**ФГБОУ ВО**

**Уфимский государственный авиационный технический университет**

**Кафедра Информатики**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 90 |  |  |  |  |  |  |  |  |  |  |  |  |
| 80 |  |  |  |  |  |  |  |  |  |  |  |  |
| 70 |  |  |  |  |  |  |  |  |  |  |  |  |
| 60 |  |  |  |  |  |  |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |  |  |  |  |  |  |
| 40 |  |  |  |  |  |  |  |  |  |  |  |  |
| 30 |  |  |  |  |  |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

**Дисциплина:** Численные методы

**Отчет по лабораторной работе № 4**

**Тема:** «Решение нелинейных уравнений и их систем»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Группа |  |  | Фамилия И. О. | Подпись | Дата | Оценка |
| МКН-318Б |  |
|  |  |
| Студент | | | Халитова А.А. |  |  |  |
| Преподаватель | | | Феоктистов Б.А. |  |  |  |
|  | | |  |  |  |  |

**Уфа 2024 г.**

**Содержание**

[1 Введение 2](#_Toc181997553)

[2 Теоретическая часть 3](#_Toc181997554)

[2.1 Вычисление определенных интегралов по квадратурным формулам прямоугольников и трапеций на равномерной сетке. 3](#_Toc181997555)

[2.2 Вычисление определенного интеграла по квадратурной формуле Симпсона. 4](#_Toc181997556)

[2.3 Реализация процесса Эйткена, метода Рунге и правила Ромберга. 4](#_Toc181997557)

[2.4 Квадратурная формула Гаусса 6](#_Toc181997558)

[2.5 Вычисление интеграла методом Монте-Карло 8](#_Toc181997559)

[3 Практическая часть 9](#_Toc181997560)

[3.1 Задание 1. 9](#_Toc181997561)

[3.2 Задание 2 11](#_Toc181997562)

[3.3 Задание 3 12](#_Toc181997563)

[3.4 Задание 4 13](#_Toc181997564)

[3.5 Задание 5 14](#_Toc181997565)

[4 Заключение 18](#_Toc181997566)

[5 Список литературы 19](#_Toc181997567)

[6 Листинг программы 20](#_Toc181997568)

# Введение

Целью лабораторной работы является получение навыка численного решения нелинейных уравнений и систем таких уравнений.

# Теоретическая часть

## Различные способы решения нелинейных уравнений.

* Метод бисекции (дихотомии).

Пусть мы нашли такие точки, что т.е. на отрезке лежит не менее одного корня уравнения. Найдем середину отрезка и вычислим . Из двух половин отрезка выберем ту, для которой ибо один из корней лежит на этой половине. Затем новый отрезок опять делим пополам и выберем ту половину, на концах которой функция имеет разные знаки, и т.д.

Если требуется найти корень с точностью , то продолжаем деление пополам до тех пор, пока длина отрезка не станет меньше

* Метод хорд.

Суть метода хорд состоит в разбиении отрезка  (при условии ) на два отрезка с помощью хорды и выборе нового отрезка от точки пересечения хорды с осью абсцисс до неподвижной точки, на котором функция меняет знак и содержит решение, причём подвижная точка приближается к ε-окрестности решения.

Итерационная формула выглядит следующим образом

Построение хорд продолжается до достижения необходимой точности решения ε.

Метод простых итераций.

Исходное уравнение заменяется эквивалентным уравнением Далее выбирается некоторое нулевое приближение и дальнейшие приближения вычисляются по формуле

Для сходимости, функцию берут в виде причем функция 𝜏(𝑥) не меняет знака на том отрезке, где идет отыскание корня.

Данный метод сходится при надлежащем выборе начального приближения и если |𝑠′(𝑥)|<1, где x – корень уравнения.

* Метод касательных (Ньютона).

Этот метод также методом касательных или методом линеаризации. Если есть некоторое приближение к корню , а имеет непрерывную производную, то уравнение можно преобразовать следующим образом:

Приближенно заменяя на значение в известной точке , получим следующий итерационный процесс:

Геометрически этот процесс означает замену на каждой итерации графика касательной к нему.

* Метод секущих.

В методе Ньютона требуется вычислять производную функции, что не всегда удобно. Можно заменить производную первой разделенной разностью, найденной по двум последним итерациям, т.е. заменить касательную секущей. Тогда получим

Для начала процесса необходимо задать и . Такие процессы, где для вычисления очередного приближения надо знать два предыдущих, называют двухшаговыми.

## Решение системы двух нелинейных уравнений методом простых итераций

Систему нелинейных уравнений можно кратко записать в векторном виде

Или более подробно в координатном виде

Нулевое приближение в случае двух переменных можно найти графически: построить на плоскости кривые и и найти точки их пересечения.

Аналогично одномерному случаю метода простых итераций заменим нелинейную систему эквивалентной системой специального вида Выберем некоторое нулевое приближение и дальнейшие приближения найдем по формулам

или

Если итерации сходятся, то они сходятся к решению уравнения (предполагается, что решение существует).

Обозначим за Достаточным условием сходимости является

# Практическая часть

Задание 1 будет выполняться для уравнения (рисунок 1) на отрезке [0,2] с заданной точностью 10-5.

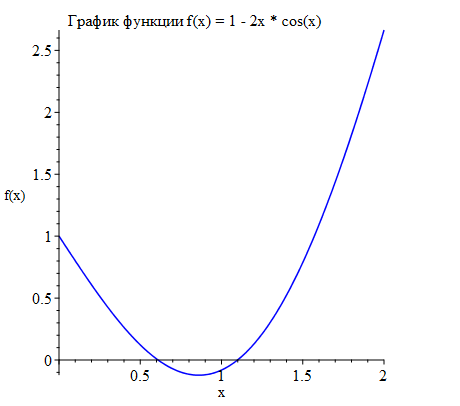
**

Рисунок 1 – график функции

## Задание 1.

1. Написать вычислительную программу на языке программирования C++ для решения нелинейного уравнения на указанном отрезке с заданной точностью методом

а) бисекции (дихотомии) (1 балл),

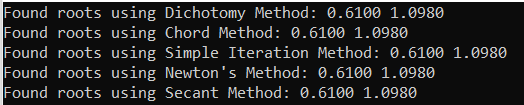
б) хорд (1 балл),

в) простых итераций (1 балл),

г) касательных (Ньютона) (1 балл),

д) секущих (1 балл).

1. Программа должна предусматривать возможность нахождения всех корней уравнения с заданной точностью.
2. С использованием написанной программы решить нелинейное уравнение согласно индивидуальному заданию (рисунок 2).
3. Выполнить сравнительный анализ реализованных методов.

  
Рисунок 2 – пример работы программы

**Вывод:** В ходе выполнения задания 1 были изучены основные принципы решения нелинейного уравнения на заданном отрезке с заданной точностью.

Метод бисекции заключается в последовательном делении отрезка пополам. Каждый новый отрезок также делится пополам. Этот процесс повторяется до нахождения корней. Метод надежен, но медленно работает.

Метод хорд использует прямую, соединяющую две точки на графике функции. Метод требует начальных приближений, которые должны быть выбраны так, чтобы функция имела разные знаки в этих точках. На каждой итерации вычисляется новая точка пересечения хорды с осью абсцисс, которая становится новым приближением к корню. Метод работает быстрее, чем метод бисекции, но не гарантирует нахождение корня, если начальные приближения выбраны неправильно. Метод хорд быстрее бисекции, но требует аккуратного выбора начальных значений.

Метод простых итераций основан на преобразовании уравнения к форме x=g(x). Итерационный процесс продолжается до тех пор, пока разность между последовательными приближениями не станет меньше заданной точности. Метод простых итераций: зависит от выбора функции g(x); может быть эффективным.

Метод Ньютона использует производную функции для нахождения корней. Начальное приближение используется для построения касательной к графику функции. Пересечение касательной с осью абсцисс дает новое приближение к корню. Процесс повторяется до достижения необходимой точности. Метод Ньютона очень быстро работает при хорошем начальном приближении, сложен в реализации из-за необходимости вычисления производной.

Метод секущих является обобщением метода хорд и использует две предыдущие точки для построения секущей линии. Этот метод также требует начальных приближений и может быть более эффективным, чем метод Ньютона в некоторых случаях. Метод секущих - хорошая альтернатива методу Ньютона, не требует производной, но также чувствителен к выбору начальных значений.

