Университет ИТМО

Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.04 Программная инженерия
Дисциплина «Вычислительная математика»

Отчет
По лабораторной работе №1
Вариант 7

Выполнил:
Зинатулин А.В.
P32121

Преподаватель:
Наумова Н.А.

Санкт-Петербург, 2023 г.

Цель работы

изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов.

Порядок выполнения лабораторной работы

Лабораторная работа состоит из двух частей: вычислительной и программной.

1 Вычислительная реализация задачи:

Задание:

1. Отделить корни заданного нелинейного уравнения графически
2. Определить интервалы изоляции корней.
3. Уточнить корни нелинейного уравнения (см. табл. 6) с точностью ε=10-2.
4. Используемые методы для уточнения каждого из 3-х корней многочлена представлены в таблице 7.
5. Вычисления оформить в виде таблиц (1-5), в зависимости от заданного метода. Для всех значений в таблице удержать 3 знака после запятой.

5.1 Для метода половинного деления заполнить таблицу 1.
5.2 Для метода хорд заполнить таблицу 2.
5.3 Для метода Ньютона заполнить таблицу 3.
5.4 Для метода секущих заполнить таблицу 4.
5.5 Для метода простой итерации заполнить таблицу 5.

6. Заполненные таблицы отобразить в отчете.

2 Программная реализация задачи:

Для нелинейных уравнений:

1. Все численные методы (см. табл. 8) должны быть реализованы в виде отдельных подпрограмм/методов/классов.
2. Пользователь выбирает уравнение, корень/корни которого требуется вычислить (3-5 функций, в том числе и трансцендентные), из тех, которые предлагает программа.
3. Предусмотреть ввод исходных данных (границы интервала/начальное приближение к корню и погрешность вычисления) из файла или с клавиатуры по выбору конечного пользователя.
4. Выполнить верификацию исходных данных. Необходимо анализировать наличие корня на введенном интервале. Если на интервале несколько корней или они отсутствуют – выдавать соответствующее сообщение. Программа должна реагировать на некорректные введенные данные.
5. Для методов, требующих начальное приближение к корню (методы Ньютона, секущих, хорд с фиксированным концом), выбор начального приближения (a или b) вычислять в программе.
6. Для метода простой итерации проверять достаточное условие сходимости метода на введенном интервале.
7. Предусмотреть вывод результатов (найденный корень уравнения, значение функции в корне, число итераций) в файл или на экран по выбору конечного пользователя.
8. Организовать вывод графика функции, график должен полностью отображать весь исследуемый интервал (с запасом).

Для систем нелинейных уравнений:

1. Пользователь выбирает предлагаемые программой системы двух нелинейных уравнений (2-3 системы).
2. Организовать вывод графика функций.
3. Начальные приближения ввести с клавиатуры.
4. Для метода простой итерации проверить достаточное условие сходимости.
5. Организовать вывод вектора неизвестных: $x1$, $x2$.
6. Организовать вывод количества итераций, за которое было найдено решение.
7. Организовать вывод вектора погрешностей: $|xi(k) - xi(k{-}1)|$
8. Проверить правильность решения системы нелинейных уравнений

Рабочие формулы методов

1) Метод половинного деления

$$x_i = \frac{a_i + b_i}{2}$$

2) Метод хорд

$$x_i = \frac{a_i f(b_i) - b_i f(a_i)}{f(b_i) - f(a_i)}$$

3) Метод касательных

$$x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}$$

4) Метод секущих

$$x_i = x_{i-1} - \frac{(x_{i-1} - x_{i-2}) f(x_{i-1})}{f(x_{i-1}) - f(x_{i-2})}$$

