Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе

«Суммирование рядов математических функций»

Выполнил:

студент группы 3822Б1ПМ1

Сомов Я.В.

Проверил:

преподаватель кафедры МОСТ

Волокитин В.Д.

Нижний Новгород

2023 г.

Оглавление

1. Постановка задачи

Вычисление математических функций — интересная с технической точки зрения задача. В данной лабораторной работе предлагается реализовать подсчёт таких математических функций, как $e^x, \sin x, \cos x, \ln(1+x)$ с помощью рядов Тейлора. Предполагается работа с типом данных float.

Цель работы — реализовать и сравнить два метода суммирования: прямой и обратный.

Для достижения цели необходимо решить следующие задачи:

- создать алгоритм вычисления слагаемого из разложения функции в ряд Тейлора;
- реализовать алгоритмы прямого и обратного суммирования;
- экспериментальным путём проверить корректность работы алгоритмов и сравнить результаты их работы.

2. Метод решения

Для упрощения одной из поставленных задач мы воспользуемся разложением функций в ряд по Маклорену. Напомним общий вид разложений для рассматриваемых в рамках лабораторной работы функций.

$$e^x = \sum_{n=0}^{+\infty} \frac{x^n}{n!} \tag{2.1}$$

$$\sin x = \sum_{n=0}^{+\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} \tag{2.2}$$

$$\cos x = \sum_{n=0}^{+\infty} \frac{(-1)^n x^{2n}}{(2n)!} \tag{2.3}$$

$$\ln(1+x) = \sum_{n=1}^{+\infty} \frac{(-1)^{n-1} x^n}{n} \tag{2.4}$$

«Наивная» реализация вычисления суммы с непосредственным подсчётом каждого из слагаемых будет работать достаточно долго (каждый раз программа будет считать факториал), поэтому будем вычислять слагаемые при помощи рекуррентных соотношений вида $a_i = K a_{i-1}$, где $K = \dfrac{a_i}{a_{i-1}}$.

Выведем рекуррентные соотношения для наших математических функций. Введём дополнительное обозначение: a_i — i-е слагаемое в разложении по Маклорену.

$$2.1 = \frac{a_i}{a_{i-1}} = \frac{\frac{x^i}{(i)!}}{\frac{x^{i-1}}{(i-1)!}} = \frac{x}{i}$$

$$2.2 = \frac{a_i}{a_{i-1}} = \frac{\frac{(-1)^i x^{2i+1}}{(2i+1)!}}{\frac{(-1)^{i-1} x^{2i-1}}{(2i-1)!}} = -\frac{x^2}{2i(2i+1)}$$

$$2.3 = \frac{a_i}{a_{i-1}} = \frac{\frac{(-1)^i x^{2i}}{(2i)!}}{\frac{(-1)^{i-1} x^{2i-2}}{(2i-2)!}} = -\frac{x^2}{2i(2i-1)}$$

$$2.4 = \frac{a_i}{a_{i-1}} = \frac{\frac{(-1)^{i-1}x^i}{i}}{\frac{(-1)^{i-2}x^{i-1}}{i-1}} = \frac{xi}{i+1}$$

Для обратного суммирования предлагается следующий алгоритм отыскания слагаемого в ряде Маклорена: ищется последнее ненулевое слагаемое, затем оно умножается на величины, обратные к полученным, до тех пор, пока не будет достигнуто самое первое слагаемое. Оно будет отдельно подсчитано в нормальном порядке и прибавлено к сумме.

3. Руководство пользователя

Работа пользователя с программой осуществляется посредством интерфейса командной строки (Command line interface, CLI). Пользователю предлагаются на выбор два режима работы: режим написания отчёта и режим непосредственного вычисления значения функции.

Программа в режиме написания отчёта будет вычислять на указанном пользователем интервале значения функций, полученных методами прямого и обратного суммирования, а также значения, полученные функциями из библиотеки math.h. Пользователю предлагается ввести три значения: нижнюю границу интервала, верхнюю границу интервала, шаг вычислений. По окончании работы программа сохранит результаты в файлы exp.log, sin.log, cos.log, ln.log, которые пользователь сможет впоследствии самостоятельно обработать (например, при помощи стороннего скрипта). Данные в файлах хранятся следующим образом: значение x; результат прямого суммирования; результат обратного суммирования; результат вычисления библиотечной функцией.

Программа в режиме непосредственного вычисления значения рассчитает значения выбранной функции, полученной методами прямого и обратного суммирования, а также значения, полученное функцией из библиотеки math.h. Пользователь выбирает одну из четырёх функций и вводит значение x, для которого будет вычислено значение. На выходе пользователь получит информацию о результатах прямого, обратного суммирования, значение, вычисленное библиотечной функцией, абсолютную и относительную погрешность методов по отношению к значению, вычисленного библиотечной функцией.

