HW1

November 27, 2021

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

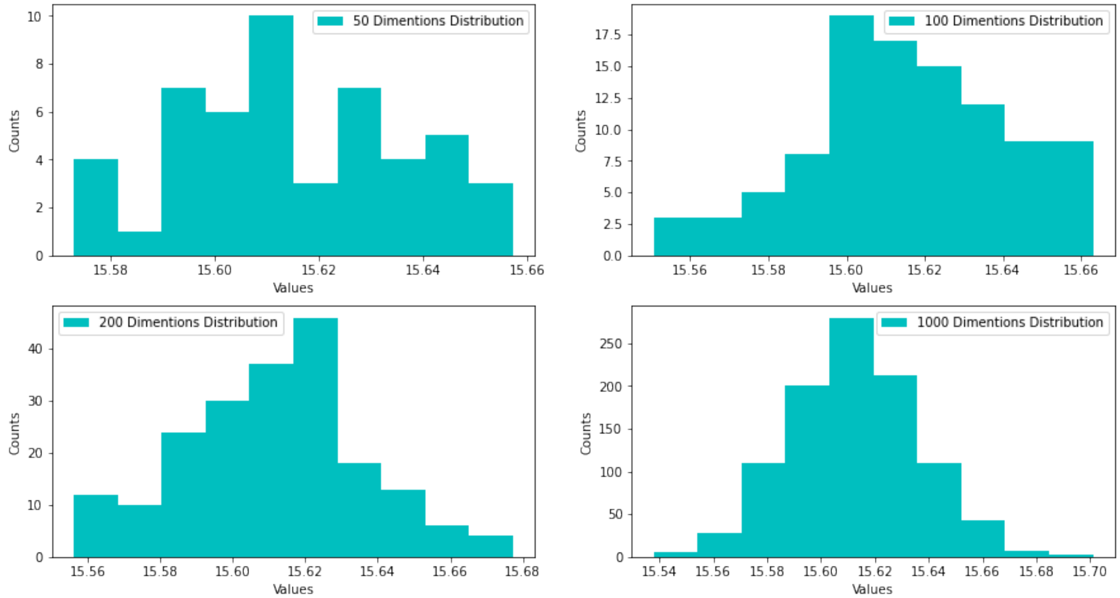
1 Question 2

[4]:
```
#################### The Functions for Question 2 ####################

## Generate identical and independece random variables with n dimension
def generate_iid( n, variable_count = 100000):
    ## n is the feature size of each X (dimension)
    ## variable_count is the count of random variables we want to make

    ## Fix the seed for reproduction to be the same each time we call the␣
↪function!
    np.random.seed(1)

    ## This will create Identical Random Variables
    ## Also It make them independence as possible
    X = []
    for _ in range(variable_count):
        x_generated = np.random.uniform(0,1,size = n)
        X.append(x_generated)

    return X

## Volume functoin is due to calculate the volume of the data
## Ex: With n = 3, we have a cube to calculate the volume
def volume_function(n, variable_count = 100000, varbose= False):
    ## n is the feature size of each X
    ## variable_count is the count of random variables we want to make
    ## verbose variable is weather to print the progress or not

    X_input = generate_iid(n, variable_count)
    ## X_input is the Random set of Variables with uniform distribution

    base = (10**(-4)) * (6 / np.sqrt(6 * np.pi))
```

```python
    value = np.zeros(n)

    for idx, x in enumerate(X_input):
        value += np.exp(-np.sqrt(3) * (x - 0.5))
        if varbose:
            print('Creating Value result matrix, Progress: ', int(idx /␣
↪len(X_input)),'%')

    value = value * base
    return value
```

[5]:
```python
##################### Distribution for n_dimentionality #####################
distribution_50_D = volume_function(n = 50)
distribution_100_D = volume_function(n = 100)
distribution_200_D = volume_function(n = 200)
distribution_1000_D = volume_function(n = 1000)
```

[6]:
```python
fig, axes = plt.subplots(2,2, figsize=(15,8))

axes[0][0].hist(distribution_50_D, color='c')
axes[0][0].legend(['50 Dimentions Distribution'])
axes[0][0].set_xlabel('Values')
axes[0][0].set_ylabel('Counts')

axes[0][1].hist(distribution_100_D, color='c')
axes[0][1].legend(['100 Dimentions Distribution'])
axes[0][1].set_xlabel('Values')
axes[0][1].set_ylabel('Counts')

axes[1][0].hist(distribution_200_D, color='c')
axes[1][0].legend(['200 Dimentions Distribution'])
axes[1][0].set_xlabel('Values')
axes[1][0].set_ylabel('Counts')

axes[1][1].hist(distribution_1000_D, color='c')
axes[1][1].legend(['1000 Dimentions Distribution'])
axes[1][1].set_xlabel('Values')
axes[1][1].set_ylabel('Counts')

plt.savefig('Q2_plot.png')
plt.show()
```



2 Question 3

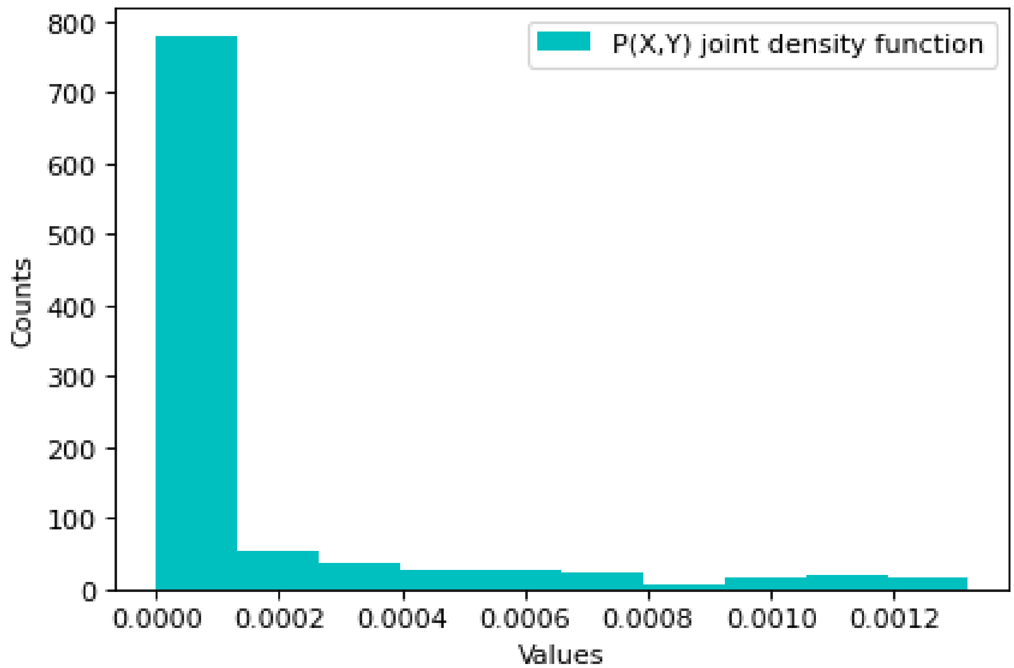
[7]:
```python
#################### Probability functions ####################

#### Note: This functions implemented here are came from calculating them by hand on paper

## X,Y joint distribution density function
def Probab_X_Y(a, b, mu, X,Y):
    var1 = 1 / (2* np.pi * a*b)
    var2 = ( np.power((Y-mu),2) / 2*(a^2))
    var3 = ( np.power((X - Y), 2) / 2*(b^2))
    value = var1* np.exp(-(var2+var3))

    return value

## Y density function
def Probab_Y(a, b, mu, X,Y):
    # above the division formula
    var1 = 2* a^2 * b^2
    var2 = ( np.power((Y-mu),2) / 2*(a^2))
    var3 = ( np.power((X - Y), 2) / 2*(b^2))

    # Calculate the above matrix
    above = var1 * np.exp(-(var2+var3))
```

```
    # below the division formula
    var5 = 4 * np.pi * a * b
    var6 = (mu * b^2) - ((b^2) + (a^2)) * Y + ((a^2) * X)

    # Calculate the below matrix
    below = var5 * var6

    # And at last calculate the function value
    value = np.divide(above, below)

    return value

## Calculate the probabily ofY given the information X
def Probab_Y_Given_X(b,Y,X):
    above = Y - X
    below = b^2

    value = above / below

    return value
```

