# Practica 1

## Regresión lineal

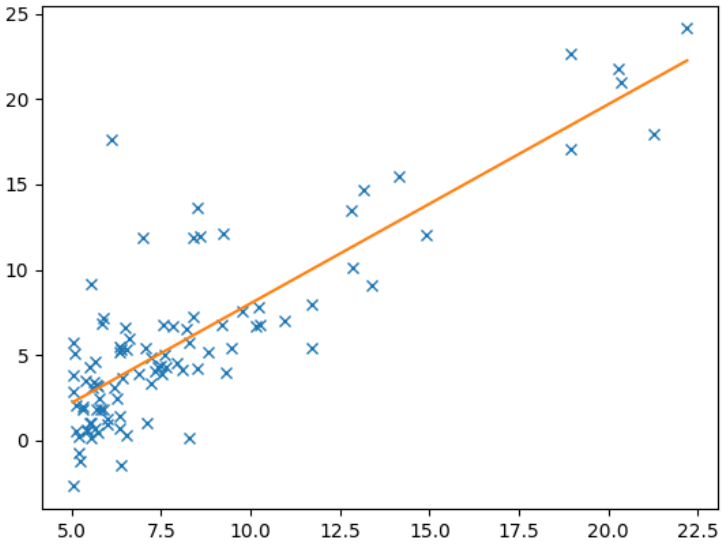
Antonio Rodriguez Hurtado y Miguel Ferreras Chumillas

## Una única variable

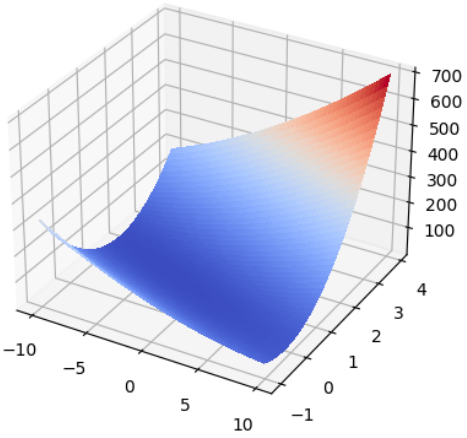
```
#   Librerias
import numpy as np
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
from matplotlib import cm

#   Declaracion de las variables
theta = [0,0]
alpha = 0.01
iteraciones = 1500
rango_th0 = [-10,10]
rango_th1 = [-1,4]

#   Convertimos el .csv en un array de numpy
valores = read_csv("ex1data1.csv", header=None).to_numpy()

#   Separamos las coordenadas en dos arrays, X e Y
X = valores[:,0]
Y = valores[:,1]

#   Obtenemos el numero de elementos m
m = len(X)

#   Funcion de hipotesis
def fun_hip(x):
    return theta[0] + (theta[1]*x)

#   Funcion de coste
def fun_coste(th):
    sum = 0;
    for i in range(m):
        sum += ((th[0] + th[1] * X[i]) - Y[i])**2
    return sum /(m*2)

#   Funcion para obtener los datos de la grafica 3D
def make_data():
    step = 0.1
    th0 = np.arange(rango_th0[0], rango_th0[1], step)
    th1 = np.arange(rango_th1[0], rango_th1[1], step)
    th0, th1 = np.meshgrid(th0, th1)
    coste = np.empty_like(th0)
    for ix, iy in np.ndindex(th0.shape):
        coste[ix, iy] = fun_coste([th0[ix, iy], th1[ix, iy]])
    return [th0, th1, coste]

#   Metodo de descenso de gradiente
```

