Universidad Autónoma de Baja California
Facultad de Ciencias Químicas e Ingeniería.

Meta 5.2-5.3

Modelos de regresión

ALUMNO:
Urias Vega Juan Daniel
Zavala Román Irvin Eduardo
Huertas Villegas Cesar

ASIGNATURA:
Inteligencia Artificial

DOCENTE:
Castro Rodríguez Juan Ramón

GRUPO:
561

Tijuana, B.C. México

01 de junio del 2022

Matriz de Diseño

Código:

```python
import numpy as np
def designMatrix(t, X):
    q,n = X.shape
    A = np.array([])
    for p in range(1,q+1):
        M = powerMatrix(t, X[p-1,:])
        if(p == 1):
            A = M
        else:
            A = np.vstack((A, M))
    return A
def powerMatrix(t, V):
    if(V.size == 0 or t == 0):              #if|V| = 0 or t = 0
        return 1                            #M = 1
    else:
        M = np.array([])                    #M = matriz vacia
        Z = V[:-1]                          #Z = V[1, n-1]
        W = V[-1]                           #W = V[n]
        for k in range(t+1):
            #M = [M | powerMatrix(t-k,Z).W^k]
            M = np.hstack((M, np.dot(powerMatrix(t-k, Z),W**k)))
        return M

#EJEMPLO DE USO
#X = np.random.randint(-50,50,(9,1))          #(0,50,(q,n))
q = 9
n = 2
X = np.arange(q*n).reshape(q,n)
t = 2
print("Design matrix: \n",designMatrix(t,X))
```

Para realizarla se utilizó la librería numpy para la creación de matrices y vectores.

Primeramente, se tiene dos funciones una que es designMatrix con parámetros t y X, donde t es el grado del polinomio y X es una matriz y powerMatrix con parámetros, t y X donde t es el grado del polinomio y V es un vector.

La función powerMatrix funciona de manera recursiva tirándose t veces hasta que esta llegue a la última iteración en donde se regresará un 1 para añadir a la matriz, cabe destacar que cada vez que se itera esta va agregando un renglón a la matriz que será retornada.

La función designMatrix, realiza q iteraciones donde q representa el tamaño de la matriz introducida como parámetro, donde se estará llamando a la función powerMatrix para regresar los renglones y luego estos serán retornados en una matriz A que contiene los renglones obtenidos en M pero estos son añadidos en forma de columnas.

Ejemplo de ejecución y resultados:

```
[Running] python -u "c:\Users\kagman\Documents\IA\Meta_5_2_3\designMatrix.py"
Design matrix:
 [[  1.   0.   0.   1.   0.   1.]
 [  1.   2.   4.   3.   6.   9.]
 [  1.   4.  16.   5.  20.  25.]
 [  1.   6.  36.   7.  42.  49.]
 [  1.   8.  64.   9.  72.  81.]
 [  1.  10. 100.  11. 110. 121.]
 [  1.  12. 144.  13. 156. 169.]
 [  1.  14. 196.  15. 210. 225.]
 [  1.  16. 256.  17. 272. 289.]]

[Done] exited with code=0 in 0.516 seconds
```

Meta 5.2.a

a. Diseñe e implemente un modelo de regresión polinomial multivariable de orden m, usando mínimos cuadrados en lotes OLS Moore-Penrose, con métricas de desempeño. Use los datos del archivo synthetic2_dataset.dat para un modelo de regresión multivariable de grado 2.

Código:

```python
import math
import numpy as np
import designMatrix as dm
import matplotlib.pyplot as plt
import scipy.stats as stats
np.set_printoptions(precision=3) #Para limitar decimales en numpy
def RegressionOLSMoorePenrose(X,Y):
    q,n = X.shape
    #A = np.ones((q,1)) #esto sirve si es regresion lineal
    A = dm.designMatrix(2, X)
    A = np.hstack((A, X))
    ATA = np.matmul(A.T,A)
    b = A.T@Y
    THETA = np.linalg.pinv(ATA)@b
    Yh = A@THETA
    e = Y-Yh
    RMSE = math.sqrt((np.square(e)).mean())
    print("RMSE: ", RMSE)
    return THETA, RMSE, Yh, e
#Cambiar por direccion de nuestra maquina
#Importante checar con que se separan los elementos en el .dat (coma o espacio)

dataset = np.loadtxt('C:\\Users\\kagman\\Documents\\IA\\Meta_5_2_3\\{}'.format("synthetic2_dataset.dat"), delimiter = ',')
dataset = np.array(dataset)
X = dataset[:,:-1]  #Toma de la primera hasta la penultima columna
Y = dataset[:,-1:]  #Ultima columna

theta, rmse, Yh, e = RegressionOLSMoorePenrose(X,Y)
```

```
#Metricas de desempeno
r, _= stats.pearsonr (Y.T[0], Yh.T[0])
q,n = X.shape
e = Y-Yh
SSE = e.T@e
MSE = SSE/q
RMSE = np.sqrt(MSE)
INDEX = r/MSE
print("SSE: ", SSE, "\n","MSE: ", MSE, "\n", "RMSE: ", RMSE, "\n","r = : ", r, "\n","INDEX: ", INDEX, "\n",)

#Grafica correlacion Y vs Yh
plt.plot(Y.T[0], Yh.T[0], '.')
m, b = np.polyfit(Y.T[0], Yh.T[0], 1)
plt.plot(Y.T[0], m*Y.T[0] + b)
plt.title("R={}".format(r))
str = "{}*{}+{}".format(round(m,3),"Target",round(b,3))
plt.ylabel(str)
plt.show()
```

Las librerías utilizadas son principalmente matplotlib para la creación de gráficas y numpy para la creación de matrices.

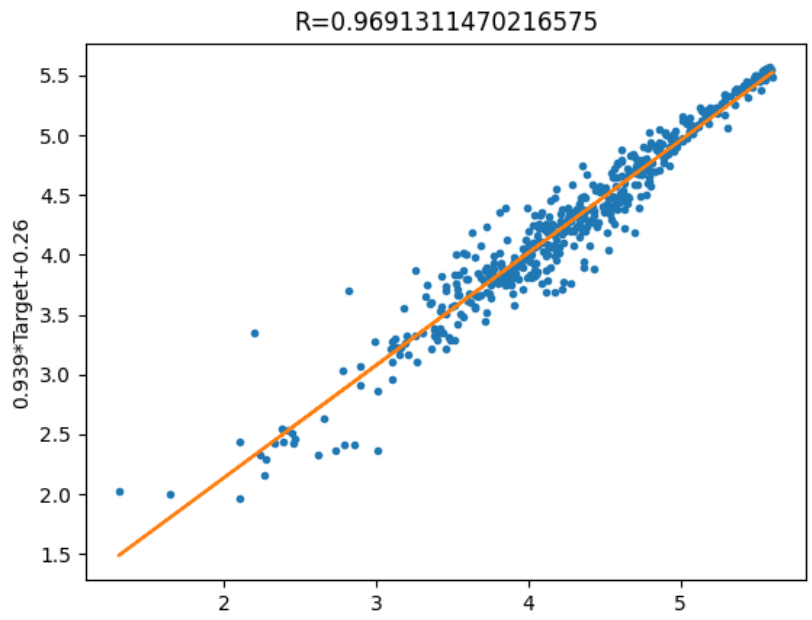
Primeramente, la función RegressionOLSMoorePenrose para poder realizar la regresión multivariable tiene dos parámetros X que son ejemplos a tomar para la regresión y Y que son los targets o resultados esperados.

La función realiza primeramente la matriz de diseño y luego a esta se le añade X en forma de columna. Luego se realiza la multiplicación entre A (matriz que almacenar la matriz de diseño y X) transpuesta y A

Luego la multiplicación entre el vector Y y la matriz A transpuesta, calculando también yh que es la respuesta estimada, e que es el error y demás parámetros que regresa la función.

El resto del código es la inicialización de los parámetros como también la lectura al archivo donde se obtendrá X y Y, la última parte del código se encarga de realizar la correlación entre la Y y Yh estimada

Resultados Obtenidos:



Con esta grafica se puede deducir que los puntos de color azul son la Yh estimada y la línea naranja es la Y, donde se puede ver una correlación de hasta un 0.9691311 muy cerca del 1, esto quiere decir que el resultado que se obtuvo es muy bueno, donde la mayoría de ellos se acercó al resultado real.

```
RMSE:  0.1841405795876781
SSE:  [[16.954]]
 MSE:  [[0.034]]
 RMSE:  [[0.184]]
 r = :  0.9691311470216575
 INDEX:  [[28.581]]
```

Y las métricas de desempeño calculadas.

Meta 5.2.b

```python
import math
import numpy as np
import designMatrix as dm
import matplotlib.pyplot as plt
import scipy.stats as stats
np.set_printoptions(precision=3) #Para limitar decimales en numpy
#Simulacion de variables estaticas
def P(x):
    P.p = x
def Theta(x):
    Theta.t = x
def RegressionRLS(X,Y):
    q,n = X.shape
    A = dm.designMatrix(1,X)
    P.p = np.zeros((X.shape[0], Y.shape[1]))
    Theta.t = np.zeros((X.shape[1], Y.shape[1]))
    for p in range(q):
        xIn = A[p,:]
        xOut = Y[p,:]
        THETA = rls(p,xIn,xOut)
    return THETA, A
def rls(k,a,y):
    npk = a.size
    m = y.size
    if(k == 0):
        Theta(np.zeros((npk,m)))
        P(1e10*np.eye(npk, npk))
    tmp1 = (P.p@a.reshape(-1,1))*(a@P.p)
    tmp2 = 1+(a@P.p)@a.T
    P(P.p-tmp1/tmp2)
    diff = y-a@Theta.t
    tmp3 = P.p@a.reshape(-1,1)
```

```
    Theta(Theta.t + tmp3*diff)
    return Theta.t
#Cambiar por direccion de nuestra maquina
#Importante checar con que se separan los elementos en el .dat (coma o espacio)
dataset = np.loadtxt('C:\\Users\\kagman\\Documents\\IA\\Meta_5_2_3\\{}'.format("synthetic2_dataset.dat"), delimiter = ',')

dataset = np.array(dataset)
X = dataset[:,:-1]  #Toma de la primera hasta la penultima columna
Y = dataset[:,-1:]  #Ultima columna
thetaHat, A = RegressionRLS(X,Y)
Yh = A@thetaHat

#Metricas de desempeno
r, _= stats.pearsonr (Y.T[0], Yh.T[0])
q,n = X.shape
e = Y-Yh
SSE = e.T@e
MSE = SSE/q
RMSE = np.sqrt(MSE)
INDEX = r/MSE
print("SSE: ", SSE, "\n","MSE: ", MSE, "\n", "RMSE: ", RMSE, "\n","r = : ", r, "\n","INDEX: ", INDEX, "\n",)

#Grafica correlacion Y vs Yh
plt.plot(Y.T[0], Yh.T[0], '.')
m, b = np.polyfit(Y.T[0], Yh.T[0], 1)
plt.plot(Y.T[0], m*Y.T[0] + b)
plt.title("R={}".format(r))
str = "{}*{}+{}".format(round(m,3),"Target",round(b,3))
plt.ylabel(str)
plt.show()
```

Se utilizan las mismas librerías que en el ejemplo anterior numpy para la creación de matrices y matplotlib para la creación de gráficas.

Primeramente, se tienen dos funciones P y Theta, las cuales son de ayuda para la creación de variables globales que conserven su valor independientemente si fueron cambiadas dentro de una función.

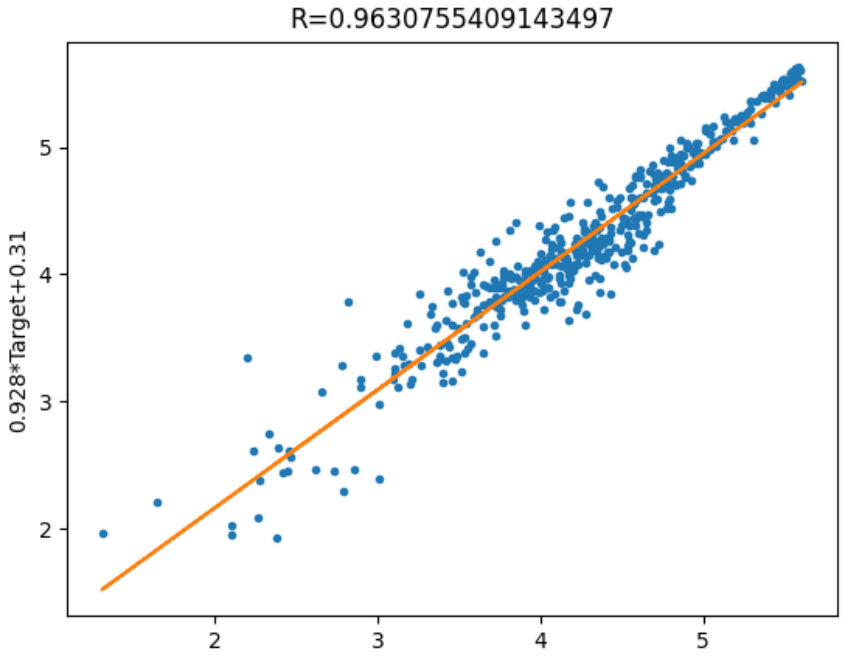
La función principal es RegressionRLS, con los mismos parámetros que la anterior X, que son todas las muestras y Y que son los resultados o target, y se tiene la función rls, donde pasas k que funciona como un indicador para realizar o no cierta función.y a , y son matrices y vectores para la función.

En la función rls se extrae el tamaño de a e y, si el parámetro k es igual a cero lo que hace es meter a Theta un arreglo de ceros, del tamaño de a e y, representados por (npk y m) y a P se le asigna el valor de, 1^e10 multiplicado por una matriz con unos en su diagonal.

Luego se tiene tres variables distintas tmp1, tmp2, tmp3, donde en cada una de ellas se almacena la multiplicación de matrices lo obtenido en P y también con los parámetros de la función como a y a transpuesta. y la diferencia que se da al restarle a Y la multiplicación matricial de a por Theta y al final retornando Theta.

Para la RegressionRLS, se obtiene las dimensiones de X y se obtiene la matriz de diseño almacenada en A, a P se le asigna un arreglo de ceros de tamaño X[0] y Y[1] y a Theta un arreglo de ceros de tamaño X[1] y Y[1]. Esta realiza un ciclo for de ’q’ iteraciones que son los renglones de X donde a Theta se la estará asignando el resultado obtenido de la función rls y al final se retorna THETA y A.

Resultados Obtenidos.



Para este ejemplo se hizo uso del synthetic2_dataset.dat

Los resultados obtenidos para esta prueba fueron bastante cercanos a él resultado esperado que sería 1, obteniendo una correlación estimada de 0.963075 que es un resultado bastante bueno, los puntos de color azul son las Yh estimadas.

Métricas de desempeño y Matriz de diseño.

```
Design matrix:
 [[  1.   0.   0.   1.   0.   1.]
 [  1.   2.   4.   3.   6.   9.]
 [  1.   4.  16.   5.  20.  25.]
 [  1.   6.  36.   7.  42.  49.]
 [  1.   8.  64.   9.  72.  81.]
 [  1.  10. 100.  11. 110. 121.]
 [  1.  12. 144.  13. 156. 169.]
 [  1.  14. 196.  15. 210. 225.]
 [  1.  16. 256.  17. 272. 289.]]
SSE:  [[20.217]]
 MSE:  [[0.04]]
 RMSE:  [[0.201]]
 r = :  0.9630755409143497
 INDEX:  [[23.818]]
```

