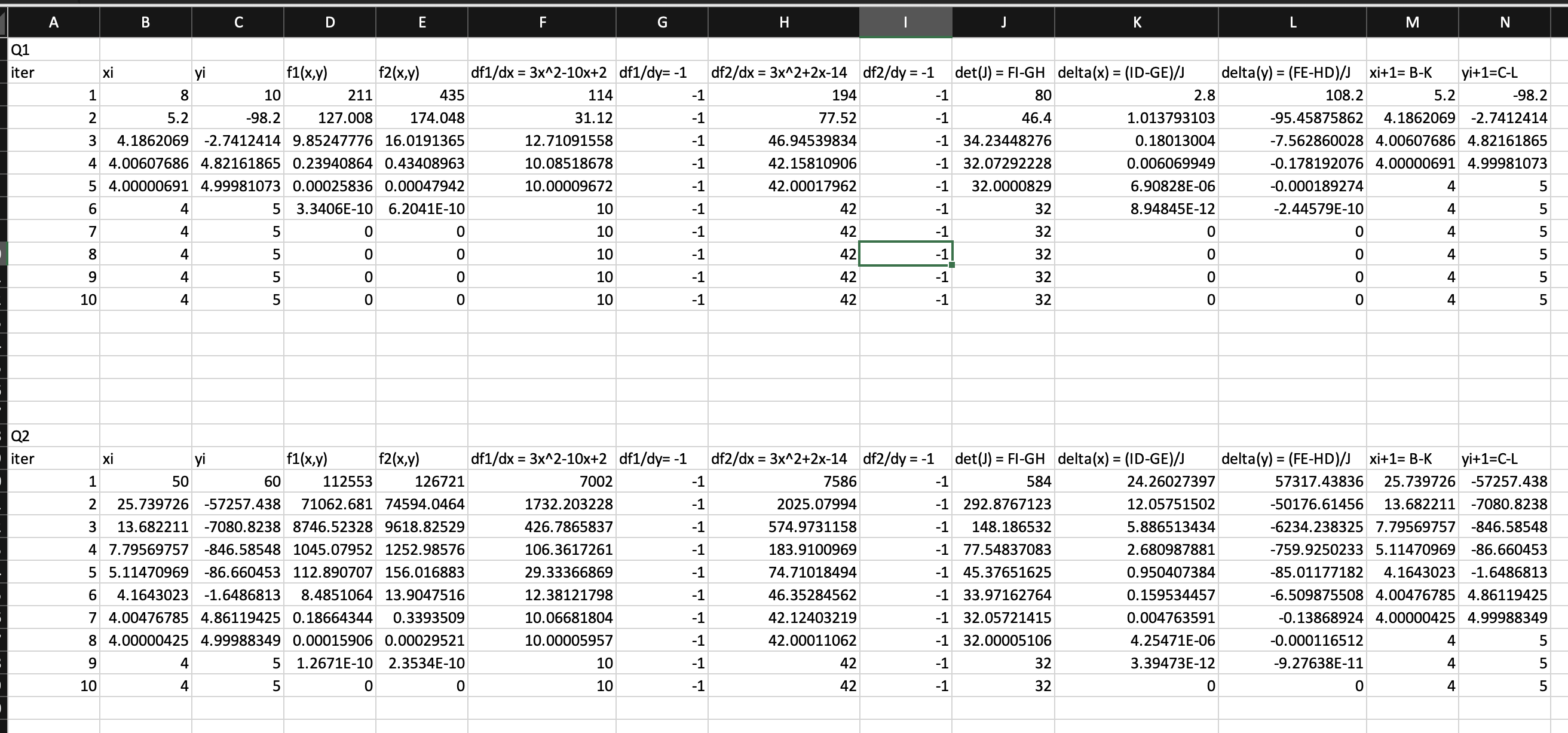
Aishwary Sharma

2022CH11437

Group-1

Q1.

Q2.

// Header files and namespaces

#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 ;

}

}

//f1(x,y,z)= x^2+y^2+z^2-4

double f1(double x, double y, double z){

return x\*x+y\*y+z\*z-4 ;

}

//f2(x,y,z)=x^2+y^2-1

double f2(double x, double y, double z){

return x\*x+y\*y-1;

}

//f3(x,y,z)= 4y^2 +z^2-4

double f3(double x, double y, double z){

return 4\*y\*y+z\*z-4;

}

double del = 1.0e-4;

double PartialX (int fno , double x, double y, double z ){

if (fno == 1){

double a = (f1(x+del,y,z)-f1(x,y,z))/del ;

return a;

}

else if(fno == 2){

double a = (f2(x+del,y,z)-f2(x,y,z))/del ;

return a;

}

else{

double a = (f3(x+del,y,z)-f3(x,y,z))/del ;

return a;

}

}

double PartialY (int fno , double x, double y, double z ){

if (fno == 1){

double a = (f1(x,y+del,z)-f1(x,y,z))/del ;

return a;

}

else if(fno == 2){

double a = (f2(x,y+del,z)-f2(x,y,z))/del ;

return a;

}

else{

double a = (f3(x,y+del,z)-f3(x,y,z))/del ;

return a;

}

}

double PartialZ (int fno , double x, double y, double z ){

if (fno == 1){

double a = (f1(x,y,z+del)-f1(x,y,z))/del ;

return a;

}

else if(fno == 2){

double a = (f2(x,y,z+del)-f2(x,y,z))/del ;

return a;

}

else{

double a = (f3(x,y,z+del)-f3(x,y,z))/del ;

return a;

