_trg, :], Y_trg)
pred = clf.predict(FPC_scores[i_tst, :])
conf = confusion_matrix(Y_tst, pred)
conf = pd.DataFrame(conf, index=['Class 1', 'Class 2', 'Class 3', 'Class 4'], columns=[
                    'Predicted 1', 'Predicted 2', 'Predicted 3', 'Predicted 4'])
print('Confusion Matrix - FPCA\n', conf)
print(f'Accuracy:{clf.score(FPC_scores[i_tst, :],Y_tst)}')
```

```
Confusion Matrix - FPCA
          Predicted 1  Predicted 2  Predicted 3  Predicted 4
Class 1            1            1            0            2
Class 2            3            4            0            2
Class 3            2            0            0            1
Class 4            0            3            0           11
Accuracy:0.5333333333333333
```

In [80]:
```
l, psi = np.linalg.eigh(Cov) #eigen-values and eigen-functions in ascending order
PC8 = psi[:, -8]
FPC_scores = diffs.dot(PC8).reshape(60,1) #principal components of the original data matrix
clf = SVC()
clf.fit(FPC_scores[i_trg, :], Y_trg)
pred = clf.predict(FPC_scores[i_tst, :])
conf = confusion_matrix(Y_tst, pred)
conf = pd.DataFrame(conf, index=['Class 1', 'Class 2', 'Class 3', 'Class 4'], columns=[
                    'Predicted 1', 'Predicted 2', 'Predicted 3', 'Predicted 4'])
print('Confusion Matrix - FPCA\n', conf)
print(f'Accuracy:{clf.score(FPC_scores[i_tst, :],Y_tst)}')
```

```
Confusion Matrix - FPCA
          Predicted 1  Predicted 2  Predicted 3  Predicted 4
Class 1            0            4            0            0
Class 2            0            5            0            4
Class 3            0            2            0            1
Class 4            0            4            0           10
Accuracy:0.5
```

In [81]:
```
l, psi = np.linalg.eigh(Cov) #eigen-values and eigen-functions in ascending order
PC10 = psi[:, -10]
FPC_scores = diffs.dot(PC10).reshape(60,1) #principal components of the original data matrix
clf = SVC()
clf.fit(FPC_scores[i_trg, :], Y_trg)
pred = clf.predict(FPC_scores[i_tst, :])
conf = confusion_matrix(Y_tst, pred)
conf = pd.DataFrame(conf, index=['Class 1', 'Class 2', 'Class 3', 'Class 4'], columns=[
                    'Predicted 1', 'Predicted 2', 'Predicted 3', 'Predicted 4'])
print('Confusion Matrix - FPCA\n', conf)
print(f'Accuracy:{clf.score(FPC_scores[i_tst, :],Y_tst)}')
```

```
Confusion Matrix - FPCA
          Predicted 1  Predicted 2  Predicted 3  Predicted 4
Class 1            3            0            1            0
Class 2            1            4            0            4
Class 3            1            1            1            0
Class 4            0            1            1           12
Accuracy:0.6666666666666666
```

In [ ]: