**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 2**

по курсу «Численные методы».

Тема: «Метод решения нелинейных уравнений и систем нелинейных уравнений».

Студент: Вельтман Л.Я.

Группа: 80-307Б

Вариант: 7

Оценка:

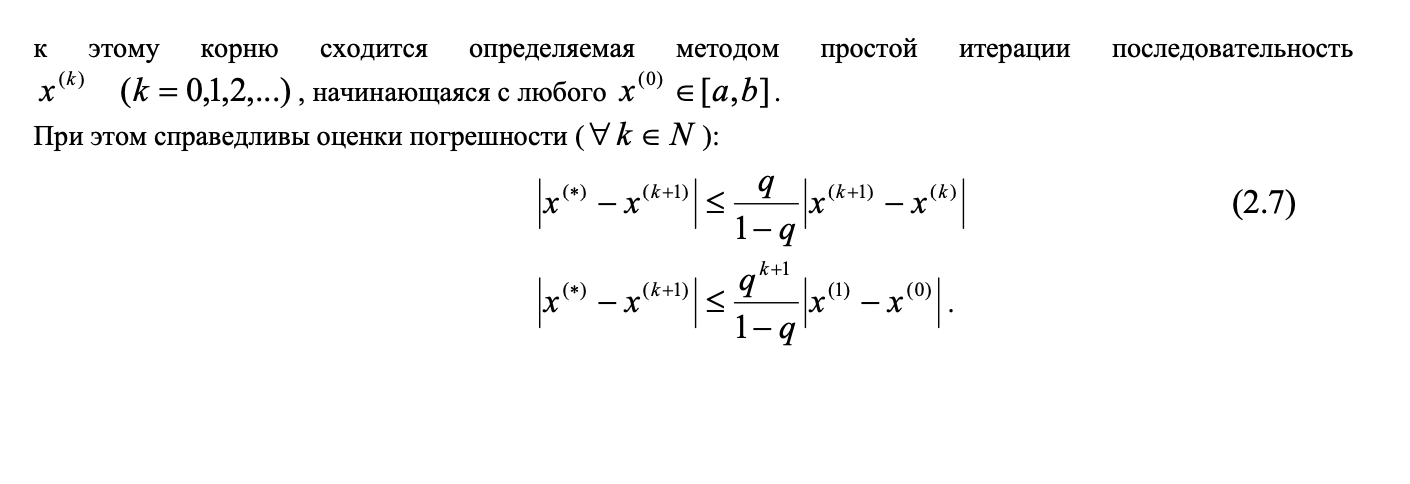
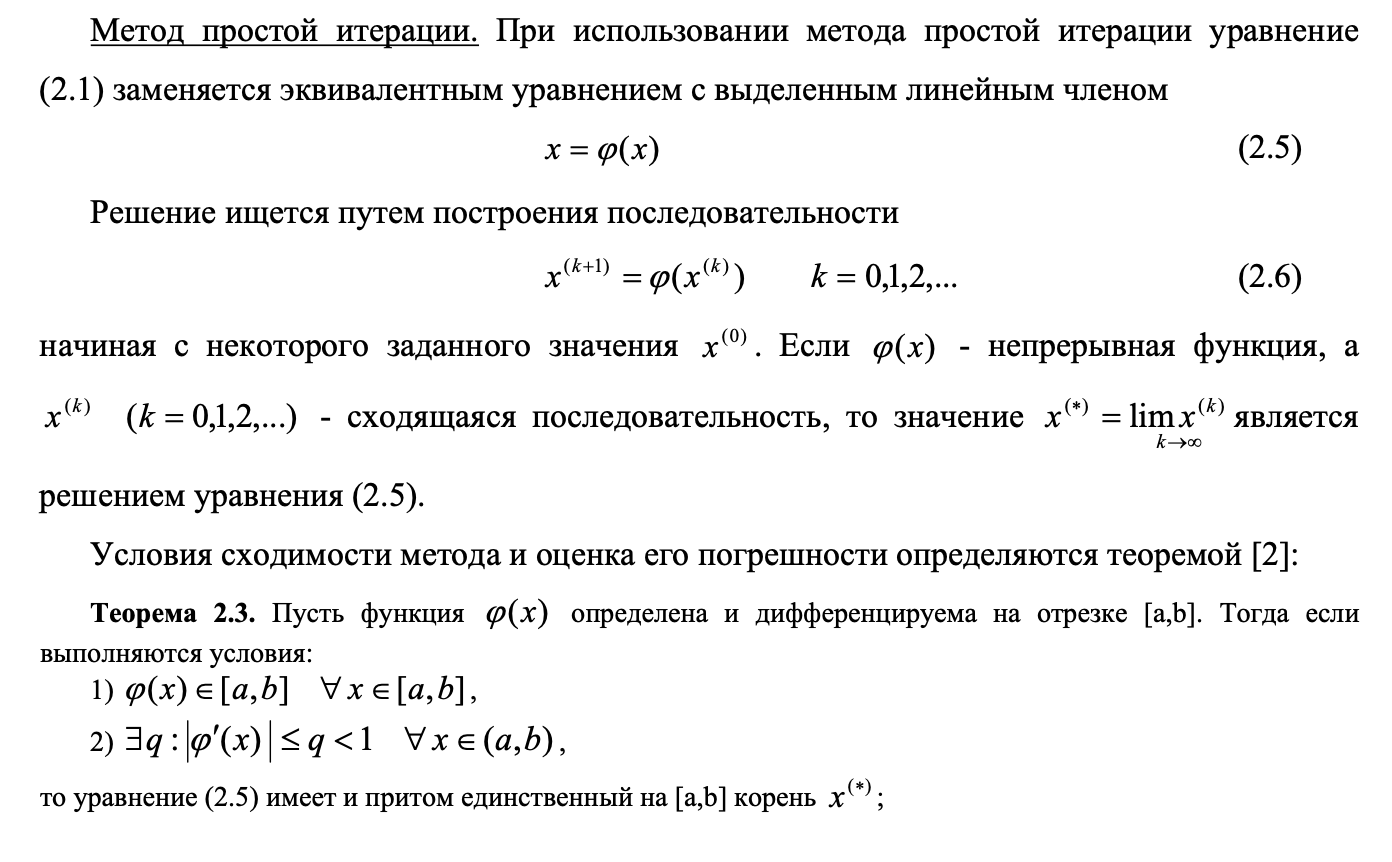
Москва, 2020

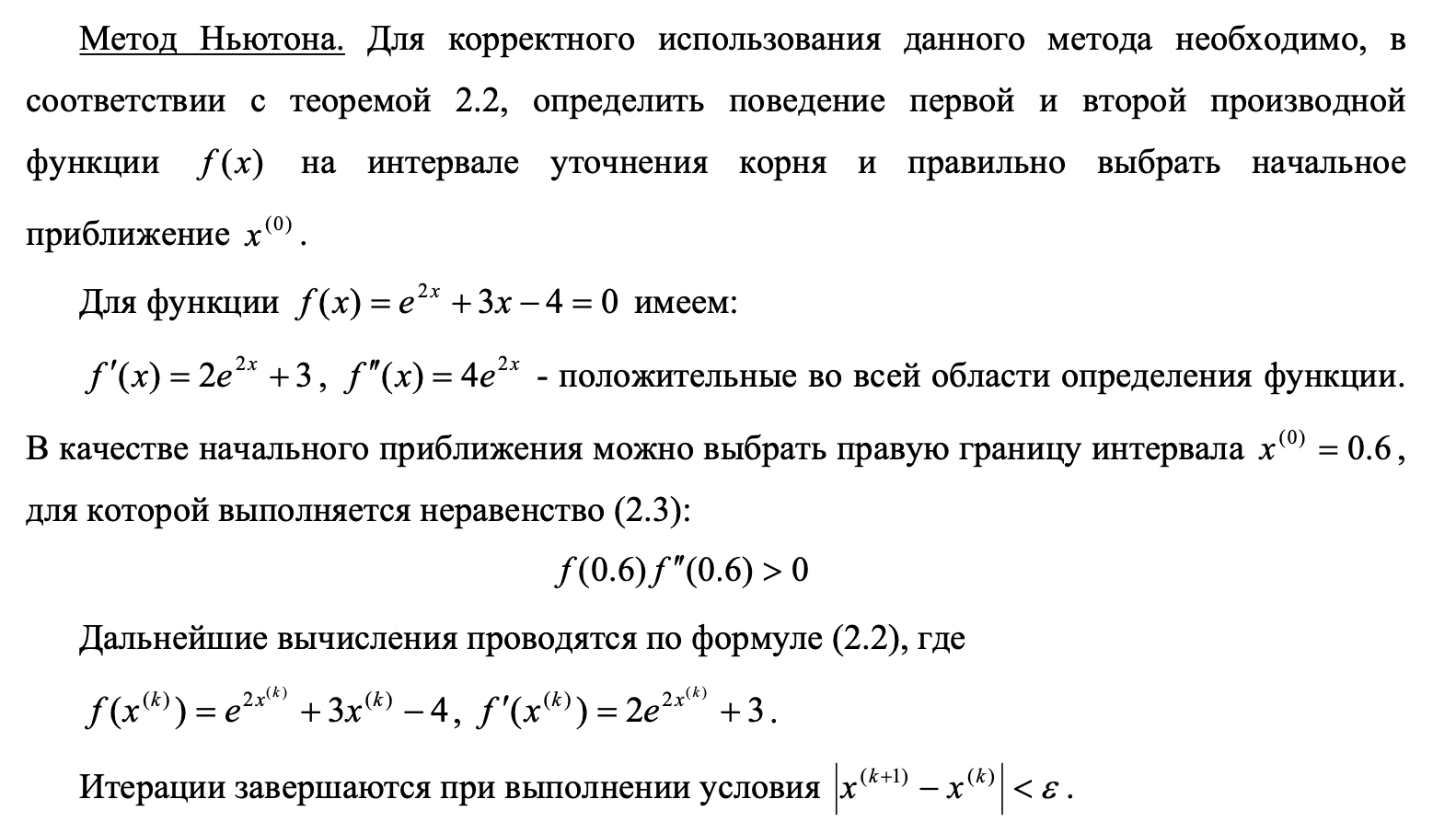
Постановка задачи.