4. Описание программной реализации

Проект состоит из пяти файлов:

- func.h — содержит прототипы функций вычисления значения слагаемого и методов прямого и обратного суммирования
- func.c — содержит реализацию функций из func.h
- output.h — содержит прототипы функций, предназначенных для работы с пользователем
- output.c — содержит реализацию функций из output.h
- main.c — главный файл программы

В таблице 4 содержится основная информация о реализованных функциях: типы возвращаемых значений, прототипы подпрограмм, краткое описание.

Таблица 1: Реализованные в программе функции

Тип возвращаемого значения	**Прототип функции**	**Описание**
float	u_expf(float x, float t, int i, int mode)	Рекуррентно вычисляет слагаемое в разложении e^x по Маклорену (на основании ранее вычисленного слагаемого). Принимает значение точки x, в которой считается значение функции, t — предыдущее слагаемое, i — номер текущего слагаемого. Параметр mode определяет режим работы: 0 для прямого и 1 для обратного суммирования

продолжение на следующей странице

Таблица 1 – *продолжение*

Тип возвращаемого значения	**Прототип функции**	**Описание**
float	u_sinf(float x, float t, int i, int mode)	Рекуррентно вычисляет слагаемое в разложении $sinx$ по Маклорену (на основании ранее вычисленного слагаемого). Принимает значение точки x, в которой считается значение функции, t — предыдущее слагаемое, i — номер текущего слагаемого. Параметр mode определяет режим работы: 0 для прямого и 1 для обратного суммирования
float	u_cosf(float x, float t, int i, int mode)	Рекуррентно вычисляет слагаемое в разложении $\cos x$ по Маклорену (на основании ранее вычисленного слагаемого). Принимает значение точки x, в которой считается значение функции, t — предыдущее слагаемое, i — номер текущего слагаемого. Параметр mode определяет режим работы: 0 для прямого и 1 для обратного суммирования
float	u_lnf(float x, float t, int i, int mode)	Рекуррентно вычисляет слагаемое в разложении $\ln(1+x)$ по Маклорену (на основании ранее вычисленного слагаемого). Принимает значение точки x, в которой считается значение функции, t — предыдущее слагаемое, i — номер текущего слагаемого. Параметр mode определяет режим работы: 0 для прямого и 1 для обратного суммирования
float	direct_sum(float x, float (*fun)(float, float, int, int))	Алгоритм прямого суммирования слагаемых из разложения по Маклорену. Принимает значение точки x, в которой считается значение функции и указатель на функцию, вычисляющую слагаемое.

продолжение на следующей странице

Таблица 1 – *продолжение*

Тип возвращаемого значения	**Прототип функции**	**Описание**
float	reverse_sum(float x, float (*fun)(float, float, int, int))	Алгоритм обратного суммирования слагаемых из разложения по Маклорену. Принимает значение точки x, в которой считается значение функции и указатель на функцию, вычисляющую слагаемое.
void	write_report(float lower_bound, float upper_bound, float delta)	Вычисляет значения всех четырёх функций на указанном отрезке с указанным шагом, выводит результаты вычислений в четыре различных текстовых файла.
float	cli_calc(float x, int chosen_function, int mode)	Вычисляет значения выбранной пользователем точки с помощью выбранного режима (0 — прямое суммирование, 1 — обратное суммирование, 2 — реализация функции из библиотеки math.h), возвращает полученное значение.

5. Подтверждение корректности

Для подтверждения корректности результатов выполнения разработанных алгоритмов используются математические функции из стандартной библиотеки языка C. Программа выводит значения, полученные с помощью этих функций, чтобы пользователь имел возможность сравнить результаты выполнения.

6. Результаты экспериментов

Рис. 1: Абсолютная погрешность вычислений на отрезке $[11; 12, 5]$ для $e^x, \sin x, \cos x$ и на отрезке $[-0, 99; 1]$ для $\ln(1 + x)$ (шаг 0,0032)

