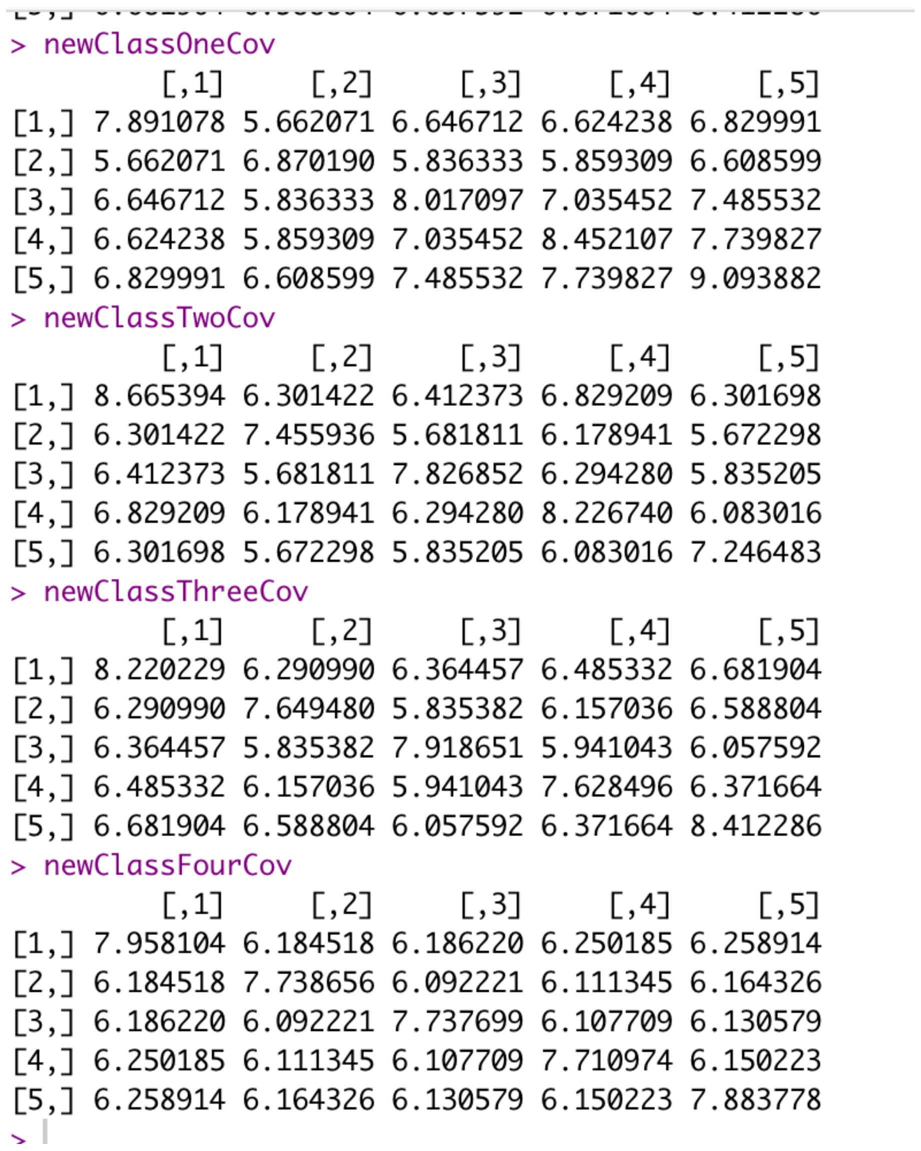
MINIPROJECT 1

ECE/CS 7720 MACHINE LEARNING AND PATTERN RECOGNITION

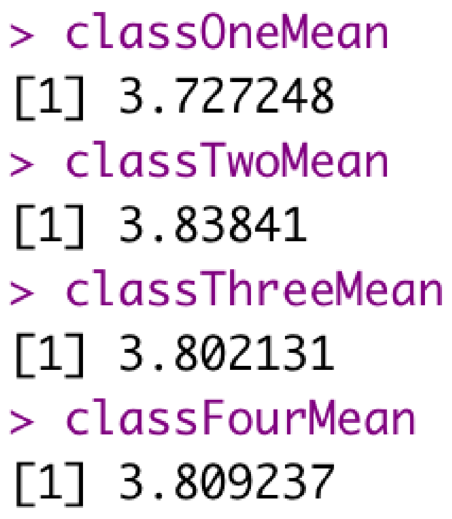
JEFFREY KERLEY
JAKCQC

Question 1.

a) Given a four-class dataset1 with 5 dimensions, compute the mean and covariance for each class. Compare the covariance you obtained by hand with the one obtained with the Matlab2 built-in function 'cov'. Did you obtain the same results? If not, explain why. If yes, why do you think I am asking this question? (i.e. what could have gone wrong?);