Meta 5.2.c

```python
import numpy as np
import designMatrix as dm
import matplotlib.pyplot as plt
import scipy.stats as stats
np.set_printoptions(precision=3) #Para limitar decimales en numpy
class optimParam:
    epochs = 0
    goal = 0
    min_grad = 0
    show = 0
def RegressionGradMSE(A,e): #
    q = A.shape[0]
    m = e.shape[1]
    gradMse = -2*A.T@e / (m*q)
    return gradMse
def RegressionMSE(A,Y,vecX):
    col = A.shape[1]
    m = Y.shape[1]
    Theta = np.resize(vecX, (col,m))
    Yh = A@Theta
    e = Y-Yh
    MSE = (np.square(e)).mean()
    return MSE, e
```

La función de RegressionGradMSE, sirve para calcular el error cuadrático medio del gradiente, para poder obtener el error que este tiene, obteniendo las dimensiones de los parámetros (A,e) y retornando el gradMse que se obtiene de la multiplicación de -2 por A transpuesta por e dividido entre m por q.

La función RegressionMSE es para obtener el error, donde se vuelven a obtener las dimensiones de los parámetros (A,Y) y se le asigna a Theta una matriz redimensionada dé vecX, con las dimensiones de col y m, donde Yh estimada es la multiplicación de A por Theta y el error es la diferencia entre Y y Yh estimada y se retorna MSE y el error.

```python
def RegressionOptimgd(X,Y,grado,oP):
    q,n = X.shape
    oP.epochs+=1
    m = Y.shape[1]
    A = dm.designMatrix(grado,X)
    vecX = np.random.rand(A.shape[1], m)
    #Para graficar
    t_arreglo = np.array([])
    goal_a = np.array([])
    perf_a = np.array([])

    #Desde linea 36 a 62 implementacion de NADAM
    wt = vecX
    mt =  np.zeros((wt.shape[0], 1))
    vt = np.zeros((wt.shape[0], 1))
    mt_gorrito = np.zeros((wt.shape[0], 1))
    vt_gorrito = np.zeros((wt.shape[0], 1))
    beta_1 = 0.975
    beta_2 = 0.999
    alpha = 0.1
    oP.epochs+=1
    for t in range(oP.epochs):
        perf, e = RegressionMSE(A,Y,wt)
        gd = RegressionGradMSE(A,e)
        #vectores anteriores
        mt_gorrito_anterior = mt_gorrito
        #Algoritmo
        mt = beta_1*mt+(1-beta_1)*gd
        vt = beta_2*vt+(1-beta_2)*gd**2
        mt_gorrito = mt/(1-beta_1**(t+1))
        vt_gorrito = vt/(1-beta_2**(t+1))
        wt = wt - (alpha/(np.sqrt(vt_gorrito)+1e-8))*(beta_1*mt_gorrito_anterior+((1-beta_1)/(1-beta_1**(t+1)))*gd)
        if(perf <= oP.goal):
            break
        elif(np.linalg.norm(gd) <  oP.min_grad):
            break
        elif(t ==  oP.epochs-1):
            break
        perf_a = np.append(perf_a, perf)
```

La función de RegressionOptimgd utilizando el método de Nadam, es para obtener el óptimo gradiente, donde se le pasa como parámetros una matriz y un vector y el grado del polinomio. Primeramente, se inicializan las variables que serán necesarias tanto para el algoritmo como para el almacenamiento de los datos y para la graficación de los mismos y también A que es la matriz de diseño.

Para el algoritmo tenemos la inicialización de mt y vt como matrices de ceros beta_1 como 0.975 y beta_2 como 0.999 que estas nos ayudaran a que el algoritmo pueda converger de una manera mucho más rápida. y alpha cómo 0.1 que es el paso inicial (La tasa de aprendizaje)

Luego se realiza un ciclo que continuará hasta que op.epochs se complete o algunas de las condiciones dentro del ciclo se cumplan tales como, que se consiga la meta, o que el mínimo gradiente sea mayor al obtenido.

