2. (30 puntos) Funciones multivariable

1. El vector Gradiente: Para cada una de las siguientes funciones multivariable: grafique su superficie, calcule el vector gradiente manualmente, evaluelo y grafique el vector unitario o de largo conveniente para su visualización, en la dirección del gradiente para los dos puntos especificados (en la misma figura de la superficie). Finalmente calcule la magnitud de tal vector gradiente en cada punto.

   a) (5 puntos) $f\left(x,y\right)=\sqrt{x^2+y^2}$, evaluación del gradiente en los puntos $P_0=(5.2,6.4)$ y $P_1=(5.2,2.3)$.

   El vector gradiente corresponde a:

   $$\frac{\delta f}{\delta x}\sqrt{x^2+y^2}\rightarrow\left(x^2+y^2\right)^{\frac{1}{2}}=\frac{(x^2+y^2)^{\frac{1}{2}-1}}{2}\cdot 2x=\frac{x}{(x^2+y^2)^{\frac{1}{2}}}=\frac{\mathbf{x}}{\sqrt{\mathbf{x^2+y^2}}}$$

   $$\frac{\delta f}{\delta y}\sqrt{x^2+y^2}\rightarrow\left(x^2+y^2\right)^{\frac{1}{2}}=\frac{(x^2+y^2)^{\frac{1}{2}-1}}{2}\cdot 2y=\frac{y}{(x^2+y^2)^{\frac{1}{2}}}=\frac{\mathbf{y}}{\sqrt{\mathbf{x^2+y^2}}}$$

   $$\mathbf{\nabla f(x,y)=\frac{x}{\sqrt{x^2+y^2}}\,\hat{i}+\frac{y}{\sqrt{x^2+y^2}}\,\hat{j}}$$

Gráfica de la superficie de la función:

In [1]:
```python
import torch
import math
import numpy as np
import random
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib import style
from scipy.stats import norm
# Requerido para algoritmo de maximización de la esperanza, de lo contrario se mezclan
%matplotlib inline
style.use('default')
```

In [35]:
```python
#Función multivariable original
def funcion_z(X,Y):
    return torch.sqrt(X**2 + Y**2)

#Calculo del Vector Gradiente (Derivadas parciales)
def dx(X,Y):
    return X / torch.sqrt(X**2 + Y**2)

def dy(X,Y):
    return Y / torch.sqrt(X**2 + Y**2)

def vector_gradiente(X,Y):
    return [dx(X,Y),dy(X,Y),0]
```

```
P0=torch.tensor([5.2,6.4], dtype = torch.float)
P1=torch.tensor([5.2,2.3], dtype = torch.float)

vector_p0=torch.tensor([vector_gradiente(P0[0],P0[1])], dtype = torch.float)
vector_p1=torch.tensor([vector_gradiente(P1[0],P1[1])], dtype = torch.float)

print("El valor del vector gradiente para el punto P0=(5.2, 6.4) es: ", vector_p0)
print("El valor del vector gradiente para el punto P1=(5.2, 2.3) es: ", vector_p1)
```

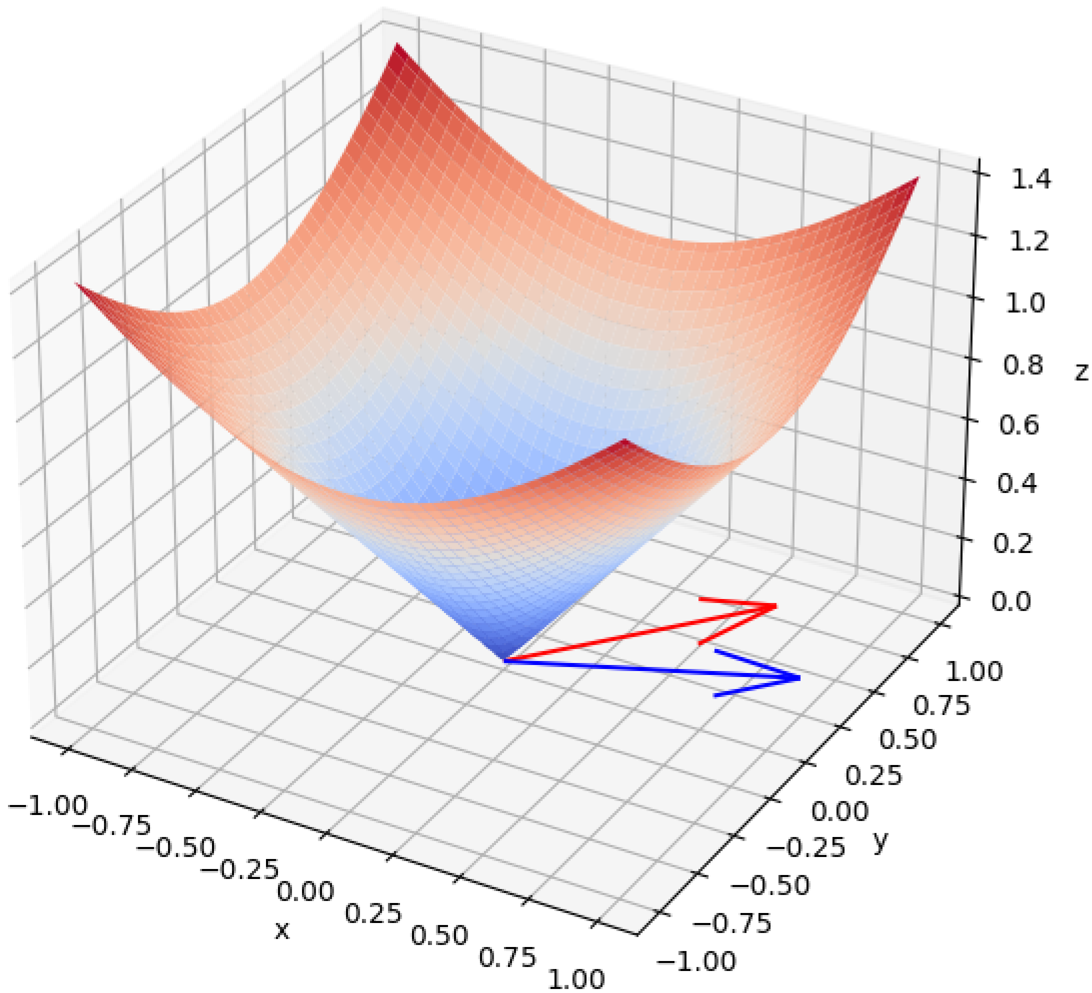
```
El valor del vector gradiente para el punto P0=(5.2, 6.4) es:  tensor([[0.6306, 0.776
1, 0.0000]])
El valor del vector gradiente para el punto P1=(5.2, 2.3) es:  tensor([[0.9145, 0.404
5, 0.0000]])
```

In [36]:

```
#Inicia el plot
N=256
arrange=[-1,1]

x_values=torch.linspace(arrange[0],arrange[1],steps=N)
y_values=torch.linspace(arrange[0],arrange[1],steps=N)

X, Y = torch.meshgrid(x_values,y_values)
Z = funcion_z(X,Y)

fig=plt.figure(figsize=(7,7))

ax = plt.axes(projection='3d')

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')

origin=[0,0,0]
ax.quiver(origin[0], origin[1], origin[2], vector_p0[:,0],vector_p0[:,1],vector_p0[:,2
ax.quiver(origin[0], origin[1], origin[2], vector_p1[:,0],vector_p1[:,1],vector_p1[:,2

ax.plot_surface(X,Y,Z,cmap=cm.coolwarm,linewidth=10,antialiased=True)
```

Out[36]:

```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x29f9908f8e0>
```



In [4]:
```
#Magnitud de los vectores:
print("La magnitud del vector P0 es: ", torch.linalg.vector_norm(vector_p0).item())
print("La magnitud del vector P1 es: ", torch.linalg.vector_norm(vector_p1).item())
```

```
La magnitud del vector P0 es:  1.0
La magnitud del vector P1 es:  1.0
```

b) (5 puntos) $z = f(x, y) = 3x^2 + 2y^4$, evaluación del gradiente en los puntos $P_0 = (0, 0)$ y $P_1 = (7.4, -6.3)$.

El vector gradiente corresponde a:

$$\frac{\delta f}{\delta x} 3x^2 + 2y^4 \rightarrow 3x^2 = \mathbf{6x}$$

$$\frac{\delta f}{\delta y} 3x^2 + 2y^4 \rightarrow 2y^4 = \mathbf{8y^3}$$

$$\mathbf{\nabla f(x, y) = 6x\hat{i} + 8y^3\hat{j}}$$

In [5]:
```
#Función multivariable original
def funcion_z(X, Y):
    return (3*X**2) + (2*Y**4)
```

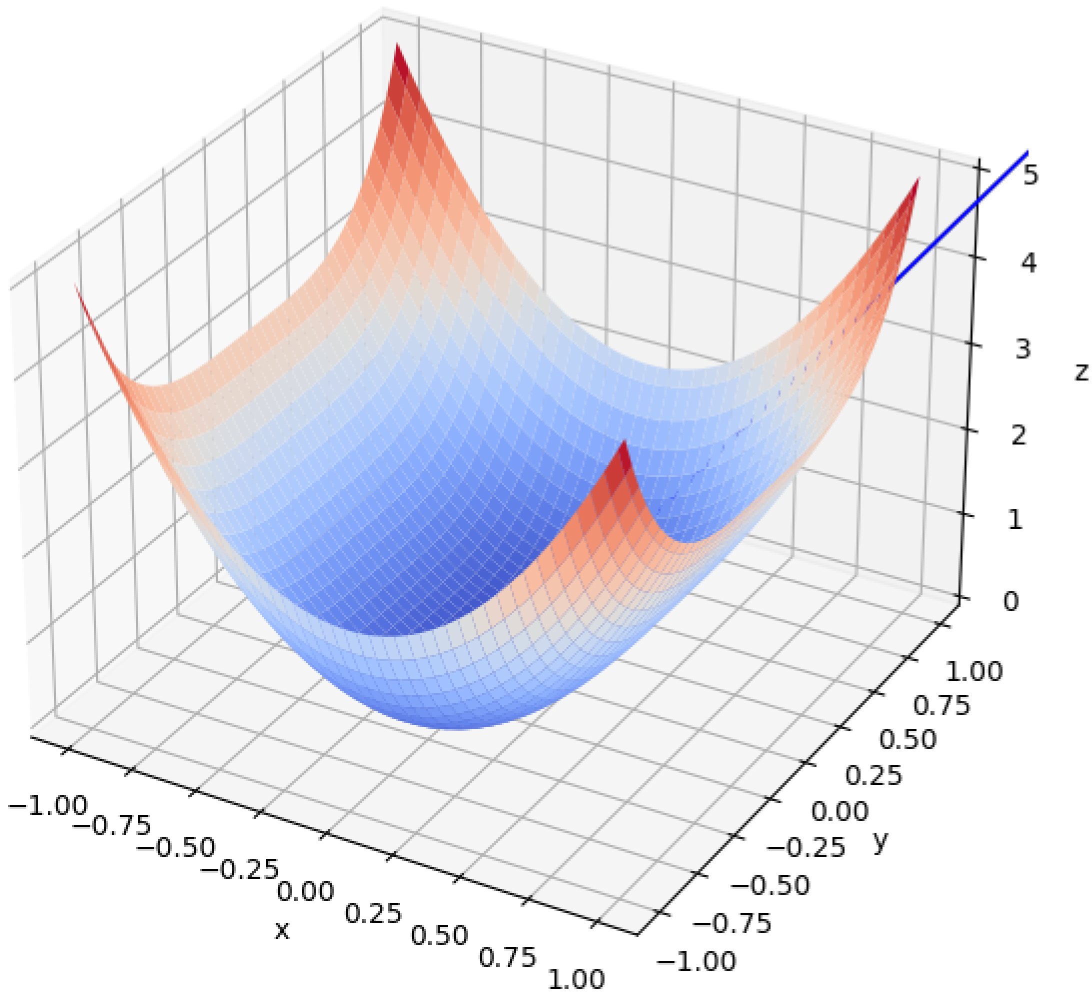
```python
#Calculo del Vector Gradiente (Derivadas parciales)
def dx(X,Y):
    return 6*X

def dy(X,Y):
    return 8*Y**3

def vector_gradiente(X,Y):
    return [dx(X,Y),dy(X,Y),0]


P0=torch.tensor([0,0], dtype = torch.float)
P1=torch.tensor([7.4,-6.3], dtype = torch.float)

vector_p0=torch.tensor([vector_gradiente(P0[0],P0[1])], dtype = torch.float)
vector_p1=torch.tensor([vector_gradiente(P1[0],P1[1])], dtype = torch.float)

print("El valor del vector gradiente para el punto P0=(0, 0) es: ", vector_p0)
print("El valor del vector gradiente para el punto P1=(7.4, -6.3) es: ", vector_p1)
```

```
El valor del vector gradiente para el punto P0=(0, 0) es:  tensor([[0., 0., 0.]])
El valor del vector gradiente para el punto P1=(7.4, -6.3) es:  tensor([[   44.4000,
-2000.3762,     0.0000]])
```

In [6]:

```python
#Inicia el plot
N=256
arrange=[-1,1]

x_values=torch.linspace(arrange[0],arrange[1],steps=N)
y_values=torch.linspace(arrange[0],arrange[1],steps=N)

X, Y = torch.meshgrid(x_values,y_values)
Z = funcion_z(X,Y)

fig=plt.figure(figsize=(7,7))

ax = plt.axes(projection='3d')

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')

origin=[0,0,0]
ax.quiver(origin[0], origin[1], origin[2], vector_p0[:,0],vector_p0[:,1],vector_p0[:,2
ax.quiver(origin[0], origin[1], origin[2], vector_p1[:,0],vector_p1[:,1],vector_p1[:,2

ax.plot_surface(X,Y,Z,cmap=cm.coolwarm,linewidth=10,antialiased=True)
```

Out[6]: `<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x29f97522e80>`



In [7]:
```
#Magnitud de los vectores:
print("La magnitud del vector P0 es: ", torch.linalg.vector_norm(vector_p0))
print("La magnitud del vector P1 es: ", torch.linalg.vector_norm(vector_p1))
```

```
La magnitud del vector P0 es:  tensor(0.)
La magnitud del vector P1 es:  tensor(2000.8689)
```

c) (5 puntos) $z = f\,(x, y) = 4x^2 + 2x + e^{2x} + 5y^2 + e^{3y} + 1$, evaluación del gradiente en los puntos $P_0 = (2, 1)$ y $P_1 = (5, 7)$.

