**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**"Уфимский университет науки и технологий"**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина:** Численные методы

**Отчет по лабораторной работе № 4**

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа ПМ-357 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Акмурзин М.Э. |  |  |  |
| Принял | Гайнетдинова А.А. |  |  |  |

**Уфа 2023**

**Цель работы:** получить навык численного решения нелинейных уравнений и систем таких уравнений.

**Теоретический материал**

***Задача 1. Решение нелинейного уравнения разными методами***

***Метод биекции (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.

***Задача 2. 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

***Задача 3. Решение системы двух нелинейных уравнений методом Ньютона***

Пусть известно некоторое приближение к корню Как и для одной переменной, запишем исходную систему в виде где Разлагая эти уравнения в ряды и ограничиваясь первыми дифференциалами, т.е. линеаризуя функцию, получим

Это система уравнений, линейных относительно приращений Все коэффициенты этой матрицы выражаются через последнее приближение Решив эту систему (например, методом исключения), найдем новое приближение

Как и для одной переменной, метод Ньютона можно формально свести к методу последовательных приближений, положив где есть матрица, обратная матрице производных.

Критерий окончания

**Практическая часть**

***Задача 1.***

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

а) биекции (дихотомии) (1 балл),

б) хорд (1 балл),

в) простых итераций (1 балл),

г) касательных (Ньютона) (1 балл),

д) секущих (1 балл).

Программа должна предусматривать возможность нахождения всех корней уравнения с заданной точностью.

1. С использованием написанной программы решить нелинейное уравнение согласно индивидуальному заданию.
2. Выполнить сравнительный анализ реализованных методов.

Уравнение:

Описание: реализованная программа позволяет найти корни заданного нелинейного уравнения на указанном отрезке с заданной точностью пятью различными методами. Для наглядности рассмотрим график данной функции на заданном отрезке.



Рисунок 1.График функции

то есть у данной функции на заданном отрезке имеется четыре решения уравнения.

Результат:

Первый вариант предоставляет возможность нахождения всех корней нелинейного уравнения на указанном отрезке методом биекции. Программа делит отрезок на подотрезки (необходимое их количество определяется вручную) и для каждого применяет данный метод, проверяя наличие корня. Если корня не существует, то программа выводит сообщение о том, что на данном подотрезке корня не существует. Если же решение существует, то применяет метод для получения приближенного значения корня уравнения с заданной точностью.

Второй вариант программы позволяет найти все корни нелинейного уравнения на указанном отрезке методом Ньютона(касательных). Программа по аналогии с первым и вторым вариантом работы делит отрезок на подотрезки и для каждого применяет данный метод, проверяя наличие корня.

Третий вариант предоставляет возможность нахождения всех корней нелинейного уравнения на указанном отрезке методом секущих. Схема работы происходит по аналогии с первым методом.

Четвертый вариант предоставляет возможность нахождения всех корней нелинейного уравнения на указанном отрезке методом хорд. Схема работы происходит по аналогии с первым методом.

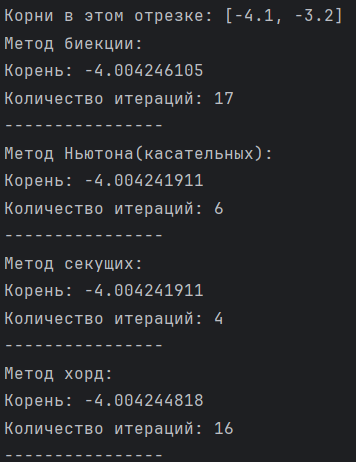


Рисунок 2. Пример выполнения программы

***Задача 2.***

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.

где искать в области с заданной точностью 10-7.



