Practica 1

Regresión Lineal

Regresión lineal con una variable;

En la primera parte de la práctica se nos pide aplicar el método de regresión lineal sobre una distribución de datos. La finalidad es encontrar la recta que mejor se ajusta a los datos dados de tal manera que podremos realizar predicciones para valores diferentes de los mismos atributos usando el modelo obtenido.

Para la resolución de la regresión con una variable tenemos :

def resuelve\_problema\_regresion\_una\_variable():

    poblacion = carga\_csv('ex1data1.csv')

    X\_ = poblacion[:, :-1]

    Y\_ = poblacion[: , -1]

    X\_m = np.shape(X\_)[0]

    Y\_m = np.shape(X\_)[1]

    X\_ = np.hstack([np.ones([X\_m,1]),X\_])

    Thetas, Costes = descenso\_gradiente(X\_, Y\_, alpha = 0.01)

#Pinta la recta de regresión lineal que mejor se ajusta a los datos de entrenamiento

    pinta\_recta\_regresion\_lineal(X\_ , Y\_, 5, 22, Thetas)

    #Pinta las gráficas de contorno

    pinta\_costes(X\_, Y\_)

Donde descenso\_gradiente resuelve el problema dados el valor de tasa de aprendizaje y las matrices que representan a los valores de entrada (X-atributos independientes, Y-atributo dependiente).

def descenso\_gradiente(X, Y, alpha):

    m = len(X)

    #construimos matriz Z

    th0 = 0.0

    th1 = 0.0

    th\_n = 0.0

    Z = np.zeros(X.shape[1])

    Z\_ = np.zeros(X.shape[1] - 1)

    alpha\_m = (alpha/m)

    Thetas = np.array([Z]) #almacena los thetas que forman parte de la hipotesis h\_theta

    Costes = np.array([]) #almacena los costes obtenidos durante el descenso de gradiente

    for i in range(1500):

        #Calculo de Theta 0

        #Sumatorio para el calculo de Theta0

        sum1 = H\_Theta(X, Z) - Y

        sum1\_ = sum1.sum()

        th0 -= alpha\_m \* sum1\_

        #Calculo Theta 1, 2, 3 ... n

        #Sumatorio para el calculo de Thetan

        for k in range(X.shape[1] - 1):

            sum2 =  (H\_Theta(X, Z) - Y) \* X[:, k + 1]

            sum2\_ = sum2.sum()

            th\_n -= alpha\_m \* sum2\_ #vamos calculando cada uno de los thn

            Z\_[k] = th\_n #almacenamos los thn calculados en un vector provisional

        #Actualizamos los nuevos thetas del vector Z

        Z[0] = th0

        for p in range(X.shape[1]-1):

            Z[p+1] = Z\_[p]

        Thetas = np.append(Thetas, [Z], axis= 0)

        #funcion de costes

        J = funcion\_coste(X,Y, Z)

        Costes = np.append(Costes, [J], axis = 0)

    return Thetas, Costes

Como vemos el método nos devuelve un vector con todos los valores de theta y costes que hemos ido obteniendo a lo largo del descenso de gradiente independientemente del número de atributos o variables que tengamos. De esta manera este será el método que utilizaremos también para el caso de la parte 2.

Aquí también nos apoyamos en la función funcion\_coste que nos permite calcular el coste correspondiente para los valores de entrenamiento con las thetas calculados para cada iteración (en este caso de manera vectorial). De esta manera podemos ver como evoluciona el coste en cada una de las iteraciones durante el descenso de gradiente.

def funcion\_coste(X, Y, Theta): #funcion de costes vectorizada

    H = H\_Theta(X,Theta)

    Aux = (H-Y)\*\*2

    sumatory = Aux.sum()/(2 \* len(X))

    return sumatory

Podemos ver como para resolver el problema utilizamos una serie de funciones auxiliares que en este caso resuelven la hipótesis htheta, en nuestro caso también de manera vectorial.

def hth(x, th): #Hipótesis modelo lineal

    return th[0] + th[1] \* x

def H\_Theta(X, Z): #Hipótesis del mocelo lineal vectorizada

    return np.dot(X, Z)

carga\_csv nos permite volcar los datos desde un archivo .cvs en una matriz.

def carga\_csv(file\_name):

    "Carga fichero csv especificado y lo devuelve en un array de numpy"

    valores = read\_csv(file\_name, header=None).values

    return valores.astype(float) #parseamos a float (suponemos que siempre trabajaremos con float)

Para representar gráficamente los resultados obtenidos usamos las funciones pinta\_recta\_regresion\_lineal y pinta\_costes.

def pinta\_recta\_regresion\_lineal(X ,Y ,x1 , x2, thetas):

    plt.scatter(np.array(X[:,1]), Y, alpha= 0.5)

    plt.plot([x1, x2], [hth(x1,thetas[-1]) , hth(x2, thetas[-1])], color = "red")

    plt.show()

def pinta\_costes(X, Y, num\_div = 100):

    x\_theta0 = np.linspace(-10, 10 ,num\_div)

    y\_theta1 = np.linspace(-1, 4, num\_div)

    xx\_thetas0, yy\_thetas1 = np.meshgrid(x\_theta0, y\_theta1) #junta las matrices que servirán de ejes para nuestra representación de lls datos

    dim\_0 = xx\_thetas0.shape[0]

    dim\_1 = xx\_thetas0.shape[1]

    J = np.zeros((dim\_0, dim\_1)) #contiene la matriz J de costes asociados a cada par de thetas introducidos, cada coste se almacena en una matriz coincidiendo con el valor de fila y columna del que se extrageron sus parametros para ser calculado

    for i in range(dim\_0):

        for j in range(dim\_1):

            Z = np.array([xx\_thetas0[i,j], yy\_thetas1[i,j]])

            J[i,j] = funcion\_coste(X, Y, Z) #vamos calvulado los costes para los diferentes valres de theta0 y theta1 almacenados en Z y teniendo en cuenta los valores de X e Y necesarios para el calculo del coste

    #dibujamos la curva de costes

    fig = plt.figure()

    ax = Axes3D(fig)

    surf = ax.plot\_surface(xx\_thetas0, yy\_thetas1, J, cmap= cm.coolwarm, linewidths= 0, antialiaseds = False)

    fig.colorbar(surf, shrink = 0.5, aspect = 5)

    plt.show()

    #mapa de nivel de costes

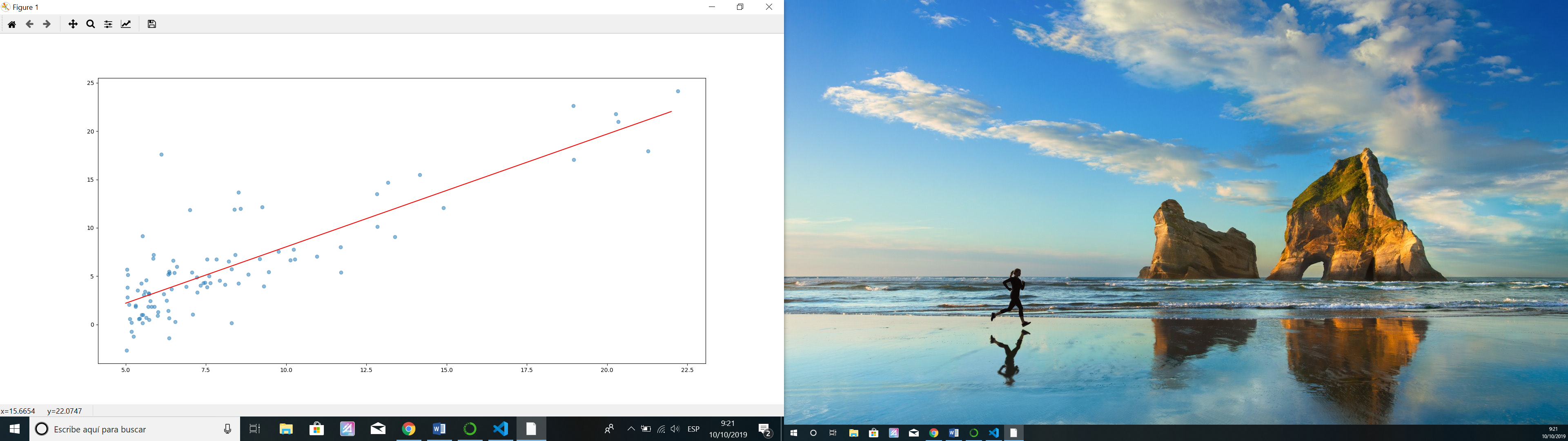
    plt.contour(xx\_thetas0, yy\_thetas1, J, np.logspace(-2, 3, 20), colors = "blue")