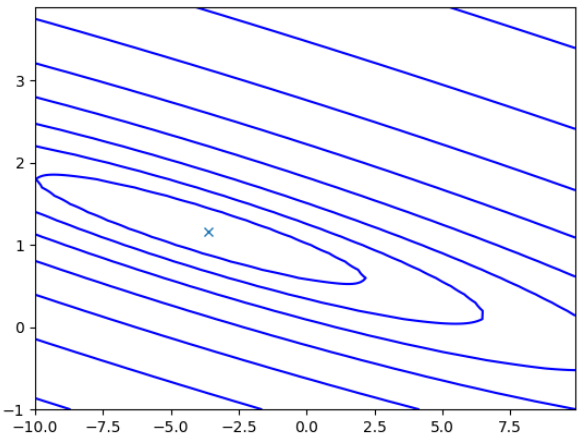
```
for i in range(iteraciones):
    sum0 = sum1 = 0
    for i in range(m):
        sum0 += fun_hip(X[i]) - Y[i]
        sum1 += (fun_hip(X[i]) - Y[i]) * X[i]
    theta[0] -= (alpha / m) * sum0
    theta[1] -= (alpha / m) * sum1

#   Dibujamos la grafica 2D
# Dibujamos los puntos [x,y] del .csv
plt.plot(X,Y, "x")
# Dibujamos la recta de la funcion de hipotesis
min_x = min(X)
max_x = max(X)
min_y = fun_hip(min_x)
max_y = fun_hip(max_x)
plt.plot([min_x, max_x], [min_y, max_y])
plt.savefig("resultado.pdf")

#   Dibujamos la grafica 3D
eje_x, eje_y, eje_z = make_data()
cos = plt.figure()
ejes =  cos.gca(projection = "3d")
surface = ejes.plot_surface(eje_x, eje_y, eje_z, cmap=cm.coolwarm, linewidth=0, antialiased=False)
plt.savefig("coste.pdf")

#   Dibujamos la grafica de contorno
con = plt.figure()
plt.plot(theta[0], theta[1], "x")
plt.contour(eje_x, eje_y, eje_z,np.logspace(-2, 3, 20), colors="blue")
plt.savefig("contorno.pdf")

plt.show()
```

El código de esta parte da como resultado tanto la gráfica de la regresión linear como la de la función de coste



A continuación se muestran las gráficas de la función de coste proporcionadas por el texto




## Parte 2: Multivariable

En esta parte se analizan una lista de datos sobre casas donde aparece tanto el numero de metros cuadrados que poseen, como el numero de habitaciones y su precio.

El objetivo de esta parte es lograr una predicción acertada del precio de una casa de ejemplo a raiz del numero de metros y el numero de habitaciones de la misma.

En primer lugar se obtienen los datos y se definen las funciones que se necesitan

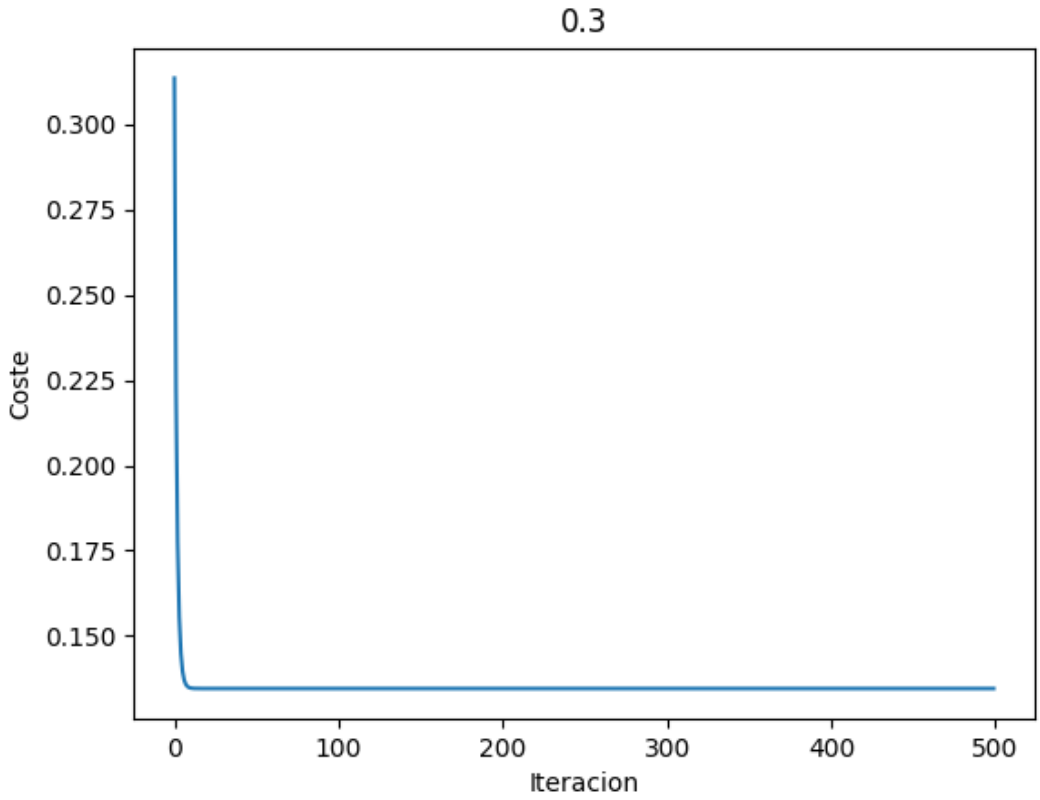
```
# Librerias
import numpy as np
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
```

