Práctica 2.5 Método de Montaña

Abarca Romero José Ángel

Lógica Difusa

2TM9

Gráficas:

10 puntos:

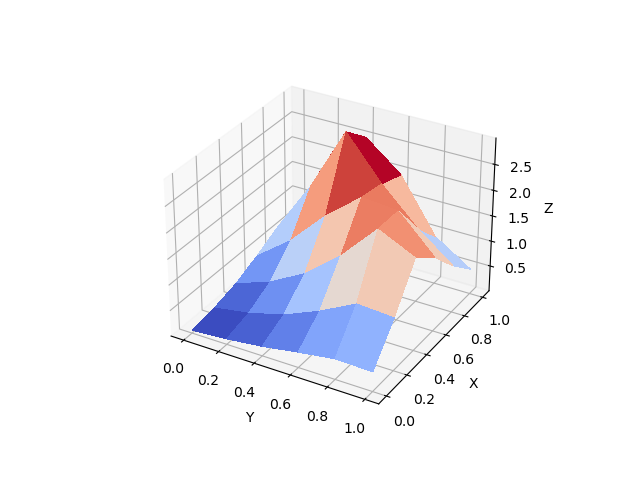


Ilustración Primera función de montaña

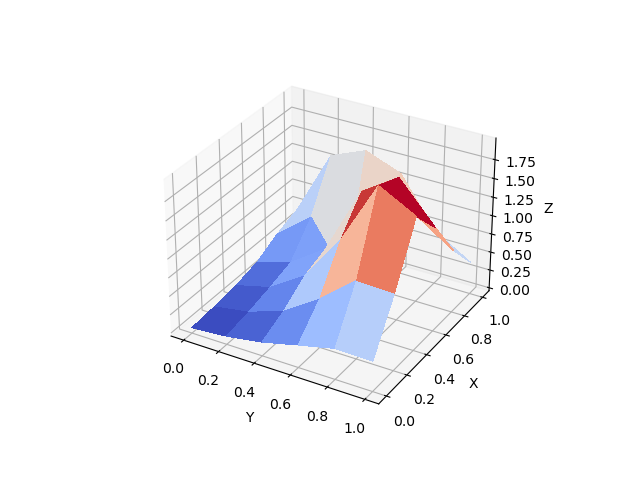


Ilustración 2 Segunda función de montaña

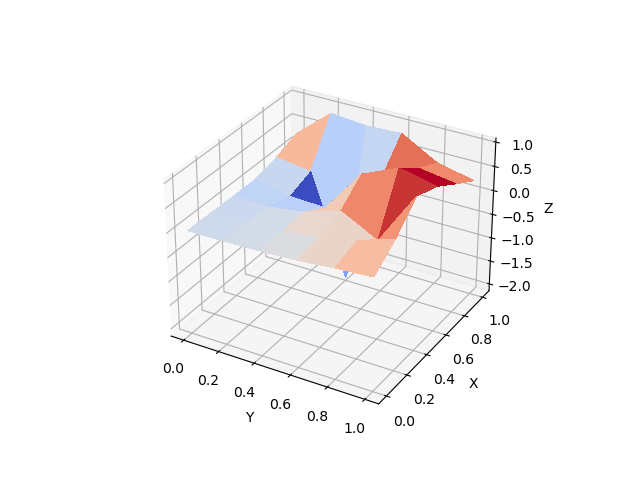


Ilustración 3 Tercera función de montaña

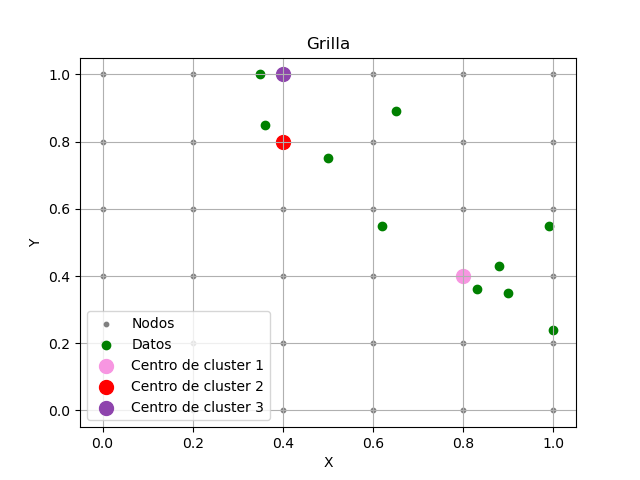