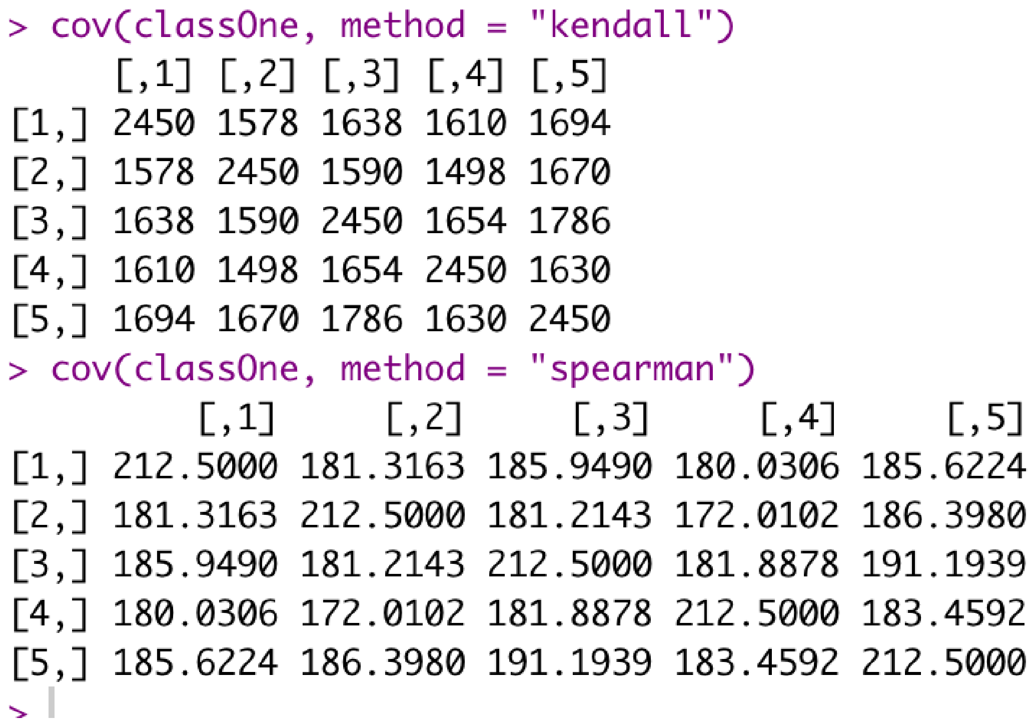


```
> newClassOneCov
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 7.891078 5.662071 6.646712 6.624238 6.829991
[2,] 5.662071 6.870190 5.836333 5.859309 6.608599
[3,] 6.646712 5.836333 8.017097 7.035452 7.485532
[4,] 6.624238 5.859309 7.035452 8.452107 7.739827
[5,] 6.829991 6.608599 7.485532 7.739827 9.093882
> newClassTwoCov
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 8.665394 6.301422 6.412373 6.829209 6.301698
[2,] 6.301422 7.455936 5.681811 6.178941 5.672298
[3,] 6.412373 5.681811 7.826852 6.294280 5.835205
[4,] 6.829209 6.178941 6.294280 8.226740 6.083016
[5,] 6.301698 5.672298 5.835205 6.083016 7.246483
> newClassThreeCov
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 8.220229 6.290990 6.364457 6.485332 6.681904
[2,] 6.290990 7.649480 5.835382 6.157036 6.588804
[3,] 6.364457 5.835382 7.918651 5.941043 6.057592
[4,] 6.485332 6.157036 5.941043 7.628496 6.371664
[5,] 6.681904 6.588804 6.057592 6.371664 8.412286
> newClassFourCov
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 7.958104 6.184518 6.186220 6.250185 6.258914
[2,] 6.184518 7.738656 6.092221 6.111345 6.164326
[3,] 6.186220 6.092221 7.737699 6.107709 6.130579
[4,] 6.250185 6.111345 6.107709 7.710974 6.150223
[5,] 6.258914 6.164326 6.130579 6.150223 7.883778
>
```



```
> classOneMean
[1] 3.727248
> classTwoMean
[1] 3.83841
> classThreeMean
[1] 3.802131
> classFourMean
[1] 3.809237
```

