Práctica 2.5 Método Sustractivo

Abarca Romero José Ángel

Lógica Difusa

2TM9

Gráficas:

10 puntos:

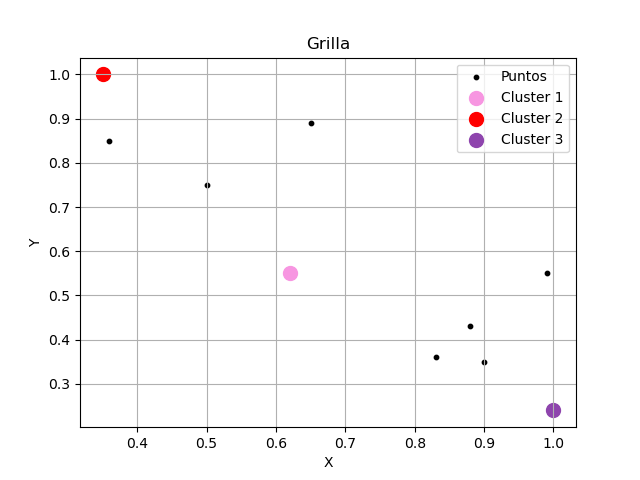


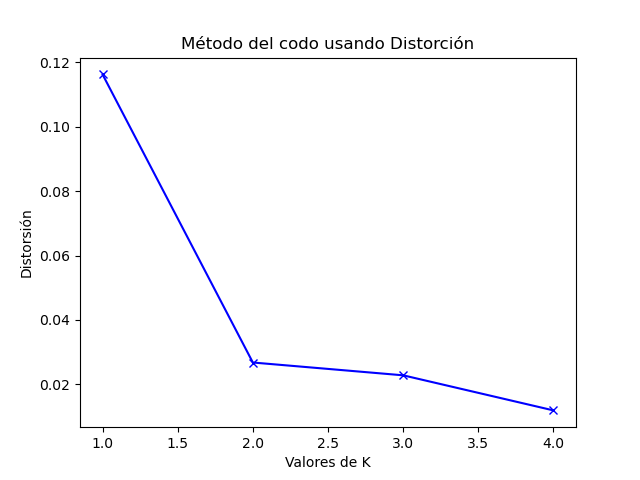
Ilustración 1 Distribución de datos y centros de clusters

Ilustración 2 Curva de distorsión para calcular el número de clusters

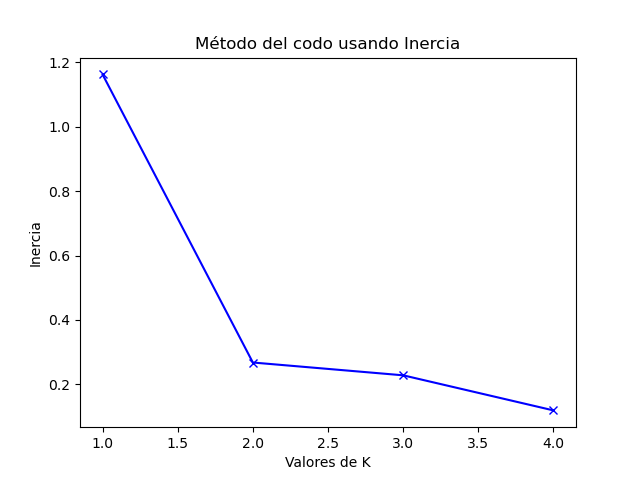


Ilustración 3 Curva de inercia para calcular el número de clusters

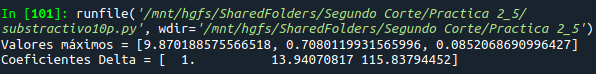


Ilustración 4 Cocientes de amplitudes

30 puntos:

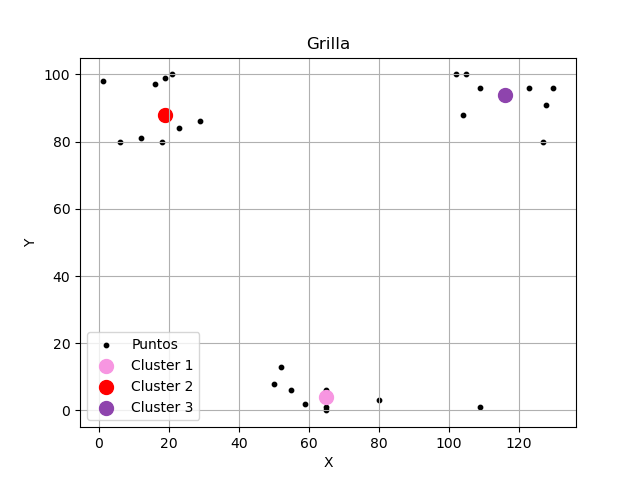


Ilustración 5 Distribución de datos y centros de clusters

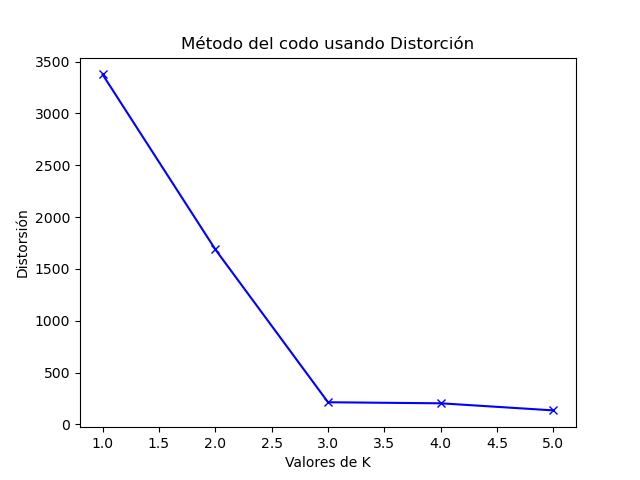


Ilustración 6 Curva de distorsión para calcular el número de clusters

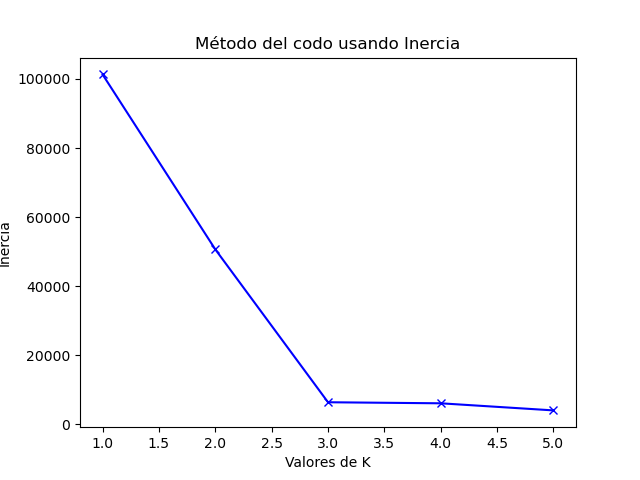


Ilustración 7 Curva de inercia para calcular el número de clusters

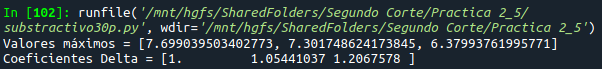


Ilustración 8 Cocientes de amplitudes

150 puntos:

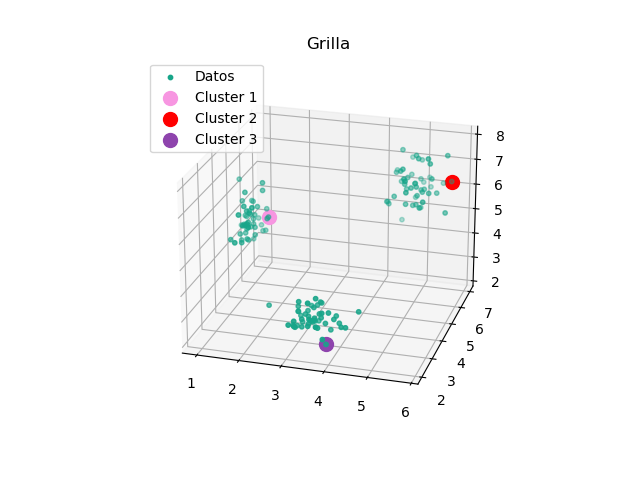


Ilustración 9 Distribución de datos y centros de clusters

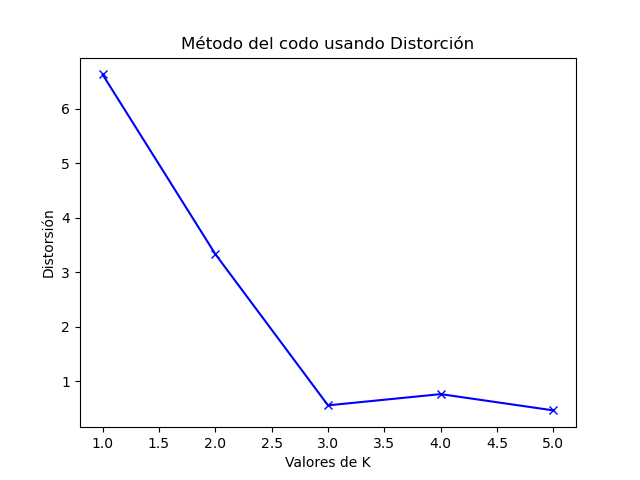


Ilustración 10 Curva de distorsión para calcular el número de clusters

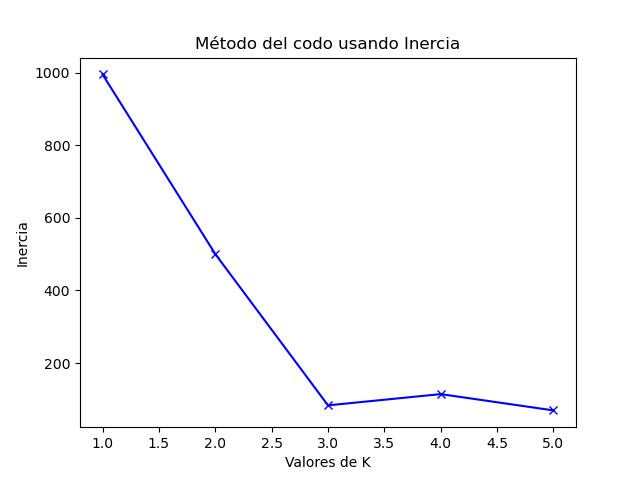


Ilustración 11 Curva de inercia para calcular el número de clusters

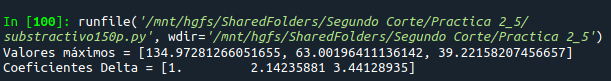


Ilustración 12 Cocientes de amplitudes

Códigos de Python:

Método sustractivo:

10 puntos:

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import cm

import math

import random

plt.close('all')

#Generación de puntos aleatorios entre 0 y 10

numero\_puntos = 10

xi = np.zeros((2,numero\_puntos))

xi[0] = [0.36, 0.65, 0.62, 0.5, 0.35, 0.9, 1, 0.99, 0.83, 0.88] # x

xi[1] = [0.85, 0.89, 0.55, 0.75, 1, 0.35, 0.24, 0.55, 0.36, 0.43] # y

# Radios

ra = 10

rb = 5

# Densidad

D = np.zeros(numero\_puntos)

for i in range(0,numero\_puntos):

    for j in range(0,numero\_puntos):

        distancia = math.sqrt( (xi[0][i] - xi[0][j])\*\*2 + \

                              (xi[1][i] - xi[1][j])\*\*2 )

        D[i] += math.exp( -distancia/(ra/2)\*\*2 )

indice\_maximo = np.argmax(D)

# Maximos

MD = []

indices\_maximos = []

MD.append(D[indice\_maximo])

indices\_maximos.append(indice\_maximo)

# Restructuración de densidades

cont = 0

umbral = 15

while MD[0]/MD[cont] < umbral:

    for i in range(0,numero\_puntos):

        distancia = math.sqrt( (xi[0][i] - xi[0][indice\_maximo])\*\*2 + \

                                  (xi[1][i] - xi[1][indice\_maximo])\*\*2 )

        D[i] = D[i] - MD[cont]\*math.exp( -distancia/(rb/2)\*\*2 )

    indice\_maximo = np.argmax(D)

    MD.append(D[indice\_maximo])

    indices\_maximos.append(indice\_maximo)

    cont += 1

colores = ["#F796E1","#FF0000","#8E44AD","#BCF558","#0000FF","#17A589"]

plt.figure(1)

plt.scatter(xi[0],xi[1],s=10,color='black',label='Puntos')

for i in range(len(MD)):

    plt.scatter(xi[0][indices\_maximos[i]],xi[1][indices\_maximos[i]],s=100,color = colores[i],label=f'Cluster {i+1}')

plt.xlabel('X')

plt.ylabel('Y')

plt.title('Grilla')

plt.grid(True)

plt.legend()

plt.show()

print("Valores máximos = {}".format(MD),\

      "Coeficientes Delta = {}".format(MD[0]/MD[:]))

30 puntos:

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import cm

import math

import random

plt.close('all')

#Generación de puntos aleatorios entre 0 y 10

numero\_puntos = 30

xi = np.zeros((2,numero\_puntos))

cont = 0

for i in range(0,2,1):

    for j in range(0,numero\_puntos,1):

        if i == 0:

            if cont < 10:

                xi[i][j] = random.randint(0,30)

            elif cont >= 10 and cont < 20:

                xi[i][j] = random.randint(50,80)

            else:

                xi[i][j] = random.randint(100,130)

        elif i == 1:

            if cont < 10 or cont > 20 :

                xi[i][j] = random.randint(80,100)

            else:

                xi[i][j] = random.randint(0,15)

        cont += 1

    cont = 0

# Radios

ra = 10

rb = 5

# Densidad

D = np.zeros(numero\_puntos)

for i in range(0,numero\_puntos):

    for j in range(0,numero\_puntos):

        distancia = math.sqrt( (xi[0][i] - xi[0][j])\*\*2 + \

                              (xi[1][i] - xi[1][j])\*\*2 )

        D[i] += math.exp( -distancia/(ra/2)\*\*2 )

indice\_maximo = np.argmax(D)

# Maximos

MD = []

indices\_maximos = []

MD.append(D[indice\_maximo])

indices\_maximos.append(indice\_maximo)

# Restructuración de densidades

cont = 0

umbral = 1.15

while MD[0]/MD[cont] < umbral:

    for i in range(0,numero\_puntos):

        distancia = math.sqrt( (xi[0][i] - xi[0][indice\_maximo])\*\*2 + \

                                  (xi[1][i] - xi[1][indice\_maximo])\*\*2 )

        D[i] = D[i] - MD[cont]\*math.exp( -distancia/(rb/2)\*\*2 )

    indice\_maximo = np.argmax(D)

    MD.append(D[indice\_maximo])

    indices\_maximos.append(indice\_maximo)

    cont += 1

colores = ["#F796E1","#FF0000","#8E44AD","#BCF558","#0000FF","#17A589"]

plt.figure(1)

plt.scatter(xi[0],xi[1],s=10,color='black',label='Puntos')

for i in range(len(MD)):

    plt.scatter(xi[0][indices\_maximos[i]],xi[1][indices\_maximos[i]],s=100,color = colores[i],label=f'Cluster {i+1}')

plt.xlabel('X')

plt.ylabel('Y')

plt.title('Grilla')

plt.grid(True)

plt.legend()

plt.show()

print("Valores máximos = {}".format(MD),\

      "Coeficientes Delta = {}".format(MD[0]/MD[:]))

150 puntos:

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import cm

import math

import random

plt.close('all')

#Generación de puntos aleatorios entre 0 y 10

xi = np.loadtxt('IrisDataBase.txt',usecols=(0,1,2)) # Definiendo variables

# Reordenamiento del arreglo

xi = xi.reshape(3,len(xi))

numero\_puntos = len(xi[0])

# Radios

ra = 10

rb = 5

# Densidad

D = np.zeros(numero\_puntos)

for i in range(0,numero\_puntos):

    for j in range(0,numero\_puntos):

        distancia = math.sqrt( (xi[0][i] - xi[0][j])\*\*2 + \

                              (xi[1][i] - xi[1][j])\*\*2 + \

                                  (xi[2][i] - xi[2][j])\*\*2 )

        D[i] += math.exp( -distancia/(ra/2)\*\*2 )

indice\_maximo = np.argmax(D)

# Maximos

MD = []

indices\_maximos = []

MD.append(D[indice\_maximo])

indices\_maximos.append(indice\_maximo)

# Restructuración de densidades

cont = 0

umbral = 3

while MD[0]/MD[cont] < umbral:

    for i in range(0,numero\_puntos):

        distancia = math.sqrt( (xi[0][i] - xi[0][indice\_maximo])\*\*2 + \

                                  (xi[1][i] - xi[1][indice\_maximo])\*\*2 + \

                                      (xi[2][i] - xi[2][indice\_maximo])\*\*2 )

        D[i] = D[i] - MD[cont]\*math.exp( -distancia/(rb/2)\*\*2 )

    indice\_maximo = np.argmax(D)

    MD.append(D[indice\_maximo])

    indices\_maximos.append(indice\_maximo)

    cont += 1

fig = plt.figure()

ax1 = fig.add\_subplot(111,projection='3d')

colores = ["#F796E1","#FF0000","#8E44AD","#BCF558","#0000FF","#17A589"]

plt.figure(1)

ax1.scatter(xi[0], xi[1],xi[2], s=10,c=colores[5],label = 'Datos')

for i in range(len(MD)):

    ax1.scatter(xi[0][indices\_maximos[i]],xi[1][indices\_maximos[i]],xi[2][indices\_maximos[i]],s=100,color = colores[i],label=f'Cluster {i+1}')

plt.title('Grilla')

plt.legend()

plt.show()

print("Valores máximos = {}".format(MD),\

      "Coeficientes Delta = {}".format(MD[0]/MD[:]))

Método del codo:

10 puntos:

import numpy as np

import matplotlib.pyplot as plt

import math

import random

plt.close('all')

# Puntos de datos

xi = np.zeros((2,30))

cont = 0

l1 = [0.36, 0.65, 0.62, 0.5, 0.35, 0.9, 1, 0.99, 0.83, 0.88] # x

l2 = [0.85, 0.89, 0.55, 0.75, 1, 0.35, 0.24, 0.55, 0.36, 0.43] # y

xi = [l1,l2]

numPuntos = len(xi[0])

#Número de clusters

K =  range(1,8)

distortions = []

inertias = []

for k in K:

    U = np.zeros((k,numPuntos))

    Um1 = np.zeros((k,numPuntos))

    for i in range(0,numPuntos,1):

        aux = random.randint(0,k-1)

        U[aux][i] = 1

        Um1[aux][i] = 1

    cont = 0

    while(True):

        #Cálculo de los centroides

        centrosxy = np.zeros((k,2))

        numx = 0

        denx = 0

        numy = 0

        deny = 0

        for i in range(0,k,1):

            for j in range(0,numPuntos,1):

                numx += U[i][j]\*xi[0][j]

                denx += U[i][j]

                numy += U[i][j]\*xi[1][j]

                deny += U[i][j]

            centrosxy[i][0] = numx/denx

            centrosxy[i][1] = numy/deny

            numx = 0

            denx = 0

            numy = 0

            deny = 0

        # #Distancias entre los centroides y los datos

        distancias = np.zeros((k,numPuntos))

        for j in range(0,k,1):

            for i in range(0,numPuntos,1):

                distancias[j][i] = math.sqrt((xi[0][i]-centrosxy[j][0])\*\*2 + (xi[1][i]-centrosxy[j][1])\*\*2)

        indices\_min = np.argmin(distancias, axis=0)

