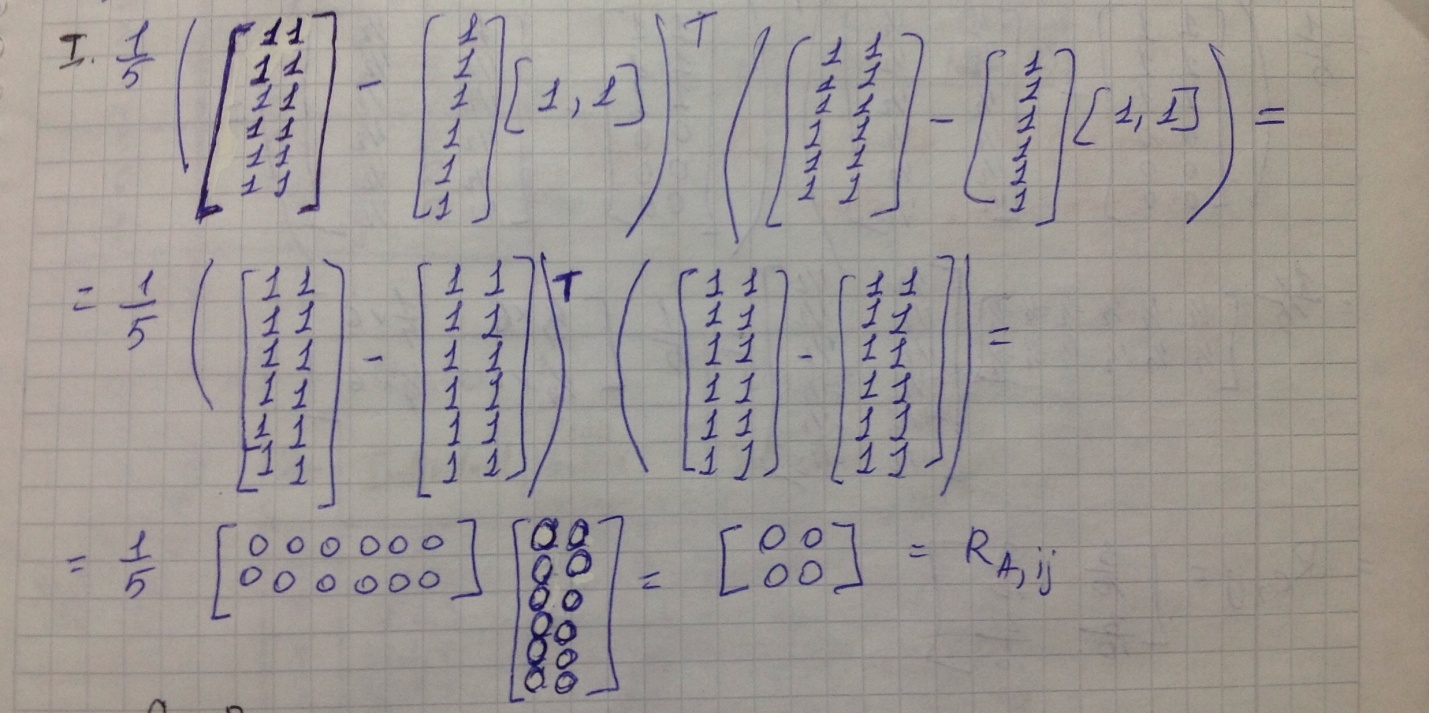
**Assignment 1 Assan Dinara**

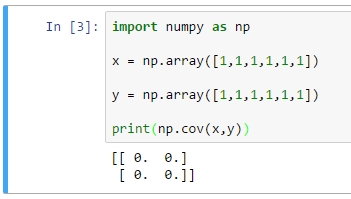
**Task 1)**

Case A:

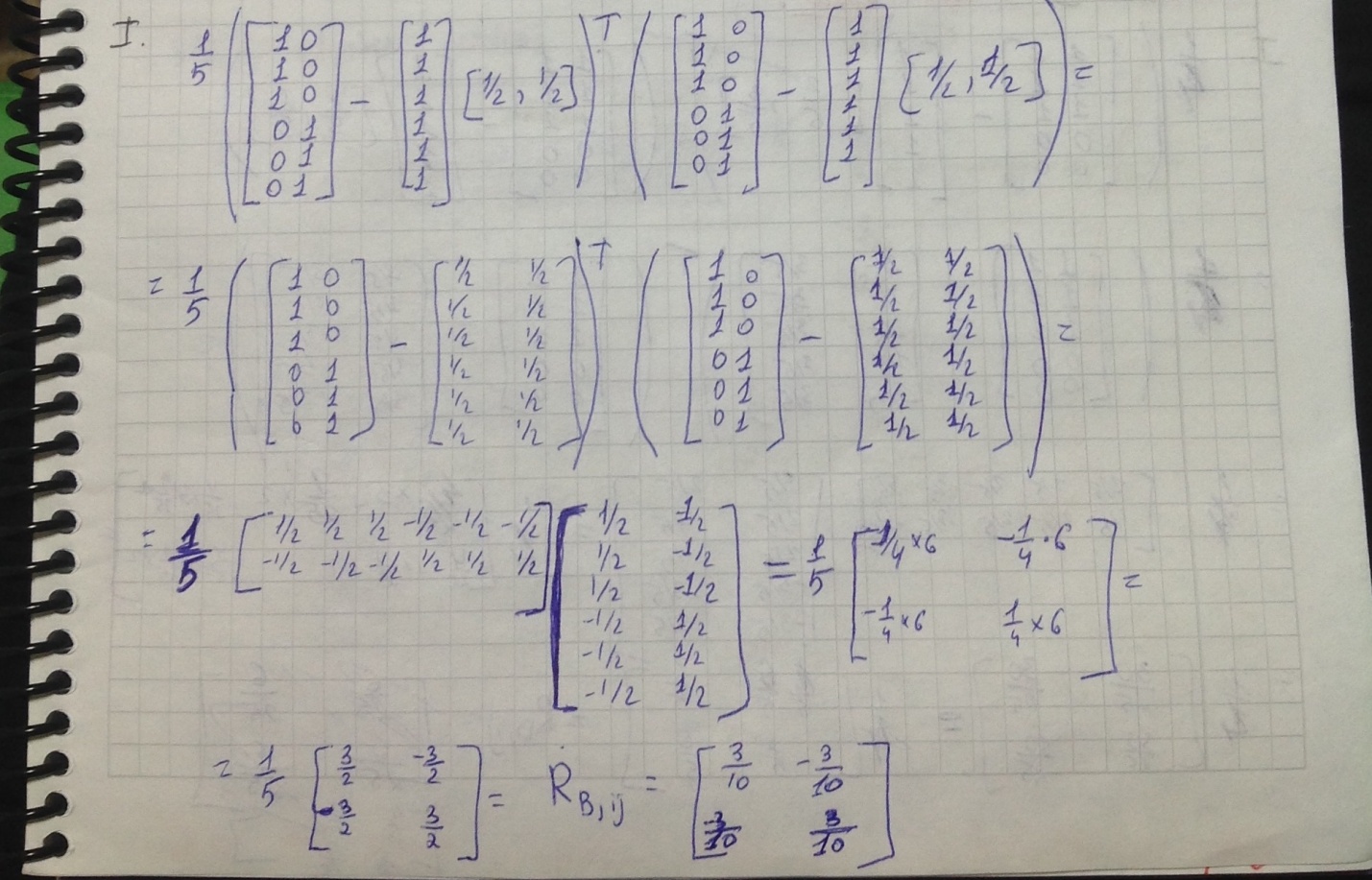
1. 
2. By own code:

import numpy as np   
  
A = np.array([[1,1], [1, 1], [1,1], [1, 1], [1, 1], [1, 1]])   
def cov(a,b):   
if len(a) != len(b):   
return   
  
a\_mean = np.mean(a)   
b\_mean = np.mean(b)   
sum = 0   
  
for i in range(0, len(a)):   
sum += ((a[i] - a\_mean) \* (b[i] - b\_mean))   
  
return sum/(len(a)-1)   
  
def printMatrix(A):   
a=A[:,0]   
b=A[:,1]   
result=[cov(a,a),cov(a,b)]   
result1=[cov(b,a),cov(b,b)]   
print("[",result)   
print(result1,"]")   
  
try:   
printMatrix(A)   
  
except(TypeError):   
print("TypeError")

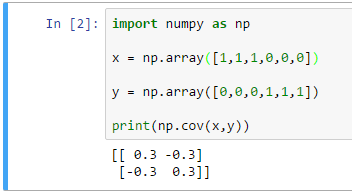
iii) by using pre-existing implementation from a python library