El vector gradiente corresponde a:

$$\frac{\delta f}{\delta x} 4x^2 + 2x + e^{2x} + 5y^2 + e^{3y} + 1 \rightarrow 8x + 2 + 2e^{2x} + 0 + 0 + 0 = 8x + 2 + 2e^{2x} = \mathbf{2(4x +}$$

$$\frac{\delta f}{\delta y} 4x^2 + 2x + e^{2x} + 5y^2 + e^{3y} + 1 \rightarrow 0 + 0 + 0 + 10y + 3e^{3y} + 0 = \mathbf{10y + 3e^{3y}}$$

$$\nabla f(x, y) = 2(4x + e^{2x} + 1)\hat{i} + 10y + 3e^{3y}\hat{j}$$

Gráfica de la superficie de la función:

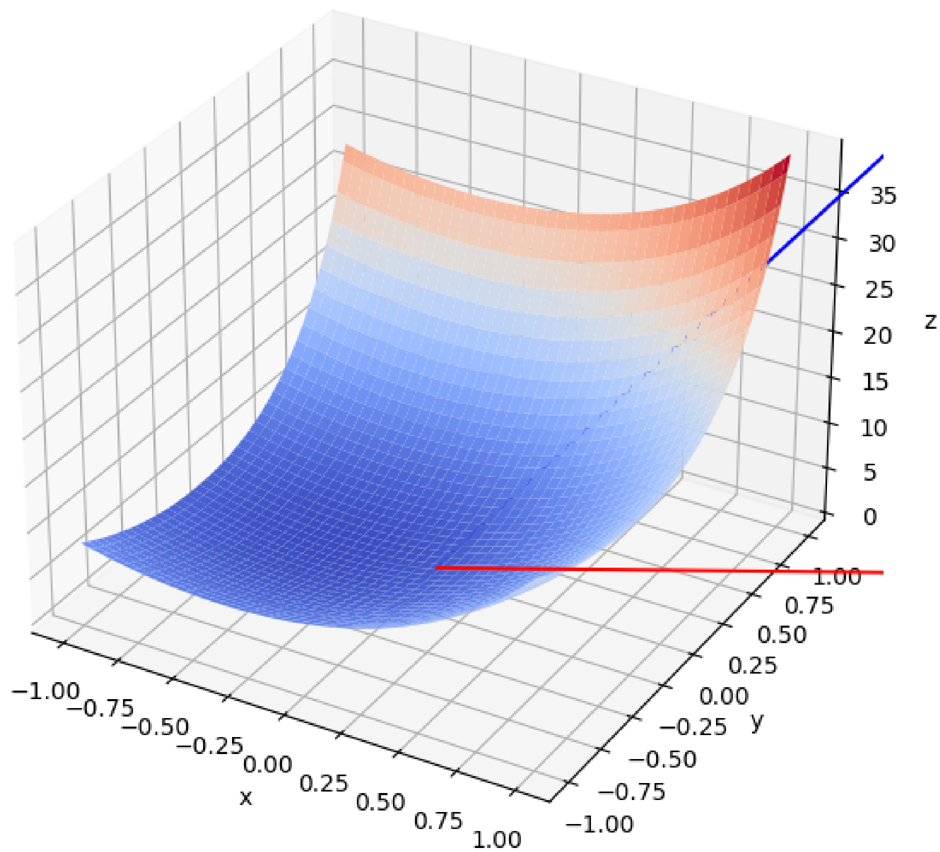
In [15]:
```python
#Función multivariable original
def funcion_z(X, Y):
    return (4*X**2) + (2*X) + (torch.e**(2*X)) + (5*Y**2) + (torch.e**(3*Y)) + 1

#Calculo del Vector Gradiente (Derivadas parciales)
def dx(X,Y):
    return 2*((4*X)+torch.e**(2*X)+1)

def dy(X,Y):
    return (10*Y)+(3*torch.e**(3*Y))

def vector_gradiente(X,Y):
    return [dx(X,Y),dy(X,Y),0]


P0=torch.tensor([2,1], dtype = torch.float)
P1=torch.tensor([5,7], dtype = torch.float)

vector_p0=torch.tensor([vector_gradiente(P0[0],P0[1])], dtype = torch.float)
vector_p1=torch.tensor([vector_gradiente(P1[0],P1[1])], dtype = torch.float)

print("El valor del vector gradiente para el punto P0=(2, 1) es: ", vector_p0)
print("El valor del vector gradiente para el punto P1=(5, 7) es: ", vector_p1)
```

```
El valor del vector gradiente para el punto P0=(2, 1) es:  tensor([[127.1963,  70.256
6,   0.0000]])
El valor del vector gradiente para el punto P1=(5, 7) es:  tensor([[4.4095e+04, 3.956
4e+09, 0.0000e+00]])
```

In [17]:
```python
#Inicia el plot
N=256
arrange=[-1,1]

x_values=torch.linspace(arrange[0],arrange[1],steps=N)
y_values=torch.linspace(arrange[0],arrange[1],steps=N)

X, Y = torch.meshgrid(x_values,y_values)
Z = funcion_z(X,Y)

fig=plt.figure(figsize=(7,7))

ax = plt.axes(projection='3d')

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')

origin=[0,0,0]
ax.quiver(origin[0], origin[1], origin[2], vector_p0[:,0],vector_p0[:,1],vector_p0[:,2
ax.quiver(origin[0], origin[1], origin[2], vector_p1[:,0],vector_p1[:,1],vector_p1[:,2

ax.plot_surface(X,Y,Z,cmap=cm.coolwarm,linewidth=10,antialiased=True)
```

Out[17]: `<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x20e39c20970>`

Figure



In [18]:
```
#Magnitud de los vectores:
print("La magnitud del vector P0 es: ", torch.linalg.vector_norm(vector_p0))
print("La magnitud del vector P1 es: ", torch.linalg.vector_norm(vector_p1))
```

```
La magnitud del vector P0 es:  tensor(145.3096)
La magnitud del vector P1 es:  tensor(3.9564e+09)
```

d) (5 puntos) $z = f\left(x,y\right) = \sin\left(x^2\right) + x\cos\left(y^3\right)$, evaluación del gradiente en los puntos $P_0 = (-2,6)$ y $P_1 = (0,4)$.

El vector gradiente corresponde a:

$$\frac{\delta f}{\delta x}\sin\left(x^2\right) + x\cos\left(y^3\right) \to cos(x^2)\cdot 2x + cos(y^3) = \mathbf{2x\cdot\cos(x^2)+\cos(y^3)}$$

$$\frac{\delta f}{\delta y}\sin\left(x^2\right) + x\cos\left(y^3\right) \to -x\cdot sen(y^3)\cdot 3y^2 = \mathbf{-3xy^2\cdot sen(y^3)}$$

$$\mathbf{\nabla f(x,y) = 2x\cdot\cos(x^2)+\cos(y^3)\,\hat{i} + -3xy^2\cdot sen(y^3)\,\hat{j}}$$

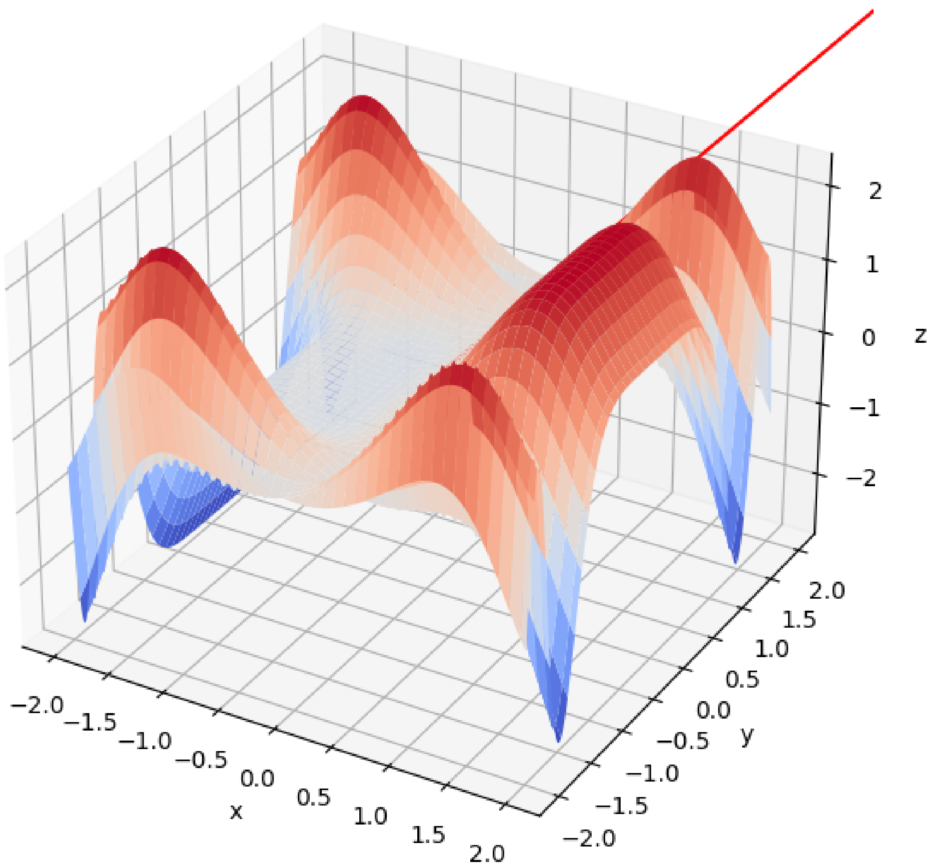
In [19]:
```python
#Función multivariable original
def funcion_z(X, Y):
    return torch.sin(X**2) + (X*torch.cos(Y**3))

#Calculo del Vector Gradiente (Derivadas parciales)
def dx(X,Y):
    return 2*X*torch.cos(X**2)+torch.cos(Y**3)

def dy(X,Y):
    return -3*X*(Y**2)*torch.sin(Y**3)

def vector_gradiente(X,Y):
    return [dx(X,Y),dy(X,Y),0]

P0=torch.tensor([-2,6], dtype = torch.float)
P1=torch.tensor([0,4], dtype = torch.float)

vector_p0=torch.tensor([vector_gradiente(P0[0],P0[1])], dtype = torch.float)
vector_p1=torch.tensor([vector_gradiente(P1[0],P1[1])], dtype = torch.float)

print("El valor del vector gradiente para el punto P0=(-2, 6) es: ", vector_p0)
print("El valor del vector gradiente para el punto P1=(0, 4) es: ", vector_p1)
```

```
El valor del vector gradiente para el punto P0=(-2, 6) es:  tensor([[  1.8966, 150.34
86,   0.0000]])
El valor del vector gradiente para el punto P1=(0, 4) es:  tensor([[0.3919, -0.0000,
0.0000]])
```

In [20]:
```python
#Inicia el plot
N=256
arrange=[-2,2]

x_values=torch.linspace(arrange[0],arrange[1],steps=N)
y_values=torch.linspace(arrange[0],arrange[1],steps=N)

X, Y = torch.meshgrid(x_values,y_values)
Z = funcion_z(X,Y)

fig=plt.figure(figsize=(7,7))

ax = plt.axes(projection='3d')

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')

origin=[0,0,0]
ax.quiver(origin[0], origin[1], origin[2], vector_p0[:,0],vector_p0[:,1],vector_p0[:,2
ax.quiver(origin[0], origin[1], origin[2], vector_p1[:,0],vector_p1[:,1],vector_p1[:,2

ax.plot_surface(X,Y,Z,cmap=cm.coolwarm,linewidth=10,antialiased=True)
```

Out[20]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x20e33ea6fa0>
```

Figure



In [21]:
```
#Magnitud de los vectores:
print("La magnitud del vector P0 es: ", torch.linalg.vector_norm(vector_p0))
print("La magnitud del vector P1 es: ", torch.linalg.vector_norm(vector_p1))
```

