

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Факультет безопасности информационных технологий

Дисциплина:

«Алгоритмы и структуры данных»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №0

«Решение кубического уравнения комбинированным методом хорд и касательных»

Выполнили:

Суханкулиев М.,
студент группы N3246

(подпись)

Проверил:

Ерофеев С. А.

(отметка о выполнении)

(подпись)

Санкт-Петербург

2025 г.

СОДЕРЖАНИЕ

Введение	3
1 Алгоритм решения уравнения	4
1.1 Описание алгоритма	4
1.2 Блок-схема алгоритма	5
2 Спецификация переменных	8
3 Реализация программы	9
3.1 Описание программы	9
3.2 Код программы	9
3.2.1 <code>headers.h</code>	9
3.2.2 <code>main.c</code>	13
4 Тестирование программы	14
4.1 Результаты тестирования	14
4.1.1 Корректность ввода	14
4.1.2 Решения кубических уравнений	15
4.1.3 Решения квадратных уравнений	16
4.1.4 Проверка условий a, b, c, d	17
4.1.5 Остальные проверки	18
4.2 Сравнение результатов с примерами из учебных пособий	18
Заключение	20
Список использованных источников	21

ВВЕДЕНИЕ

Цель работы – Разработать программу для решения кубического уравнения вида $ax^3 + bx^2 + cx + d = 0$, $a, b, c, d \in \mathbb{R}$ комбинированным методом хорд и касательных.

Для достижения поставленной цели необходимо решить следующие **задачи**:

- Разработать алгоритм решения кубического уравнения и представить его в виде блок-схемы;
- Составить спецификацию всех переменных;
- Реализовать программу на языке C с учетом корректной обработки всех возможных случаев;
- Провести тестирование программы.

Замечание: после нахождения первого корня используются методы упрощения уравнения (метод Горнера) и решения квадратного уравнения для нахождения оставшихся корней.

1 АЛГОРИТМ РЕШЕНИЯ УРАВНЕНИЯ

1.1 Краткое описание алгоритма

Пошаговый алгоритм решения уравнения $ax^3 + bx^2 + cx + d = 0$:

1) Проверка коэффициентов

– Если $a = 0$, то уравнение квадратное или линейное.

– Если $a = 0, b \neq 0$ – используем формулы Виета.

$$x = \frac{-c \pm \sqrt{c^2 - 4bd}}{2b}$$

– Если $a = 0, b = 0, c \neq 0$ – решаем линейное уравнение.

$$x = -\frac{d}{c}$$

– Если $a = 0, b = 0, c = 0$ – проверяем d ($d = 0$ – бесконечное множество решений, $d \neq 0$ – нет решений).

– Если $a \neq 0$ – продолжаем решение кубического уравнения.

2) Поиск интервала, содержащего корень

– Находим критические точки решая $f'(x) = 3ax^2 + 2bx + c = 0$.

$$D_1 = b^2 - 3ac$$

$$x_1 = \frac{b \pm \sqrt{D_1}}{3a}$$

– Определяем интервал $[left, right]$, на котором происходит смена знака.

– Если смены знака нет – расширяем границы интервала.

3) Применение комбинированного метода хорд и касательных

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

– Выполняем итерации метода до достижения точности ε .

$$|x_{n+1} - x_n| < \varepsilon$$

4) Нахождение оставшихся корней

– Используем метод Горнера для понижения степени уравнения.

$$A = a, \quad B = b + A \cdot x_{new}, \quad C = c + b \cdot x_{new}$$

– Решаем оставшееся квадратное уравнение $Ax^2 + Bx + C = 0$.