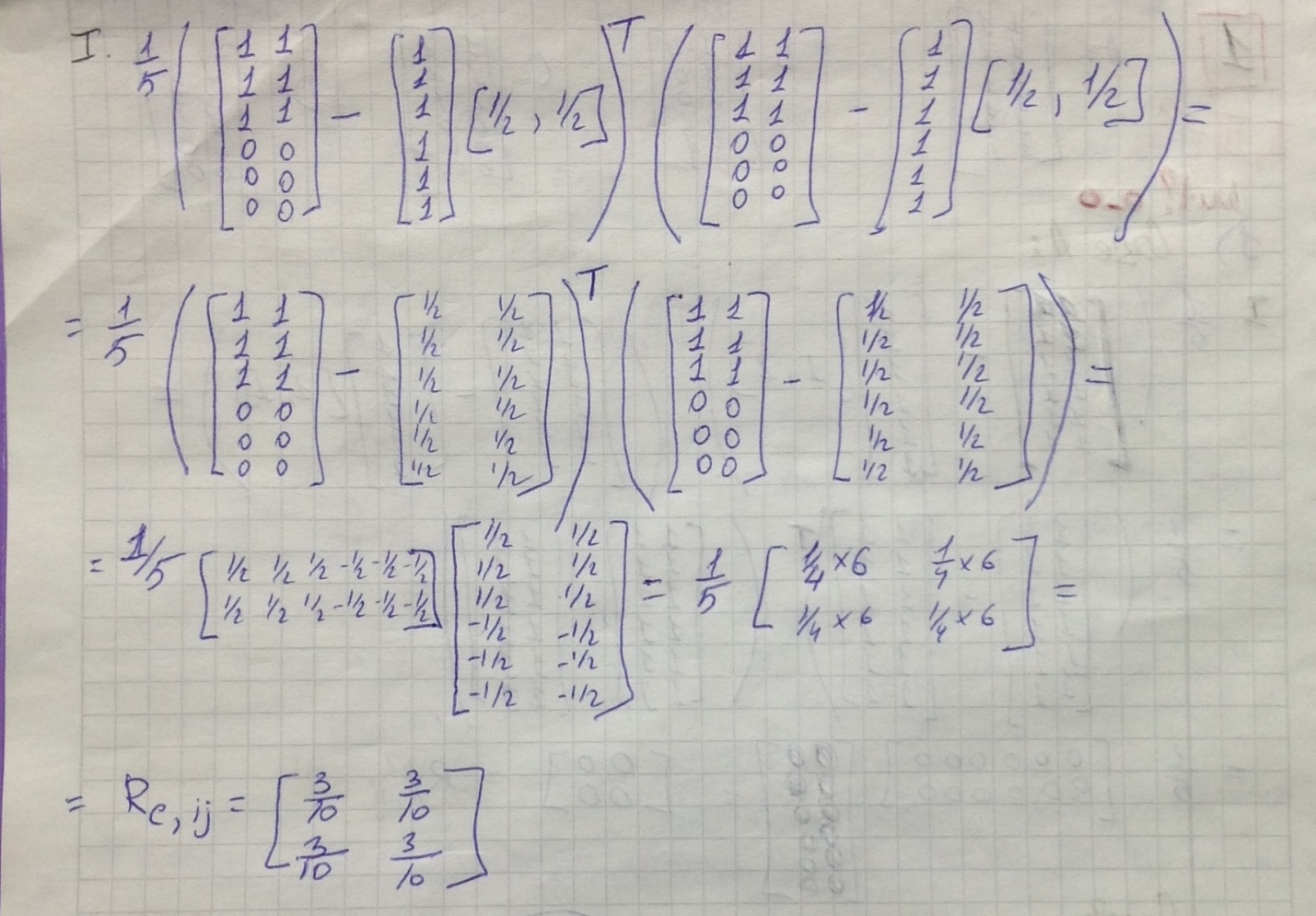
Case B:

1. 
2. import numpy as np   
     
   B = np.array([[1,0], [1, 0], [1,0], [0, 1], [0, 1], [0, 1]])   
   def cov(a,b):   
     
   if len(a) != len(b):   
   return   
     
   a\_mean = np.mean(a)   
   b\_mean = np.mean(b)   
     
   sum = 0   
     
   for i in range(0, len(a)):   
   sum += ((a[i] - a\_mean) \* (b[i] - b\_mean))   
     
   return sum/(len(a)-1)   
     
     
   def printMatrix(A):   
   a=A[:,0]   
   b=A[:,1]   
     
   result=[cov(a,a),cov(a,b)]   
   result1=[cov(b,a),cov(b,b)]   
   print("[",result)   
   print(result1,"]")   
     
   try:   
   printMatrix(B)   
   except(TypeError):   
   print("TypeError")

iii) by using pre-existing implementation from a python library



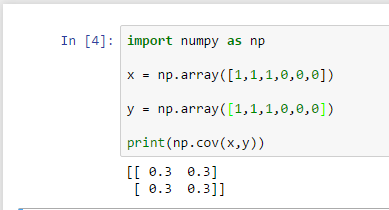
Case C:

i) 

1. import numpy as np   
     
   C = np.array([[1,1], [1, 1], [1,1], [0, 0], [0, 0], [0, 0]])   
     
   def cov(a,b):   
     
   if len(a) != len(b):   
   return   
     
   a\_mean = np.mean(a)   
   b\_mean = np.mean(b)   
     
   sum = 0   
     
   for i in range(0, len(a)):   
   sum += ((a[i] - a\_mean) \* (b[i] - b\_mean))   
     
   return sum/(len(a)-1)   
     
     
   def printMatrix(A):   
   a=A[:,0]   
   b=A[:,1]   
   result=[cov(a,a),cov(a,b)]   
   result1=[cov(b,a),cov(b,b)]   
   print("[",result)   
   print(result1,"]")   
     
   try:

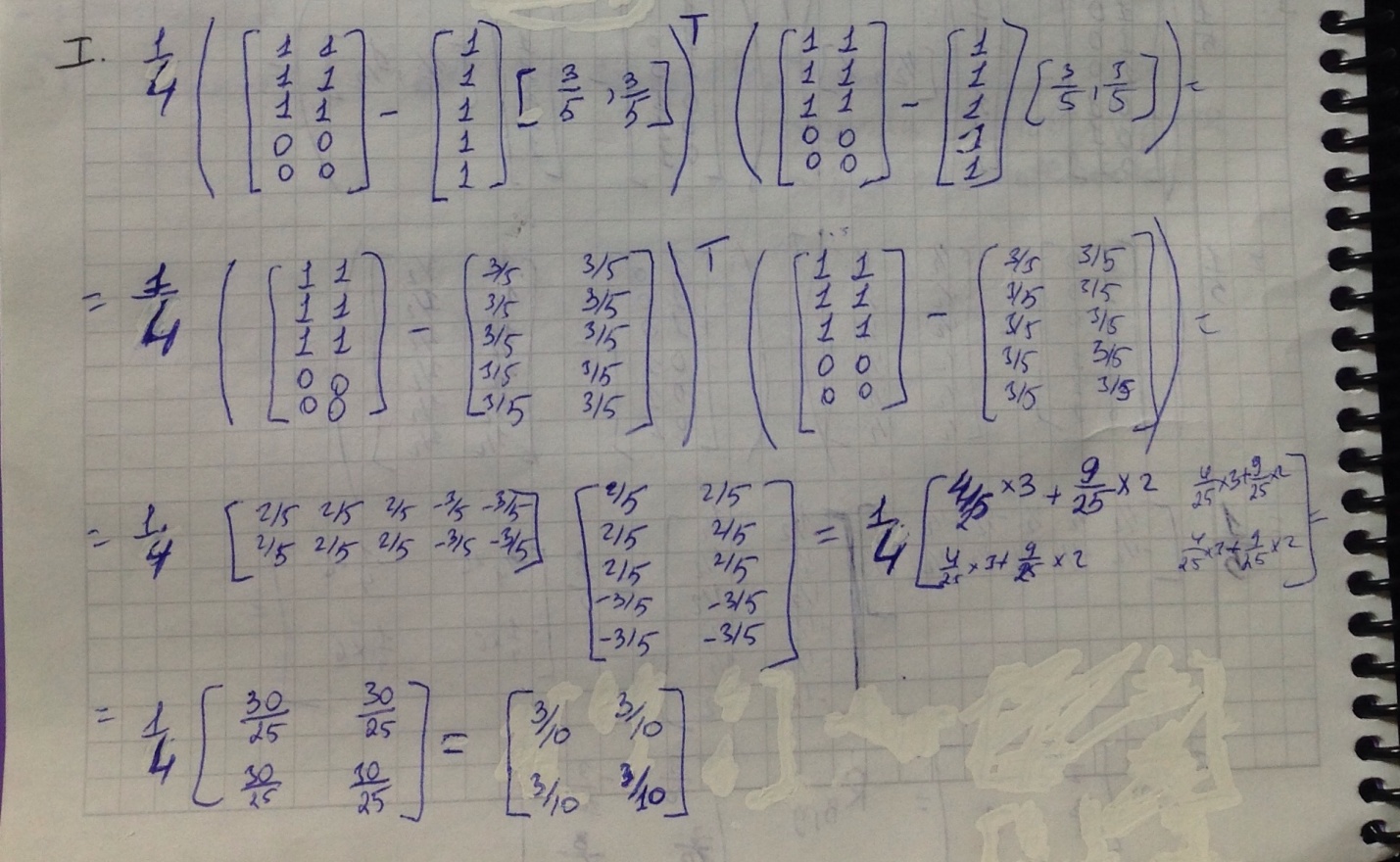
printMatrix(C)   
except(TypeError):   
print("TypeError")

