**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ ИМЕНИ ПАТРИСА ЛУМУМБЫ**

**Факультет физико-математических и естественных наук**

**Кафедра «Математического моделирования и искусственного интеллекта»**

Компьютерный практикум

Лабораторная работа №1

**«Операции с комплексными числами и   
поиск корней уравнения»**

|  |  |
| --- | --- |
| Студент | Плугин Никита |
| Группа | НБбд-01-23 |
|  |  |

**Москва**

**20****24**

Оглавление

[Введение. 3](#_Toc169268609)

[1. Теоретическая основа. 4](#_Toc169268610)

[1.1. Операции с комплексными числами 4](#_Toc169268611)

[1.2. Методы поиска корней уравнения 5](#_Toc169268612)

[2. Алгоритмы. 7](#_Toc169268613)

[2.1. Операции с комплексными числами 7](#_Toc169268614)

[2.2. Методы поиска корней уравнения 8](#_Toc169268615)

[3. Программа реализации алгоритмов 12](#_Toc169268616)

[3.1. Операции с комплексными числами 12](#_Toc169268617)

[3.2. Методы поиска корней уравнения 13](#_Toc169268618)

[Заключение. 14](#_Toc169268619)

# Введение.

В данной работе будут рассмотрены алгоритмы поиска корней уравнения и операции с комплексными числами. Целью данной работы является ознакомление с данными методами.

В первой части работы приведена теория

Во второй части работы приведены алгоритмы данных методов.

В третьей части реализована сама программа.

# Теоретическая основа.

## Операции с комплексными числами

Комплексные числа представляют собой расширение вещественных чисел и записываются в виде , где и — вещественные числа, а — мнимая единица, такая что . Здесь называется действительной частью, а — мнимой частью комплексного числа .

Основные операции с комплексными числами

1. Сложение:

Для двух комплексных чисел и сумма определяется как:

2. Вычитание:

Для двух комплексных чисел разность определяется как:

3. Умножение:

Для двух комплексных чисел произведение определяется как:

Поскольку , то формула принимает вид:

4. Деление:

Для двух комплексных чисел частное определяется как:

Для упрощения необходимо умножить числитель и знаменатель на сопряжённое комплексное число к знаменателю :

Для возведения комплексного числа в степень удобно использовать показательное представление комплексных чисел, основанное на формуле Эйлера:

где — модуль числа, а — аргумент числа (угол в полярных координатах).

Тогда возведение в степень определяется как:

Извлечение корня -й степени из комплексного числа также удобно проводить в показательной форме. Корни из комплексного числа определяются по формуле:

Это означает, что -й корень из комплексного числа имеет различных значений, расположенных равномерно на комплексной плоскости по углам

## Методы поиска корней уравнения

Метод дихотомии, или метод бисекции, основан на теореме Больцано-Коши: если непрерывная функция имеет разные знаки на концах отрезка , то на этом отрезке существует хотя бы один корень уравнения .

Алгоритм:

1. Найти середину отрезка .

2. Вычислить .

3. Если , то корень находится на отрезке ; иначе, на отрезке .

4. Повторять шаги 1-3 до достижения заданной точности.

Метод простых итераций основывается на преобразовании исходного уравнения в эквивалентное уравнение .

Алгоритм:

1. Начальное приближение .

2. Итерационный процесс .

3. Продолжать итерации, пока не будет достигнута заданная точность .

Метод хорд — это численный метод для нахождения корней уравнения, который можно рассматривать как обобщение метода Ньютона, где касательная заменяется хордой.

Алгоритм:

1. Начальные приближения и .

2. Итерационный процесс:

3. Продолжать итерации, пока не будет достигнута заданная точность.

Метод Ньютона, или метод касательных, использует производную функции для нахождения корней уравнения .

Алгоритм:

1. Начальное приближение .

