Assignment 2

Omar Ghaleb
COMP 5107

In this assignment we have two classes X_1 and X_2 with means M_1 and M_2 where:

$$M_1 = \begin{bmatrix} 3 & 1 & 4 \end{bmatrix}, \quad M_2 = \begin{bmatrix} -3 & 1 & -4 \end{bmatrix}$$

and covariance matrices as follows:

$$\sum_{X_1} = \begin{bmatrix} a^2 & \beta ab & \alpha ac \\ \beta ab & b^2 & \beta bc \\ \alpha ac & \beta bc & c^2 \end{bmatrix}, \quad \sum_{X_2} = \begin{bmatrix} c^2 & \alpha bc & \beta ac \\ \alpha bc & b^2 & \alpha ab \\ \beta ac & \alpha ab & a^2 \end{bmatrix}$$

The parameters used in this assignment is as follows:

$$a = 2, \quad b = 3, \quad c = 4, \quad \alpha = 0.1, \quad \beta = 0.2, \quad \#points = 5000$$

This resulted the covariance matrices to have the following values:

$$\sum_{X_1} = \begin{bmatrix} 4 & 1.2 & 0.8 \\ 1.2 & 9 & 2.4 \\ 0.8 & 2.4 & 16 \end{bmatrix}, \quad \sum_{X_2} = \begin{bmatrix} 16 & 1.2 & 1.6 \\ 1.2 & 9 & 0.6 \\ 1.6 & 0.6 & 4 \end{bmatrix}$$

a. Gaussian random vectors generation:

In the first part we are required to generate Gaussian random vectors from uniform random variables only. The following method is used to create a 3D-vector that represents a single point.

Listing 1: Gaussian vector generation

```
# generating gaussian random vectors from Uniform random variables
def generate_point():
    dim = 3
    point = []
    for d in range(0, dim):
        z = 0
        for i in range(0, 12):
```

```
            rand = np.random.uniform(0, 1)
            z = z + rand
        z = z - 6
        point.append([z])
    point = np.array(point)
    return point
```

b. Simultaneous diagonalization Matrix:

In this part we are asked to create the diagonalizing matrix that is used for simultaneous diagonalization. The formula for the diagonalization:

$$V_1 = P^T_{overall} X_1 \quad and \quad V_2 = P^T_{overall} X_2$$

where

$$P_{overall} = (P^T_{Z_2})(\Lambda^{-1/2}_{X_1})(P^T_{X_1})$$

and $P^T_{Z_2}$ is the eigenvector matrix of covariance of Z_2 ($\sum_{Z_2}$). And

$$\sum_{Z_2} = (\Lambda^{-1/2}_{X_1} P^T_{Z_2}) \sum_{X_2} (P_{Z_2} \Lambda^{-1/2}_{X_1})$$

$$\sum_{Z_1} = I, \quad \sum_{Z_2} = \begin{bmatrix} 0.310 & 0.302 & -0.210 \\ 0.302 & 4.063 & -0.332 \\ -0.210 & -0.332 & 1.026 \end{bmatrix}$$

Thus,

$$P_{overall} = \begin{bmatrix} -0.511 & 0.067 & 0.004 \\ 0.024 & -0.002 & -0.251 \\ -0.004 & -0.340 & 0.051 \end{bmatrix}$$

