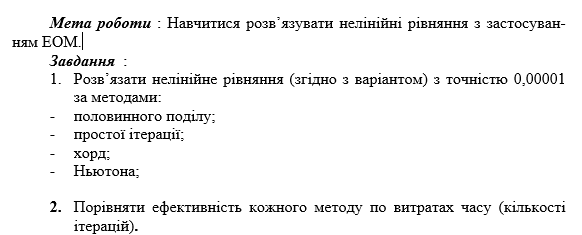
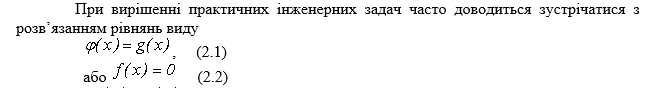
**Лабораторна робота №2**

Розв’язання нелінійних рівнянь за чисельними методами

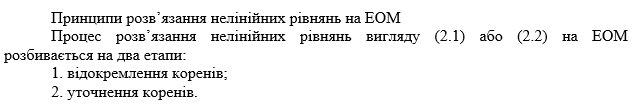


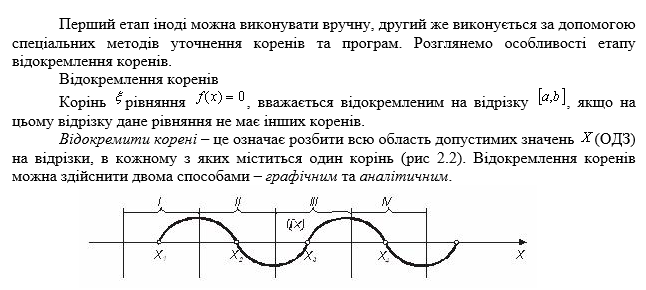


Короткі теоретичні відомості

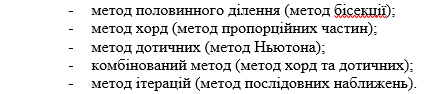












**Хід роботи**

1. Методика розв’язання задачі
2. Текст програми

#include <iostream>

#include <string>

#include <cmath>

const double DEF\_EPSILON = 0.00001;

int iterativeCounter = 0;

// Возвращает значение функции в точке х

double getF(double x) {

return log10(x) - 7 / (2 \* x + 6);

}

// Возвращает значение производной в точке х

double getFPrime(double x) {

return 1 / (x \* log(10)) + 14 / ((2 \* x + 6) \* (2 \* x + 6));

}

// Преобразованная функция F для метода итераций

double getG(double x) {

return x - 4.13 \* getF(x);

}

double bisection(double l, double r, double eps = DEF\_EPSILON);

double bisection\_recursive(double l, double r, double eps = DEF\_EPSILON);

double falsePosition(double l, double r, double eps = DEF\_EPSILON);

double falsePosition\_recursive(double l, double r, double eps = DEF\_EPSILON);

double newtonRaphson(double l, double r, double eps = DEF\_EPSILON);

double newtonRaphson\_recursive(double l, double r, double eps = DEF\_EPSILON);

double iterative(double x0, double eps = DEF\_EPSILON);

double iterative\_recursive(double x0, double eps = DEF\_EPSILON);

void processRes(std::string method, double res);

void checkRoots(double l, double r);

void clearIterativeCounter();

void printHello();

void run();

template <typename T>

T prompt(const char label[]);

int main() {

printHello();

while (true) {

try {

run();

}

catch (std::runtime\_error err) {

std::cout << err.what() << std::endl;

}

if (prompt<std::string>("Repeat (0 - no): ") == "0") {

break;

}

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

}

return 0;