5) Метод простой итерации

$$x_i = \varphi(x_{i-1})$$

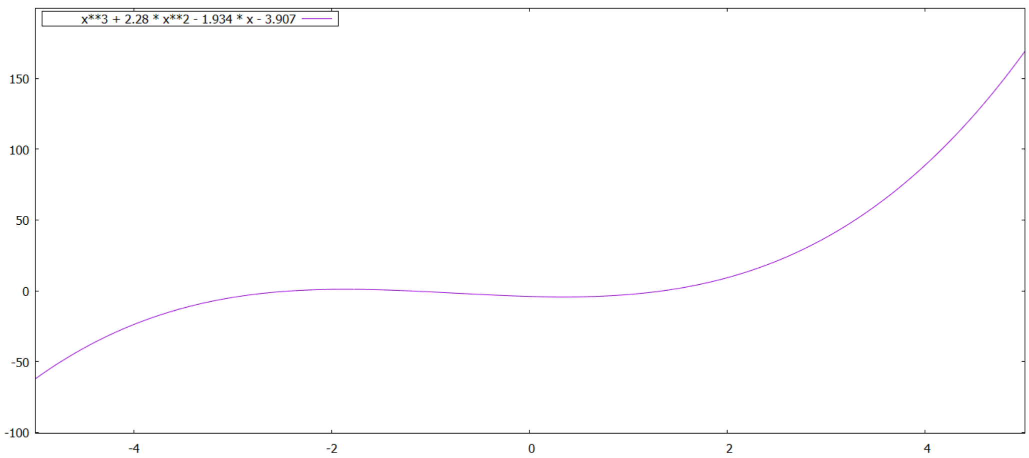
6) Метод Ньютона

$$J(x_1, x_2, \dots, x_n) \begin{pmatrix} \Delta x_1 \\ \dots \\ \Delta x_n \end{pmatrix} = \begin{pmatrix} -\mathrm{F}_1(x_1, x_2, \dots, x_n) \\ \dots \\ -\mathrm{F}_\mathrm{n}(x_1, x_2, \dots, x_n) \end{pmatrix}$$

7) Метод простой итерации

$$X_i = \varphi(X_{i-1})$$

График функции на исследуемом интервале



Вычислительная часть лабораторной работы

Таблица 1 - Крайний левый корень – метод половинного деления

№	a	b	x	F(a)	F(b)	F(x)	| a – b |
1	-3	-2	-2.5	-4.585	1.081	-0.447	1
2	-2.5	-2	-2.25	-0.447	1.081	0.596	0.5
3	-2.5	-2.25	-2.375	-0.447	0.596	0.150	0.25
4	-2.5	-2.375	-2.436	-0.447	0.150	-0.128	0.125
5	-2.436	-2.735	-2.406	-0.129	0.150	0.016	0.063
6	-2.438	-2.406	-2.421	-0.129	0.016	-0.552	0.031
7	-2.422	-2.406	-2.414	-0.055	0.016	-0.019	0.016
8	-2.414	-2.406	-2.410	-0.020	0.016	-0.002	0.008

Таблица 2 – средний корень – метод простой итерации

№	x_i	x_{i+1}	$f(x_i + x_{i+1})$	$\|f(x_{i+1}) - f(x_i)\|$
1	-1.4	-1.137	-0.229	0.263
2	-1.137	-1.252	0.126	0.115
3	-1.252	-1.189	-0.065	0.063
4	-1.189	-1.221	-0.035	0.032
5	-1.221	-1.204	-0.018	0.017
6	-1.204	-1.213	-0.010	0.009

Таблица 3 – крайний правый корень – метод хорд

№	a	b	x	F(a)	F(b)	F(x)	$\|x_{k+1} - x_k\|$
1	1.2	1.4	1.334	-1.217	0.59	-0.056	
2	1.334	1.4	1.340	-0.056	0.59	0.002	0.06