Рисунок 3. Пример выполнения программы 2

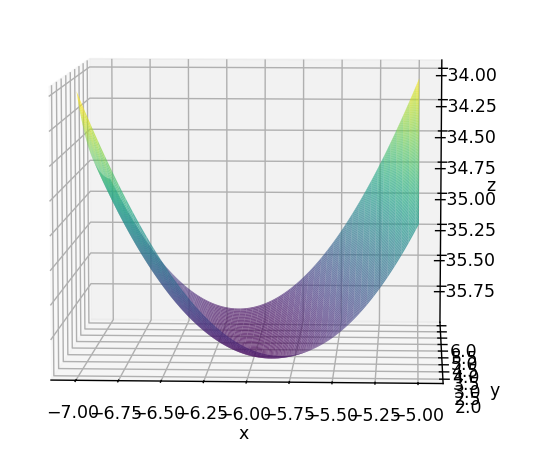


Рисунок 4. График функции по x

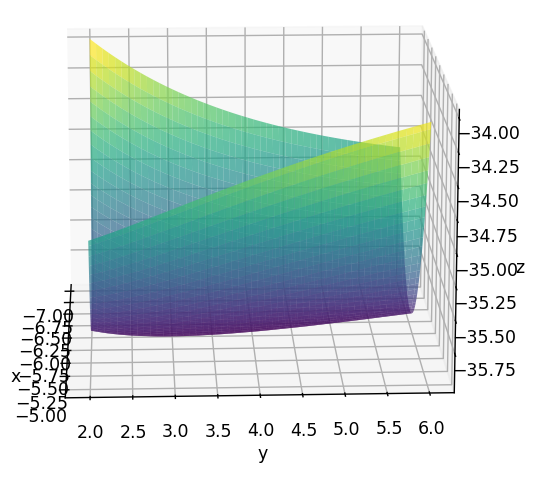


Рисунок 5. График функции по Y

***Задача 3***

1. Написать вычислительную программу на языке программирования C++ для решения системы двух нелинейных уравнений методом Ньютона. При этом предусмотреть две возможности: а) точное задание всех производных, б) приближенное вычисление производных по точно заданным функциям с заданной точностью.
2. С использованием написанной программы решить задачу о поиске минимума функции двух переменных *F*(*x*,*y*) сведением к системе двух нелинейных уравнений с использованием необходимого условия экстремума. Выполнить сравнительный анализ двух указанных в п.1) реализаций метода.

Написана программа для решения системы двух нелинейных уравнений методом Ньютона.

****

Рисунок 6. Результаты нахождения минимума методом Ньютона и его значение.

**Вывод**

В ходе лабораторной работы были получены навыки численного решения нелинейных уравнений и систем таких уравнений.

**Список литературы**

1. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы.
2. Калиткин Н.Н. Численные методы.

**Приложение**

Main.cpp

#include "task.h"

int main() {

SetConsoleOutputCP(CP\_UTF8);

task1::main\_task1();

cout << "Методом простых итераций:\n";

task2::main\_task2();

cout << "Методом Ньютона:\n";

task3::main\_task3();

system("python ../src/check\_minimum.py");

return 0;

}

Task.h

#include <iostream>

#include <vector>

#include <windows.h>

#include <math.h>

#include <iomanip>

using namespace std;

namespace task1 {

const double a = -5, b = 5, epsilon = 1e-5;

double foo\_task1(double x);

double foo\_task1\_diff(double x);

double x1\_foo\_task1(double x);

double x2\_foo\_task2(double x);

double x1\_foo\_task1\_diff(double x);

double x2\_foo\_task2\_diff(double x);

void BIM(double a, double b, int &num\_it);

void NEM(double a, double b, int &num\_it);

void SEM(double a, double b, int &num\_it);

void CHM(double a, double b, int &num\_it);

void SIM(double a, int &n\_num);

void main\_task1();

}

namespace task2 {

double f(double x, double y);

double diff\_x(double x, double y);

double diff\_y(double x, const double y);

double phi\_x(double x, double y);

double phi\_y(double x, double y);

pair<double, double> SimpleIterations(double x\_1, double x\_2, double y\_1, double y\_2);

void main\_task2();

}

namespace task3 {

vector<double> Gauss(std::vector<std::vector<double>> matrix);

double df\_x\_x(double x, double y);

double df\_x\_y(double x, double y);

double df\_y\_x(double x, double y);

double df\_y\_y(double x, double y);

pair<double, double> Newton(double x\_1, double x\_2, double y\_1, double y\_2);

void main\_task3();

}

Task1.cpp

#include "task.h"

double task1::foo\_task1(double x) {

return 20 \* log(2 + cos(x)) - x - 10;

}

double task1::foo\_task1\_diff(double x) {

return (-20 \* sin(x)) / (2 + cos(x)) - 1;

}

double task1::x1\_foo\_task1(double x) {

return 20 \* log(2 + cos(x)) - 10;

}

double task1::x2\_foo\_task2(double x) {

return acos(exp((x / 20.0 + 0.5)) - 2);

}

double task1::x1\_foo\_task1\_diff(double x) {

return (-20 \* sin(x)) / (2 + cos(x));

}

double task1::x2\_foo\_task2\_diff(double x) {

return (-exp(x / 20.0 + 0.5) / (20 \* sqrt(-(exp(x / 20.0 + 0.5) - 2) \* (exp(x / 20.0 + 0.5) - 2) + 1)));

}

void task1::main\_task1() {

vector<double> answer = {-4.00424191074621, -2.11371014988160, 1.77077248495832, 4.80946952510987};

int n\_bim = 0, n\_nem = 0, n\_sem = 0, n\_chm = 0, n\_sim = 0;

int N = 11;

double h = (b - a) / N;

double x\_cur = a;

while (x\_cur < b) {

double x0 = x\_cur;

double x1 = x\_cur + h;

cout << setprecision(2) << "Корни в этом отрезке: [" << x0 << ", " << x1 << "]" << endl;

BIM(x0, x1, n\_bim);

NEM(x0, x1, n\_nem);

SEM(x0, x1, n\_sem);

CHM(x0, x1, n\_chm);

x\_cur += h;

}

cout << "Корни:\n";

for (auto i: answer) {

cout << i << " ";

}

cout << endl;

}

void task1::BIM(double a, double b, int &num\_it) {

double c = (a + b) / 2.0;

if ((b - a) < 2 \* epsilon) {

cout << "Метод биекции:" << endl;

cout << "Корень: " << fixed << setprecision(9) << c << endl;

cout << "Количество итераций: " << num\_it << endl;

cout << "----------------\n";

return;

} else {

num\_it++;

if (foo\_task1(a) \* foo\_task1(c) <= 0) {

BIM(a, c, num\_it);

} else if (foo\_task1(c) \* foo\_task1(b) < 0) {

BIM(c, b, num\_it);

}

}

}

// Метод касательных(Ньютона)

void task1::NEM(double a, double b, int &num\_it) {

double x0 = a;

double x1 = x0 - foo\_task1(x0) / foo\_task1\_diff(x0);

// что-то похожее на градиентный спуск, х1 будет постоянно двигаться, пока не сойдется

while (abs(x1 - x0) > epsilon) {

x0 = x1;

x1 = x0 - foo\_task1(x0) / foo\_task1\_diff(x0);

num\_it++;

}

if (a <= x1 && x1 <= b) {

cout << "Метод Ньютона(касательных):" << endl;

cout << "Корень: " << fixed << setprecision(9) << x1 << endl;

cout << "Количество итераций: " << num\_it << endl;

cout << "----------------\n";

}

}

// Метод секущих

void task1::SEM(double a, double b, int &num\_it) {

int count\_itr = 0;

double x0 = a;

double x1 = b;

double x2 = x1 - foo\_task1(x1) \* (x1 - x0) / (foo\_task1(x1) - foo\_task1(x0));

// отличается лишь точным вычислением производной от ньютона

while (abs(x2 - x1) > epsilon) {

x0 = x1;

x1 = x2;

x2 = x1 - foo\_task1(x1) \* (x1 - x0) / (foo\_task1(x1) - foo\_task1(x0));

count\_itr++;

}

if (a <= x2 && x2 <= b) {

cout << "Метод секущих:" << endl;

cout << "Корень: " << fixed << setprecision(9) << x2 << endl;

cout << "Количество итераций: " << count\_itr << endl;

cout << "----------------\n";