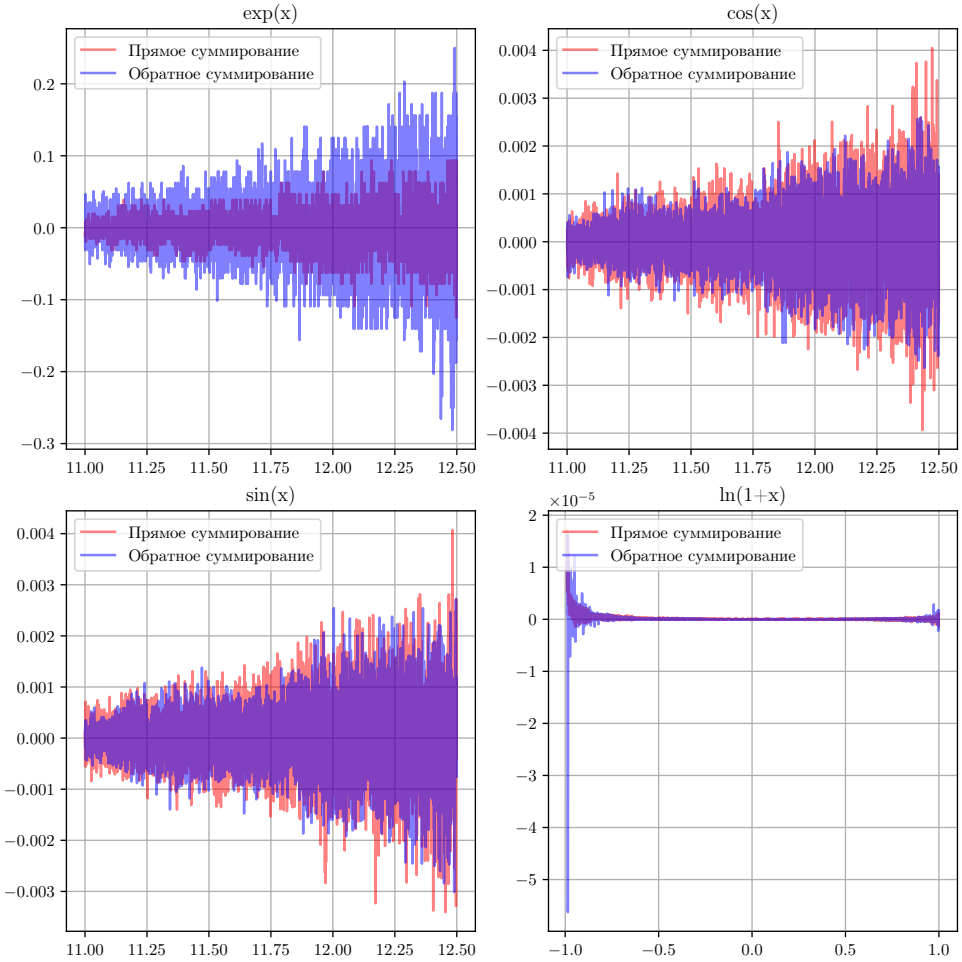


Рис. 2: Относительная погрешность вычислений на отрезке $[11; 12, 5]$ для $e^x, \sin x, \cos x$ и на отрезке $[-0, 99; 1]$ для $\ln(1 + x)$ (шаг 0,0032)

