Practica 1

Aurora García Jover
Anass Carreau Allagui

Parte 1: para la primera parte implementaremos el descenso de gradiente para un x de una sola variable, primero leeremos los datos del archivo, y luego aplicaremos el descenso de gradiente sobre 1500 iteraciones utilizando un alpha de 0,01
-leemos de archivo:

```
#metodo que se encrga de la carga de los datos desde el fichero
def carga_csv(file_name):
    """carga el fichero csv especificado y lo
 devuelve en un array de numpy
    """
    valores = read_csv(file_name, header=None).to_numpy()
    # suponemos que siempre trabajaremos con float
    return valores.astype(float)
```

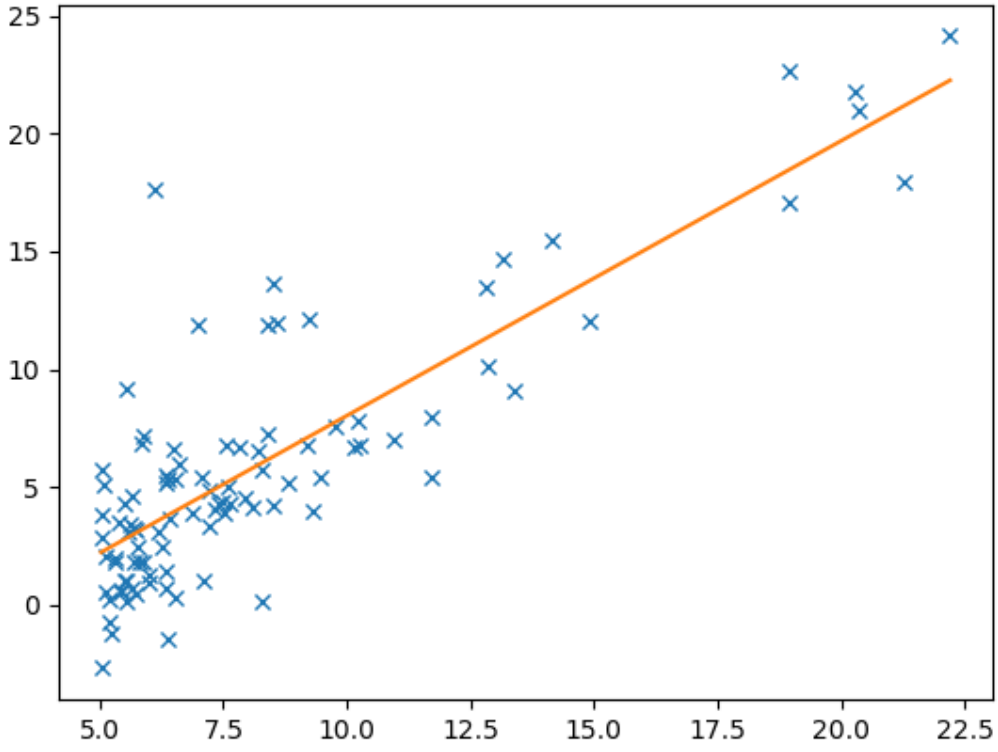
-Aplicamos el descenso de gradiente:

```
datos=carga_csv("ex1data1.csv")
alpha=0.01
ite=1500
X=datos[:,0]
Y=datos[:,1]
m=len(X)
theta0 = 0
theta1 = 0
```

```
for _ in range(ite) :
     sum0=0
     sum1=0
     for i in range(m):
         sum0+=theta0+theta1 * X[i] - Y[i]
         sum1+=(theta0+theta1 * X[i] - Y[i])*X[i]
     theta0=theta0- (alpha/m) *sum0
     theta1=theta1- (alpha/m) *sum1
plt.plot(X, Y, "x")
min_x = min(X)
max_x = max(X)
min_y = theta0 + theta1 * min_x
max_y = theta0 + theta1 * max_x
```

```
plt.plot([min_x, max_x], [min_y, max_y])
plt.savefig("resultado.png")
```

-conseguimos la siguiente grafica:



Después pasamos a dibujar las graficas:

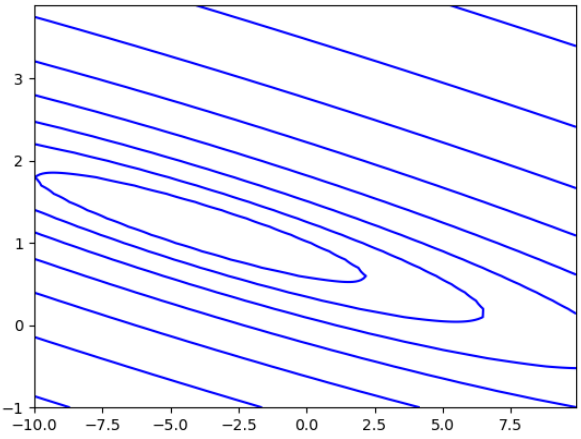
```
#metodo que se encarga d calcular el coste
def coste(X,Y,theta):
    m=len(X)
    sum=0
    #for i in range(m):
    #   sum+=np.square(theta[0]+theta[1]*X[i] - Y[i])
    #esto sustituiria el for de arriba
    aux = np.square(theta[0]+theta[1]*X - Y)
    sum = np.sum(aux)
    return sum/(2*m)

def make_data ( t0_range , t1_range , X , Y ) :
```

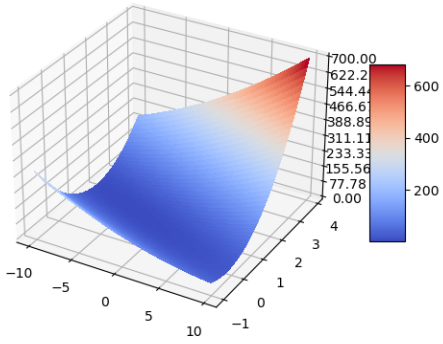
```
    """ Genera las matrices Theta0 , Theta1 , Coste para generar un
plot en 3D
        del coste para valores de theta_0 en el intervalo t0_range y
        valores de theta_1 en el intervalor t1_range"""

    step = 0.1
    Theta0 = np.arange ( t0_range[0], t0_range[1], step )
    Theta1 = np.arange ( t1_range[0], t1_range[1], step )
    Theta0 , Theta1 = np.meshgrid (Theta0 , Theta1)
    Coste = np.empty_like (Theta0)
    for ix , iy in  np.ndindex(Theta0.shape):
        Coste [ ix , iy ] = coste(X, Y ,[Theta0 [ix ,iy ] , Theta1[ix
,iy]])
    return [ Theta0 , Theta1 , Coste ]
```

```
datos=carga_csv("ex1data1.csv")
alpha=0.01
ite=1500
X=datos[:,0]
Y=datos[:,1]
m=len(X)
theta0 = 0
theta1 = 0

data=make_data( [-10,10], [-1,4],X,Y )

plt.contour(data[0], data[1], data[2],
 np.logspace(-2, 3, 20), colors='blue')
plt.savefig("grafica2D.png")

#para pintar una figura 3D
fig = plt.figure()
ax = fig.gca(projection='3d')

surf=ax.plot_surface(data[0],data[1],data[2],cmap=cm.coolwarm,linewidth
=0,antialiased=False)



ax.set_zlim(0,700)
ax.zaxis.set_major_locator(LinearLocator(10))
```

```
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))


fig.colorbar(surf, shrink=0.5, aspect=5)
plt.savefig("grafica3D.png")
plt.show()
```

y con ello conseguimos estas dos graficas:





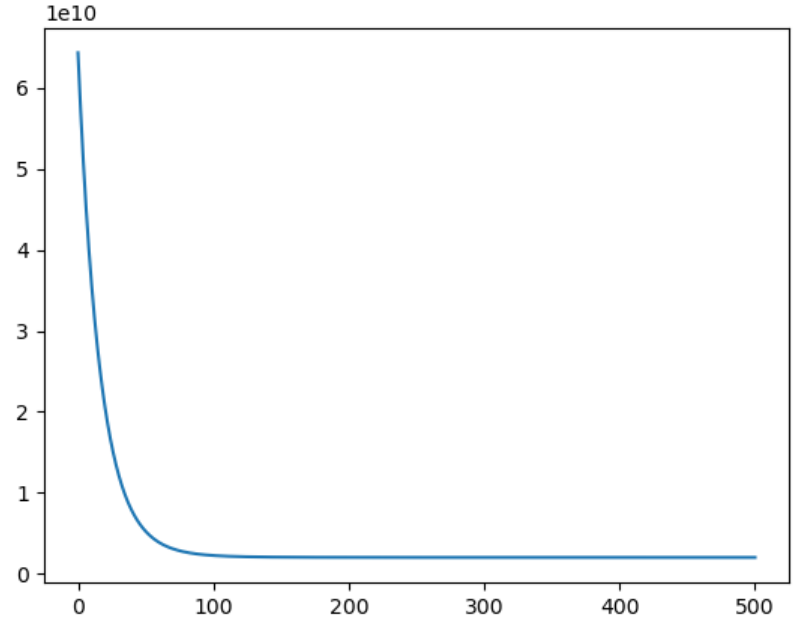
Parte 2: para la segunda parte para varios valores hemos implementado un método descenso gradiente que usa los datos normalizados y el método gradiente que aparece en las transparencias con un alpha (0.01,0.02,...) y 1500 iteraciones

Para normalizar usamos la formula a partir de la segunda columna .
 (x-media)/desviacion
Usamos mean y std para la media y desviación estándar.

```
def normalizacion(X):
   m = np.shape(X)[0]
   n = np.shape(X)[1]
   medias = np.mean(X,0,dtype=np.float64)
   desviacion = np.std(X,0,dtype=np.float64)
   newMatriz = np.copy(X)
   for i in np.arange(n-1):
     newMatriz[:,i+1] =  (X[:,i+1]-medias[i+1]) / desviacion[i+1]
   return newMatriz,medias,desviacion
```

```
def descenso_gradiente(X,Y,alpha):
  matriz_norm, medias, desviacion = normalizacion(X)
  ite = 1500
  n = np.shape(X)[1]
  Costes = np.zeros(ite)
  Thetas = np.zeros(n)
  for i in range(ite):
    Thetas = gradiente(matriz_norm,Y,Thetas,alpha)
    Costes[i] = coste(matriz_norm,Y,Thetas)
  plt.figure()
  x = np.linspace(0, 500, ite, endpoint = True)
  x = np.reshape(x, (ite, 1))
  plt.plot(x, Costes)
  plt.savefig("coste.png")
  plt.show()
  return Thetas,Costes,medias,desviacion
```

```
def coste(X, Y, Theta):
    H = np.dot(X, Theta)
    Aux = (H - Y) ** 2
    return Aux.sum() / (2 * len(X))
```



conseguimos esta gráfica de coste

Para la ecuación normal hemos hecho un metodo con funciones numpy
θ = (XT X) ^−1 X^T y

```
def Ecuacion_normal(X,Y):
  X_tras=np.transpose(X)
  inv_matr_XtX=linalg.pinv(np.matmul(X_tras,X))
  return np.matmul(np.matmul(inv_matr_XtX,X_tras),Y)
```

Para comprobar si tanto la ecuación normal como el descenso de gradiente nos daban el mismo resultado aplicando la función h(x) = θTx :

-para el descenso de gradiente normalizamos los valores, creamos la matriz

```
extension=(1650-medias[1])/desviacion[1]
habitaciones=(3-medias[2])/desviacion[2]
prediccion=[1,extension,habitaciones]
print(np.matmul(np.transpose(Thetas),prediccion))
```

-para la ecuacion normal no hay que normalizar los valores

```
prediccion=[1,1650,3]
print(np.matmul(np.transpose(Thetas),prediccion))
```

conseguimos un resultado muy aproximado

```
Resultado Regresion Lineal Ecuacion  Normal :
293081.4643349892
```

```
Resultado Regresion Lineal Descenso Gradiente :
293098.4666757651
```