Код программы

https://github.com/uvuv-643/Computational_Mathematics

Листинг кода:

```
//
// Created by artem on 23.02.2023.
//

#include "CHalfDividingMethod.h"


enum MethodResult CHalfDividingMethod::validateBorder(CFunctionSV* function_data, float
border_left, float border_right) {
    if (function_data->f(border_left) * function_data->f(border_right) > 0) {
        return WRONG_NUMBER_OF_SOLUTIONS;
    }
    float dx = (border_right - border_left) / DELTA;
    bool derivative_sign = function_data->f_derivative(border_left) > 0;
    for (float current_point = border_left; current_point <= border_right;
current_point += dx) {
        bool derivative_xi_sign = function_data->f_derivative(current_point) > 0;
        if (derivative_sign != derivative_xi_sign) {
            return DERIVATIVE_MUST_BE_SAME_SIGN;
        }
    }
    return METHOD_CAN_BE_APPLIED;
}

CHalfDividingResult CHalfDividingMethod::performMethod(CFunctionSV* function_data,
float initial_border_left, float initial_border_right, float eps) {
```

```
    CFloat border_left = initial_border_left;
    CFloat border_right = initial_border_right;
    SingleFunctionMethodData initial_data(function_data, border_left, border_right,
eps);
    MethodResult validation_result = validateBorder(function_data, border_left,
border_right);
    CHalfDividingResult result = *new CHalfDividingResult(validation_result,
initial_data);
    if (result.getMethodResult() == METHOD_CAN_BE_APPLIED) {
        do {
            CFloat new_border = (border_right + border_left) / 2;
            result.append(new_border, border_left, border_right);
            if (function_data->f(border_left) * function_data->f(new_border) < 0) {
                border_right = new_border;
            } else if (function_data->f(border_right) * function_data->f(new_border) <
0) {
                border_left = new_border;
            } else if (function_data->f(new_border) == 0) {
                return result;
            } else {
                return *new CHalfDividingResult(WRONG_NUMBER_OF_SOLUTIONS,
initial_data);
            }
        } while (result.getCountOfIterations() < LIMIT_OF_ITERATIONS && 2 *
(border_right - border_left) > eps);
        result.setMethodResult(METHOD_WAS_SUCCESSFULLY_FINISHED);
    }
    return result;
}
```

```
//
// Created by artem on 24.02.2023.
//

#include "CIterationsMethod.h"

enum MethodResult CIterationsMethod::validateBorder(CFunctionSV* function_data, float
border_left, float border_right) {
    if (function_data->f(border_left) * function_data->f(border_right) > 0) {
        return WRONG_NUMBER_OF_SOLUTIONS;
    }
    float dx = (border_right - border_left) / DELTA;
    for (float current_point = border_left; current_point <= border_right;
current_point += dx) {
        if (abs(function_data->phi_derivative(current_point)) >= 1) {
            return LIPSCHITZ_CONSTANT_GREATER_THAN_ONE;
        }
    }
    return METHOD_CAN_BE_APPLIED;
}

CIterationsResult CIterationsMethod::performMethod(CFunctionSV* function_data, float
initial_border_left, float initial_border_right, float eps) {
    CFloat border_left = initial_border_left;
    CFloat border_right = initial_border_right;
    SingleFunctionMethodData initial_data(function_data, border_left, border_right,
eps);
    MethodResult validation_result = validateBorder(function_data, border_left,
border_right);
    CIterationsResult result = *new CIterationsResult(validation_result, initial_data);
    if (result.getMethodResult() == METHOD_CAN_BE_APPLIED) {
        vector<CFloat> xs;
        xs.push_back(border_left);
        for (size_t current_iteration = 0; current_iteration < LIMIT_OF_ITERATIONS;
current_iteration++) {
            CFloat x_prev = xs[current_iteration];
            CFloat x_current = function_data->phi(x_prev);
```