Listing 2: Simultaneous Diagonalization

```
# creating the covariance matrices with the parameters
sigma_x1, sigma_x2 = covariance_matrix(a1, b1, c1, alpha1, beta1)

# eigenvalues and eigenvectors respectively
w_x1, v_x1 = np.linalg.eig(sigma_x1)
lambda_x1 = np.diag(w_x1)

w_x2, v_x2 = np.linalg.eig(sigma_x2)
lambda_x2 = np.diag(w_x2)

# create point matrices for the two classes X1 and X2
```

```python
z1_matrix, x1_matrix = generate_point_matrix(v_x1, lambda_x1, m1, number_of_points)
z2_matrix, x2_matrix = generate_point_matrix(v_x2, lambda_x2, m2, number_of_points)

# transform points for two classes in Y world
y1_matrix = v_x1.transpose() @ x1_matrix
y2_matrix = v_x1.transpose() @ x2_matrix

# transform points for the two classes in Z world
z1 = np.diag(np.power(w_x1, -0.5)) @ v_x1.transpose() @ x1_matrix
z2 = np.diag(np.power(w_x1, -0.5)) @ v_x1.transpose() @ x2_matrix

# covariance matrix of z1 and z2
sigma_z1 = np.diag(np.power(w_x1, -0.5)) @ np.diag(w_x1) @ np.diag(np.power(w_x1, -0.5))
sigma_z2 = np.diag(np.power(w_x1, -0.5)) @ v_x1.transpose() @ sigma_x2 @ v_x1 @ np.diag(np.power(w_x1, -0.5))

# eigenvalues and eigenvectors of z2 covariance
w_z1, v_z1 = np.linalg.eig(sigma_z1)
w_z2, v_z2 = np.linalg.eig(sigma_z2)

# the diagonalizing matrix p_overall
p_overall = v_z2.transpose() @ np.diag(np.power(w_x1, -0.5)) @ v_x1.transpose()

# transform points for the two classes in V
v1_matrix = p_overall @ x1_matrix
v2_matrix = p_overall @ x2_matrix
```

c. Generating 5000 points and plots:

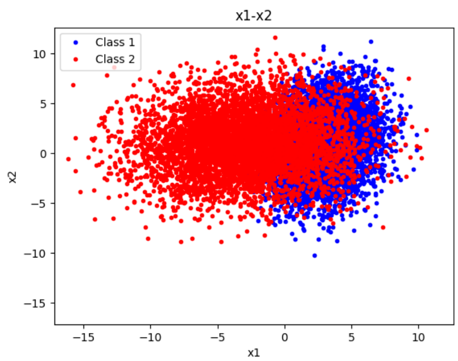
In this part we will show the code used for generating 5000 points from the Gaussian and transformed back to X world.

Listing 3: Generating 5000 points

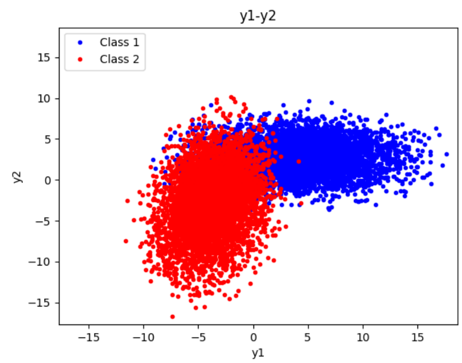
```python
# generate points from gaussian distribution and transform it back to class distribution
def generate_point_matrix(v, lambda_x, m, points):
    # create initial point
    z_matrix = generate_point()

    # convert them back to the classes distributions
```

```
    x_matrix = v @ np.power(lambda_x, 0.5) @ z_matrix + m

    # generate number of points and append them in an array
    for j in range(1, points):
        z_point = generate_point()
        z_matrix = np.append(z_matrix, z_point, axis=1)

        x = v @ np.power(lambda_x, 0.5) @ z_point + m
        x_matrix = np.append(x_matrix, x, axis=1)

    return z_matrix, x_matrix
```

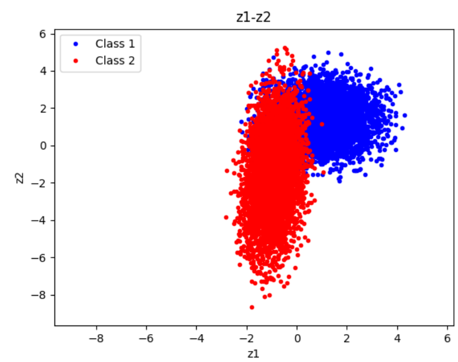
The following graphs show the points generated in the X world before diagonalization.