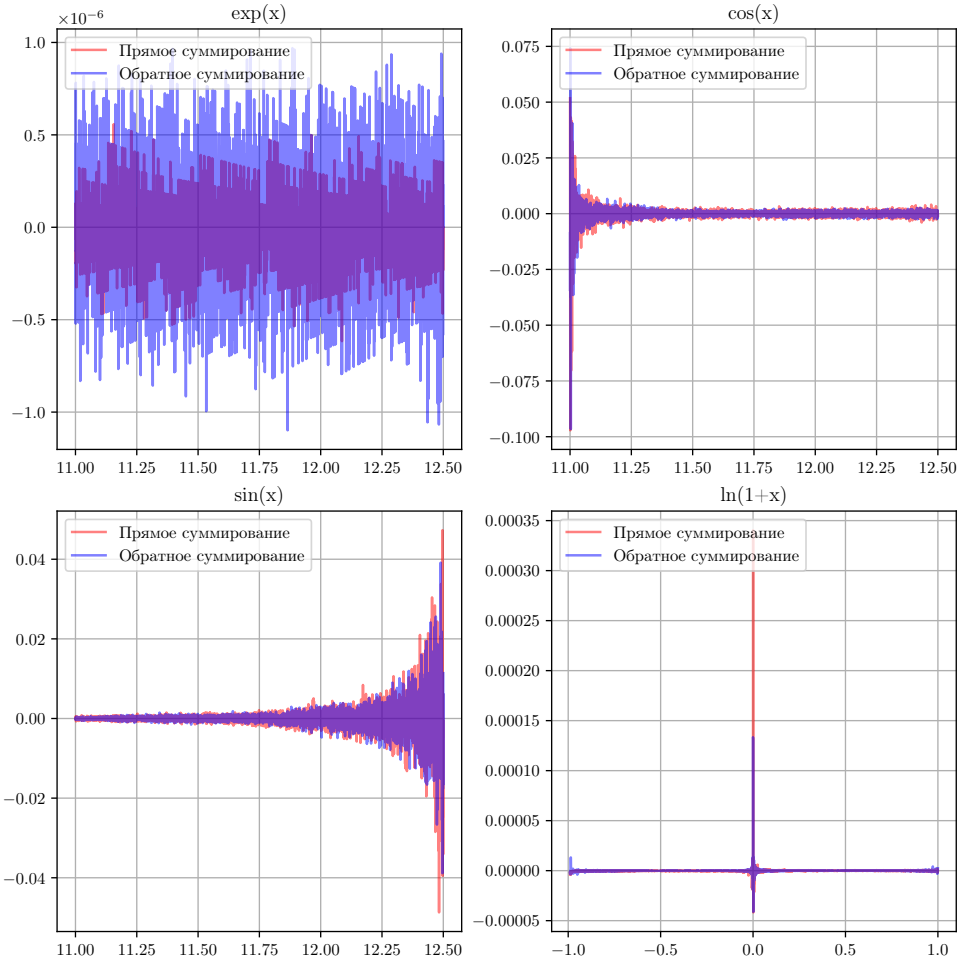


Рис. 3: Абсолютная погрешность вычислений на отрезке $[0,1;1,5]$ для $e^x, \sin x, \cos x$ и на отрезке $[0,1;1,0]$ для $\ln(1+x)$ (шаг 0,0032)

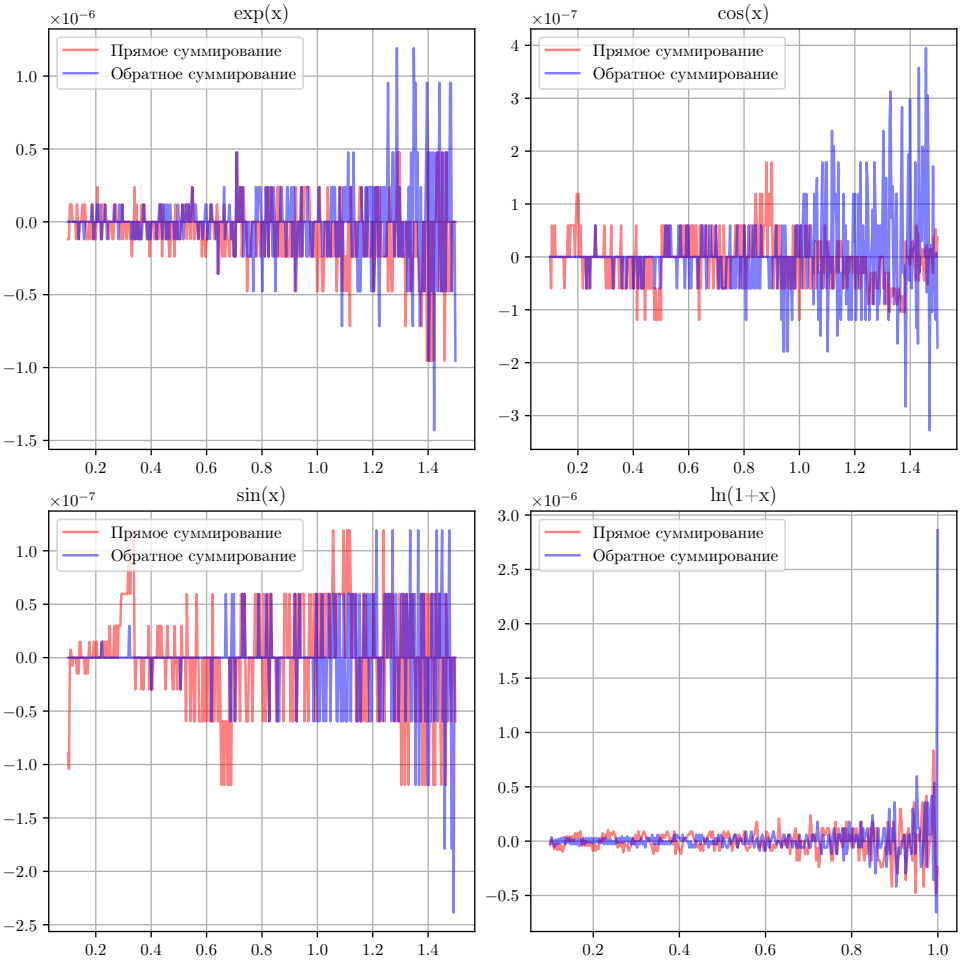


Рис. 4: Относительная погрешность вычислений на отрезке $[0,1;1,5]$ для $e^x, \sin x, \cos x$ и на отрезке $[0,1;1,0]$ для $\ln(1+x)$ (шаг 0,0032)

