PRACTICA 1: REGRESION LINEAL

Una variable:

Objetivo:

Dado una conjunto de datos y para valores de x, encontrar los parámetros en la función lineal (hipótesis) h(x) para los cuales se minimice el valor de la función de coste $J(\theta_0, \theta_1)$

Funciones:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Método:

Se utilizará un algoritmo para aplicar el método de descenso de gradiente para una variable, que irá actualizando los valores de θ_0 y θ_1 simultáneamente durante un número arbitrario de repeticiones.

ALGORITMO:

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \\
&\quad \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)} \\
&\}
\end{aligned}
$$

$\alpha = 0{,}015$ 1500 iteraciones

Código:

Función hipótesis:

```
def h(x, param0, param1):
    return param0 + (x*param1)
```

Función coste:

```
def cost(param0, param1, X, Y):
    sumat = 0

    i = 0
    while(i < len(X)):
        sumat = sumat + ((h(X[i]-Y[i],param0,param1))**2)
        i = i + 1

    return sumat/(2*len(X))
```

Algoritmo de actualización de parámetros:

```
i = 0
M = 1500
alfa = 0.01
#PARAMETROS DE LA FUNCION HIPOTESIS
param0 = 1.0
param1 = 1.0

c = 0.0
min_cost = float(cost(param0, param1, X, Y))

while(i < M):
    c = float(cost(param0, param1, X, Y))

    if(c < min_cost): #SI EL COSTE SE REDUCE SE GUARDA COMO EL MINIMO
        min_cost = c

    #ACTUALIZAMOS PARAMETROS
    sum0 = 1.0/len(X) * np.sum([param0 + param1*X[i] -
                                Y[i] for i in range(len(X))])

    sum1 = 1.0/len(X) * np.sum([(param0 + param1*X[i] -
                                 Y[i])*X[i] for i in range(len(X))])

    aux0 = param0
    aux1 = param1

    param0 = aux0 - alfa * sum0
    param1 = aux1 - alfa * sum1

    i = i + 1
```

RESULTADOS DE EJECUCIÓN:

Los valores finales de los parámetros son:

θ_0 = -3,57

θ_1 = 1,16

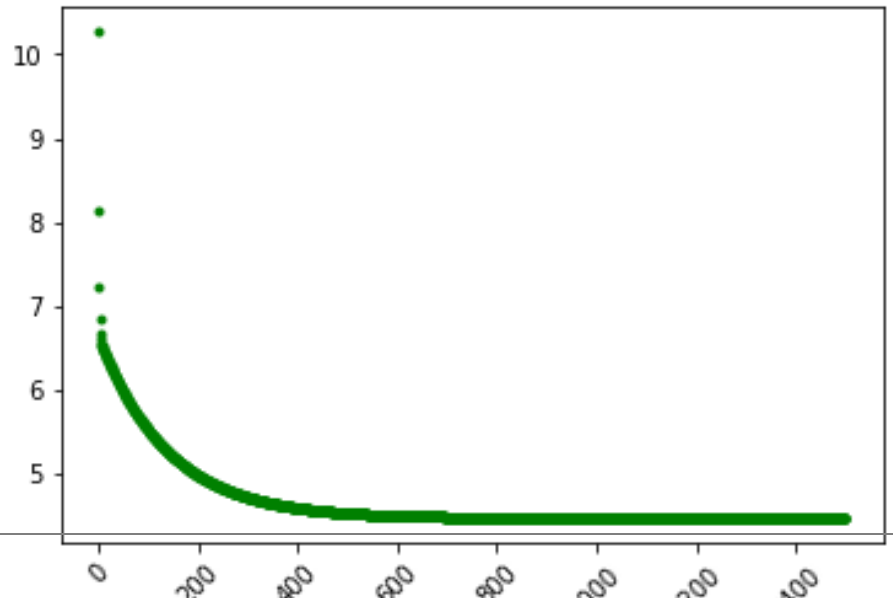
El valor de J(-3,57, 1,16) es 0.7143680244742163

Evolución del coste:

Valor de la función coste cada 100 iteraciones:

```
5.531390347077951
4.989011994198174
4.7256254706904715
4.597721286467151
4.535609223465742
4.505446735868344
4.490799409502088
4.483686462700642
4.480232316125686
4.478554934177768
4.477740374014163
4.4773448121917925
4.477152721831487
4.477059440062528
4.47701414112923
```

Representación gráfica de la evolución del coste a lo largo de las 1500 iteraciones:

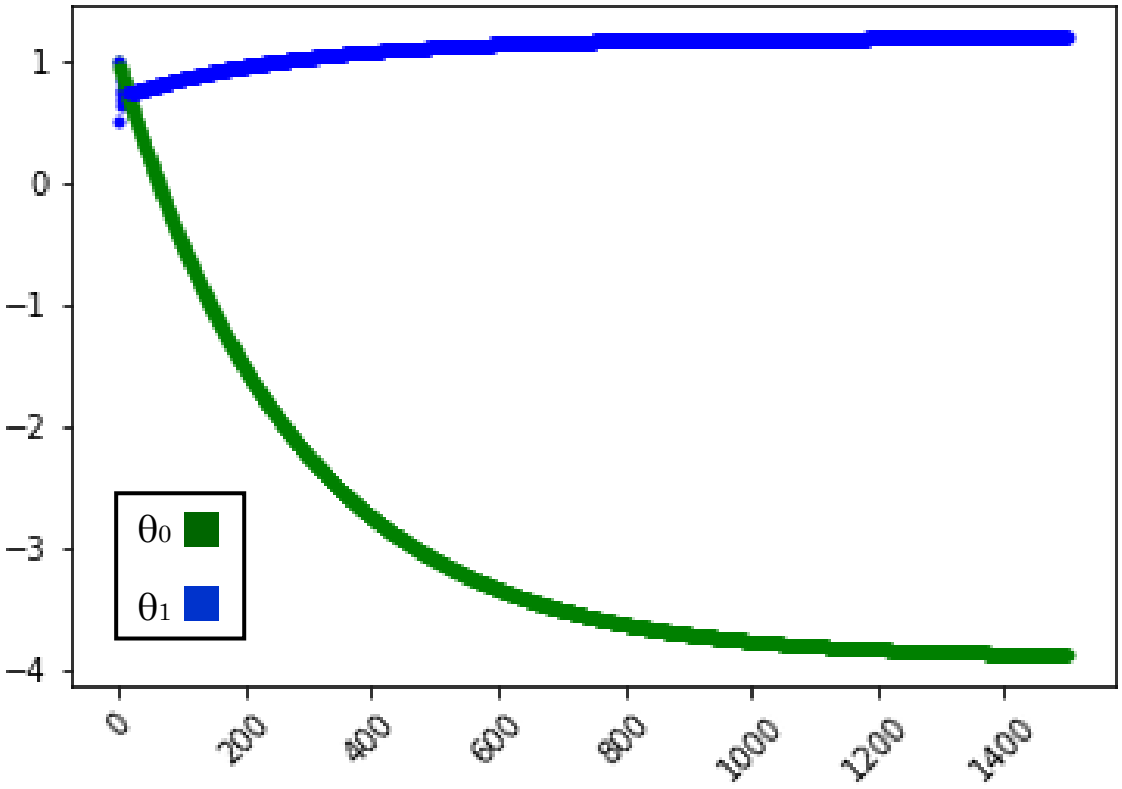


Evolución de los parámetros:

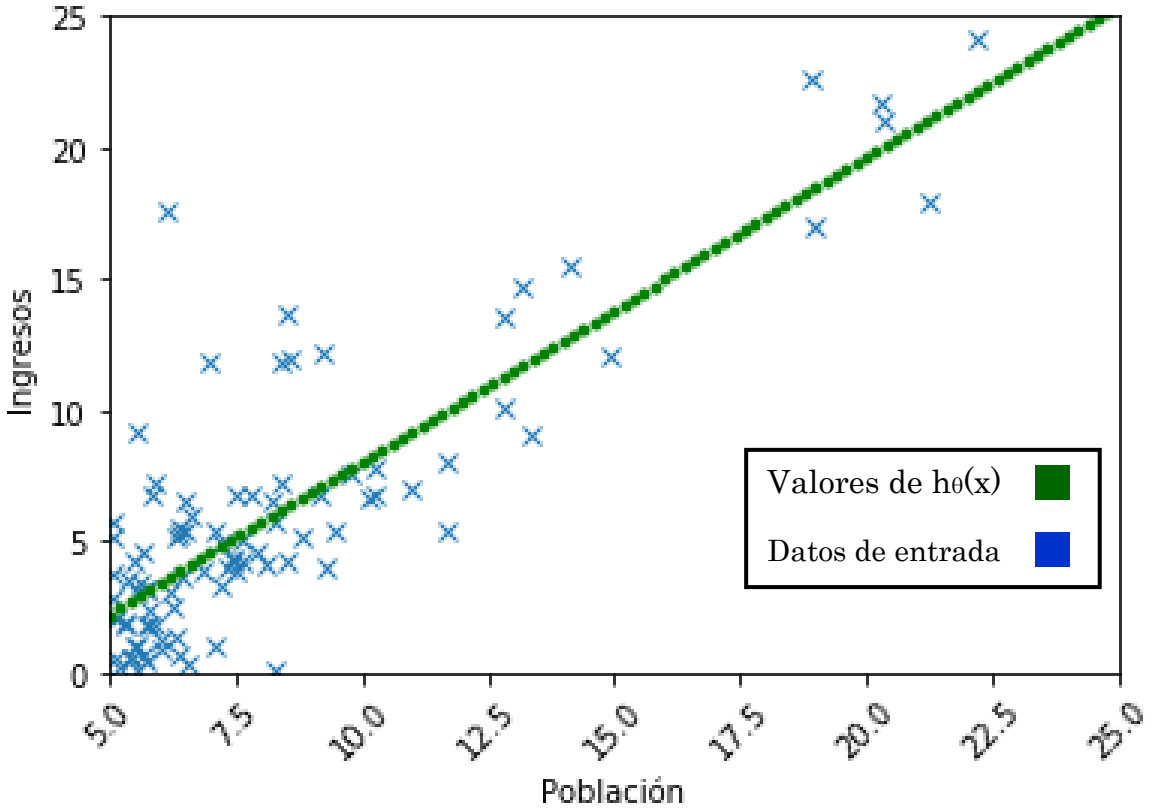
Valor del coste cada 100 iteraciones :

θ_0	θ_1
0.16698077852081894	0.7848851677225066
-0.5037096785824149	0.8522632989961753
-1.0636809433616699	0.9085184864499181
-1.5312108061433816	0.9554869323778995
-1.92155971570547	0.9947017143871796
-2.2474688892708268	1.0274428259494683
-2.5195761933061114	1.0547789561383485
-2.7467633697189378	1.0776023722670818
-2.9364459411200348	1.096658044016744
-3.0948152578280923	1.1125679596833007
-3.227040587173412	1.1258514282401846
-3.337437841413684	1.136942029887598
-3.4296104516399963	1.1462017683644037
-3.5065669858798865	1.1539328869623589
-3.57081935052134	1.1603877339243296

Representación gráfica de la evolución de los parámetros a lo largo de las 1500 iteraciones:



GRAFICA FINAL



Varias variables:

Nuevos objetivos:

En esta ocasión, se han modificado los algoritmos para funcionar con un número arbitrario de variables de entrada. En este caso concreto se utilizaran 2 variables de entrada.

Normalización de los datos:

Para poder utilizar los datos de entrada, todos deben estar en la misma escala de magnitud. Mediante este proceso, todos los datos mantienen la proporción entre ellos, pero su valor se encontrará aproximadamente entre -1 y 1.

ALGORITMO:

$$x_i \leftarrow \frac{x_i - \mu_i}{S_i}$$

S_i: range or standard deviation

Código:

```
def normalize(X):

    ac_metros = []
    ac_precio = []
    ac_habit = []

    for row in X:
        ac_metros.append(row[1])
        ac_habit.append(row[2])

    m_metros = np.mean(ac_metros)
    std_metros = np.std(ac_metros)

    m_habit = np.mean(ac_habit)
    std_habit = np.std(ac_habit)

    for row in X:
        row[1] = (row[1] - m_metros)/std_metros
        row[2] = (row[2] - m_habit)/std_habit

    return [X,[m_metros, m_habit], [std_metros, std_habit]]
```

Nota: esta función devuelve no sólo los datos de entrada normalizados, si no también los valores de la media y la desviación estándar. Esto es para poder normalizar nuevos datos de entrada, en caso de que queramos hacer predicciones.