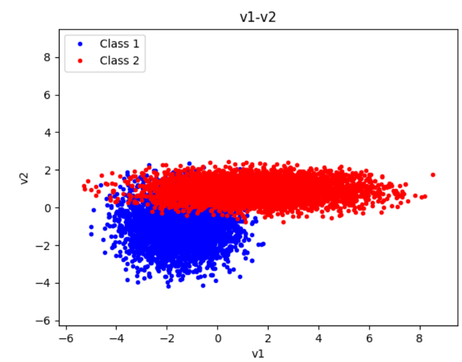
(a) x1-x2



(b) y1-y2

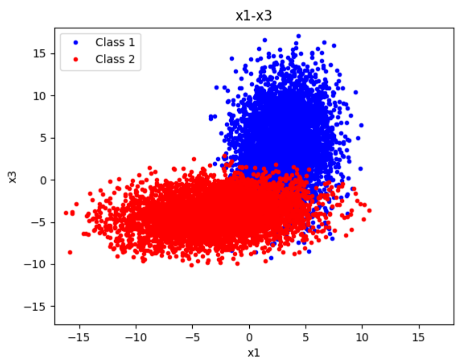


(c) z1-z2

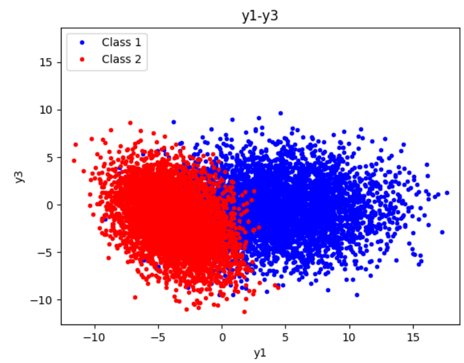


(d) v1-v2

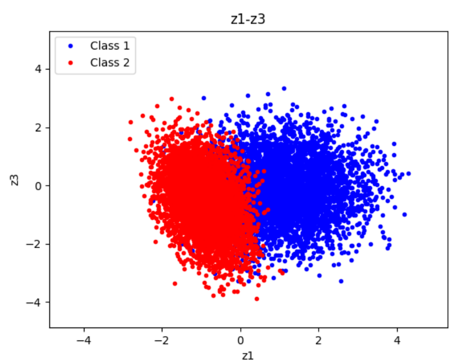
Figure 1: Transitions of (d1-d2) from X to V



