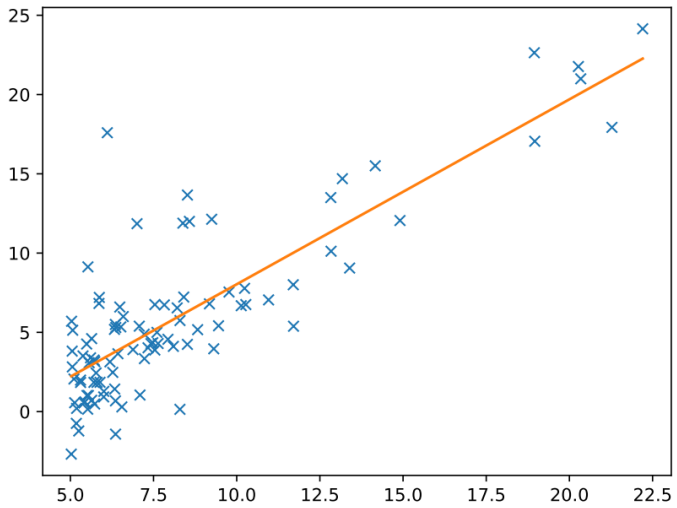
Práctica 1: Regresión Lineal

David Godoy Ruiz
Eva Lucas Leiro

Regresión lineal con una variable:

El cálculo de la regresión lineal para una sola variable está calculado con un bucle que realiza n iteraciones (en este caso 1500) del descenso de gradiente que está implementado de forma iterativa, recorriendo los vectores X e Y que contienen los puntos del fichero de datos, y calculando $h_\theta(x)$ para actualizar theta en cada iteración para minimizar el sumatorio de la función de coste.

```
def regresion_unica():
    datos = carga_csv('ex1data1.csv')
    X = datos[:, 0]
    Y = datos[:, 1]
    m = len(X)
    alpha = 0.01
    theta_0 = theta_1 = 0
    for _ in range(1500):
        sum_0 = sum_1 = 0
        for i in range(m):
            sum_0 += (theta_0 + theta_1 * X[i]) - Y[i]
            sum_1 += ((theta_0 + theta_1 * X[i]) - Y[i]) * X[i]
        theta_0 = theta_0 - (alpha / m) * sum_0
        theta_1 = theta_1 - (alpha / m) * sum_1
```



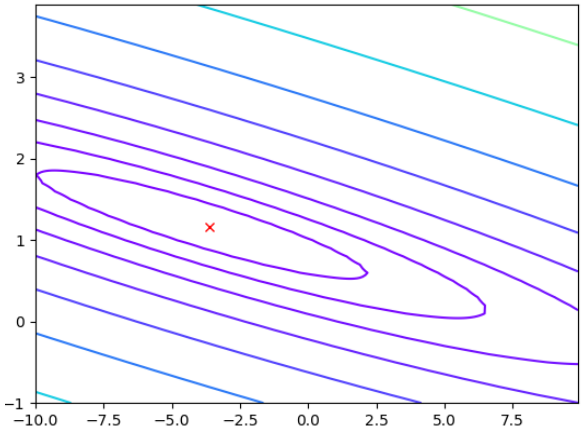
Esta gráfica muestra la recta de regresión aportada por theta_0 y theta_1 tras las 1500 iteraciones, generada de esta forma.

```
    plt.plot(X, Y, "x")
    min_x = min(X)
    max_x = max(X)
    min_y = theta_0 + theta_1 * min_x
    max_y = theta_0 + theta_1 * max_x
    plt.plot([min_x, max_x], [min_y, max_y])
    plt.savefig("resultado.pdf")
    plt.clf()
    return [theta_0, theta_1]
```

- Visualización de la función de coste

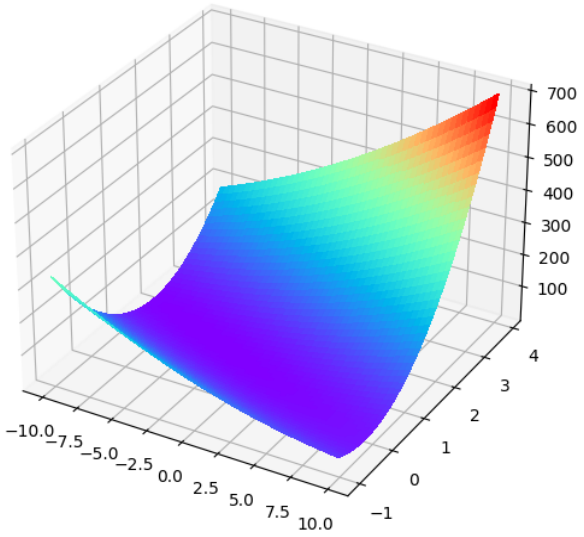
Para visualizar la función de coste utilizamos el método make_data para sacar una función que muestra la variación el coste en función de los valores de theta_0 y theta_1

```
def coste(X,Y,theta):
    aux = 0
    m = len(X)
    for i in range(m):
        aux += ((theta[0] + theta[1] * X[i])-Y[i])**2
    return aux/(2*m)
#--------------------------------------------------------------------------

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

    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]
#--------------------------------------------------------------------------
```



En el caso de dibuja_contorno se muestra la variación del coste y marcado con una x roja el resultado de la regresión lineal, mostrando que es prácticamente el punto con coste menor

```
def dibuja_contorno(puntito, c):
    plt.plot(puntito[0],puntito[1], "x", color='red')
    plt.contour(c[0],c[1],c[2], np.logspace(-2, 3, 20), cmap='rainbow')

    plt.savefig("contorno.png")
    plt.clf()
```



Dibuja_superficie muestra los costes de forma más gráfica, viéndose también a donde tiende a “bajar” el descenso de gradiente, mostrado en las zonas más azul oscuro

```
def dibuja_superficie(c):
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.plot_surface(c[0],c[1],c[2], cmap='rainbow',linewidth=0,
antialiased=False)

    plt.savefig("superficie.png")
    plt.clf()
```

Por último, graficas1 muestra todas las gráficas relacionadas con este apartado

```
def graficas1():
    datos = carga_csv('ex1data1.csv')
    X = datos[:, 0]
    Y = datos[:, 1]

    puntito=regresion_unica()
    c = make_data([-10,10],[-1,4],X,Y)

    dibuja_contorno(puntito, c)
    dibuja_superficie(c)
```

Regresión con varias variables:

Para realizar la regresión con varias variables es necesario primero normalizar X con este método, que saca las medias de cada columna, es decir x_n, y sus desviaciones para luego normalizar los valores.

```
def normalizar_mat(X):
    mu = np.mean(X, 0)
    sigma = np.std(X, 0)
    X_norm = (X-mu)/sigma
    return X_norm, mu, sigma
```

Para hacerla regresión primero es necesario poner una columna de unos en la matriz X, ya que θ_0 no tiene una x que la multiplique. Luego inicializamos el array de thetas a cero e iteramos las veces que sean necesarias (en este caso 1500) el método de descenso de gradiente

```
def regresion_varias(alpha):
    datos = carga_csv('ex1data2.csv')

    X = datos[:, :-1]
    Y = datos[:, -1]

    X_norm, mu, sigma = normalizar_mat(X)

    m = np.shape(X)[0]
    n = np.shape(X)[1]

    # añadimos una columna de 1's a la X
    X_norm = np.hstack([np.ones([m, 1]), X_norm])

    theta = np.full(n+1, 0)
    costes = []

    for i in range(1500):
        theta = gradiente_varias(X_norm, Y, theta, alpha)
        costes.append(coste(X_norm, Y, theta))
```