En todo caso que no se cumplan las condiciones se realiza el siguiente código siendo el mismo caso para mt como vt, donde beta(1,2) se multiplica por (mt,vt) más (1-beta(1,2)) multiplicado por el gradiente en caso de beta_1 y multiplicado por el gradiente al cuadrado para beta_2. De igual forma para mt_gorrito y vt_gorrtito es la división de (mt,vt) entre (1-beta(1,2)**(t+1)).

Por último, para calcular wt que es un arreglo de n variables que serán modificados con ayuda de las demás intentando hacer que se acerque a 1 ya sea por la izquierda o por la derecha o sea un vector de resultados.

Para obtener wt se aplica la siguiente fórmula.

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{S}_t + \epsilon}} \left(\beta_1 \hat{V}_{t-1} + \frac{1-\beta_1}{1-\beta_1^t} \cdot \frac{\partial L}{\partial w_t} \right)$$

Donde los valores previamente calculados se utilizarán para ayudar a ajustar los valores y tratar de aproximarse al 1.

```
63      #Grafica dinamica
64      for i in range(0,t,t//6):
65          t_arreglo = np.array(range(0,i,1))
66          goal_a = np.zeros(i)+oP.goal
67          plt.yscale("log")
68          plt.plot(t_arreglo, perf_a[:i], 'b')
69          plt.plot(t_arreglo, goal_a[:i], 'r')
70          plt.ylabel("Perf")
71          plt.xlabel("Epochs")
72          plt.pause(0.1)
73      vecX = wt
74      t_arreglo = np.arange(t)
75      goal_a = np.zeros(t)+oP.goal
76      plt.yscale("log")
77      plt.plot(t_arreglo, perf_a, 'b')
78      plt.plot(t_arreglo, goal_a, 'r')
79      plt.ylabel("Perf")
80      plt.xlabel("Epochs")
81      plt.show()
82
83      print("Perf",perf,"Grad",gd[0][0], "Epochs", t)
84      Theta = np.resize(vecX, (A.shape[1],m))
85      return Theta, A
```

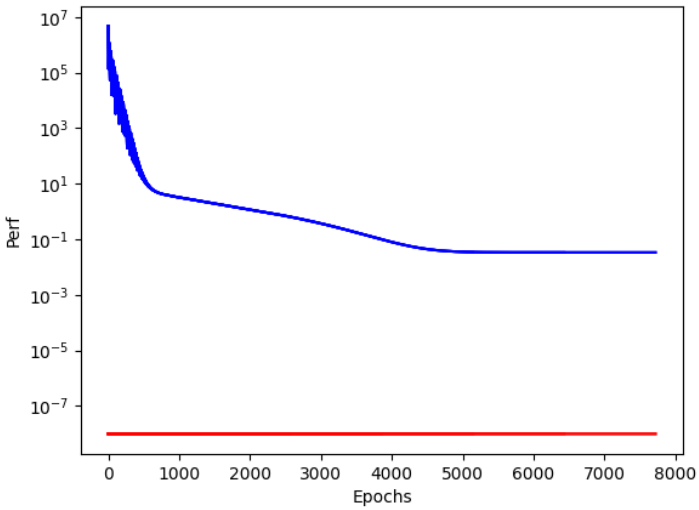
Esta sección del código funciona para poder imprimir como se estuvieron comportando los datos y el algoritmo como tal, esperando que todos pudieran acercarse cada vez más al 1 o al 0 y también cuentas iteraciones le tomó para que este pudiera converger.

Al final del algoritmo se imprimen los resultados obtenidos y se retorna Theta y la matriz de diseño (A).

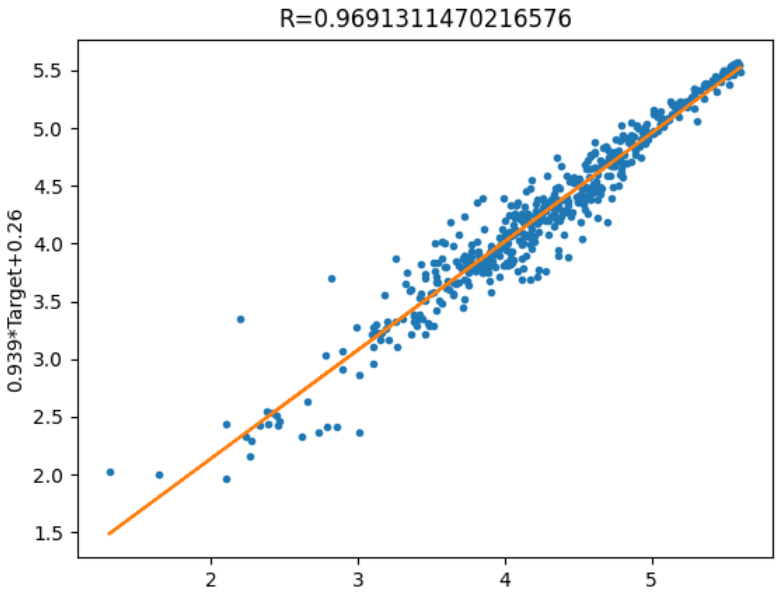
```
dataset = np.loadtxt('C:\\Users\\kagman\\Documents\\IA\\Meta_5_2_3\\{}'.format("synthetic2_dataset.dat"), delimiter = ',')
dataset = np.array(dataset)
X = dataset[:,:-1]  #Toma de la primera hasta la penultima columna
Y = dataset[:,-1:]  #Ultima columna
oP = optimParam()
oP.epochs = 100000
oP.goal = 1e-8
oP.min_grad = 1e-10
oP.show = 20

n = X.shape[1]
m = Y.shape[1]
#theta = np.random.rand(n+1, m)

grado = 2
thetaHat,A = RegressionOptimgd(X,Y,grado,oP)
Yh = A@thetaHat

#Metricas de desempeno
r, _= stats.pearsonr (Y.T[0], Yh.T[0])
q,n = X.shape
e = Y-Yh
SSE = e.T@e
MSE = SSE/q
RMSE = np.sqrt(MSE)
INDEX = r/MSE
print("SSE: ", SSE, "\n","MSE: ", MSE, "\n", "RMSE: ", RMSE, "\n","r = : ", r, "\n","INDEX: ", INDEX, "\n",)

#Grafica correlacion Y vs Yh
plt.plot(Y.T[0], Yh.T[0], '.')
m, b = np.polyfit(Y.T[0], Yh.T[0], 1)
plt.plot(Y.T[0], m*Y.T[0] + b)
plt.title("R={}".format(r))
str = "{}*{}+{}".format(round(m,3),"Target",round(b,3))
plt.ylabel(str)
plt.show()
```

Código para la lectura del archivo utilizado para las pruebas synthetic2_dataset.dat y la inicialización de los parámetros necesarios para el algoritmo, asi como tambien el cálculo de las métricas de desempeño y la impresión de la correlación que existe entre Y y Yh estimada.

Resultados Obtenidos.



Gráfica que muestra el comportamiento del algoritmo con la finalidad de acercarse al 0.



Gráfica que representa la correlación que existe entre Y y Yh estimada, que podemos obtener una muy buena correlación de 0.9691311 muy cerca del 1, los puntos azules representan a Yh estimada.

Métricas de desempeño y Matriz de diseño.

```
Design matrix:
 [[  1.   0.   0.   1.   0.   1.]
 [  1.   2.   4.   3.   6.   9.]
 [  1.   4.  16.   5.  20.  25.]
 [  1.   6.  36.   7.  42.  49.]
 [  1.   8.  64.   9.  72.  81.]
 [  1.  10. 100.  11. 110. 121.]
 [  1.  12. 144.  13. 156. 169.]
 [  1.  14. 196.  15. 210. 225.]
 [  1.  16. 256.  17. 272. 289.]]
Perf 0.03390775305088601 Grad -1.477767508362149e-11 Epochs 7723
SSE:  [[16.954]]
 MSE:  [[0.034]]
 RMSE:  [[0.184]]
 r = :  0.9691311470216576
 INDEX:  [[28.581]]
```