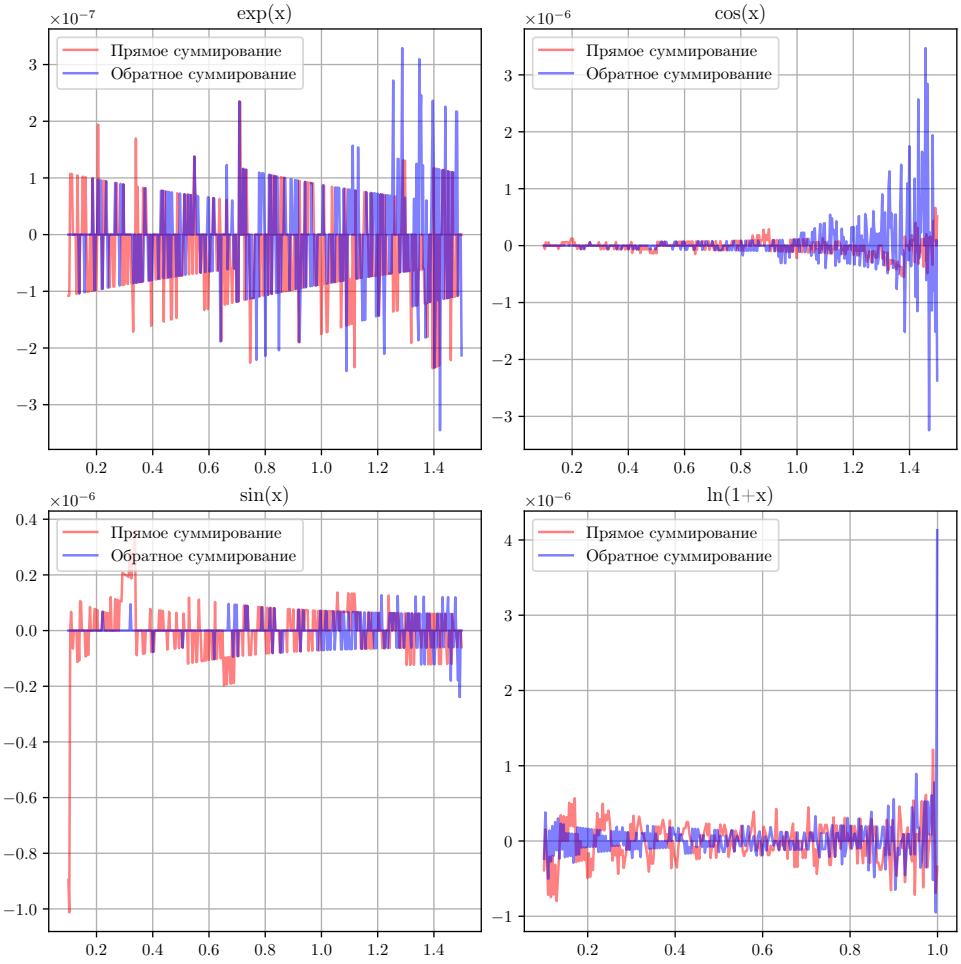


Рис. 5: Абсолютная погрешность вычислений на отрезке $[-7,5;-6]$ для $e^x, \sin x, \cos x$ (шаг 0,0032)

