CMPT 726/410 Fall 2023 Assignment 1

Wenhe Wang (301586596 #)

November 10, 2023

©Simon Fraser University. All Rights Reserved. Sharing this publicly constitutes both academic misconduct and copyright violation.

Due: Thursday, November 23, 2023 at 11:59 pm Pacific Time

Important: Make sure to download the zip archive associated with this assignment, which contains essential data and starter code that are required to complete the programming component of the assignment.

This assignment is designed to be substantially more challenging than the quizzes and requires thorough understanding of the course material and extensive thought, so **start early!** If you are stuck, come to office hours. Make sure to check the discussion board often for updates, clarifications, corrections and/or hints.

Partial credit will be awarded to reasonable attempts at solving each question, even if they are not completely correct.

There will be office hours dedicated to assignment-related questions. Times and Zoom links will be posted on Canvas. If you have a question that you would like to ask during office hours, make sure to post a brief summary of your question in the office hours thread on the Canvas discussion board. During office hours, questions will be answered in the order they appeared on the office hours thread. We may not be able to get to all questions, so please start early and plan ahead.

Requests for extensions will not be considered under any circumstances. Make sure you know how to submit your assignment to Crowdmark and leave sufficient time to deal with any technical difficulties, Internet outages, server downtimes or any other unanticipated events. It is possible to update your assignment after submission, so we recommend uploading a version of your assignment at least several hours before the deadline, even if it is incomplete.

Submission Instructions

Carefully follow the instructions below when submitting your assignment. Not following instructions will result in point deductions.

1. You should typeset your assignment in LaTeX using this document as a template. We recommend using Overleaf to compile your LaTeX. Include images in the section they go with and **do not** put them in an appendix.
2. At the beginning of the assignment (see above), please copy the following statement and sign your signature next to it. (macOS Preview and FoxIt PDF Reader, among others, have tools to let you sign a PDF file.) We want to make it *extra* clear so that no one inadvertently cheats.

   "I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted."
3. At the start of each problem—not subproblem; you only need to do this twice—please state: (1) who you received help from on the problem, and (2) who you provided help to.
4. For all theory questions, make sure to **show your work** and include all key steps in your derivations and proofs. If you apply a non-trivial theorem or fact, you must state the theorem or fact you used, either by name (e.g.: Jensen's inequality) or by writing down the general statement of the theorem or fact. You may use theorems and facts covered in lecture without proof. If you use non-trivial theorems and facts not covered in lecture, you must prove them. All conditions should be checked before applying a theorem, and if a statement follows from a more general fact, the reason why it is a special case of the general fact should be stated. The direction of logical implication must be clearly

stated, e.g.: if statement A implies statement B, statement B implies statement A, or statement A is equivalent to statement B (i.e.: statement A is true if and only if statement B is true). You should **number** the equations that you refer to in other parts and refer to them by their numbers. You must **highlight the final answer or conclusion** in your PDF submission, either by changing the font colour to red or drawing a box around it.

5. For all programming questions, starter code is provided as a Jupyter notebook. You should work out your solution in the notebook. We recommend uploading your notebook to a hosted service like Google Colab to avoid the installation of dependencies and minimize setup time. When you are done, take a screenshot of your code and output and include it in your PDF. In addition, you should download the Jupyter notebook with your solutions (`File` → `Download` → `Download .ipynb`) and also download your code (`File` → `Download` → `Download .py`). You need to **submit both** as a zip archive named "`CMPT726-410_A1_⟨Last Name⟩_⟨Student ID⟩.zip`", whose directory structure should be the same as that of the zip archive for the starter code. Do **NOT** include any data files we provided. Please include a short file named **README** listing your name and student ID. The PDF should not be included in the zip archive and should be submitted **separately**. Please make sure that your code doesn't take up inordinate amounts of time or memory. If your code cannot be executed, your solution cannot be verified.

6. Assignment submissions will be accepted through Crowdmark. The submission portal will open a week before the assignment is due. Make sure to set aside at least **one hour** to submit the assignment. Crowdmark will ask you to split the PDF by each part of each question. Make sure to upload the correct pages for each part, and double check when you are done.

Python Configuration

We recommend using Google Colab to complete the parts of this assignment that require coding. However, if you would like to set up your own Python environment on your computer, follow the instructions below.

1. Ensure you have Python 3 installed. If you don't, we recommend miniconda as a package manager. To ensure you're running Python 3, open a terminal in your operating system and execute the following command: `python --version` **Do not proceed until you're running some (recent) version of Python 3.**

2. Install `scikit-learn` and `scipy`:
`conda install -y -c conda-forge scikit-learn scipy matplotlib`

3. To check whether your Python environment is configured properly for this homework, ensure the following Python script executes without error. Pay attention to errors raised when attempting to import any dependencies. Resolve such errors by manually installing the required dependency (e.g. execute `conda install -c conda-forge numpy` for import errors relating to the numpy package).

```
import sys
if sys.version_info[0] < 3:
    raise Exception("Python 3 not detected.")

import numpy as np
import matplotlib.pyplot as plt
from scipy import io
```

4. Please use only the packages listed above.

Collaboration and Academic Integrity

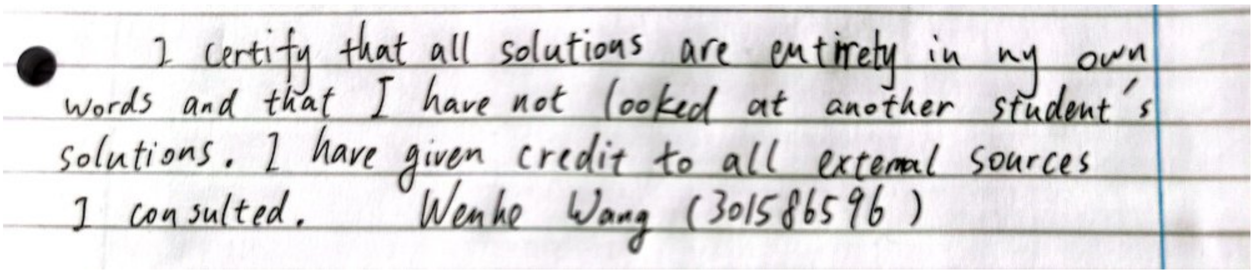
While collaboration is encouraged, all of your submitted work must be your own. Concretely, that means you may discuss general approaches to problems with other students, but you must write your solutions on your own. For more information, see the course syllabus under "Academic Integrity." If you got help from or gave help to other students, please note their names at start of every question. Additionally, sign the declaration at the bottom of this page.

Warning: Copying others' solutions, seeking help from others not in this course, posting questions online or entering questions into automated question answering systems are considered cheating. Consequences are severe and could lead to suspension or expulsion. If you become aware of such instances, you must report them here: https://forms.gle/9fw3oMLyhD1A81qy5.

Page Numbers

Declaration

I certify that all solutions are entirely in my own words and that I have not looked at solutions other than my own. I have given credit to all external sources I consulted.



I certify that all solutions are entirety in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted. Wenhe Wang (301586596)

Wenhe Wang (301586596 #)

Question 1 Linear Algebra and Calculus [35 points]

List your collaborators on this question below.

Solution.

Students I got help from: No

Students I gave help to: No

External sources that I consulted (websites, books, notes, etc.): Google, Wikipedia

Part 1(a) [5 points]

Let A be an $n \times n$ matrix with eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$. **Prove $\lambda_1^k, \lambda_2^k, ..., \lambda_n^k$ are eigenvalues of A^k.** Justify each step that is not a simple algebraic manipulation.

Solution.

1. Assume λ is an eigenvalue of A with a corresponding non-zero eigenvector v. By definition of eigenvalue and eigenvector, $Av = \lambda v$.

2. If we want to prove the statement to show $A^k v = \lambda^k v$ for any positive integer k is satisfied, We need to do Mathematical Induction on k.

When k = 1: $A^1 v = Av = \lambda^1 v \ = \lambda v$ is proved by the definition of eigenvalue and eigenvector.

When k = 2: Multiply both sides of $Av = \lambda v$ by A, we get:

$$A^2 v = A(Av) = A(\lambda v) = \lambda(Av) = \lambda(\lambda v) = \lambda^2 v.$$

Since v is non-zero, this shows that v is also an eigenvector of A^2 with eigenvalue λ^2.

For any positive integer k: Assume $A^k v = \lambda^k v$ is true. Prove $A^{k+1} v = \lambda^{k+1} v$.

Multiply both sides of $A^k v = \lambda^k v$ by A, we get:

$$A(A^k v) = A(\lambda^k v)$$

Because λ^k is a scalar, we can move this term outside the bracket by the rule of Compatibility of the product of scalars with scalar multiplication:

$$A^{k+1} v = \lambda^k (Av)$$

Use the definition of the eigenvector ($Av = \lambda v$) on the right side:

$$A^{k+1} v = \lambda^k (\lambda v) = \lambda^{k+1} v$$

This proves that v is also an eigenvector of A^{k+1} with eigenvalue λ^{k+1}.

Final Conclusion: YOUR FINAL ANSWER OR CONCLUSION HERE

From the induction above, $A^k v = \lambda^k v$ holds for all positive integers k, so λ^k is an eigenvalue of A^k with eigenvector v.

This proof holds for any eigenvalue λ of A and its eigenvector v because any eigenvalue and eigenvector holds this property $Av = \lambda v$, such that it holds for all eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$ of A. Thus, $\lambda_1^k, \lambda_2^k, ..., \lambda_n^k$ are eigenvalues of A^k.

Part 1(b) [5 points]

Let A be an $m \times m$ matrix. Consider a decomposition of $A = BCD^\top$, where C is a diagonal matrix. **If $D^\top B = -2I$, where I is the identity matrix, derive an expression for A^n in terms of B, C and D for any $n \geq 1$.** Simplify the expression as much as possible and show your work.

Solution.

1. A^n can be written as:

$$A^n = A \times A \times \cdots \times A$$

2. Substituting $A = BCD^\top$ from the property given by the statement, we have:

$$A^n = (BCD^\top)(BCD^\top)\cdots(BCD^\top)$$

3. To simplify this equation, apply the property $D^\top B = -2I$ to replace all $D^\top B$ terms in the middle of the equation, and we have:

$$A^n = (-2)^{n-1} BC^n D^\top$$

Here, because I is the identity matrix which can be neglected when it multiplies any other matrix will just get the other matrix itself. And -2 is a scalar that can be taken out from the bracket, and there are $n-1$ terms of it, thus we have the term $(-2)^{n-1}$. Also note that, because C is a diagonal matrix, the C^n term in the final equation of A^n is a diagonal matrix with each diagonal element raised to the power n.

Final Conclusion: YOUR FINAL ANSWER OR CONCLUSION HERE

$$A^n = (-2)^{n-1} BC^n D^\top$$

Where C^n is a diagonal matrix with each element in the diagonal raised to the power n.

Part 1(c) [5 points]

Let A be a 2×2 matrix. Suppose the singular value decomposition of A is $I\Sigma V^\top$, where I is the identity matrix. Let $\vec{v}_1$ and $\vec{v}_2$ denote the right-singular vectors of A, which are associated with the singular values 5 and 2 respectively. Consider a vector $\vec{\beta} = 3\vec{v}_1 - \vec{v}_2$. **What is $A\vec{\beta}$?** Show your work.

Solution.

1. The SVD of A is given by the statement where

$$A = I\Sigma V^\top$$

By the property of SVD, we know that I is the identity matrix. And A is a 2×2 matrix, Σ is a 2×2 diagonal matrix with singular values on the diagonal, and V is a 2×2 matrix whose columns are $\vec{v}_1$ and $\vec{v}_2$. Thus, we have:

$$\Sigma = \begin{pmatrix} 5 & 0 \\ 0 & 2 \end{pmatrix}, \quad V = \begin{pmatrix} | & | \\ \vec{v}_1 & \vec{v}_2 \\ | & | \end{pmatrix}$$

2. So we can rewrite $A\vec{\beta}$ as:

$$A\vec{\beta} = I\Sigma V^\top \vec{\beta} = \Sigma V^\top \vec{\beta}$$

Where I is the identity matrix and it does not change the product, so we can get rid of it.

3. Plug $\vec{\beta} = (3\vec{v}_1 - \vec{v}_2)$ into the equation, we have:

$$\Sigma V^\top \vec{\beta} = \Sigma V^\top (3\vec{v}_1 - \vec{v}_2)$$

Since V is orthogonal, $V^\top \vec{v}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $V^\top \vec{v}_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, so

$$V^\top \vec{\beta} = V^\top 3\vec{v}_1 - V^\top \vec{v}_2 = 3\begin{pmatrix} 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ -1 \end{pmatrix}$$

4. Finally, we have:

$$A\vec{\beta} = \Sigma V^\top \vec{\beta} = \Sigma \begin{pmatrix} 3 \\ -1 \end{pmatrix} = \begin{pmatrix} 5 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 3 \\ -1 \end{pmatrix} = \begin{pmatrix} 5 \cdot 3 \\ 2 \cdot (-1) \end{pmatrix} = \begin{pmatrix} 15 \\ -2 \end{pmatrix}$$

Final Conclusion: YOUR FINAL ANSWER OR CONCLUSION HERE

$$A\vec{\beta} = \begin{pmatrix} 15 \\ -2 \end{pmatrix}$$

Part 1(d) [5 points]

If A, B and BC are symmetric matrices, prove that $(ABC)^\top = BCA$. Justify each step that is not a simple algebraic manipulation.

Solution.

1. By the property of symmetry and symmetric A, B and BC matrices given by the statement, we have:

$$A = A^\top, B = B^\top, BC = (BC)^\top = C^\top B^\top$$

2. Substitute the BC in ABC with $C^\top B^\top$, and by properties of matrix transposition we have:

$$(ABC)^\top = (AC^\top B^\top)^\top = (BCA^\top) = BCA$$

Final Conclusion: YOUR FINAL ANSWER OR CONCLUSION HERE

We have shown that $(ABC)^\top = BCA$ using the properties of matrix transposition and the symmetry of the matrices A, B, and BC

Part 1(e) [5 points]

Let A be a symmetric matrix. **Prove that A can be expressed as $B - C$, where B and C are in the form $UD_1U^\top$ and $UD_2U^\top$ respectively, with U being orthogonal and D_1, D_2 being diagonal with strictly positive entries.** Justify each step that is not a simple algebraic manipulation.

Solution.

1. By the Spectral Theorem, we know that the symmetric matrix A can be expressed as $A = UDU^\top$, where U is an orthogonal matrix (meaning $U^\top = U^{-1}$) and D is a diagonal matrix containing the eigenvalues of A.

2. We split our diagonal matrix D into two diagonal matrices $D1$ and $D2$, where D_1 contains the positive entries and zero otherwise, and D_2 contains the absolute values of the negative entries and zero otherwise. Both D_1 and D_2 will have strictly positive entries.

3. So A can be re-expressed by using $D = D_1 - D_2$:

$$A = UDU^\top = U(D_1 - D_2)U^\top = UD_1U^\top - UD_2U^\top = B - C$$

Final Conclusion: YOUR FINAL ANSWER OR CONCLUSION HERE

We have shown that the symmetric matrix A can be expressed as $A = B - C$, where $B = UD_1U^\top$ and $C = UD_2U^\top$ with U orthogonal and D_1, D_2 diagonal with strictly positive entries.

Part 1(f) [5 points]

Derive a quadratic form $\begin{bmatrix} x \\ y \end{bmatrix}^\top A \begin{bmatrix} x \\ y \end{bmatrix}$ **that is equivalent to the quadratic function** $ax^2 + 2bxy + cy^2$**, where the off-diagonal entries of the matrix** A **should differ by exactly 10.** Show your work.

Solution.

1. The quadratic form can be written as:

$$\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Expanding it, we have:

$$a'x^2 + (b' + c')xy + d'y^2$$

2. By the given function $ax^2 + 2bxy + cy^2$, we have:

$$\begin{aligned} a' &= a \\ b' + c' &= 2b \\ d' &= c \end{aligned}$$

3. By the given property that the off-diagonal entries b' and c' differ by exactly 10, we have:

$$|b' - c'| = 10$$

4. Now a' and c' are certain, and we have two equations for finding b' and c':

$$\begin{aligned} b' + c' &= 2b \\ |b' - c'| &= 10 \end{aligned}$$

We can solve these equations to find b' and c' with two different solutions.

5. Once we have b' and c', we can form the matrix A as:

$$A = \begin{bmatrix} a & b' \\ c' & c \end{bmatrix}$$

Final Conclusion: YOUR FINAL ANSWER OR CONCLUSION HERE

This matrix A can represent the quadratic form equivalent to the quadratic function $ax^2 + 2bxy + cy^2$, where the off-diagonal entries of matrix A would differ by exactly 10.

Part 1(g) [5 points]

Write down the second-order Taylor expansion of $f(x, y) = \sin x + xy + e^y$ around $x = 0$ and $y = 0$ in matrix form. Show your work.

Solution.

To find the second-order Taylor expansion of $f(x, y) = \sin x + xy + e^y$ around $x = 0$ and $y = 0$, we need to write down the second-order Taylor expansion of $f(x, y)$:

$$f(x, y) \approx f(0, 0) + \nabla f(0, 0) \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \frac{1}{2} \begin{bmatrix} x & y \end{bmatrix} H(0, 0) \begin{bmatrix} x \\ y \end{bmatrix}$$

Where ∇f is the gradient of f and H is the Hessian matrix of f.

1. We can easily calculate the f(0,0):

$$f(0, 0) = \sin(0) + 0 \cdot 0 + e^0 = 1$$

2. Because the gradient of f is a vector of partial derivatives. So,

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} \cos x + y \\ x + e^y \end{bmatrix}$$

Therefore,

$$\nabla f(0, 0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

3. The Hessian matrix H consists of second-order partial derivatives:

$$H(x, y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} -\sin x & 1 \\ 1 & e^y \end{bmatrix}$$

At the point (0,0),

$$H(0, 0) = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

4. In all, the second-order Taylor expansion of f around (0,0) is:

$$f(x, y) \approx 1 + \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \frac{1}{2} \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Final Conclusion: YOUR FINAL ANSWER OR CONCLUSION HERE

$$f(x, y) \approx 1 + \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \frac{1}{2} \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Question 2 Probability [35 points]

List your collaborators on this question below.

Solution.

Students I got help from: No

Students I gave help to: No

External sources that I consulted (websites, books, notes, etc.): Google, Wikipedia

Part 2(a) [5 points]

Find a 2×3 matrix A such that $\|A\|_F = \sqrt{8}$ and $\|A\|_2 = \sqrt{5}$, where $\|\cdot\|_F$ denotes the Frobenius norm and $\|\cdot\|_2$ denotes the spectral norm. Explain each step of your reasoning.

Solution.

1. The Frobenius norm of a matrix is defined as the square root of the sum of the absolute squares of its elements. For a matrix $A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$, it is calculated as

$$\|A\|_F = \sqrt{a^2 + b^2 + c^2 + d^2 + e^2 + f^2}$$

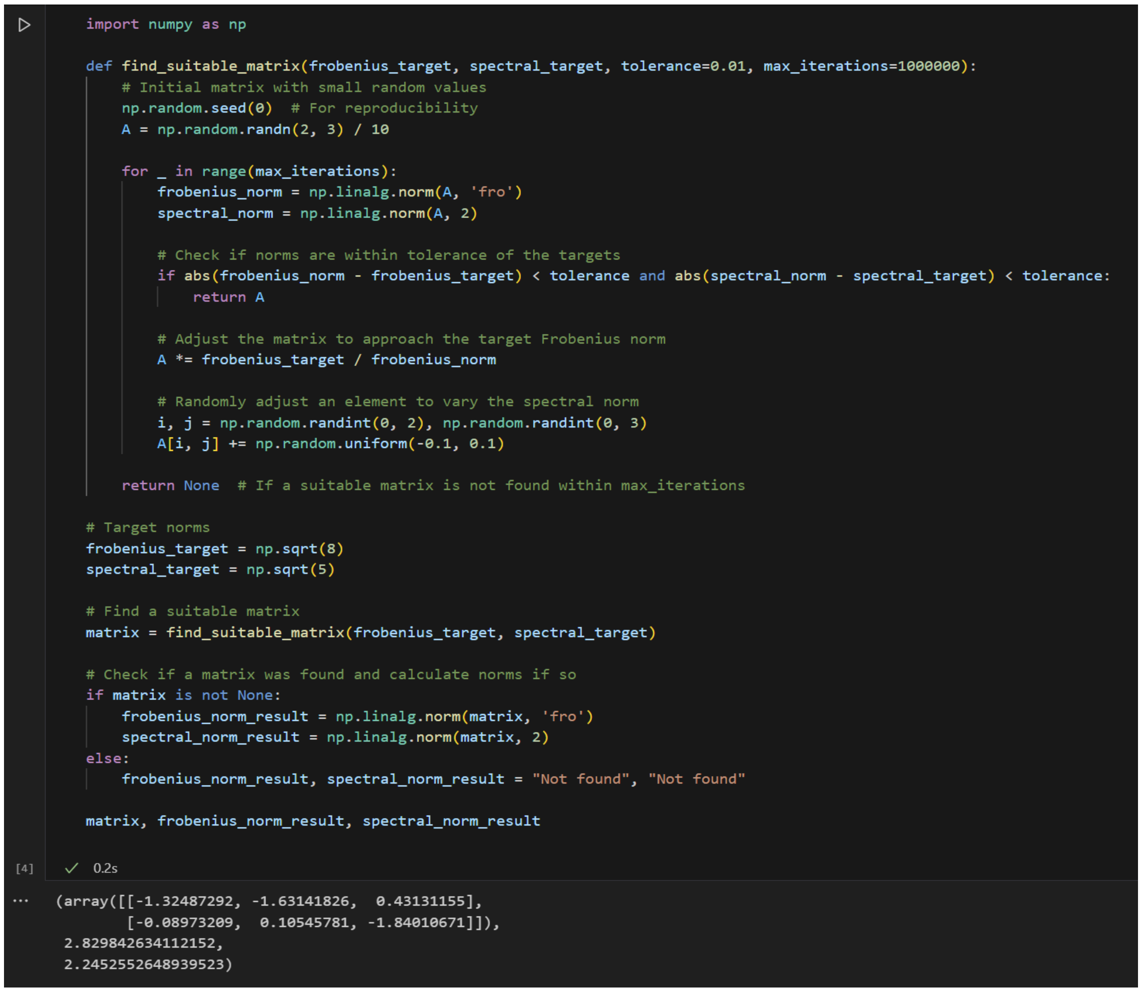
Given that $\|A\|_F = \sqrt{8}$, we have

$$a^2 + b^2 + c^2 + d^2 + e^2 + f^2 = 8$$

2. The spectral norm of a matrix is the largest singular value of that matrix. But we don't have an explicit formula for setting up such conditions because we need to find all the singular values of A first.

3. We can start by assigning values to the entries of A that satisfy the Frobenius norm condition and then adjust them to meet the spectral norm condition. A simple approach is to distribute the Frobenius norm evenly across the entries and then adjust as needed.

4. After constructing such a matrix, we would need to iteratively calculate its spectral norm (largest singular value) and ensure it equals $\sqrt{5}$.



```python
import numpy as np

def find_suitable_matrix(frobenius_target, spectral_target, tolerance=0.01, max_iterations=1000000):
    # Initial matrix with small random values
    np.random.seed(0)  # For reproducibility
    A = np.random.randn(2, 3) / 10

    for _ in range(max_iterations):
        frobenius_norm = np.linalg.norm(A, 'fro')
        spectral_norm = np.linalg.norm(A, 2)

        # Check if norms are within tolerance of the targets
        if abs(frobenius_norm - frobenius_target) < tolerance and abs(spectral_norm - spectral_target) < tolerance:
            return A

        # Adjust the matrix to approach the target Frobenius norm
        A *= frobenius_target / frobenius_norm

        # Randomly adjust an element to vary the spectral norm
        i, j = np.random.randint(0, 2), np.random.randint(0, 3)
        A[i, j] += np.random.uniform(-0.1, 0.1)

    return None  # If a suitable matrix is not found within max_iterations

# Target norms
frobenius_target = np.sqrt(8)
spectral_target = np.sqrt(5)

# Find a suitable matrix
matrix = find_suitable_matrix(frobenius_target, spectral_target)

# Check if a matrix was found and calculate norms if so
if matrix is not None:
    frobenius_norm_result = np.linalg.norm(matrix, 'fro')
    spectral_norm_result = np.linalg.norm(matrix, 2)
else:
    frobenius_norm_result, spectral_norm_result = "Not found", "Not found"

matrix, frobenius_norm_result, spectral_norm_result
```

[4] ✓ 0.2s

```
(array([[-1.32487292, -1.63141826,  0.43131155],
        [-0.08973209,  0.10545781, -1.84010671]]),
 2.829842634112152,
 2.2452552648939523)
```

Final Conclusion: YOUR FINAL ANSWER OR CONCLUSION HERE

As the code shows, one possible 2×3 matrix A can be:

$$\begin{bmatrix} -1.32487292 & -1.63141826 & 0.43131155 \\ -0.08973209 & 0.10545781 & -1.84010671 \end{bmatrix}$$

The calculated norms are:

Frobenius norm ($\|A\|_F$): Approximately 2.829842634112152

Spectral norm ($\|A\|_2$): Approximately 2.2452552648939523

Part 2(b) [5 points]

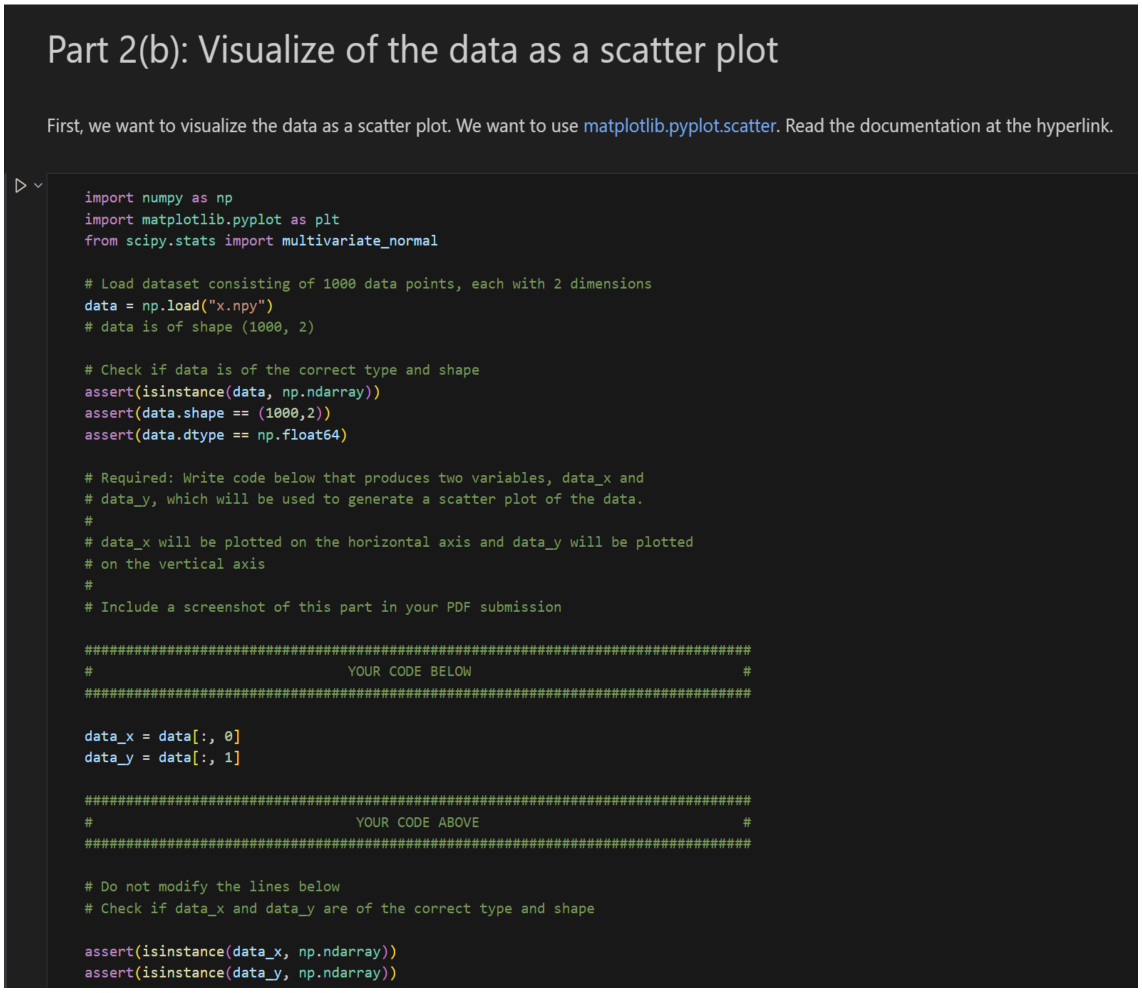
You are given samples of a 2-dimensional random vector $\vec{X}$ in the `x.npy` file. In this part, we want to try to fit a distribution to this data. First, plot the samples as a scatter plot to check how the samples are distributed. Subsequently, you have to fit a Gaussian distribution to the data. Given samples $\{\vec{x}_i\}_{i=1}^N$ of a random vector $\vec{X}$, the true mean $\vec{\mu} := E[\vec{X}]$ and true covariance $\Sigma := E[(\vec{X} - E[\vec{X}])(\vec{X} - E[\vec{X}])^\top]$ of $\vec{X}$ can be estimated with the sample mean $\widehat{\vec{\mu}}$ and the (biased) sample covariance $\widehat{\Sigma}$, the formulas of which are below:

$$\widehat{\vec{\mu}} = \frac{1}{N} \sum_{i=1}^{N} \vec{x}_i \tag{1}$$

$$\widehat{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} \left(\vec{x}_i - \widehat{\vec{\mu}}\right) \left(\vec{x}_i - \widehat{\vec{\mu}}\right)^\top \tag{2}$$

Find the sample mean and the sample covariance matrix by implementing the equations above on the given samples. You are not permitted to use the built-in Numpy functions for computing the sample mean and covariance, but you may use built-in Numpy functions for other elementary operations, such as summation, matrix multiplication, transposes, etc. **Then plot the contours of Gaussian distribution with the estimated parameters on the scatter plot of the samples. Include a screenshot of your code, a screenshot of the output and the plot of Gaussian contours in your PDF submission.**

Solution.



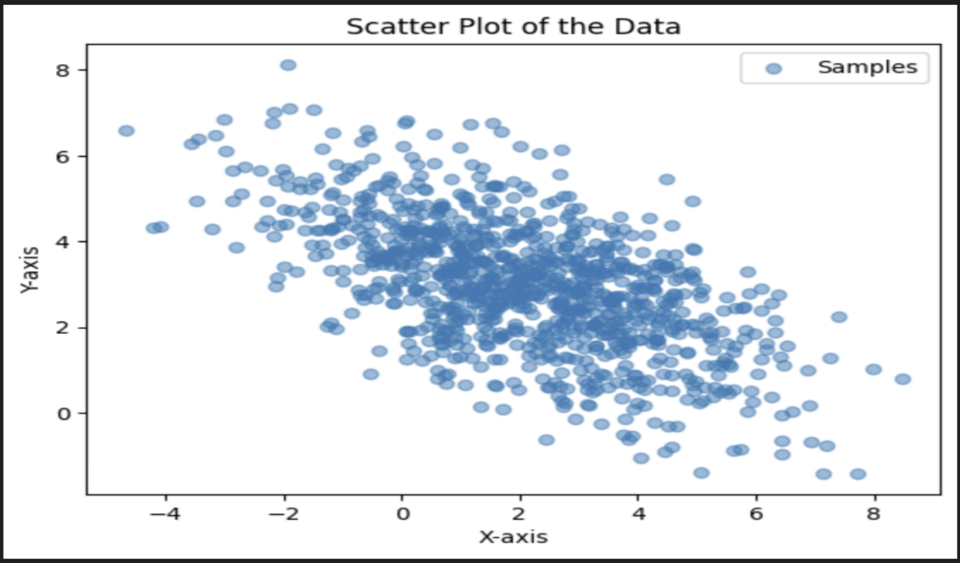
Part 2(b): Visualize of the data as a scatter plot

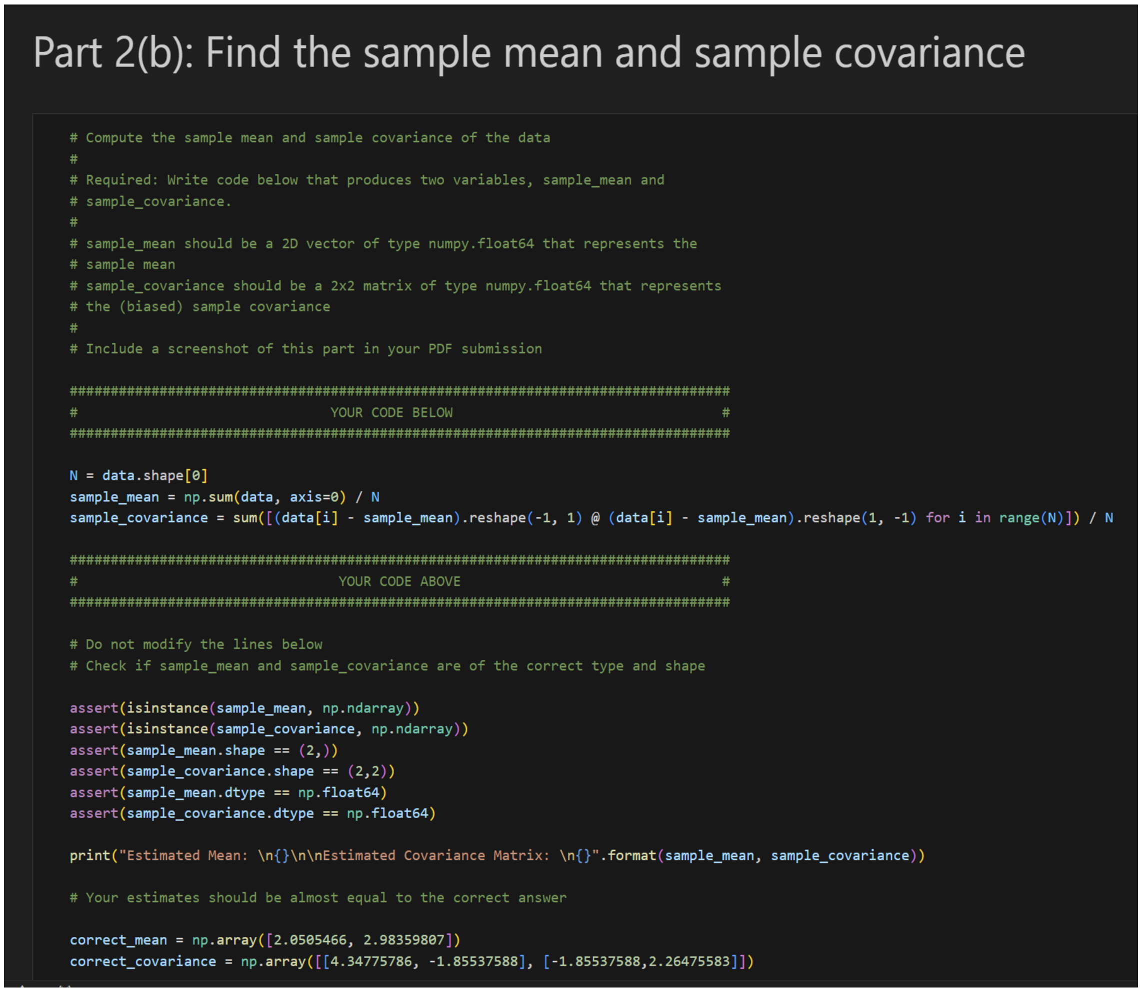
First, we want to visualize the data as a scatter plot. We want to use matplotlib.pyplot.scatter. Read the documentation at the hyperlink.

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

# Load dataset consisting of 1000 data points, each with 2 dimensions
data = np.load("x.npy")
# data is of shape (1000, 2)

# Check if data is of the correct type and shape
assert(isinstance(data, np.ndarray))
assert(data.shape == (1000,2))
assert(data.dtype == np.float64)

# Required: Write code below that produces two variables, data_x and
# data_y, which will be used to generate a scatter plot of the data.
#
# data_x will be plotted on the horizontal axis and data_y will be plotted
# on the vertical axis
#
# Include a screenshot of this part in your PDF submission

###################################################################################
#                                YOUR CODE BELOW                                  #
###################################################################################

data_x = data[:, 0]
data_y = data[:, 1]

###################################################################################
#                                 YOUR CODE ABOVE                                 #
###################################################################################

# Do not modify the lines below
# Check if data_x and data_y are of the correct type and shape

assert(isinstance(data_x, np.ndarray))
assert(isinstance(data_y, np.ndarray))
```

```
    assert(isinstance(data_x, np.ndarray))
    assert(isinstance(data_y, np.ndarray))
    assert(data_x.shape == (1000,))
    assert(data_y.shape == (1000,))
    assert(data_x.dtype == np.float64)
    assert(data_y.dtype == np.float64)

    # Plot the scatter plot of the given data
    plt.scatter(data_x, data_y, label='Samples', alpha=0.5)

    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title('Scatter Plot of the Data')
    plt.legend()
    plt.show()
```

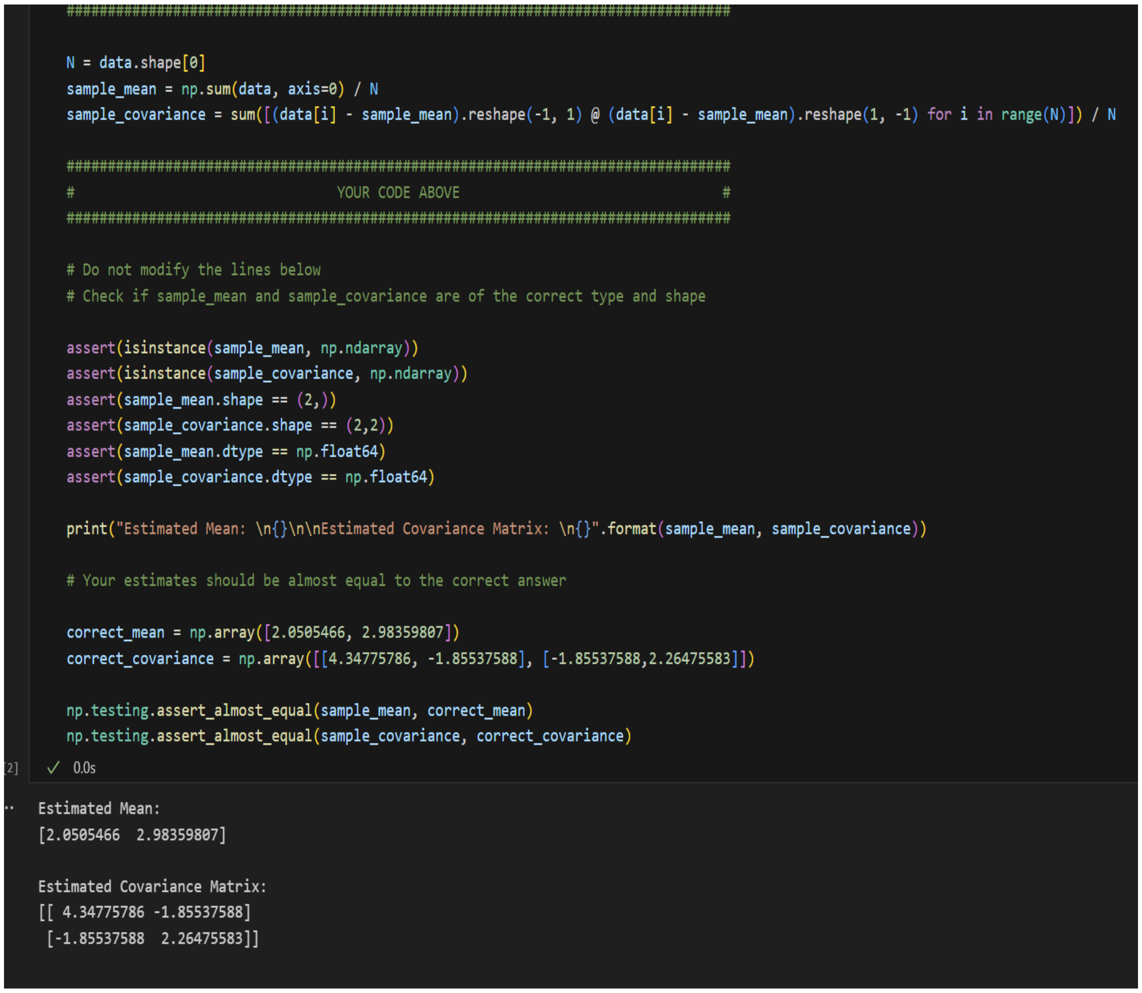
```
✓ 1.0s
```





Part 2(b): Find the sample mean and sample covariance

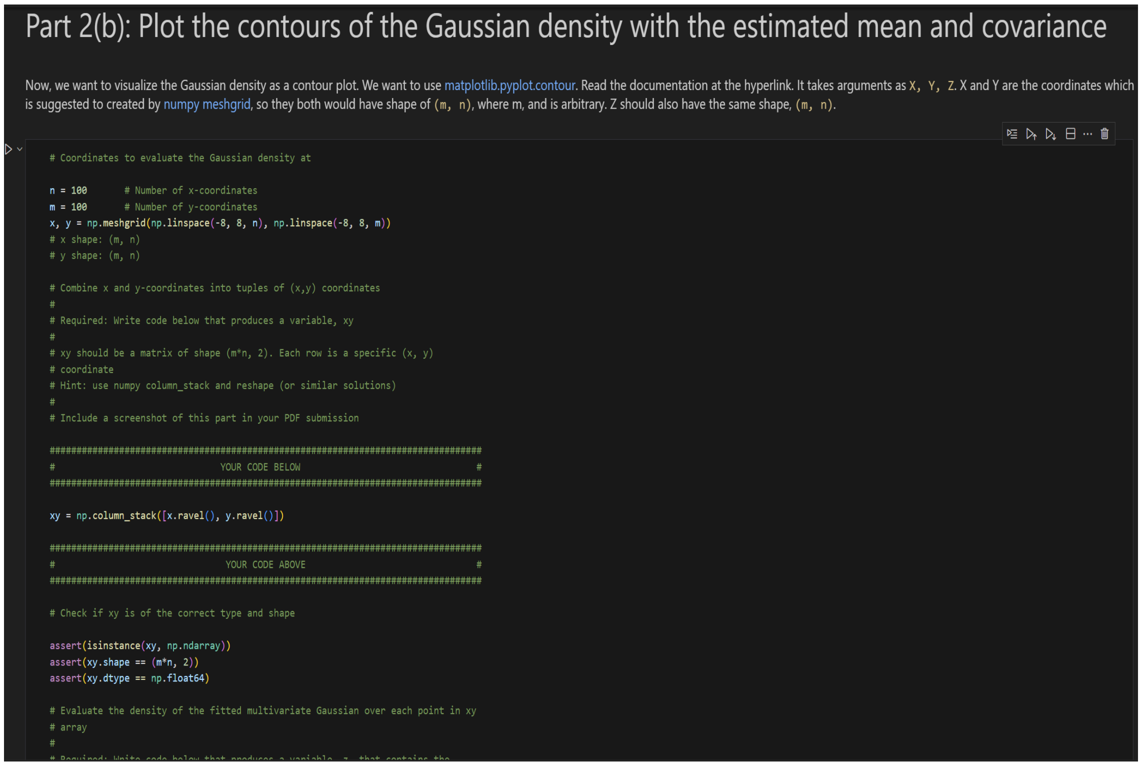
```python
# Compute the sample mean and sample covariance of the data
#
# Required: Write code below that produces two variables, sample_mean and
# sample_covariance.
#
# sample_mean should be a 2D vector of type numpy.float64 that represents the
# sample mean
# sample_covariance should be a 2x2 matrix of type numpy.float64 that represents
# the (biased) sample covariance
#
# Include a screenshot of this part in your PDF submission

#################################################################################
#                               YOUR CODE BELOW                                 #
#################################################################################

N = data.shape[0]
sample_mean = np.sum(data, axis=0) / N
sample_covariance = sum([(data[i] - sample_mean).reshape(-1, 1) @ (data[i] - sample_mean).reshape(1, -1) for i in range(N)]) / N

#################################################################################
#                                YOUR CODE ABOVE                                #
#################################################################################

# Do not modify the lines below
# Check if sample_mean and sample_covariance are of the correct type and shape

assert(isinstance(sample_mean, np.ndarray))
assert(isinstance(sample_covariance, np.ndarray))
assert(sample_mean.shape == (2,))
assert(sample_covariance.shape == (2,2))
assert(sample_mean.dtype == np.float64)
assert(sample_covariance.dtype == np.float64)

print("Estimated Mean: \n{}\n\nEstimated Covariance Matrix: \n{}".format(sample_mean, sample_covariance))

# Your estimates should be almost equal to the correct answer

correct_mean = np.array([2.0505466, 2.98359807])
correct_covariance = np.array([[4.34775786, -1.85537588], [-1.85537588,2.26475583]])
```