Ilustración 4 Distribución de datos y centros de clusters

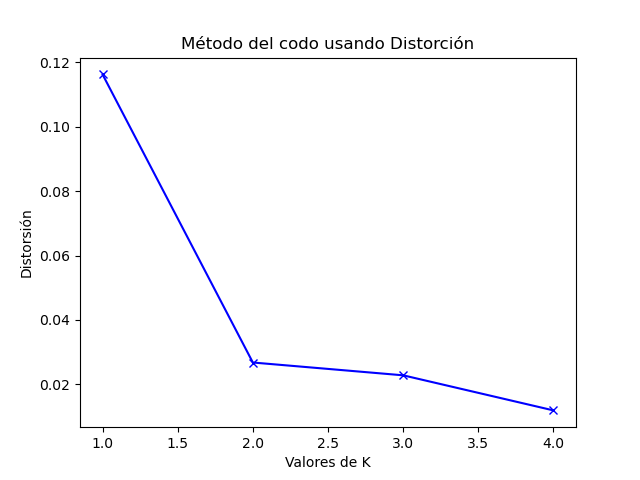


Ilustración 5 Curva de distorsión para calcular el número de clusters

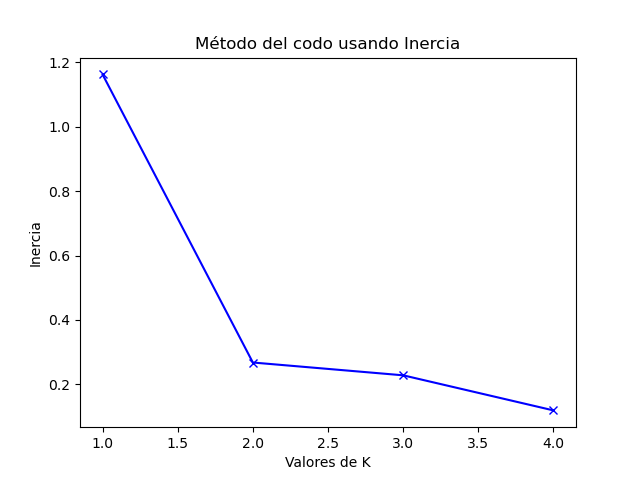


Ilustración 6 Curva de inercia para calcular el número de clusters

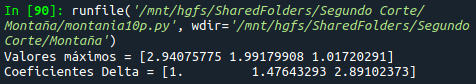


Ilustración 7 Cocientes de amplitudes

30 puntos:

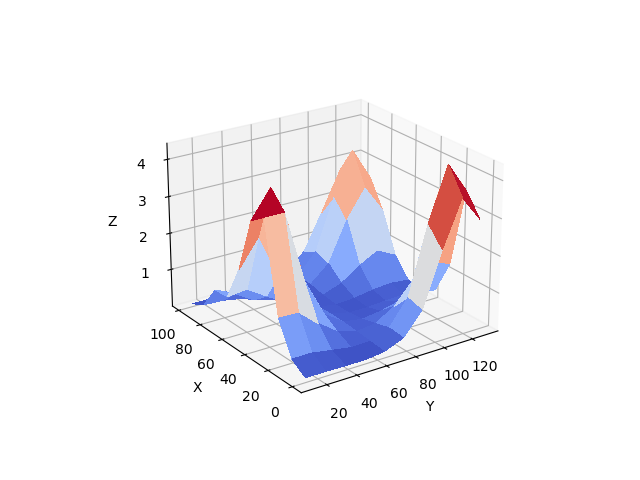


Ilustración 8 Primera función de montaña

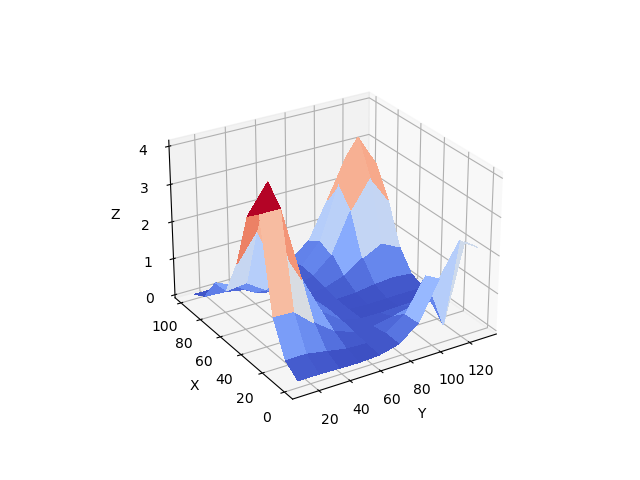


Ilustración 9 Segunda función de montaña

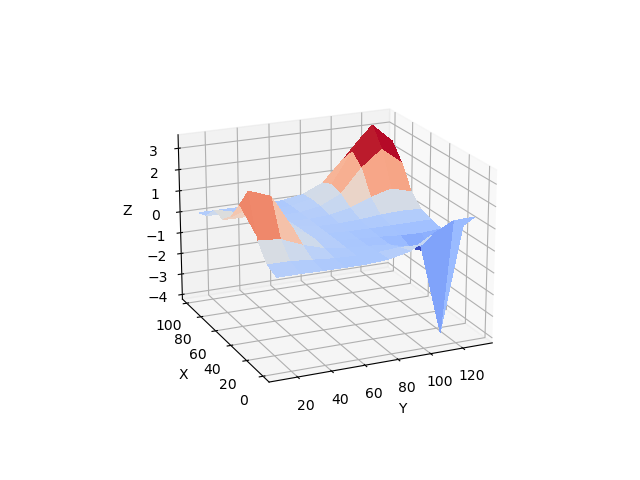


Ilustración 10 Tercera función de montaña

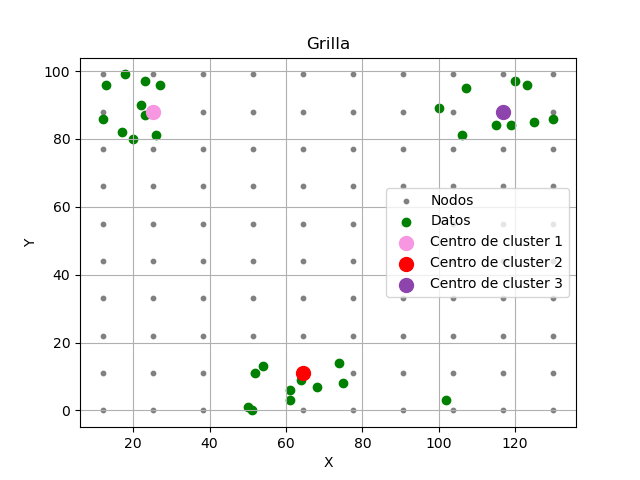


Ilustración 11 Distribución de datos y centros de clusters