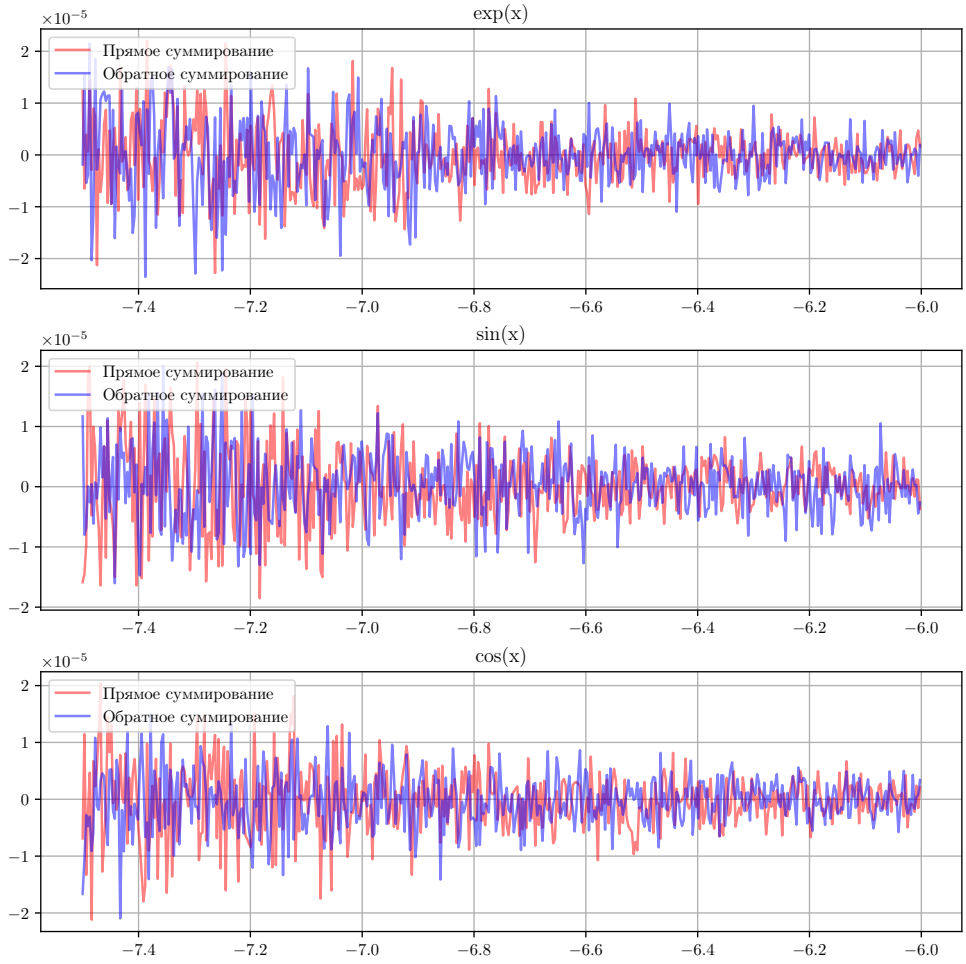
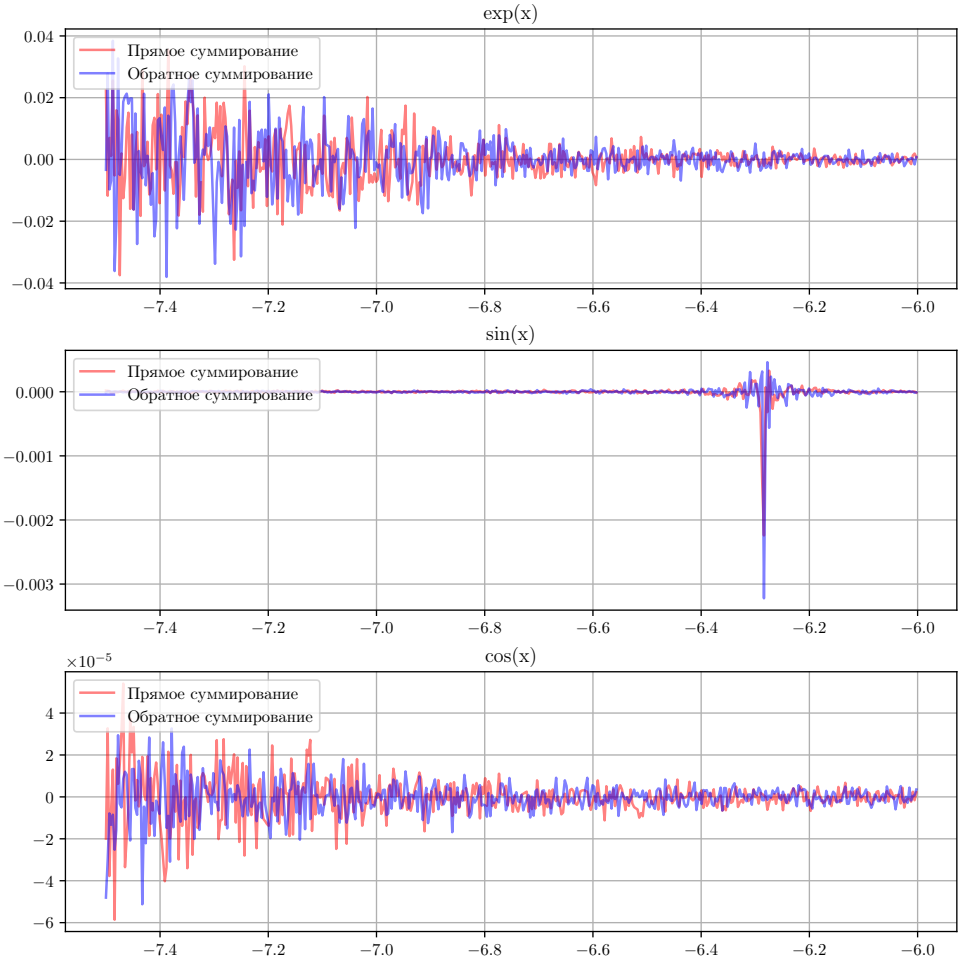


Рис. 6: Относительная погрешность вычислений на отрезке $[-7,5;-6]$ для $e^x, \sin x, \cos x$ и на отрезке $[0,1;1,0]$ для $\ln(1+x)$ (шаг 0,0032)



Полученные в результате выполнения данные свидетельствуют о том, что метод обратного суммирования в большинстве случаев работает точнее, чем прямое суммирование.