2. Итерационный процесс:

3. Продолжать итерации, пока не будет достигнута заданная точность

# Алгоритмы.

## Операции с комплексными числами

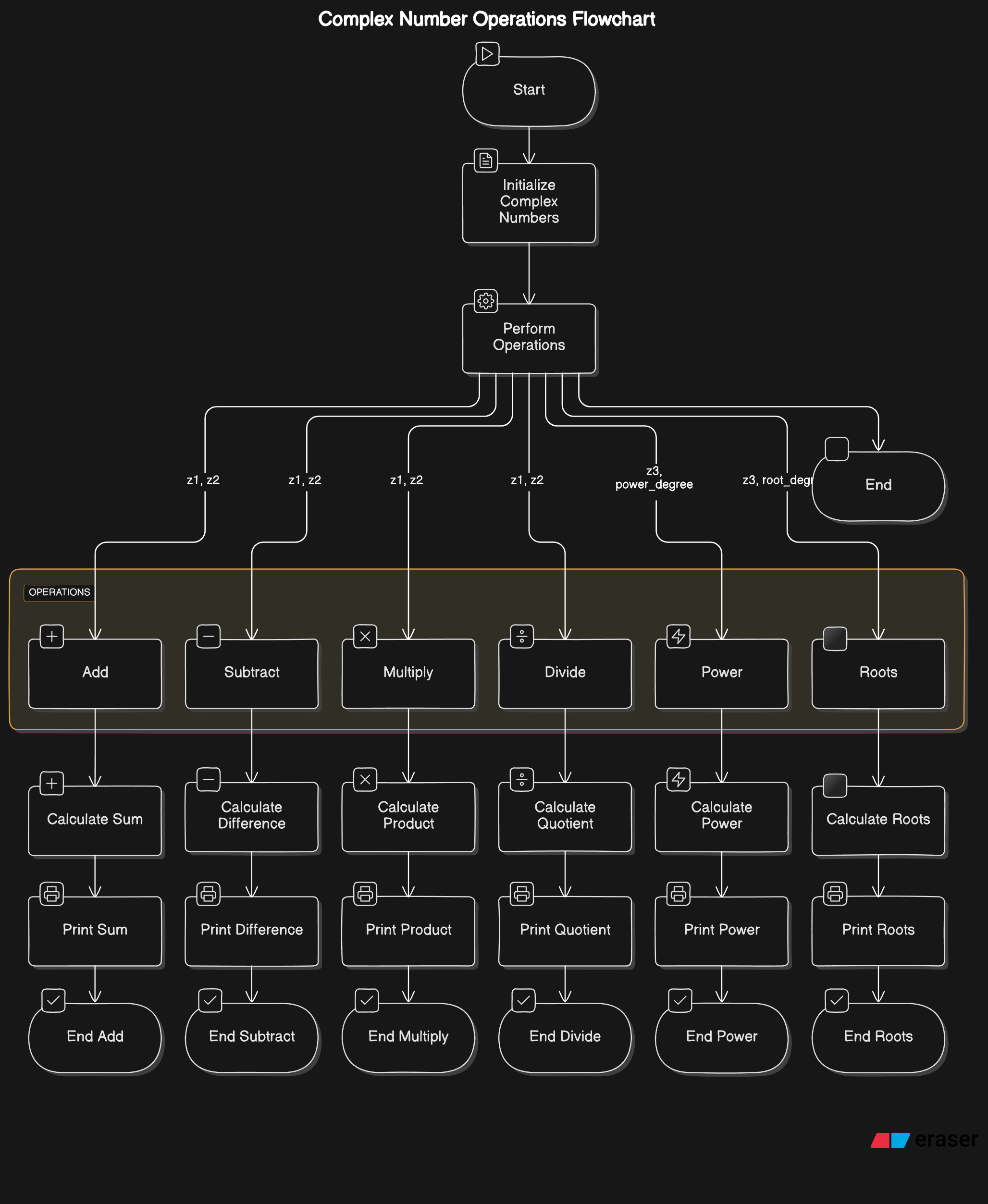


Рисунок 2‑1 Блок-схема программы для операций с комплексными числами

## Методы поиска корней уравнения

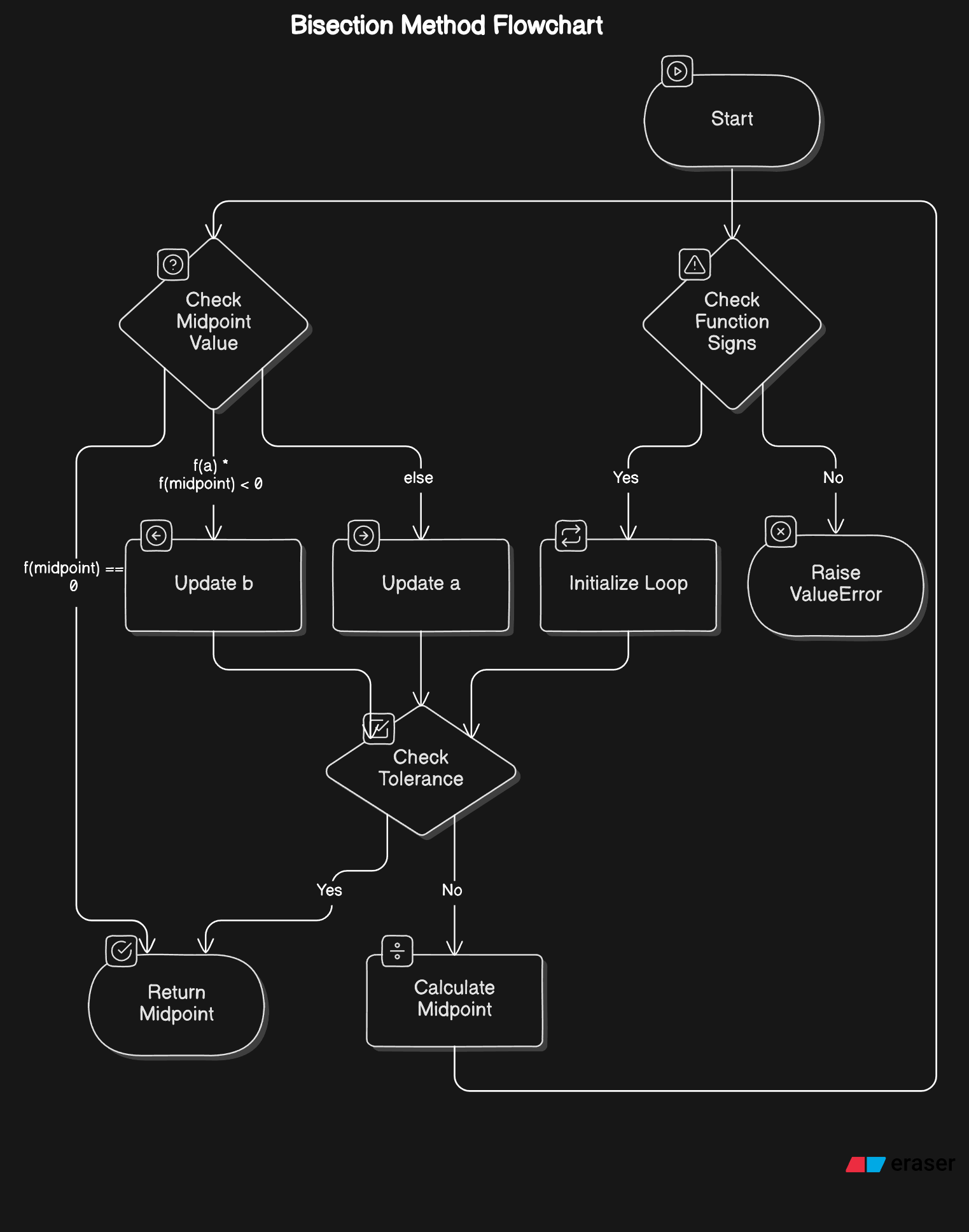


Рисунок 2‑2 Блок-схема метода бисекций

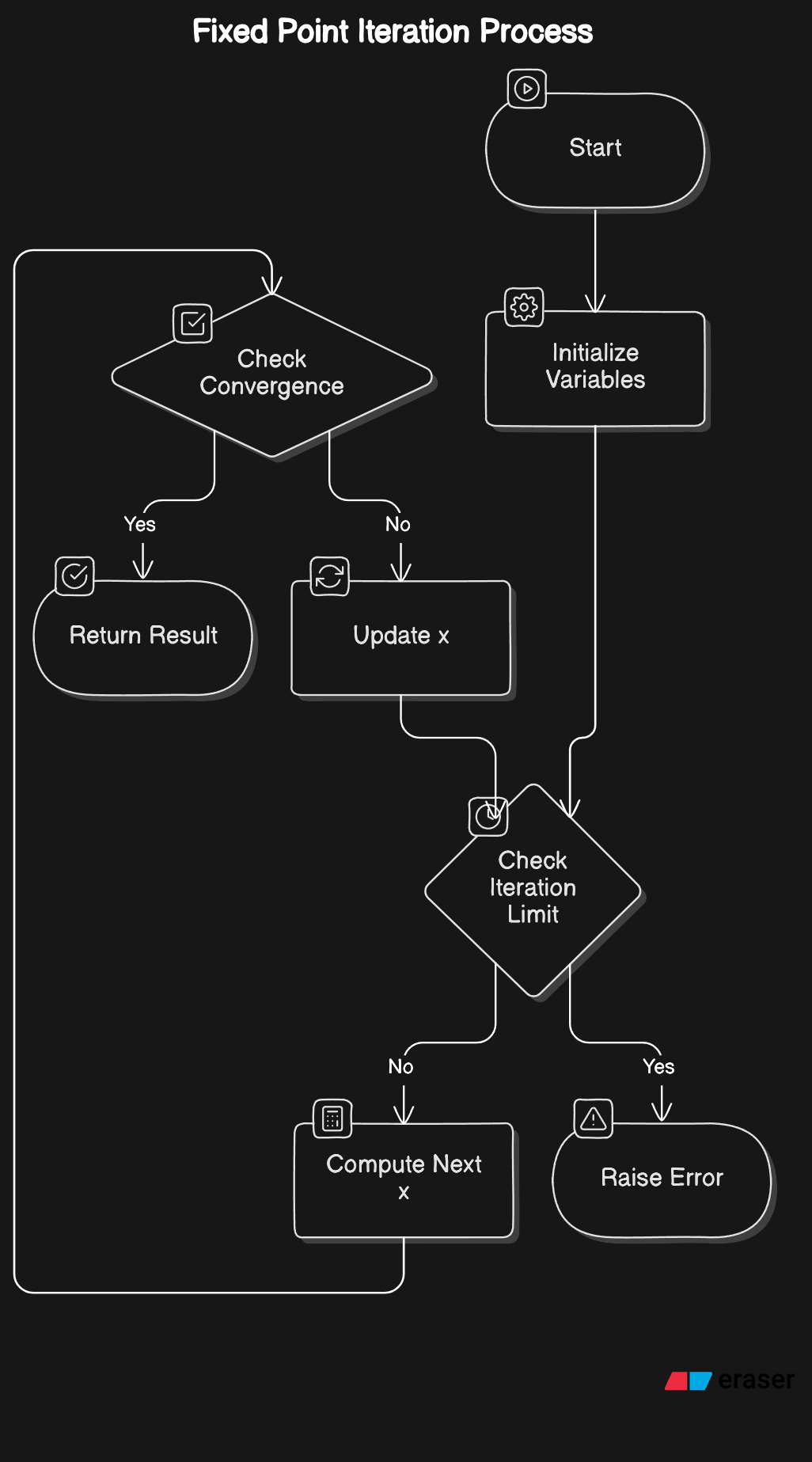


Рисунок 2‑3 Блок-схема метода простых итераций

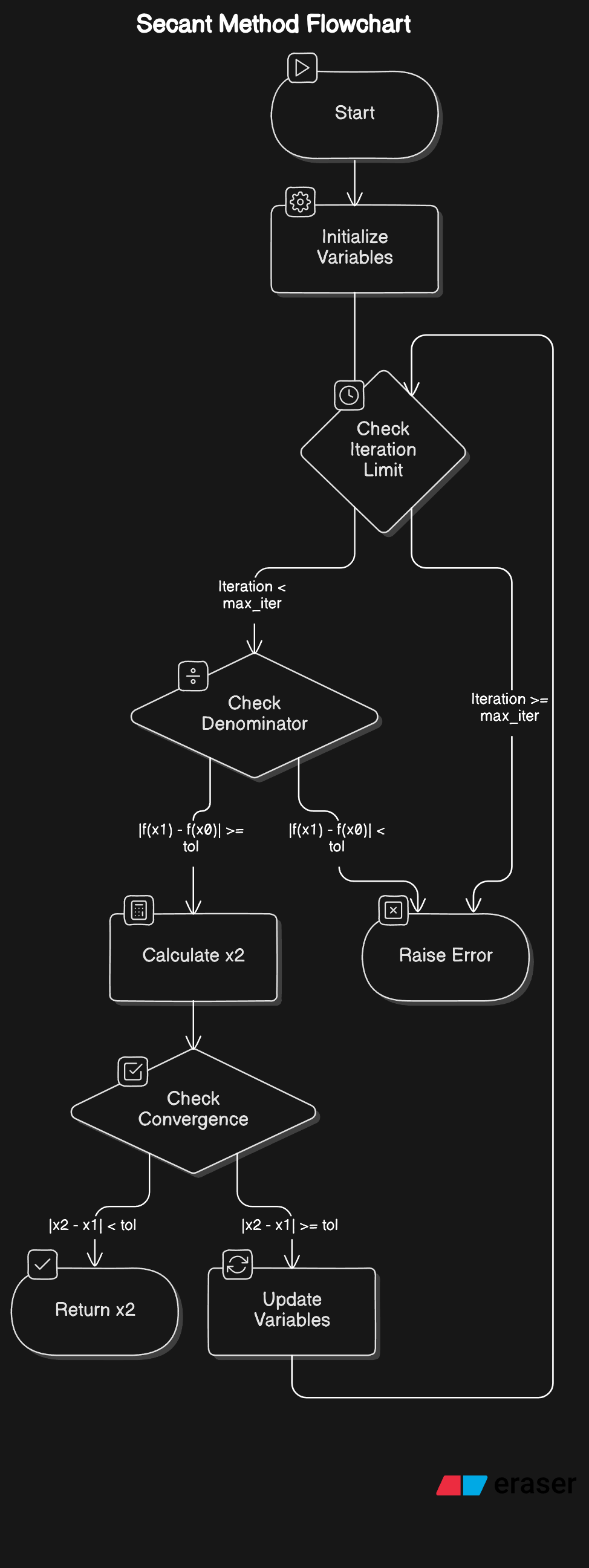


Рисунок 2‑4 Блок-схема метода хорд

