Name: Aishwary Sharma

Entry Number: 2022CH11437

Group-1

Q1. (Linearization)

// Header files and namespaces

#include<bits/stdc++.h>

using namespace std;

// k = km\*c^2 / (cs+c^2)

// (1/k) = (cs/km)(1/c^2)+ (1/km)

// Y = a1\*X + a0

void lstf(vector<int> v){

for(int i= 0;i< v.size();i++){

cout << v[i]<< " ";

}

}

vector<double> summation1reg(vector<double>x , vector<double>y){

//y\_pred = a1\*x + a0

//a1 = (N\*SIGMA(xi\*yi)-SIGMA(xi)\*SIGMA(yi))/N\*SIGMA(xi^2)-(SIGMA(xi))^2

//a0 = SIGMA(yi)/N - (SIGMA(xi)/N)\*a1

int N = x.size();

double sum\_xi = 0;

for(int i= 0; i<N;i++){

sum\_xi += x[i];

}

// cout <<"sum\_xi = " <<sum\_xi ;

double sum\_yi = 0;

for(int i= 0; i<N;i++){

sum\_yi += y[i];

}

//cout <<"sum\_yi = " <<sum\_yi ;

double sum\_xi\_sq = 0;

for(int i= 0; i<N;i++){

sum\_xi\_sq += (x[i])\*(x[i]);

}

//cout <<"sum\_xi\_sq = " <<sum\_xi\_sq;

double sum\_xi\_yi = 0;

for(int i= 0; i<N;i++){

sum\_xi\_yi += (x[i])\*(y[i]);

}

//cout <<"sum\_xi\_yi = " <<sum\_xi\_yi ;

double a1 = ((N\*sum\_xi\_yi)-(sum\_xi\*sum\_yi))/((N\*sum\_xi\_sq)-(sum\_xi\*sum\_xi));

double a0 = ((sum\_yi/N)-(sum\_xi/N)\*a1);

cout << "y\_pred = "<< a1 << "x + "<< a0 << endl ;

vector<double> params = {a0,a1} ;

return params ;

}

vector<double> y\_prediction (double a0, double a1, vector<double> x) {

vector<double> y\_pred ;

for(auto val : x ){

double value = a1\*val + a0 ;

y\_pred.push\_back(value);

cout<< value << endl;

}

return y\_pred;

}

int main() {

// taking values of c

ifstream in("Input-1.txt");

// 5 values of c

double c ;

vector<double> X;

for(int i = 0; i<5; i++){

in >> c;

X.push\_back(1/(c\*c));

}

double k ;

vector<double> y ;

for(int i = 0; i<5; i++){

in >> k ;

y.push\_back(1/k);

}

double a0 , a1;

vector<double> params = summation1reg(X,y);

a0 = params[0] ; a1= params[1] ;

double kmax , cs ;

// a0 = 1/kmax

// a1 = cs/kmax

kmax = 1/a0 ;

cs = a1\*kmax ;

cout << "kmax = " << kmax << endl ;

cout << "cs = " << cs << endl ;

// Given c = 2 mg/L

double x = 0.25 ;

double yg = a1\*x + a0 ;

//k = 1/y ;

double k\_pred = 1 / yg ;

cout << "Growth rate at c = 2mg/L = " << k\_pred ;

}

Input-

0.5

0.8

1.5

2.5

4

1.1

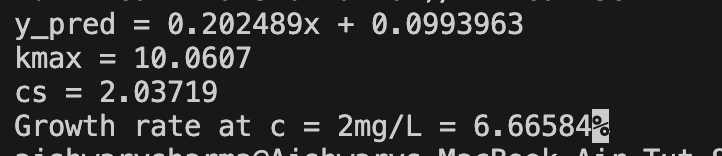
2.4

5.3

7.6

8.9

Output-



Q2.(Non Linear Regression)

#include<bits/stdc++.h>

using namespace std;

void lstf(vector<vector<double>>arr){

for(int i = 0 ; i < arr.size() ; i++){

for(int j = 0; j< arr[0].size() ; j++){

cout << arr[i][j] << " ";

}

cout << endl ;

}

cout << endl ;

}

void lstf1D(vector<double> v){

for(int i = 0 ; i<v.size(); i++){

cout << v[i] << " ";

}

cout << endl ;