7. Заключение

В результате проведения лабораторной работы была достигнута поставленная цель и выполнены следующие исследовательские задачи:

- разработаны и реализованы алгоритмы прямого и обратного суммирования, вычисления слагаемых разложения функций в ряд Маклорена;
- экспериментальным путём проверена корректность и работоспособность разработанных алгоритмов.

.

8. Приложение

Листинг 1: func.h

```h
#pragma once

#define EPS 1.0e-7f

// mode:
// 1 - считает слагаемые для обратного суммирования
// 0 - считаем слагаемые для прямого суммирования

float u_expf(float x, float t, int i, int mode);
float u_sinf(float x, float t, int i, int mode);
float u_cosf(float x, float t, int i, int mode);
float u_lnf(float x, float t, int i, int mode);
float direct_sum(float x, float (*fun)(float, float, int, int));
float reverse_sum(float x, float (*fun)(float, float, int, int));
```

Листинг 2: func.c

```c
#include "func.h"
#include "math.h"

float u_expf(float x, float t, int i, int mode)
{
        if (i == 0) return 1.0f;
        return (mode ?
                t * i / x :
                t * x / i);
}

float u_sinf(float x, float t, int i, int mode)
{
        if (i == 0) return x;
        return (mode ?
                -1.0f * t * (2 * i * (2 * i + 1)) / (x * x) :
                -1.0f * t * x * x / (2 * i * (2 * i + 1)));
}

float u_cosf(float x, float t, int i, int mode)
{
        if (i == 0) return 1.0f;
        return (mode ?
                -1.0f * t * (2 * i * (2 * i - 1)) / (x * x) :
                -1.0f * t * x * x / (2 * i * (2 * i - 1)));
}

float u_lnf(float x, float t, int i, int mode)
{
```

```
        if (i == 0) return x;
        return (mode ?
                -1.0f * t * (i + 1) / (i * x) :
                -1.0f * t * i * x / (i + 1));
}

float direct_sum(float x, float (*fun)(float, float, int, int))
{
        float res = 0.0f;
        float term = 0.0f;

        for (int i = 0; fabs(term = fun(x, term, i, 0)) > EPS; i++)
                res += term;

        return res;
}

float reverse_sum(float x, float (*fun)(float, float, int, int))
{
        float res = 0.0f;
        float term = 0.0f;
        int i = 0;

        // ищем последнее ненулевое слагаемое...
        while (fabs(term = fun(x, term, i, 0)) > EPS) i++;

        // добавляем слагаемые с конца
        for (; i > 0; i--)
        {
                res += term;
                term = fun(x, term, i, 1);
        }

        // добавляем неучтённое при обратном суммировании первое слагаемое
        return res + fun(x, 0.0f, 0, 0);
}
```

Листинг 3: output.h

```
#pragma once

#define MATH_FUNCTIONS_COUNT 4

typedef float (*std_math_func)(float x);
typedef float (*user_math_func)(float x, float t, int i, int mode);

void write_report(float lower_bound, float upper_bound, float delta);
float cli_calc(float x, int chosen_function, int mode);
```