```
    ################################################################################

    N = data.shape[0]
    sample_mean = np.sum(data, axis=0) / N
    sample_covariance = sum([(data[i] - sample_mean).reshape(-1, 1) @ (data[i] - sample_mean).reshape(1, -1) for i in range(N)]) / N

    ################################################################################
    #                                YOUR CODE ABOVE                               #
    ################################################################################

    # Do not modify the lines below
    # Check if sample_mean and sample_covariance are of the correct type and shape

    assert(isinstance(sample_mean, np.ndarray))
    assert(isinstance(sample_covariance, np.ndarray))
    assert(sample_mean.shape == (2,))
    assert(sample_covariance.shape == (2,2))
    assert(sample_mean.dtype == np.float64)
    assert(sample_covariance.dtype == np.float64)

    print("Estimated Mean: \n{}\n\nEstimated Covariance Matrix: \n{}".format(sample_mean, sample_covariance))

    # Your estimates should be almost equal to the correct answer

    correct_mean = np.array([2.0505466, 2.98359807])
    correct_covariance = np.array([[4.34775786, -1.85537588], [-1.85537588,2.26475583]])

    np.testing.assert_almost_equal(sample_mean, correct_mean)
    np.testing.assert_almost_equal(sample_covariance, correct_covariance)
```

[2] ✓ 0.0s

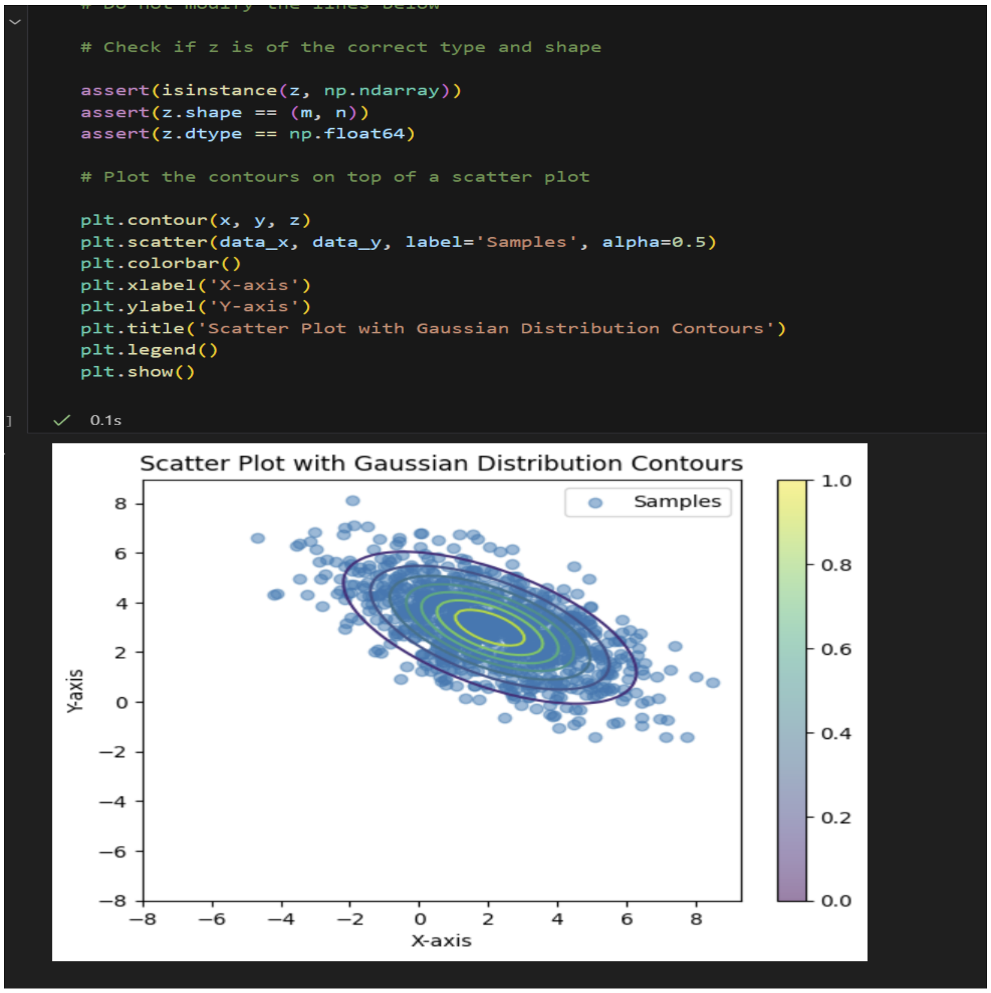
```
Estimated Mean:
[2.0505466  2.98359807]

Estimated Covariance Matrix:
[[ 4.34775786 -1.85537588]
 [-1.85537588  2.26475583]]
```



Part 2(b): Plot the contours of the Gaussian density with the estimated mean and covariance

Now, we want to visualize the Gaussian density as a contour plot. We want to use matplotlib.pyplot.contour. Read the documentation at the hyperlink. It takes arguments as `X, Y, Z`. X and Y are the coordinates which is suggested to created by numpy meshgrid, so they both would have shape of `(m, n)`, where m, and is arbitrary. Z should also have the same shape, `(m, n)`.

```
# Coordinates to evaluate the Gaussian density at

n = 100      # Number of x-coordinates
m = 100      # Number of y-coordinates
x, y = np.meshgrid(np.linspace(-8, 8, n), np.linspace(-8, 8, m))
# x shape: (m, n)
# y shape: (m, n)

# Combine x and y-coordinates into tuples of (x,y) coordinates
#
# Required: Write code below that produces a variable, xy
#
# xy should be a matrix of shape (m*n, 2). Each row is a specific (x, y)
# coordinate
# Hint: use numpy column_stack and reshape (or similar solutions)
#
# Include a screenshot of this part in your PDF submission

################################################################################
#                               YOUR CODE BELOW                                #
################################################################################

xy = np.column_stack([x.ravel(), y.ravel()])

################################################################################
#                               YOUR CODE ABOVE                                #
################################################################################

# Check if xy is of the correct type and shape

assert(isinstance(xy, np.ndarray))
assert(xy.shape == (m*n, 2))
assert(xy.dtype == np.float64)

# Evaluate the density of the fitted multivariate Gaussian over each point in xy
# array
#
# Required: Write code below that produces a variable, z, that contains the
```



```
    # Do not modify the lines below

    # Check if z is of the correct type and shape

    assert(isinstance(z, np.ndarray))
    assert(z.shape == (m, n))
    assert(z.dtype == np.float64)

    # Plot the contours on top of a scatter plot

    plt.contour(x, y, z)
    plt.scatter(data_x, data_y, label='Samples', alpha=0.5)
    plt.colorbar()
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title('Scatter Plot with Gaussian Distribution Contours')
    plt.legend()
    plt.show()
```

✓ 0.1s

Final Conclusion: YOUR FINAL ANSWER OR CONCLUSION HERE

As the images shown above

Part 2(c) [5 points]

Then consider two other random vectors $\vec{Y} = A\vec{X} + \vec{b}$, and $\vec{Z} = C\vec{X} + \vec{d}$. **Find A and $\vec{b}$ such that the probability density of $\vec{Y}$ is the same as the probability density $\vec{X}$ rotated about the origin by 30 degrees counterclockwise and then translated by $[-1, -1]$, and C and $\vec{d}$ such that the probability density of $\vec{Z}$ is the same as the probability density $\vec{X}$ translated by $[-1, -1]$ and then rotated about the origin by 30 degrees counterclockwise. Plot the probability density of $\vec{Y}$ and $\vec{Z}$ as contour plots.** Show your work and justify why your answers achieve the desired effect.

Solution.

1. For $\vec{Y}$ Rotate then Translate

Rotation by 30° counterclockwise is achieved by the rotation matrix:

$$A = \begin{bmatrix} \cos(30°) & -\sin(30°) \\ \sin(30°) & \cos(30°) \end{bmatrix}$$

Translation by [-1, -1] is achieved by:

$$\vec{b} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

2. For $\vec{Z}$ Translate then Rotate

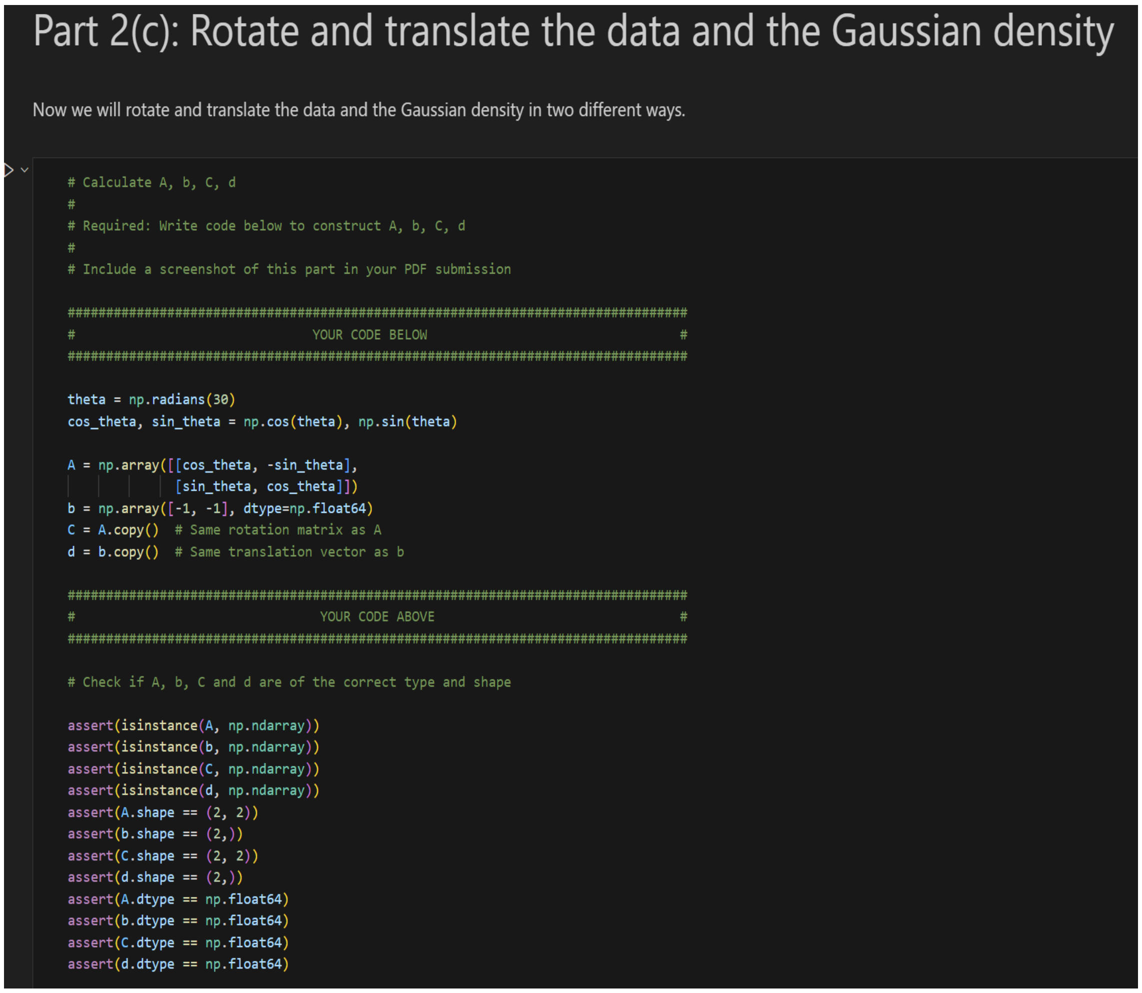
Translation by [-1, -1] is achieved by:

$$\vec{d} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

Rotation by 30° counterclockwise is achieved by the rotation matrix:

$$C = \begin{bmatrix} \cos(30°) & -\sin(30°) \\ \sin(30°) & \cos(30°) \end{bmatrix}$$

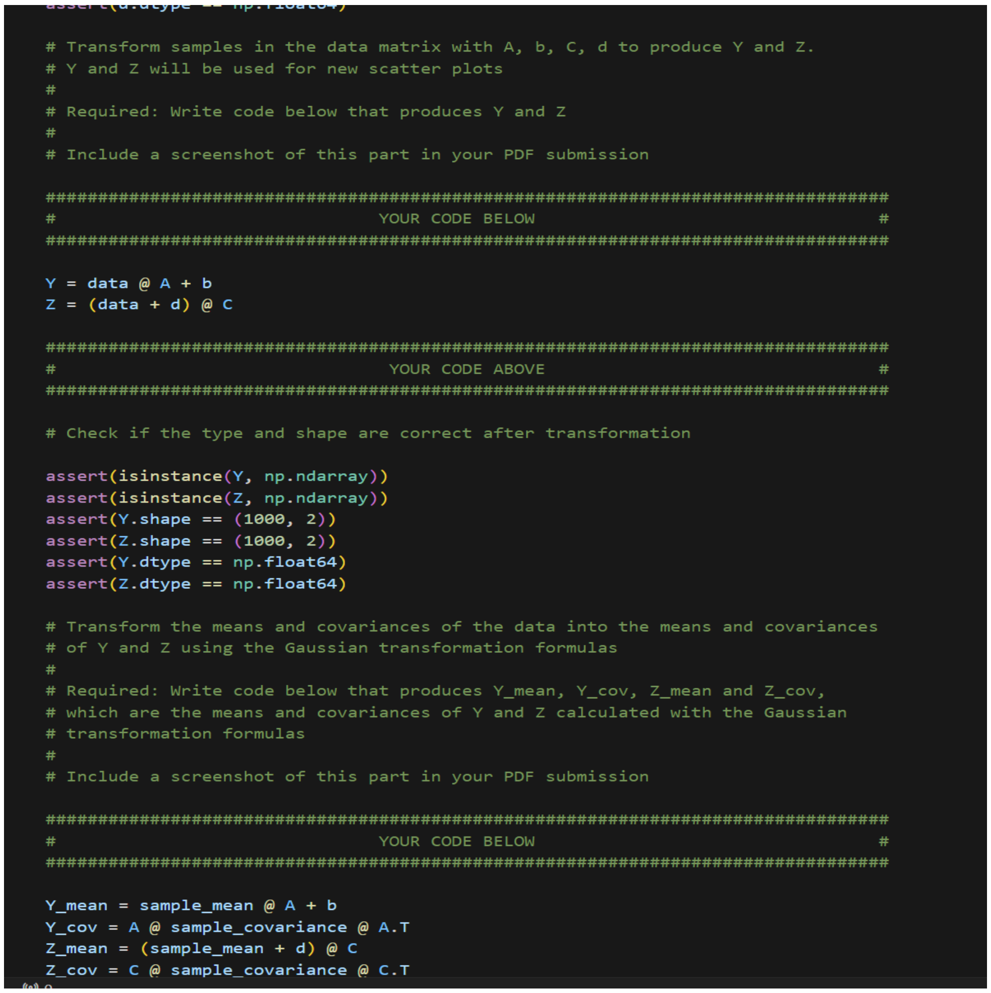
Final Conclusion: YOUR FINAL ANSWER OR CONCLUSION HERE



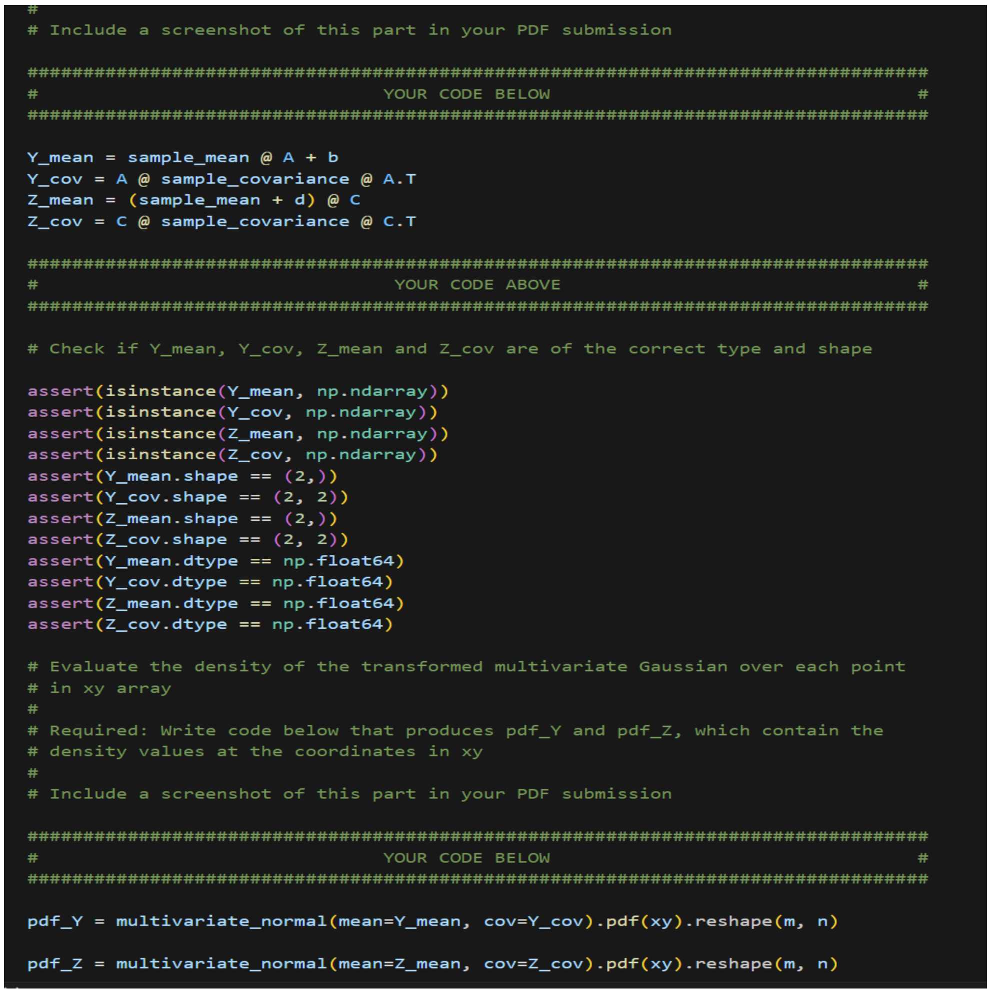
Part 2(c): Rotate and translate the data and the Gaussian density

Now we will rotate and translate the data and the Gaussian density in two different ways.

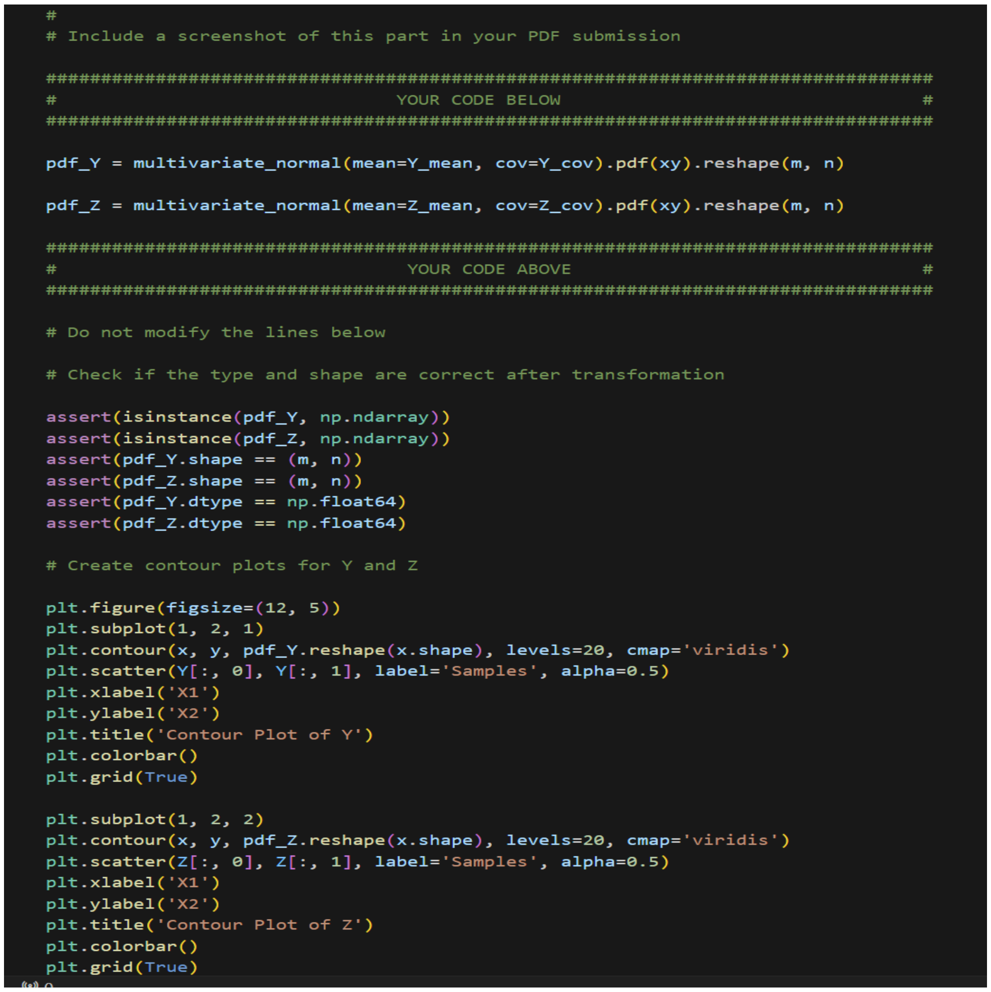
```
# Calculate A, b, C, d
#
# Required: Write code below to construct A, b, C, d
#
# Include a screenshot of this part in your PDF submission

###################################################################################
#                                YOUR CODE BELOW                                  #
###################################################################################

theta = np.radians(30)
cos_theta, sin_theta = np.cos(theta), np.sin(theta)

A = np.array([[cos_theta, -sin_theta],
              [sin_theta, cos_theta]])
b = np.array([-1, -1], dtype=np.float64)
C = A.copy()  # Same rotation matrix as A
d = b.copy()  # Same translation vector as b

###################################################################################
#                                 YOUR CODE ABOVE                                 #
###################################################################################

