Санкт-Петербургский государственный университет

Прикладная математика и информатика

Отчет по лабораторной работе №1 по вычислительному практикуму

**Приближенное решение нелинейных уравнений**

Выполнил:

Чернов Павел Олегович

группа 221

Санкт-Петербург

2022

1. **Введение**

ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

Дано алгебраическое уравнение

причем, известно, что все интересующие вычислителя корни находятся на отрезке[A, B], на котором функция определена и непрерывна.

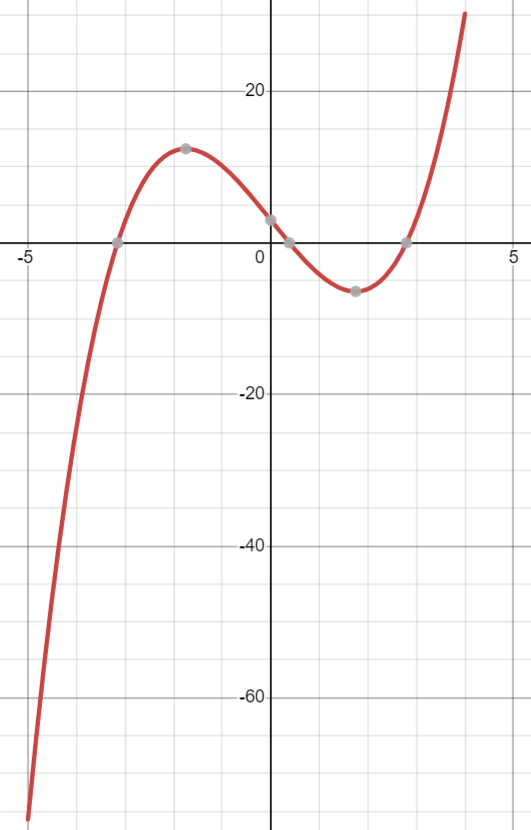
Требуется найти все корни уравнения на [A, B] нечетной кратности (здесь A, B, – параметры задачи).

Решение задачи разбить на два этапа:

1. Процедура отделения корней уравнения на отрезке [A, B]
2. Уточнение корней уравнения на отрезках перемены знака вида [ai, bi]
   1. Методом половинного деления (методом бисекции);
   2. Методом Ньютона (методом касательных);
   3. Модифицированным методом Ньютона;
   4. Методом секущих

с заданной точностью ε > 0.

График функции:



1. **Ход работы**
2. Представленная ниже программа позволяет обнаружить корни нечетной кратности на заданном отрезке, а также вычислить их приближенное значение с некоторой погрешностью.
3. Для приближенного нахождения значений корней используются методы бисекции, Ньютона, модифицированный Ньютона и секущих.
4. Исходным параметром задачи является функция . В программе можно задать интервал, на котором будут искаться корни, шаг, с которым интервал будет проходиться, и допустимую погрешность обнаружения корней. Ввод осуществляется при помощи клавиатуры.
5. Программа написана на языке программирования С++, в расчетах используются переменные типа double.
6. **Приложение**

#include <windows.h>

#include <ctime>

#include <iomanip>

#include <vector>

#include <cmath>

#include <iostream>

#include "Header.h"

double f(double x) { //исследуемая функция

return pow(2, -x) + 0.5 \* x \* x - 10;

//return sin(x) + x\*x\*x - 9 \* x + 3;

}

double df(double x) { //производная f

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

//return cos(x) + 3 \* x \* x - 9;

}

double d2f(double x) { //вторая производная f

return log(2) \* log(2) \* pow(2, -x) + 1;

//return -sin(x) + 6 \* x;

}

void intervals\_detector(double left, double right, double step, std::vector <std::pair<double, double>>& interval) {

double lBoard = left, rBoard = lBoard + step;

while (rBoard <= right) {

if (f(lBoard) \* f(rBoard) < 0) {

std::cout << "На промежутке [" << lBoard << "; " << rBoard << "] есть как минимум один корень.\n";

interval.push\_back(std::make\_pair(lBoard, rBoard));

}

lBoard = rBoard, rBoard += step;

}

}

double bisection(double left, double right, double epsilon) {

std::cout << "\n|Метод бисекции|\n";

double middle;

int count = 0; //кол-во итераций

std::cout << "Начальное приближение к корню: " << (right + left) / 2.0 << "\n";

while (right - left > 2 \* epsilon) {

middle = (right + left) / 2.0;

if (f(left) \* f(middle) <= 0) {

right = middle;

}

else {

left = middle;

}

++count;

}

double x\_approx = (right + left) / 2.0;

std::cout << "За " << count << " шага(ов) был найден приближенный корень: " << x\_approx <<

"\n|Xm - Xm-1| = " << right - left <<

"\nАбсолютная величина невязки: " << abs(f(x\_approx)) << "\n";

return x\_approx;

}

double Newton(double x0, double epsilon) {

std::cout << "\n|Метод Ньютона|\n";

double x\_approx = x0, x\_last = x0 + epsilon \* 2;

int count = 0; //кол-во итераций

std::cout << "Начальное приближение к корню: " << x0 << "\n";

while (abs(x\_approx - x\_last) > epsilon) {

x\_last = x\_approx;

x\_approx = x\_last - f(x\_last) / df(x\_last);

++count;

}

std::cout << "За " << count << " шага(ов) был найден приближенный корень: " << x\_approx <<

"\n|Xm - Xm-1| = " << abs(x\_approx - x\_last) <<

"\nАбсолютная величина невязки: " << abs(f(x\_approx)) << "\n";

return x\_approx;

}

double Newton\_modified(double x0, double epsilon) {//неправильно x\_last считаю

std::cout << "\n|Модифицированный метод Ньютона|\n";

double x\_approx = x0, x\_last = x0 + epsilon \* 2, df\_x0 = df(x0);

int count = 0; //кол-во итераций

std::cout << "Начальное приближение к корню: " << x0 << "\n";

while (abs(x\_approx - x\_last) > epsilon) {

x\_last = x\_approx;

x\_approx = x\_last - f(x\_last) / df\_x0;

++count;

}

std::cout << "За " << count << " шага(ов) был найден приближенный корень: " << x\_approx <<

"\n|Xm - Xm-1| = " << abs(x\_approx - x\_last) <<

"\nАбсолютная величина невязки: " << abs(f(x\_approx)) << "\n";

return x\_approx;

}

double secants(double x0, double x1, double epsilon) {

std::cout << "\n|Метод секущих|\n";

double x\_approx = x1, x\_before\_last = x0, x\_last = x1 + epsilon \* 2;

int count = 0; //кол-во итераций

std::cout << "Начальное приближение к корню: " << x1 << "\n";

while (abs(x\_approx - x\_last) > epsilon) {

x\_before\_last = x\_last;

x\_last = x\_approx;

x\_approx = x\_last - f(x\_last) \* (x\_last - x\_before\_last) / (f(x\_last) - f(x\_before\_last));

++count;

}

std::cout << "За " << count << " шага(ов) был найден приближенный корень: " << x\_approx <<

"\n|Xm - Xm-1| = " << abs(x\_approx - x\_last) <<

"\nАбсолютная величина невязки: " << abs(f(x\_approx)) << "\n";

return x\_approx;

}

double selection\_of\_x0(double left, double right) { //функция выбора х0 для методов Ньютона

double x0 = left + (right - left) \* (double(rand()) /RAND\_MAX);

//std::cout << x0 << "\n";

while (abs(df(x0)) > 1e-8 && f(x0) \* d2f(x0) > 0) {

x0 = left + (right - left) \* (double(rand()) / RAND\_MAX);

}

return x0;

}

std::vector <std::pair<double, double>> repeat\_splitting(std::vector <std::pair<double, double>>& interval) { //функция дополнительного деления отрезков, если они не удовл. пользователя

std::cout << "Обнаружено " << interval.size() << " промежутка(ов) с корнями, желаете исследовать их подробнее?\n" <<

"Если да, нажмите 1, иначе 0: ";

short int switcher;

std::cin >> switcher;

switch (switcher) {

case 1:

{

std::cout << "Введите, на сколько частей хотите разбить полученные промежутки: ";

int n;

std::cin >> n;

std::vector <std::pair<double, double>> new\_interval;

for (int i = 0; i < interval.size(); ++i) {

std::vector <std::pair<double, double>> i\_interval;

intervals\_detector(interval[i].first, interval[i].second, (interval[i].second - interval[i].first) / n, i\_interval);

new\_interval.insert(new\_interval.end(), i\_interval.begin(), i\_interval.end());

}

interval = repeat\_splitting(new\_interval);

}

break;

default:

break;

}

return interval;

}

int main() {

setlocale(LC\_ALL, "ru");

srand(time(NULL));

std::cout << std::fixed << std::setprecision(16);

SetConsoleTitleA("Computational methods of solving nonlinear equations");

float A, B; //границы отрезка (-5; 4)

int n; //количество разбиений отрезка

std::cout << "Введите границы отрезка: ";

std::cin >> A >> B;

std::cout << "Введите количество разбиений: ";

std::cin >> n;

double step = (B - A) / n \* 1.0; //шаг по отрезку

std::vector <std::pair<double, double>> interval;

intervals\_detector(A, B, step, interval);

if (interval.empty()) {

std::cout << "С шагом " << step << " на отрезке не было найдено ни одного корня, попробуйте шаг поменьше: ";

std::cin >> step;

}

else {

interval = repeat\_splitting(interval);

}

std::cout << "\nУточнение корней\n" << "Введите допустимую погрешность: ";

double err\_rate; //1e-9;

std::cin >> err\_rate;

for (int i = 0; i < interval.size(); ++i) {

std::cout << "\nНахождение корня на отрезке [" << interval[i].first << "; " << interval[i].second << "]\n";

bisection(interval[i].first, interval[i].second, err\_rate);

Newton(selection\_of\_x0(interval[i].first, interval[i].second), err\_rate);

Newton\_modified(selection\_of\_x0(interval[i].first, interval[i].second), err\_rate);

secants(interval[i].first, interval[i].second, err\_rate);

}

return 0;

}

https://github.com/Chernovuk/Chislaki