        #Actualización de U

        for i in range(0,numPuntos,1):

            indice = indices\_min[i]

            for j in range(0,k,1):

                if j == indice:

                    Um1[j][i] = 1

                else:

                    Um1[j][i] = 0

        cont += 1

        if np.array\_equal(U,Um1):

            break

        U = Um1

        #Final del while#

    distancia = 0

    pertenenciaU = np.argmax(U,axis=0)

    #Continuación del ciclo for k in K

    for i in range(0,numPuntos,1):

        indice = pertenenciaU[i]

        distancia += (xi[0][i]-centrosxy[indice][0])\*\*2 + (xi[1][i]-centrosxy[indice][1])\*\*2

    inertias.append(distancia)

    distancia = distancia/numPuntos

    distortions.append(distancia)

plt.figure(1)

plt.plot(K, distortions, 'bx-')

plt.xlabel('Valores de K')

plt.ylabel('Distorsión')

plt.title('Método del codo usando Distorción')

plt.show()

plt.figure(2)

plt.plot(K, inertias, 'bx-')

plt.xlabel('Valores de K')

plt.ylabel('Inercia')

plt.title('Método del codo usando Inercia')

plt.show()

30 puntos:

import numpy as np

import matplotlib.pyplot as plt

import math

import random

plt.close('all')

#Generación de puntos aleatorios entre 0 y 10

xi = np.zeros((2,30))

cont = 0

for i in range(0,2,1):

    for j in range(0,30,1):

        if i == 0:

            if cont < 10:

                xi[i][j] = random.randint(0,30)

            elif cont >= 10 and cont < 20:

                xi[i][j] = random.randint(50,80)

            else:

                xi[i][j] = random.randint(100,130)

        elif i == 1:

            if cont < 10 or cont > 20 :

                xi[i][j] = random.randint(80,100)

            else:

                xi[i][j] = random.randint(0,15)

        cont += 1

    cont = 0

#Número de clusters

K =  range(1,8)

distortions = []

inertias = []

for k in K:

    U = np.zeros((k,30))

    Um1 = np.zeros((k,30))

    for i in range(0,30,1):

        aux = random.randint(0,k-1)

        U[aux][i] = 1

        Um1[aux][i] = 1

    cont = 0

    while(True):

        #Cálculo de los centroides

        centrosxy = np.zeros((k,2))

        numx = 0

        denx = 0

        numy = 0

        deny = 0

        for i in range(0,k,1):

            for j in range(0,30,1):

                numx += U[i][j]\*xi[0][j]

                denx += U[i][j]

                numy += U[i][j]\*xi[1][j]

                deny += U[i][j]

            centrosxy[i][0] = numx/denx

            centrosxy[i][1] = numy/deny

            numx = 0

            denx = 0

            numy = 0

            deny = 0

        # #Distancias entre los centroides y los datos

        distancias = np.zeros((k,30))

        for j in range(0,k,1):

            for i in range(0,30,1):

                distancias[j][i] = math.sqrt((xi[0][i]-centrosxy[j][0])\*\*2 + (xi[1][i]-centrosxy[j][1])\*\*2)

        indices\_min = np.argmin(distancias, axis=0)

        #Actualización de U

        for i in range(0,30,1):

            indice = indices\_min[i]

            for j in range(0,k,1):

                if j == indice:

                    Um1[j][i] = 1

                else:

                    Um1[j][i] = 0

        cont += 1

        if np.array\_equal(U,Um1):

            break

        U = Um1

        #Final del while#

    distancia = 0

    pertenenciaU = np.argmax(U,axis=0)