```
La magnitud del vector P0 es:  tensor(150.3606)
La magnitud del vector P1 es:  tensor(0.3919)
```

2. (10 puntos) En general, investigue ¿qué es y que indica la matriz Hessiana? Sobre las aplicaciones de esta matriz, se pueden citar los siguientes puntos:

   - Se define como una matriz cuadrada de nxn que se compone de las segundas derivada parciales de la función multivariable, por ejemplo para funciones de dos variables:

$$\mathbf{H}_f(x,y) = \begin{pmatrix} \frac{\delta^2 f}{\delta x^2} & \frac{\delta^2 f}{\delta y \delta x} \\ \frac{\delta^2 f}{\delta y \delta x} & \frac{\delta^2 f}{\delta y^2} \end{pmatrix}$$

- Por Teorema de Schwarz se puede decir que $\frac{\delta^2 f}{\delta y \delta x} = \frac{\delta^2 f}{\delta x \delta y}$

- Permite encontrar ya sea máximos o mínimos de funciones multivariable. Para conseguir esto, el procedimiento a seguir consiste de obtener los puntos críticos de la función (igualando a cero el vector gradiente y obteniendo los puntos respectivos del despeje) y operarlos con la matriz Hessiana.

- Una vez obtenido el resultado, la matriz que se consiguió debe ser evaluada bajo el criterio de: definida positiva, definida negativa, indefinida, etc (lo cual puede ser establecido con el Criterio de los valores propios o con el Criterio de Sylvester).

- Dependiendo del tipo de matriz obtenida, así podrá definirse el punto que fue utilizado para el cálculo (máximo, mínimo o un punto neutro).

- También permite saber si una función es cóncava o convexa con respecto a un conjunto de puntos pertenecientes a la función, aplicando nuevamente el concepto de definida positiva, definida negativa, indefinida, etc.

- Fuentes utilizadas para cálculo y usos:
  - Matriz Hessiana (o Hessiano)
  - La matriz hessiana

a) Para cada una de los puntos 1.a, 1.b, 1.c, y 1.d calcule la matriz Hessiana:

- 1.a Para $f(x, y) = \sqrt{x^2 + y^2}$
  - $\frac{\delta f}{\delta x} = \frac{x}{\sqrt{x^2+y^2}}$
  - $\frac{\delta f}{\delta y} = \frac{y}{\sqrt{x^2+y^2}}$
  - $\frac{\delta^2 f}{\delta x^2} = \frac{y^2}{\sqrt{(x^2+y^2)^3}}$
  - $\frac{\delta^2 f}{\delta y^2} = \frac{x^2}{\sqrt{(x^2+y^2)^3}}$
  - $\frac{\delta^2 f}{\delta y \delta x} = \frac{\delta^2 f}{\delta x \delta y} = \frac{2xy^2-x^3}{(x^2+y^2)\sqrt{(x^2+y^2)^3}}$
  - $\mathbf{H}_f(x, y) = \begin{pmatrix} \frac{\delta^2 f}{\delta x^2} & \frac{\delta^2 f}{\delta x \delta y} \\ \frac{\delta^2 f}{\delta y \delta x} & \frac{\delta^2 f}{\delta y^2} \end{pmatrix} = \begin{pmatrix} \frac{y^2}{\sqrt{(x^2+y^2)^3}} & \frac{2xy^2-x^3}{(x^2y^2)\sqrt{(x^2+y^2)^3}} \\ \frac{2xy^2-x^3}{(x^2+y^2)\sqrt{(x^2+y^2)^3}} & \frac{x^2}{\sqrt{(x^2+y^2)^3}} \end{pmatrix}$

- 1.b Para $z = f(x, y) = 3x^2 + 2y^4$
  - $\frac{\delta f}{\delta x} = 6x$
  - $\frac{\delta f}{\delta y} = 8y^3$

- $\frac{\delta^2 f}{\delta x^2} = 6$
- $\frac{\delta^2 f}{\delta y^2} = 24y^2$
- $\frac{\delta^2 f}{\delta y \delta x} = \frac{\delta^2 f}{\delta x \delta y} = 0$
- $\mathbf{H}_f(x,y) = \begin{pmatrix} \frac{\delta^2 f}{\delta x^2} & \frac{\delta^2 f}{\delta x \delta y} \\ \frac{\delta^2 f}{\delta y \delta x} & \frac{\delta^2 f}{\delta y^2} \end{pmatrix} = \begin{pmatrix} 6 & 0 \\ 0 & 24y^2 \end{pmatrix}$

- 1.c Para $z = f(x,y) = 4x^2 + 2x + e^{2x} + 5y^2 + e^{3y} + 1$
  - $\frac{\delta f}{\delta x} = 2(4x + e^{2x} + 1)$
  - $\frac{\delta f}{\delta y} = 10y + 3e^{3y}$
  - $\frac{\delta^2 f}{\delta x^2} = 4(e^{2x} + 2)$
  - $\frac{\delta^2 f}{\delta y^2} = 9e^{3y} + 10$
  - $\frac{\delta^2 f}{\delta y \delta x} = \frac{\delta^2 f}{\delta x \delta y} = 0$
  - $\mathbf{H}_f(x,y) = \begin{pmatrix} \frac{\delta^2 f}{\delta x^2} & \frac{\delta^2 f}{\delta x \delta y} \\ \frac{\delta^2 f}{\delta y \delta x} & \frac{\delta^2 f}{\delta y^2} \end{pmatrix} = \begin{pmatrix} 4(e^{2x} + 2) & 0 \\ 0 & 9e^{3y} + 10 \end{pmatrix}$

- 1.b Para $z = f(x,y) = \sin(x^2) + x\cos(y^3)$
  - $\frac{\delta f}{\delta x} = 2x \cdot cos(x^2) + cos(y^3)$
  - $\frac{\delta f}{\delta y} = -3xy^2 \cdot sen(y^3)$
  - $\frac{\delta^2 f}{\delta x^2} = 2cos(x^2) - 2x^2 sen(x^2)$
  - $\frac{\delta^2 f}{\delta y^2} = -3xy(2sen(y^3) + 3y^3 cos(y^3))$
  - $\frac{\delta^2 f}{\delta y \delta x} = \frac{\delta^2 f}{\delta x \delta y} = -3y^2 sen(y^3)$
  - $\mathbf{H}_f(x,y) = \begin{pmatrix} \frac{\delta^2 f}{\delta x^2} & \frac{\delta^2 f}{\delta x \delta y} \\ \frac{\delta^2 f}{\delta y \delta x} & \frac{\delta^2 f}{\delta y^2} \end{pmatrix} = \begin{pmatrix} 2cos(x^2) - 2x^2 sen(x^2) & -3y^2 sen(y^3) \\ -3y^2 sen(y^3) & -3xy(2sen(y^3) + 3y \end{pmatrix}$

3. (30 puntos) Probabilidades: Algoritmo de Maximización de la Esperanza

A continuación, implemente el algoritmo de maximización de la esperanza (descrito en el material del curso), usando la definición y descripción de las siguientes funciones como base:

In [39]:
```
# Constantes
# Rangos para inicializar mu
MU_START = 10
MU_END = 50
# Rangos para inicializar sigma
SIGMA_START = 3.1
SIGMA_END = 6.2
# Dato para solucion heuristica
HEURISTIC_STEP = 5
```

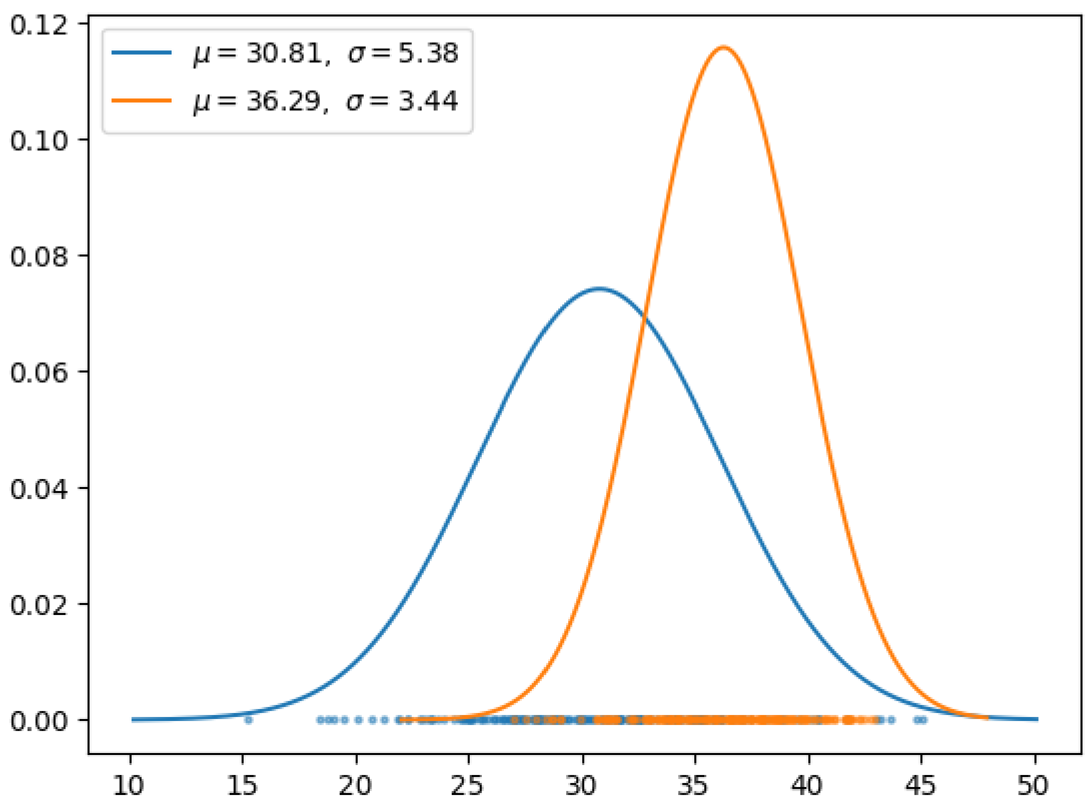
In [40]:
```
#  Graficar observacion con su funcion de densidad de probabilidad
def plot_observation(observation, show=False):
    mu = torch.mean(observation)
    sigma = torch.std(observation, unbiased=True)
    x_axis = torch.arange(min(observation) - 5, max(observation) + 5, 0.01)
    plt.scatter(observation.numpy(), torch.zeros(len(observation)), s=5, alpha=0.5)
    plt.plot(x_axis.numpy(), norm.pdf(x_axis.numpy(), mu.numpy(), sigma.numpy()),
             label=r'$\mu=' + str(round(mu.item(), 2)) + r',\ \sigma=' + str(round(sig
    if show:
        plt.legend()
        plt.show()

#  Graficar las distribuciones aleatorias junto con las observaciones
def plot_gaussian_distribution_and_observations(distribution_parameters, observations,
    for observation in observations:
        plot_observation(observation)
    param_number = 1
    for parameters in distribution_parameters:
        mu = parameters[0]
        sigma = parameters[1]
        x_axis = torch.arange(mu / 2, mu * 2, 0.01)
        plt.plot(x_axis.numpy(), norm.pdf(x_axis.numpy(), mu.numpy(), sigma.numpy()),
                 label=r'$\mu_' + str(param_number) + r'=' + str(round(mu.item(), 2))
                       r',\ \sigma_' + str(param_number) + '=' + str(round(sigma.item(
        param_number += 1
    if show:
        plt.legend()
        plt.show()
```

1. Función generate_data:

In [41]:
```
#  Genera datos siguiendo una distribucion gaussiana
def generate_data(n_observations: int, k_parameters=2, show=False, heuristic=False):
    gaussian_distributions = []
    heuristic_mu = random.uniform(MU_START, MU_END) if heuristic else 0
    for k in range(k_parameters):
        mu = torch.tensor(random.uniform(MU_START, MU_END)) if not heuristic else torc
```

```
        heuristic_mu += HEURISTIC_STEP if heuristic else 0
        sigma = torch.tensor(random.uniform(SIGMA_START, SIGMA_END))
        normal_dist = torch.distributions.Normal(mu, sigma)
        sample = normal_dist.sample((n_observations, 1)).squeeze()
        gaussian_distributions.append(sample)
    for distribution in gaussian_distributions:
        plot_observation(distribution)
    if show:
        plt.legend()
        plt.show()
    return gaussian_distributions
```

In [42]:
```
plt.figure()
dataset = generate_data(200, show=False, heuristic=True)
plt.legend()
plt.show()
print(dataset)
```



```
[tensor([30.9652, 23.9691, 28.7981, 43.2340, 32.3662, 32.5551, 29.6710, 31.5756,
        31.6417, 28.8218, 32.3227, 29.9909, 28.1359, 35.1679, 34.2593, 31.6853,
        37.4501, 35.1688, 29.0616, 34.4197, 33.7305, 35.8051, 28.2845, 34.1819,
        31.7376, 28.0745, 27.6887, 26.6259, 26.5418, 36.1843, 37.0435, 18.8275,
        36.2401, 30.1164, 32.7155, 28.4911, 40.1670, 23.3525, 33.6660, 28.7961,
        32.5634, 30.0695, 38.1831, 25.8671, 35.3855, 29.4276, 25.5340, 31.7076,
        28.9375, 34.6924, 35.2259, 30.4895, 36.5806, 26.1659, 31.1273, 27.9157,
        23.0997, 26.1185, 33.8151, 28.2101, 34.9177, 28.6465, 35.3893, 35.2104,
        27.4315, 29.2590, 38.9361, 31.5801, 32.0660, 28.5591, 40.5025, 33.2855,
        22.3752, 44.8588, 23.4348, 35.3773, 32.3732, 28.0195, 25.2423, 32.1818,
        25.0839, 32.3762, 26.9003, 29.3822, 27.2485, 34.5039, 36.8043, 36.1858,
        32.5994, 35.9709, 24.9799, 35.4977, 32.6507, 25.1007, 32.6232, 36.1658,
        24.7293, 26.5523, 34.9503, 26.7479, 30.0014, 32.1010, 25.6673, 34.1563,
        35.3548, 31.2324, 34.7143, 34.8068, 34.9614, 31.3271, 45.1226, 31.4799,
        22.3554, 34.2364, 28.0151, 34.9978, 34.2181, 24.8030, 27.2197, 27.0746,
        35.4216, 33.8795, 32.3446, 28.2546, 30.9249, 15.2223, 22.9653, 21.2761,
        27.0232, 29.8401, 38.9312, 37.2357, 27.0027, 27.1587, 18.4485, 28.6706,
        19.0380, 26.3683, 20.7836, 41.7450, 39.4881, 27.5038, 32.5210, 37.1226,
        25.1215, 27.3763, 39.6606, 40.3426, 20.1609, 24.3522, 26.8969, 27.3511,
        35.7663, 35.1160, 27.8676, 27.9218, 33.0191, 31.9227, 32.4096, 28.9541,
        23.7000, 43.0200, 21.9861, 19.5275, 23.4484, 35.9190, 32.8450, 35.4759,
        30.2220, 32.1761, 38.7943, 43.6646, 30.5714, 24.9510, 28.3212, 24.6196,
        22.8526, 28.7477, 34.0735, 35.4019, 29.8494, 30.3463, 30.3888, 40.4951,
        31.1510, 29.8265, 29.1240, 21.9108, 27.5637, 25.6281, 28.0732, 30.8417,
        29.0864, 25.1468, 36.2714, 40.4969, 31.7381, 33.0612, 28.1122, 27.0139]), ten
sor([39.9230, 39.6465, 37.9976, 35.5204, 39.8744, 42.2790, 39.4298, 27.0651,
        41.9726, 38.4099, 31.5542, 41.8621, 38.5630, 36.6328, 38.7610, 37.0182,
        34.3182, 30.6214, 41.6903, 31.2341, 33.8939, 35.8579, 30.6952, 38.0364,
        32.4746, 33.6763, 29.0847, 36.9116, 31.0555, 30.9564, 42.9442, 38.0166,
        39.8304, 32.9269, 34.7647, 40.4249, 37.8073, 37.8912, 37.2044, 38.3955,
        39.2567, 37.8964, 37.4399, 29.0964, 37.9931, 31.4399, 34.4020, 40.1678,
        42.4791, 33.9179, 36.6593, 37.9876, 33.6316, 42.3090, 32.0577, 39.9046,
        30.9903, 35.3843, 36.6869, 35.9859, 36.9302, 41.6832, 33.8825, 34.5968,
        39.1948, 36.6980, 40.6994, 39.0458, 32.2271, 41.8288, 33.8357, 33.6104,
        34.1875, 30.7162, 38.6935, 31.5321, 37.3117, 34.8304, 38.0295, 34.0022,
        36.7446, 36.4578, 37.1443, 39.3112, 38.9071, 35.8546, 37.7689, 41.6605,
        35.1871, 33.0174, 38.9717, 36.5567, 28.0241, 32.8216, 34.0863, 40.7533,
        36.0917, 35.9125, 32.2261, 35.2882, 38.1424, 34.8101, 34.5960, 36.2790,
        37.9740, 35.7000, 33.9464, 28.6762, 41.6599, 38.4894, 37.5504, 38.6249,
        41.0356, 36.5799, 37.2192, 38.3563, 41.8823, 35.4486, 38.5926, 38.5859,
        34.2267, 33.4495, 35.4715, 31.1928, 37.1212, 41.2231, 29.9204, 38.0247,
        37.5635, 42.7579, 39.9938, 37.4861, 36.4855, 35.4988, 40.9614, 38.5179,
        36.9692, 35.0986, 35.0276, 34.2868, 40.7520, 32.9428, 39.5552, 33.0442,
        31.4350, 34.6910, 35.9850, 31.1288, 31.3124, 34.3027, 38.7505, 37.7974,
        38.6946, 28.6867, 39.3265, 36.5373, 39.6695, 34.7325, 41.7805, 38.7825,
        37.8073, 28.4683, 36.9260, 35.6490, 34.3258, 33.3095, 27.5454, 33.3738,
        37.4623, 36.5154, 39.3145, 41.7850, 35.6135, 35.8413, 36.1174, 32.8190,
        40.3012, 33.2864, 37.0981, 37.4921, 36.3460, 33.6297, 38.2177, 32.8899,
        35.0208, 41.2049, 32.2191, 39.0522, 36.8013, 33.0550, 39.6518, 39.1412,
        36.4886, 33.9737, 31.2476, 36.8885, 34.1752, 38.2275, 40.8002, 33.5039])]
```

2. Función init_random_parameters:

In [43]:
```
#  Genera una matris k x 2 con mu y sigma aleatorios
def init_random_parameters(k_parameters=2):
    p_matrix = []
```