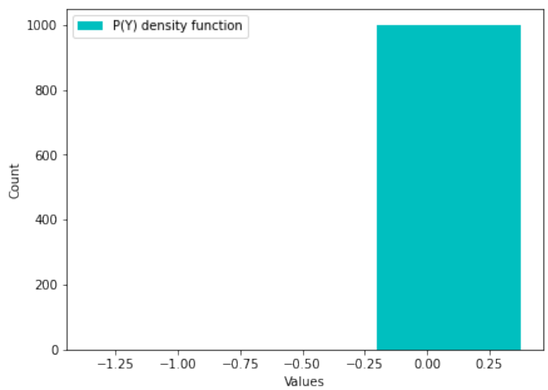
[8]:
```
np.random.seed(1)
X = np.random.normal(0,1, 1000)
Y = np.random.normal(0,1, 1000)
```

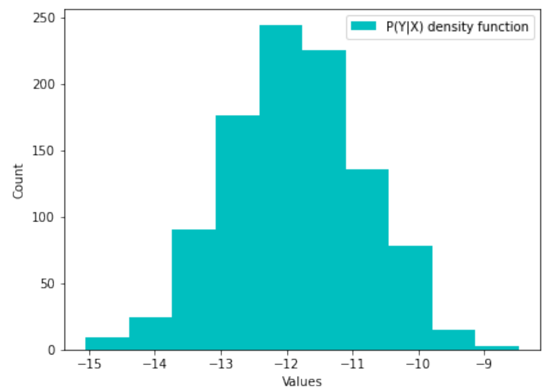
[9]:
```
p_x_y = Probab_X_Y(40, 3, 0, X, Y)
```

[10]:
```
plt.figure(figsize=(6, 4), dpi=80)
plt.hist(p_x_y ,color='c')
plt.legend(["P(X,Y) joint density function"])
plt.xlabel("Values")
plt.ylabel("Counts")
plt.show()
```



[11]:
```
##### Show two graphs side by side #####

fig, axes = plt.subplots(1,2,figsize=(15,5))


p_y = Probab_Y(40, 3, 0, X, Y)
axes[0].hist(p_y,bins=3,color='c')
axes[0].legend(["P(Y) density function"])
axes[0].set_xlabel("Values")
axes[0].set_ylabel("Count")



p_y_given_x = Probab_Y_Given_X(3, Y= Y,X = 11.9)
axes[1].hist(p_y_given_x, color='c')
axes[1].legend(["P(Y|X) density function"])
axes[1].set_xlabel("Values")
axes[1].set_ylabel("Count")
plt.savefig("Q3_plot.png")

plt.show()
```





3 Question 4

[11]:
```
################ Computing eignvalues and eignvectors to chack the results of␣
↪the calculation by hand ################
matrix = np.matrix('64 -24; -25 64')

eignvalue, eignvector = np.linalg.eig(matrix)

print('eign value: \n',eignvalue)
print('eign vector: \n',eignvector)
```

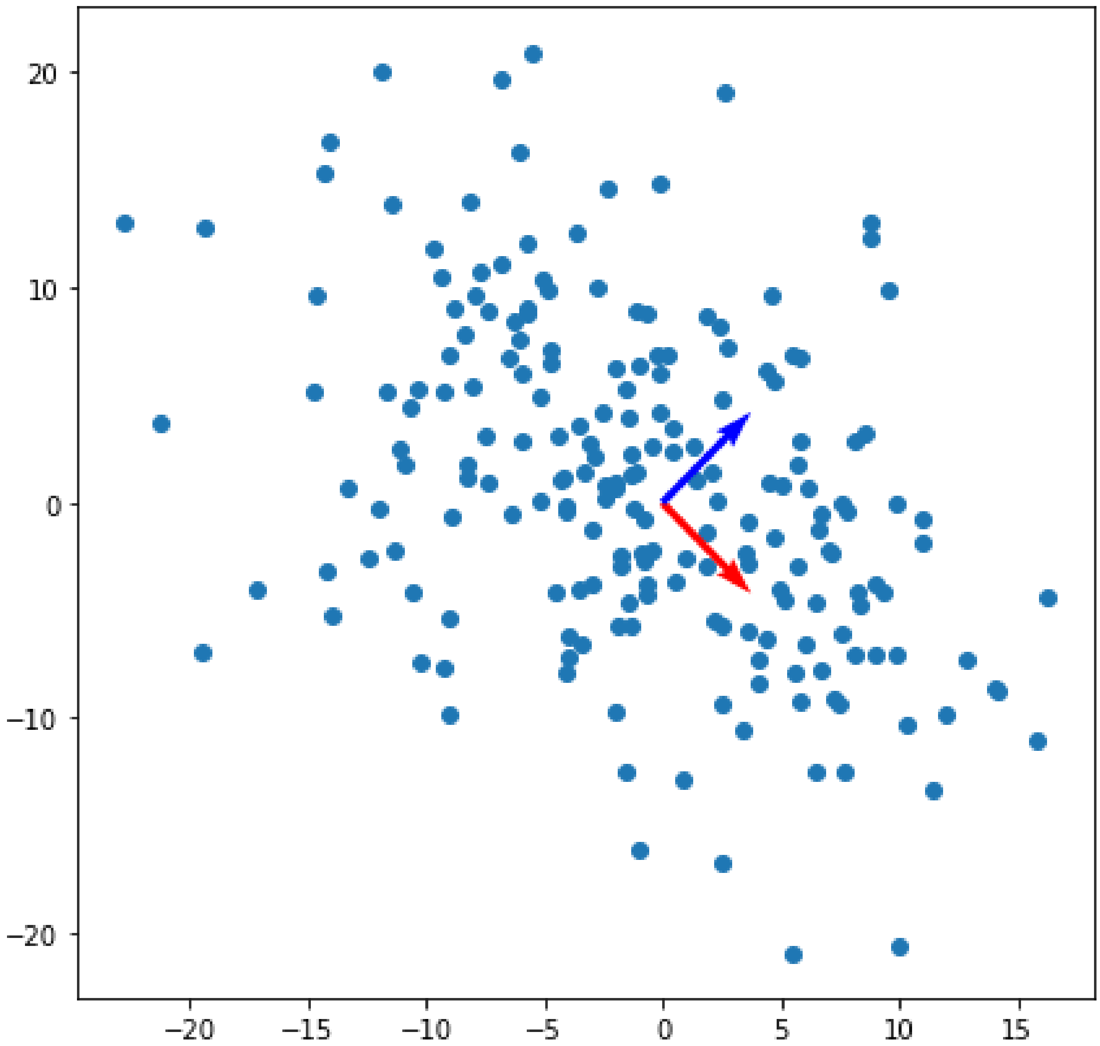
```
eign value:
 [88.49489743 39.50510257]
eign vector:
 [[ 0.69985421  0.69985421]
 [-0.71428571  0.71428571]]
```

[12]:
```
################ Question 4 Last part ################
np.random.seed(10)

mean = [0 ,0]
cov = np.matrix('64 -25;-25 64')
x = np.random.multivariate_normal(mean, cov, 200)
```

[13]:
```
################ Plot the data and their eignvector ################

fig = plt.figure(figsize=(7,7))
X = x[:,0]
Y = x[:,1]

plt.scatter(X, Y)
```

```python
## our plot origin
origin = [0, 0]

## get each eignvector
eig_vec1 = eignvector[:,0]
eig_vec2 = eignvector[:,1]

## Convert matrixes to array
eig_vec1 = np.asarray(eig_vec1).flatten()
eig_vec2 = np.asarray(eig_vec2).flatten()

print(eig_vec1)
print(eig_vec2)

plt.quiver(*origin ,*eig_vec1, color = 'r',scale=8)
plt.quiver(*origin ,*eig_vec2, color = 'b',scale=8)


plt.show()
```