}

}

vector<vector<double>> JacIntoFuncV(vector<vector<double> >jac,vector<vector<double>> funcV ){

vector<vector<double>> res(3,vector<double>(1,0));

for(int i = 0; i<3 ;i++){

double ele = 0;

for(int j=0;j<3;j++){

ele += jac[i][j]\*funcV[j][0];

}

res[i][0] = ele;

}

return res;

}

//finding inverse

std::vector<std::vector<double>> inverseMatrix(const std::vector<std::vector<double>>& mat) {

int n = mat.size();

// Check if the input matrix is 3x3

if (n != 3 || mat[0].size() != 3 || mat[1].size() != 3 || mat[2].size() != 3) {

// Return an empty matrix as an error indicator

return std::vector<std::vector<double>>(3, std::vector<double>(3, 0.0));

}

// Create the identity matrix

std::vector<std::vector<double>> identity(3, std::vector<double>(3, 0.0));

for (int i = 0; i < 3; i++) {

identity[i][i] = 1.0;

}

std::vector<std::vector<double>> augmented(3, std::vector<double>(6, 0.0));

// Create an augmented matrix [mat | identity]

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

augmented[i][j] = mat[i][j];

augmented[i][j + 3] = identity[i][j];

}

}

// Perform Gaussian elimination with partial pivoting

for (int k = 0; k < 3; k++) {

// Find the pivot

double maxVal = 0.0;

int pivotRow = -1;

for (int i = k; i < 3; i++) {

if (fabs(augmented[i][k]) > maxVal) {

maxVal = fabs(augmented[i][k]);

pivotRow = i;

}

}

if (pivotRow == -1) {

// The matrix is singular; return an empty matrix as an error indicator

return std::vector<std::vector<double>>(3, std::vector<double>(3, 0.0));

}

// Swap rows

std::swap(augmented[k], augmented[pivotRow]);

double pivot = augmented[k][k];

// Normalize the pivot row

for (int j = k; j < 6; j++) {

augmented[k][j] /= pivot;

}

// Eliminate non-zero entries below the pivot

for (int i = 0; i < 3; i++) {

if (i == k) {

continue;

}

double factor = augmented[i][k];

for (int j = k; j < 6; j++) {

augmented[i][j] -= factor \* augmented[k][j];

}

}

}

// Extract the inverse from the right half of the augmented matrix

std::vector<std::vector<double>> inverse(3, std::vector<double>(3, 0.0));

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

inverse[i][j] = augmented[i][j + 3];

}

}

return inverse;

}

void final(){

vector<vector<double>> xiV(3,vector<double>(1,0));

cout << "Give initial Guess: " << endl ;

for(int i =0 ;i<3;i++){

cin >> xiV[i][0];

}

int itr = 1;

while(true){

//creating Jacobian Matrix

vector<vector<double>> Jac(3,vector<double>(3,0));

for(int i = 1;i<4;i++){

Jac[i-1][0] = PartialX(i,xiV[0][0],xiV[1][0],xiV[2][0]);

Jac[i-1][1] =PartialY(i,xiV[0][0],xiV[1][0],xiV[2][0]);

Jac[i-1][2] =PartialZ(i,xiV[0][0],xiV[1][0],xiV[2][0]);

}

// creating function vector

vector<vector<double>> funcV(3,vector<double>(1,0));

funcV[0][0]= f1(xiV[0][0],xiV[1][0],xiV[2][0]);

funcV[1][0]= f2(xiV[0][0],xiV[1][0],xiV[2][0]);

funcV[2][0]= f3(xiV[0][0],xiV[1][0],xiV[2][0]);

// creating xi+1V

vector<vector<double>> xiplus1V(3,vector<double>(1,0));

// jacobinv

vector<vector<double>> jacobInv = inverseMatrix(Jac);

vector<vector<double>> JxV = JacIntoFuncV(jacobInv,funcV) ;

xiplus1V[0][0]= xiV[0][0]-JxV[0][0];

xiplus1V[1][0]= xiV[1][0]-JxV[1][0];

xiplus1V[2][0]= xiV[2][0]-JxV[2][0];

if(abs(xiplus1V[0][0]-xiV[0][0])<1e-04 and abs(xiplus1V[1][0]-xiV[1][0])<1e-04 and abs(xiplus1V[2][0]-xiV[2][0])<1e-04){

cout << "x = " << xiplus1V[0][0] <<endl ;

cout << "y = " << xiplus1V[1][0] <<endl ;

cout << "z = " << xiplus1V[2][0] <<endl ;

break;

}

else {

double a = sqrt(pow(xiplus1V[0][0],2)+pow(xiplus1V[1][0],2)+pow(xiplus1V[2][0],2));

double b = sqrt(pow(0.866025,2)+pow(0.5,2)+pow(1.73205,2));

double error = sqrt(pow((xiplus1V[0][0]-xiV[0][0]),2)+pow((xiplus1V[1][0]-xiV[1][0]),2)+pow((xiplus1V[2][0]-xiV[2][0]),2))/a;

double error2 = sqrt(pow((xiplus1V[0][0]-0.866025),2)+pow((xiplus1V[1][0]-0.5),2)+pow((xiplus1V[2][0]-1.73205),2))/b;

cout << "After iteration " << itr << " relative error is " << error\*100 << endl ;

cout << "After iteration " << itr << " true error is " << error2\*100 << endl ;

itr ++;

xiV[0][0] =xiplus1V[0][0];

xiV[1][0] = xiplus1V[1][0];

xiV[2][0] = xiplus1V[2][0];

}

}

}

int main() {

final();

}

Terminal:

