**Анализ уравнений**

Я изучаю задачи, которые включают сравнение методов секущих и бисекции, выделение корней уравнений и построение графиков. Это требует точных вычислений и анализа функций.

1. Сначала я анализирую зависимость ошибки от номера итерации, устанавливая точность 10^-15 и максимальное количество шагов. Затем фиксирую отклонения и строю график зависимости.
2. Интересно, как ошибка меняется при различной точности, настраивая точность от 10^-15 до 10^-1 и фиксируя значения для каждой настройки.
3. Наконец, я проверяю зависимость относительной ошибки от изменений в исходных данных, анализируя влияние на точность решения.

**Изучая влияние**

Вношу случайные помехи в коэффициент уравнения (1-5%), высчитываю результаты и их отклонения, строю график на Python.

**Понимание задачи**

Пользователь хочет реализовать методы в C, работать с функциями, добавить разрыв и провести анализ с графиками в Python.

Уточняю аспекты: генерирую код на C, избегая прямого исполнения и ошибок, руководствуюсь полисами.

Implementation steps: Implementing methods in C, creating functions, and generating graphs in Python.

**Разделяю коды**

Я собираюсь написать код на C, разделяя главное, функции вычислений и функции f(x). Затем перехожу к анализу данных и рисованию графиков на Python, объясняя результаты.

**Создание функций**

Я пишу код на C, чтобы определить три функции f(x) и две методы для нахождения корней, используя бифуркацию и метод секущих.

**Оцениваю методы**

Я использую методы половинного деления и хорд для нахождения корней уравнения и сохраняю ошибки. Интересно, насколько точно это помогает.

**Создание и анализ**

В основной программе я использую методы для функции и интервала, накапливая погрешности в массив **errors** и записывая их в файлы. Затем, в Python, визуализирую данные на графике зависимости погрешности от номера итерации.

**Создание и анализ**

Я использую Python и Matplotlib для построения графиков функций f1, f2 и f3 на различных интервалах, чтобы визуально отделить корни функций.

**Анализируя погрешность**

Проверяю зависимость погрешности от точности, модифицируя основную программу и строя график, чтобы понять, как точность влияет на результат.

**Модифицирую функцию**

Я добавляю случайные возмущения в функцию **f1** и экспериментирую с эффектом на относительную погрешность решения на основе метода биссекции.

**Внос шума в данные**

Я добавляю функцию, которая вводит шум в исходные данные, обновляя коэффициент и добавляя случайный фактор к переменной **x**.

**Применение метода бисекции**

Использую метод бисекции для нахождения корней функции с шумом, обновляя интервал и коэффициенты на каждом шаге, пока не достигаются условия завершения.

**Визуализируя результаты**

Я создаю график зависимости относительной погрешности от возмущения, используя данные из файла с результатами.

**Выводы**

* **Отделение корней**: Графики функций позволили визуально определить приближенные интервалы, где находятся корни. При слишком большом шаге можно пропустить близко расположенные корни.
* **Зависимость погрешности от номера итерации**: Погрешность уменьшается с увеличением номера итерации. Метод половинного деления обеспечивает стабильное снижение ошибки, метод хорд может сходиться быстрее, но не всегда стабильно.
* **Зависимость погрешности от заданной точности**: Полученная погрешность не превышает заданную точность, что подтверждается графиком, где точки располагаются ниже биссектрисы.
* **Зависимость относительной погрешности от возмущения**: Относительная погрешность решения увеличивается с ростом возмущения исходных данных, что показывает чувствительность метода к изменениям в коэффициентах уравнения.

**Заключение**

Представленные реализации и исследования позволяют сравнить методы половинного деления и хорд, а также понять их поведение при различных условиях. Построенные графики наглядно демонстрируют полученные результаты и помогают сделать соответствующие выводы.

**1. Реализация методов на языке C**

**Файлы функций f(x)**

**functions.h**

#ifndef FUNCTIONS\_H

#define FUNCTIONS\_H

double f1(double x);

double f2(double x);

double f3(double x);

#endif

**functions.c**

#include <math.h>

#include "functions.h"

double f1(double x) {

return 0.1 \* x \* x - x \* log(x);

}

double f2(double x) {

return x \* x \* x \* x - 3 \* x \* x + 75 \* x - 9999;

}

// Функция с разрывом в точке x=2

double f3(double x) {

if (fabs(x - 2.0) < 1e-8) {

return 1e8; // Большое значение для моделирования разрыва

} else {

return (x - 2.0) \* (x + 3.0);