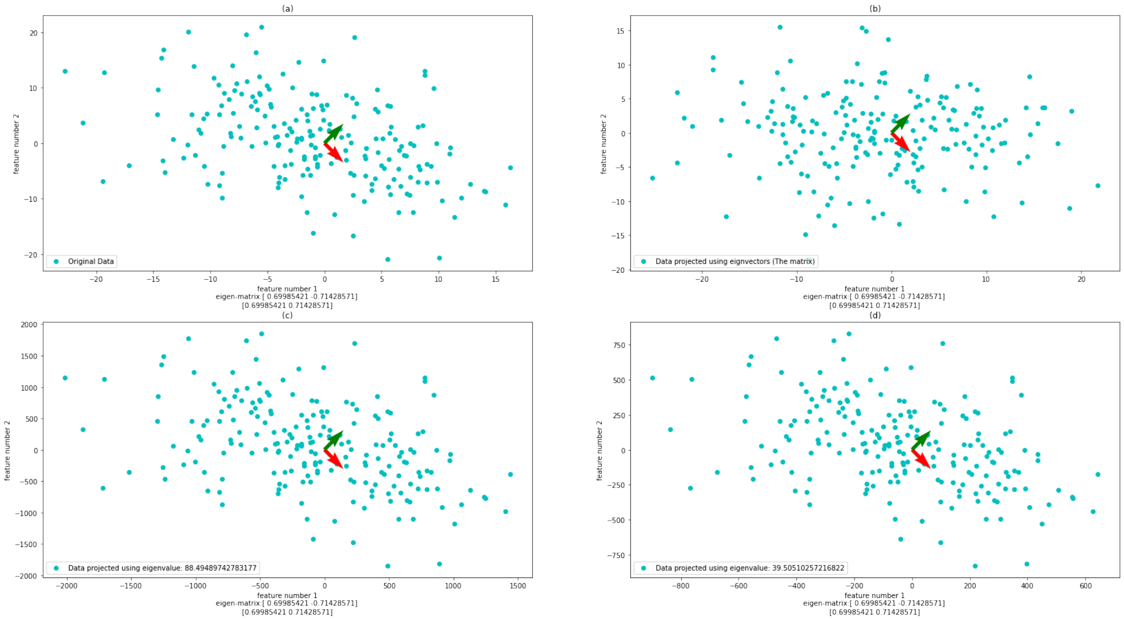
```
[ 0.69985421 -0.71428571]
[0.69985421 0.71428571]
```



[14]:
```
## To avoid duplicate code we made this function to plot data using scatter and␣
↪quiver plot
def plot_scatter(X,Y, ax,legend , title, description='', color='c', origin= [0,␣
↪0]):
    eig_vec1, eig_vec2 = calculate_eigen_vectors(X,Y)

    ax.scatter(X , Y, c=color)
    ax.legend([legend], loc='lower left')
    ax.set_xlabel(f'feature number 1\neigen-matrix:%s \n%s' %␣
↪(eig_vec1,eig_vec2), description)
    ax.set_ylabel('feature number 2')
    ax.set_title(title)

    ax.quiver(*origin ,*eig_vec1, color = 'r',scale=18)
```

```
    ax.quiver(*origin ,*eig_vec2, color = 'g',scale=18)

def calculate_eigen_vectors(X,Y):
    cov = np.cov(X,Y)
    _, eigen_vector = np.linalg.eig(cov)

    ## get each eignvector
    eig_vec1 = eigen_vector[:,0]
    eig_vec2 = eigen_vector[:,1]

    ## Convert matrixes to array
    eig_vec1 = np.asarray(eig_vec1).flatten()
    eig_vec2 = np.asarray(eig_vec2).flatten()

    return eig_vec1, eig_vec2
```

[19]:
```
############## Projecting Data using covariance vectors ################



## transform data
## Here we are multiplying each feature vector to covariance eignvectors (A 2␣
↪by 2 matrix as eignvectors)
x_transformed = x.dot(eignvector)

X_new = x_transformed[:,0]
Y_new = x_transformed[:,1]
## Refining X_new and Y_new as a simple vector
X_new = np.asarray(X_new).flatten()
Y_new = np.asarray(Y_new).flatten()

fig, axes = plt.subplots(2, 2, figsize=(28,15))

plot_scatter(X,Y, axes[0][0], "Original Data", "(a)")
plot_scatter(X_new, Y_new, axes[0][1], "Data projected using eignvectors (The␣
↪matrix)","(b)")
plot_scatter(X * eignvalue[0], Y * eignvalue[0], axes[1][0], f"Data projected␣
↪using eigenvalue: %s" % eignvalue[0], "(c)")
plot_scatter(X * eignvalue[1], Y * eignvalue[1], axes[1][1], f"Data projected␣
↪using eigenvalue: %s" % eignvalue[1], "(d)")

plt.show()
```



<!DOCTYPE html>

```
        <h2>    </h2>
        <br>
        <span>
                                                                                      ).

        </span>
        <h5 dir='ltr'> Au =  u </h5>
    </div>
    <br>
    <div dir='ltr'>
        <h2> English </h2>
        <br>
        <span>
             As we can see here in the projection of data with eigen-vectors or each eigen-valu
        </span>
        <h5> Au =  u </h5>
    </div>
</body>
```

4 Question 5

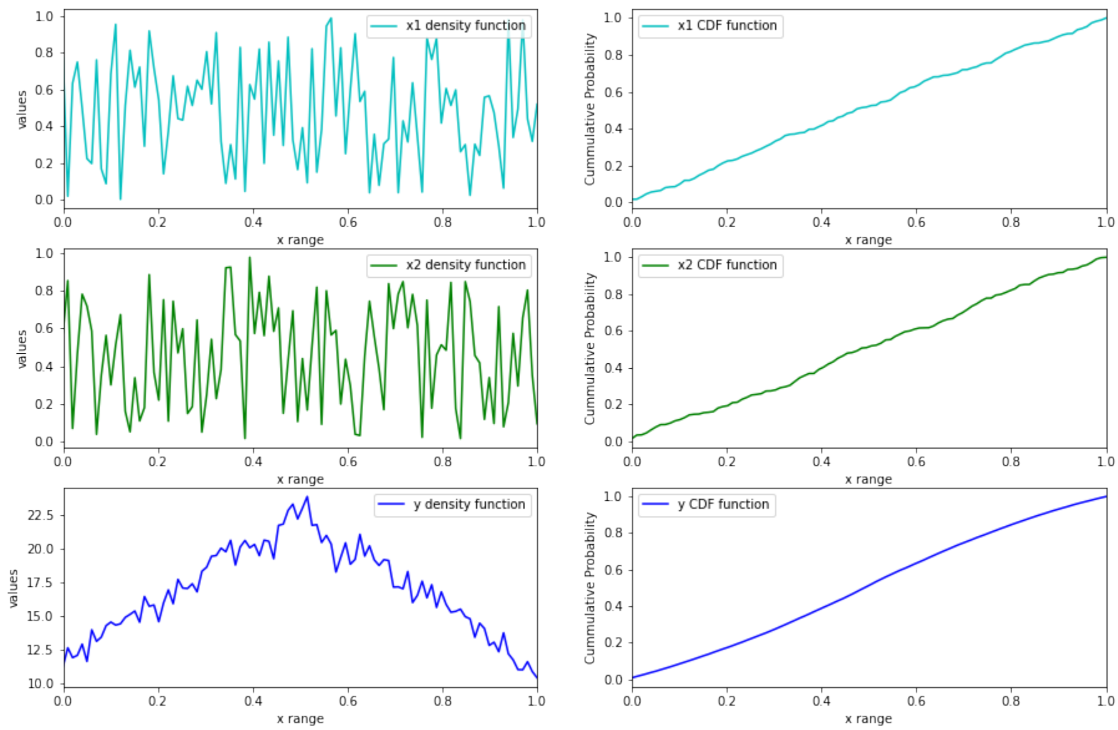
[15]:
```
#################### sum of two uniform distribution with i.i.d. property ####################

np.random.seed(10)
```

```
x1 = np.random.uniform(0,1,100)
x2 = np.random.uniform(0,1,100)

y = np.convolve(x1,x2,mode='same')
```

[16]:
```
##################### Generate Cummulative Distribution Probability␣
↪#####################
def generate_cdf(x1):
    ## Input is the pdf(probabilty density function) array

    ## This array will contain the cdf values
    cdf_arr = np.zeros(len(x1))

    total_value = 0
    for idx, x in enumerate(x1):
        cdf_arr[idx] = x / x1.sum()
        total_value += cdf_arr[idx]

        cdf_arr[idx] = total_value


    return cdf_arr

x1_cdf = generate_cdf(x1)
x2_cdf = generate_cdf(x2)
y_cdf = generate_cdf(y)
```