## Задание 2

Задание 2 будет выполняться для функции (рисунок 3)

на области определения поиска x × y [0,1] ×[0,1]

с абсолютной погрешностью 10-5.

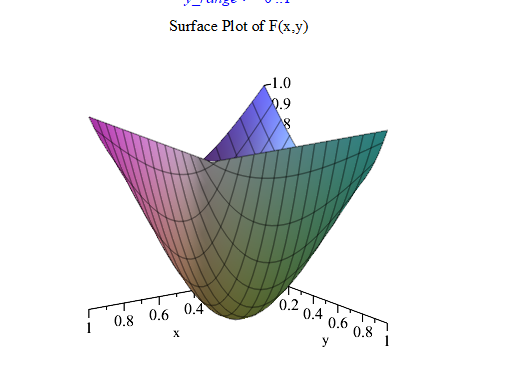


Рисунок 3 – график поверхности.

1. Написать вычислительную программу на языке программирования C++ для решения системы двух нелинейных уравнений методом простых итераций с заданной точностью.
2. С использованием написанной программы найти численно минимум заданной функции двух переменных в указанной области путем численного решения системы двух нелинейных уравнений, получающихся на основе необходимых условий экстремума.

Рисунок 4 - график зависимости абсолютной погрешности квадратурной формулы Симпсона

Рисунок 5 – выполнение программы

**Вывод:** В ходе выполнения задания 2 были изучены основные принципы вычисления определенного интеграла с помощью квадратурной формулы Симпсона. В результате анализа было определено оптимальное количество узлов равномерной сетки = 8, обеспечивающее вычисление интеграла с указанной величиной абсолютной погрешности = 10-6.

Построен график зависимости абсолютной погрешности квадратурной формулы от количества узлов.

# Заключение

В ходе выполнения лабораторной работы по численному интегрированию были решены несколько задач, связанных с применением различных квадратурных формул.

Были созданы программы на языке C++ для вычисления интегралов с использованием квадратурных формул.

Для каждой квадратурной формулы была построена зависимость абсолютной погрешности от количества узлов сетки.

Лабораторная работа позволила углубить понимание численных методов интегрирования и их применения в различных задачах.

# Список литературы

1. Бахвалов Н. С., Жидков Н. П., Кобельков Г. М. Численные методы. — М.: Физматлит, 2013. — 304 с.

2. Калиткин Н. Н. Численные методы. — М.: Издательский центр «Академия», 2013. — 304 с.

3. Самарский А. А. Введение в численные методы. — М.: Наука, 2001. — 256 с.

# Листинг программы

**First\_ex.cpp**

#include <iostream>

#include <cmath>

#include <vector>

#include <fstream>

#include "Constants.h"

using namespace std;

double f(double x) {

return (x \* x \* x) \* sqrt(1 - x \* x);

}

double rect\_method(int n) {

double h = (b - a) / n;

double integral = 0.0;

for (int i = 0; i < n; i+=1) {

double x\_i = a + (i + 0.5) \* h;

integral += f(x\_i);

}

integral \*= h;

return integral;

}

double trap\_method(int n) {

double h = (b - a) / n;

double integral = (f(a) + f(b)) / 2.0;

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

integral += f(a + i \* h);

}

integral \*= h;

return integral;

}

void find\_nodes\_for\_rect() {

int n = 1;

ofstream rect("rect\_first.txt");

while (true) {

double rect\_integral = rect\_method(n);

double error\_rect = fabs(rect\_integral - integral\_value);

rect << n << " " << error\_rect << endl;

//cout << n << " " << rect\_integral << " - " << integral\_value << " = " << error\_rect << endl;

if (error\_rect < delta) {

cout << "Nodes for rectangle method: " << n << ", Integral: " << rect\_integral << ", Error: " << error\_rect << endl;

rect.close();

return;

}

n += 1;

}

}

void find\_nodes\_for\_trap() {

int n = 1;

ofstream trap("trap\_first.txt");

while (true) {

double trap\_integral = trap\_method(n);

double error\_trap = fabs(trap\_integral - integral\_value);

//cout << n << " " << trap\_integral << " - " << integral\_value << " = " << error\_trap << endl;

trap << n << " " << error\_trap << endl;

if (error\_trap < delta) {

cout << "Nodes for trapezoidal method: " << n << ", Integral: " << trap\_integral << ", Error: " << error\_trap << endl;

trap.close();

return;