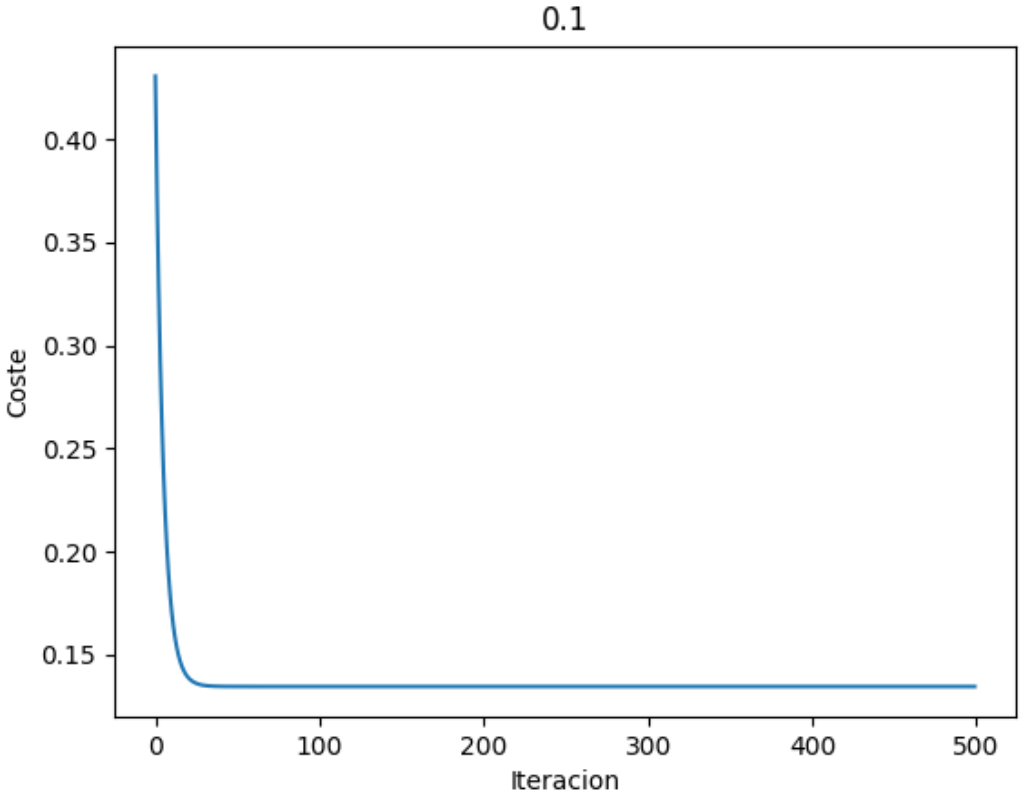
```
from matplotlib import cm


#  Convertimos el .csv en un array de numpy
valores = read_csv("ex1data2.csv", header=None).to_numpy().astype(float)

#  Separamos las coordenadas en dos arrays, X e Y
X = valores[:, :-1]
Y = valores[:, -1]

#  Obtenemos las dimensiones de la matriz X (n x m)
filas = np.shape(X)[0]
columnas = np.shape(X)[1]


def normalize(X):
    '''Normaliza los datos de la matriz X
    restandoles su media y dividiendolos por la desviacion estandar'''
    mu = np.mean(X, axis=0)
    X_mean = X - mu
    sigma = np.std(X, axis=0)
    X_norm = X_mean / sigma
    return X_norm, mu, sigma


# Normalizamos los datos
theta = [0] * (columnas + 1)

valores_norm, mu, sigma = normalize(valores)
X_norm = valores_norm[:, :-1]
Y_norm = valores_norm[:, -1]


# Añadimos una columna de 1s a X
X_norm = np.hstack([np.ones([filas,1]),X_norm])

#   Funcion de coste
def fun_coste(X,Y,theta):
    hip = np.dot(X, theta)
    aux = (hip - Y) **2
    return aux.sum() / (2 * filas)

# Metodo de descenso de gradiente vectorizado
def gradiente(X, Y, theta, alpha):
    th = theta
    for i in range(columnas):
        aux = ((np.dot(X, theta) - Y) * X[:, i])
        th[i] -= ((alpha/filas) * aux.sum())
    return th, fun_coste(X, Y, th)

# Ecuacion normal
def ecuacion_normal(X, Y):
    X= np.hstack([np.ones([filas,1]),X])
    th = np.linalg.inv(np.transpose(X).dot(X)).dot(np.transpose(X)).dot(Y)
    return th
```