iii) by using pre-existing implementation from a python library



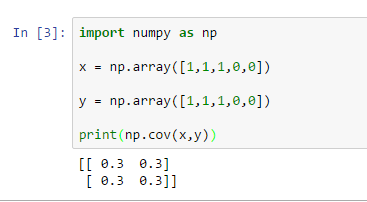
Case D:

i)



ii) import numpy as np   
  
D = np.array([[1,1], [1, 1], [1,1], [0, 0], [0, 0]])   
  
def cov(a,b):   
  
if len(a) != len(b):   
return   
  
a\_mean = np.mean(a)   
b\_mean = np.mean(b)   
  
sum = 0   
  
for i in range(0, len(a)):   
sum += ((a[i] - a\_mean) \* (b[i] - b\_mean))   
  
return sum/(len(a)-1)   
  
  
def printMatrix(A):   
a=A[:,0]   
b=A[:,1]   
result=[cov(a,a),cov(a,b)]   
result1=[cov(b,a),cov(b,b)]   
print("[",result)   
print(result1,"]")   
  
try:   
printMtrix(D)   
except(TypeError):   
print("TypeError")

iii) by using pre-existing implementation from a python library



**Task 2)** X are the data instances. The initial cluster centers are [0, 1.8] and [0.3, 2]

**x =**

Group 1 with mean vector [0, 1.8]: (0, 1.8), (0, 2), (0, -2), (0, -2.5)

Group 2 with mean vector [0.3 , 2]: (0.3, 2), (0.8, 2), (1, 2)

New cluster centers after one iteration:

New Cluster 1 center: [0, 0.2]

New Cluster 2 center:[0.7, 2]

**Does the cluster assignment change after one iteration? Answer: Yes, you can see it below.**

New Groups after first iteration:

Group 1 with mean vector [0, 0.2]: (0, 0.2), (0, -2), (0, -2.5)=🡺 new cluster 1 center: [0, -1.4]

Group 2 with mean vector [0.7, 2]: (0.7, 2), (0, 1.8) , (0, 2), (1, 2)=🡺new cluster 2 center: [0.4 , 2.0]

**Does the cluster assignment change after second iteration? Answer: No, you can see it below. The Groups with new cluster centers have the same vectors from X data instances**