$$x = -\frac{B \pm \sqrt{B^2 - 4AC}}{2A}$$

1.2 Блок-схема алгоритма

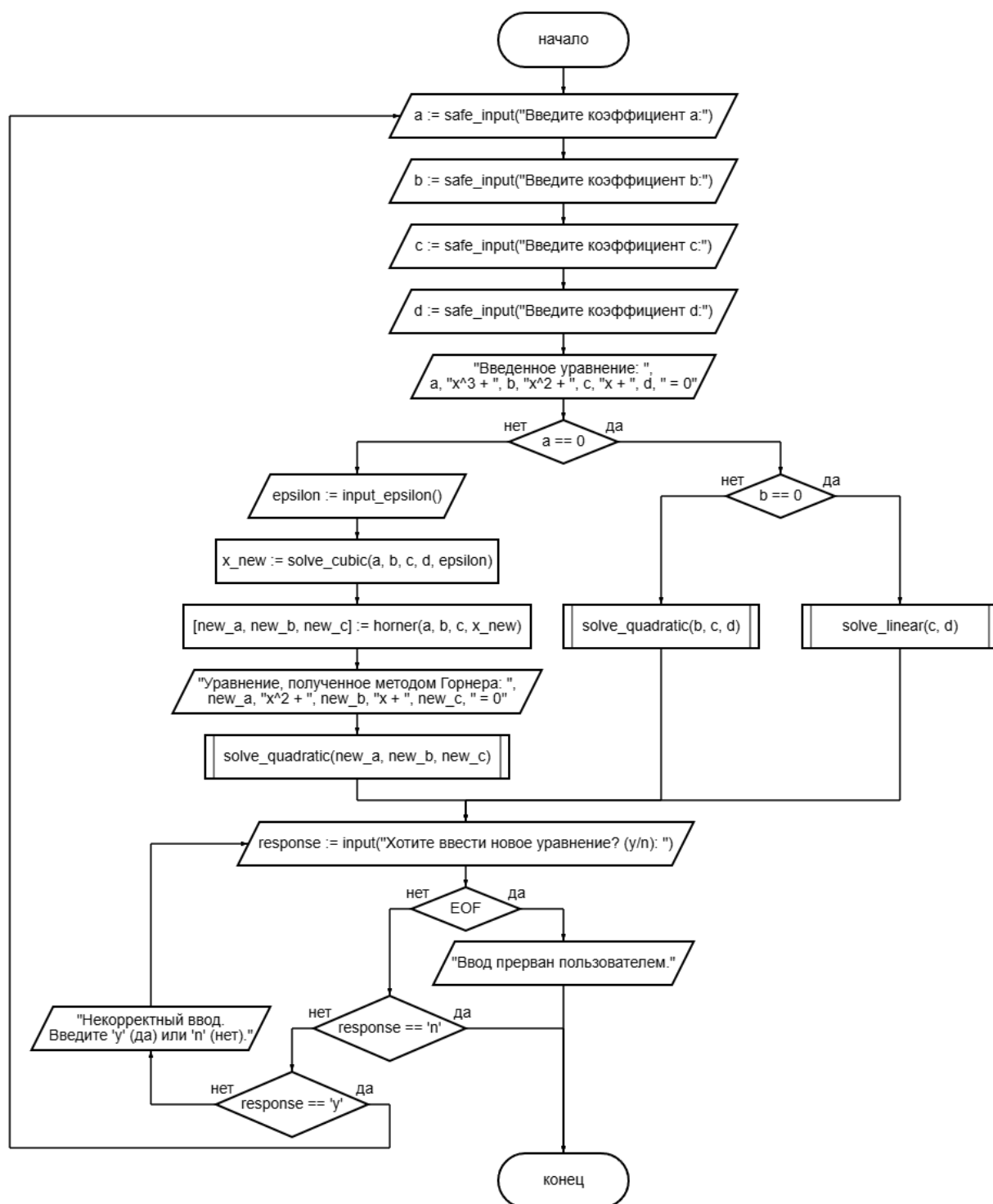


Рисунок 1 – Основная схема алгоритма

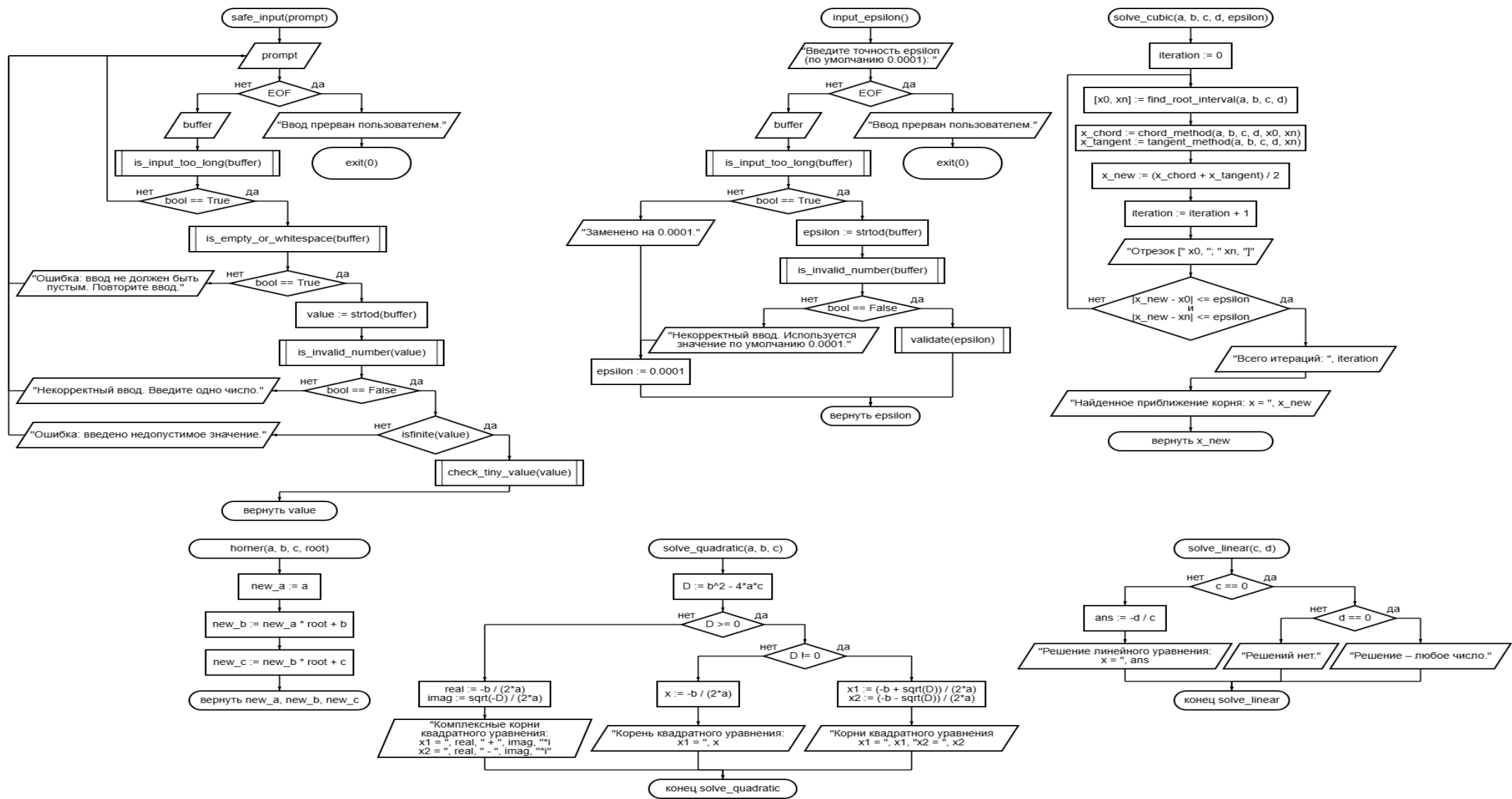


Рисунок 2 – Функции, используемые в основной схеме

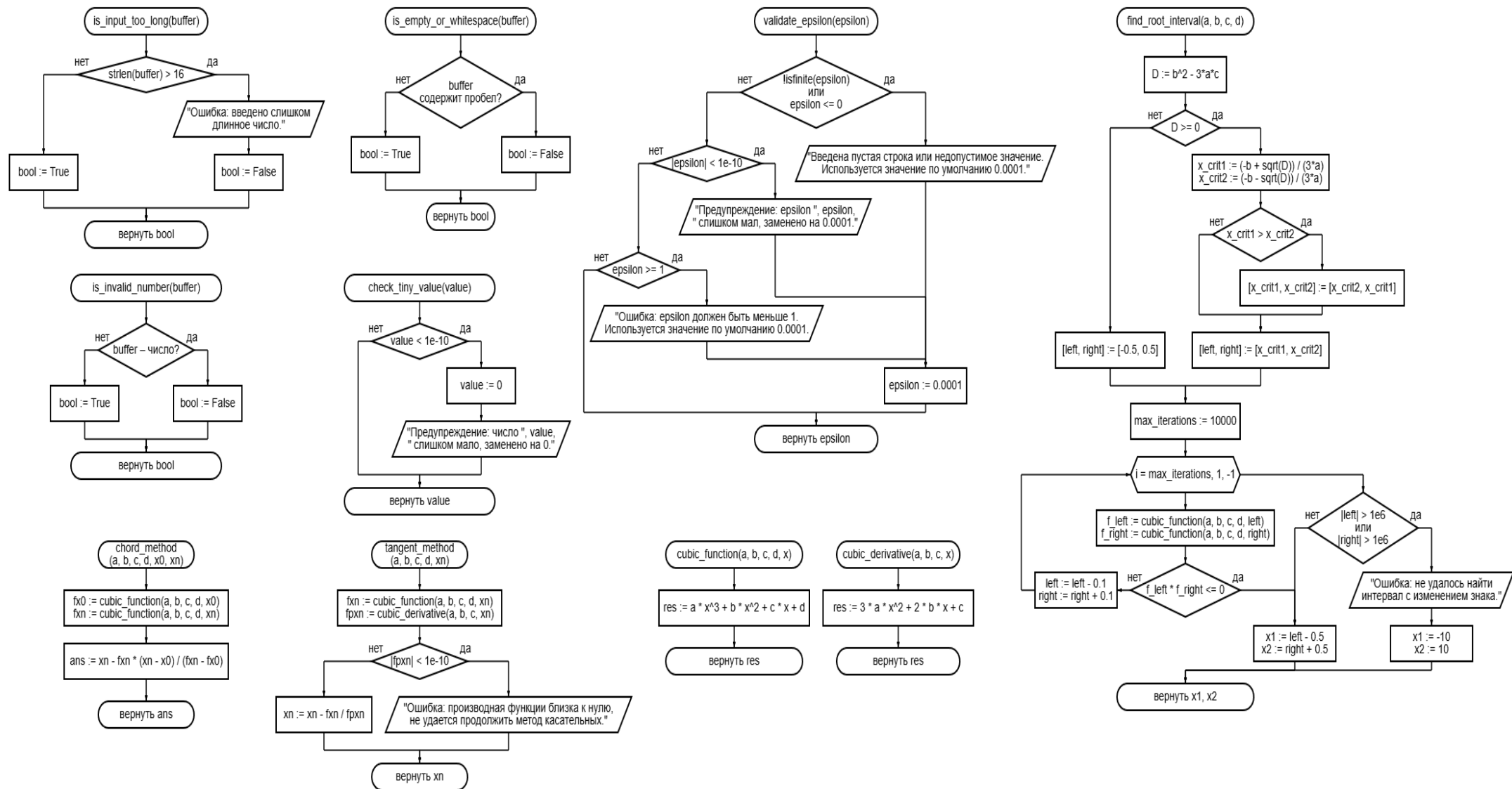


Рисунок 3 – Остальные функции

2 СПЕЦИФИКАЦИЯ ПЕРЕМЕННЫХ

Таблица 1 – Описание переменных, типы данных и их размеры

Переменная	Описание	Тип использования	Тип	Размер (байт)	Диапазон значений
a	Коэффициент при x^3	Входная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
b	Коэффициент при x^2	Входная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
c	Коэффициент при x	Входная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
d	Свободный член	Входная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
new_a	Коэффициент при x^2 после метода Горнера	Промежуточная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
new_b	Коэффициент при x после метода Горнера	Промежуточная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
new_c	Свободный член после метода Горнера	Промежуточная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
epsilon	Точность вычислений	Входная	double	8	(1e-10, 1) (ограничено программой)
x_new	Найденный корень кубического уравнения	Выходная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
x0	Левая граница интервала для поиска корня	Промежуточная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
xn	Правая граница интервала для поиска корня	Промежуточная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
x_chord	Значение корня, найденное методом хорд	Промежуточная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
x_tangent	Значение корня, найденное методом касательных	Промежуточная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
x	Корень линейного уравнения	Выходная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
D	Дискриминант квадратного уравнения	Промежуточная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
x1	Первый корень квадратного уравнения	Выходная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
x2	Второй корень квадратного уравнения	Выходная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
real	Действительная часть комплексного корня	Выходная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
imag	Мнимая часть комплексного корня	Выходная	double	8	$[-1.7977e + 308, 1.7977e + 308]$
buffer	Буфер для обработки вводов (чисел, у/п)	Промежуточная	char[18]	18	ASCII-символы (длина до 16 + \0)

Примечание: размер переменных в памяти указан для стандартных платформ x86-64.

3 РЕАЛИЗАЦИЯ ПРОГРАММЫ

3.1 Описание программы

Программа написана на языке C и состоит из двух основных файлов:

- 1) Заголовочный файл (`headers.h`) – содержит функции и объявления структур, которые используются в основном файле программы.
- 2) Основной файл (`main.c`) – включает основную функцию `main()`, которая управляет процессом выполнения программы.

3.2 Код программы

3.2.1 `headers.h`

```
#ifndef HEADERS_H
#define HEADERS_H

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <string.h>

#define EPSILON_DEFAULT 0.0001
#define TINY_VALUE 1e-10
#define INPUT_LIMIT 16

void clear_stdin() {
    int c;
    while ((c = getchar()) != '\n' && c != EOF);
}

int is_input_too_long(const char *buffer) {
    if (buffer[strlen(buffer) - 1] != '\n') {
        printf("Ошибка: введено слишком длинное число.\n");
        clear_stdin();
        return 1;
    }
    return 0;
}

int is_empty_or_whitespace(const char *str) {
    while (*str) {
        if (!isspace((unsigned char)*str)) return 0;
        str++;
    }
    return 1;
}

int is_invalid_number(const char *endptr) {
    while (isspace((unsigned char)*endptr)) endptr++;
    return *endptr != '\0' && *endptr != '\n';
}

double check_tiny_value(double value) {
    if (fabs(value) < TINY_VALUE && value != 0) {
        printf("Предупреждение: число %.5e слишком мало, заменено на 0.\n", value);
        return 0.0;
    }
}
```

```

    }
    return value;
}

double safe_input(const char *prompt) {
    char buffer[INPUT_LIMIT + 2];
    char *endptr;
    double value;
    while (1) {
        printf("%s", prompt);
        if (fgets(buffer, sizeof(buffer), stdin) == NULL) {
            printf("\nВвод прерван пользователем.\n");
            exit(0);
        }
        if (is_input_too_long(buffer)) continue;
        if (is_empty_or_whitespace(buffer)) {
            printf("Ошибка: ввод не должен быть пустым. Повторите ввод.\n");
            continue;
        }
        value = strtod(buffer, &endptr);
        if (is_invalid_number(endptr)) {
            printf("Некорректный ввод. Введите одно число.\n");
            continue;
        }
        if (!isfinite(value)) {
            printf("Ошибка: введено недопустимое значение.\n");
            continue;
        }
        return check_tiny_value(value);
    }
}

void solve_linear(double c, double d) {
    if (c == 0) {
        printf(d == 0 ? "\nРешение - любое число.\n" : "\nРешений нет.\n");
    } else {
        printf("\nРешение линейного уравнения: \n x = %.4f\n", -d / c);
    }
}

double round_to_4(double value) {
    return round(value * 10000.0) / 10000.0;
}

void solve_quadratic(double a, double b, double c, double epsilon) {
    double D = round_to_4(b * b - 4 * a * c);
    if (D > 0) {
        double sqrt_D = sqrt(D);
        double x1 = round_to_4((-b + sqrt_D) / (2 * a));
        double x2 = round_to_4((-b - sqrt_D) / (2 * a));
        if (fabs(x1) <= epsilon) x1 = 0.0;
        if (fabs(x2) <= epsilon) x2 = 0.0;
        printf("\nКорни квадратного уравнения: \n x1 = %.4f \n x2 = %.4f \n\n", x1, x2);
    } else if (D == 0) {
        double x = round_to_4(-b / (2 * a));
        if (fabs(x) <= epsilon) x = 0.0;
        printf("\nКорень квадратного уравнения: \n x1 = %.4f \n\n", x);
    } else {
        double real = round_to_4(-b / (2 * a));
        double imag = round_to_4(sqrt(-D) / (2 * a));
        if (fabs(real) <= epsilon) real = 0.0;
        if (fabs(imag) <= epsilon) imag = 0.0;
        printf("\nКомплексные корни квадратного уравнения: \n x1 = %.4f + %.4f*i \n x2 = %.4f - %.4f*i \n\n", real, imag, real, imag);
    }
}

double cubic_function(double a, double b, double c, double d, double x) {