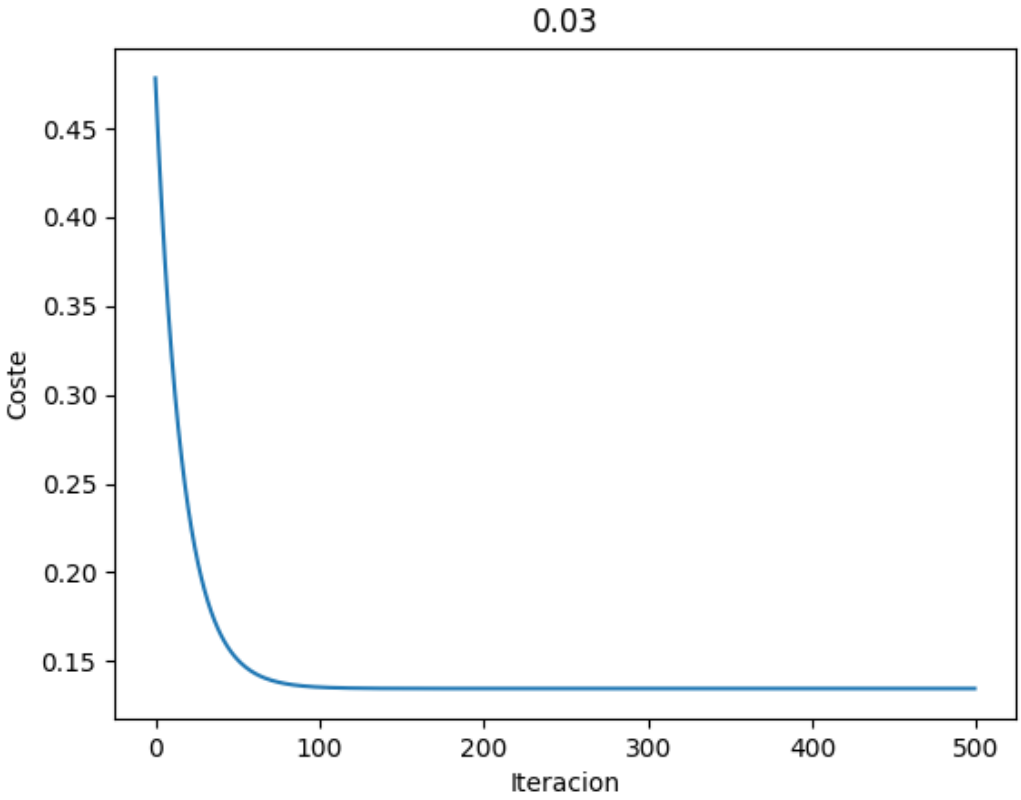
Se hacen pruebas con diferentes valores de alpha para ver los cambios en la funcion de coste