Реализовать методы для решения нелинейных уравнений:

1. Метод простых итераций
2. Метод Ньютона (касательных)

Описание методов.





Общая информация.

Данная работа позволяет решать нелинейные уравнения методами Ньютона и простых итераций. Что касается технических деталей реализации, программа написана на языке С++.

Запуск программы.

Чтобы воспользоваться программой, необходимо скомпилировать файл main.cpp и запустить полученный исполняемый файл:

*g++ -std=c++11 main.cpp -o run*

*run.exe*

Результаты.

Вариант 1.

|  |  |
| --- | --- |
| 0.0  1.0  0.001  2^x + x \* x - 2  // a  // b  // eps  // funcs | ANSWER:  Newton's method  x = 0.653483  Iterations: 2  Simple iterations method  x = 0.653043  Iterations: 33 |

Выводы.

Изучила методы простой итерации и Ньютона для решения нелинейных уравнений.

Исходный код.

/\*

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.

\*/

#include <iostream>

#include <cmath>

#include <vector>

//const double e = 2.71828182846;

double foo(double x) {

return (pow(2, x) + x \* x - 2);

}

double firstDiff(double x) {

return (pow(2, x) \* log(2) + 2 \* x);

}

double secondDiff(double x) {

return (pow(2, x) \* log(2) \* log(2) + 2);

}

int NewtonsMethod(double a, double b, double eps, double& ans) {

int k = 0;

double xk\_1 = /\*0.6\*/a + eps;

double xk = xk\_1;

if (foo(a) \* foo(b) < 0) {

double criteria = eps;

while (criteria >= eps) {

if (foo(xk\_1) \* secondDiff(xk\_1) > 0) {

xk = xk\_1;

xk\_1 = xk - (foo(xk)) / firstDiff(xk);

++k;

criteria = abs(xk\_1 - xk);

} else {

if (xk\_1 < b) {

xk\_1 += eps;

} else {

throw "Try to find another length [a;b]";

}

}

}

} else {

throw "Try to find another length [a;b]";

}

ans = xk\_1;

return k + 1;

}

double phi(double x) {

return sqrt(2 - pow(2, x));

}

double phiDiff(double x) {

return (-pow(2, x)\* log(2)) / (2 \* sqrt(2 - pow(2, x)));

}

int LyambdaSolution(double xk, double& xk\_1, double eps) {

int k = 0;

double lyambda = eps;

while (true)

if (abs(1 - lyambda \* firstDiff(xk\_1)) < 1) {

double criteria = eps;

while (criteria >= eps) {

xk = xk\_1;

xk\_1 = phi(xk);

++k;

criteria = abs(xk\_1 - xk);

}

break;

} else {

lyambda += eps;

}

return k + 1;

}

int SimpleIterationsMethod(double a, double b, double eps, double& ans) {

double x0 = (a + b) / 2, xk;

double xk\_1 = xk = x0;

int k = 0;

if (phi(x0) >= a && phi(x0) <= b) {

if (abs(phiDiff(x0)) < 1) {

double criteria = eps;

while (criteria >= eps) {

xk = xk\_1;

xk\_1 = phi(xk);

++k;

criteria = abs(xk\_1 - xk);

}

} else {

k = LyambdaSolution(xk, xk\_1, eps);

}

} else {

k = LyambdaSolution(xk, xk\_1, eps);

}

ans = xk\_1;

return k + 1;

}

int main(int argc, const char \* argv[]) {

double accuracy = 0.001;

double a = 0.0, b = 1.0, ans1 = 0.0, ans2 = 0.0;

int iterNewton = NewtonsMethod(a, b, accuracy, ans1);

int iterSimple = SimpleIterationsMethod(a, b, accuracy, ans2);

std::cout << "ANSWER:" << std::endl;

std::cout << "Newton's method" << std::endl;

std::cout << "x = " << ans1 << std::endl;

std::cout << "Iterations: " << iterNewton << std::endl;

std::cout << "Simple iterations method" << std::endl;

std::cout << "x = " << ans2 << std::endl;

std::cout << "Iterations: " << iterSimple << std::endl;

return 0;

}