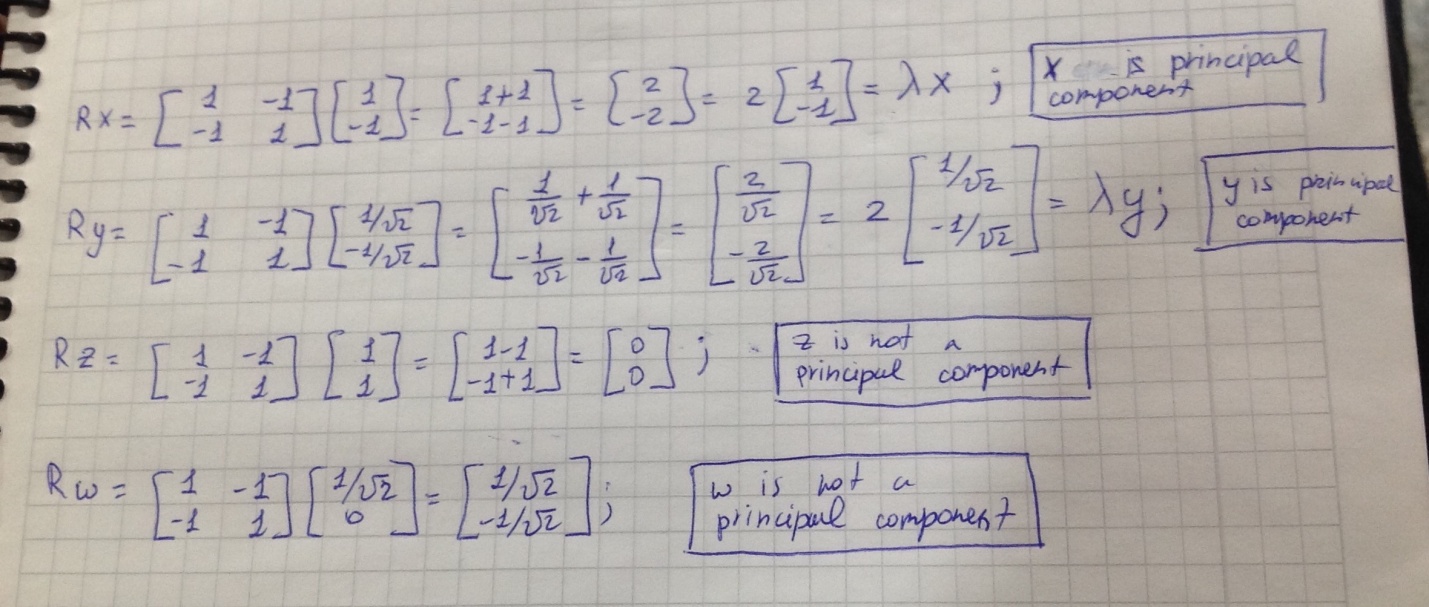
Groups after second iteration:

Group 1 with mean vector [0, -1.4]: (0, -1.4), (0, -2), (0, -2.5)=🡺 new cluster 1 center: [0, -1.4]

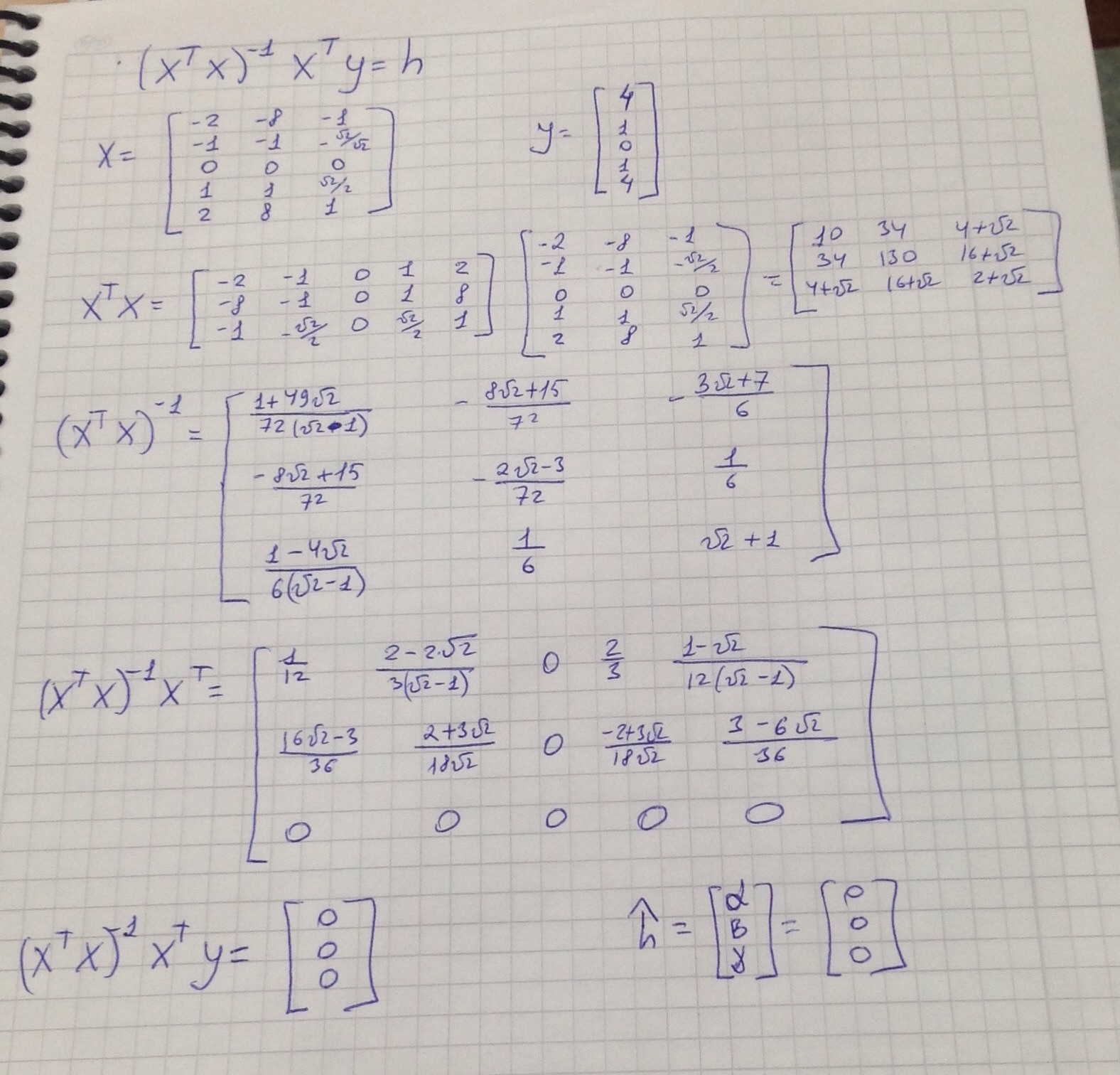
Group 2 with mean vector [0.4, 2.0]: (0.4, 2.0), (0, 1.8) , (0, 2), (1, 2)=🡺new cluster 2 center: [0.4 , 2.0]