}

vector<vector<double>> transpose( vector<vector<double>>& matrix) {

int rows = matrix.size();

int cols = matrix[0].size();

// Initialize the transpose matrix with dimensions cols x rows

vector<vector<double>> transpose(cols, vector<double>(rows, 0));

// Fill in the transpose matrix

for (int i = 0; i < rows; ++i) {

for (int j = 0; j < cols; ++j) {

transpose[j][i] = matrix[i][j];

}

}

return transpose;

}

vector<vector<double>> matrixMultiply( vector<vector<double>>& A, vector<vector<double>>& B) {

int m = A.size(); // Number of rows in A

int n = A[0].size(); // Number of columns in A

int k = B.size(); // Number of rows in B

int p = B[0].size(); // Number of columns in B

if(n != k){

cout << "matrix multiply dimension error" << endl ;

exit(1);

}

// Initialize the result matrix C with dimensions mxp

vector<vector<double>> C(m, vector<double>(p, 0));

// Perform matrix multiplication

for (int i = 0; i < m; ++i) {

for (int j = 0; j < p; ++j) {

for (int k = 0; k < n; ++k) {

C[i][j] += A[i][k] \* B[k][j];

}

}

}

return C;

}

void pivot(vector < vector<double> >& v , vector < vector<double> >& I, int k ){

// k --> arr[k][k]

int n = v.size() ;

double big = v[k][k] ;

int p = k ;

for(int i = k+1 ; i< n; i++){

if(v[i][k] > big){

big = v[i][k];

p = i ;

}

}

if(p != k ){

swap(v[p],v[k]);

swap(I[p],I[k]);

}

}

vector<vector<double>> inverseMatrix( vector<vector<double>>& mat) {

int n = mat.size();

int m = mat[0].size();

if (n != m){

cout << "Dimension error" << endl ;

exit(1);

}

vector<vector<double>>Iden(n,vector<double>(n,0));

for(int i = 0 ; i< n; i++){

Iden[i][i] = 1 ;

}

for(int k = 0; k<n ; k++){

pivot(mat,Iden,k);

double div = mat[k][k];

if(abs(div) < 1e-05 ){

cout << "Not invertible";

exit(1);

}

for(int j = k ; j< n ; j++){

mat[k][j]= mat[k][j]/div;

}

for(int j = 0 ; j< n ; j++){

Iden[k][j]= Iden[k][j]/div;

}

for(int l = 0; l< n; l++){

if(l == k){

continue;

}

else{

double factor = mat[l][k];

for(int j = k ; j<n; j++ ){

mat[l][j] = mat[l][j] -factor\*mat[k][j];

if(abs(mat[l][j])<1e-05){

mat[l][j] = 0;

}

}

for(int j = 0 ; j<n; j++ ){

Iden[l][j] = Iden[l][j] -factor\*Iden[k][j];

if(abs(Iden[l][j])<1e-05){

Iden[l][j] = 0;

}

}

}

}

}

return Iden ;

}

double fns(double a4, double b4, double x){

return a4\*x\*exp(b4\*x);

}

double Diff\_a4 (double a4, double b4, double x ){

double del = 1.0e-8;

double a = (fns(a4+del,b4,x)-fns(a4,b4,x))/del ;

return a;

}

double Diff\_b4 (double a4, double b4, double x ){

double del = 1.0e-8;

double a = (fns(a4,b4+del,x)-fns(a4,b4,x))/del ;

return a;

}

int main(){

ifstream in("Input-2.txt");

int m ; // no. of training examples

in >> m ;

//cout << m << endl ;

vector<double> Yi(m) ;

vector<double> Xi(m) ;

for(int i=0 ; i< m; i++){

in >> Xi[i] ;

}

for(int i=0 ; i< m; i++){

in >> Yi[i] ;

}

//lstf1D(Xi);

//lstf1D(Yi);

// matrix D

double a4 = 1.0 ; double b4 = 1.0 ;

while(true){

vector<vector<double>> D(m) ;

for(int i = 0 ; i< m; i++){

vector<double> ele = {Yi[i]-fns(a4,b4,Xi[i])} ;

D[i]=ele;

}

//cout << "D: ";