# Check if A, b, C and d are of the correct type and shape

assert(isinstance(A, np.ndarray))
assert(isinstance(b, np.ndarray))
assert(isinstance(C, np.ndarray))
assert(isinstance(d, np.ndarray))
assert(A.shape == (2, 2))
assert(b.shape == (2,))
assert(C.shape == (2, 2))
assert(d.shape == (2,))
assert(A.dtype == np.float64)
assert(b.dtype == np.float64)
assert(C.dtype == np.float64)
assert(d.dtype == np.float64)
```



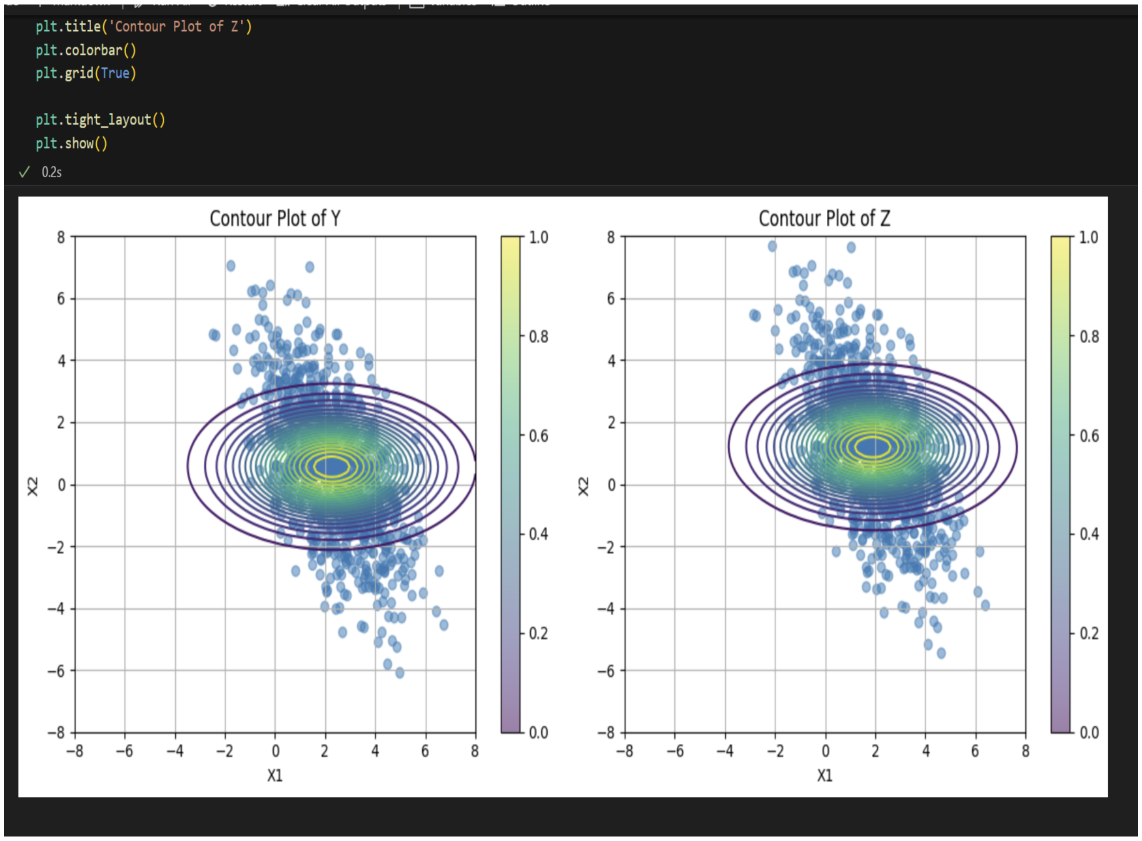
```
    assert(d.dtype == np.float64)

    # Transform samples in the data matrix with A, b, C, d to produce Y and Z.
    # Y and Z will be used for new scatter plots
    #
    # Required: Write code below that produces Y and Z
    #
    # Include a screenshot of this part in your PDF submission

    ################################################################################
    #                                YOUR CODE BELOW                               #
    ################################################################################

    Y = data @ A + b
    Z = (data + d) @ C

    ################################################################################
    #                                 YOUR CODE ABOVE                              #
    ################################################################################

    # Check if the type and shape are correct after transformation

    assert(isinstance(Y, np.ndarray))
    assert(isinstance(Z, np.ndarray))
    assert(Y.shape == (1000, 2))
    assert(Z.shape == (1000, 2))
    assert(Y.dtype == np.float64)
    assert(Z.dtype == np.float64)

    # Transform the means and covariances of the data into the means and covariances
    # of Y and Z using the Gaussian transformation formulas
    #
    # Required: Write code below that produces Y_mean, Y_cov, Z_mean and Z_cov,
    # which are the means and covariances of Y and Z calculated with the Gaussian
    # transformation formulas
    #
    # Include a screenshot of this part in your PDF submission

    ################################################################################
    #                                YOUR CODE BELOW                               #
    ################################################################################

    Y_mean = sample_mean @ A + b
    Y_cov = A @ sample_covariance @ A.T
    Z_mean = (sample_mean + d) @ C
    Z_cov = C @ sample_covariance @ C.T
```



```
#
# Include a screenshot of this part in your PDF submission

##################################################################################
#                                 YOUR CODE BELOW                                #
##################################################################################

Y_mean = sample_mean @ A + b
Y_cov = A @ sample_covariance @ A.T
Z_mean = (sample_mean + d) @ C
Z_cov = C @ sample_covariance @ C.T

##################################################################################
#                                 YOUR CODE ABOVE                                #
##################################################################################

# Check if Y_mean, Y_cov, Z_mean and Z_cov are of the correct type and shape

assert(isinstance(Y_mean, np.ndarray))
assert(isinstance(Y_cov, np.ndarray))
assert(isinstance(Z_mean, np.ndarray))
assert(isinstance(Z_cov, np.ndarray))
assert(Y_mean.shape == (2,))
assert(Y_cov.shape == (2, 2))
assert(Z_mean.shape == (2,))
assert(Z_cov.shape == (2, 2))
assert(Y_mean.dtype == np.float64)
assert(Y_cov.dtype == np.float64)
assert(Z_mean.dtype == np.float64)
assert(Z_cov.dtype == np.float64)

# Evaluate the density of the transformed multivariate Gaussian over each point
# in xy array
#
# Required: Write code below that produces pdf_Y and pdf_Z, which contain the
# density values at the coordinates in xy
#
# Include a screenshot of this part in your PDF submission

##################################################################################
#                                 YOUR CODE BELOW                                #
##################################################################################

pdf_Y = multivariate_normal(mean=Y_mean, cov=Y_cov).pdf(xy).reshape(m, n)

pdf_Z = multivariate_normal(mean=Z_mean, cov=Z_cov).pdf(xy).reshape(m, n)
```



```
#
# Include a screenshot of this part in your PDF submission

###################################################################################
#                                YOUR CODE BELOW                                  #
###################################################################################

pdf_Y = multivariate_normal(mean=Y_mean, cov=Y_cov).pdf(xy).reshape(m, n)

pdf_Z = multivariate_normal(mean=Z_mean, cov=Z_cov).pdf(xy).reshape(m, n)

###################################################################################
#                                YOUR CODE ABOVE                                  #
###################################################################################

# Do not modify the lines below

# Check if the type and shape are correct after transformation

assert(isinstance(pdf_Y, np.ndarray))
assert(isinstance(pdf_Z, np.ndarray))
assert(pdf_Y.shape == (m, n))
assert(pdf_Z.shape == (m, n))
assert(pdf_Y.dtype == np.float64)
assert(pdf_Z.dtype == np.float64)

# Create contour plots for Y and Z

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.contour(x, y, pdf_Y.reshape(x.shape), levels=20, cmap='viridis')
plt.scatter(Y[:, 0], Y[:, 1], label='Samples', alpha=0.5)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Contour Plot of Y')
plt.colorbar()
plt.grid(True)

plt.subplot(1, 2, 2)
plt.contour(x, y, pdf_Z.reshape(x.shape), levels=20, cmap='viridis')
plt.scatter(Z[:, 0], Z[:, 1], label='Samples', alpha=0.5)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Contour Plot of Z')
plt.colorbar()
plt.grid(True)
```



```
plt.title('Contour Plot of Z')
plt.colorbar()
plt.grid(True)