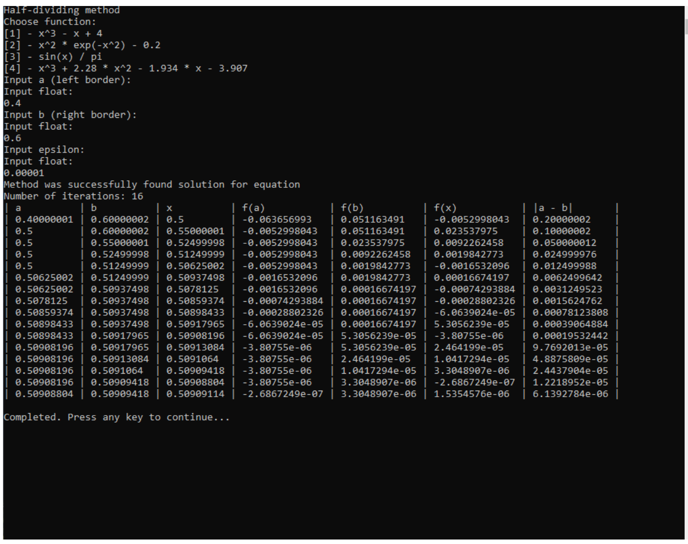
```
            xs.push_back(x_current);
            if (abs(function_data->f(x_current)) < eps) break;
        }
        for (size_t i = 0; i < xs.size() - 1; i++) {
            result.append(xs[i], xs[i + 1]);
        }
        result.setMethodResult(METHOD_WAS_SUCCESSFULLY_FINISHED);
    }
    return result;
}
```

```
//
// Created by artem on 26.02.2023.
//

#include "CNewtonMethod.h"

CNewtonResult CNewtonMethod::performMethod(CFunctionMV* f, CFunctionMV* g, float
initial_x, float initial_y, float eps) {
    CFloat x = initial_x;
    CFloat y = initial_y;
    MultipleFunctionMethodData initial_data(f, g, x, y, eps);
    CNewtonResult result = *new CNewtonResult(METHOD_CAN_BE_APPLIED, initial_data);
    for (size_t current_iteration = 0; current_iteration < LIMIT_OF_ITERATIONS;
current_iteration++) {
        float a = f->f_derivative_x(x, y);
        float b = f->f_derivative_y(x, y);
        float c = g->f_derivative_x(x, y);
        float d = g->f_derivative_y(x, y);
        float f_value = f->f(x, y);
        float g_value = g->f(x, y);
        CFloat dx = 0, dy = 0;
        if (a != 0 && (d - (b * c / a) != 0)) {
            dy = (-g_value + c * f_value / a) / (d - (b * c / a));
            dx = (-f_value - b * dy) / a;
        }
        x += dx;
        y += dy;
        result.append(x, y, dx, dy);
        if (abs(f->f(x, y)) <= eps && abs(g->f(x, y)) <= eps) break;
    }
    result.setMethodResult(METHOD_WAS_SUCCESSFULLY_FINISHED);
    return result;
}
```

```
//
// Created by artem on 24.02.2023.
//

#include "CSecantMethod.h"

enum MethodResult CSecantMethod::validateBorder(CFunctionSV* function_data, float
border_left, float border_right) {
    if (function_data->f(border_left) * function_data->f(border_right) > 0) {
        return WRONG_NUMBER_OF_SOLUTIONS;
    }
    float dx = (border_right - border_left) / DELTA;
    bool derivative_sign = function_data->f_derivative(border_left) > 0;
    bool second_derivative_sign = function_data->f_second_derivative(border_left) > 0;
    for (float current_point = border_left; current_point <= border_right;
current_point += dx) {
        bool derivative_xi_sign = function_data->f_derivative(current_point) > 0;
        bool second_derivative_xi_sign = function_data-
>f_second_derivative(current_point) > 0;
        if (derivative_sign != derivative_xi_sign) {
```

```
            return DERIVATIVE_MUST_BE_SAME_SIGN;
        }
        if (second_derivative_sign != second_derivative_xi_sign) {
            return SECOND_DERIVATIVE_MUST_BE_SAME_SIGN;
        }
    }
    return METHOD_CAN_BE_APPLIED;
}

CSecantResult CSecantMethod::performMethod(CFunctionSV* function_data, float
initial_border_left, float initial_border_right, float eps) {
    CFloat border_left = initial_border_left;
    CFloat border_right = initial_border_right;
    SingleFunctionMethodData initial_data(function_data, border_left, border_right,
eps);
    MethodResult validation_result = validateBorder(function_data, border_left,
border_right);
    CSecantResult result = *new CSecantResult(validation_result, initial_data);
    if (result.getMethodResult() == METHOD_CAN_BE_APPLIED) {
        vector<CFloat> xs;
        CFloat epsilon = (border_right - border_left) / DELTA;
        if (function_data->f(border_left) * function_data-
>f_second_derivative(border_left) > 0) {
            xs.emplace_back(border_left);
            xs.emplace_back(border_left + epsilon);
        } else {
            xs.emplace_back(border_right);
            xs.emplace_back(border_right - epsilon);
        }
        for (size_t current_iteration = 0; current_iteration < LIMIT_OF_ITERATIONS;
current_iteration++) {
            CFloat x_prev = xs[current_iteration + 1];
            CFloat x_prev_prev = xs[current_iteration];
            CFloat x_current = x_prev - function_data->f(x_prev) * (x_prev -
x_prev_prev) / (function_data->f(x_prev) - function_data->f(x_prev_prev));
            xs.push_back(x_current);
            if (abs(function_data->f(x_current)) < eps) break;
        }
        for (size_t i = 0; i < xs.size() - 2; i++) {
            result.append(xs[i], xs[i + 1], xs[i + 2]);
        }
        result.setMethodResult(METHOD_WAS_SUCCESSFULLY_FINISHED);
    }
    return result;
}
```