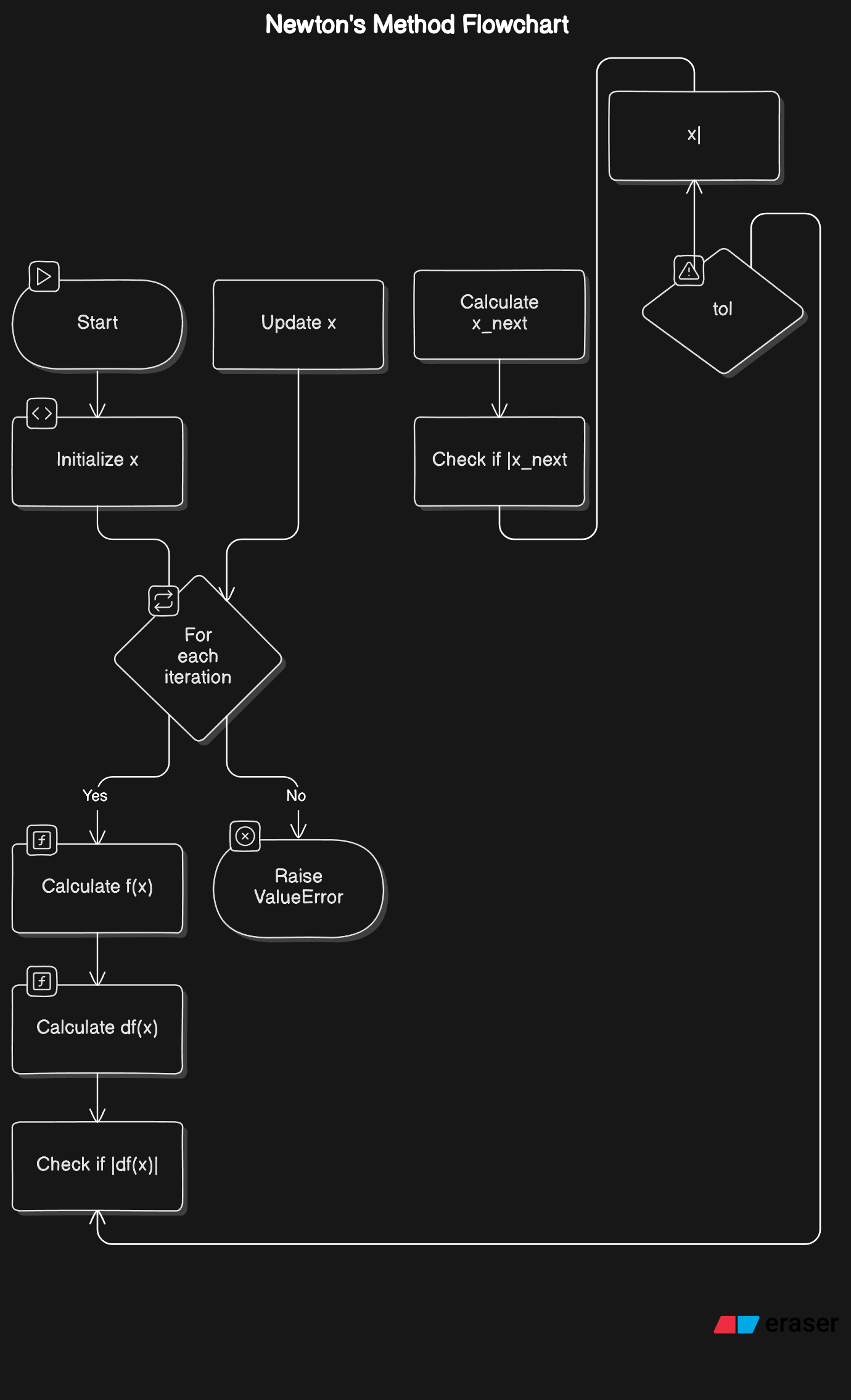


Рисунок 2‑5 Блок-схема метода Ньютона

# Программа реализации алгоритмов

## Операции с комплексными числами

#include <iostream>

#include <cmath>

struct Complex {

double real;

double imag;

};

Complex add(const Complex& a, const Complex& b) {

return {a.real + b.real, a.imag + b.imag};

}

Complex subtract(const Complex& a, const Complex& b) {

return {a.real - b.real, a.imag - b.imag};

}

Complex multiply(const Complex& a, const Complex& b) {

return {

a.real \* b.real - a.imag \* b.imag,

