Часть 1. Вычисления. Погрешности вычислений

Задача 1.

Цель работы: изучение и оценка точности представления чисел.

Выполнение: текст программы на рисунке 1

```
#include <iostream>
#include <cmath>

using namespace std;

int task_1() {
    float y;
    cout << "До инициализации y = " << y << endl;
    y = 1;
    cout << "После инициализации y = " << y << endl;
    y = y / 6000;
    y = exp(y);
    y = sqrt(y);
    y = y / 14;
    y = 14 * y;
    y = y * y;
    y = log(y);
    y = 6000 * y;
    cout << "После преобразований y = " << y << endl;
    return 0;
}
```

Рисунок 1 — Текст программы

Примерные результаты при запуске программы:

До инициализации y = 4.34683e-41

После инициализации y = 1

После преобразований y = 0.999844

Абсолютная погрешность: $\Delta = |1 - 0.999844| = 0.000156$

Относительная погрешность: $\delta = \Delta / |A| = 0.000156 / 1 = 0.000156$

В этой задаче основными источниками погрешностей будут следующие:

- Погрешности округления: Связаны с ограниченным количеством разрядов, используемых для представления чисел типа float. Эти погрешности возникают на каждом этапе арифметических операций и накапливаются.
- Погрешности операций: Возникают при выполнении математических операций над приближенными числами. Например, функции exp, sqrt и log в стандартной библиотеке C++ могут давать небольшие погрешности из-за алгоритмов их вычисления.

Вывод: в ходе данной лабораторной работы была создана и протестирована программа на языке C++ для оценки погрешностей представления чисел и выполнения вычислений над числами типа float. Абсолютная и относительная погрешности оказались значительными, что показывает необходимость учета этих погрешностей при проведении вычислений высокой точности.

Задача 2

Цель работы: разработка программы для вычисления значений гиперболических функций и оценки погрешности вычислений при использовании различных типов данных (float, double, long double).

Выполнение: текст программы на рисунке 2

```cpp
#include <iostream>
#include <cmath>
#include <iomanip>

using namespace std;

void task_2() {
    long double x;
    cout << "Enter x: ";
    cin >> x;

    long double y1 = (exp(x) - exp(-x)) / 2;
    long double y2 = (exp(x) + exp(-x)) / 2;
    long double y = pow(y2, 2) - pow(y1, 2);

    cout << setprecision(16);
    cout << "x: " << x << endl;
    cout << "y1 (sh(x)) = " << setw(20) << y1 << endl;
    cout << "y2 (ch(x)) = " << setw(20) << y2 << endl;
    cout << "y (y2^2 - y1^2) = " << setw(20) << y << endl;
    cout << "Абсолютная погрешность: " << setw(20) << fabs(1 - y) << endl;
    cout << "Относительная погрешность: " << setw(20) << fabs(1 - y) / 1 << endl;
}
```

Рисунок 2 — Текст программы

Результаты при использовании float на рисунке 3

x	y1	y2	y	Δ	δ
5	74.20320892333984	74.20995330810547	1.00095546245575	0.0009554624557495117	0.0009554624557495117
10	11013.232421875	11013.232421875	0	1	1
15	1634508.625	1634508.625	0	1	1
20	242582592	242582592	0	1	1
25	36002451456	36002451456	0	1	1

Рисунок 3 — Результаты при использовании float

Результаты при использовании double на рисунке 4

x	y1	y2	y	Δ	δ
5	74.20321057778875	74.20994852478785	1.000000000001819	1.818989403545856e-12	1.818989403545856e-12
10	11013.23287470339	11013.23292010332	1.000000029802322	2.980232238769531e-08	2.980232238769531e-08
15	1634508.686235902	1634508.686236208	1	0	0
20	242582597.7048951	242582597.7048951	0	1	1
25	36002449668.69294	36002449668.69294	0	1	1

Рисунок 4 — Результаты при использовании double

Результаты при использовании long double на рисунке 5

x	y1	y2	y	Δ	δ
5	74.20321057778876	74.20994852478784	0.9999999999999991	8.881784197001252e-16	8.881784197001252e-16
10	11013.23287470339	11013.23292010332	1	0	0
15	1634508.686235902	1634508.686236208	1.000000238418579	2.384185791015625e-07	2.384185791015625e-07
20	242582597.7048951	242582597.7048951	1.00390625	0.00390625	0.00390625
25	36002449668.69294	36002449668.69294	0	1	1

Рисунок 5 — Результаты при использовании long double

Вывод: изменение типов переменных на double и long double значительно влияет на точность вычислений, особенно при больших значениях аргумента. Использование типов double и long double позволяет достичь более высокой точности, что особенно важно в задачах, требующих высокого уровня точности и минимизации ошибок округления.

Задача 3

Цель работы: проверка тригонометрического тождества с использованием численных методов.

Выполнение: для повышения точности вычисления был выбран тип long double, так как погрешность при его использовании минимальная. Текст программы на рисунке 6.

```cpp
#include "task_3.h"

#include <iostream>
#include <cmath>
#include <iomanip>

using namespace std;

void task_3() {
    long double x;
    cout << "Введите значение x (в радианах): ";
    cin >> x;

    long double sin2x = pow(x:sin(x), y:2); // sin^2(x)
    long double cos2x = pow(x:cos(x), y:2); // cos^2(x)
    long double result = sin2x + cos2x; // sin^2(x) + cos^2(x)

    cout << fixed << setprecision(n:16);
    cout << "sin^2(x) = " << setw(n:20) << sin2x << endl;
    cout << "cos^2(x) = " << setw(n:20) << cos2x << endl;
    cout << "sin^2(x) + cos^2(x) = " << setw(n:20) << result << endl;

    // Оценка погрешности
    long double delta = fabs(x:result - 1);
    cout << "Погрешность = " << setw(n:20) << delta << endl;
}
```

Рисунок 6 — Текст программы

Результаты работы на рисунке 7 (т к A = 1, абсолютная погрешность и относительная совпадают)

x	$sin^2(x)$	$cos^2(x)$	$sin^2(x)+cos^2(x)$	Δ δ
0	0.0000000000000000	1.0000000000000000	1.0000000000000000	0.0000000000000000
1.04	0.7437410511671797	0.2562589488328203	1.0000000000000000	0.0000000000000000
1.57	0.9999993658637698	0.0000006341362302	1.0000000000000000	0.0000000000000000
3.14	0.0000025365433124	0.9999974634566876	1.0000000000000000	0.0000000000000000

Рисунок 7 — результаты выполнения программы

Вывод: в ходе данной лабораторной работы была разработана и протестирована программа для проверки тригонометрического тождества $sin^2(x) + cos^2(x) = 1$. Программа продемонстрировала высокую точность вычислений, с минимальными погрешностями, что подтверждает надежность стандартных математических функций в языке программирования C++. Работа также подчеркнула важность учета погрешностей при выполнении численных вычислений и их оценки.

Часть 2. Программирование разветвляющегося вычислительного процесса

Цель работы: разработка программы для решения квадратных уравнений и изучение различных случаев решений, таких как два различных корня, один корень и отсутствие корней.

Задание: дано квадратное уравнение $ax^2+bx+c=0$, где a, b, c – действительные числа. Выяснить обладает ли оно действительными корнями, и если да, то найти их. В противном случае выдать сообщение об отсутствии действительных корней.

Проект программы: проект программы изображен на рисунке 8

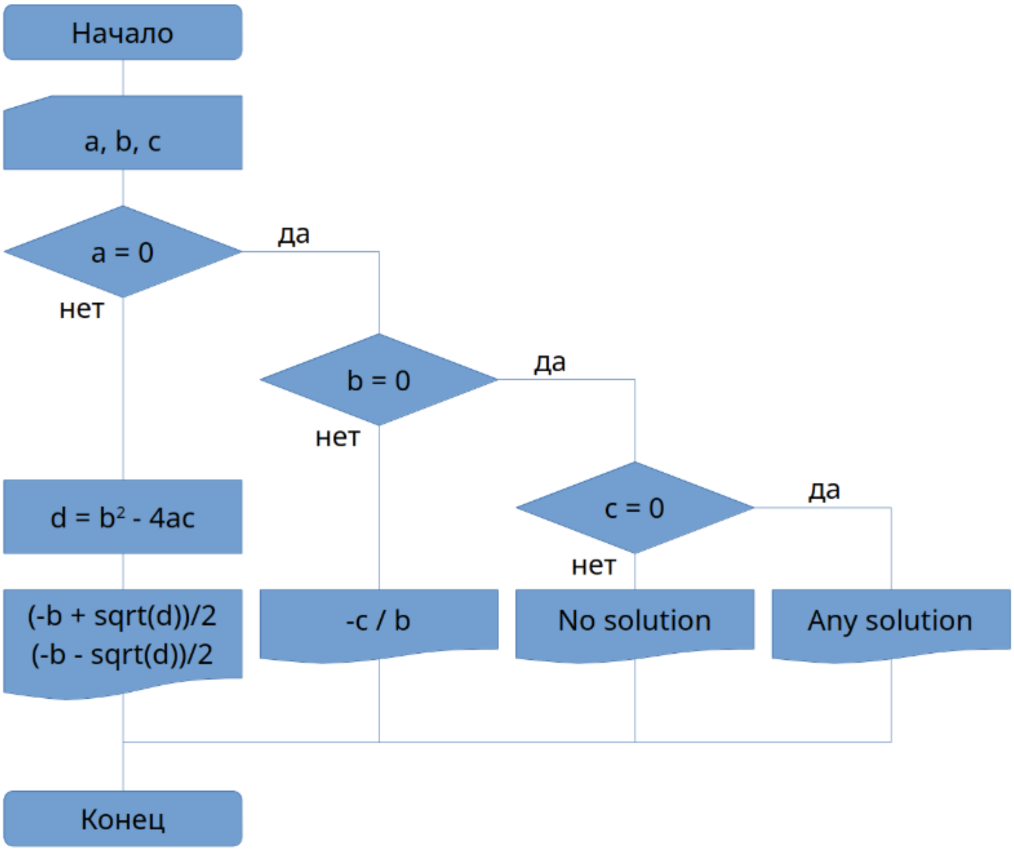

Рисунок 8 — Блок-схема алгоритма

Текст программы: текст программы изображен на рисунке 9

```cpp
#include <cmath>
#include <iostream>

using namespace std;

int main() {
    double a, b, c;
    cout << "This program solves ax^2+bx+c=0\n";
    cout << "Enter a: ";
    cin >> a;
    cout << "Enter b: ";
    cin >> b;
    cout << "Enter c: ";
    cin >> c;

    if (a == 0) {
        if (b == 0) {
            if (c == 0) {
                cout << "x = any (infinitely many solutions)\n";
            } else {
                cout << "No solution exists\n";
            }
        } else {
            cout << "x = " << -c / b << endl;
        }
    } else {
        const double d = pow(b, 2) - 4 * a * c;
        if (d < 0) {
            cout << "No solution exists\n";
        } else if (d == 0) {
            cout << "x = " << -b / (2 * a) << endl;
        } else {
            cout << "x1 = " << (-b + sqrt(d)) / (2 * a) << endl;
            cout << "x2 = " << (-b - sqrt(d)) / (2 * a) << endl;
        }
    }
    return 0;
}
```

Рисунок 9 — текст программы

Тестовые данные и результаты тестирования: тестовые данные и результаты тестирования изображены на рисунке 10

a	b	c	Результат
1	-3	2	x1 = 2 x2 = 1
1	2	1	x = -1
1	1	1	No solution exists
0	0	0	x = any (infinitely many solutions)
0	2	-4	x = 2

Рисунок 10 — Тестовые данные и результат выполнения

Вывод: в результате выполнения лабораторной работы была создана программа для решения квадратного уравнения вида $ax^2+bx+c=0$. Программа корректно обрабатывает все возможные случаи, включая линейные уравнения и ситуации, когда решений нет. Тестирование показало, что программа работает правильно и дает ожидаемые результаты для различных наборов входных данных.

Часть 3. Программирование циклического процесса.

Цель работы: вычисление площади, ограниченной функцией y = ln(x) и осью x на интервале [a, b], с заданной точностью eps с использованием метода трапеций, а также определение зависимости числа итераций от точности вычислений.

Задание: Решить задачу с точностью ξ, организовав итерационный цикл. Значение точности вводится с клавиатуры. Вычислить значение площади, ограниченной функцией y=ln(x) и осью x на интервале по формуле: $S=\frac{b-a}{n}\sum_{i=1}^{n} f(x_i)$, где n – количество отрезков разбиения, a,b – начало и конец интервала. Проверить программу для точности $\xi=10^{-3}$, 10^{-4} и a=1, b=2. Определить, как изменяется число итераций при изменении точности. Считать точным значением площади 0,3862943611199.