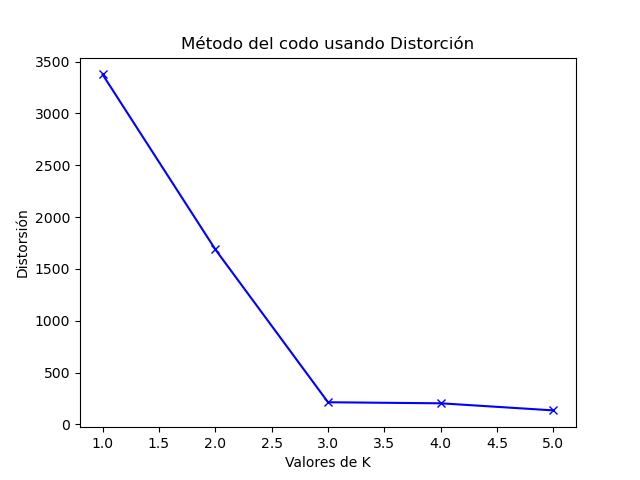


Ilustración 12 Curva de distorsión para calcular el número de clusters

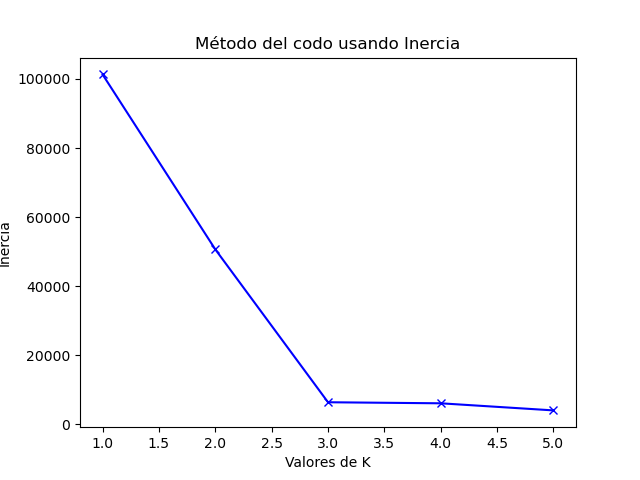


Ilustración 13 Curva de inercia para calcular el número de clusters

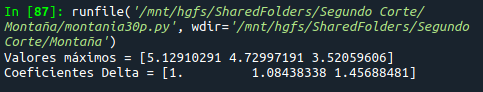


Ilustración 14 Cocientes de amplitudes

150 puntos:

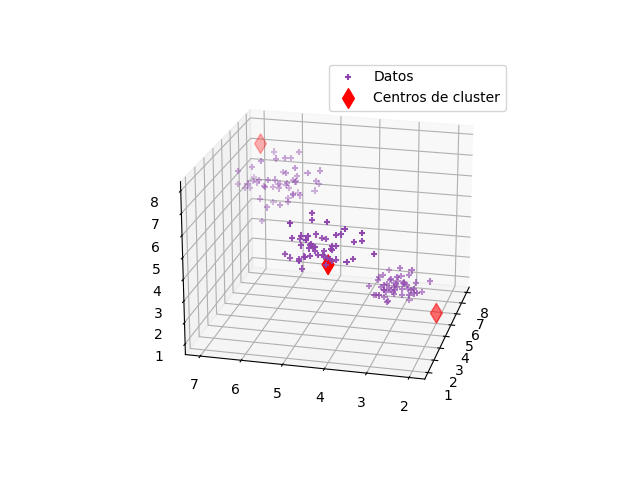


Ilustración 15 Distribución de datos y centros de clusters

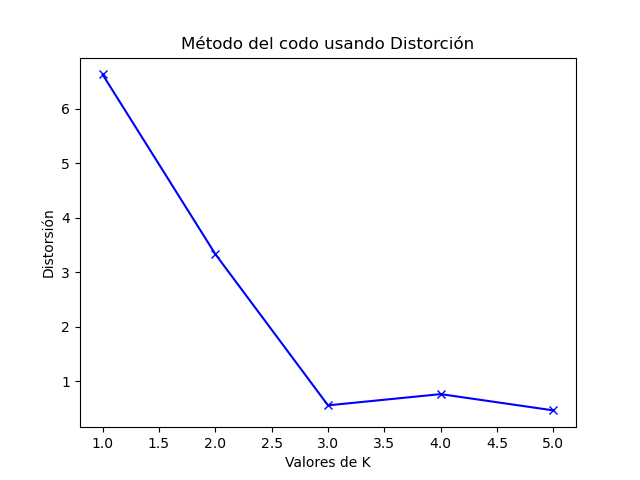


Ilustración 16 Curva de distorsión para calcular el número de clusters

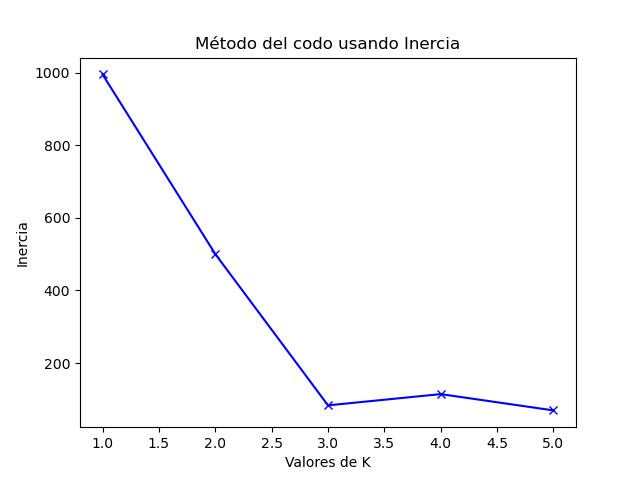


Ilustración 17 Curva de inercia para calcular el número de clusters

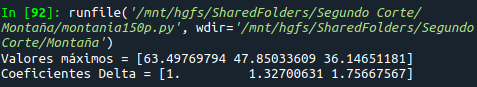


Ilustración 18 Cocientes de amplitudes

Códigos de Python:

Método de montaña:

10 puntos:

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import cm

import math

plt.close('all')

# Datos

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

#Grilla

xk = [l1,l2]

limites = [[min(l1), min(l2)],

           [max(l1), max(l2)]]

#Resolución de la grilla

resolucion = 6

num\_clusters = resolucion\*\*2

x = np.linspace(0,1,resolucion)

y = np.linspace(0,1,resolucion)

grid =  np.meshgrid(x,y)

X\_grid, Y\_grid = grid

# Función de montaña

M = np.zeros((resolucion,resolucion))

alpha = 5.4

beta = 5.4

# Condición de paro

delta = 2

N = range(0,2) # Número de clústers

# Obtención de la primera montaña

for i in range(len(X\_grid)):

    for j in range(len(Y\_grid)):

        for k in range(len(xk[0])):

            M[i][j] += math.e \*\* (-alpha\*math.sqrt( (X\_grid[0][i]-xk[0][k])\*\*2 + (Y\_grid[j][0]-xk[1][k])\*\*2 ))

# Coordenadas del valor máximo (N1)

indices\_maximo = np.argmax(M)

fila\_maximo, columna\_maximo = np.unravel\_index(indices\_maximo, M.shape)

# Valor máximo (pico de la montaña)

M1 = M[fila\_maximo][columna\_maximo]

# Valores máximos de los picos de montaña

valores\_maximos = []

valores\_maximos = np.append(valores\_maximos, M1)

# Coordenadas del valor máximo dentro de la matriz M[10][10]

coordenadas\_maximos = np.zeros((1,2))

coordenada\_maximo = [fila\_maximo,columna\_maximo]

coordenadas\_maximos[0] = coordenada\_maximo

# Plot 3D de la montaña

fig, ax = plt.subplots(subplot\_kw={"projection": "3d"})

ax.set\_xlabel('Y')

ax.set\_ylabel('X')

ax.set\_zlabel('Z')

surf = ax.plot\_surface(X\_grid, Y\_grid, M, cmap=cm.coolwarm,linewidth=0, antialiased=False)