}

}

**Файлы методов расчета**

**methods.h**

#ifndef METHODS\_H

#define METHODS\_H

typedef double (\*FuncPtr)(double);

double bisection(FuncPtr f, double a, double b, double tol, int max\_iter, int \*iter\_count, double \*errors);

double secant(FuncPtr f, double x0, double x1, double tol, int max\_iter, int \*iter\_count, double \*errors);

#endif

**methods.c**

#include <math.h>

#include "methods.h"

double bisection(FuncPtr f, double a, double b, double tol, int max\_iter, int \*iter\_count, double \*errors) {

double fa = f(a);

double fb = f(b);

if (fa \* fb >= 0) {

return NAN; // Корень не отделен

}

double c, fc;

\*iter\_count = 0;

for (int i = 0; i < max\_iter; i++) {

c = (a + b) / 2.0;

fc = f(c);

errors[i] = fabs(fc);

\*iter\_count += 1;

if (fabs(fc) < tol || fabs(b - a) / 2.0 < tol) {

return c;

}

if (fa \* fc < 0) {

b = c;

fb = fc;

} else {

a = c;

fa = fc;

}

}

return c;

}

double secant(FuncPtr f, double x0, double x1, double tol, int max\_iter, int \*iter\_count, double \*errors) {

double f0 = f(x0);

double f1\_val = f(x1);

double x2, f2;

\*iter\_count = 0;

for (int i = 0; i < max\_iter; i++) {

if (fabs(f1\_val - f0) < 1e-15) {

return x1; // Избежание деления на ноль

}

x2 = x1 - f1\_val \* (x1 - x0) / (f1\_val - f0);

f2 = f(x2);

errors[i] = fabs(f2);

\*iter\_count += 1;

if (fabs(f2) < tol) {

return x2;

}

x0 = x1;

f0 = f1\_val;

x1 = x2;

f1\_val = f2;

}

return x1;

}

**Основная программа**

**main.c**

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <time.h>

#include "functions.h"

#include "methods.h"

#define MAX\_ITER 1000

int main() {

srand(time(NULL)); // Инициализация генератора случайных чисел

FuncPtr f = f1; // Выберите функцию f1, f2 или f3

double a = 1.0, b = 2.0; // Интервал для f1

double tol = 1e-15;

int iter\_count;

double errors[MAX\_ITER];

// 2) Отделение корней и построение графика функции

// (Построение графика выполняется в Python)

// 3) Зависимость погрешности от номера итерации

double root = bisection(f, a, b, tol, MAX\_ITER, &iter\_count, errors);

// Сохранение данных для построения графика

FILE \*file = fopen("errors\_bisection.txt", "w");

for (int i = 0; i < iter\_count; i++) {

fprintf(file, "%d %e\n", i + 1, errors[i]);

}

fclose(file);

// Аналогично для метода хорд

root = secant(f, a, b, tol, MAX\_ITER, &iter\_count, errors);

file = fopen("errors\_secant.txt", "w");

for (int i = 0; i < iter\_count; i++) {

fprintf(file, "%d %e\n", i + 1, errors[i]);

}

fclose(file);

// 4) Зависимость погрешности от заданной точности

double tolerances[] = {1e-15, 1e-14, 1e-13, 1e-12, 1e-11,

1e-10, 1e-9, 1e-8, 1e-7, 1e-6,

1e-5, 1e-4, 1e-3, 1e-2, 1e-1};

int num\_tol = sizeof(tolerances) / sizeof(tolerances[0]);

file = fopen("accuracy\_results.txt", "w");

for (int i = 0; i < num\_tol; i++) {

tol = tolerances[i];

root = bisection(f, a, b, tol, MAX\_ITER, &iter\_count, errors);

fprintf(file, "%e %e\n", tol, fabs(f(root)));

}

fclose(file);

// 5) Зависимость относительной погрешности от возмущения

double perturbations[] = {0.01, 0.02, 0.03, 0.04, 0.05};

int num\_perturbations = sizeof(perturbations) / sizeof(perturbations[0]);

double original\_root = bisection(f, a, b, 1e-15, MAX\_ITER, &iter\_count, errors);

file = fopen("perturbation\_results.txt", "w");

for (int i = 0; i < num\_perturbations; i++) {

double perturbation = perturbations[i];

// Вносим возмущение

double root\_perturbed = bisection(f, a \* (1 + perturbation), b \* (1 + perturbation), 1e-15, MAX\_ITER, &iter\_count, errors);

double relative\_error = fabs(root\_perturbed - original\_root) / fabs(original\_root);

fprintf(file, "%f %e\n", perturbation \* 100, relative\_error);

}

fclose(file);

return 0;

}

**2. Построение графиков функций и отделение корней**

**plot\_functions.py**

import numpy as np

import matplotlib.pyplot as plt

def f1(x):

return 0.1 \* x\*\*2 - x \* np.log(x)

def f2(x):

return x\*\*4 - 3 \* x\*\*2 + 75 \* x - 9999

def f3(x):

y = np.where(np.abs(x - 2.0) < 1e-8, np.nan, (x - 2.0) \* (x + 3.0))

return y

# График для f1

x = np.linspace(1, 2, 500)

plt.figure(figsize=(8, 6))

plt.plot(x, f1(x), label='f1(x)')

plt.axhline(0, color='black', linewidth=0.5)

plt.xlabel('x')

plt.ylabel('f1(x)')

plt.title('График функции f1(x)')

plt.legend()

plt.grid(True)

plt.show()

# График для f2

x = np.linspace(5, 10, 1000)

plt.figure(figsize=(8, 6))

plt.plot(x, f2(x), label='f2(x)')

plt.axhline(0, color='black', linewidth=0.5)

plt.xlabel('x')

plt.ylabel('f2(x)')

plt.title('График функции f2(x)')

plt.legend()

plt.grid(True)

plt.show()

# График для f3

x = np.linspace(0, 5, 1000)

plt.figure(figsize=(8, 6))

plt.plot(x, f3(x), label='f3(x)')

plt.axhline(0, color='black', linewidth=0.5)

plt.xlabel('x')

plt.ylabel('f3(x)')

plt.title('График функции f3(x) с разрывом')

plt.legend()

plt.grid(True)

plt.show()

**3. Зависимость погрешности от номера итерации**

**plot\_errors.py**

import matplotlib.pyplot as plt

# Метод половинного деления

iterations = []

errors = []

with open('errors\_bisection.txt', 'r') as file:

for line in file:

n, err = line.strip().split()

iterations.append(int(n))

errors.append(float(err))

plt.figure(figsize=(8,6))

plt.semilogy(iterations, errors, marker='o', label='Метод половинного деления')

plt.xlabel('Номер итерации')

plt.ylabel('Абсолютная ошибка')

plt.title('Погрешность от номера итерации (Метод половинного деления)')

plt.grid(True, which='both')

plt.legend()

plt.show()

# Метод хорд

iterations = []

errors = []

with open('errors\_secant.txt', 'r') as file:

for line in file:

n, err = line.strip().split()

iterations.append(int(n))

errors.append(float(err))

plt.figure(figsize=(8,6))

plt.semilogy(iterations, errors, marker='o', color='red', label='Метод хорд')

plt.xlabel('Номер итерации')

plt.ylabel('Абсолютная ошибка')

plt.title('Погрешность от номера итерации (Метод хорд)')

plt.grid(True, which='both')

plt.legend()

plt.show()

**4. Зависимость погрешности от заданной точности**

**plot\_accuracy.py**

import numpy as np

import matplotlib.pyplot as plt

tolerances = []

errors = []

with open('accuracy\_results.txt', 'r') as file:

for line in file:

tol, err = line.strip().split()

tolerances.append(float(tol))

errors.append(float(err))

plt.figure(figsize=(8,6))

plt.loglog(tolerances, errors, marker='o', label='Полученная погрешность')

plt.plot(tolerances, tolerances, linestyle='--', label='Биссектриса')

plt.xlabel('Заданная точность')

plt.ylabel('Полученная погрешность')

plt.title('Зависимость погрешности от заданной точности')

plt.grid(True, which='both')

plt.legend()

plt.show()

**5. Зависимость относительной погрешности решения от возмущения исходных данных**

**plot\_perturbation.py**

import matplotlib.pyplot as plt

perturbations = []

relative\_errors = []

with open('perturbation\_results.txt', 'r') as file:

for line in file:

p, err = line.strip().split()

perturbations.append(float(p))

relative\_errors.append(float(err))

plt.figure(figsize=(8,6))

plt.plot(perturbations, relative\_errors, marker='o')

plt.xlabel('Величина возмущения (%)')

plt.ylabel('Относительная погрешность')

plt.title('Зависимость относительной погрешности от возмущения')

plt.grid(True)

plt.show()