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

**Отчет по лабораторной работе**

**«Вычисление математических функций с использованием рядов»**

**Выполнил**:

студент группы 3824Б1ПМ1

Бутусов А.П.

**Проверил**:

преподаватель каф. ВВСП,

Волокитин В.Д.

Нижний Новгород

2025

**Содержание**

[Постановка задачи 3](#_Toc192441262)

[Метод решения 4](#_Toc192441263)

[Руководство пользователя 6](#_Toc192441264)

[Описание программной реализации 9](#_Toc192441265)

[Подтверждение корректности 14](#_Toc192441266)

[Результаты экспериментов 17](#_Toc192441267)

[Заключение 27](#_Toc192441268)

[Приложение 28](#_Toc192441269)

# Постановка задачи

В рамках данной лабораторной работы необходимо:

Требуется реализовать на языке программирования C подсчёт значений четырёх математических функций:

с использованием разложения этих функций в ряды Тейлора. Для каждой функции необходимо реализовать три метода суммирования:

* Прямое суммирование — последовательное сложение членов ряда.
* Обратное суммирование — сложение членов ряда в обратном порядке.
* Попарное суммирование — сложение членов ряда попарно (например, первый с последним, второй с предпоследним и т.д.).

# Метод решения

Требуется реализовать подсчёт значений четырёх математических функций: , , , . Для этого будем использовать разложения этих функций в ряды Тейлора:

* …
* …
* …
* … (для )

Для оптимизации вычислений используются рекуррентные формулы, которые позволяют вычислять каждый следующий член ряда на основе предыдущего:

Для :

Для :

Для :

Для :

Для вычисления суммы ряда реализованы три метода суммирования:

1. Прямое суммирование: члены ряда складываются последовательно от первого до последнего.

* Члены ряда складываются последовательно, начиная с первого.
* Алгоритм:
* Инициализировать переменную sum первым членом ряда.
* Для каждого следующего члена вычислить его значение и добавить к sum.
* Повторять до достижения заданного количества слагаемых.

1. Обратное суммирование: члены ряда складываются в обратном порядке (от последнего к первому).

* Члены ряда складываются в обратном порядке, начиная с последнего.
* Алгоритм:
* Вычислить все члены ряда и сохранить их в массив.
* Пройтись по массиву в обратном порядке, суммируя члены.

1. Попарное суммирование: члены ряда складываются попарно (первый с последним, второй с предпоследним и т.д.).

* Члены ряда складываются попарно: первый с последним, второй с предпоследним и т.д.
* Алгоритм:
* Вычислить все члены ряда и сохранить их в массив.
* Сложить элементы массива попарно, начиная с краёв.
* Если количество элементов нечётное, добавить средний элемент отдельно.

# Руководство пользователя

Описание работы программы:

Программа предназначена для вычисления значений математических функций , , , с использованием разложения этих функций в ряды Тейлора. Для каждой функции реализованы три метода суммирования: прямое, обратное и попарное. Программа также сравнивает результаты с точными значениями, полученными с использованием стандартных библиотечных функций, и вычисляет погрешность.

Действия пользователя:

Шаг 1: Запуск программы

1. Запустите программу. Появится консольное меню с предложением ввести значение для . (рис.1)

* Для функций , , можно вводить любое значение
* Для функции значение x должно сходиться в интервале , так как ряд Тейлора для сходится только при



Рис 1. Ввод значения x

Шаг 2: Выбор количества слагаемых в сумме

1. Пользователь должен ввести целое число, которое определяет количество слагаемых, используемых для вычисления суммы ряда. Чем больше количество слагаемых, тем выше точность вычислений, но тем дольше выполняется программа. (рис. 2)



Рис 2. Ввод количества слагаемых

Шаг 3: Вывод результатов.

1. После ввода данных программа вычислит значения функций , , , с использованием трёх методов суммирования (прямого, обратного и попарного). Для каждой функции будут выведены (рис.3):

* Значение, вычисленное с использованием прямого суммирования.
* Значение, вычисленное с использованием обратного суммирования.
* Значение, вычисленное с использованием попарного суммирования.
* Точное значение функции, полученное с использованием библиотеки math.h.
* Абсолютная погрешность для каждого метода (разница между вычисленным и точным значением).