# Cálculo de las siguientes montañas

cont = 0

while valores\_maximos[0]/valores\_maximos[cont] < delta: # Condición de paro

    if cont == 0:

        delta = 1.6

    for i in range(len(X\_grid)):

        for j in range(len(Y\_grid)):

            acumulable = 0

            for k in range(len(coordenadas\_maximos)):

                coordenadaX = int(coordenadas\_maximos[k][0])

                coordenadaY = int(coordenadas\_maximos[k][1])

                # Distancia euclidiana

                d = math.sqrt((X\_grid[0][coordenadaX]-X\_grid[0][i])\*\*2 + (Y\_grid[coordenadaY][0]-Y\_grid[j][0])\*\*2 )

                acumulable += math.exp( -beta\*d)

            M[i][j] = M[i][j] - valores\_maximos[cont]\*acumulable

    # Coordenadas del valor máximo (N1)

    indices\_maximo = np.argmax(M)

    fila\_maximo, columna\_maximo = np.unravel\_index(indices\_maximo, M.shape)

    # Valor máximo (pico de la montaña)

    M1 = M[fila\_maximo][columna\_maximo]

    valores\_maximos = np.append(valores\_maximos, M1)

    coordenada\_maximo = [fila\_maximo,columna\_maximo]

    coordenadas\_maximos = np.append(coordenadas\_maximos,[coordenada\_maximo],axis=0)

    # Plot 3D de la montaña

    fig, ax = plt.subplots(subplot\_kw={"projection": "3d"})

    ax.set\_xlabel('Y')

    ax.set\_ylabel('X')

    ax.set\_zlabel('Z')

    surf = ax.plot\_surface(X\_grid, Y\_grid, M, cmap=cm.coolwarm,linewidth=0, antialiased=False)

    cont += 1

# Plotear la grilla y puntos de datos

plt.figure(7)

plt.scatter(X\_grid, Y\_grid, s = 10 ,color='grey',label = "Nodos")

plt.scatter(l1,l2,color = 'green',label = "Datos")

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

for i in range(len(coordenadas\_maximos)):

    plt.scatter(X\_grid[0][int(coordenadas\_maximos[i][0])],Y\_grid[int(coordenadas\_maximos[i][1])][0],

                s = 100, c = colores[i], label = "Centro de cluster {}".format(i+1))

plt.xlabel('X')

plt.ylabel('Y')

plt.title('Grilla')

plt.grid(True)

plt.legend()

plt.show()

# Coeficiente delta

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

      "Coeficientes Delta = {}".format(valores\_maximos[0]/valores\_maximos[:]))

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

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

l1 = xi[0] # x

l2 = xi[1] # y

#Grilla

xk = [l1,l2]

limites = [[min(l1), min(l2)],

           [max(l1), max(l2)]]

#Resolución de la grilla

resolucion = 10

num\_clusters = resolucion\*\*2

x = np.linspace(limites[0][0],limites[1][0],resolucion)

y = np.linspace(limites[0][1],limites[1][1],resolucion)

grid =  np.meshgrid(x,y)

X\_grid, Y\_grid = grid

# Función de montaña

M = np.zeros((resolucion,resolucion))

alpha = 0.1

beta = 0.1

# Condición de paro

delta = 2

# Obtención de la primera montaña

for i in range(len(X\_grid)):

    for j in range(len(Y\_grid)):

        for k in range(len(xk[0])):

            M[i][j] += math.e \*\* (-alpha\*math.sqrt( (X\_grid[0][i]-xk[0][k])\*\*2 + (Y\_grid[j][0]-xk[1][k])\*\*2 ))

# Coordenadas del valor máximo (N1)

indices\_maximo = np.argmax(M)

fila\_maximo, columna\_maximo = np.unravel\_index(indices\_maximo, M.shape)

# Valor máximo (pico de la montaña)

M1 = M[fila\_maximo][columna\_maximo]

# Valores máximos de los picos de montaña

valores\_maximos = []

valores\_maximos = np.append(valores\_maximos, M1)