}

}

// Метод хорд

void task1::CHM(double a, double b, int &num\_it) {

double x0 = a;

double x1 = x0 - foo\_task1(x0) \* (x0 - b) / (foo\_task1(x0) - foo\_task1(b));

// здесь мы как бы строим хорды от фиксированного конца б

while (abs(x1 - x0) > epsilon) {

x0 = x1;

x1 = x0 - foo\_task1(x0) \* (x0 - b) / (foo\_task1(x0) - foo\_task1(b));

num\_it++;

}

if (a <= x1 && x1 <= b) {

cout << "Метод хорд:" << endl;

cout << "Корень: " << fixed << setprecision(9) << x1 << endl;

cout << "Количество итераций: " << num\_it << endl;

cout << "----------------\n";

}

}

//// Метод простых итераций

//void task1::SIM(double a, int &n\_num) {

// double x0 = a;

// double x1 = x2\_foo\_task2(x0);

// while (abs(x1 - x0) > epsilon) {

// x0 = x1;

// x1 = x2\_foo\_task2(x0);

// n\_num++;

// }

// cout << "Метод простых итераций:" << endl;

// cout << "Корень: " << x1 << fixed << setprecision(9) << endl;

// cout << "Количество итераций: " << n\_num << endl;

// cout << "----------------\n";

// } else if ((dF2(a) < 1) && (dF2(b) < 1)) {

// int num\_it = 0;

// double x0 = b;

// double x1 = F2(x0);

// // смотрим пока разница вида f(f....(n-1)...f(b)...) f(f....(n)...f(b)...) не сойдется

// while (abs(x1 - x0) > delta) {

// x0 = x1;

// x1 = F2(x0);

// num\_it++;

// }

// cout << "Метод простых итераций:" << endl;

// cout << "f(x) = 6/x-2/x^2-1/x^3" << endl;

// cout << "Корень: " << fixed << setprecision(9) << x1 << endl;

// cout << "Количество итераций: " << num\_it << endl;

// cout << "----------------\n";

// } else if ((dF3(a) < 1) && (dF3(b) < 1)) {

// int num\_it = 0;

// double x0 = b;

// double x1 = F3(x0);

// // смотрим пока разница вида f(f....(n-1)...f(b)...) f(f....(n)...f(b)...) не сойдется

// while (abs(x1 - x0) > delta) {

// x0 = x1;

// x1 = F3(x0);

// num\_it++;

// }

// cout << "Метод простых итераций:" << endl;

// cout << "f(x) = x^3/6+1/3+1/6x" << endl;

// cout << "Корень: " << fixed << setprecision(9) << x1 << endl;

// cout << "Количество итераций: " << num\_it << endl;

// cout << "----------------\n";

// }

//}

Task2.cpp

#include "task.h"

namespace task2 {

const double a = -7, b = -5, c = 2, d = 6, epsilon = 1e-7;

}

double task2::f(double x, double y) {

return 1 - sin(x + log(y)) + x \* x + 12 \* x;

}

double task2::diff\_x(double x, double y) {

return -cos(x + log(y)) + 2 \* x + 12;

}

double task2::diff\_y(double x, const double y) {

return -cos(x + log(y)) / y;

}

double task2::phi\_x(double x, double y) {

return x - 1. / (sin(-6 + log(y)) + 2) \* diff\_x(x, y);

}

double task2::phi\_y(double x, double y) {

return y - 16. / (cos(x + log(4)) + sin(x + log(4))) \* diff\_y(x, y);

}

pair<double, double> task2::SimpleIterations(double x\_1, double x\_2, double y\_1, double y\_2) {

if (x\_1 > x\_2) {

std::swap(x\_1, x\_2);

}

if (y\_1 > y\_2) {

std::swap(y\_1, y\_2);

}

double x = (x\_1 + x\_2) / 2;

double y = (y\_1 + y\_2) / 2;

double x\_next = phi\_x(x, y);

double y\_next = phi\_y(x, y);

while (std::max(abs(x\_next - x), abs(y\_next - y)) > epsilon) {

x = x\_next;

y = y\_next;

x\_next = phi\_x(x, y);

y\_next = phi\_y(x, y);