}

n += 1;

}

}

void first() {

cout << "\nRect method" << endl;

find\_nodes\_for\_rect();

cout << "\nTrap method" << endl;

find\_nodes\_for\_trap();

}

**Second\_ex.cpp**

#include <iostream>

#include <cmath>

#include <vector>

#include <fstream>

#include "Constants.h"

using namespace std;

double f(double x);

double simpson(int n) {

if (n % 2 != 0) {

n++;

}

double h = (b - a) / n;

double integral = f(a) + f(b);

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

double x\_i = a + i \* h;

if (i % 2 == 0) {

integral += 2 \* f(x\_i);

}

else {

integral += 4 \* f(x\_i);

}

}

integral \*= h / 3;

return integral;

}

void find\_nodes\_for\_simpson() {

ofstream simp("simp\_second.txt");

int n = 2;

while (true) {

double simpson\_integral = simpson(n);

double error\_simpson = fabs(simpson\_integral - integral\_value);

simp << n << " " << error\_simpson << endl;

if (error\_simpson < delta) {

cout << "Nodes for Simpson's method: " << n << ", Integral: " << simpson\_integral

<< ", Error: " << error\_simpson << endl;

simp.close();

return;

}

n += 2;

}

}

void second() {

cout << "\n Simpson method \n" << endl;

find\_nodes\_for\_simpson();

}

**Third\_ex.cpp**

#include <iostream>

#include <cmath>

#include <vector>

#include <fstream>

#include "Constants.h"

using namespace std;

double f(double x);

double trap\_method(int n);

double eitken(int n) {

double q = 0.5;

double f1 = trap\_method(n);

double f2 = trap\_method(n \* q);

double f3 = trap\_method(n \* q \* q);

double f\_end = f1 + (f1 - f2) \* (f1 - f2) / (2 \* f2 - f1 - f3);

return f\_end;

}

double order\_of\_accuracy\_eitken(int n) {

double q = 0.5;

double f1 = trap\_method(n);

double f2 = trap\_method(n \* q);

double f3 = trap\_method(n \* q \* q);

return (-1. / log(q)) \* log((f3 - f2) / (f2 - f1));

}

double Runge\_method(int n, double p) {

double q = 0.5;

double f1 = trap\_method(n);

double f2 = trap\_method(n \* q);

double sigma = pow(q, p) / (pow(q, p) - 1);

return sigma \* f1 + (1 - sigma) \* f2;

}

double Romberg\_method(int n, double delta) {

double error = 10000;

vector<vector<double>> R;

R.push\_back({ trap\_method(n) });

while (error > delta) {

vector<double> cur\_R(R.size() + 1);

cur\_R[0] = trap\_method(n);

for (int i = 1; i < cur\_R.size(); i++) {

cur\_R[i] = cur\_R[i - 1] + (1.0 / pow(4, i)) \* (cur\_R[i - 1] - R.back()[i - 1]);

}

R.push\_back(cur\_R);

n \*= 2;

error = fabs(integral\_value - R[R.size() - 1][R.size() - 1]);

}

return R[R.size() - 1][R.size() - 1];

}

void third() {

double eitken\_val = eitken(110);

double order\_of\_accurancy = order\_of\_accuracy\_eitken(110);

cout << "Eitken method integral = " << eitken\_val << " with error " << abs(eitken\_val - integral\_value)

<< " and order " << order\_of\_accurancy << endl;

double runge\_int = Runge\_method(110, order\_of\_accurancy);

cout << "Runge method integral = " << runge\_int << " with error " << abs(runge\_int - integral\_value) << endl;

double delta\_romberg = 0.000001;

double romberg\_int = Romberg\_method(110, delta\_romberg);

cout << "Romberg method integral = " << romberg\_int << " with error " << abs(romberg\_int - integral\_value) << endl;

}

**Fourth\_ex.cpp**

#include <iostream>

#include <cmath>

#include <vector>

#include <fstream>

#include "Constants.h"

using namespace std;

double f(double x);

double g(double t) {

double an = -1;

double bn = 1;

double alfa = (b - a) / (bn - an);

double beta = (a + b) / (bn - an);

return f(alfa \* t + beta);