```

```

    return a * x * x * x + b * x * x + c * x + d;
}

double cubic_derivative(double a, double b, double c, double x) {
    return 3 * a * x * x + 2 * b * x + c;
}

void find_root_interval(double a, double b, double c, double d, double *x1, double
*x2) {
    double D = b * b - 3 * a * c;
    double left, right;
    if (D >= 0) {
        double sqrt_D = sqrt(D);
        double x_crit1 = (-b + sqrt_D) / (3 * a);
        double x_crit2 = (-b - sqrt_D) / (3 * a);
        if (x_crit1 > x_crit2) {
            double temp = x_crit1;
            x_crit1 = x_crit2;
            x_crit2 = temp;
        }
        left = x_crit1;
        right = x_crit2;
    } else {
        left = -0.5;
        right = 0.5;
    }
    // Проверка смены знака
    double f_left = cubic_function(a, b, c, d, left);
    double f_right = cubic_function(a, b, c, d, right);
    int max_iterations = 10000;
    while (f_left * f_right > 0 && max_iterations-- > 0) {
        left -= 0.1;
        right += 0.1;
        f_left = cubic_function(a, b, c, d, left);
        f_right = cubic_function(a, b, c, d, right);
        if (fabs(left) > 1e6 || fabs(right) > 1e6) {
            printf("Ошибка: не удалось найти интервал с изменением знака.\n");
            *x1 = -10;
            *x2 = 10;
            return;
        }
    }
    *x1 = left - 0.5;
    *x2 = right + 0.5;
}

double chord_method(double a, double b, double c, double d, double x0, double xn) {
    double fx0 = cubic_function(a, b, c, d, x0);
    double fxn = cubic_function(a, b, c, d, xn);
    return xn - fxn * (xn - x0) / (fxn - fx0);
}

double tangent_method(double a, double b, double c, double d, double xn) {
    double fxn = cubic_function(a, b, c, d, xn);
    double fpxn = cubic_derivative(a, b, c, xn);
    if (fabs(fpxn) < TINY_VALUE) {
        printf("Ошибка: производная функции близка к нулю, не удается продолжить метод касательных.\n");
        return xn; // Избегаем деление на ноль
    }
    return xn - fxn / fpxn;
}

double validate_epsilon(double epsilon) {
    if (!isfinite(epsilon) || epsilon <= 0) {
        printf("Введена пустая строка или недопустимое значение. Используется значение по умолчанию %.4f.\n", EPSILON_DEFAULT);
        return EPSILON_DEFAULT;
    }
}

```

```

    }
    if (fabs(epsilon) < TINY_VALUE) {
        printf("Предупреждение: epsilon %.5e слишком мал, заменено на %.4f.\n",
epsilon, EPSILON_DEFAULT);
        return EPSILON_DEFAULT;
    }
    if (epsilon >= 1) {
        printf("Ошибка: epsilon должен быть меньше 1. Используется значение по
умолчанию %.4f.\n", EPSILON_DEFAULT);
        return EPSILON_DEFAULT;
    }
    return epsilon;
}

double input_epsilon() {
    char buffer[INPUT_LIMIT + 2];
    char *endptr;
    double epsilon;
    printf("Введите точность epsilon (по умолчанию %.4f): ", EPSILON_DEFAULT);
    if (fgets(buffer, sizeof(buffer), stdin) == NULL) {
        printf("\nВвод прерван пользователем.\n");
        exit(0);
    }
    if (is_input_too_long(buffer)) {
        printf("Заменено на %.4f.\n", EPSILON_DEFAULT);
        return EPSILON_DEFAULT;
    }
    epsilon = strtod(buffer, &endptr);
    if (is_invalid_number(endptr)) {
        printf("Некорректный ввод. Используется значение по умолчанию %.4f.\n",
EPSILON_DEFAULT);
        return EPSILON_DEFAULT;
    }
    return validate_epsilon(epsilon);
}

// Основная функция решения кубического уравнения методом хорд и касательных
void solve_cubic(double a, double b, double c, double d, double epsilon, double *root)
{
    double x0, xn, x_chord, x_tangent, x_new;
    int iteration = 0;
    find_root_interval(a, b, c, d, &x0, &xn);
    x_chord = chord_method(a, b, c, d, x0, xn);
    x_tangent = tangent_method(a, b, c, d, xn);
    x_new = (x_chord + x_tangent) / 2;
    do {
        x0 = x_chord;
        xn = x_tangent;
        x_chord = chord_method(a, b, c, d, x0, xn);
        x_tangent = tangent_method(a, b, c, d, xn);
        x_new = (x_chord + x_tangent) / 2;
        x_new = round_to_4(x_new); // Гарантированное округление на каждой итерации
        if (fabs(x_new) <= epsilon) {
            x_new = 0.0; // Принудительное округление к 0
        }
        iteration++;
        printf("Отрезок [%.4f; %.4f]\n", x0, xn);
    } while (fabs(x_new - x0) > epsilon && fabs(x_new - xn) > epsilon);
    x_new = round_to_4(x_new); // Финальное округление перед выводом
    if (fabs(x_new) <= epsilon) {
        x_new = 0.0;
    }

    printf("Всего итераций: %d\n", iteration);
    printf("\nНайденное приближение корня:\nх = %.4f\n\n", x_new);
    *root = x_new;
}

```

```

void horner(double a, double b, double c, double root, double *new_a, double *new_b,
double *new_c, double epsilon) {
    *new_a = a;
    *new_b = round_to_4((*new_a) * root + b);
    if (fabs(*new_b) <= epsilon) *new_b = 0.0; // Устранение ложных комплексных корней
    *new_c = round_to_4((*new_b) * root + c);
    if (fabs(*new_c) <= epsilon) *new_c = 0.0;
}

#endif

```

3.2.2 main.c

```

#include "headers1.h"

int main() {
    while (1) {
        double a = safe_input("Введите коэффициент a: ");
        double b = safe_input("Введите коэффициент b: ");
        double c = safe_input("Введите коэффициент c: ");
        double d = safe_input("Введите коэффициент d: ");
        printf("Введенное уравнение:\n");
        printf("%.4f*x^3 + %.4f*x^2 + %.4f*x + %.4f = 0\n\n", a, b, c, d);
        if (a == 0) {
            double epsilon = input_epsilon(); // Считать epsilon один раз
            (b == 0) ? solve_linear(c, d) : solve_quadratic(b, c, d, epsilon);
        } else {
            double new_a, new_b, new_c, x_new;
            double epsilon = input_epsilon();
            solve_cubic(a, b, c, d, epsilon, &x_new);
            horner(a, b, c, x_new, &new_a, &new_b, &new_c, epsilon);
            printf("Уравнение, полученное методом Горнера:\n");
            printf("%.4f*x^2 + %.4f*x + %.4f = 0\n", new_a, new_b, new_c);
            solve_quadratic(new_a, new_b, new_c, epsilon);
        }
        char response[10];
        while (1) {
            printf("Хотите ввести новое уравнение? (y/n): ");
            if (fgets(response, sizeof(response), stdin) == NULL) {
                printf("\nВвод прерван пользователем.\n");
                exit(0);
            }
            if (response[0] == 'n' && response[1] == '\n') {
                exit(0);
            }
            if (response[0] == 'y' && response[1] == '\n') {
                break;
            }
            printf("Некорректный ввод. Введите 'y' (да) или 'n' (нет).\n");
        }
    }
    return 0;
}

```


4.1.2 Решения кубических уравнений

```
(kali㉿kali)-[~/Desktop/algos1]
$ ./main
Введите коэффициент a: 3
Введите коэффициент b: 18
Введите коэффициент c: 27
Введите коэффициент d: 0
Введенное уравнение:
 $3x^3 + 18x^2 + 27x + 0 = 0$ 

Введите точность epsilon (по умолчанию 0.0001):
Введена пустая строка или недопустимое значение. Используется значение по умолчанию 0.0001.
Отрезок [-4.6667; 0.3333]
Отрезок [-0.7778; 0.0556]
Отрезок [-0.0436; 0.0019]
Отрезок [-0.0001; 0.0000]
Всего итераций: 4

Найденное приближение корня:
x = -0.0000

Уравнение, полученное методом Горнера:
 $3x^2 + 18x + 27 = 0$ 

Корень квадратного уравнения:
x1 = -3.0000

Найденные корни уравнения:
roots = [-3, 0]

Хотите ввести новое уравнение? (y/n): █
```

Рисунок 6 – Решение кубического уравнения

```
(kali㉿kali)-[~/Desktop/algos1]
$ ./main
Введите коэффициент a: 1
Введите коэффициент b: -6
Введите коэффициент c: 11
Введите коэффициент d: -6
Введенное уравнение:
 $1x^3 + -6x^2 + 11x + -6 = 0$ 

Введите точность epsilon (по умолчанию 0.0001):
Введена пустая строка или недопустимое значение. Используется значение по умолчанию 0.0001.
Отрезок [2.0000; 3.0076]
Отрезок [2.0000; 3.0001]
Отрезок [2.0000; 3.0000]
Отрезок [1.9326; 3.0000]
Всего итераций: 4

Найденное приближение корня:
x = 3.0000

Уравнение, полученное методом Горнера:
 $1x^2 + -3x + 2 = 0$ 

Корни квадратного уравнения:
x1 = 2.0000
x2 = 1.0000

Найденные корни уравнения:
roots = [1, 2, 3]

Хотите ввести новое уравнение? (y/n): █
```

Рисунок 7 – Решение кубического уравнения 2


```

(kali㉿kali)-[~/Desktop/algos1]
$ ./main
Введите коэффициент a: 1
Введите коэффициент b: 0
Введите коэффициент c: 0
Введите коэффициент d: -1000000
Введенное уравнение:
 $1x^3 + 0x^2 + 0x + -1000000 = 0$ 

Введите точность epsilon (по умолчанию 0.0001):
Введена пустая строка или недопустимое значение. Используется значение по умолчанию 0.0001.
Отрезок [98.8107; 100.0036]
Отрезок [100.0000; 100.0000]
Всего итераций: 2

Найденное приближение корня:
x = 100.0000

Уравнение, полученное методом Горнера:
 $1x^2 + 100x + 10000 = 0$ 

Комплексные корни квадратного уравнения:
x1 = -50.0000 + 86.6025*i
x2 = -50.0000 - 86.6025*i

Найденные корни уравнения:
roots = [-50 ± 86.6025*i, 100]

Хотите ввести новое уравнение? (y/n): █