```
    for k in range(k_parameters):
        mu = torch.tensor(random.uniform(MU_START, MU_END))
        sigma = torch.tensor(random.uniform(SIGMA_START, SIGMA_END))
        p_matrix.append([mu, sigma])
    p_matrix = torch.tensor(p_matrix)
    return p_matrix
```

In [44]:
```
random_parameters = init_random_parameters()
print(random_parameters)
```

```
tensor([[18.3479,  5.0695],
        [38.7589,  5.6826]])
```

3. Función calculate_likelihood_gaussian_observation:

In [45]:
```
#  Calcula la verosimilitud de una observacion con respecto a un mu y sigma para cada
#  Se utiliza la funcion de densidad de probabilidad gaussiana
def calculate_likelihood_gaussian_observation(x_n, mu_k, sigma_k):
    def probability_density_function(x, mu, sigma):
        return (1/math.sqrt(2 * math.pi * sigma**2)) * math.e**(-(1/2) * ((x-mu) / sig
    return probability_density_function(x_n, mu_k, sigma_k)
```

In [46]:
```
calculate_likelihood_gaussian_observation(42.3022, 28.0083, 1.5705)
```

Out[46]:
```
2.612131073896484e-19
```

Otra función alterna que calcula la verosimilitud para todo un conjunto de datos es:

In [47]:
```
def calculate_likelihood_dataset(parameters:torch.tensor, samples:torch.tensor):
  mean = parameters[:, 0][:, None]
  std = parameters[:, 1][:, None]

  bpart = (1 / torch.sqrt(2 * math.pi * std**2))
  fpart = math.e**(-(1/2) * ((samples.repeat(2, 1) - mean) / std)**2)

  return torch.nan_to_num(bpart * fpart)
```

In [48]:
```
#  Devuelve una matriz con todos los calculos de verosimilitud.
calculate_likelihood(random_parameters, dataset[0])
```

```
Out[48]: tensor([[3.5548e-03, 4.2556e-02, 9.4015e-03, 4.6028e-07, 1.7199e-03, 1.5505e-03,
          6.4954e-03, 2.6153e-03, 2.5276e-03, 9.3113e-03, 1.7612e-03, 5.6303e-03,
          1.2202e-02, 3.2025e-04, 5.7117e-04, 2.4711e-03, 6.4980e-05, 3.2006e-04,
          8.4351e-03, 5.1692e-04, 7.8810e-04, 2.0938e-04, 1.1526e-02, 5.9913e-04,
          2.4048e-03, 1.2490e-02, 1.4412e-02, 2.0746e-02, 2.1313e-02, 1.6138e-04,
          8.7626e-05, 7.8344e-02, 1.5524e-04, 5.3176e-03, 1.4182e-03, 1.0632e-02,
          7.4707e-06, 4.8341e-02, 8.1906e-04, 9.4094e-03, 1.5434e-03, 5.4327e-03,
          3.7293e-05, 2.6195e-02, 2.7748e-04, 7.2224e-03, 2.8814e-02, 2.4427e-03,
          8.8801e-03, 4.3523e-04, 3.0829e-04, 4.4702e-03, 1.2220e-04, 2.3961e-02,
          3.2812e-03, 1.3258e-02, 5.0717e-02, 2.4308e-02, 7.4912e-04, 1.1861e-02,
          3.7677e-04, 9.9949e-03, 2.7679e-04, 3.1145e-04, 1.5804e-02, 7.7627e-03,
          2.0627e-05, 2.6092e-03, 2.0224e-03, 1.0350e-02, 5.6069e-06, 1.0247e-03,
          5.7398e-02, 9.0664e-08, 4.7567e-02, 2.7899e-04, 1.7133e-03, 1.2752e-02,
          3.1212e-02, 1.9006e-03, 3.2551e-02, 1.7106e-03, 1.8964e-02, 7.3649e-03,
          1.6849e-02, 4.9034e-04, 1.0417e-04, 1.6122e-04, 1.5129e-03, 1.8698e-04,
          3.3443e-02, 2.5753e-04, 1.4704e-03, 3.2407e-02, 1.4931e-03, 1.6347e-04,
          3.5634e-02, 2.1241e-02, 3.6891e-04, 1.9941e-02, 5.6036e-03, 1.9849e-03,
          2.7751e-02, 6.0865e-04, 2.8318e-04, 3.1135e-03, 4.2922e-04, 4.0460e-04,
          3.6629e-04, 2.9685e-03, 6.8969e-08, 2.7469e-03, 5.7577e-02, 5.7933e-04,
          1.2773e-02, 3.5777e-04, 5.8591e-04, 3.4984e-02, 1.7018e-02, 1.7885e-02,
          2.7091e-04, 7.2056e-04, 1.7403e-03, 1.1660e-02, 3.6258e-03, 6.5074e-02,
          5.1975e-02, 6.6603e-02, 1.8198e-02, 6.0257e-03, 2.0707e-05, 7.6139e-05,
          1.8324e-02, 1.7378e-02, 7.8679e-02, 9.8990e-03, 7.7969e-02, 2.2512e-02,
          7.0116e-02, 1.8642e-06, 1.3177e-05, 1.5404e-02, 1.5799e-03, 8.2720e-05,
          3.2230e-02, 1.6114e-02, 1.1427e-05, 6.4323e-06, 7.3820e-02, 3.9024e-02,
          1.8985e-02, 1.6257e-02, 2.1497e-04, 3.3129e-04, 1.3496e-02, 1.3227e-02,
          1.1947e-03, 2.1823e-03, 1.6797e-03, 8.8196e-03, 4.5072e-02, 5.6579e-07,
          6.0828e-02, 7.6593e-02, 4.7439e-02, 1.9375e-04, 1.3187e-03, 2.6130e-04,
          5.0654e-03, 1.9065e-03, 2.3099e-05, 3.0226e-07, 4.3000e-03, 3.3692e-02,
          1.1363e-02, 3.6609e-02, 5.3025e-02, 9.5958e-03, 6.4038e-04, 2.7447e-04,
          6.0009e-03, 4.7812e-03, 4.6872e-03, 5.6426e-06, 3.2428e-03, 6.0624e-03,
          8.2179e-03, 6.1473e-02, 1.5077e-02, 2.8062e-02, 1.2496e-02, 3.7759e-03,
          8.3482e-03, 3.2016e-02, 1.5190e-04, 5.6339e-06, 2.4042e-03, 1.1663e-03,
          1.2313e-02, 1.8255e-02],
         [2.7410e-02, 2.3739e-03, 1.5107e-02, 5.1487e-02, 3.7287e-02, 3.8686e-02,
          1.9543e-02, 3.1577e-02, 3.2043e-02, 1.5218e-02, 3.6966e-02, 2.1350e-02,
          1.2233e-02, 5.7497e-02, 5.1311e-02, 3.2351e-02, 6.8366e-02, 5.7503e-02,
          1.6368e-02, 5.2450e-02, 4.7461e-02, 6.1332e-02, 1.2841e-02, 5.0756e-02,
          3.2723e-02, 1.1987e-02, 1.0526e-02, 7.1855e-03, 6.9611e-03, 6.3356e-02,
          6.7077e-02, 1.4962e-04, 6.3635e-02, 2.2084e-02, 3.9881e-02, 1.3722e-02,
          6.8081e-02, 1.7793e-03, 4.6983e-02, 1.5097e-02, 3.8748e-02, 2.1808e-02,
          6.9844e-02, 5.3550e-03, 5.8862e-02, 1.8232e-02, 4.6802e-03, 3.2509e-02,
          1.5766e-02, 5.4345e-02, 5.7866e-02, 2.4351e-02, 6.5231e-02, 6.0252e-03,
          2.8492e-02, 1.1369e-02, 1.5756e-03, 5.9145e-03, 4.8084e-02, 1.2534e-02,
          5.5865e-02, 1.4412e-02, 5.8885e-02, 5.7768e-02, 9.6281e-03, 1.7358e-02,
          7.0170e-02, 3.1609e-02, 3.5087e-02, 1.4021e-02, 6.6976e-02, 4.4148e-02,
          1.0999e-03, 3.9460e-02, 1.8504e-03, 5.8812e-02, 3.7339e-02, 1.1770e-02,
          4.1477e-03, 3.5931e-02, 3.8800e-03, 3.7361e-02, 7.9564e-03, 1.7994e-02,
          9.0248e-03, 5.3041e-02, 6.6171e-02, 6.3363e-02, 3.9015e-02, 6.2243e-02,
          3.7122e-03, 5.9544e-02, 3.9397e-02, 3.9078e-03, 3.9192e-02, 6.3262e-02,
          3.3325e-03, 6.9889e-03, 5.6081e-02, 7.5207e-03, 2.1411e-02, 3.5341e-02,
          4.9414e-03, 5.0572e-02, 5.8673e-02, 2.9203e-02, 5.4495e-02, 5.5122e-02,
          5.6155e-02, 2.9851e-02, 3.7501e-02, 3.0908e-02, 1.0888e-03, 5.1147e-02,
          1.1753e-02, 5.6394e-02, 5.1016e-02, 3.4407e-03, 8.9324e-03, 8.4782e-03,
          5.9084e-02, 4.8558e-02, 3.7127e-02, 1.2717e-02, 2.7143e-02, 1.3219e-05,
          1.4758e-03, 6.1808e-04, 8.3218e-03, 2.0486e-02, 7.0172e-02, 6.7726e-02,
          8.2601e-03, 8.7394e-03, 1.1815e-04, 1.4520e-02, 1.7026e-04, 6.5158e-03,
```

```
4.7162e-04, 6.1151e-02, 6.9628e-02, 9.8746e-03, 3.8433e-02, 6.7353e-02,
3.9423e-03, 9.4431e-03, 6.9326e-02, 6.7530e-02, 3.3149e-04, 2.8227e-03,
7.9466e-03, 9.3595e-03, 6.1113e-02, 5.7164e-02, 1.1187e-02, 1.1392e-02,
4.2152e-02, 3.4049e-02, 3.7607e-02, 1.5846e-02, 2.0963e-03, 5.2999e-02,
9.0069e-04, 2.2873e-04, 1.8623e-03, 6.1962e-02, 4.0849e-02, 5.9413e-02,
2.2714e-02, 3.5889e-02, 7.0203e-02, 4.8365e-02, 2.4865e-02, 3.6668e-03,
1.2994e-02, 3.1768e-03, 1.3964e-03, 1.4873e-02, 4.9973e-02, 5.8963e-02,
2.0539e-02, 2.3467e-02, 2.3727e-02, 6.7003e-02, 2.8651e-02, 2.0409e-02,
1.6677e-02, 8.6607e-04, 1.0082e-02, 4.8633e-03, 1.1982e-02, 2.6598e-02,
1.6490e-02, 3.9846e-03, 6.3790e-02, 6.6996e-02, 3.2727e-02, 4.2467e-02,
1.2137e-02, 8.2938e-03]])
```

4. Función calculate_membership_dataset:

In [49]:
```
#  Calcula la pertenencia de cada observación a los parámetros
#  Se utiliza one hot vector
def calculate_membership_dataset(x_dataset, parameters_matrix):
    likelihood_matrix = []
    for dataset in x_dataset:
        for data in dataset:
            data_likelihood = []
            for matrix in parameters_matrix:
                mu = matrix[0]
                sigma = matrix[1]
                likelihood = calculate_likelihood_gaussian_observation(data.item(), mu
                data_likelihood.append(likelihood)
            for index in range(len(data_likelihood)):
                data_likelihood[index] = 0 if data_likelihood[index] != max(data_likel
            likelihood_matrix.append(data_likelihood)
    likelihood_matrix = torch.tensor(likelihood_matrix)
    return likelihood_matrix
```

In [50]:
```
membership_matrix = calculate_membership_dataset(dataset, random_parameters)
print(membership_matrix)
```

```
tensor([[0, 1],
        [1, 0],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [1, 0],
        [1, 0],
        [1, 0],
        [1, 0],
        [0, 1],
        [0, 1],
        [1, 0],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [1, 0],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [1, 0],
        [0, 1],
        [0, 1],
        [1, 0],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [1, 0],
        [0, 1],
        [1, 0],
        [1, 0],
```

```
[1, 0],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[1, 0],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[1, 0],
[0, 1],
[1, 0],
[0, 1],
[0, 1],
[1, 0],
[1, 0],
[0, 1],
[1, 0],
[0, 1],
[1, 0],
[0, 1],
[1, 0],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[1, 0],
[0, 1],
[0, 1],
[1, 0],
[0, 1],
[0, 1],
[1, 0],
[1, 0],
[0, 1],
[1, 0],
[0, 1],
[0, 1],
[1, 0],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[1, 0],
[0, 1],
```

```
            [1, 0],
            [0, 1],
            [0, 1],
            [1, 0],
            [1, 0],
            [1, 0],
            [0, 1],
            [0, 1],
            [0, 1],
            [0, 1],
            [0, 1],
            [1, 0],
            [1, 0],
            [1, 0],
            [1, 0],
            [0, 1],
            [0, 1],
            [0, 1],
            [1, 0],
            [1, 0],
            [1, 0],
            [0, 1],
            [1, 0],
            [1, 0],
            [1, 0],
            [0, 1],
            [0, 1],
            [1, 0],
            [0, 1],
            [0, 1],
            [1, 0],
            [1, 0],
            [0, 1],
            [0, 1],
            [1, 0],
            [1, 0],
            [1, 0],
            [1, 0],
            [0, 1],
            [0, 1],
            [1, 0],
            [1, 0],
            [0, 1],
            [0, 1],
            [0, 1],
            [0, 1],
            [1, 0],
            [0, 1],
            [1, 0],
            [1, 0],
            [1, 0],
            [0, 1],
            [0, 1],
            [0, 1],
            [0, 1],
            [0, 1],
            [0, 1],
```

```
[0, 1],
[0, 1],
[1, 0],
[0, 1],
[1, 0],
[1, 0],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[1, 0],
[1, 0],
[1, 0],
[1, 0],
[0, 1],
[0, 1],
[1, 0],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[1, 0],
[1, 0],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[1, 0],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
```

```
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
```

```
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [1, 0],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
        [0, 1],
```

[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[1, 0],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],
[0, 1],