}

vector<double> Roots(int n) {

switch (n) {

case 1:

return { 0 };

break;

case 2:

return { -0.577350269, 0.577350269 };

break;

case 3:

return { -0.774596669, 0, 0.774596669 };

break;

case 4:

return { -0.861136312, -0.339981044, 0.339981044, 0.861136312 };

break;

case 5:

return { -0.906179846, -0.538469310, 0, 0.538469310, 0.906179846 };

break;

case 6:

return { -0.932469514, -0.661209386, -0.238619186, 0.238619186, 0.661209386, 0.932469514 };

break;

default:

return {};

break;

}

}

vector<double> Weights(int n) {

switch (n) {

case 1:

return { 2 };

break;

case 2:

return { 1, 1 };

break;

case 3:

return { 0.5555556, 0.8888889, 0.5555556 };

break;

case 4:

return { 0.347854845, 0.652145155, 0.652145155, 0.347854845 };

break;

case 5:

return { 0.236926885, 0.478628670, 0.568888889, 0.478628670, 0.236926885 };

break;

case 6:

return { 0.171324492, 0.360761573, 0.467913935, 0.467913935, 0.360761573, 0.171324492 };

break;

default:

return {};

}

}

double Gauss\_method(int n) {

double integral = 0;

vector<double> r = Roots(n);

vector<double> w = Weights(n);

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

integral += w[i] \* g(r[i]);

}

return integral \*= (b - a) / 2;

}

void fourth() {

ofstream test("test.txt");

for (int i = 1; i <= 6; i++) {

double Gauss\_integral = Gauss\_method(i);

double error = abs(Gauss\_integral - integral\_value);

if (error < delta) {

cout << i << " " << Gauss\_integral << " " << error << endl;

}

test << i << " " << error << endl;

}

}

**Sixth\_ex.cpp**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <cmath>

#include <vector>

#include <fstream>

#include "Constants.h"

using namespace std;

double f(double x);

double monteCarloIntegration(int numPoints) {

double integral = 0.0;

double maxY = f(b);

double minY = f(a);

double lower\_line = 0;

double upper\_line = 0;

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

double x = (double)(rand()) / RAND\_MAX \* (b - a);

double y = (double)(rand()) / RAND\_MAX \* (maxY - minY);

if (y <= f(x)) {

lower\_line += 1.0;

}

else {

upper\_line += 1.0;

}

}

double Sk = b \* f(b) \* (lower\_line / numPoints);

return Sk;

}

void sixth() {

srand((time(0)));

const int sampleSize = 100;

vector<double> errors;

for (int points = 10; points <= 500; points += 10) {

double totalError = 0.0;

double totalIntegral = 0.0;

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

double integral = monteCarloIntegration(points);

totalIntegral += integral;

totalError += abs(integral - integral\_value);

}

double meanError = totalError / sampleSize;

errors.push\_back(meanError);

cout <<"Integral "<< totalIntegral /sampleSize<< " Count of nodes: " << points << " Math expectation " << meanError << endl;

}

ofstream error\_6("errors\_6.txt");

for (size\_t i = 0; i < errors.size(); ++i) {

error\_6 << (i + 1) \* 10 << " " << errors[i] << endl;

}

error\_6.close();

}

**Source.cpp**

void first();

void second();

void third();

void fourth();

void sixth();

int main() {

fourth();

}

**Constants.h**

#pragma once

#ifndef MATH\_CONSTANTS\_H

#define MATH\_CONSTANTS\_H

const double a = 0;

const double b = 0.5;

const double delta = 0.000001;

const double integral\_value = 0.01425484;

#endif // !MATH\_CONSTANTS\_H

Графики были построены на языке программирования python

**one\_plot.py**

import matplotlib.pyplot as plt  
  
def read\_from\_file():  
 data = []  
 with open("../cm\_lab2/test.txt", "r") as file:  
 #with open("../cm\_1/data.txt", "r") as file:  
 lines = file.readlines()  
 for line in lines:  
 x, y = line.split()  
 data.append((float(x), float(y)))  
 return data  
  
data = read\_from\_file()  
  
x\_values = []  
y\_values = []  
  
for point in data:  
 x\_values.append(point[0])  
 y\_values.append(point[1])  
  
plt.figure(figsize=(10, 10))  
  
plt.title('Oshibka Priblizheniya splines')  
plt.xlabel('X')  
plt.ylabel('Y')  
plt.axhline(0, color='black', linewidth=1, ls='-')  
plt.axvline(0, color='black', linewidth=1, ls='-')  
plt.grid(color='gray', linestyle='--', linewidth=0.5)  
  
plt.plot(x\_values, y\_values, label=f'Graph')  
  
plt.legend()  
plt.show()

