

## 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

---

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

```

8         rand = np.random.uniform(0, 1)
9         z = z + rand
10        z = z - 6
11        point.append([z])
12    point = np.array(point)
13    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:

$$V_1 = P_{overall}^T X_1 \quad \text{and} \quad V_2 = P_{overall}^T X_2$$

where

$$P_{overall} = (P_{Z_2}^T)(\Lambda_{X_1}^{-1/2})(P_{X_1}^T)$$

and  $P_{Z_2}^T$  is the eigenvector matrix of covariance of  $Z_2$  ( $\Sigma_{Z_2}$ ). And

$$\Sigma_{Z_2} = (\Lambda_{X_1}^{-1/2} P_{Z_2}^T) \Sigma_{X_2} (P_{Z_2} \Lambda_{X_1}^{-1/2})$$

$$\Sigma_{Y_1} = \begin{bmatrix} 16.426 & 0 & 0 \\ 0 & 3.752 & 0 \\ 0 & 0 & 8.820 \end{bmatrix}, \quad \Sigma_{Y_2} = \begin{bmatrix} 5.228 & 2.392 & -2.505 \\ 2.392 & 15.118 & -1.861 \\ -2.505 & -1.861 & 8.653 \end{bmatrix}$$

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

Thus,

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

$$\Sigma_{V_1} = I, \quad \Sigma_{V_2} = \begin{bmatrix} 4.13 & 0 & 0 \\ 0 & 0.24 & 0 \\ 0 & 0.6 & 1.03 \end{bmatrix}$$

---

Listing 2: Simultaneous Diagonalization

---

```

1  # creating the covariance matrices with the parameters
2  sigma_x1, sigma_x2 = covariance_matrix(a1, b1, c1, alpha1, beta1)
3

```

```

4  # eigenvalues and eigenvectors respectively
5  w_x1, v_x1 = np.linalg.eig(sigma_x1)
6  lambda_x1 = np.diag(w_x1)
7
8  w_x2, v_x2 = np.linalg.eig(sigma_x2)
9  lambda_x2 = np.diag(w_x2)
10
11 # create point matrices for the two classes X1 and X2
12 z1_matrix, x1_matrix = generate_point_matrix(v_x1, lambda_x1, m1, ←
    number_of_points)
13 z2_matrix, x2_matrix = generate_point_matrix(v_x2, lambda_x2, m2, ←
    number_of_points)
14
15 # transform points for two classes in Y world
16 y1_matrix = v_x1.transpose() @ x1_matrix
17 y2_matrix = v_x1.transpose() @ x2_matrix
18
19 # covariances of y1 and y2 (classes 1 and 2 in Y world)
20 sigma_y1 = v_x1.transpose() @ sigma_x1 @ v_x1
21 sigma_y2 = v_x1.transpose() @ sigma_x2 @ v_x1
22
23 # transform points for the two classes in Z world
24 z1 = np.diag(np.power(w_x1, -0.5)) @ v_x1.transpose() @ x1_matrix
25 z2 = np.diag(np.power(w_x1, -0.5)) @ v_x1.transpose() @ x2_matrix
26
27 # covariance matrix of z1 and z2
28 sigma_z1 = np.diag(np.power(w_x1, -0.5)) @ np.diag(w_x1) @ np.diag←
    (np.power(w_x1, -0.5))
29 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))
30
31 # eigenvalues and eigenvectors of z2 covariance
32 w_z1, v_z1 = np.linalg.eig(sigma_z1)
33 w_z2, v_z2 = np.linalg.eig(sigma_z2)
34
35 # the diagonalizing matrix p_overall
36 p_overall = v_z2.transpose() @ np.diag(np.power(w_x1, -0.5)) @ ←
    v_x1.transpose()
37
38 # transform points for the two classes in V
39 v1_matrix = p_overall @ x1_matrix
40 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

---

```
1 # generate points from gaussian distribution and transform it back↔
   to class distribution
2 def generate_point_matrix(v, lambda_x, m, points):
3     # create initial point
4     z_matrix = generate_point()
5
6     # convert them back to the classes distributions
7     x_matrix = v @ np.power(lambda_x, 0.5) @ z_matrix + m
8
9     # generate number of points and append them in an array
10    for j in range(1, points):
11        z_point = generate_point()
12        z_matrix = np.append(z_matrix, z_point, axis=1)
13
14        x = v @ np.power(lambda_x, 0.5) @ z_point + m
15        x_matrix = np.append(x_matrix, x, axis=1)
16
17    return z_matrix, x_matrix
```

---

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

(a)(b)  
 $x_1 y_1 -$   
 $x_2 y_2$

(c)(d)  
 $z_1 - v_1 -$   
 $z_2 v_2$

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

(a)(b)  
 $x_1 y_1 -$   
 $x_3 y_3$

(c)(d)  
 $z_1 - v_1 -$   
 $z_3 v_3$

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

(a)(b)  
 $x_2 y_2 -$   
 $x_3 y_3$

(c)(d)  
 $z_2 - v_2 -$   
 $z_3 v_3$

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

(a)(b)  
 $x - y -$   
 $3D3D$

(c)(d)  
 $z - v -$   
 $3D3D$

Figure 4: Transitions in 3D from X to V