```
        [0, 1],
        [0, 1]])
```

Otra función alterna y simplificada que calcula matriz de membresía es:

In [51]:
```python
def calculate_membership_dataset_alt(parameters:torch.tensor, samples:torch.tensor):
  original = calculate_likelihood(parameters, samples)
  transpose_o = torch.t(original)
  maxvalues = torch.amax(transpose_o, 1)
  return torch.where(original == maxvalues, 1.0, 0.0)
```

In [52]:
```python
calculate_membership_dataset_alt(random_parameters, dataset[0])
```

Out[52]:
```
tensor([[0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 0., 0., 1., 0., 0., 0., 0.,
         0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1.,
         0., 1., 1., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
         1., 0., 1., 0., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0., 0., 0., 0., 0.,
         1., 0., 0., 1., 0., 0., 1., 1., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 1., 0., 1., 0., 0., 1., 1., 1., 0., 0., 0., 0., 0., 1.,
         1., 1., 1., 0., 0., 0., 1., 1., 1., 0., 1., 1., 1., 0., 0., 1., 0., 0.,
         1., 1., 0., 0., 1., 1., 1., 1., 0., 0., 1., 1., 0., 0., 0., 0., 1., 0.,
         1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 1., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 0., 0., 1., 0., 0., 0., 0.,
         1., 1.],
        [1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
         1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 1., 1., 0., 1., 1., 1., 1.,
         1., 0., 1., 1., 1., 1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1., 0.,
         1., 0., 0., 0., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1.,
         0., 1., 0., 1., 1., 0., 0., 1., 0., 1., 0., 1., 0., 1., 1., 1., 1., 1.,
         0., 1., 1., 0., 1., 1., 0., 0., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1.,
         1., 1., 1., 1., 0., 1., 0., 1., 1., 0., 0., 0., 1., 1., 1., 1., 1., 0.,
         0., 0., 0., 1., 1., 1., 0., 0., 0., 1., 0., 0., 0., 1., 1., 0., 1., 1.,
         0., 0., 1., 1., 0., 0., 0., 0., 1., 1., 0., 0., 1., 1., 1., 1., 0., 1.,
         0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 0., 0., 1., 1., 1.,
         1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 1., 1., 0., 1., 1., 1., 1.,
         0., 0.]])
```

5. Función recalculate_parameters:

In [53]:
```python
#  Se realiza el recálculo de los parámetros
def recalculate_parameters(x_dataset, membership_data):
    membership_data = torch.transpose(membership_data, 0, 1)
    complete_dataset = torch.Tensor()
    new_parameters = []
    for dataset in x_dataset:
        complete_dataset = torch.cat((complete_dataset, dataset))
    for k in membership_data:
        data_set_one = []
        for one_hot_data in range(len(k)):
            if k[one_hot_data].item() == 1:
                data_set_one.append(complete_dataset[one_hot_data])
        data_set_one = torch.Tensor(data_set_one)
        mu = torch.mean(data_set_one)
        sigma = torch.std(data_set_one, unbiased=True)
        # Heurístico: En caso de que ninguna distribución tenga membresía con los pará
```

```
        if mu.item() != mu.item() or sigma.item() != sigma.item():  # if nan
            params = init_random_parameters(1)
            mu = params[0][0]
            sigma = params[0][1]
            new_parameters.append([mu.item(), sigma.item()])
        else:
            new_parameters.append([mu.item(), sigma.item()])
    new_parameters = torch.Tensor(new_parameters)
    return new_parameters
```

In [54]:
```
membership_matrix = calculate_membership_dataset(dataset, random_parameters)
recalculate_parameters(dataset, membership_matrix)
```

Out[54]:
```
tensor([[25.0848,  2.8309],
        [35.2814,  3.7737]])
```

Otra función alterna para el recálculo de parámetros es:

In [55]:
```
def recalculate_parameters_alt(one_hot_vector, samples):
  values_per_membership = one_hot_vector * samples
  transpose = torch.t(values_per_membership)

  n_aux = torch.count_nonzero(transpose, 0)
  mean = torch.sum(values_per_membership, 1) / n_aux

  anti_neg_mean = torch.where(transpose == 0, 1, 0)
  anti_neg_mean = (anti_neg_mean * mean) + transpose