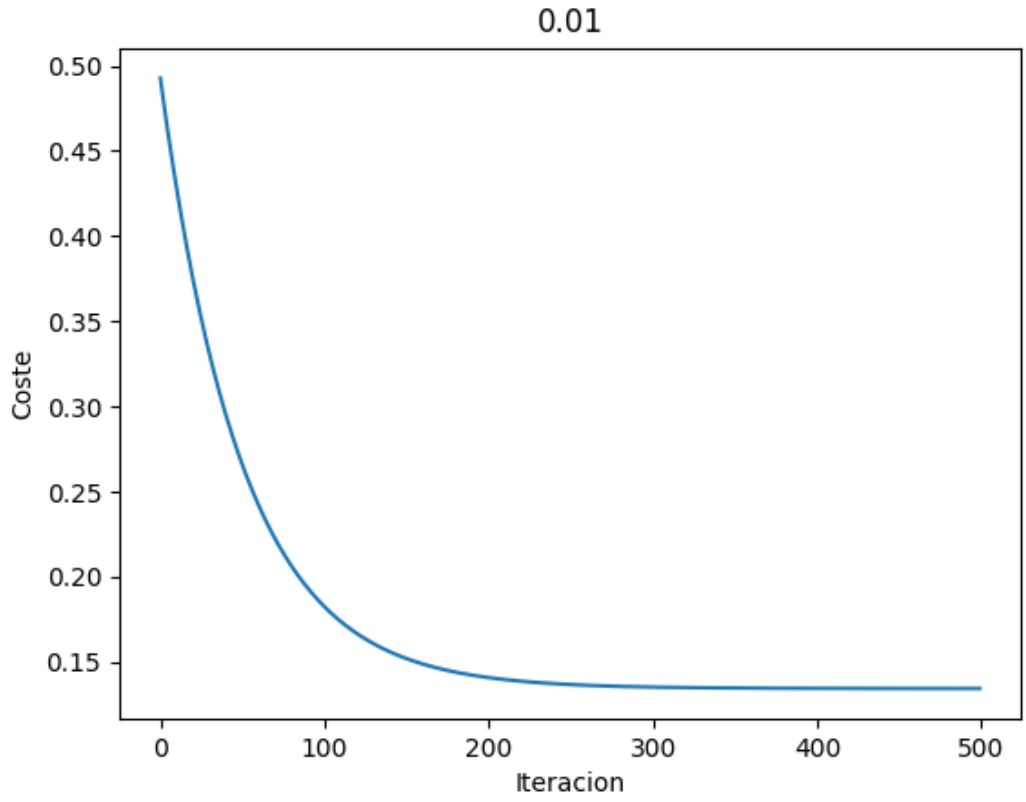
```
# Pruebas con alpha
for alpha in [0.3, 0.1, 0.03, 0.01]:
    theta = np.zeros(columnas + 1)
    costes = []
    plt.figure(1)
    plt.title(alpha)
    plt.xlabel('Iteracion')
    plt.ylabel('Coste')

    for i in range(500):
        Theta, cost = gradiente(X_norm, Y_norm, theta, alpha)
        costes.append(cost)

    plt.plot(costes)
    plt.show()
```

Los resultados de este fragmento son las siguientes gráficas:




0.1
0.40
0.35
0.30
0.25
0.20
0.15
Coste
0
100
200
300
400
500
Iteracion


0.03
0.45
0.40
0.35
0.30
0.25
0.20
0.15
Coste
0
100
200
300
400
500
Iteracion




Se puede ver claramente que a menor alpha mas tarde alcanza el coste minimo, esto se debe a que los cambios en theta son mas pequeños a menor alpha y por tanto necesita mas iteraciones para llegar.

Por último comprobamos que las funciones son correctas con la prediccion comparandola a la ecuacion normal

```
# Comprobacion de las funciones
# Calculamos las thetas
thg = np.zeros(columnas + 1)
alpha = 0.1

for i in range(3000):
    thg, cost = gradiente(X_norm, Y_norm, thg, alpha)

the = ecuacion_normal(X, Y)

# Creamos datos de prueba y los normalizamos
casa = np.array([1650,3])
casa_norm = (casa - mu[:-1])
casa_norm = casa_norm / sigma[:-1]

# Prediccion con descenso del gradiente
gradient_predict = np.matmul(np.append(np.array([1]),casa_norm), thg) * sigma[-1]
gradient_predict = gradient_predict + mu[-1]

# Prediccion con ecuación normal
normal_predict = np.matmul(np.append(np.array([1]),casa), the)
```

```
print('Precio con', casa[0], 'pies cuadrados y', casa[1], 'habitaciones:')
print('Resultado con descenso del gradiente:', gradient_predict)
print('Resultado con ecuación normal:', normal_predict)
```

Resultado:

```
Precio con 1650 pies cuadrados y 3 habitaciones:
Resultado con descenso del gradiente: 293237.21718712733
Resultado con ecuación normal: 293081.4643348959
```