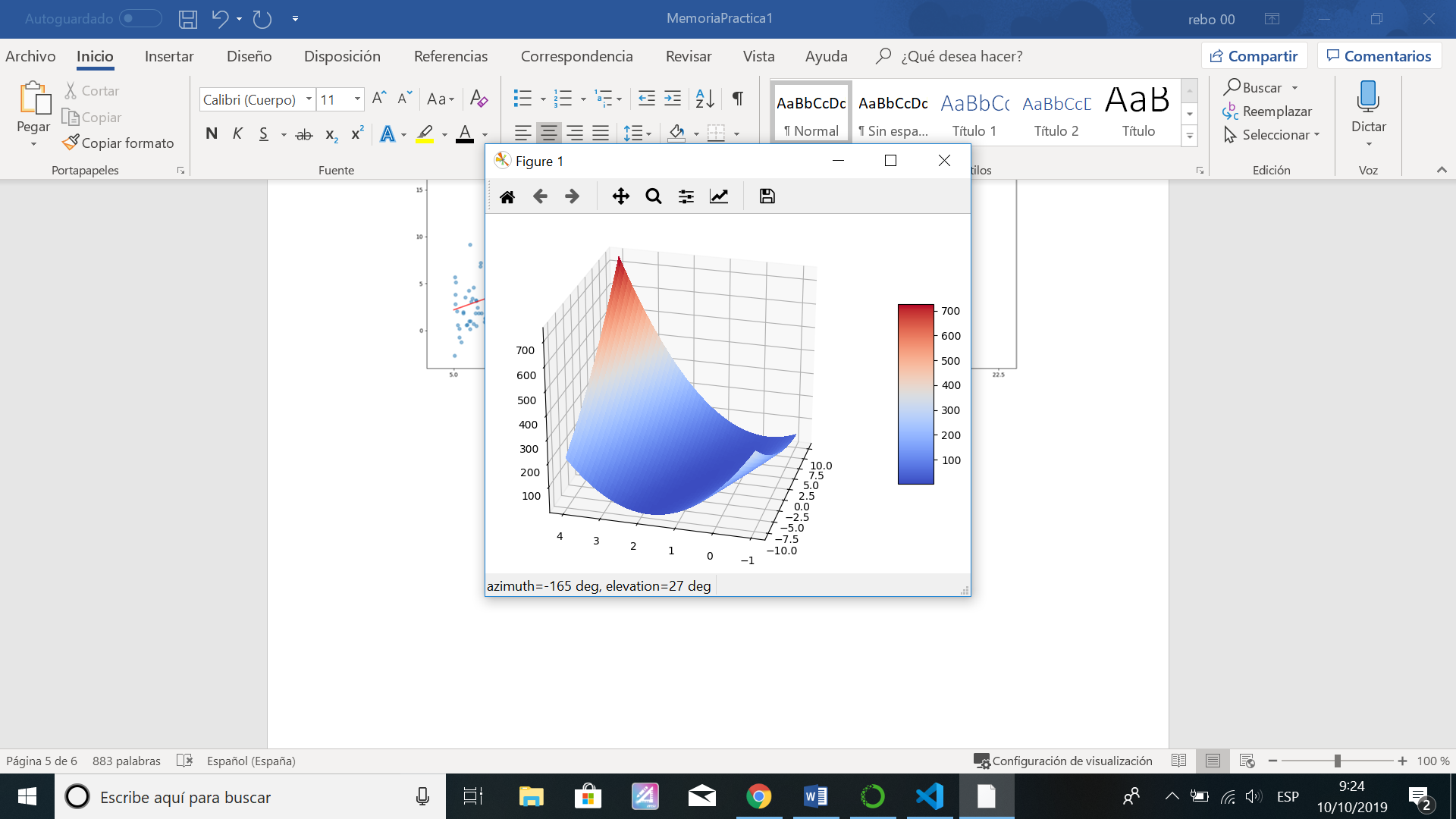
    plt.show()