  std = torch.sqrt(torch.sum(torch.t((anti_neg_mean - mean)**2), 1) / n_aux)
  return torch.nan_to_num(torch.t(torch.stack([mean, std])))
```

In [56]:
```
membership_dataset = calculate_membership_dataset_alt(random_parameters, dataset[0])
recalculate_parameters_alt(membership_dataset, dataset[0])
```

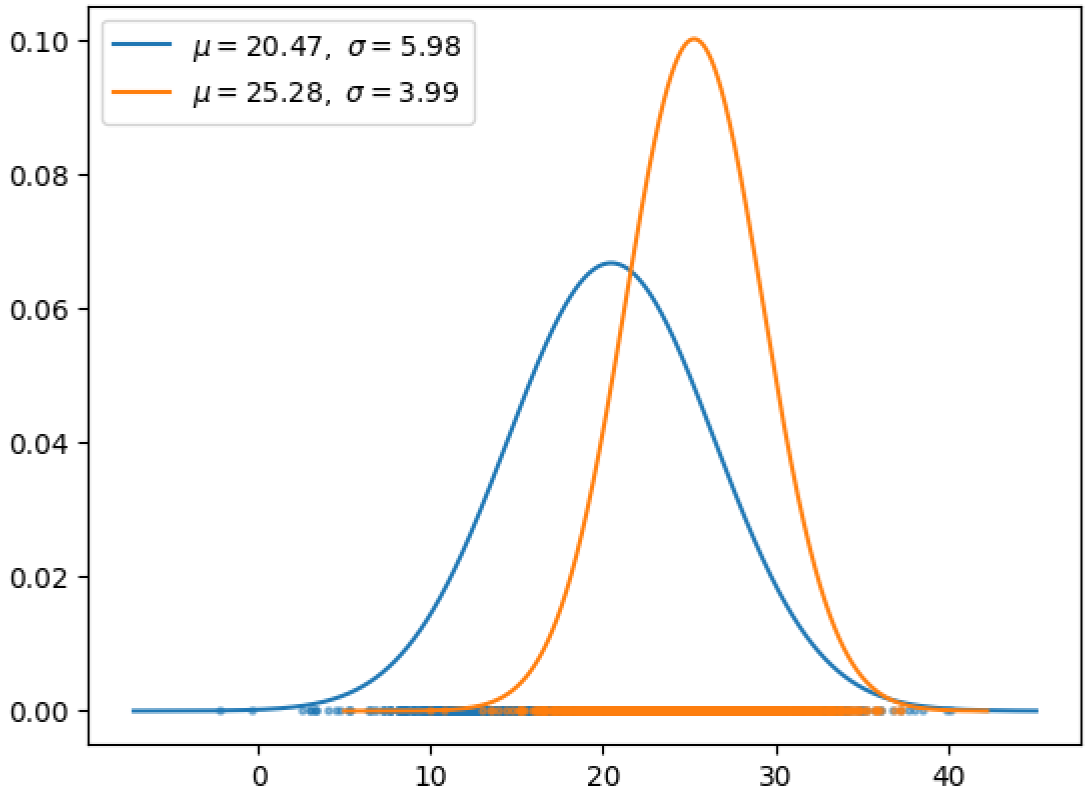
Out[56]:
```
tensor([[24.9713,  2.8216],
        [33.6141,  3.8052]])
```

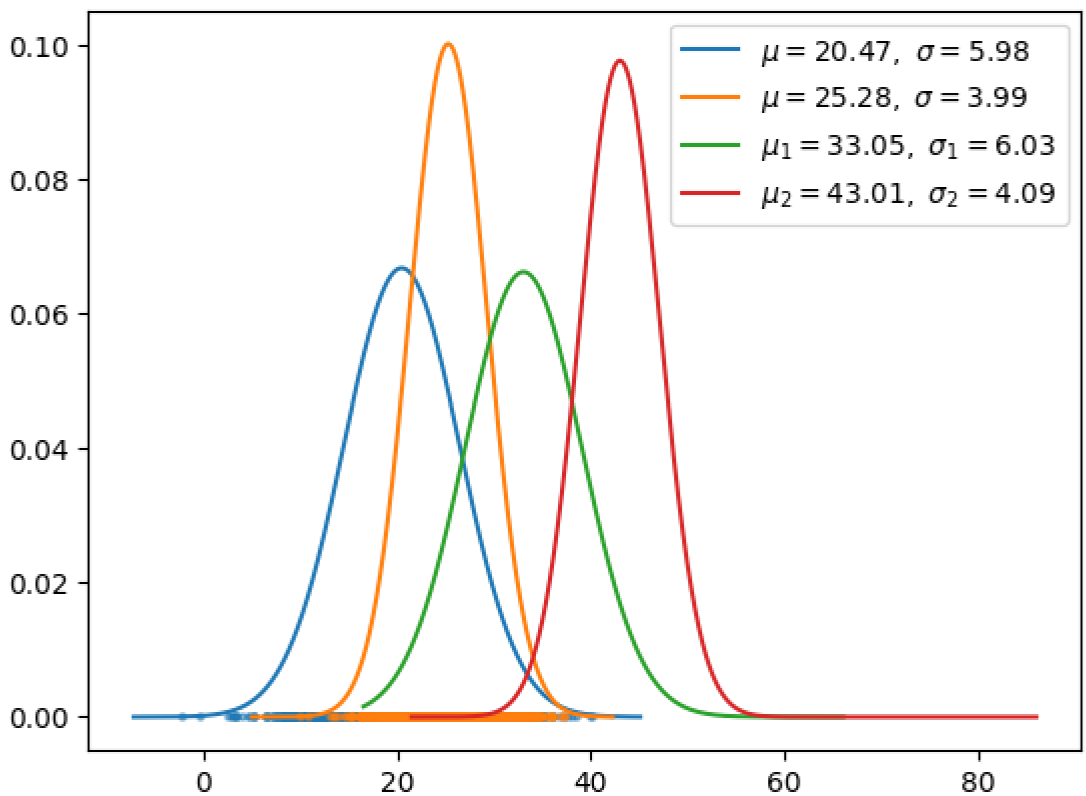
In [57]:
```
#  Algoritmo de maximizacion de la esperanza junto
def expectation_maximization(observations=200, k_parameters=2, iterations=5, heuristic
    my_data = generate_data(observations, k_parameters, show=True, heuristic=heuristic
    parameters = init_random_parameters(k_parameters)
    plot_gaussian_distribution_and_observations(parameters, my_data, show=True)
    for iteration in range(iterations):
        membership_data = calculate_membership_dataset(my_data, parameters)
        parameters = recalculate_parameters(my_data, membership_data)
        plot_gaussian_distribution_and_observations(parameters, my_data, show=True)
```

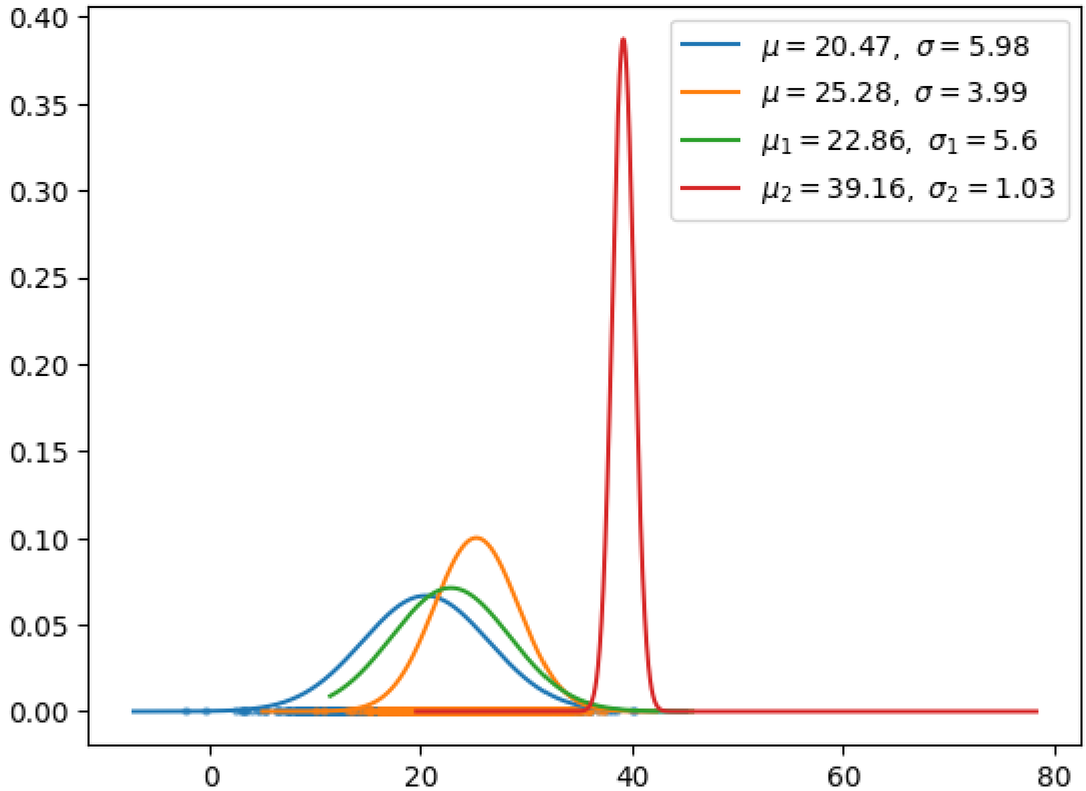
6. Ejecuta 5 corridas, donde por cada una documente los parámetros a los que se arribó.

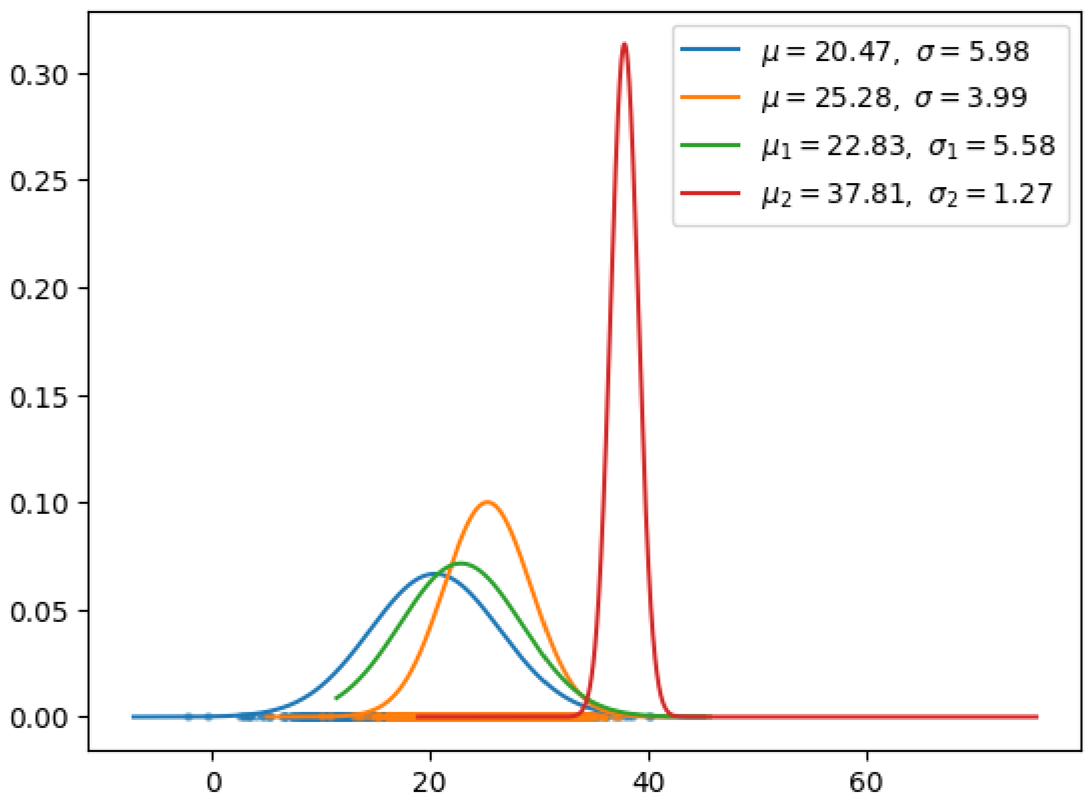
A continuación se muestran las corridas, los parámetros a los que se arribó se imprimen en los gráficos.

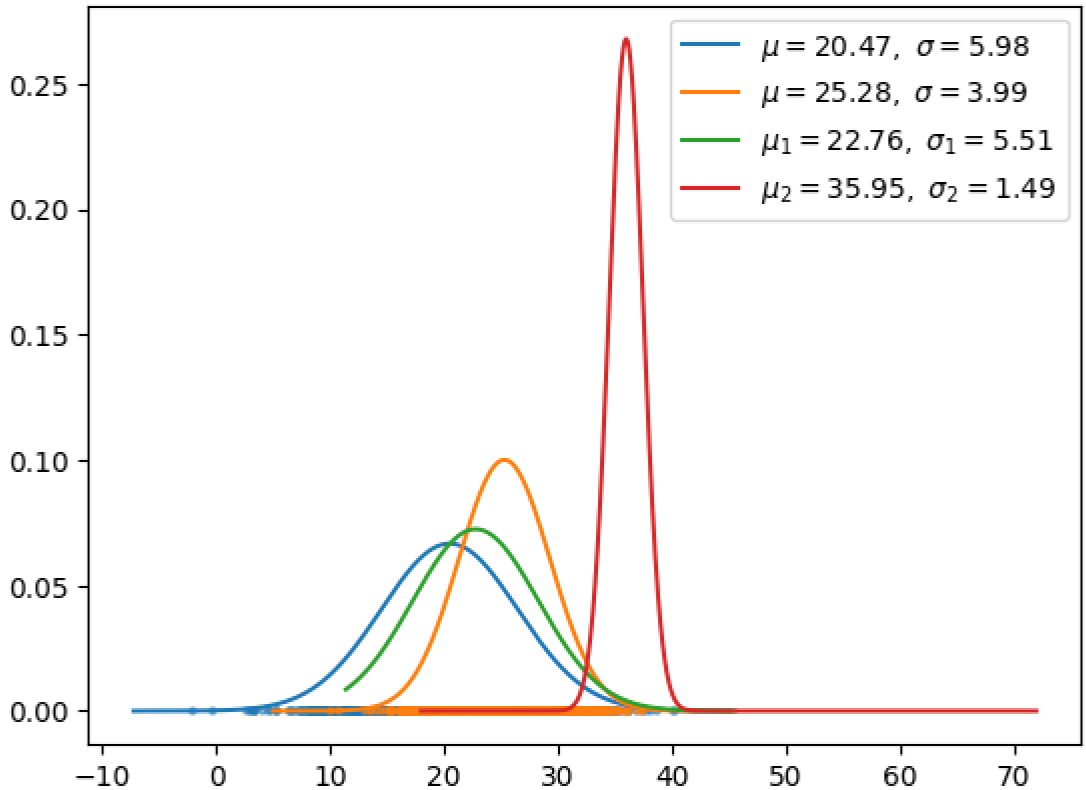
In [28]:
```
expectation_maximization(observations=2000, heuristic=True, k_parameters=2, iterations
```


$\mu = 20.47,\ \sigma = 5.98$
$\mu = 25.28,\ \sigma = 3.99$
0.10
0.08
0.06
0.04
0.02
0.00
0
10
20
30
40

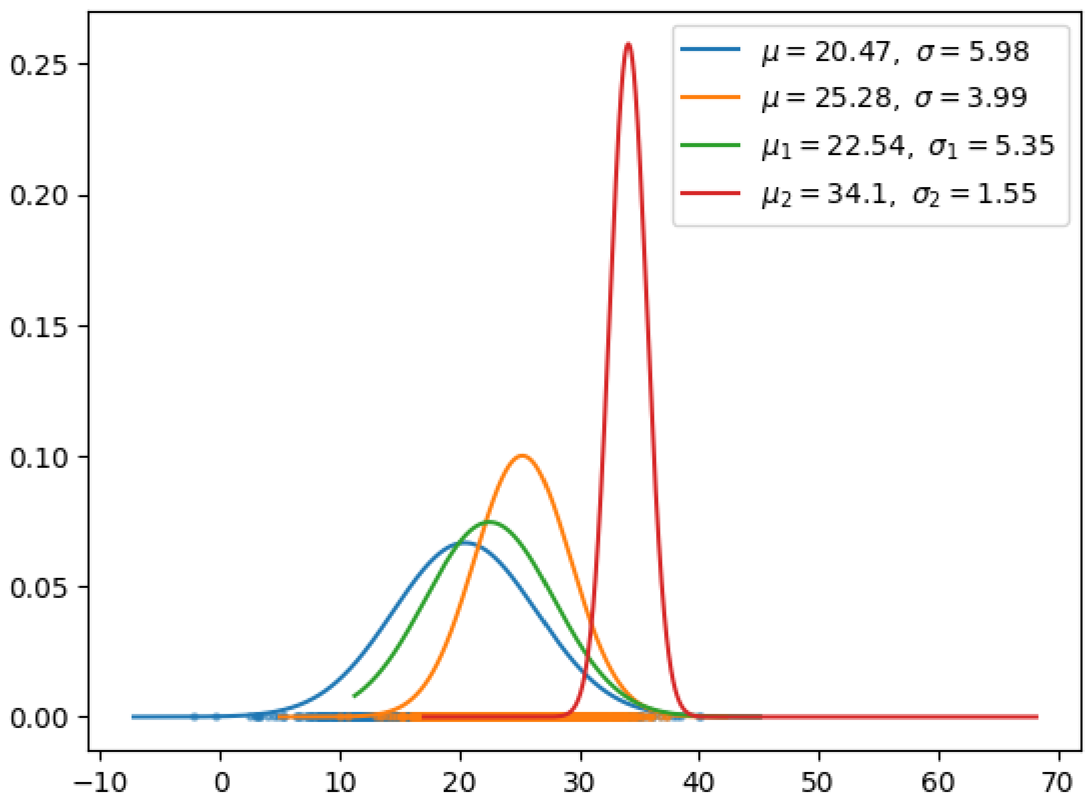

$\mu = 20.47,\ \sigma = 5.98$
$\mu = 25.28,\ \sigma = 3.99$
$\mu_1 = 33.05,\ \sigma_1 = 6.03$
$\mu_2 = 43.01,\ \sigma_2 = 4.09$
0.10
0.08
0.06
0.04
0.02
0.00
0
20
40
60
80


$\mu = 20.47,\ \sigma = 5.98$
$\mu = 25.28,\ \sigma = 3.99$
$\mu_1 = 22.86,\ \sigma_1 = 5.6$
$\mu_2 = 39.16,\ \sigma_2 = 1.03$
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80