# Coordenadas del valor máximo dentro de la matriz M

coordenadas\_maximos = np.zeros((1,2))

coordenada\_maximo = [fila\_maximo,columna\_maximo]

coordenadas\_maximos[0] = coordenada\_maximo

# Plot 3D de la primera montaña

fig, ax = plt.subplots(subplot\_kw={"projection": "3d"})

ax.set\_xlabel('Y')

ax.set\_ylabel('X')

ax.set\_zlabel('Z')

surf = ax.plot\_surface(X\_grid, Y\_grid, M, cmap=cm.coolwarm,linewidth=0, antialiased=False)

# Cálculo de las siguientes montañas

cont = 0

while valores\_maximos[0]/valores\_maximos[cont] < delta: # Condición de paro

    if cont == 0:

        delta = 1.3

    for i in range(len(X\_grid)):

        for j in range(len(Y\_grid)):

            acumulable = 0

            for k in range(len(coordenadas\_maximos)):

                coordenadaX = int(coordenadas\_maximos[k][0])

                coordenadaY = int(coordenadas\_maximos[k][1])

                # Distancia euclidiana

                d = math.sqrt((X\_grid[0][coordenadaX]-X\_grid[0][i])\*\*2 + (Y\_grid[coordenadaY][0]-Y\_grid[j][0])\*\*2 )

                acumulable += math.exp( -beta\*d)

            M[i][j] = M[i][j] - valores\_maximos[cont]\*acumulable

    # Coordenadas del valor máximo (N1)

    indices\_maximo = np.argmax(M)

    fila\_maximo, columna\_maximo = np.unravel\_index(indices\_maximo, M.shape)

    # Valor máximo (pico de la montaña)

    M1 = M[fila\_maximo][columna\_maximo]

    valores\_maximos = np.append(valores\_maximos, M1)

    coordenada\_maximo = [fila\_maximo,columna\_maximo]

    coordenadas\_maximos = np.append(coordenadas\_maximos,[coordenada\_maximo],axis=0)

    # Plot 3D de la montaña

    fig, ax = plt.subplots(subplot\_kw={"projection": "3d"})

    ax.set\_xlabel('Y')

    ax.set\_ylabel('X')

    ax.set\_zlabel('Z')

    surf = ax.plot\_surface(X\_grid, Y\_grid, M, cmap=cm.coolwarm,linewidth=0, antialiased=False)

    cont += 1

# Plotear la grilla y puntos de datos

plt.figure(7)

plt.scatter(X\_grid, Y\_grid, s = 10 ,color='grey',label = "Nodos")

plt.scatter(l1,l2,color = 'green',label = "Datos")

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

for i in range(len(coordenadas\_maximos)):

    plt.scatter(X\_grid[0][int(coordenadas\_maximos[i][0])],Y\_grid[int(coordenadas\_maximos[i][1])][0],

                s = 100, c = colores[i], label = "Centro de cluster {}".format(i+1))

plt.xlabel('X')

plt.ylabel('Y')

plt.title('Grilla')

plt.grid(True)

plt.legend()

plt.show()

# Coeficiente delta

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

      "Coeficientes Delta = {}".format(valores\_maximos[0]/valores\_maximos[:]))

150 puntos:

import numpy as np

import matplotlib.pyplot as plt

import math

plt.close('all')

# Puntos a graficar

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

# Reordenamiento del arreglo

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

l1 = xk[0] # x

l2 = xk[1] # y

l3 = xk[2] # z

#Grilla

limites = [[min(l1), min(l2),min(l3)],

            [max(l1), max(l2),max(l3)]]

#Resolución de la grilla

resolucion = 10

num\_clusters = resolucion\*\*3

x = np.linspace(limites[0][0],limites[1][0],resolucion)

y = np.linspace(limites[0][1],limites[1][1],resolucion)

z = np.linspace(limites[0][2],limites[1][2],resolucion)

grid =  np.meshgrid(x,y,z)

