**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский Авиационный Институт»**

**(Национальный Исследовательский Университет)**

**Институт: №8 «Информационные технологии   
и прикладная математика»   
Кафедра: 806 «Вычислительная математика   
и программирование»**

Лабораторная работа № 2   
по курсу «Численные методы»

Группа: М8О-308Б-21

Студент: Белоносов К. А.

Преподаватель: Ревизников Д. Л.

Дата: 22.05.2024

Москва, 2024

**ОГЛАВЛЕНИЕ**

[1 Тема 3](#_Toc158983147)

[2 Задание 3](#_Toc158983148)

[3 Теория 4](#_Toc158983149)

[4 Ход лабораторной работы 5](#_Toc158983150)

[5 Выводы 6](#_Toc158983151)

# **Тема**

Методы решения нелинейных уравнений и систем нелинейных уравнений

# **Задание**

2.1. Реализовать методы простой итерации и Ньютона решения нелинейных уравнений в виде программ, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения найти положительный корень нелинейного уравнения (начальное приближение определить графически). Проанализировать зависимость погрешности вычислений от количества итераций.



2.2. Реализовать методы простой итерации и Ньютона решения систем нелинейных уравнений в виде программного кода, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения решить систему нелинейных уравнений (при наличии нескольких решений найти то из них, в котором значения неизвестных являются положительными); начальное приближение определить графически. Проанализировать зависимость погрешности вычислений от количества итераций.

a = 4

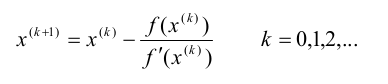
# **Теория**

Численное решение нелинейных (алгебраических или трансцендентных) уравнений вида f(x) = 0 (2.1) заключается в нахождении значений x, удовлетворяющих (с заданной точностью) данному уравнению и состоит из следующих основных этапов:

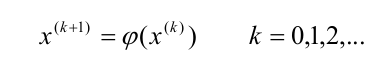
1. Отделение (изоляция, локализация) корней уравнения.

2.Уточнение с помощью некоторого вычислительного алгоритма конкретного выделенного корня с заданной точностью.

Целью первого этапа является нахождение отрезков из области определения функции f(x) , внутри которых содержится только один корень решаемого уравнения. Иногда ограничиваются рассмотрением лишь какой-нибудь части области определения, вызывающей по тем или иным соображениям интерес. Для реализации данного этапа используются графические или аналитические способы.

**Метод Ньютона** (метод касательных). При нахождении корня уравнения (2.1) методом Ньютона, итерационный процесс определяется формулой:

**Метод простой итерации.** При использовании метода простой итерации уравнение (2.1) заменяется эквивалентным уравнением с выделенным линейным членом x = ϕ(x) Решение ищется путем построения последовательности:



# **Х****од лабораторной работы**

Код был реализован на языке C++

2.1) Метод Ньютона и простой итерации для решения уравнения:

#ifndef SOLVING\_METHODS\_H

#define SOLVING\_METHODS\_H

#include <functional>

#include <stdexcept>

#include <utility>

#include <iostream>

#include "../lab1/Matrix.h"

using namespace numeric;

double find\_q(const std::function<double(double, double)>& phi\_d, double a, double b, double lambda) {

double q = -1e9;

double x = a;

while(x <= b) {

q = std::max(q, std::abs(phi\_d(x, lambda)));

x += 0.01;

}

return q;

}

double find\_q(

const numeric::Matrix<std::function<double(const std::vector<double>&)>>& phi\_d,

const std::vector<double>& a,

const std::vector<double>& b,

double step = 0.01

) {

if (a.size() != b.size() || a.size() == 0) {

throw std::invalid\_argument("Invalid dimension boundaries or empty boundaries provided.");

}

size\_t n = a.size();

std::vector<double> x(n);

double q = 0;

std::function<void(size\_t)> iterate = [&](size\_t dim) {

if (dim == n) {

numeric::Matrix<double> values(phi\_d.rows(), phi\_d.cols());

for (size\_t i = 0; i < phi\_d.rows(); ++i) {

for (size\_t j = 0; j < phi\_d.cols(); ++j) {

values[i][j] = std::abs(phi\_d[i][j](x));

}

}

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

double row\_sum = 0;

for (size\_t j = 0; j < values.cols(); ++j) {

row\_sum += values[i][j];

}

q = std::max(q, row\_sum);

}

} else {

for (double v = a[dim]; v <= b[dim]; v += step) {

x[dim] = v;

iterate(dim + 1);

}

}

};

iterate(0);

return q;

}

double iterationMethod(const std::function<double(double, double)>& f, const std::function<double(double, double)>& f\_d, double a, double b, double x0, double lambda, double eps) {

double q = find\_q(f\_d, a, b, lambda);

if(q >= 1) {

std::cout << q << "\n";

throw std::invalid\_argument("Function is not valid");

}

double x\_cur = x0;

double continueIteration = 1.0;

int countOfIterations = 0;

do {

double x\_next = f(x\_cur, lambda);

continueIteration = q / (1.0 - q) \* std::abs(x\_next - x\_cur);

x\_cur = x\_next;

++countOfIterations;