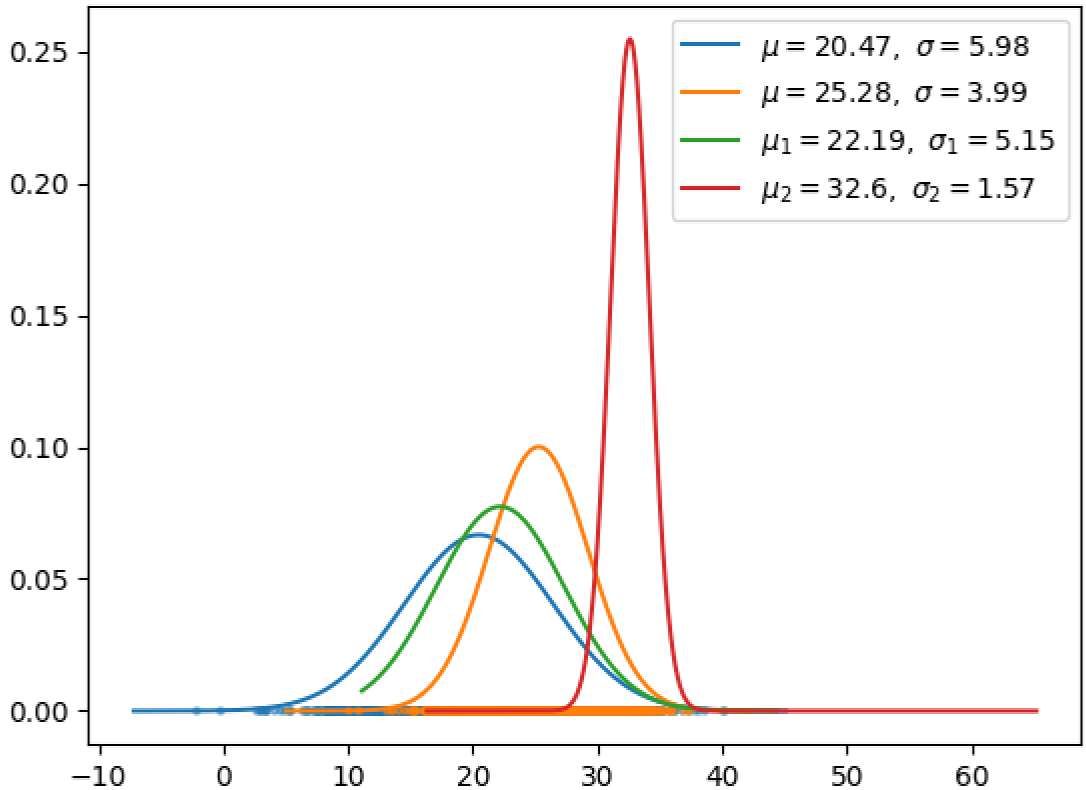

$\mu = 20.47,\ \sigma = 5.98$
$\mu = 25.28,\ \sigma = 3.99$
$\mu_1 = 22.83,\ \sigma_1 = 5.58$
$\mu_2 = 37.81,\ \sigma_2 = 1.27$
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60



$\mu = 20.47,\ \sigma = 5.98$
$\mu = 25.28,\ \sigma = 3.99$
$\mu_1 = 22.76,\ \sigma_1 = 5.51$
$\mu_2 = 35.95,\ \sigma_2 = 1.49$
0.25
0.20
0.15
0.10
0.05
0.00
−10 0 10 20 30 40 50 60 70



$\mu = 20.47,\ \sigma = 5.98$
$\mu = 25.28,\ \sigma = 3.99$
$\mu_1 = 22.54,\ \sigma_1 = 5.35$
$\mu_2 = 34.1,\ \sigma_2 = 1.55$
0.25
0.20
0.15
0.10
0.05
0.00
−10 0 10 20 30 40 50 60 70



7. Proponga una mejor heurística para inicializar los parámetros del modelo aleatoriamente.

La mejora heurística consiste en inicializar las observaciones de manera que algunos de sus datos se mezclen. Esto se logra inicializando μ's que tengan una diferencia pequeña definida por una constante que hemos denominado "HEURISTIC_STEP". Del mismo modo, se puede aumentar la cantidad de observaciones y el sigma se puede hacer más grande.

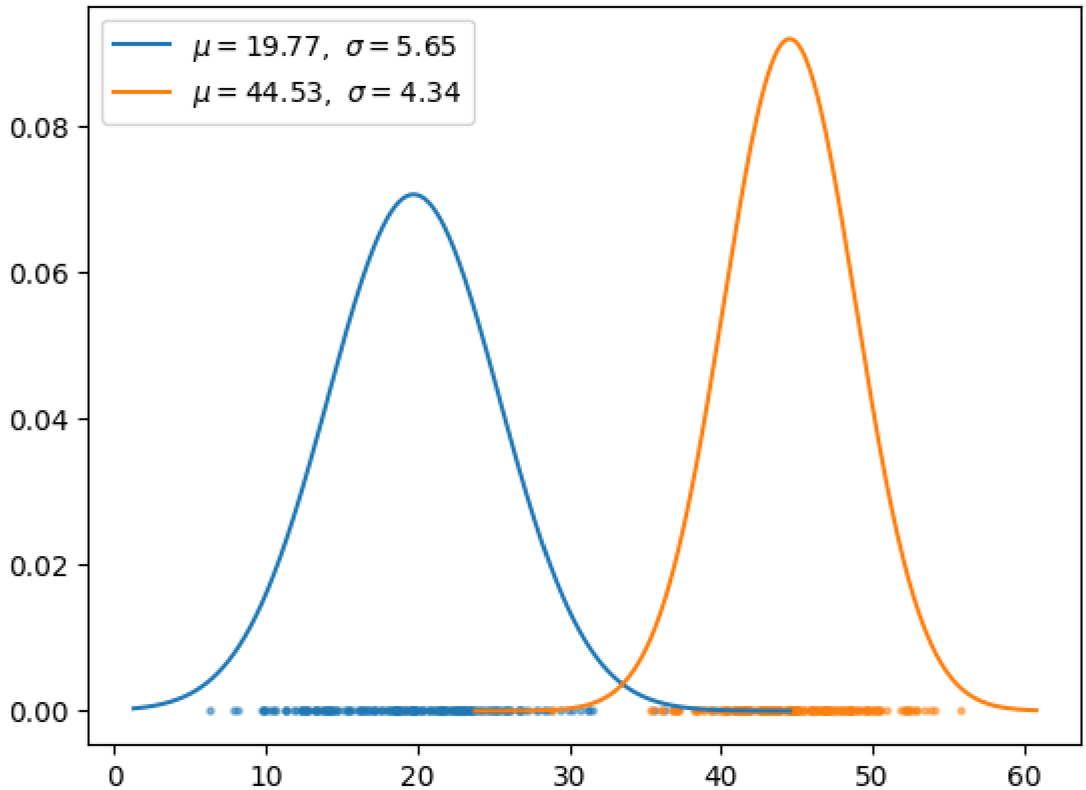
También, en caso de que la membresía de todas las observaciones se ajusten a un y solo un set de parámetros, se implementó la reinicialización aleatoria de los parámetros para aquel set que no tuvo ninguna membresía; esto fue colocado en la función recalculate_parameters.

Sin esta mejora, el algoritmo tiende mucho a quedarse atascado después de la primera iteración, no sucede el 100% de las veces, pero tiene tendencia a suceder bastante.

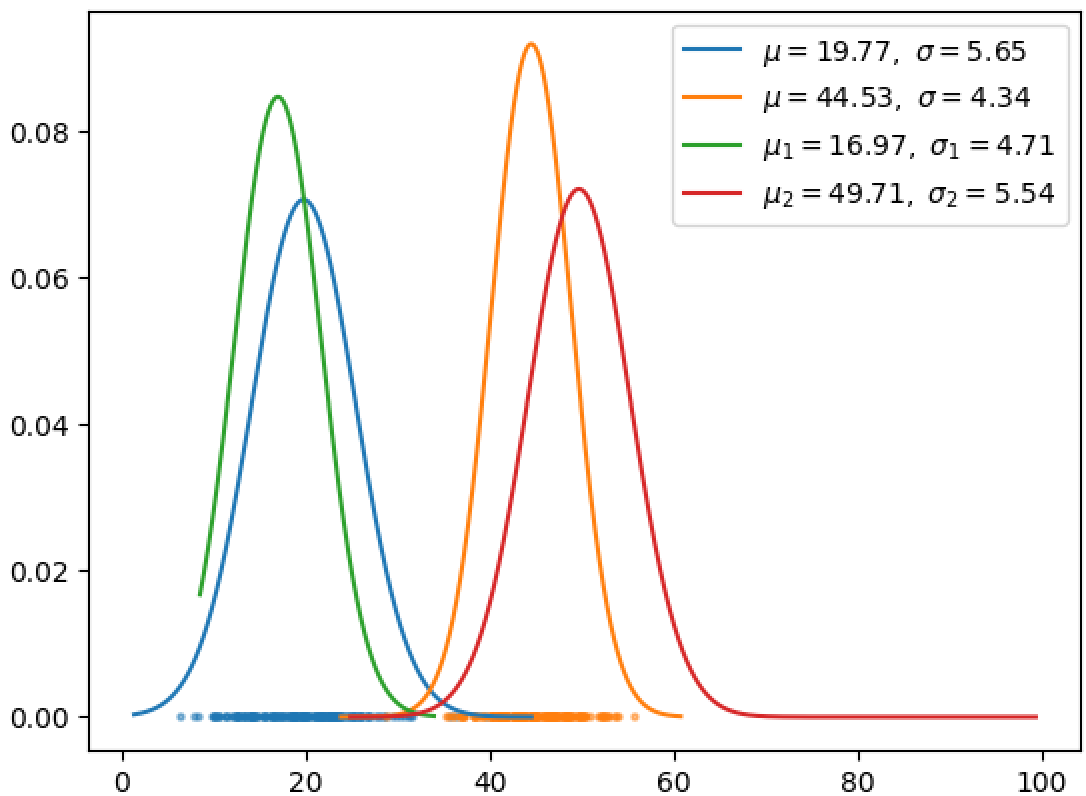
Se pone un ejemplo para ejecutar con el parámetro heuristic=False, en caso de que no suceda el comportamiento anteriormente descrito (lo cual es poco probable), se adjunta un ejemplo al final donde se vé que sí sucede:

(Se sigue tomando en cuenta la reinicialización aleatoria, porque de lo contrario podría causar errores)

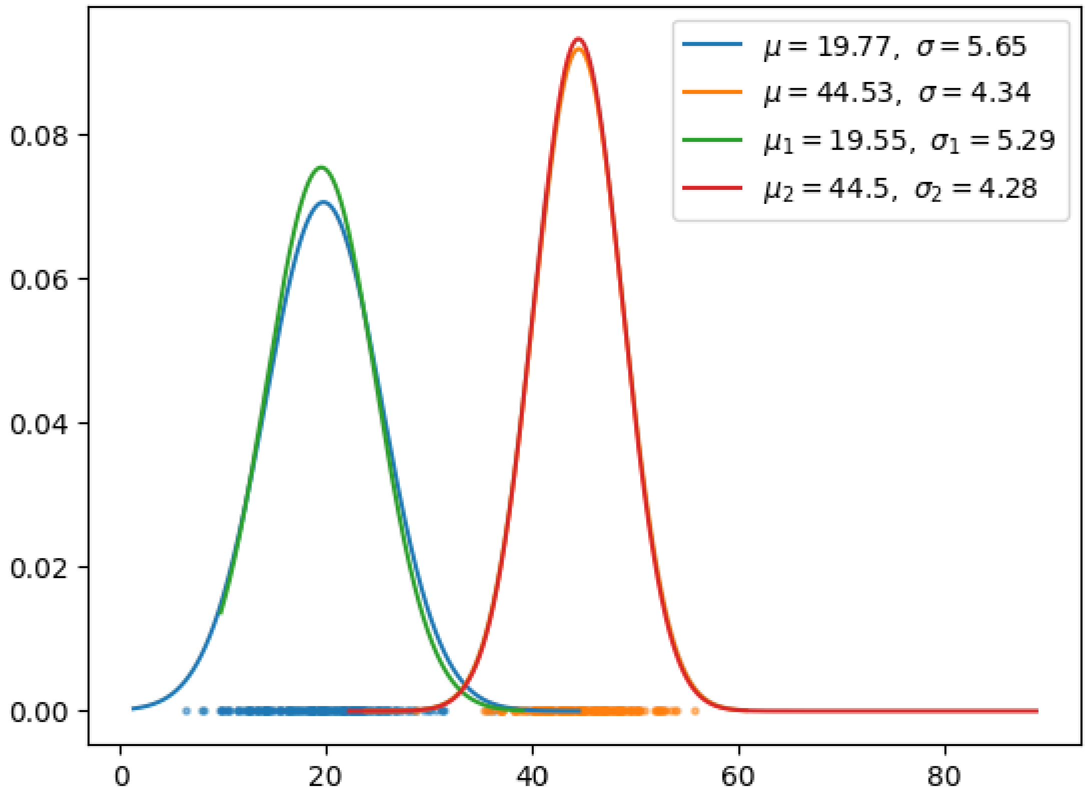
In [49]:
```
expectation_maximization(observations=200, heuristic=False, k_parameters=2, iterations
```