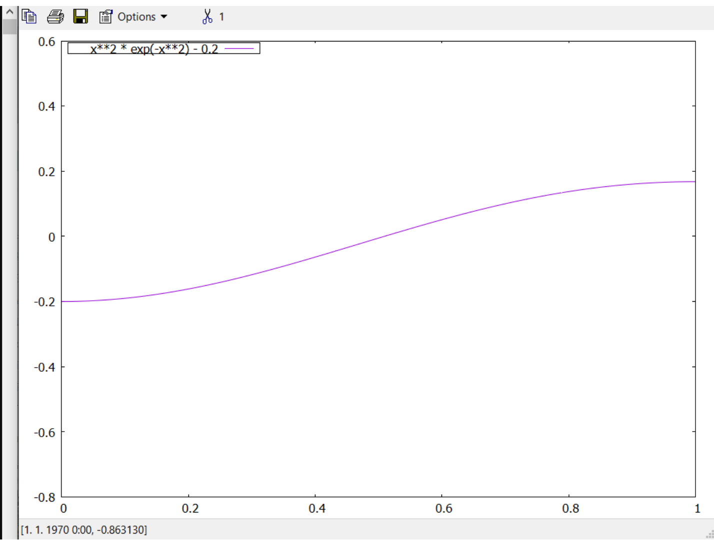
Результат выполнения программы при различных исходных данных

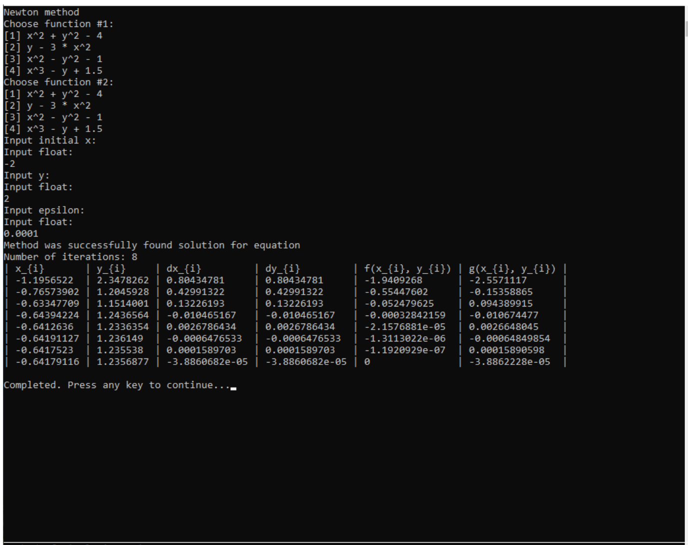


```
Half-dividing method
Choose function:
[1] - x^3 - x + 4
[2] - x^2 * exp(-x^2) - 0.2
[3] - sin(x) / pi
[4] - x^3 + 2.28 * x^2 - 1.934 * x - 3.907
Input a (left border):
Input float:
0.4
Input b (right border):
Input float:
0.6
Input epsilon:
Input float:
0.00001
Method was successfully found solution for equation
Number of iterations: 16
```