```

Рисунок 8 – Решение кубического уравнения с комплексными корнями

4.1.3 Решения квадратных уравнений

```

(kali㉿kali)-[~/Desktop/algos1]
$ ./main
Введите коэффициент a: 0
Введите коэффициент b: 3
Введите коэффициент c: 18
Введите коэффициент d: 27
Введенное уравнение:
 $0x^3 + 3x^2 + 18x + 27 = 0$ 

Корень квадратного уравнения:
x1 = -3.0000

Найденные корни уравнения:
roots = [-3]

Хотите ввести новое уравнение? (y/n): █

```

Рисунок 9 – Решение квадратного уравнения


```
(kali㉿kali)-[~/Desktop/algos1]
$ ./main
Введите коэффициент a: 0
Введите коэффициент b: 1
Введите коэффициент c: 2
Введите коэффициент d: 5
Введенное уравнение:
 $0x^3 + 1x^2 + 2x + 5 = 0$ 

Комплексные корни квадратного уравнения:
x1 = -1.0000 + 2.0000*i
x2 = -1.0000 - 2.0000*i

Найденные корни уравнения:
roots = [-1 ± 2*i]

Хотите ввести новое уравнение? (y/n): █
```

Рисунок 10 – Решение квадратного уравнения с комплексными корнями

4.1.4 Проверка условий a, b, c, d

```
(kali㉿kali)-[~/Desktop/algos1]
$ ./main
Введите коэффициент a: 0
Введите коэффициент b: 0
Введите коэффициент c: 1
Введите коэффициент d: 5
Введенное уравнение:
 $0x^3 + 0x^2 + 1x + 5 = 0$ 

Решение линейного уравнения:
x = -5.0000

Найденные корни уравнения:
roots = [-5]

Хотите ввести новое уравнение? (y/n): █
```

Рисунок 11 – Решение линейного уравнения

```
(kali㉿kali)-[~/Desktop/algos1]
$ ./main
Введите коэффициент a: 0
Введите коэффициент b: 0
Введите коэффициент c: 0
Введите коэффициент d: 5
Введенное уравнение:
 $0x^3 + 0x^2 + 0x + 5 = 0$ 

Решений нет.
```

Рисунок 12 – $a = 0, b = 0, c = 0, d \neq 0$

```
(kali@kali)-[~/Desktop/algos1]
$ ./main
Введите коэффициент a: 0
Введите коэффициент b: 0
Введите коэффициент c: 0
Введите коэффициент d: 0
Введенное уравнение:
0*x^3 + 0*x^2 + 0*x + 0 = 0

Решение — любое число.
```

Рисунок 13 – $a = 0, b = 0, c = 0, d = 0$

4.1.5 Остальные проверки

Хотите ввести новое уравнение? (y/n):
Ввод прерван пользователем.

Рисунок 14 – Проверка на EOF стоит везде

Найденные корни уравнения:
roots = [-1.6506, -0.1747 ± 1.5469*i]

Хотите ввести новое уравнение? (y/n): y
Введите коэффициент a: █

Хотите ввести новое уравнение? (y/n): 5
Некорректный ввод. Введите 'y' (да) или 'n' (нет).
Хотите ввести новое уравнение? (y/n): h
Некорректный ввод. Введите 'y' (да) или 'n' (нет).
Хотите ввести новое уравнение? (y/n): asdasdasdsadads
Некорректный ввод. Введите 'y' (да) или 'n' (нет).

Рисунок 15 – Доп. проверка ввода

4.2 Сравнение результатов с примерами из учебных пособий

2) $x^3 - 0.2x^2 + 0.5x + 1.5 = 0$. Ответ: $x \approx -0.946$.

```
(kali@kali)-[~/Desktop/algos1]
$ ./main
Введите коэффициент a: 1
Введите коэффициент b: -0.2
Введите коэффициент c: 0.5
Введите коэффициент d: 1.5
Введенное уравнение:
1*x^3 + -0.2000*x^2 + 0.5000*x + 1.5000 = 0

Введите точность epsilon (по умолчанию 0.0001):
Введена пустая строка или недопустимое значение. Используется значение по умолчанию 0.0001.
Отрезок [-0.3818; 0.7218]
Отрезок [-1.8690; -0.4802]
Отрезок [-0.6775; -1.2772]
Отрезок [-0.8747; -1.0150]
Отрезок [-0.9422; -0.9501]
Отрезок [-0.9464; -0.9464]
Всего итераций: 6

Найденное приближение корня:
x = -0.9464

Уравнение, полученное методом Горнера:
1*x^2 + -1.1464*x + 1.5850 = 0

Комплексные корни квадратного уравнения:
x1 = 0.5732 + 1.1209*i
x2 = 0.5732 - 1.1209*i

Найденные корни уравнения:
roots = [-0.9464, 0.5732 ± 1.1209*i]

Хотите ввести новое уравнение? (y/n): █
```