Ambos bloques de código nos pintan en pantalla la recta de regresión lineal que mejor se ajusta a los valores de entrenamiento y las gráficas de contorno respectivamente.

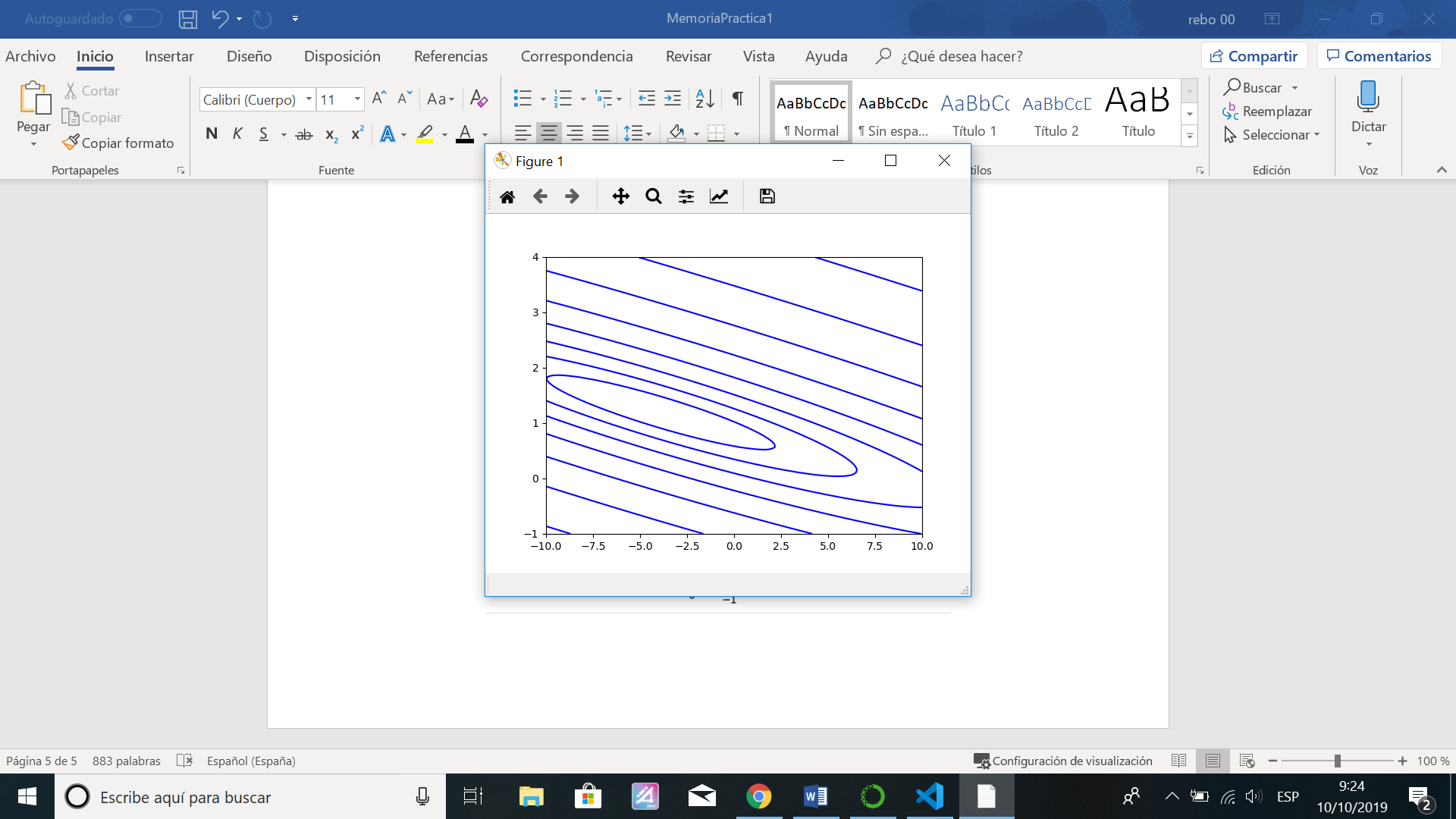
Para nuestra población de valores de entrenamiento y un valor de tasa de aprendizaje de Alpha = 0.01 los resultados obtenidos son acordes lo esperado: El valor del coste disminuye a medida que avanza el descenso de gradiente.