FORMULAS MODIFICADAS:

Función hipótesis:

```
def hvect(X, params):
    Xmul = np.multiply(X,params)
    return np.sum(Xmul[i] for i in range(len(Xmul)))
```

Función de coste:

$$J(\theta) = \frac{1}{2m}(X\theta - \vec{y})^T(X\theta - \vec{y})$$

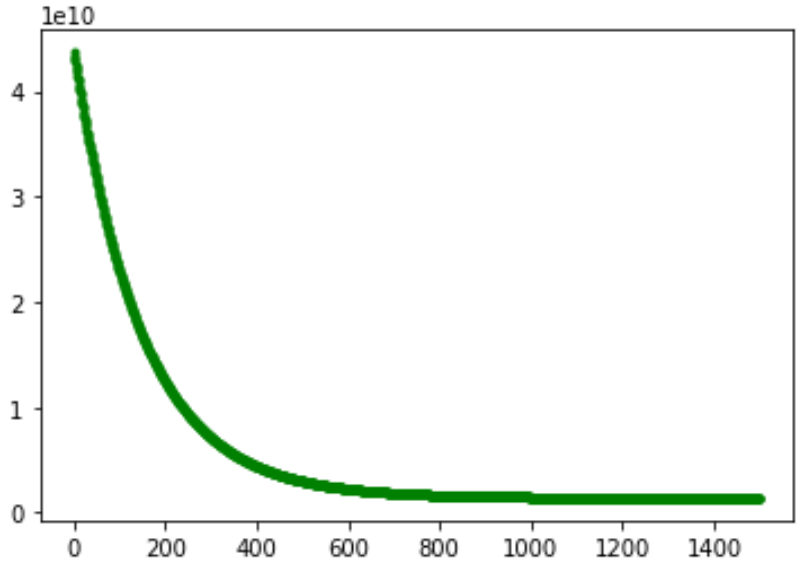
```
def costvector(params,X,Y):
    return 1.0/(len(X)*3) * np.sum([(float(hvect(X[i],params)) - Y[i])**2
                                    for i in range(len(X))])
```

Actualización de parámetros:

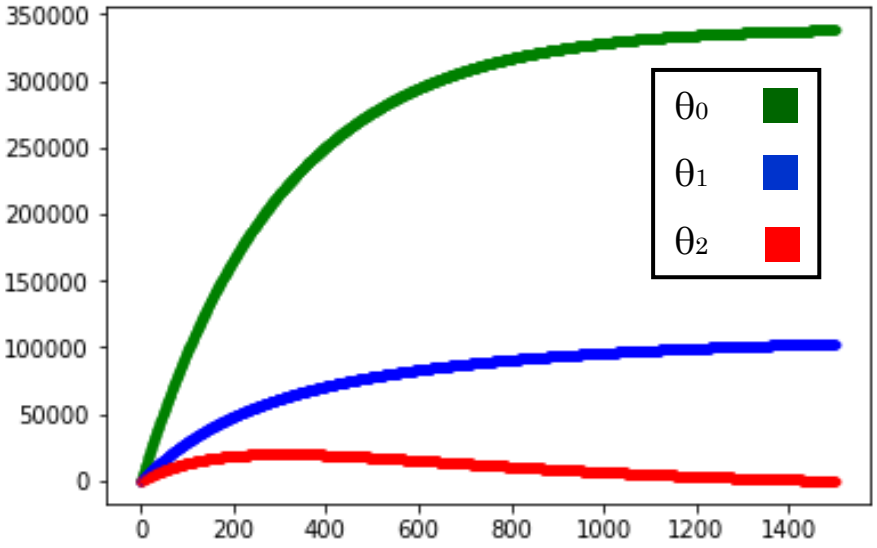
```
m = 0
M = 1500
alfa = 0.01

params = [1.0, 1.0, 1.0]

c = 0.0
min_cost = 10000.0

norm = normalize(X)
X = norm[0]

while(m < M): #M = 1500 EN ESTE CASO
    c = float(costvector(params, X, Y))

    if(c < min_cost):
        min_cost = c

    if(m %100 == 0):
        print(c)

    plt.plot(c,m,"g.")

    sumat = []
    aux = []

    for j in range(len(X[0])):
        sum = 1.0/(len(X)*3) * np.sum([(hvect(X[i],params) - Y[i])
                                       *X[i][j] for i in range(len(X))])
        sumat.append(sum)

    for p in params:
        aux.append(p)

    for i in range(len(params)):
        params[i] = aux[i] - alfa * sumat[i]

    m = m + 1
```

RESULTADO DE EJECUCION:

Evolución de los valores de la función de coste a lo largo de 1500 iteraciones:



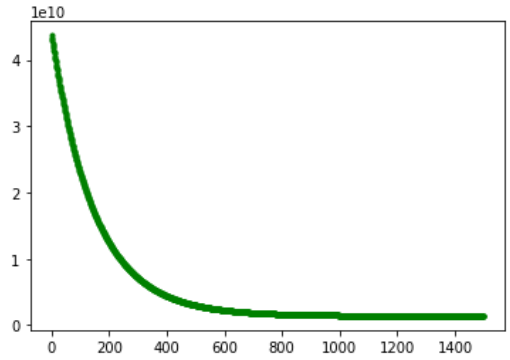
Evolución de los tres parámetros:



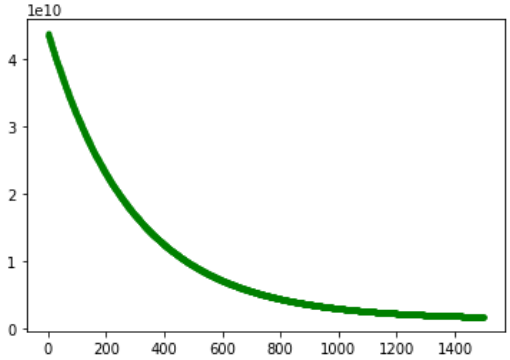
VARIANDO LA TASA DE APRENDIZAJE:

Comprobamos la evolución de la función para diferentes valores de α:

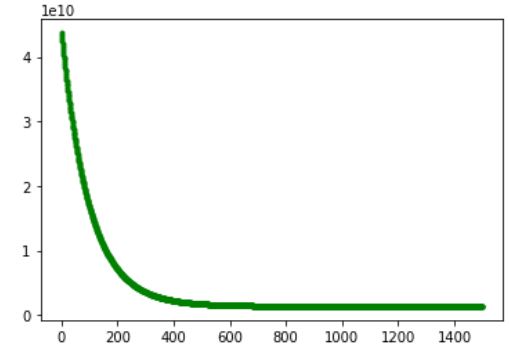
α = 0.01:



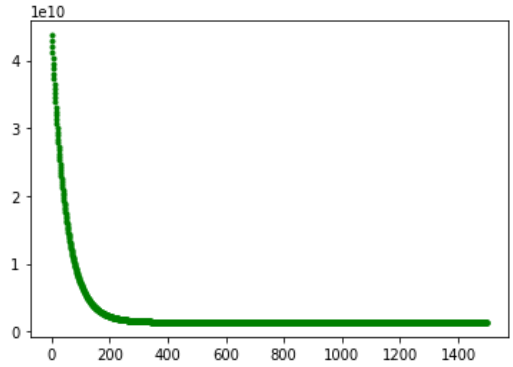
α = 0.005:



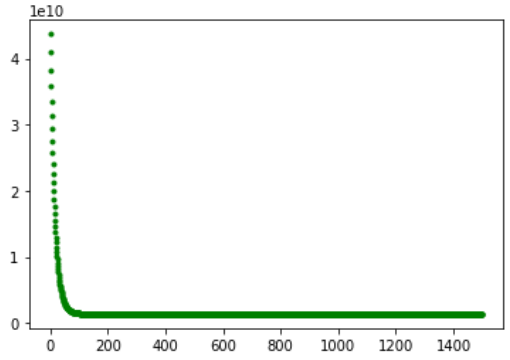
α = 0.015:



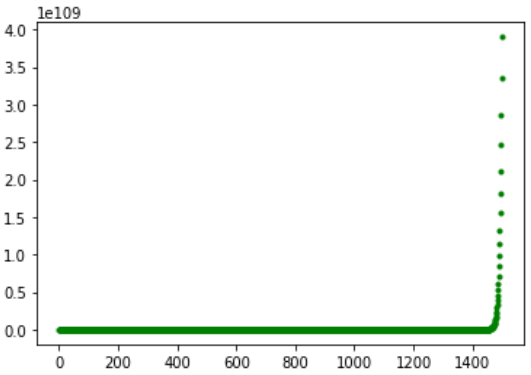
α = 0.03:



α = 0.1:



α = 4:



METODO DE LA ECUACIÓN NORMAL:

Para calcular los parámetros siguiendo el método de la ecuación normal, no hace falta normalizar los parámetros.

Ecuación:

$$\theta = (X^T X)^{-1} X^T y$$

Implementación:

```
def ecnormal(X, Y):
    Xt = np.transpose(X)
    Xmul = np.matmul(Xt, X)
    Xi = np.linalg.pinv(Xmul)

    Xb  = np.matmul(Xt, Y)
    return np.matmul(Xi,Xb)
```

Diferencia en el resultado:

Para un piso de 1650 m y 3 habitaciones, se predicen los siguientes precios

- Descenso del gradiante: `293365.776802749`
- Ecuación normal: `293081.46433498873`