[17]:
```
def plot_my_graph(ax, x, y, legend, xlabel, ylabel, color = 'c', xlim = [0,1]):
    ax.plot(x,y,color=color)
    ## if xlim was allowed
    if(xlim != False):
        ax.set_xlim(xlim)

    ax.legend([legend])
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)


fig, axes = plt.subplots(3,2,figsize = (15,10))


plot_my_graph(axes[0][0], x=np.linspace(0,1, num=100),
             y= x1,
             legend="x1 density function",
             xlabel= "x range",
             ylabel= "values")
```

```
plot_my_graph(axes[0][1], x=np.linspace(0,1, num=100),
             y= x1_cdf,
             legend="x1 CDF function",
             xlabel= "x range",
             ylabel= "Cummulative Probability")



plot_my_graph(axes[1][0], x=np.linspace(0,1, num=100),
             y= x2, legend="x2 density function",
             xlabel= "x range",
             ylabel= "values",
             color='g')

plot_my_graph(axes[1][1], x=np.linspace(0,1, num=100),
             y= x2_cdf, legend="x2 CDF function",
             xlabel= "x range",
             ylabel= "Cummulative Probability",
             color='g')



plot_my_graph(axes[2][0], x=np.linspace(0,1, num=100),
             y= y, legend="y density function",
             xlabel= "x range",
             ylabel= "values",
             color='b')

plot_my_graph(axes[2][1], x=np.linspace(0,1, num=100),
             y= y_cdf, legend="y CDF function",
             xlabel= "x range",
             ylabel= "Cummulative Probability",
             color='b')

plt.show()
```

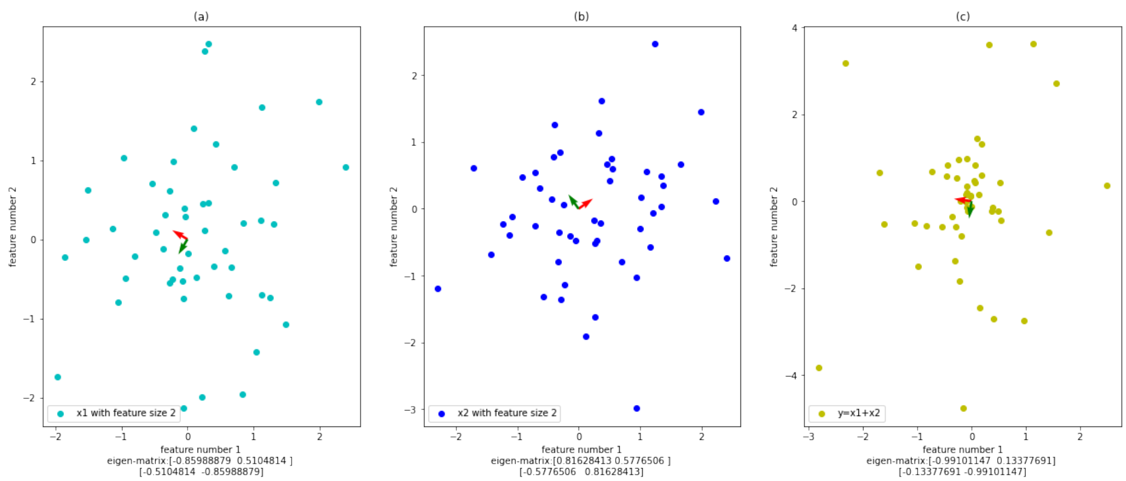


[18]:
```
#################### Question 5 part b ####################

## A normal distribution using feature size = 2
mean = [0 ,0]
cov = np.matrix('[1 0; 0 1]')

np.random.seed(10)

x1 = np.random.multivariate_normal(mean,cov,size=50)
x2 = np.random.multivariate_normal(mean,cov,size=50)
```

[20]:
```
## to calculate the convolution of x1 and x2 (multidiminsional variables)
def multidimensional_convolve(x1, x2, mode):
    assert len(x1) == len(x2)

    ## start with an empty array
    y = []
    for i in range(0, len(x1)):
        conv = np.convolve(x1[i], x2[i], mode= mode)
        y.append(conv)

    return y

y = multidimensional_convolve(x1,x2, mode= 'same')
```

```
y = np.array(y)
```

[21]:
```
fig, axes = plt.subplots(1, 3, figsize=(21,8))

plot_scatter(x1[:,0], x1[:,1], axes[0],
              legend='x1 with feature size 2',
              title='(a)')

plot_scatter(x2[:,0], x2[:,1], axes[1],
            legend='x2 with feature size 2',
            title='(b)',
            color='b')


plot_scatter(y[:,0], y[:,1], axes[2],
             legend='y=x1+x2',
             title='(c)',
             color='y')
```

