Tarea 5: Regresión por gradiente

Alumno: Ricardo De León

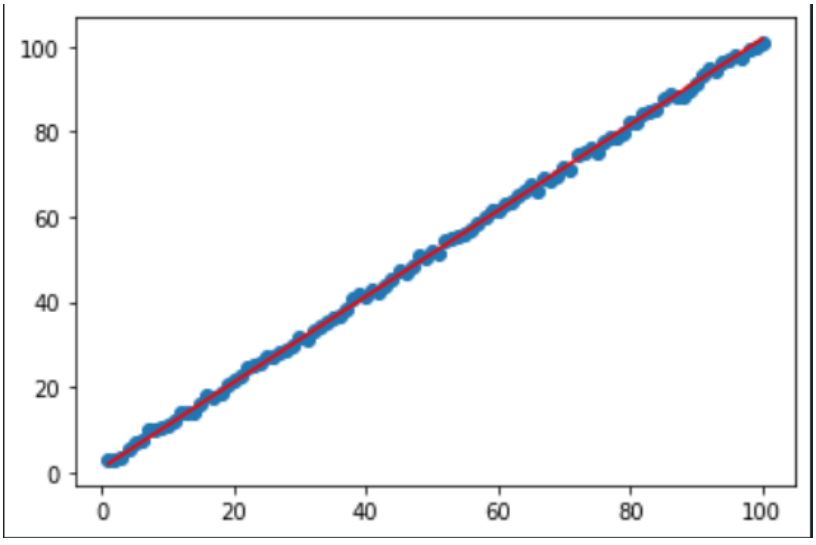
Genere un conjunto de datos (x_i, y_i) , en el que $x_i = i$ para $i = 1, \ldots, 100$ e $y_i = x_i + 3\sigma_i$, con σ_i un número aleatorio en [0,1). Utilice el método de gradiente descendente para estimar una recta de regresión lineal $y = mx + b$, de forma que el error cuadrático medio se aproxime con 4 cifras significativas. Indique sus valores iniciales para m y b , el valor de η, el error final obtenido, así como el número de épocas que fueron necesarias para obtenerlo. Grafique los puntos junto con la recta obtenida.

Generalice el método de gradiente descendente para encontrar la regresión lineal múltiple en un conjunto de datos con entradas $(X_1, X_2, \ldots, X_n)$ y salida Y, es decir, $Y = \alpha_0 + \alpha_1 X_1 + \ldots + \alpha_n X_n$. Utilice el modelo para encontrar la ecuación de regresión lineal para los datos en el siguiente archivo . De nuevo, garantice que el error cuadrático medio se aproxime con 4 cifras significativas. Indique sus valores iniciales y finales para las α_i, el valor de η, el error final obtenido, así como el número de épocas que fueron necesarias para obtenerlo.

1.- Resultados del algoritmo

```
iter # 232 Y = 1.007x + 1.0b, error recta = 161.9, norma_error 0.127300000000000
```

Grafica con la recta obtenida:



Tarea 5: Regresión por gradiente

Código:

```python
import sympy as sp
import numpy as np
from sigfig import round
import matplotlib.pyplot as plt


# Define sample data
np.random.seed(123)
xi = np.array([i for i in range(1,101)])
yi = np.array([xi[i] + 3 * np.random.uniform(0,1) for i in range(len(xi))])


# Define the learning rate
alpha = 0.0001

# Define the number of iterations
num_iterations = 1000

# Define the initial values for the a and b
a = 1
b = 1
iter = 1
n_error_t = 0

for i in range(num_iterations):
    iter+=i
    y_pred = a * xi + b
    error_recta = np.mean(np.sum(y_pred - yi)**2)

    grad_a = np.mean((a * xi + b - yi) * xi)
    grad_b = np.mean(a * xi + b - yi)


    a-= alpha * grad_a
    b-= alpha * grad_b

    # Update variable values
    grad_lst = [grad_a,grad_b]

    # Evaluate norm to break the loop
    v = sp.Matrix(grad_lst)
    n_error = sp.trigsimp(v.norm())

    if(float("{:.3e}".format(n_error)) == float("{:.3e}".format(n_error_t))):
        break

    n_error_t = n_error

print(f'iter # {iter} Y = {round(a, sigfigs=4)}x + {round(b, sigfigs=4)}b, error recta =
{round(error_recta, sigfigs=4)}, norma_error {round(n_error, sigfigs=4)}')

# Plot the dataset
plt.scatter(xi, yi)

# Calculate the final predictions for the line of regression
y_pred = a * xi + b

# Plot the line of regression
plt.plot(xi, y_pred, color='red')

# Show the plot
plt.show()
```

Tarea 5: Regresión por gradiente

2.- Resultados del algoritmo

```
iter # 1768 Y = 1.142e+00x1 + -1.810e+00x2 + 5.980e-01x3 + 1.779e+00b,  norma_error 0.221850288035414,
error recta 4.546e+01
```

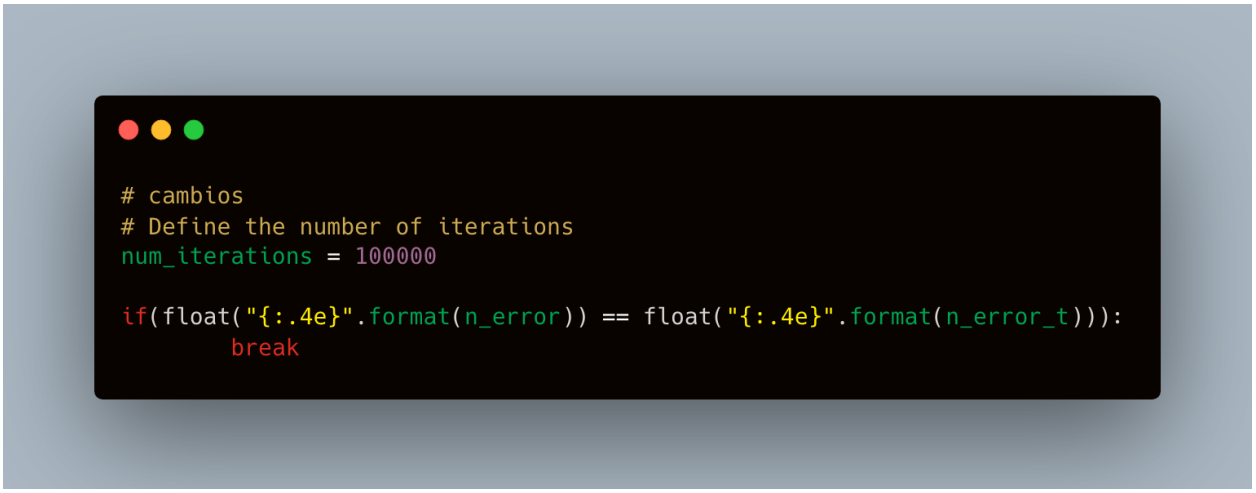
Código:

```python
import sympy as sp
import numpy as np
import pandas as pd

df = pd.read_excel('/Users/khrw896/Documents/Master/IDI/IDI2/5/tareaRGD.xlsx')

X = np.array(df.iloc[:,:-1])
y = np.array(df.iloc[:,-1])

# Initialize a and b
_, n_variables = X.shape
a = np.ones(n_variables)
b = 1


# Define the learning rate
alpha = 0.0014

# Define the number of iterations
num_iterations = 10000
iter = 1
y_pred_finals = []
norm_errors_finals = []
n_error_t = 0


for i in range(num_iterations):
    grad_lst = []
    iter += 1

    y_pred = np.dot(X, a) + b
    error_recta = np.mean(np.sum(y_pred - y)**2)

    grad_a = -2*np.dot(X.T, y - y_pred)
    grad_b = -2*np.mean(y - y_pred)

    a -= alpha * grad_a
    b -= alpha * grad_b

    # Update variable values
    for i in range(len(grad_a)):
        grad_lst.append(grad_a[i])

    grad_lst.append(grad_b)

    # Evaluate norm to break the loop
    v = sp.Matrix(grad_lst)
    n_error = sp.trigsimp(v.norm())

    if(float("{:.3e}".format(n_error)) == float("{:.3e}".format(n_error_t))):
        break
    n_error_t = n_error

print(f'iter # {iter} Y = {"{:.3e}".format(a[0])}x1 + {"{:.3e}".format(a[1])}x2 + {"
{:.3e}".format(a[2])}x3 + {"{:.3e}".format(b)}b,  norma_error {n_error}, error recta {"
{:.3e}".format(error_recta)}')
```

Nota: Si dejo el error cuadrático lo subo una cifra más y aumento las iteraciones a 10000 el resultado error se acerca más al 0, por lo tanto, tenemos un mejor resultado de la recta en la regresión.

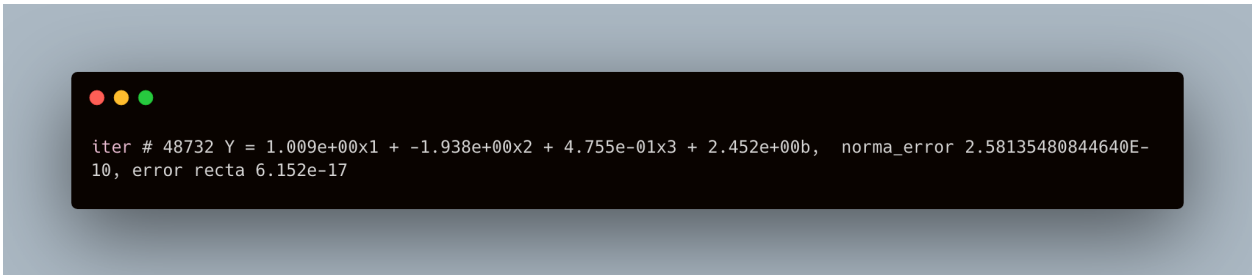
Tarea 5: Regresión por gradiente

Éste sería el resultado con esos cambios:



```
# cambios
# Define the number of iterations
num_iterations = 100000

if(float("{:.4e}".format(n_error)) == float("{:.4e}".format(n_error_t))):
        break
```

Resultado:



```
iter # 48732 Y = 1.009e+00x1 + -1.938e+00x2 + 4.755e-01x3 + 2.452e+00b,  norma_error 2.58135480844640E-
10, error recta 6.152e-17
```