    #Continuación del ciclo for k in K

    for i in range(0,30,1):

        indice = pertenenciaU[i]

        distancia += (xi[0][i]-centrosxy[indice][0])\*\*2 + (xi[1][i]-centrosxy[indice][1])\*\*2

    inertias.append(distancia)

    distancia = distancia/30

    distortions.append(distancia)

plt.figure(1)

plt.plot(K, distortions, 'bx-')

plt.xlabel('Valores de K')

plt.ylabel('Distorsión')

plt.title('Método del codo usando Distorción')

plt.show()

plt.figure(2)

plt.plot(K, inertias, 'bx-')

plt.xlabel('Valores de K')

plt.ylabel('Inercia')

plt.title('Método del codo usando Inercia')

plt.show()

150 puntos:

import numpy as np

import matplotlib.pyplot as plt

import math

import random

plt.close('all')

# Puntos a graficar

xi = np.loadtxt('IrisDataBase.txt',usecols=(0,1,2)) # Definiendo variables

# Reordenamiento del arreglo

xi = xi.reshape(3,len(xi))

l1 = xi[0] # x

l2 = xi[1] # y

l3 = xi[2] # z

numPuntos = len(xi[0])

#Número de clusters

K =  range(1,8)

distortions = []

inertias = []

for k in K:

    U = np.zeros((k,numPuntos))

    Um1 = np.zeros((k,numPuntos))

    for i in range(0,numPuntos,1):

        aux = random.randint(0,k-1)

        U[aux][i] = 1

        Um1[aux][i] = 1

    cont = 0

    while(True):

        #Cálculo de los centroides

        centroxyz = np.zeros((k,3))

        numx = 0

        denx = 0

        numy = 0

        deny = 0

        numz = 0

        denz = 0

        for i in range(0,k,1):

            for j in range(0,numPuntos,1):

                numx += U[i][j]\*xi[0][j]

                denx += U[i][j]

                numy += U[i][j]\*xi[1][j]

                deny += U[i][j]

                numz += U[i][j]\*xi[2][j]

                denz += U[i][j]

            centroxyz[i][0] = numx/denx

            centroxyz[i][1] = numy/deny

            centroxyz[i][2] = numz/denz

            numx = 0

            denx = 0

            numy = 0

            deny = 0

            numz = 0

            denz = 0

        # #Distancias entre los centroides y los datos

        distancias = np.zeros((k,numPuntos))

        for j in range(0,k,1):

            for i in range(0,numPuntos,1):

                distancias[j][i] = math.sqrt((xi[0][i]-centroxyz[j][0])\*\*2 + (xi[1][i]-centroxyz[j][1])\*\*2 \

                                             + (xi[2][i]-centroxyz[j][2])\*\*2 )

        indices\_min = np.argmin(distancias, axis=0)

        #Actualización de U

        for i in range(0,numPuntos,1):

            indice = indices\_min[i]

            for j in range(0,k,1):

                if j == indice:

                    Um1[j][i] = 1

                else:

                    Um1[j][i] = 0

        cont += 1

        if np.array\_equal(U,Um1):

            break

        U = Um1

        #Final del while#

    distancia = 0

    pertenenciaU = np.argmax(U,axis=0)

    #Continuación del ciclo for k in K

    for i in range(0,numPuntos,1):

        indice = pertenenciaU[i]

        distancia += (xi[0][i]-centroxyz[indice][0])\*\*2 + (xi[1][i]-centroxyz[indice][1])\*\*2 \

            + (xi[2][i]-centroxyz[indice][2])\*\*2

    inertias.append(distancia)

    distancia = distancia/numPuntos

    distortions.append(distancia)

plt.figure(1)

plt.plot(K, distortions, 'bx-')

plt.xlabel('Valores de K')

plt.ylabel('Distorsión')

plt.title('Método del codo usando Distorción')

plt.show()

plt.figure(2)

plt.plot(K, inertias, 'bx-')

plt.xlabel('Valores de K')

plt.ylabel('Inercia')

plt.title('Método del codo usando Inercia')

plt.show()