All my “hand” calculated covariance matrix are the same as the built in cov(). I think the differences that could arise would be method of achieving said matrix. Ie- in r, there are several methods of finding this covariance matrix and return values for class one as such,



```
> cov(classOne, method = "kendall")
     [,1] [,2] [,3] [,4] [,5]
[1,] 2450 1578 1638 1610 1694
[2,] 1578 2450 1590 1498 1670
[3,] 1638 1590 2450 1654 1786
[4,] 1610 1498 1654 2450 1630
[5,] 1694 1670 1786 1630 2450
> cov(classOne, method = "spearman")
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 212.5000 181.3163 185.9490 180.0306 185.6224
[2,] 181.3163 212.5000 181.2143 172.0102 186.3980
[3,] 185.9490 181.2143 212.5000 181.8878 191.1939
[4,] 180.0306 172.0102 181.8878 212.5000 183.4592
[5,] 185.6224 186.3980 191.1939 183.4592 212.5000
>
```

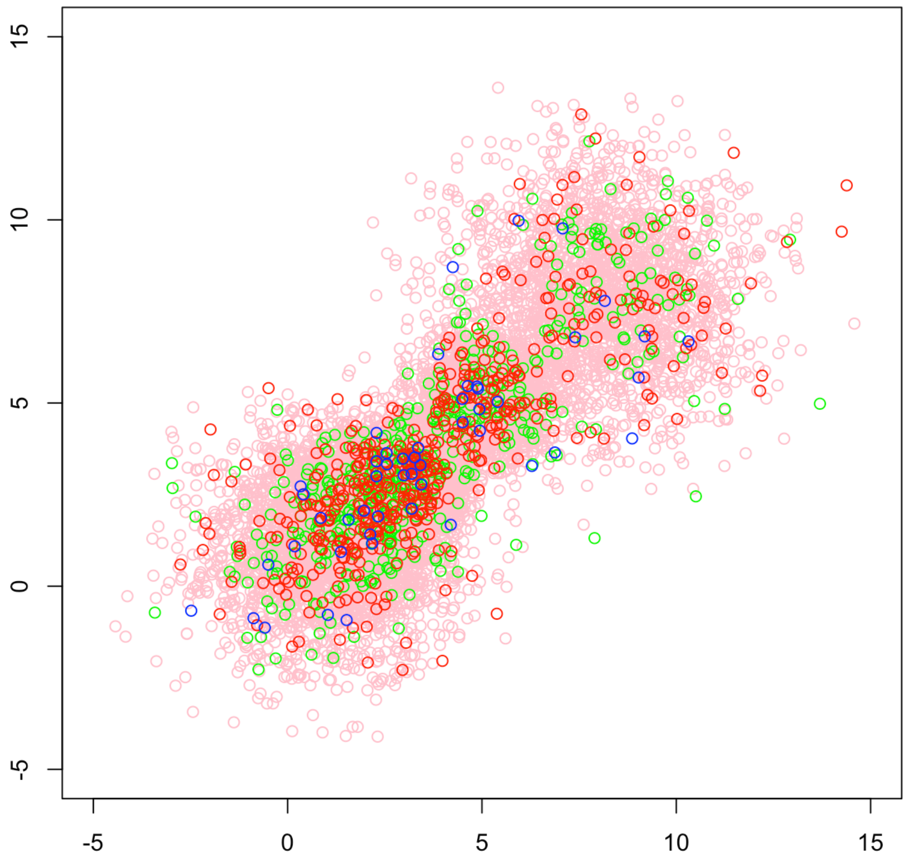
. What could have gone wrong outside of method would be messing up the order of operations on the matrix/columns, since order does matter to such a degree as well as different n x m matrix dim.

b) Write a Matlab2 function to compute all 5 eigenvectors and eigenvalues for each class. (you can NOT use any Matlab2 function or toolbox, but the basic operations: +, -, *, /, and a polynomial solver);