X\_grid, Y\_grid, Z\_grid = grid

# Función de montaña

M = np.zeros((resolucion,resolucion,resolucion))

alpha = 0.379

beta = 0.5

# Condicion de paro

delta = 2

N = range(0,2) # Número de clústers

# Obtención de la primera montaña

for i in range(len(X\_grid)):

    for j in range(len(Y\_grid)):

        for k in range(len(Z\_grid)):

            for l in range(len(xk[0])):

                M[i][j][k] += math.e \*\* \

                    (-alpha\*math.sqrt( (X\_grid[0][i][0]-xk[0][l])\*\*2 \

                                      + (Y\_grid[j][0][0]-xk[1][l])\*\*2  \

                                      + (Z\_grid[0][0][k]-xk[2][l])\*\*2))

# Coordenadas del valor máximo (N1)

indices\_maximo = np.argmax(M)

x\_maximo, y\_maximo, z\_maximo = np.unravel\_index(indices\_maximo, M.shape)

# Valor máximo (pico de la montaña)

M1 = M[x\_maximo][y\_maximo][z\_maximo]

# Valores máximos de los picos de montaña

valores\_maximos = []

valores\_maximos = np.append(valores\_maximos, M1)

# Coordenadas del valor máximo dentro de la matriz M[10][10]

coordenadas\_maximos = np.zeros((1,3))

coordenada\_maximo = [x\_maximo,y\_maximo,z\_maximo]

coordenadas\_maximos[0] = coordenada\_maximo

# Cálculo de las siguientes montañas

cont = 0

while valores\_maximos[0]/valores\_maximos[cont] < delta: # Condición de paro

    if cont == 0:

        delta = 1.5

    for i in range(len(X\_grid)):

        for j in range(len(Y\_grid)):

            for k in range(len(Z\_grid)):

                acumulable = 0

                for l in range(len(coordenadas\_maximos)):

                    coordenadaX = int(coordenadas\_maximos[l][0])

                    coordenadaY = int(coordenadas\_maximos[l][1])

                    coordenadaZ = int(coordenadas\_maximos[l][2])

                    d = math.sqrt((X\_grid[0][coordenadaX][0]-X\_grid[0][i][0])\*\*2 +\

                                  (Y\_grid[coordenadaY][0][0]-Y\_grid[j][0][0])\*\*2 +\

                                  (Z\_grid[0][0][coordenadaZ]-Z\_grid[0][0][k])\*\*2)

                    acumulable += math.exp( -beta\*d)

                M[i][j][k] = M[i][j][k] - valores\_maximos[cont]\*acumulable

    indices\_maximo = np.argmax(M)

    # Coordenadas del valor máximo (N1)

    x\_maximo, y\_maximo, z\_maximo = np.unravel\_index(indices\_maximo, M.shape)

    # Valor máximo (pico de la montaña)

    M1 = M[x\_maximo][y\_maximo][z\_maximo]

    valores\_maximos = np.append(valores\_maximos, M1)

    coordenada\_maximo = [x\_maximo,y\_maximo,z\_maximo]

    coordenadas\_maximos = np.append(coordenadas\_maximos,[coordenada\_maximo],axis=0)

    cont += 1

# # Creamos la figura

fig = plt.figure()

# # Creamos el plano 3D

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

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

marcadores = ['d','+','P']

# Puntos de datos

ax1.scatter(xk[0], xk[1],xk[2], c=colores[2], marker=marcadores[1],label = 'Datos')

# Centros de cluster

coordenadas\_maximos = coordenadas\_maximos.T

ax1.scatter(coordenadas\_maximos[0],coordenadas\_maximos[1],coordenadas\_maximos[2],s = 100, c = colores[1], marker = marcadores[0],label='Centros de cluster')

plt.grid(True)

plt.legend()

plt.show()

# Coeficiente delta

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

      "Coeficientes Delta = {}".format(valores\_maximos[0]/valores\_maximos[:]))

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()