- Implementación vectorizada del descenso de gradiente

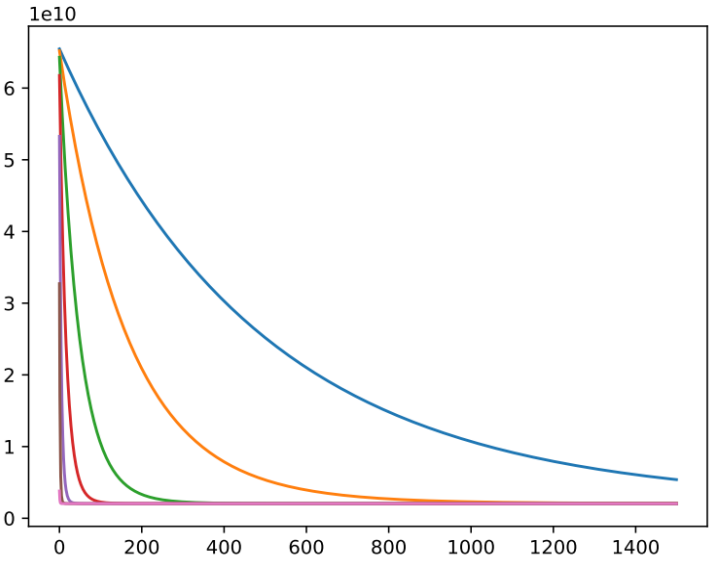
Esta implementación del descenso de gradiente es similar a la anterior pero en este caso implementado con vectores de tal forma que solo hay que hacer un bucle que itere por las diferentes thetas calculando previamente el gradiente de la función de coste y actualizando cada theta en consecuencia.

```
def gradiente_varias(X, Y, Theta, alpha):
    NuevaTheta = Theta
    m = np.shape(X)[0]
    n = np.shape(X)[1]
    H = np.dot(X, Theta)
    Aux = (H - Y)
    for i in range(n):
        Aux_i = Aux * X[:, i]
        NuevaTheta[i] -= (alpha / m) * Aux_i.sum()
    return NuevaTheta
```

Para calcular el coste dependiendo del alpha con el que se calcula el descenso de gradiente se aplica la fórmula y en cada iteración del descenso de gradiente se guarda el valor para luego mostrarlo en esta

gráfica donde se puede ver la diferencia drástica del ritmo al que desciende el coste cuanto menor sea alpha.

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



graficas2_1() muestra el resultado de la regresión mediante las thetas y el precio predicho de una casa de 1650 m^2 y 3 habitaciones, mientras que graficas2_1_alphas() muestra la gráfica que se muestra arriba.

```
def graficas2_1():
    theta, _, precio = regresion_varias(0.01)
    print("Precio lineal:", precio)
    print("theta lineal:", theta)
    plt.clf()

#-------------------------------------------------------------------------

def graficas2_1_alphas():
    alphas = [0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1]
    for i in range(len(alphas)):
        _, costes,_ = regresion_varias(alphas[i])
        plt.plot(costes)
    plt.savefig("resultado2_1.pdf")
    plt.clf()
```

```
    x1 = (1650-mu[0])/sigma[0]
    x2 = (3-mu[1])/sigma[1]
    precio = theta[0] + theta[1]*x1 + theta[2]*x2
```

Fórmula para predecir el coste según theta

- Ecuación normal

Para calcular theta según la ecuación normal se calcula con la fórmula:

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

```
def ecuacion_normal():
    datos = carga_csv('ex1data2.csv')

    X = datos[:, :-1]
    Y = datos[:, -1]

    m = np.shape(X)[0]

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

    theta = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(X), X)),
np.transpose(X)), Y)
    print("Precio normal:", theta[0] + theta[1]*1650 + theta[2]*3)
    print("theta normal:", theta)
```

Comparando los datos de la regresión con varias variables con los de la ecuación normal las thetas son drásticamente diferentes:

Fórmula para regresión: $h_\theta(x) = 340313 + 109222*x_1 - 6352*x_2$
Fórmula para normal: $h_\theta(x) = 89597.9 + 139.2*x_1 - 8738.0*x_2$

Sin embargo, a la hora de predecir el valor del piso de 1650 m^2 y 3 habitaciones los resultados son prácticamente iguales:

Precio regresión: 293031.3425581505
Precio normal: 293081.4643349715