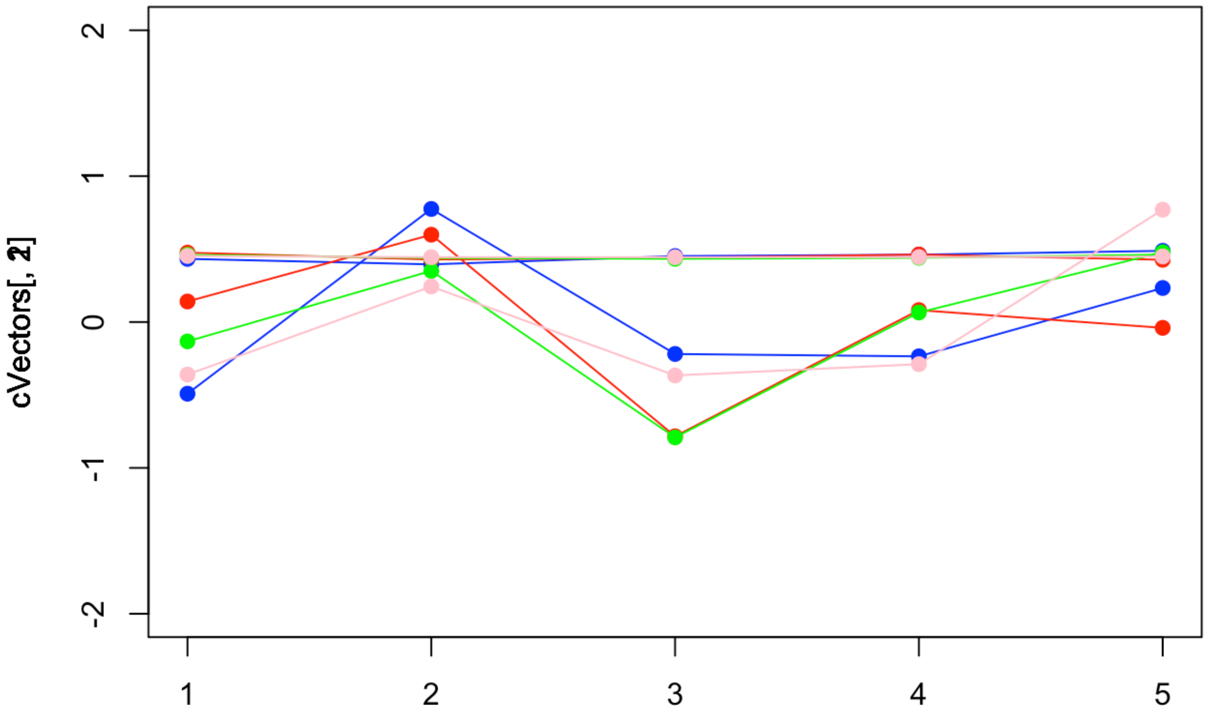
Code in plain text, R code is attached as well.

```
#QR decomposition to solve for 5 eigen vectors
getEigOrtho = function(covMat, colSize)
{
  Q = calcQMatrix(covMat, colSize)
  counterNew = 0
  while(counterNew < 50000)
  {
    baseQ = Q
    cMat = covMat %*% Q
    Q = calcQMatrix(cMat, colSize)
    checker = all.equal(Q, baseQ )
    if(checker[1]==TRUE)
    {
      print(counterNew)
      counterNew = 100000000
    }
    counterNew = counterNew + 1
  }

  return(Q)
}
## finding Q decom at each iteration for eigen vector algorithm
calcQMatrix = function(cMat, colSize)
{
  cont = 1
  calcW = 2
  while(cont <= colSize)
  {
    if(cont == 1)
    {
      cU = cMat[,1]
      eMat = matrix(cU / sqrt(sum(cU^2)), ncol =1)
    }
    if(cont > 1)
    {
      cStep = cont - 1
      cA = cMat[,cont]
      cU = cA - eMat[,cStep]*(cA %*% eMat[,cStep])
      cStep = cStep-1
      while(cStep > 0)
      {
```

```
    cU = cU - eMat[,cStep]*(cA %*% eMat[,cStep])
    cStep = cStep - 1
   }
   newCol = matrix(cU / sqrt(sum(cU^2)), ncol = 1)
   eMat = cbind(eMat, newCol)

  }
  cont = cont + 1
 }
 return (eMat)

}
#eigen values from each vector
getEigenValueFromVec = function(covMat, eigenMat, cClassValue, colSize)
{
 if(colSize == 0)
 {
  return(cClassValue)
 }
 currentAX = covMat %*% eigenMat[,colSize]
 cEigenValue = (currentAX / eigenMat[,colSize])[1]
 cClassValue[,colSize] = cEigenValue
 print(cClassValue)
 getEigenValueFromVec(covMat, eigenMat, cClassValue, colSize-1)

}
```
c) Plot the first two dimensions of each of the four-classes in the dataset using different colors and display the corresponding principal vectors of each class. (you may use any Matlab2 function for plotting); **The principle vectors, first two dimensions, aka, the largest and second-largest Eigenvectors, would be the corresponding vectors. All 4 classes have almost == principle vectors**



15
10
5
0
-5
-5
0
5
10
15



2
1
0
-1
-2
cVectors[, 2]
1
2
3
4
5

d) If you had to 'guess' the Priors of each class, what would they be? Why? (what assumptions are you making?)

If we assume that our training data is not reflective of the real data, then we can assume the prior of each class would follow as a uniform prior distribution, meaning that all priors are the same. If we assume that the training data is reflective of the real data, we can assume the priors would be based on the number of samples/number of classes, which would order the classes in the following Class 4 prior > Class 3 prior > Class 2 prior > Class 1 prior, where class 2 and class 3 have similar priors as they have the same sample size.