**Task 3)** Covariance matrix R is which, if any of following vectors **x,y,z,w** are principal component?

x= , y= , z= , w=



X and Y vectors didn’t get turned, when we multiplied it by covariance matrix. It just get to be longer, but it keeps going on the same direction. And therefore X and Y are eigenvectors. And Principal component equals to the eigenvector. Therefore X and Y are principal components.

**Task 4)** ****

h = due to the symmetry. Due to the symmetry, Initial function that was given in the assignment was not appropriate. It was too complicated for this of graph.

**Task 5)**

**import** pandas  
**from** sklearn.discriminant\_analysis **import** LinearDiscriminantAnalysis  
**from** sklearn **import** neighbors  
**from** sklearn **import** cross\_validation  
  
**import** matplotlib.pyplot **as** plt  
data=pandas.read\_csv(**"data.csv"**, header=**None**)  
X = data.values[:,0:2]  
y = data.values[:,2]  
indx = (y==1) | (y==2)  
X=X[indx,:]  
y=y[indx]  
  
clf = LinearDiscriminantAnalysis()  
*#clf = neighbors.KNeighborsClassifier (1 , weights = ' uniform ')*X\_train , X\_test , y\_train , y\_test = cross\_validation.train\_test\_split (X ,y ,test\_size =0.3 ,random\_state =0)  
clf.fit ( X\_train , y\_train )  
err = clf.predict ( X\_test ) != y\_test  
  
plt.figure (1)  
**for** i **in** [1 , 2]:  
 II = y\_test == i  
 plt . plot ( X\_test [II , 0] , X\_test [ II , 1] ,**'o '** , label = str (i ))  
plt.plot ( X\_test [ err , 0] , X\_test [ err , 1] , **'ro'** )  
**try**:  
 plt.axis(**' image '**)  
**except** (ValueError):  
 print(**"Value Exception"**)  
plt.figure (2)  
**for** i **in** [1 , 2]:  
 II = y\_train == i  
 plt . plot ( X\_train [II , 0] , X\_train [II , 1] , **'o '**, label = str (i ))  
**try**:  
 plt.axis (**" image "**)  
**except**(ValueError):  
 print(**"Value Exception"**)  
plt.show ()  
  
int=clf.intercept\_  
w=clf.coef\_  
print(w);

**LDA**  performs in this way as in the lecture 11, due to the much amount of errors.