**plot\_with\_count.py**

import matplotlib.pyplot as plt  
  
  
def read\_from\_file():  
 data = []  
 with open("../cm\_1/spline.txt", "r") as file:  
  
 lines = file.readlines()  
 i = 0  
 while i < len(lines):  
 N = int(lines[i].strip()) # Считываем количество точек  
 i += 1  
  
 points = []  
 for \_ in range(N):  
 x, y = map(float, lines[i].strip().split())  
 points.append((x, y))  
 i += 1  
  
 data.append(points)  
  
 return data  
  
  
plt.figure(figsize=(10, 10))  
data = read\_from\_file()  
  
for index, points in enumerate(data): # индексация  
 x\_values = [point[0] for point in points]  
 y\_values = [point[1] for point in points]  
  
 plt.plot(x\_values, y\_values, )  
 plt.plot(x\_values, y\_values, label=f'Graph {index + 1}') # для задания 1  
  
plt.title('Многочлены Лагранжа степени 1-15')  
  
plt.xlabel('X')  
plt.ylabel('Y')  
  
  
plt.axhline(0, color='black', linewidth=0.001, ls='-')  
plt.axvline(0, color='black', linewidth=0.001, ls='-')  
plt.grid(color='gray', linestyle='--', linewidth=0.5)  
  
plt.legend()  
plt.show()

**pogreshnost\_only.py**

import matplotlib.pyplot as plt  
  
  
def read\_from\_file():  
 data = []  
 with open("../cm\_lab2/test.txt", "r") as file:  
 lines = file.readlines()  
 for line in lines:  
 x, y = line.split()  
 data.append((float(x), float(y)))  
 return data  
  
  
data = read\_from\_file()  
  
x\_values = []  
y\_values = []  
  
for point in data:  
 x\_values.append(point[0])  
 y\_values.append(point[1])  
  
plt.figure(figsize=(10, 10))  
  
#plt.xscale('log')  
plt.yscale('log')  
  
plt.title('Deltas Gauss ')  
plt.xlabel('N')  
plt.ylabel('Deltas')  
plt.axhline(0, color='black', linewidth=1, ls='-')  
plt.axvline(0, color='black', linewidth=1, ls='-')  
plt.grid(color='gray', linestyle='--', linewidth=0.5)  
  
plt.plot(x\_values, y\_values, label='Graph', color='blue', marker='o')  
  
plt.legend()  
plt.show()

**monte-carlo.py**

import matplotlib.pyplot as plt  
import numpy as np  
  
def f(x):  
 return (x \*\* 3) \* ((1 - x \*\* 2) \*\* 0.5)  
  
def read\_from\_file():  
 data = []  
 with open("../cm\_lab2/Monte-carlo-500points.txt", "r") as file:  
 lines = file.readlines()  
 for line in lines:  
 x, y = line.split()  
 data.append((float(x), float(y)))  
 return data  
  
data = read\_from\_file()  
  
if not data:  
 print("No data to plot.")  
else:  
 x\_values = [point[0] for point in data]  
 y\_values = [point[1] for point in data]  
  
 plt.figure(figsize=(10, 10))  
  
 plt.title('Monte Carlo graph')  
 plt.xlabel('X')  
 plt.ylabel('Y')  
 plt.axhline(0, color='black', linewidth=1, ls='-')  
 plt.axvline(0, color='black', linewidth=1, ls='-')  
 plt.grid(color='gray', linestyle='--', linewidth=0.5)  
  
 for x, y in zip(x\_values, y\_values):  
 if y > f(x):  
 plt.scatter(x, y, color='red')  
 else:  
 plt.scatter(x, y, color='green')  
  
 x\_range = np.linspace(min(x\_values), max(x\_values), 100) # Create a range of x values  
 plt.plot(x\_range, [f(x) for x in x\_range], color='blue', label='f(x)', linewidth=2)  
  
 plt.legend()  
 plt.show()