Bivariate Normal distribution

Q1.

Q1: 2D example (in numpy) to show that $(\boldsymbol{x}-\boldsymbol{\mu})\ T\Sigma\ -1\ (\boldsymbol{x}-\boldsymbol{\mu})$ is a scalar (a 1 × 1 matrix).

```python
import numpy as np

#task 2 q1

#5 = mean of x
#-2 = mean of y
mu = np.array([[5],
               [-4]])


sigma = np.array([[2,-3],
                  [-3,8]])


x = np.array([[8],[8]])











a = (x-mu).T @ np.linalg.inv(sigma)@ (x - mu)





print(a)

print('-------')
```

Output:

```
[[82.28571429]]
-------
```

Briefly describe how this quadratic form is related to the Mahalanobis distance, and therefore how it can be interpreted.

Mahalanobis distance is the measurement between a point and the distribution of other points in the data set. It takes into consideration the covariance of the data and can be applied in any dimensional space. This quadratic form is related to Mahalanobis, because it is the same equation to work out the Mahalanobis distance. This means the scalar result is the distance between point x and the distribution. Whilst taking into consideration the covariance which is a measure of the distribution.

Q2.

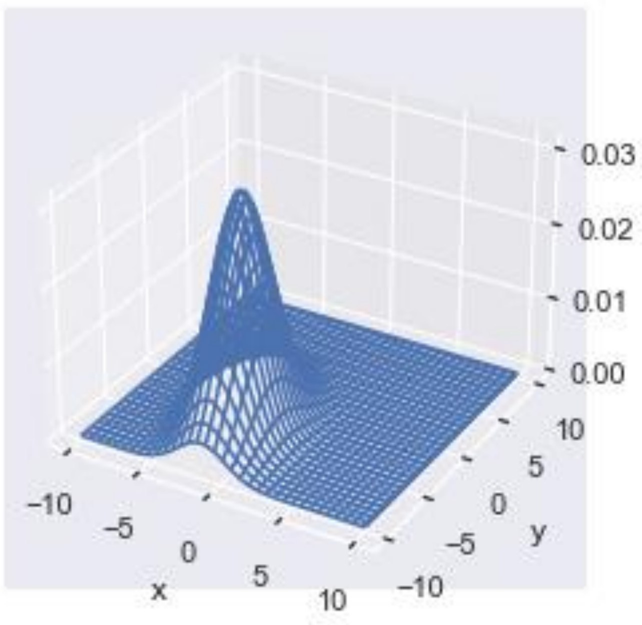
Code:

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import seaborn as sns
sns.set_theme()


#mean
mu = np.array([-2,
               -4])
#covariance matrix
sigma = np.array([[3,-1],
                 [-1,9]])


#Task 2 Q 2 wireframe plot
x = np.linspace(-10,10,101)
y = np.linspace(-10,10,101)

X, Y = np.meshgrid(x,y)
Z = np.zeros(X.shape)


#Traverse through the len of y and x
for i in range(len(y)):
    for j in range(len(x)):
        point= np.array([X[i,j],Y[i,j]])
        Z[i,j] =(stats.multivariate_normal.pdf(point, mu, sigma))


fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.plot_wireframe(X,Y,Z)

plt.xlabel('x ')
plt.ylabel('y')
plt.show()


# Contour plot
plt.figure()
plt.contour(X,Y,Z)

plt.axis('equal')
plt.show()


V,P = np.linalg.eigh(sigma)
#V = eigenvalues for each vector
print('eigenvalues\n'
      , V)
#p = eigenvector for matrix M
print('eigenvector\n'
      , P)

#Matrix D is a matrix made up of
# the eigenvalues put into a matrix using diagonalisation
# Diagonalisation
# (matrix decomposition)
D = np.diag(V)
#D = eigenvalues put into a matrix
print('D eigenvalues into a matrix\n'
      , D)

#PDP^-1
magic = P@D@np.linalg.inv(P)
print('magic\n'
      ,magic)

#contour plot with P mapped onto it

plt.figure()


plt.plot([mu[0],mu[0]+P[0,0]],[mu[1],mu[1]+P[1,0]],'black')
plt.plot([mu[0],mu[0]+P[0,1]],[mu[1],mu[1]+P[1,1]],'black')


plt.title('Contour with P mapped on it')
plt.xlabel('x ')
plt.ylabel('y')

plt.axis('equal')
plt.contour(X,Y,Z)


plt.show()
```

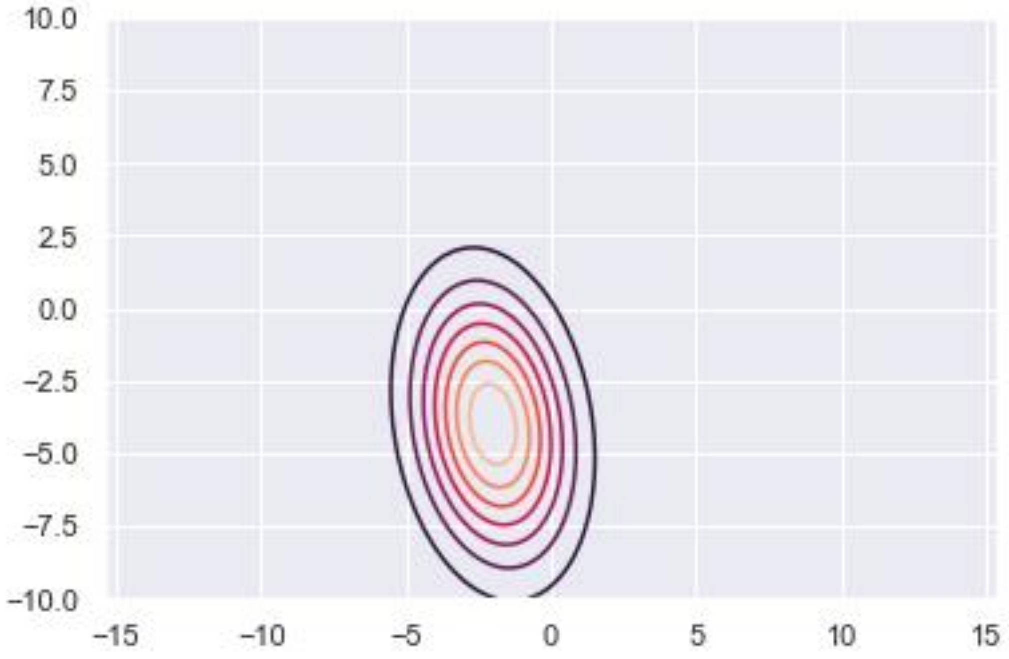
Output:

```
eigenvalues
 [2.83772234 9.16227766]
eigenvector
 [[-0.98708746 -0.16018224]
 [-0.16018224  0.98708746]]
D eigenvalues into a matrix
 [[2.83772234 0.        ]
 [0.         9.16227766]]
magic
 [[ 3. -1.]
 [-1.  9.]]
```

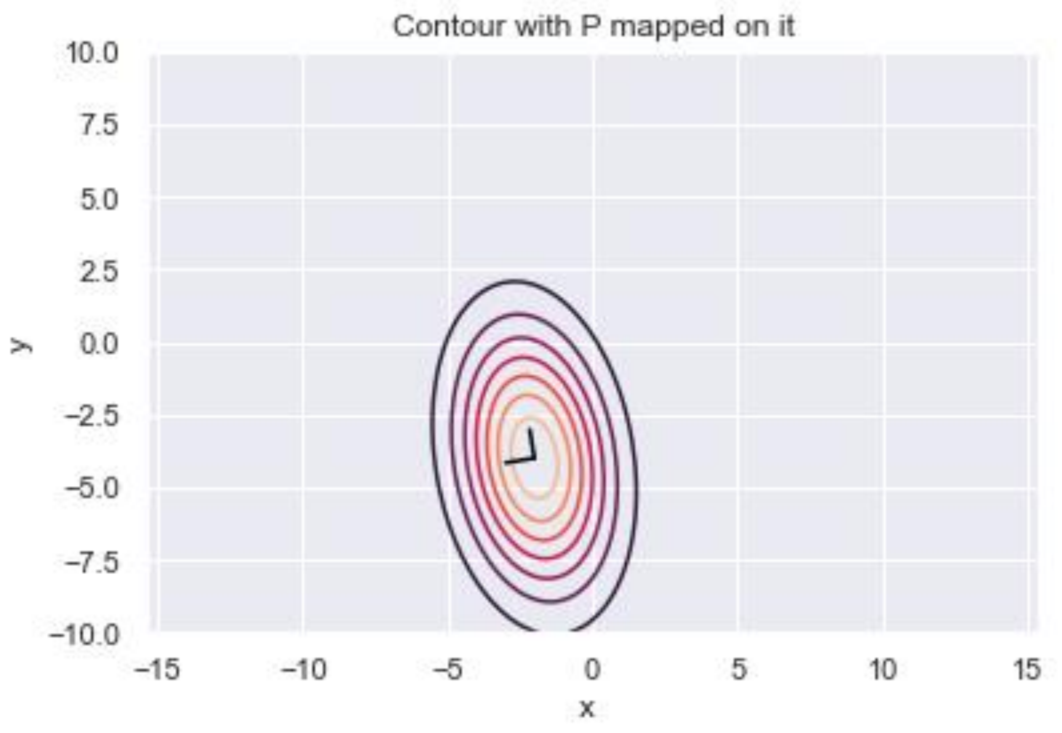
Output of plots:



0.03
0.02
0.01
0.00
−10
−5
0
5
10
x
−10
−5
0
5
10
y



10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−15
−10
−5
0
5
10
15



Describe and discuss what you discover.

The eigenvectors of P plot perpendicular to each other starting on the mean. As they are eigenvectors, the only transformation which will affect them should be a scalar matrix transformation.

The contour plot is a 2D representation of the 3D wireframe plot. The area under the wireframe plot can be used to work out the probability. Additionally, the area under the plot should equate to 1.

The wireframe plot looks like a gaussian distribution which is a 3D normal distribution.

The mean is the centre point of the rings on the contour plot and the centre point of the corresponding plot on the wireframe plot.

The eigenvectors are used to create the axis for a principal component analysis. The axis would be PCA 1 and PCA 2 if the axis were rotated so that they were level for example straight up and straight across without having to tilt your head or screen. The eigenvector is called the singular vector for PC1 and PC2. The first eigenvector would make up the singular vector for PC1 the second eigen vector would make up the singular vector for PC2 (https://www.youtube.com/watch?v=FgakZw6K1QQ, 11 minutes to 14 minutes).

Tas 2 Q3:

Code:

```
#Task 2 Q3.

#Random sample using original mean and covariance vector
#plotted over contour plot

N = 100

```

```python
A =(stats.multivariate_normal.rvs(size=N, mean = mu, cov = sigma))

x = A[:,0]
y = A[:,1]

plt.plot([mu[0],mu[0]+P[0,0]],[mu[1],mu[1]+P[1,0]],'black')
plt.plot([mu[0],mu[0]+P[0,1]],[mu[1],mu[1]+P[1,1]],'black')




plt.scatter(x,y)
plt.contour(X,Y,Z)
plt.axis('equal')
plt.title('Contour with P mapped on it and scatter plot')
plt.show()


#Calculate the
#sample mean vector and the sample covariance
#matrix C and compare these to μ and Σ.

#Smu = sample mean
Smu1 = np.mean(A[:,0])
Smu2 = np.mean(A[:,1])

Smu = np.array([Smu1, Smu2])

print('Original mean\n',
      mu,
      '\nSample mean\n',
      Smu)
#Smu is almost exactly the same as the original vector mean

#C = sample covariance matrix
C = np.cov(x,y)

print('Original Covarience matrix\n',
      sigma,
      '\nCovarience matrix\n',
      C)
#C is almost the exact same as the original sigma as well.

#as N is increased the closer the new mean is to the old mean
# and the closer sigma is to the new covariance matrix

#Diagonalise C as C = PDP^-1

V,P = np.linalg.eigh(C)
#V = eigenvalues for each vector
print('eigenvalues\n'
      , V)
#p = eigenvector for matrix M
print('eigenvector\n'
      , P)

#Matrix D is a matrix made up of
# the eigenvalues put into a matrix using diagonalization
# Diagonalization
# (matrix decomposition)
D = np.diag(V)
#D = eigenvalues put into a matrix
print('D eigenvalues into a matrix\n'
      , D)