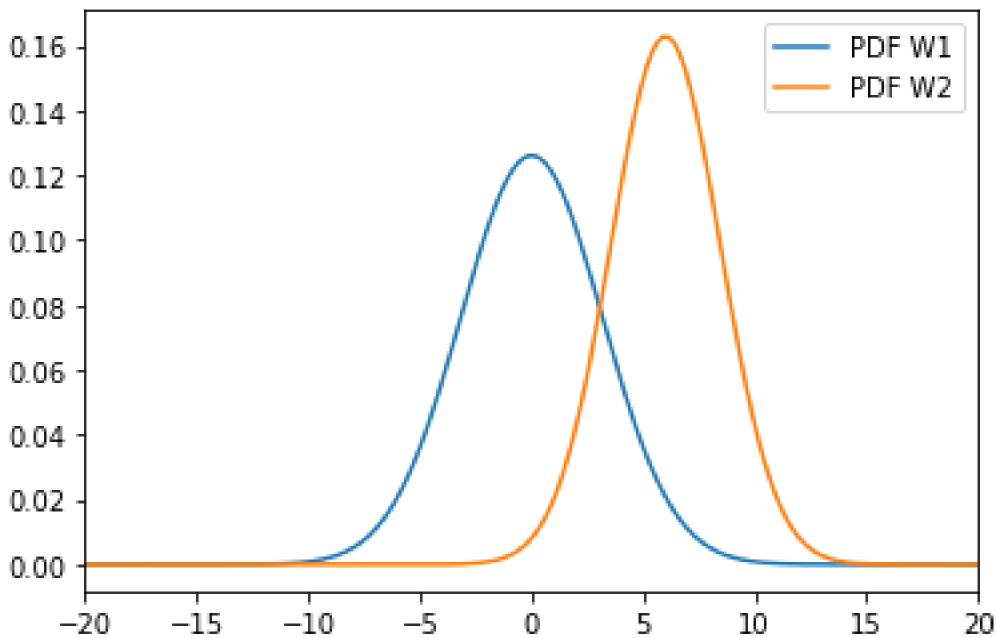


5 Question 8

[42]:
```
################### Question 8 ###################
from scipy.optimize import fsolve

x = np.linspace(-50,50, num=400)

## Calculating the density of conditional probabilty for class w1
k1 = 1 / np.sqrt(20 * np.pi)
density_probab_cond1 =  k1 * np.exp( -1 * x**2 / 20 )
```

```
## Calculating the density of conditional probabilty for class w2
k2 = 1 / np.sqrt(12 * np.pi)
density_probab_cond2 = k2 * np.exp(-1 * (x-6)**2 / 12)
```

[47]:
```
plt.figure(figsize=(6,4))
plt.plot(x, density_probab_cond1)
plt.plot(x, density_probab_cond2)
plt.legend(['PDF W1', 'PDF W2'])
plt.xlim([-20, 20])
plt.show()
```



[63]:
```
## We have calculated the decision boundaries
## points are the dicision boindary that are calculating using RISKs of w1 and␣
↪w2

point = 15 + 3*np.sqrt(15)
point2 = 15 - 3*np.sqrt(15)

## Plot the PDFs again
plt.figure(figsize=(6,4))
plt.plot(x, density_probab_cond1)
plt.plot(x, density_probab_cond2)
plt.legend(['PDF W1', 'PDF W2'])

## Plot the decision boundary
```

```
plt.plot(np.linspace(point,point, 20), np.linspace(-2,2, 20), 'k')
plt.plot(np.linspace(point2,point2, 20), np.linspace(-2,2, 20), 'k')
plt.ylim([-0.01, 0.2])
plt.xlim([-15, 35])

## Add text to plot to show the regions
plt.text(-14, 0.1, 'Region 1', style='italic',
        bbox={'facecolor':'yellow', 'alpha': 0.5})

plt.text(16, 0.1, 'Region 2' ,style= 'italic',
        bbox={'facecolor':'green', 'alpha': 0.5})

plt.text(28, 0.1, 'Region 1', style= 'italic',
        bbox={'facecolor':'yellow', 'alpha': 0.5})



plt.show()
```