plt.tight_layout()
plt.show()
```

✓ 0.2s

Part 2(d) [5 points]

Suppose the random vector $\vec{X}$ is distributed according to a multivariate Gaussian with mean $\vec{0}$ and covariance matrix $AA^\top$. In other words, $\vec{X} \sim \mathcal{N}(\vec{0}, AA^\top)$. **Find a unit vector $\vec{b}$ such that the variance of the random variable $\left|\vec{b}^\top \vec{X}\right|$ is minimized, i.e.,** $\arg\min_{\vec{b}:\|\vec{b}\|_2=1} \mathrm{Var}\left(\left|\vec{b}^\top \vec{X}\right|\right)$. Show your work and justify why your answer is correct.

Hint 1: For any random variable U, $\mathrm{Var}(|U|) = \mathbb{E}[|U|^2] - \mathbb{E}[|U|]^2 = \mathbb{E}[U^2] - \mathbb{E}[|U|]^2 = \mathrm{Var}(U) + \mathbb{E}[U]^2 - \mathbb{E}[|U|]^2$
Hint 2: For any random variable U, $\mathrm{Var}(\sigma U) - \mathbb{E}[|\sigma U|]^2 = \sigma^2 \mathrm{Var}(U) - \sigma^2 \mathbb{E}[|U|]^2 = \sigma^2(\mathrm{Var}(U) - \mathbb{E}[|U|]^2)$

Solution.

1. Since $\vec{X} \sim \mathcal{N}(\vec{0}, AA^\top)$, the random variable $U = \vec{b}^\top \vec{X} \sim \mathcal{N}(\vec{0}, \vec{b}^\top AA^\top \vec{b})$.

2. From Hint 1,

$$\mathrm{Var}(|U|) = \mathrm{Var}(U) + \mathbb{E}[U]^2 - \mathbb{E}[|U|]^2$$

Since U has mean 0, $\mathbb{E}[U] = 0$, the equation can be simplified to

$$\mathrm{Var}(|U|) = \mathrm{Var}(U) - \mathbb{E}[|U|]^2$$

3. If U is a normally distributed variable with mean 0 and variance σ^2, then its probability density function (pdf) is given by:

$$f(u) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{u^2}{2\sigma^2}\right)$$

4. The expectation of $|U|$ can be calculated as the integral of $|U|$ times its probability density function over all possible values. Due to the symmetry of the normal distribution around zero, this can be simplified to twice the integral from 0 to infinity:

$$\mathbb{E}[|U|] = 2\int_0^\infty u \cdot f(u)\, du.$$

5. Substituting the PDF of U into the integral, we have:

$$\mathbb{E}[|U|] = 2\int_0^\infty u \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{u^2}{2\sigma^2}\right) du = \sqrt{\frac{2}{\pi}} \cdot \sigma.$$

This integral is a standard result in probability theory and is derived using integration by substitution. Thus, the standard deviation $\sigma(U)$ is the square root of the variance of U:

$$\mathbb{E}[|U|] = \sqrt{\frac{2}{\pi}} \cdot \sigma(U).$$

6. Substitute $\mathbb{E}[|U|]$ in the expression for $\mathrm{Var}(|U|)$, we have:

$$\mathrm{Var}(|U|) = \mathrm{Var}(U) - \mathbb{E}[|U|]^2 = \sigma(U)^2 - (\sqrt{\frac{2}{\pi}} \cdot \sigma)^2 = (1 - \frac{2}{\pi}) \cdot \sigma(U)^2$$

Where $(1 - \frac{2}{\pi})$ is always positive, so we just need to minimize $\sigma(U)^2$ which is $\vec{b}^\top AA^\top \vec{b}$.

This expression needs to be minimized subject to $\|\vec{b}\|_2 = 1$.

7. To solve this optimization problem, we can use the method of Lagrange multipliers or observe that the minimization problem is equivalent to minimizing $\vec{b}^\top AA^\top \vec{b}$ subject to $\|\vec{b}\|_2 = 1$. This is a standard eigenvalue problem, and the solution is the eigenvector corresponding to the smallest eigenvalue of $AA^\top$.

Therefore, $\vec{b}$ should be chosen as the eigenvector corresponding to the smallest eigenvalue of the covariance matrix $AA^\top$. This eigenvector minimizes the variance of the linear combination $\vec{b}^\top \vec{X}$ and thus minimizes the variance of $\left|\vec{b}^\top \vec{X}\right|$.

Final Conclusion: YOUR FINAL ANSWER OR CONCLUSION HERE

$\vec{b}$ should be chosen as the eigenvector corresponding to the smallest eigenvalue of the covariance matrix $AA^\top$.

Part 2(e) [5 points]

40 people play a game in which they have to choose an integer between 1 and 100 inclusive, independently of others. **For a particular pair of people, what is the probability that they choose the same integer?** Show your work.

Solution.

Since the choices are independent, the probability that both people choose the same integer is the probability that the second person chooses the same integer as the first person. The first person can choose any number, and then the second person must choose that same number.

The probability that they choose the same integer is given by:

$$P(\text{same choice}) = \frac{\text{Number of matching choices}}{\text{Total choices for the second person}}$$

Since there is only 1 matching choice (the same number as the first person chose) and 100 total choices for the second person, the probability is:

$$P(\text{same choice}) = \frac{1}{100}$$

Thus, the probability that a particular pair of people choose the same integer is 1%.

Final Conclusion: YOUR FINAL ANSWER OR CONCLUSION HERE

1%.

Part 2(f) [5 points]

Consider the same game as in the previous part. **Find the expected number of pairs of people that choose the same integer.** Show your work.

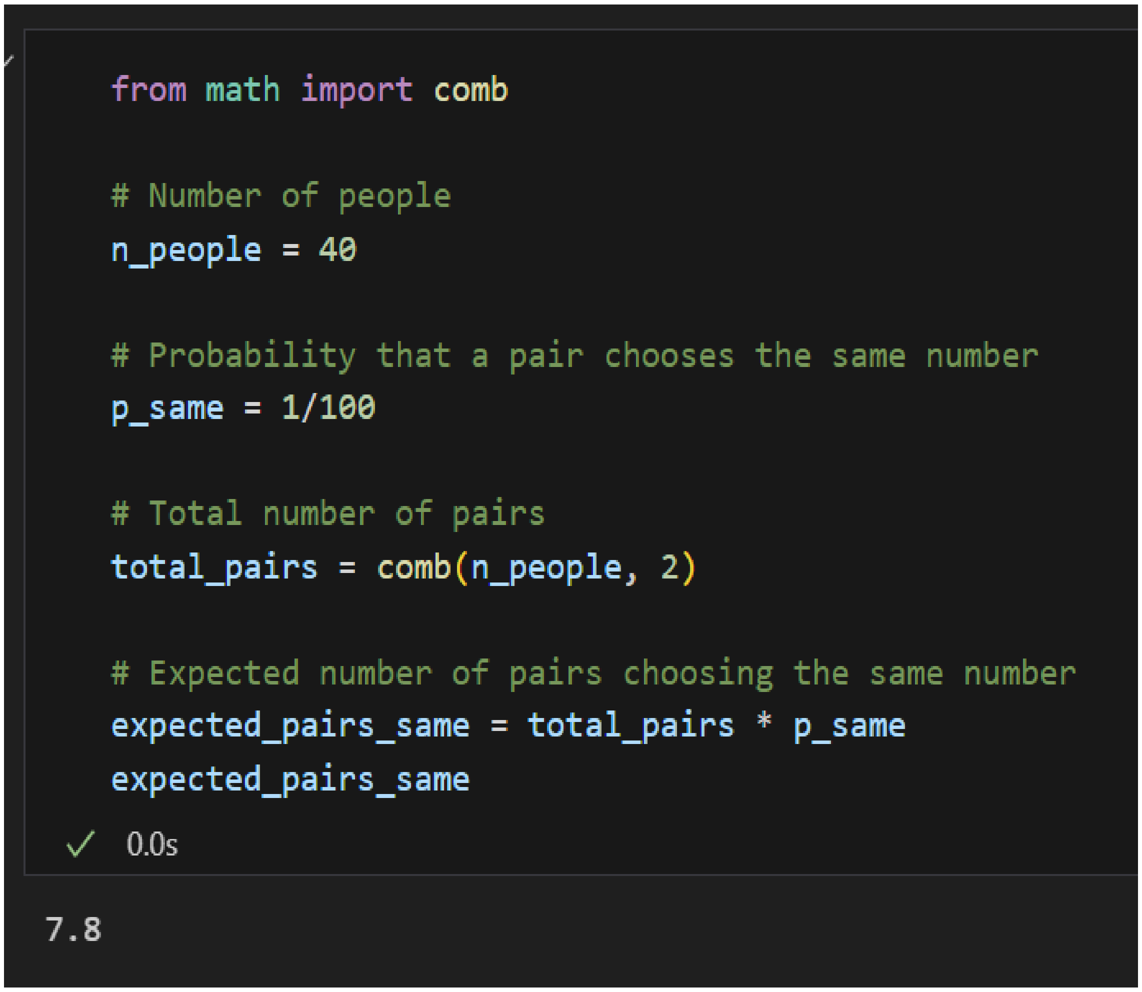
Hint: This expectation is of a sum of appropriately defined Bernoulli random variables.

Solution.

1. Number of Possible Pairs in total: $\binom{40}{2}$
2. Probability of a Pair Choosing the Same Number: $\frac{1}{100}$
3. Expected Number of Pairs Choosing the Same Number: The expected number of pairs choosing the same number is the total number of pairs multiplied by the probability of each pair choosing the same number.

$$ExpectedNumberofPairs = \binom{40}{2} \times \frac{1}{100}$$

Final Conclusion: YOUR FINAL ANSWER OR CONCLUSION HERE



```
from math import comb

# Number of people
n_people = 40

# Probability that a pair chooses the same number
p_same = 1/100

# Total number of pairs
total_pairs = comb(n_people, 2)

# Expected number of pairs choosing the same number
expected_pairs_same = total_pairs * p_same
expected_pairs_same
```

✓ 0.0s

```
7.8
```

7.8.

Part 2(g) [5 points]

Consider the same game as in the previous two parts. **Find the probability that the number 50 is chosen by at least two people.** Show your work.

Solution.

The probability of at least two people choosing the number 50 is the complement of the probability that zero or only one person chooses it. By the property of Bernoulli distribution,

1. Probability that Nobody Chooses 50

$$P(\text{no one chooses } 50) = \binom{40}{0}\left(\frac{1}{100}\right)^0\left(\frac{99}{100}\right)^{40}$$

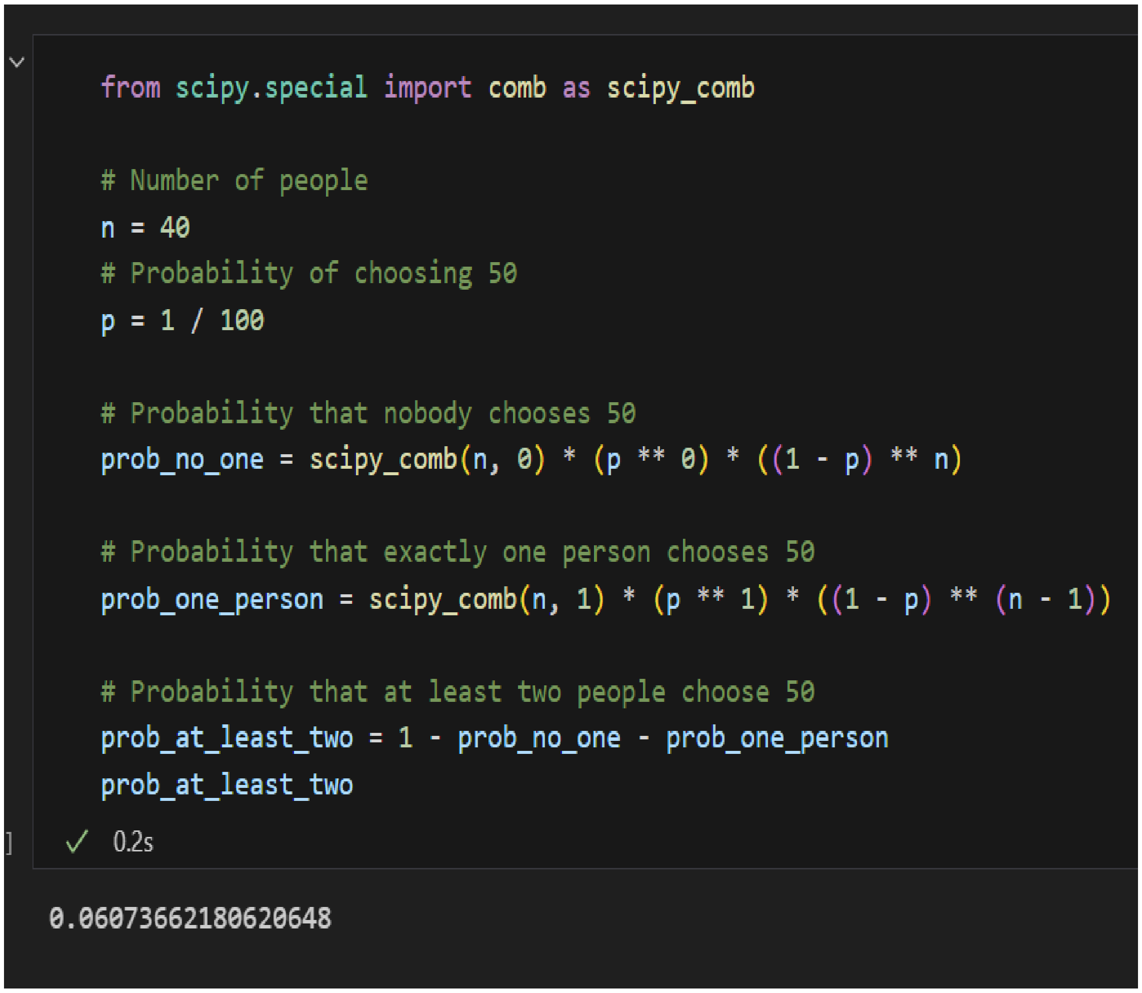
2. Probability that Only One Person Chooses 50

$$P(\text{exactly one person chooses } 50) = \binom{40}{1}\left(\frac{1}{100}\right)^1\left(\frac{99}{100}\right)^{39}$$

3. Probability that At Least Two People Choose 50

$$P(\text{at least two people choose } 50) = 1 - P(\text{no one chooses } 50) - P(\text{only one person chooses } 50)$$

Final Conclusion: YOUR FINAL ANSWER OR CONCLUSION HERE



```
from scipy.special import comb as scipy_comb

# Number of people
n = 40
# Probability of choosing 50
p = 1 / 100

# Probability that nobody chooses 50
prob_no_one = scipy_comb(n, 0) * (p ** 0) * ((1 - p) ** n)

# Probability that exactly one person chooses 50
prob_one_person = scipy_comb(n, 1) * (p ** 1) * ((1 - p) ** (n - 1))

# Probability that at least two people choose 50
prob_at_least_two = 1 - prob_no_one - prob_one_person
prob_at_least_two
```

✓ 0.2s

```
0.06073662180620648
```

6.07%.