#PDP^-1
```

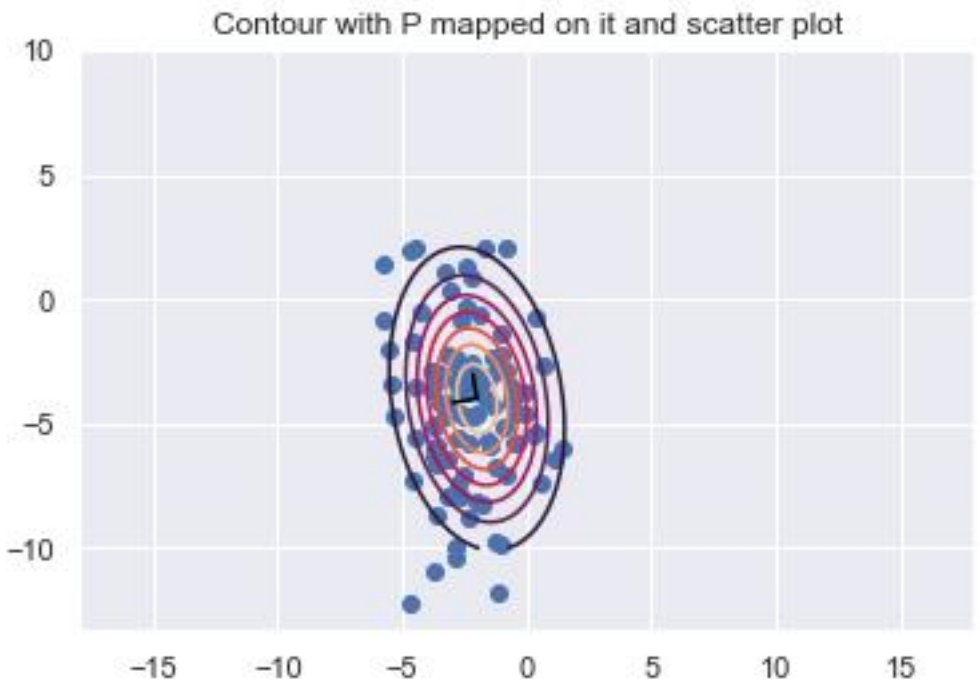
```
magic = P@D@np.linalg.inv(P)
print('magic\n'
      ,magic)


print('C matrix\n',
      C,
      '\nmagic\n',
      magic)
#magic = C

#mean is taken away from A (The random sample)
plt.figure()

#subtracting the mean
A1 = A - Smu

#P is used as the matrix transformation
#A1 was transposed for the matrix multiplication.
Mt = P@A1.T

x = Mt[0]
y = Mt[1]

plt.scatter(x,y)

plt.axis("equal")
plt.show()

newC = np.cov(x,y)

print('D\n',
      D)
print('newC\n',
      newC)

#D makes up the covariance of X and Y as D makes up
#the diagonal line from top left to bottom right for the new
#covariance matrix.
```

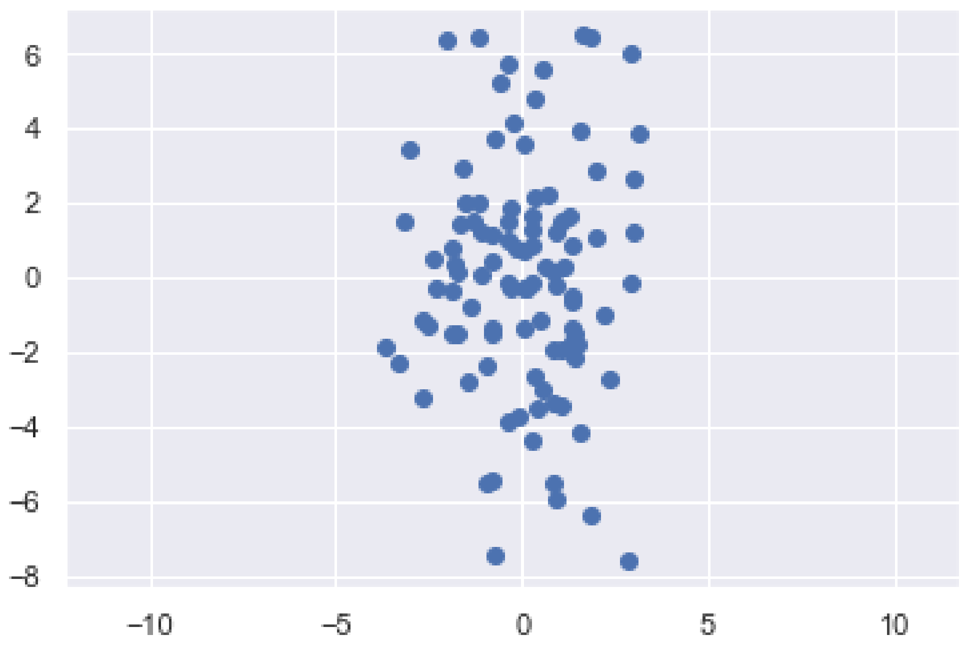
Output:

```
Original mean
 [-2 -4]
Sample mean
 [-2.40161701 -4.39742354]
Original Covarience matrix
 [[ 3 -1]
 [-1  9]]
Covarience matrix
 [[ 2.45142388 -0.50237218]
 [-0.50237218  9.59244174]]
eigenvalues
 [2.41625509 9.62761053]
eigenvector
 [[-0.99755859 -0.06983454]
 [-0.06983454  0.99755859]]
D eigenvalues into a matrix
 [[2.41625509 0.        ]
 [0.         9.62761053]]
```

```
magic
 [[ 2.45142388 -0.50237218]
 [-0.50237218  9.59244174]]
C matrix
 [[ 2.45142388 -0.50237218]
 [-0.50237218  9.59244174]]
magic
 [[ 2.45142388 -0.50237218]
 [-0.50237218  9.59244174]]
D
 [[2.41625509 0.        ]
 [0.         9.62761053]]
newC
 [[ 2.41625509e+00 -3.05857425e-16]
 [-3.05857425e-16  9.62761053e+00]]
```

Output plots:





Sample covariance C was very similar to sigma, the sample mean was the same as the original mean. Lines 26 – 52 of the code. Lines 1 – 10 on the output. As the sample size was increased the closer the numbers were to each other.

The diagonal entries in D are the variance of x and y of the sample bivariate normal distribution. The variance of x is the top left of the matrix and the variance of y is the bottom right of the matrix. Lines 103 – 112 of the code. Lines 28 – 33 on the output.

The contour plot looks very similar to a confidence ellipsoid. Most of the points on the scatter plot fit into the contour which would suggest if it were a confidence ellipsoid that it would be a high percentage ellipsoid such as a 95% confidence ellipsoid. A 95% ellipsoid represents the region where 95% of the samples can be estimated to be from. It is likely that each ring represents a different confidence level as the rings go towards the centre, they get smaller and smaller suggesting the confidence level is decreasing. For example, the most outer ring might be 95% confidence level but the third or fourth ring inwards might be 90% confidence or lower (Computer vision for dummies, 2014).

As Principal component analysis is about dimensional reduction to make comparing the variance between variables easier for multidimensional data. The transformation is used to align the data with the PCA 1 and PCA 2 generated from the eigenvectors in question 2. The data keeps its value despite being moved because it is moved using P. The data points appear to be thinner than they are wide. The data points have reflected and moved in an upwards direction after the transformation whilst maintaining their original structure. For example, the data points weren't moved from each other so the variation should remain the same which it does as can be seen by C and newC in the code. The data points are still clustered as a line up the middle. This suggests that the data is not correlated to each other as when the y axis increases its value the data points along the x axis stays in the same location with little variation. D tells us that the variation for x is quite low at 2.41 whilst y has a much bigger variation at 9.6. This coincides with what was plotted in the scatter graph.

A covariance matrix represents 2 components the first is variance. The variance is the spread of data. So, in this example sigma covariance matrix, or C covariance matrix or newC covariance matrix shows the spread of data for x and y, along the diagonal from top left to the bottom right. The variance for x is the top left number (2.45142388) and the variance for y is the bottom right number (9.62761053). The second component is covariance. Covariance is the same number from the top right to bottom left. The covariance shows the directional relationship between the two random variables. As this number is small for C at -0.5 this suggests that there is little relationship between the variables. Negative covariances typically tell us that the variables move in opposite directions (Hae-Young Kim, 2018).

Reference list

Computer vision for dummies (2014). How to draw a covariance error ellipse? Retrieved November 26, 2020, from https://www.visiondummy.com/2014/04/draw-error-ellipse-representing-covariance-matrix/#:~:text=The%20following%20figure%20shows%20a,from%20the%20underlying%20Gaussian%20distribution

Hae-Young Kim (2018). Statistical notes for clinical researchers: covariance and correlation. *Restor Dent Endod 43*(1). doi: 10.5395/rde.2018.43.e4

youtube (2018, April 2). StatQuest: Principal Component Analysis (PCA), Step-by-Step. Retrieved November 26, 2020, from https://www.youtube.com/watch?v=FgakZw6K1QQ