//lstf(D);

vector<vector<double>> Zj(m) ;

for(int i = 0 ; i< m; i++){

vector<double> ele = {Diff\_a4(a4,b4,Xi[i]), Diff\_b4(a4,b4,Xi[i])} ;

Zj[i] = ele;

}

//lstf(Zj);

vector<vector<double>> Zj\_T = transpose(Zj);

//lstf(Zj\_T);

vector<vector<double>> Zj\_T\_Zj = matrixMultiply(Zj\_T,Zj);

//lstf(Zj\_T\_Zj);

vector<vector<double>> Zj\_T\_Zj\_inverse = inverseMatrix(Zj\_T\_Zj);

vector<vector<double>> Zj\_T\_D = matrixMultiply(Zj\_T,D);

vector<vector<double>> Del\_A = matrixMultiply(Zj\_T\_Zj\_inverse, Zj\_T\_D);

cout << "Del\_A : " << endl ;

lstf(Del\_A);

double error = max(abs(Del\_A[0][0]/a4 ), abs(Del\_A[0][0]/b4) ) ;

cout << "Error: " << error << endl ;

if(error < 1e-05){

break ;

}

a4 = a4 + Del\_A[0][0];

b4 = b4 + Del\_A[1][0];

}

cout << "Alpha4 = " << a4 << endl ;

cout << "Beta4 = " << b4 << endl ;

}

Input-

9

0.1

0.2

0.4

0.6

0.9

1.3

1.5

1.7

1.8

0.75

1.25

1.45

1.25

0.85

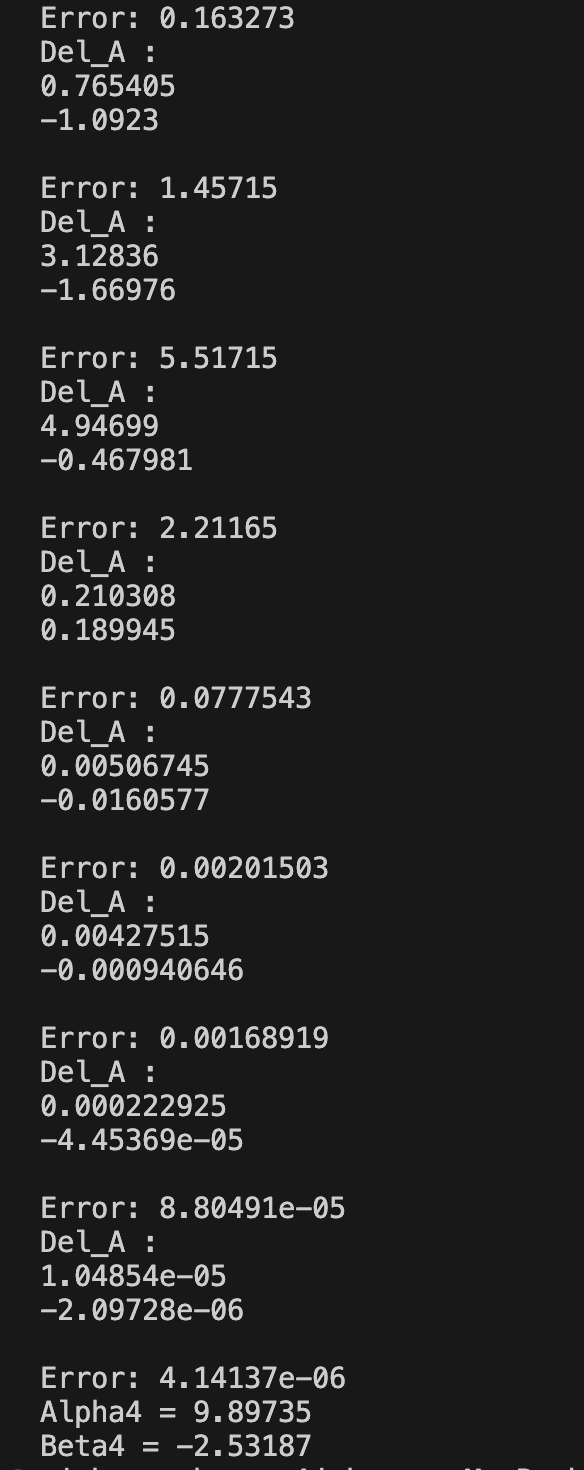
0.55

0.35

0.28

0.18

Output-



Graph:

