Practica 1

Regresión Lineal

Aaron Reboredo Vázquez, Pablo Martín García

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 en
trenamiento
    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 hi
potesis h_theta
    Costes = np.array([]) #almacena los costes obtenidos durante el desce
nso 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 pro
visional


        #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 h_{theta}, 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 siempr
e 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()
```

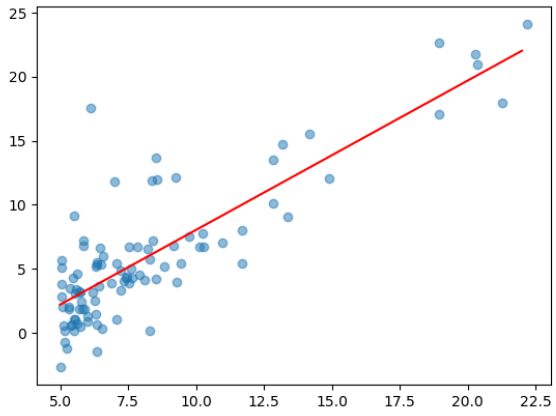
```
def pinta_costes(X, Y, num_div = 100):

    step = 0.1

    x_theta0 = np.arange(-10, 10 ,step)
    y_theta1 = np.arange(-1, 4, step)

    xx_thetas0, yy_thetas1 = np.meshgrid(x_theta0, y_theta1) #junta las m
atrices 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 asociado
s 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 pa
rametros 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 p
ara 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()
```

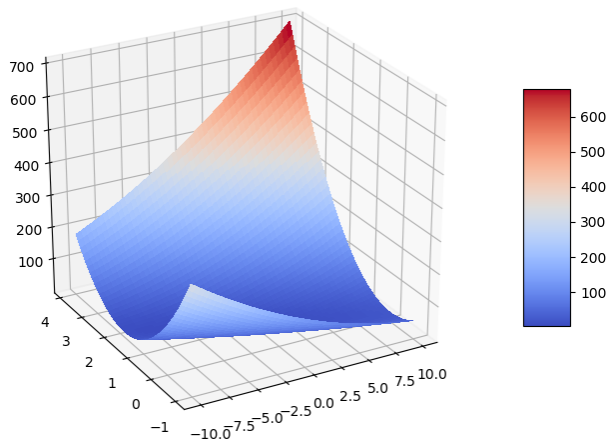
Ambos bloques de código nos pintan en pantalla la recta de regresión lineal que mejor se ajusta a los valores de entrenamiento y las gráficas de contorno respectivamente.

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.

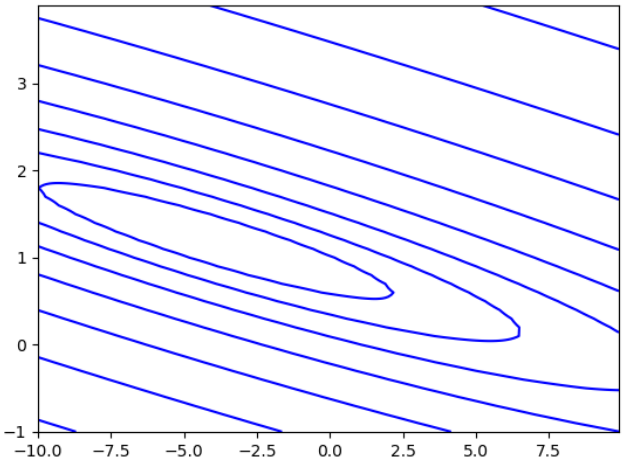
A medida que avanzamos en el descenso de gradiente vamos obteniendo las diferentes thetas que hacen que la hipótesis cada vez se aproxime más a la recta que mejor se ajusta a los datos de entrenamiento y que en este caso se corresponden con los valores obtenidos en la última iteración.



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 h_{theta} 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, 0.0022) #los va
lores de theta aquí son los obtenidos normalizando la matriz, esto es, ne
cesitamos "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.mean(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
```

Anteriormente, ya para la primera parte de la práctica usamos el método `descenso_gradiente`. En esta ocasión y como el método estaba generalizado para el cálculo de la regresión lineal independientemente del número de variables lo volveremos a utilizar.

Así mismo y para comprobar los resultados se nos plantea utilizar la ecuación Normal para el cálculo de las Thetas que mejor se aproximan a nuestra regresión con el fin de comprobar que el método de descenso de gradiente funciona correctamente.

Dicha ecuación normal viene dada por el siguiente bloque de código y nos devuelve un array con los n Thetas que junto con la hipótesis predicen los valores de la variable dependiente en función de los parámetros de entrada. A esta función se le pasan como parámetros de entrada la matriz de atributos dependientes e independientes en este caso sin normalizar.

```
def normalEcuation(X,Y): #nos da los valores de theta que mejor se ajusta
n a nuestra regresión lineal

    #theta = (XT * X)**-1 * XT * Y

    XT = np.transpose(X)

    XT__X = np.matmul(XT, X)

    XT__X_Inv = np.linalg.pinv(XT__X, rcond = 10**(-15))

    XT__X_Inv__XT = np.matmul(XT__X_Inv, XT)

    thetas = np.matmul(XT__X_Inv__XT, Y)

    return thetas
```

Como hemos mencionado anteriormente podemos comprobar el grado de eficacia en la predicción del gradiente descendiente comparando los resultaos obtenidos de utilizar las thetas obtenidas anteriormente (descenso de gradiente y ecuación normal)

En el ejemplo de la práctica para hacer la prueba utilizaremos como entrada de 1650 y 3 habitaciones.

```
    #testeo_predicción_de_precios_descensoDeGradiente-VS-EcuacionNormal

    #Normalizamos la entrada para el caso de gradiente descendiente con l
os valores de sigma y mu que obteníamos de normalizar la matriz de entrad
a original
    precio_test_normalizado = (1650 - mu[0])/sigma[0]
    habitaciones_test_normalizado = (3 - mu[1])/sigma[1]

    shape_thetas = np.shape(Thetas)[0]-1

    prediccion_normal_ecuation = H_Theta([[1.0, 1650, 3]] ,Thetas_normal_
Ecuation)
    prediccion_gradiente_descendiente = H_Theta([[1.0, precio_test_normal
izado, habitaciones_test_normalizado]], Thetas[shape_thetas])

    print("Prediccion ecuación normal " , prediccion_normal_ecuation)
    print("Prediccion gradiente descendiente", prediccion_gradiente_desce
ndiente)
```

Valores obtenidos :

```
Prediccion ecuación normal  [293081.46433499]
Prediccion gradiente descendiente [293326.46787842]
```

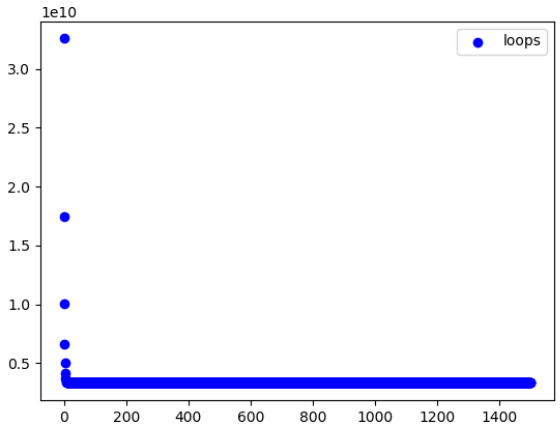
Dependiendo del valor de Alpha o tasa de aprendizaje la predicción por gradiente descendiente se aproxima más o menos. En nuestro caso hemos encontrado un Alpha (= 0.0022) que hace que el cálculo se ajuste bastante bien al valor obtenido en el caso de utilizar la ecuación normal.

En el enunciado se nos propone visualizar en pantalla el avance del valor del coste durante el método de descenso de gradiente en función de los valores de Alpha utilizados.

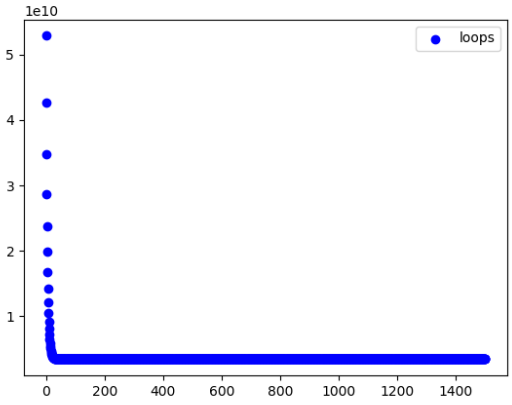
La función utilizada es la que hemos llamado `draw_function :`

```
def draw_function(costes, iteraciones):

    x_axis = np.linspace(0, iteraciones, iteraciones)

    plt.figure()
    plt.scatter(x_axis, costes, c = 'blue', label = 'loops' )
    plt.legend()
    plt.show()
```