Листинг 4: output.c

```
#include "output.h"
#include "func.h"
#include "stdio.h"
#include "math.h"

void write_report(float lower_bound, float upper_bound, float delta)
{
```

```
8           float x;
9           float stdres, dirres, revres;
10          std_math_func std_math_func[] = { expf, sinf, cosf, logf };
11          user_math_func user_math_func[] = { u_expf, u_sinf, u_cosf, u_lnf };
12          const char filename[MATH_FUNCTIONS_COUNT][255] = { "exp.log", "sin.log",
            ↪  "cos.log", "ln.log" };
13          FILE* f;
14
15          for (int i = 0; i < MATH_FUNCTIONS_COUNT - 1; i++)
16          {
17                  x = lower_bound;
18                  fopen_s(&f, filename[i], "w");
19                  while (x < upper_bound)
20                  {
21                          stdres = std_math_func[i](x);
22                          dirres = direct_sum(x, user_math_func[i]);
23                          revres = reverse_sum(x, user_math_func[i]);
24                          fprintf(f, "%.10f %.10f %.10f %.10f\n", x, stdres, dirres,
                            ↪  revres);
25                          x += delta;
26                  }
27                  fclose(f);
28          }
29
30          // ln(1+x) обрабатываем отдельно, поскольку он не существует для x < -1.0
31          x = (lower_bound < -0.99f ? -0.99f : lower_bound);
32          // переставим верхнюю границу, если она больше 1.0 (иначе программа будет
            ↪  работать слишком долго)
33          x = (lower_bound > 1.0f ? x = -0.99f : x);
34          (upper_bound > 1.0f ? upper_bound = 1.0f : 0);
35          fopen_s(&f, filename[3], "w");
36          while (x < upper_bound)
37          {
38                  stdres = std_math_func[3](1 + x);
39                  dirres = direct_sum(x, user_math_func[3]);
40                  revres = reverse_sum(x, user_math_func[3]);
41                  fprintf(f, "%.10f %.10f %.10f %.10f\n", x, stdres, dirres, revres);
42                  x += delta;
43          }
44          fclose(f);
45  }
46
47  float cli_calc(float x, int chosen_function, int mode)
48  {
49          std_math_func std_math_func[] = { expf, sinf, cosf, logf };
50          user_math_func user_math_func[] = { u_expf, u_sinf, u_cosf, u_lnf };
51
52          switch (mode)
53          {
54          case 0:
55                  return direct_sum(x, user_math_func[chosen_function]);
56          case 1:
57                  return reverse_sum(x, user_math_func[chosen_function]);
58          case 2:
59                  return (chosen_function == 3 ? std_math_func[chosen_function](1+x) :
                    ↪  std_math_func[chosen_function](x));
60          default:
61                  return NAN;
62          }
63  }
```

Листинг 5: main.c

```c
#include "func.h"
#include "output.h"

#include "math.h"
#include "stdio.h"
#include "stdlib.h"
#include "locale.h"

int main()
{
        int choice;

        float lower_bound, upper_bound, delta;
        float x;
        float result_dir, result_rev, result_std;

        setlocale(LC_CTYPE, "RUSSIAN");

        printf("Суммирование рядов математических функций\n");
        printf("-----------------\n");
        printf("Выбор:\n");
        printf("1. Создать отчёт\n");
        printf("2. Посчитать значение функции напрямую\n");
        printf("> ");
        scanf_s("%d", &choice);

        switch (choice)
        {
        case 1:
                printf("Введите нижнюю, верхнюю границу вычисления значений функций и
                ↪  шаг вычислений. Программа создаст четыре лог-файла, содержащих
                ↪  значения, вычисленных на указанном промежутке.\n> ");
                scanf_s("%f%f%f", &lower_bound, &upper_bound, &delta);
                write_report(lower_bound, upper_bound, delta);
                break;
        case 2:
                printf("Выберите функцию (введите любую из цифр):\n");
                printf("1. exp(x):\n");
                printf("2. sin(x):\n");
                printf("3. cos(x):\n");
                printf("4. ln(1+x):\n");
                printf("> ");
                scanf_s("%d", &choice);

                if ((choice > MATH_FUNCTIONS_COUNT) || (choice < 1))
                {
                        printf("Введено недопустимое значение.\n");
                }

                else
                {
                        choice--;
                        printf("Введите значение x, для которого хотите вычислить
                        ↪  значение.\n> ");
                        scanf_s("%f", &x);
                        result_dir = cli_calc(x, choice, 0);
                        result_rev = cli_calc(x, choice, 1);
```

```
55                          result_std = cli_calc(x, choice, 2);
56                          printf("Полученные значения: прямое суммирование, обратное
                            ↪  суммирование, функция из math.h\n");
57                          printf("%.8f %.8f %.8f\n", result_dir, result_rev, result_std);
58                          printf("Абсолютная погрешность\n");
59                          printf("%.8f %.8f\n", result_dir - result_std, result_rev -
                            ↪  result_std);
60                          printf("Относительная погрешность\n");
61
62                          (fabs(result_std) < EPS) ?
63                                  printf("%.8f %.8f\n", result_dir - result_std,
                                    ↪  result_rev - result_std) :
64                                  printf("%.8f %.8f\n", (result_dir - result_std) /
                                    ↪  result_std, (result_rev - result_std) /
                                    ↪  result_std);
65
66                  }
67
68                  break;
69          default:
70                  printf("Выбор не распознан.\n");
71                  break;
72          }
73
74          return 0;
75  }
```