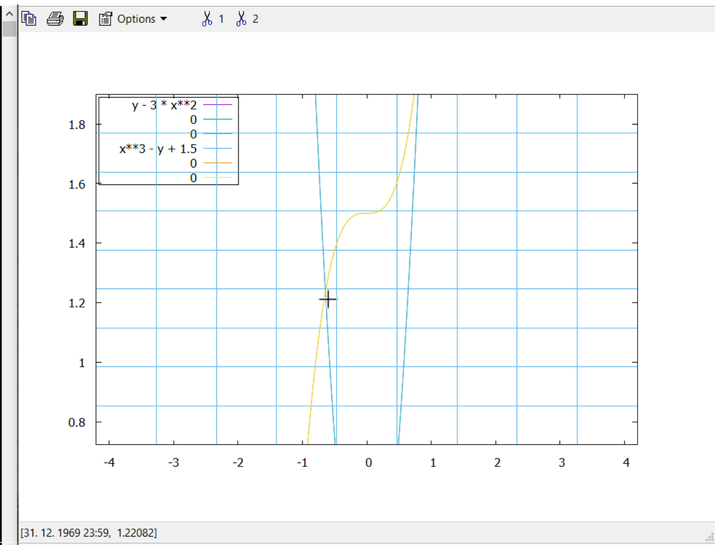
a	b	x	f(a)	f(b)	f(x)	|a - b|
0.40000001	0.60000002	0.5	-0.063656993	0.051163491	-0.0052998043	0.20000002
0.5	0.60000002	0.55000001	-0.0052998043	0.051163491	0.023537975	0.10000002
0.5	0.55000001	0.52499998	-0.0052998043	0.023537975	0.0092262458	0.050000012
0.5	0.52499998	0.51249999	-0.0052998043	0.0092262458	0.0019842773	0.024999976
0.5	0.51249999	0.50625002	-0.0052998043	0.0019842773	-0.0016532096	0.012499988
0.50625002	0.51249999	0.50937498	-0.0016532096	0.0019842773	0.00016674197	0.0062499642
0.50625002	0.50937498	0.5078125	-0.0016532096	0.00016674197	-0.00074293884	0.0031249523
0.5078125	0.50937498	0.50859374	-0.00074293884	0.00016674197	-0.00028802326	0.0015624762
0.50859374	0.50937498	0.50898433	-0.00028802326	0.00016674197	-6.0639024e-05	0.00078123808
0.50898433	0.50937498	0.50917965	-6.0639024e-05	0.00016674197	5.3056239e-05	0.00039064884
0.50898433	0.50917965	0.50908196	-6.0639024e-05	5.3056239e-05	-3.80755e-06	0.00019532442
0.50908196	0.50917965	0.50913084	-3.80755e-06	5.3056239e-05	2.464199e-05	9.7692013e-05
0.50908196	0.50913084	0.5091064	-3.80755e-06	2.464199e-05	1.0417294e-05	4.8875809e-05
0.50908196	0.5091064	0.50909418	-3.80755e-06	1.0417294e-05	3.3048907e-06	2.4437904e-05
0.50908196	0.50909418	0.50908804	-3.80755e-06	3.3048907e-06	-2.6867249e-07	1.2218952e-05
0.50908804	0.50909418	0.50909114	-2.6867249e-07	3.3048907e-06	1.5354576e-06	6.1392784e-06

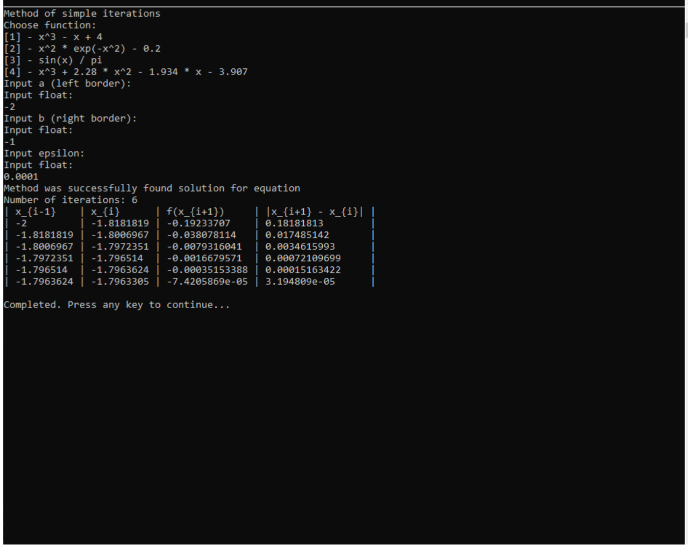
```
Completed. Press any key to continue...
```