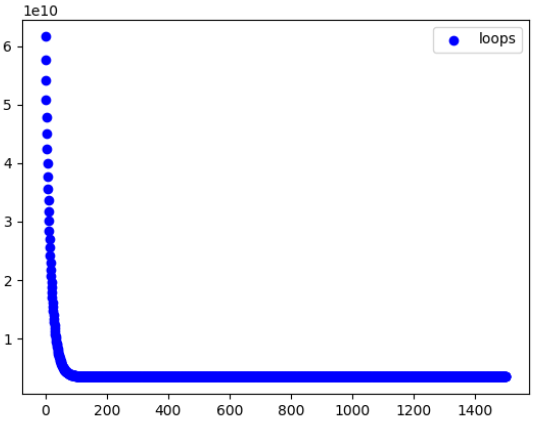
Alpha = 0.3



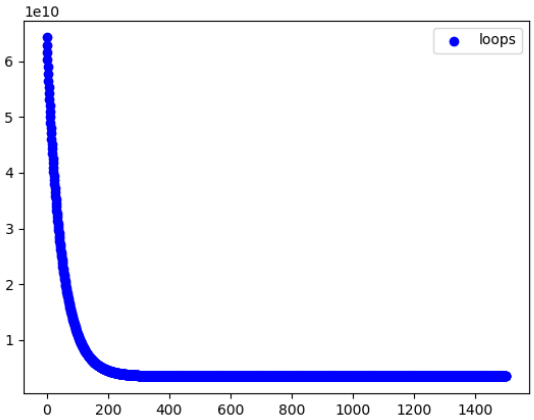
Alpha = 0.1



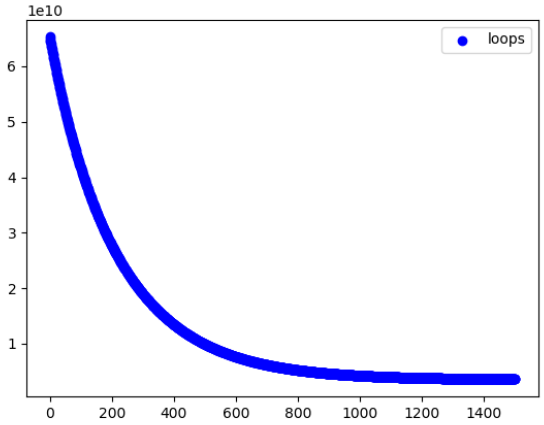
Alpha = 0.03



Alpha = 0.01



Alpha = 0.0022



Con estos resultados podemos ver como los saltos en el valor del coste son mayores para el caso de los valores de tasa de aprendizaje más altos, que es el resultado que cabría esperar.

A medida que nos acercamos al valor de coste que representa el mínimo absoluto el valor de coste comienza a estabilizarse y se puede decir que “tiende” al valor resultado o el coste correspondiente a los cálculos hechos con los thetas que mejor representan nuestra hipótesis.

Código completo :

```
import numpy as np
import matplotlib.pyplot as plt
from pandas.io.parsers import read_csv
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from mpl_toolkits.mplot3d import Axes3D

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 siempr
e trabajaremos con float)

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)

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):

    step = 0.1

    x_theta0 = np.arange(-10, 10 ,step)
    y_theta1 = np.arange(-1, 4, step)

    xx_thetas0, yy_thetas1 = np.meshgrid(x_theta0, y_theta1) #junta las m
atrices 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 asociado
s 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 pa
rametros 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 p
ara 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()

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.mean(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

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


def normalEcuation(X,Y): #nos da los valores de theta que mejor se ajusta
n a nuestra regresión lineal

    #theta = (XT * X)**-1 * XT * Y

    XT = np.transpose(X)
```

```
    XT__X = np.matmul(XT, X)

    XT__X_Inv = np.linalg.pinv(XT__X, rcond = 10**(-15))

    XT__X_Inv__XT = np.matmul(XT__X_Inv, XT)

    thetas = np.matmul(XT__X_Inv__XT, Y)

    return thetas

def draw_function(costes, iteraciones):

    x_axis = np.linspace(0, iteraciones, iteraciones)

    plt.figure()
    plt.scatter(x_axis, costes, c = 'blue', label = 'loops' )
    plt.legend()
    plt.show()

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 hi
potesis h_theta
    Costes = np.array([]) #almacena los costes obtenidos durante el desce
nso 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 pro
visional


        #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

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 d
e entrenamiento
    pinta_recta_regresion_lineal(X_ , Y_, 5, 22, Thetas)

    #Pinta las gráficas de contorno
    pinta_costes(X_, Y_)


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, 0.01) #los valo
res de theta aquí son los obtenidos normalizando la matriz, esto es, nece
sitamos "desnormalizarlos"
    Thetas_normal_Ecuation = normalEcuation(X, Y)

    #Representación de como avanza la función de costes en funcion
    draw_function(Costes, 1500)


    #testeo_predicción_de_precios_descensoDeGradiente-VS-EcuacionNormal

    #Normalizamos la entrada para el caso de gradiente descendiente con l
os valores de sigma y mu que obteníamos de normalizar la matriz de entrad
a original
    precio_test_normalizado = (1650 - mu[0])/sigma[0]
    habitaciones_test_normalizado = (3 - mu[1])/sigma[1]

    shape_thetas = np.shape(Thetas)[0]-1

    prediccion_normal_ecuation = H_Theta([[1.0, 1650, 3]] ,Thetas_normal_
Ecuation)
    prediccion_gradiente_descendiente = H_Theta([[1.0, precio_test_normal
izado, habitaciones_test_normalizado]], Thetas[shape_thetas])

    print("Prediccion ecuación normal " , prediccion_normal_ecuation)
    print("Prediccion gradiente descendiente", prediccion_gradiente_desce
ndiente)




#resuelve_problema_regresion_una_variable()
#resuelve_problema_regresion_varias_variables()
```