Выполнение: схема алгоритма изображена на рисунке 11, текст программы изображен на рисунке 12

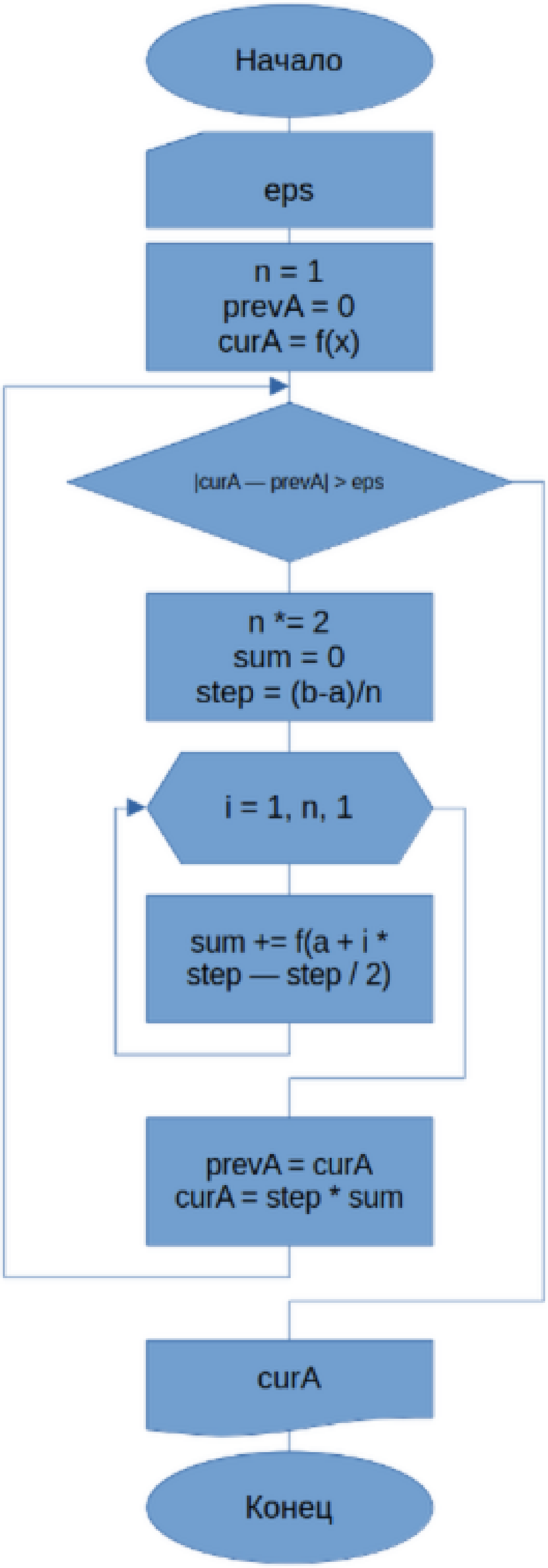

Рисунок 11 — схема алгоритма

```cpp
#include <cmath>
#include <iostream>
#include <iomanip>

using namespace std;

typedef long double ld;

ld function(const ld x) {
    return log(x);
}

ld integrate(const ld a, const ld b, const ld eps) {
    int n = 1;
    ld previousArea = 0;
    ld currentArea = ((b - a) / n) * function((b - a) / 2);

    while (fabs(currentArea - previousArea) > eps) {
        n *= 2;
        ld sum = 0;
        ld step = (b - a) / n;
        for (int i = 1; i <= n; ++i) {
            sum += function(a + i * step - step / 2);
        }
        previousArea = currentArea;
        currentArea = step * sum;
    }
    return currentArea;
}

int main() {
    ld eps;
    cout << "Enter eps: ";
    cin >> eps;

    ld area = integrate(1, 2, eps);
    cout << "The estimated area is: " << setprecision(16) << area << endl;
    return 0;
}
```

Рисунок 12 — текст программы

Тестовые данные и результат выполнения: на рисунке 13

eps	Вывод
0.1	0.3875883104947483
0.001	0.3866193655376412
0.0001	0.3863147041444535
0.00000001	0.386294362361654

Рисунок 13 - Тестовые данные и результат выполнения

Вывод: программа вычисляет площадь под кривой y = ln(x) с разной точностью и показывает, что число итераций увеличивается при уменьшении значения ϵ. Это подтверждает необходимость большего количества интервалов для достижения более высокой точности. Результаты программы близки к точному значению площади 0.3862943611199, что свидетельствует о корректности реализации метода трапеций.