a.real \* b.imag + a.imag \* b.real

};

}

Complex divide(const Complex& a, const Complex& b) {

double denominator = b.real \* b.real + b.imag \* b.imag;

return {

(a.real \* b.real + a.imag \* b.imag) / denominator,

(a.imag \* b.real - a.real \* b.imag) / denominator

};

}

Complex power(const Complex& a, int n) {

double r = std::sqrt(a.real \* a.real + a.imag \* a.imag);

double theta = std::atan2(a.imag, a.real);

double r\_pow = std::pow(r, n);

double angle = n \* theta;

return {r\_pow \* std::cos(angle), r\_pow \* std::sin(angle)};

}

void roots(const Complex& a, int n) {

double r = std::sqrt(a.real \* a.real + a.imag \* a.imag);

double theta = std::atan2(a.imag, a.real);

double r\_root = std::pow(r, 1.0 / n);

for (int k = 0; k < n; ++k) {

double angle = (theta + 2 \* M\_PI \* k) / n;

Complex root = {r\_root \* std::cos(angle), r\_root \* std::sin(angle)};

std::cout << "Root " << k + 1 << ": " << root.real << " + " << root.imag << "i" << std::endl;

}

}

int main() {

Complex z1 = {-1, 1};

Complex z2 = {-3, -1};

Complex sum = add(z1, z2);

Complex difference = subtract(z1, z2);

Complex product = multiply(z1, z2);

Complex quotient = divide(z1, z2);

std::cout << "Sum: " << sum.real << " + " << sum.imag << "i" << std::endl;

std::cout << "Difference: " << difference.real << " + " << difference.imag << "i" << std::endl;

std::cout << "Product: " << product.real << " + " << product.imag << "i" << std::endl;

std::cout << "Quotient: " << quotient.real << " + " << quotient.imag << "i" << std::endl;

Complex z3 = {1, 2};

int power\_degree = 4;

Complex z3\_power = power(z3, power\_degree);

std::cout << z3.real << " + " << z3.imag << "i ^ " << power\_degree << " = "

<< z3\_power.real << " + " << z3\_power.imag << "i" << std::endl;

int root\_degree = 3;

std::cout << "Roots of " << z3.real << " + " << z3.imag << "i ^ (1/" << root\_degree << "):" << std::endl;

roots(z3, root\_degree);

return 0;

}

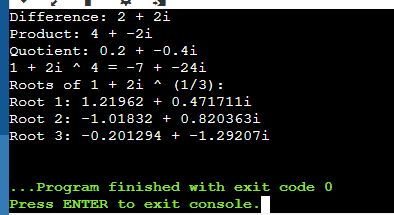


Рисунок 3‑1 Результат операций с комплексными числами

## Методы поиска корней уравнения

import math

# Функция f(x)

def f(x):

return math.exp(-x) - math.sqrt(x - 1)

# Производная функции f(x) для метода Ньютона

def df(x):

return -math.exp(-x) - 1/(2\*math.sqrt(x - 1))

# Метод дихотомии (бисекции)

def bisection(a, b, tol=1e-6):

if f(a) \* f(b) >= 0:

raise ValueError("Function values at the endpoints must have different signs")

while (b - a) / 2.0 > tol:

midpoint = (a + b) / 2.0

if f(midpoint) == 0:

return midpoint

elif f(a) \* f(midpoint) < 0:

b = midpoint

else:

a = midpoint

return (a + b) / 2.0

# Метод простых итераций

def fixed\_point\_iteration(g, x0, tol=1e-6, max\_iter=1000):

x = x0

for \_ in range(max\_iter):

x\_next = g(x)

if abs(x\_next - x) < tol:

return x\_next

x = x\_next

raise ValueError("Fixed point iteration did not converge")

# Метод хорд (секущих)

def secant(x0, x1, tol=1e-6, max\_iter=1000):

for \_ in range(max\_iter):

if abs(f(x1) - f(x0)) < tol:

raise ValueError("Denominator in secant method is too small")

x2 = x1 - f(x1) \* (x1 - x0) / (f(x1) - f(x0))

if abs(x2 - x1) < tol:

return x2

x0, x1 = x1, x2

raise ValueError("Secant method did not converge")

# Метод Ньютона

def newton(x0, tol=1e-6, max\_iter=1000):

x = x0

for \_ in range(max\_iter):

f\_x = f(x)

df\_x = df(x)

if abs(df\_x) < tol:

raise ValueError("Derivative is too small")

x\_next = x - f\_x / df\_x

if abs(x\_next - x) < tol:

return x\_next

x = x\_next

raise ValueError("Newton's method did not converge")

# Основная программа

if \_\_name\_\_ == "\_\_main\_\_":

# Установить начальные значения и отрезок для методов

a, b = 1, 2.5 # Отрезок для метода дихотомии

x0 = 1.1 # Начальное приближение для методов итераций и Ньютона

x1 = 1.3 # Второе начальное приближение для метода хорд

try:

root\_bisection = bisection(a, b)

print(f"Bisection method root: {root\_bisection:.6f}")

except ValueError as e:

print(e)

try:

g = lambda x: 1 + math.exp(-x)

root\_fixed\_point = fixed\_point\_iteration(g, x0)

print(f"Fixed point iteration root: {root\_fixed\_point:.6f}")

except ValueError as e:

print(e)

try:

root\_secant = secant(x0, x1)

print(f"Secant method root: {root\_secant:.6f}")

except ValueError as e:

print(e)

try:

root\_newton = newton(x0)

print(f"Newton's method root: {root\_newton:.6f}")

except ValueError as e:

print(e)

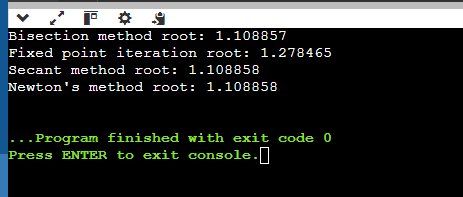


Рисунок 3‑2 Поиск корней уравнения

# Заключение.

В данной работе был показаны операции с комплексными числами и методы поиска корней уравнения