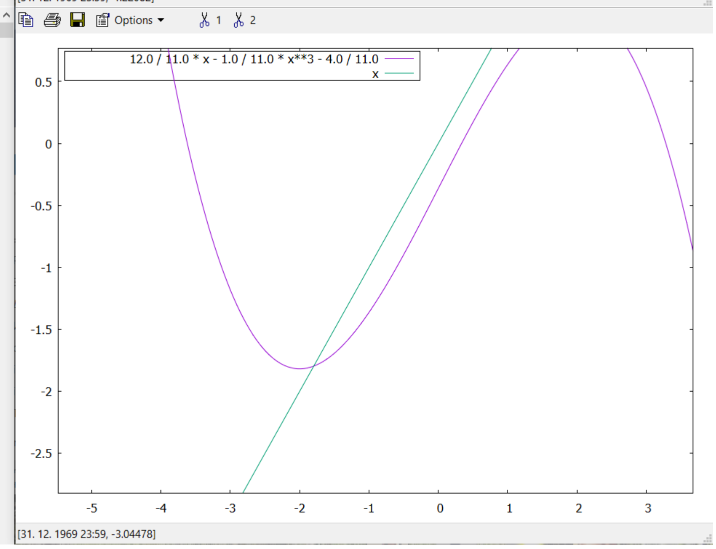


```
Newton method
Choose function #1:
[1] x^2 + y^2 - 4
[2] y - 3 * x^2
[3] x^2 - y^2 - 1
[4] x^3 - y + 1.5
Choose function #2:
[1] x^2 + y^2 - 4
[2] y - 3 * x^2
[3] x^2 - y^2 - 1
[4] x^3 - y + 1.5
Input initial x:
Input float:
-2
Input y:
Input float:
2
Input epsilon:
Input float:
0.0001
Method was successfully found solution for equation
Number of iterations: 8
| x_{i}       | y_{i}     | dx_{i}          | dy_{i}          | f(x_{i}, y_{i}) | g(x_{i}, y_{i}) |
| -1.1956522  | 2.3478262 | 0.80434781      | 0.80434781      | -1.9409268      | -2.5571117      |
| -0.76573902 | 1.2045928 | 0.42991322      | 0.42991322      | -0.55447602     | -0.15358865     |
| -0.63347709 | 1.1514001 | 0.13226193      | 0.13226193      | -0.052479625    | 0.094389915     |
| -0.64394224 | 1.2436564 | -0.010465167    | -0.010465167    | -0.00032842159  | -0.010674477    |
| -0.6412636  | 1.2336354 | 0.0026786434    | 0.0026786434    | -2.1576881e-05  | 0.0026648045    |
| -0.64191127 | 1.236149  | -0.0006476533   | -0.0006476533   | -1.3113022e-06  | -0.00064849854  |
| -0.6417523  | 1.235538  | 0.0001589703    | 0.0001589703    | -1.1920929e-07  | 0.00015890598   |
| -0.64179116 | 1.2356877 | -3.8860682e-05  | -3.8860682e-05  | 0               | -3.8862228e-05  |

Completed. Press any key to continue...
```





```
Method of simple iterations
Choose function:
[1] - x^3 - x + 4
[2] - x^2 * exp(-x^2) - 0.2
[3] - sin(x) / pi
[4] - x^3 + 2.28 * x^2 - 1.934 * x - 3.907
Input a (left border):
Input float:
-2
Input b (right border):
Input float:
-1
Input epsilon:
Input float:
0.0001
Method was successfully found solution for equation
Number of iterations: 6
| x_{i-1}    | x_{i}      | f(x_{i+1})      | |x_{i+1} - x_{i}| |
| -2         | -1.8181819 | -0.19233707     | 0.18181813        |
| -1.8181819 | -1.8006967 | -0.038078114    | 0.017485142       |
| -1.8006967 | -1.7972351 | -0.0079316041   | 0.0034615993      |
| -1.7972351 | -1.796514  | -0.0016679571   | 0.00072109699     |
| -1.796514  | -1.7963624 | -0.00035153388  | 0.00015163422     |
| -1.7963624 | -1.7963305 | -7.4205869e-05  | 3.194809e-05      |

Completed. Press any key to continue...
```



Вывод

В ходе выполнения лабораторной работы я научился использовать итерационные методы для решения нелинейных уравнений и систем нелинейных уравнений. Эмпирическими методами были получены результаты скорости сходимости основных численных методов решения нелинейных уравнений.