}

return {x\_next, y\_next};

}

void task2::main\_task2(){

std::pair<double, double> min = SimpleIterations(a, b, c, d);

std::cout << "x = " << min.first << ", y = " << min.second << ", f(x,y) = " << f(min.first, min.second)

<< std::endl;

}

Task3.cpp

#include "task.h"

namespace task3 {

const double a = -7, b = -5, c = 2, d = 6, epsilon = 1e-7;

}

double task3::df\_x\_x(double x, double y) {

return sin(x + log(y)) + 2;

}

double task3::df\_x\_y(double x, double y) {

return sin(x + log(y)) / y;

}

double task3::df\_y\_x(double x, double y) {

return sin(x + log(y)) / y;

}

double task3::df\_y\_y(double x, double y) {

return (cos(x + log(y)) + sin(x + log(y))) / (y \* y);

}

pair<double, double> task3::Newton(double x\_1, double x\_2, double y\_1, double y\_2) {

if (x\_1 > x\_2) {

std::swap(x\_1, x\_2);

}

if (y\_1 > y\_2) {

std::swap(y\_1, y\_2);

}

double x = (x\_1 + x\_2) / 2;

double y = (y\_1 + y\_2) / 2;

std::vector<double> answer;

std::vector<std::vector<double>> matrix(2);

matrix[0].resize(3);

matrix[1].resize(3);

matrix[0][0] = df\_x\_x(x, y);

matrix[0][1] = df\_x\_y(x, y);

matrix[1][0] = df\_y\_x(x, y);

matrix[1][1] = df\_y\_y(x, y);

matrix[0][2] = -task2::diff\_x(x, y);

matrix[1][2] = -task2::diff\_y(x, y);

answer = Gauss(matrix);

double x\_next = x + answer[0];

double y\_next = y + answer[1];

while (std::max(abs(answer[0]), abs(answer[1])) > epsilon) {

x = x\_next;

y = y\_next;

matrix[0][0] = df\_x\_x(x, y);

matrix[0][1] = df\_x\_y(x, y);

matrix[1][0] = df\_y\_x(x, y);

matrix[1][1] = df\_y\_y(x, y);

matrix[0][2] = -task2::diff\_x(x, y);

matrix[1][2] = -task2::diff\_y(x, y);

answer = Gauss(matrix);

x\_next = x + answer[0];

y\_next = y + answer[1];

}

return {x\_next, y\_next};

}

void task3::main\_task3() {

std::pair<double, double> min = Newton(a, b, c, d);

std::cout << "x = " << min.first << ", y = " << min.second << ", f(x,y) = " << task2::f(min.first, min.second)

<< std::endl;

}

std::vector<double> task3::Gauss(std::vector<std::vector<double>> matrix) {

int dim = matrix.size();

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

for (int k = 0; k < dim - 1; ++k) {

double max = 0;

double m\_index = 0;

for (int m = k; m < dim; ++m) {

if (abs(matrix[m][k]) > max) {

max = abs(matrix[m][k]);

m\_index = m;

}

}

if (max == 0) {

std::cout << "There is no single solution" << std::endl;

return {-1};

}

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

double temp = matrix[k][i];

matrix[k][i] = matrix[m\_index][i];

matrix[m\_index][i] = temp;

}

}

//std::cout << "The result of the shuffle of rows: " << std::endl;

//PrintMatrix(matrix);

//����������

for (int j = i + 1; j < dim; ++j) {

double coeff = -matrix[j][i] / matrix[i][i];

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

matrix[j][k] += coeff \* matrix[i][k];

}

}

//PrintMatrix(matrix);

}

//std::cout << "The result of a direct process: " << std::endl;

std::vector<double> answer(dim);

answer[dim - 1] = matrix[dim - 1][dim] / matrix[dim - 1][dim - 1];

for (int i = dim - 2; i >= 0; --i) {

answer[i] = matrix[i][dim];

for (int j = i + 1; j < dim; ++j) {

answer[i] -= matrix[i][j] \* answer[j];

}

answer[i] /= matrix[i][i];

}

return answer;

}