}

void run() {

double eps = prompt<double>("Epsilon (if 0 default - 0.00001): ") / 10;

if (eps <= 0) eps = DEF\_EPSILON;

std::cout << "Enter [a, b]:" << std::endl;

double l = prompt<double>("a: ");

double r = prompt<double>("b: ");

checkRoots(l, r);

std::cout << "Results: " << std::endl << std::endl;

processRes("Bisection Method (iterative)", bisection(l, r, eps));

processRes("Bisection Method (recursive)", bisection\_recursive(l, r, eps));

std::cout << std::endl;

processRes("False-Position Method (iterative)", falsePosition(l, r, eps));

processRes("False-Position Method (recursive)", falsePosition\_recursive(l, r, eps));

std::cout << std::endl;

processRes("NewtonRaphson Method (iterative)", newtonRaphson(l, r, eps));

processRes("NewtonRaphson Method (recursive)", newtonRaphson\_recursive(l, r, eps));

std::cout << std::endl;

processRes("Iterative Method (iterative)", iterative((l + r) / 2, eps));

processRes("Iterative Method (recursive)", iterative\_recursive((l + r) / 2, eps));

std::cout << std::endl;

}

void processRes(std::string method, double res) {

std::cout << method << ": " << res << "; Iterations: " << iterativeCounter << std::endl;

clearIterativeCounter();

}

void printHello() {

std::cout << " \* \* \* Numerical Methods - Root Finding \* \* \* " << std::endl << std::endl;

}

double bisection(double l, double r, double eps) {

checkRoots(l, r);

double m, my;

while (true) {

iterativeCounter++;

m = (l + r) / 2;

my = getF(m);

if (abs(my) <= eps || abs(r - l) < eps) {

break;

}

if (my \* getF(l) < 0) r = m;

else l = m;

}

return m;

}

double bisection\_recursive(double l, double r, double eps) {

checkRoots(l, r);

iterativeCounter++;

double m = (l + r) / 2;

double my = getF(m);

if (abs(my) < eps || abs(r - l) < eps) return m;

if (my \* getF(l) < 0) return bisection\_recursive(l, m);

else return bisection\_recursive(m, r);

}

double falsePosition(double l, double r, double eps) {

checkRoots(l, r);

double ly, ry, mx, my;

while (true) {

iterativeCounter++;

ly = getF(l);

ry = getF(r);

mx = l - ly \* (r - l) / (ry - ly);

my = getF(mx);

if (abs(my) <= eps || abs(r - l) < eps) {

break;

}

if (ly \* my < 0) r = mx;

else l = mx;

}

return mx;

}

double falsePosition\_recursive(double l, double r, double eps) {

checkRoots(l, r);

iterativeCounter++;

double ly = getF(l);

double ry = getF(r);

double mx = l - ly \* (r - l) / (ry - ly);

double my = getF(mx);

if (abs(my) <= eps || abs(r - l) < eps) {

return mx;

}

if (ly \* my < 0) return falsePosition\_recursive(l, mx);

else return falsePosition\_recursive(mx, r);

}

double newtonRaphson(double l, double r, double eps) {

checkRoots(l, r);

double x;

while (true) {

iterativeCounter++;

if (abs(r - l) < eps) {

x = (l + r) / 2;

break;

}

if (getF(r) \* getFPrime(r) > 0) std::swap(l, r);

x = l - getF(l) / getFPrime(l);

if (abs(x - l) < eps) break;

l = x;

}

return x;

}

double newtonRaphson\_recursive(double l, double r, double eps) {

iterativeCounter++;

if (abs(r - l) < eps) return (l + r) / 2;

if (getF(r) \* getFPrime(r) > 0) std::swap(l, r);

double x = l - getF(l) / getFPrime(l);

if (abs(x - l) < eps) return x;

return newtonRaphson\_recursive(x, r, eps);

}

double iterative(double x0, double eps) {

while (true) {

iterativeCounter++;

double x1 = getG(x0);

if (abs(x1 - x0) <= eps) break;

x0 = x1;

}

return x0;

}

double iterative\_recursive(double x0, double eps) {

iterativeCounter++;

double x1 = getG(x0);

if (abs(x1 - x0) <= eps) return x0;

return iterative\_recursive(x1);

}

void checkRoots(double l, double r) {

if (getF(l) \* getF(r) < 0) return;

throw std::runtime\_error("No roots on this range");

}

void clearIterativeCounter() {

iterativeCounter = 0;

}

1. Результат виконання

**Висновок:** на цій лабораторній роботі навчився розв’язувати нелінійні рівняння на ЕОМ.