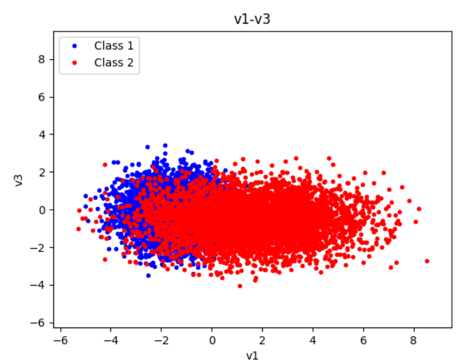
(a) x1-x3



(b) y1-y3

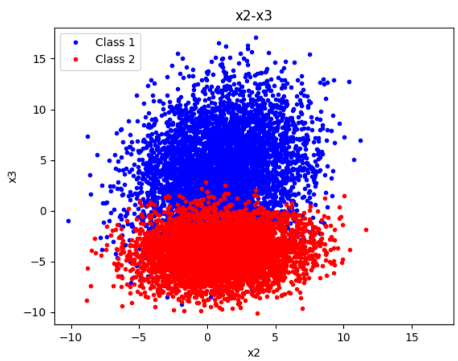


(c) z1-z3

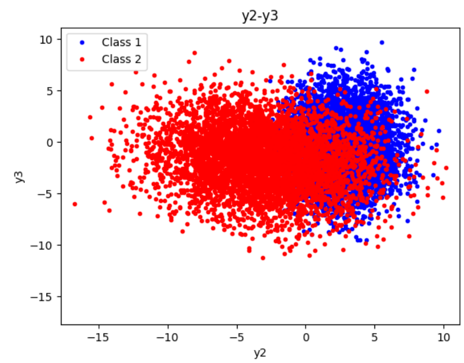


(d) v1-v3

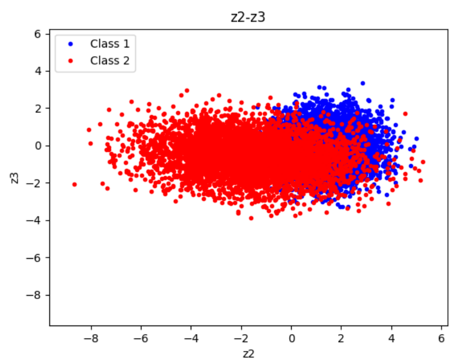
Figure 2: Transitions of (d1-d3) from X to V



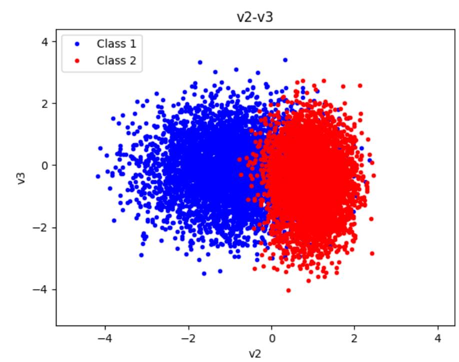
(a) x2-x3



(b) y2-y3

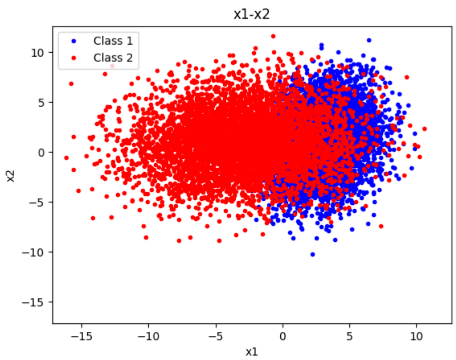


(c) z2-z3

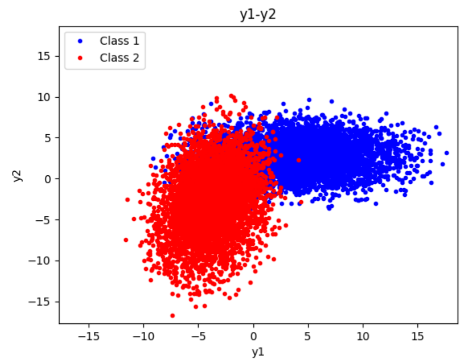


(d) v2-v3

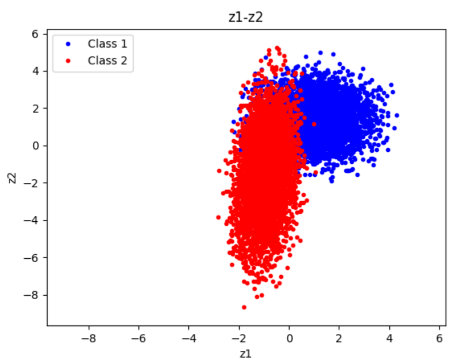
Figure 3: Transitions of (d2-d3) from X to V



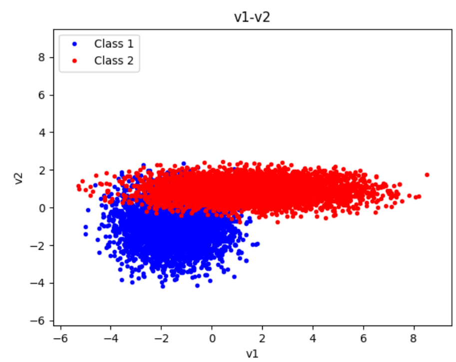
(a) x-3D



(b) y-3D



(c) z-3D



(d) v-3D

Figure 4: Transitions in 3D from X to V