} while (continueIteration > eps);

std::cout << "Iterations count: " << countOfIterations << "\n";

return x\_cur;

}

double newtonMethod(const std::function<double(double)>& f, const std::function<double(double)>& f\_d, const std::function<double(double)>& f\_dd,double a, double b, double eps) {

double x0 = a;

if (!(f(x0) \* f\_dd(x0) > eps)) {

x0 = b;

}

double x\_cur = x0;

double dx = 1.0;

int countOfIterations = 0;

do {

double x\_next = x\_cur - f(x\_cur) / f\_d(x\_cur);

dx = std::abs(x\_next - x\_cur);

x\_cur = x\_next;

++countOfIterations;

} while (dx > eps);

std::cout << "Iterations count: " << countOfIterations << "\n";

return x\_cur;

}

std::vector<double> iterationMethodSystem(

const std::vector<std::function<double(const std::vector<double>&)>>& F,

const numeric::Matrix<std::function<double(const std::vector<double>&)>>& J,

const std::vector<double>& initial\_guess,

const std::vector<double>& a,

const std::vector<double>& b,

double eps) {

std::vector<double> x = initial\_guess;

std::vector<double> x\_next = initial\_guess;

std::size\_t n = initial\_guess.size();

bool converged = true;

double q = find\_q(J, a, b);

if(q >= 1) {

std::cout << q << "\n";

throw std::invalid\_argument("Function is not valid");

}

while (converged) {

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

x\_next[i] = F[i](x);

}

if (numeric::norm((q / (1.0 - q)) \* numeric::diffVector(x\_next, x)) < eps) {

converged = false;

}

x = x\_next;

}

return x;

}

std::vector<double> newtonMethodLU(

const std::vector<std::function<double(const std::vector<double>&)>>& F,

const Matrix<std::function<double(const std::vector<double>&)>>& J,

const std::vector<double>& initialGuess,

double eps

) {

std::vector<double> x = initialGuess;

size\_t n = x.size();

bool continueIteration = true;

while(continueIteration) {

std::vector<double> Fx(n);

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

Fx[i] = F[i](x);

}

Matrix<double> Jx(n, n + 1);

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

for (size\_t j = 0; j < n; ++j) {

Jx[i][j] = J[i][j](x);

}

Jx[i][n] = -Fx[i];

}

LUMatrix<double> lu(Jx);

Matrix<double> dx = lu.solve();

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

x[i] += dx[0][i];

}

if (norm(dx[0]) < eps) {

continueIteration = false;

}

}

return x;

}

#endif //SOLVING\_METHODS\_H

Вывод работы программы:

Iteration method:

Iterations count: 86

0.0914981

Newton method:

Iterations count: 6

0.0914901

2.2) Метод Ньютона и простой итерации для решения системы уравнений:

#include "Solving\_methods.h"

#include "../lab1/Matrix.h"

#include <iostream>

#include <valarray>

using namespace std;

int main() {

vector<function<double(const std::vector<double>&)>> F(2);

F[0] = [](const std::vector<double>& x) {

return (x[0] \* x[0] + 16) \* x[1] - 64;

};

F[1] = [](const std::vector<double>& x) {

return pow(x[0] - 2, 2) + pow(x[1] - 2, 2) - 16;

};

vector<function<double(const std::vector<double>&)>> PHI(2);

PHI[0] = [](const std::vector<double>& x) {

return 2 - sqrt(16 - pow(x[1] - 2, 2));

};

PHI[1] = [](const std::vector<double>& x) {

return 64 / (x[0] \* x[0] + 16);

};

Matrix<std::function<double(const std::vector<double>&)>> J(2, 2);

J[0][0] = [](const std::vector<double>& x) {

return 2 \* x[0] \* x[1];

};

J[0][1] = [](const std::vector<double>& x) {

return x[0] \* x[0] + 16;

};

J[1][0] = [](const std::vector<double>& x) {

return 2 \* x[0] - 4;

};

J[1][1] = [](const std::vector<double>& x) {

return 2 \* x[1] - 4;

};

Matrix<std::function<double(const std::vector<double>&)>> PHI\_D(2, 2);

PHI\_D[1][0] = [](const std::vector<double>& x) {

return -128\*x[0]/pow(x[0] \* x[0] + 16, 2);

};

PHI\_D[1][1] = [](const std::vector<double>& x) {

return 0;

};

PHI\_D[0][0] = [](const std::vector<double>& x) {

return 0;

};

PHI\_D[0][1] = [](const std::vector<double>& x) {

return (x[1] - 2) / sqrt(16 - pow(x[1] - 2, 2));

};

cout << "\nIteration method:\n";

printVector(iterationMethodSystem(PHI, PHI\_D, {-1, 3}, {-2, 1}, {1, 3.5}, 0.0001));

cout << "\nNewton method:\n";

printVector(newtonMethodLU(F, J,{-1, 3}, 0.001));

return 0;

}

Вывод работы программы:

Iteration method:

-1.76626 3.34735

Newton method:

-1.78537 3

# **Выводы**

В этой лабораторной работе рассматриваются численные методы решения нелинейных уравнений и систем нелинейных уравнений. Были реализованы метод Ньютона и метод простых итераций для решения уравнения и решения целой системы уравнений.