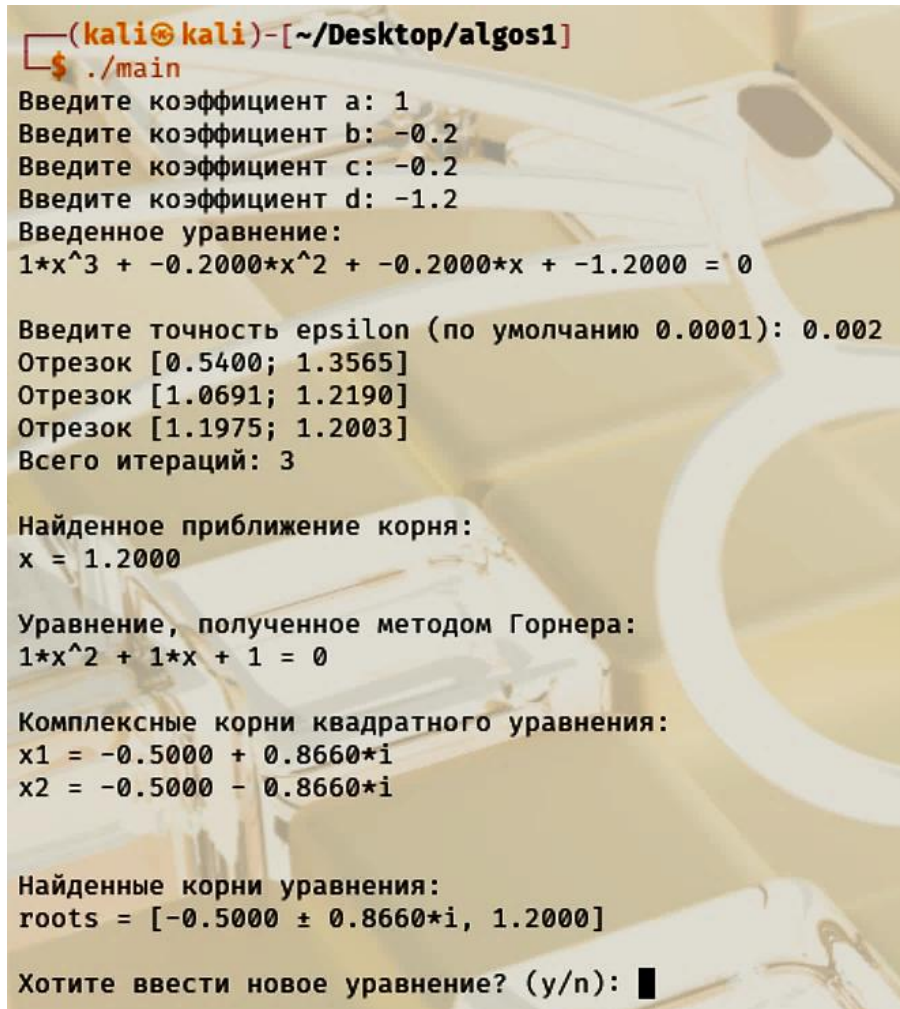
Рисунок 16 – Пример из первого источника (1)

Пример. Вычислить с точностью до 0,0005 единственный положительный корень уравнения

$$f(x) \equiv x^5 - x - 0,2 = 0.$$

Таким образом, $\xi = 1,198 + 0,002\theta$, где $0 < \theta \leq 1$.

Заметим, что точный корень уравнения (5) есть $\xi = 1,2$.



```
(kali@kali)-[~/Desktop/algos1]
$ ./main
Введите коэффициент a: 1
Введите коэффициент b: -0.2
Введите коэффициент c: -0.2
Введите коэффициент d: -1.2
Введенное уравнение:
1*x^3 + -0.2000*x^2 + -0.2000*x + -1.2000 = 0

Введите точность epsilon (по умолчанию 0.0001): 0.002
Отрезок [0.5400; 1.3565]
Отрезок [1.0691; 1.2190]
Отрезок [1.1975; 1.2003]
Всего итераций: 3

Найденное приближение корня:
x = 1.2000

Уравнение, полученное методом Горнера:
1*x^2 + 1*x + 1 = 0

Комплексные корни квадратного уравнения:
x1 = -0.5000 + 0.8660*i
x2 = -0.5000 - 0.8660*i

Найденные корни уравнения:
roots = [-0.5000 ± 0.8660*i, 1.2000]

Хотите ввести новое уравнение? (y/n): █
```

Рисунок 17 – Пример из второго источника (2)

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была разработана программа на языке C для решения кубического уравнения методом хорд и касательных. Так же программа включает в себя корректную обработку различных случаев, таких как линейные и квадратные уравнения, а также возможность нахождения комплексных корней. В результате были выполнены все этапы, от поиска интервала для корня до применения методов и нахождения всех корней уравнения.

Тестирование программы показало её корректную работу при различных входных данных, что подтверждает её практическую применимость для решения кубических уравнений.

Выполнение работы позволило закрепить навыки разработки алгоритмов для решения математических задач, а также углубить знания в области структур данных и численных методов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Воробьева Г. Н., Данилова А. Н. – Практикум по вычислительной математике: Учеб. пособие для техникумов. – 2-е изд., перераб. и доп. – М.: Высш. школа, 1990. – 208 с.
2. Демидович Б. П., Марон И. А. – Основы вычислительной математики: Учебное пособие. 8-е изд., стер. – СПб.: Издательство «Лань» – 2011. – 672 с.: ил. – (Учебники для вузов. Специальная литература).
3. Керниган, Брайан у., Ритчи, Деннис М. – Язык программирования С, 2-3 издание.: Пер. с англ. – М.: Издательский дом «Вильямс». – 2009. – 304 с.: ил. – Парал. тит. англ.