A medida que avanzamos en el descenso de gradiente vamos obteniendo las diferentes thetas que hacen que la hipótesis cada vez se aproxime más a la recta que mejor se ajusta a los datos de entrenamiento y que en este caso se corresponden con los valores obtenidos en la última iteración.

Recta de regresión lineal



Coste,Theta1,Theta0



Theta1, Theta0

Parte 2 regresión con varias variables:

La idea en este parte de la práctica es la misma, encontrar una serie de valores de Theta para la hipótesis htheta que nos permita convertirla en un medio para predecir valores de una variable dependiente en función de una serie de valores o parámetros que funcionarán como nuestros atributos o variables independientes.

En este caso el método que resuelve nuestro problema es el siguiente :

def resuelve\_problema\_regresion\_varias\_variables():

    poblacion = carga\_csv('ex1data2.csv')

    X = poblacion[: , :-1]

    Y = poblacion[: , -1]

    X\_normalizada, mu, sigma = normaliza(X)

    X = np.hstack([np.ones([np.shape(X)[0], 1]), X])

    X\_shape\_1 = np.shape(X\_normalizada)[0]

    X\_normalizada = np.hstack([np.ones([X\_shape\_1, 1]), X\_normalizada]) #le añadimos la columna de unos a la matriz ya normalizada

    Thetas, Costes = descenso\_gradiente(X\_normalizada, Y, 0.0025) #los valores de theta aquí son los obtenidos normalizando la matriz, esto es, necesitamos "desnormalizarlos"

    Thetas\_normal\_Ecuation = normalEcuation(X, Y)

En este caso y debido a la naturaleza de los valores o atributos de entrenamiento nos vemos obligados a normalizar las matriz que contiene los valores que funcionarán como nuestros atributos independientes (X).

Para ello nos ayudamos del método normaliza que nos da como resultado una matriz normalizada a partir de otra de entrada y los valores de media y desviación estándar utilizados en el proceso.

def normaliza(X):

    X\_normalizada = np.zeros((X.shape[0], X.shape[1]))

    mu = np.zeros(X.shape[1])

    sigma = np.zeros(X.shape[1])

    np.median(X, axis = 0, out = mu)

    np.std(X, axis = 0, out = sigma)

    for i in range(X.shape[0]):

        for j in range(X.shape[1]):

            X\_normalizada[i,j] = (X[i,j] - mu[j])/sigma[j]

    return X\_normalizada, mu, sigma