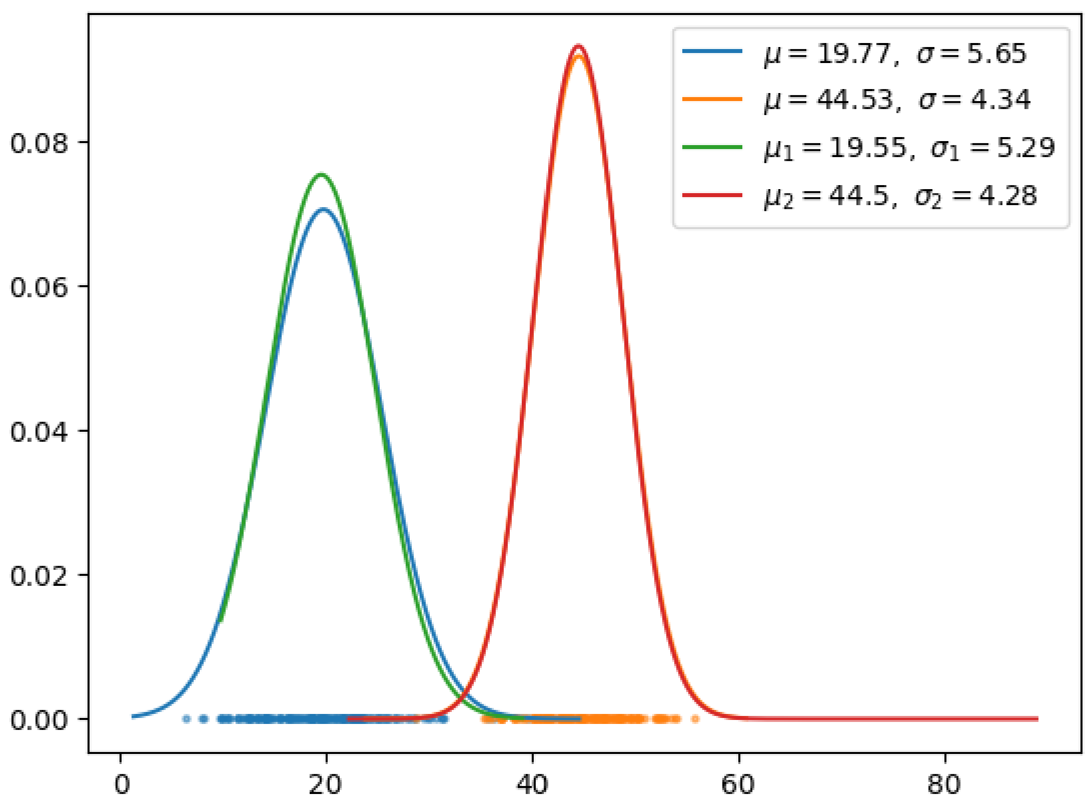
$\mu = 19.77,\ \sigma = 5.65$
$\mu = 44.53,\ \sigma = 4.34$
0.00
0.02
0.04
0.06
0.08
0
10
20
30
40
50
60



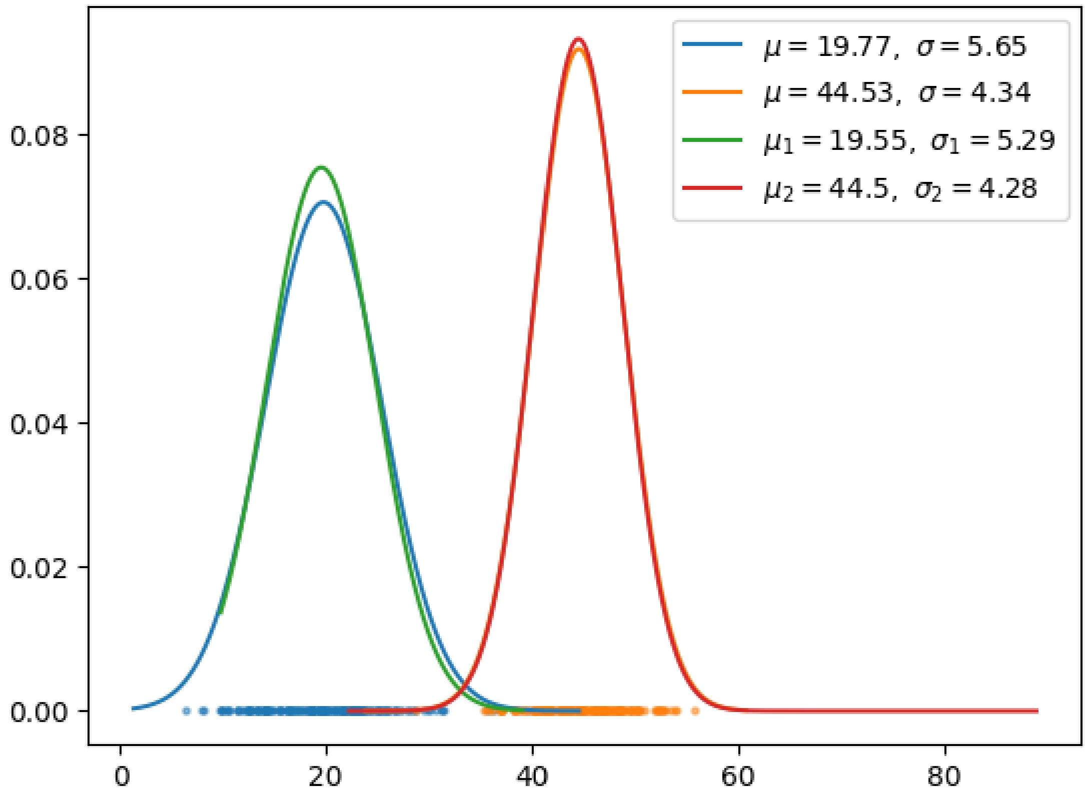
$\mu = 19.77,\ \sigma = 5.65$
$\mu = 44.53,\ \sigma = 4.34$
$\mu_1 = 16.97,\ \sigma_1 = 4.71$
$\mu_2 = 49.71,\ \sigma_2 = 5.54$
0.00
0.02
0.04
0.06
0.08
0
20
40
60
80
100

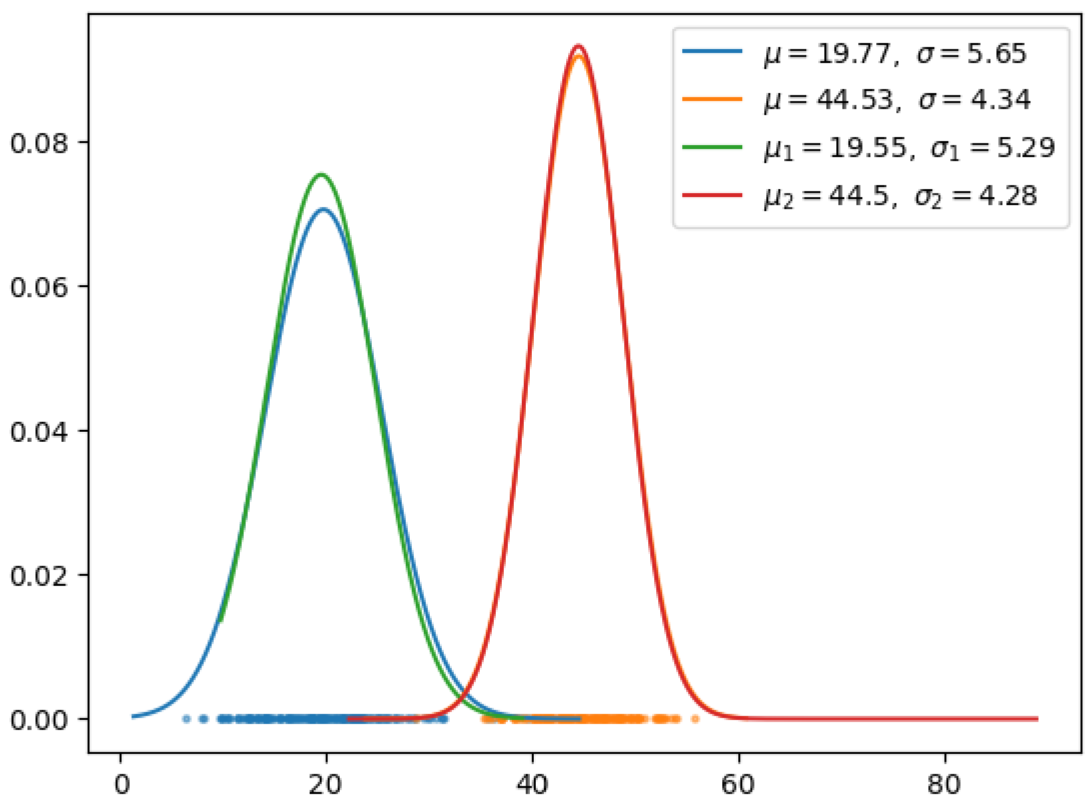


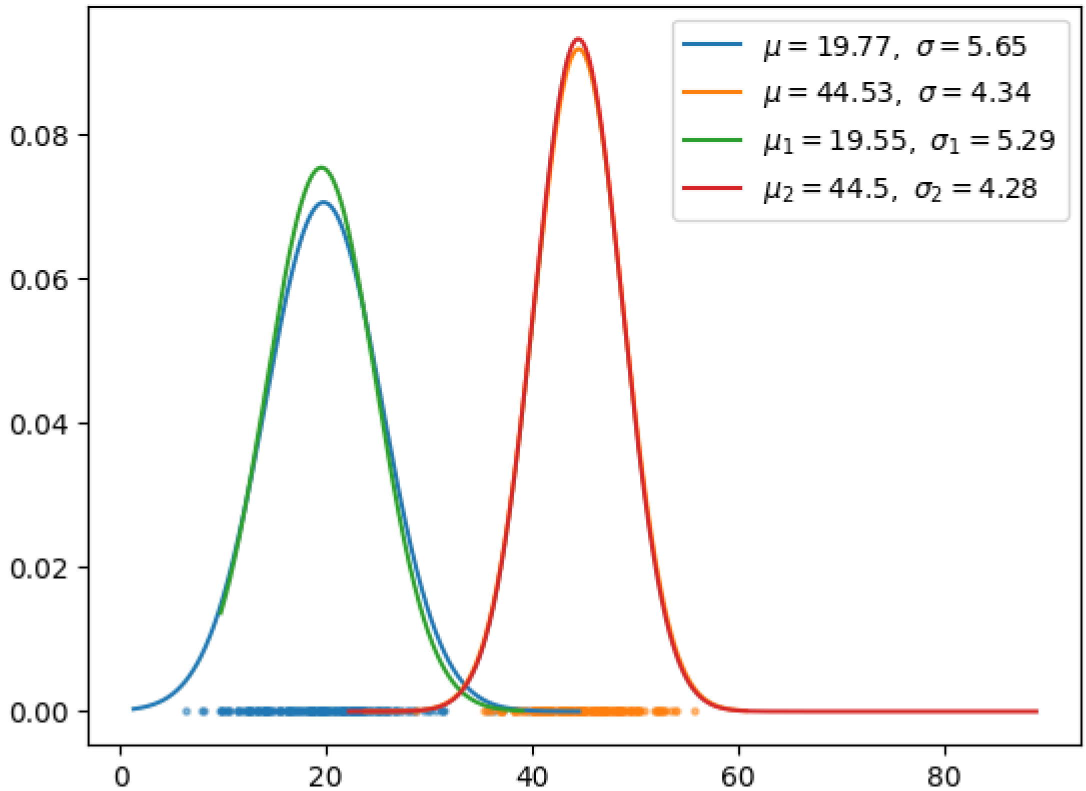
$\mu = 19.77,\ \sigma = 5.65$
$\mu = 44.53,\ \sigma = 4.34$
$\mu_1 = 19.55,\ \sigma_1 = 5.29$
$\mu_2 = 44.5,\ \sigma_2 = 4.28$
0.08
0.06
0.04
0.02
0.00
0
20
40
60
80



$\mu = 19.77,\ \sigma = 5.65$
$\mu = 44.53,\ \sigma = 4.34$
$\mu_1 = 19.55,\ \sigma_1 = 5.29$
$\mu_2 = 44.5,\ \sigma_2 = 4.28$
0.08
0.06
0.04
0.02
0.00
0
20
40
60
80

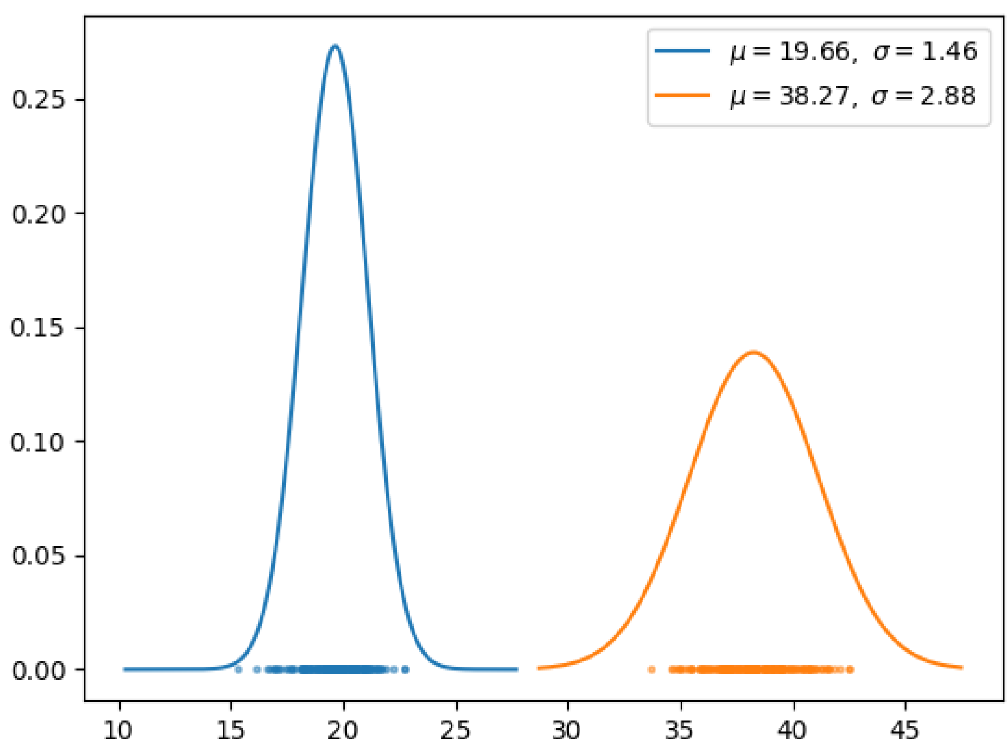

$\mu = 19.77,\ \sigma = 5.65$
$\mu = 44.53,\ \sigma = 4.34$
$\mu_1 = 19.55,\ \sigma_1 = 5.29$
$\mu_2 = 44.5,\ \sigma_2 = 4.28$
0.00
0.02
0.04
0.06
0.08
0
20
40
60
80

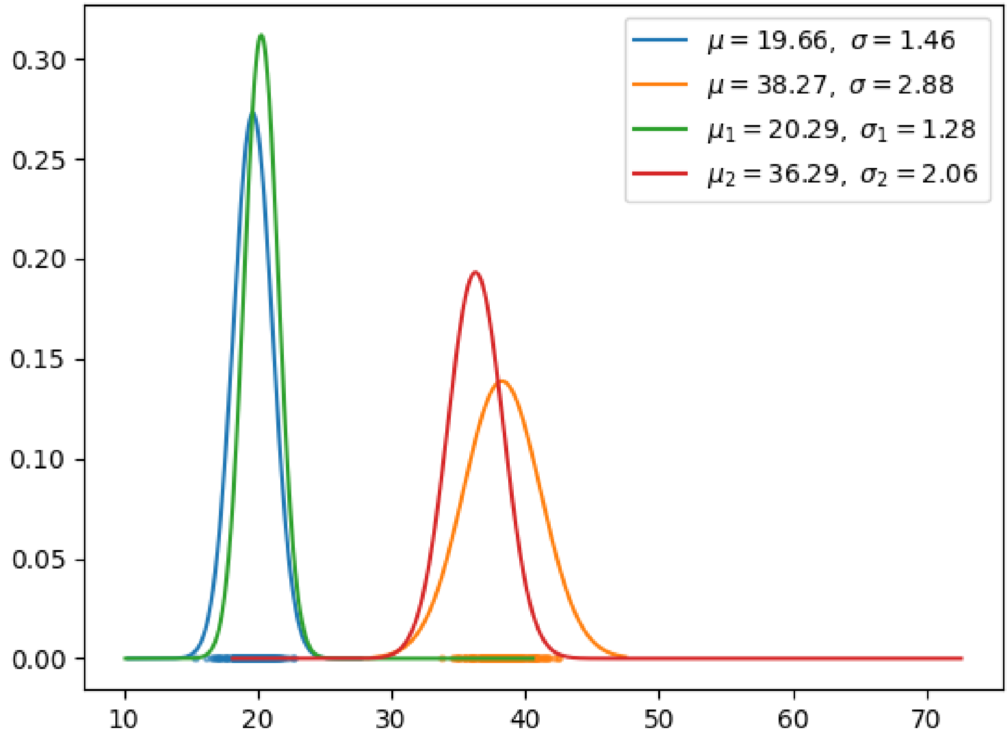

$\mu = 19.77,\ \sigma = 5.65$
$\mu = 44.53,\ \sigma = 4.34$
$\mu_1 = 19.55,\ \sigma_1 = 5.29$
$\mu_2 = 44.5,\ \sigma_2 = 4.28$
0.00
0.02
0.04
0.06
0.08
0
20
40
60
80

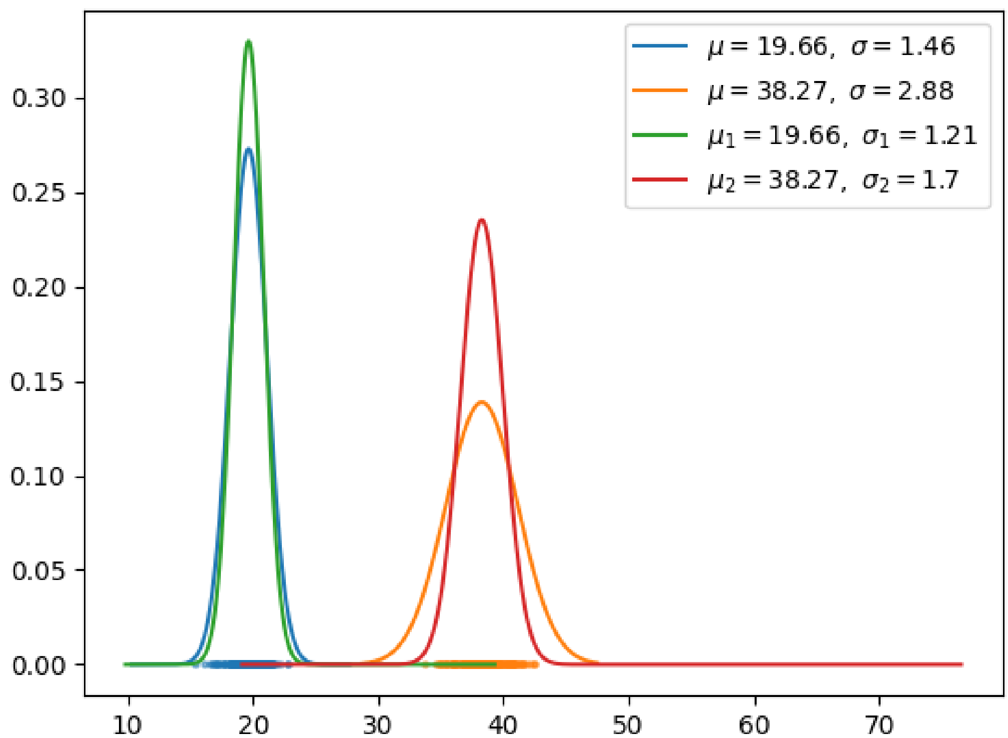


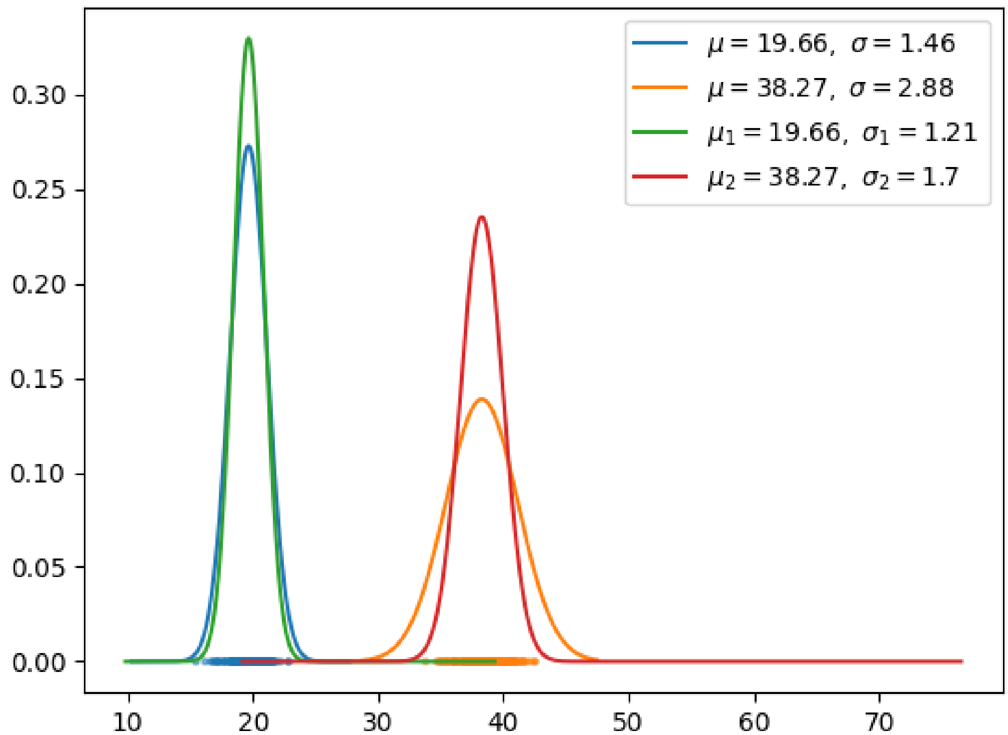
Ejemplo de imágenes:

IN [ ] : expectation_maximization(observations=200, heuristic=False, k_parameters=2, iterations=5)




$\mu = 19.66,\ \sigma = 1.46$
$\mu = 38.27,\ \sigma = 2.88$
$\mu_1 = 20.29,\ \sigma_1 = 1.28$
$\mu_2 = 36.29,\ \sigma_2 = 2.06$
0.30
0.25
0.20
0.15
0.10
0.05
0.00
10
20
30
40
50
60
70


$\mu = 19.66,\ \sigma = 1.46$
$\mu = 38.27,\ \sigma = 2.88$
$\mu_1 = 19.66,\ \sigma_1 = 1.21$
$\mu_2 = 38.27,\ \sigma_2 = 1.7$
0.30
0.25
0.20
0.15
0.10
0.05
0.00
10
20
30
40
50
60
70



Como se puede ver, después de la primera interación, los parámetros no volvieron a cambiar y así se mantiene hasta el final.

Con la mejora, esto no ha sucedido, por lo que nuestra propuesta tiene potencial para mejorar la inicialización de los parámetros.