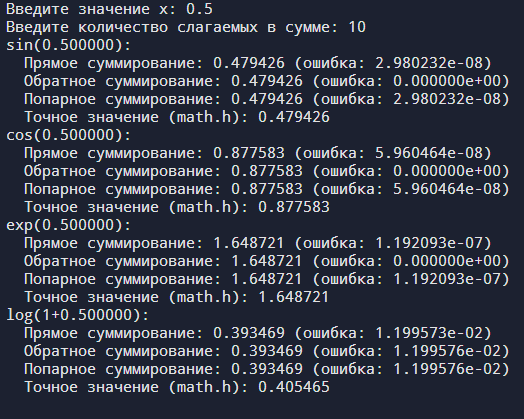


Рис 3. Вывод результатов суммирования рядов

Особые случаи:

Если пользователь вводит значение , для которого ряд Тейлора не сходится (например, для ), программа выведет сообщение об ошибке (рис.4):

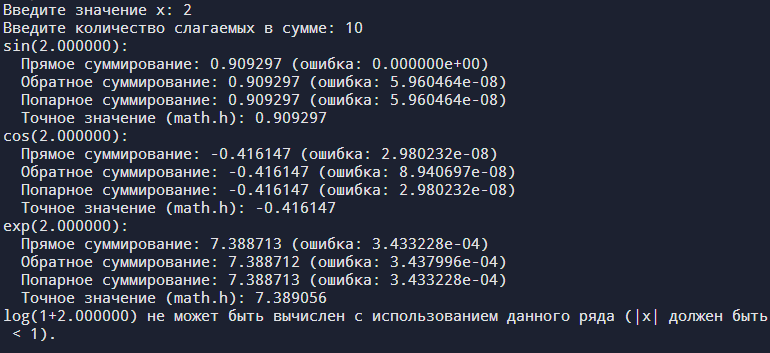


Рис 4. Ошибка для логарифма.

Рекомендации:

Для достижения высокой точности рекомендуется использовать большое количество слагаемых (например, 20–30).

Если значение близко к границам интервала сходимости ряда (например, для ), количество слагаемых следует увеличить.

Шаг 4. Завершение работы.

1. После вывода результатов программа завершает свою работу. Пользователь может запустить программу снова для вычисления значений функций при других входных данных.

# Описание программной реализации

Данная программа создана с использованием IDE Visual Studio и компилятор MSVC. Она состоит из одного текстового файла: “Laba2”.

Программа реализует вычисление значений математических функций , , , с использованием разложения этих функций в ряды Тейлора. Для каждой функции применяются три метода суммирования: прямое, обратное и попарное. Программа также сравнивает результаты с точными значениями, полученными с использованием стандартных библиотечных функций, и вычисляет погрешность.

Использованы следующие библиотеки:

* #include <stdio.h> - библиотека используется для ввода и вывода данных в консольном приложении;
* #include <math.h> - библиотека, содержащая множество математических функций для выполнения сложных расчётов.
* #include <stdlib.h> - библиотека используется для работы с динамической памятью;

1. Структура программы.

Программа состоит из следующих основных компонентов:

1) Функции для вычисления членов ряда:

next\_sin\_term, next\_cos\_term, next\_exp\_term, next\_log\_term - вычисляют следующий член ряда для соответствующих функций.

2) Функции суммирования:

* direct\_sum – прямое суммирование.
* reverse\_sum - обратное суммирование.
* pairwise\_sum - попарное суммирование.

3) Основная функция main

Организует ввод данных, вызов функций и вывод результатов.

2. Функции для вычисления членов ряда.

Каждая функция вычисляет следующий член ряда на основе предыдущего, используя рекуррентные формулы (они описаны в “Метод решения”)

3. Функции суммирования.

1)Прямое суммирование:

* Члены ряда складываются последовательно, начиная с первого.
* Реализация:

float direct\_sum(float x, float (\*next\_term)(float, int, float), float first\_term, int max\_terms) {

float sum = first\_term;

float term = first\_term;

for (int n = 1; n < max\_terms; n++) {

term = next\_term(x, n, term);

sum += term;

}

return sum;

}

2)Обратное суммирование:

* Члены ряда складываются в обратном порядке, начиная с последнего. Используется динамический массив.
* Реализация:

float reverse\_sum(float x, float (\*next\_term)(float, int, float), float first\_term, int max\_terms) {

// Выдение память для массива terms

float\* terms = (float\*)malloc(max\_terms \* sizeof(float));

if (terms == NULL) {

// Обработка ошибки выделения памяти

return 0.0;

}

terms[0] = first\_term;

for (int n = 1; n < max\_terms; n++) {

terms[n] = next\_term(x, n, terms[n - 1]);

}

float sum = 0.0;

for (int n = max\_terms - 1; n >= 0; n--) {

sum += terms[n];

}

// Освобождение выделенную память

free(terms);

return sum;

}

3)Попарное суммирование:

* Члены ряда складываются попарно (первый с последним, второй с предпоследним и т.д.). Используется динамический массив.
* Реализация:

float pairwise\_sum(float x, float (\*next\_term)(float, int, float), float first\_term, int max\_terms) {

float\* terms = (float\*)malloc(max\_terms \* sizeof(float));

if (terms == NULL) {

return 0.0;

}

// Заполняем массив terms

terms[0] = first\_term;

for (int n = 1; n < max\_terms; n++) {

terms[n] = next\_term(x, n, terms[n - 1]);

}

// Выполняем попарное суммирование

int size = max\_terms;

while (size > 1) {

int new\_size = (size + 1) / 2;

for (int i = 0; i < new\_size; i++) {

if (2 \* i + 1 < size) {

terms[i] = terms[2 \* i] + terms[2 \* i + 1];

}

else {

terms[i] = terms[2 \* i];

}

}

size = new\_size;

}

float result = terms[0];

free(terms);

return result;

}

float terms[max\_terms];

terms[0] = first\_term;

for (int n = 1; n < max\_terms; n++) {

terms[n] = next\_term(x, n, terms[n - 1]);

}

int size = max\_terms;

while (size > 1) {

int new\_size = (size + 1) / 2;

for (int i = 0; i < new\_size; i++) {

if (2 \* i + 1 < size) {

terms[i] = terms[2 \* i] + terms[2 \* i + 1];

}

else {

terms[i] = terms[2 \* i];

}

}

size = new\_size;

}

return terms[0];

}

4. Основная функция main

1) Ввод данных:

* Пользователь вводит значение и количество слагаемых max\_terms
* Для проверяется, что

2)Вычисление точных значений:

* Используются функции библиотеки math.h: sinf(x), cosf(x), expf(x), log1pf(x)

3) Вычисление значений функций с использованием рядов:

* Для каждой функции вызываются функции суммирования direct\_sum, reverse\_sum, pairwise\_sum

4) Вывод результатов:

* Для каждой функции выводятся значения, полученные разными методами, точные значения и погрешности.

5) Особенности реализации

* Используется тип данных float для экономии памяти и повышения производительности (в ущерб точности).
* Для вычисления абсолютной погрешности используется функция fabsf

Прототипы функций, используемых в программе:

float next\_sin\_term(float x, int n, float prev\_term)

float next\_cos\_term(float x, int n, float prev\_term)

float next\_exp\_term(float x, int n, float prev\_term)

float next\_log\_term(float x, int n, float prev\_term)

float direct\_sum(float x, float (\*next\_term)(float, int, float), float first\_term, int max\_terms)

float reverse\_sum(float x, float (\*next\_term)(float, int, float), float first\_term, int max\_terms)

float\* terms = (float\*)malloc(max\_terms \* sizeof(float))

float pairwise\_sum(float x, float (\*next\_term)(float, int, float), float first\_term, int max\_terms)

# Подтверждение корректности

Для подтверждения корректности реализации вычисления функций , , и с использованием рядов Тейлора и различных методов суммирования (прямого, обратного и попарного) в программе проведены следующие шаги:

1) Сравнение с точными значениями

Результаты, полученные с использованием реализованных методов, сравниваются с точными значениями функций, вычисленными с помощью стандартных библиотечных функций (например, sin(x), cos(x), exp(x), log1p(x) из библиотеки math.h). Это позволяет оценить точность каждого метода.

Пример сравнения для :

float exact\_sin = sinf(x);

float exact\_cos = cosf(x);

float exact\_exp = expf(x);

float exact\_log = log1pf(x); // log1pf(x) = log(1 + x)

float sin\_direct = direct\_sum(x, next\_sin\_term, x, max\_terms);

float sin\_reverse = reverse\_sum(x, next\_sin\_term, x, max\_terms);

float sin\_pairwise = pairwise\_sum(x, next\_sin\_term, x, max\_terms);

printf("sin(%f):\n", x);

printf(" Прямое суммирование: %f (ошибка: %e)\n", sin\_direct, fabsf(sin\_direct - exact\_sin));

printf(" Обратное суммирование: %f (ошибка: %e)\n", sin\_reverse, fabsf(sin\_reverse - exact\_sin));

printf(" Попарное суммирование: %f (ошибка: %e)\n", sin\_pairwise, fabsf(sin\_pairwise - exact\_sin));

printf(" Точное значение (math.h): %f\n", exact\_sin);

Аналогичные сравнения выполняются для , и .

2) Анализ погрешности

Погрешность вычисляется как абсолютная разница между значением, полученным с использованием ряда, и точным значением:

Для каждого метода суммирования (прямого, обратного, попарного) вычисляется погрешность и анализируется её зависимость от:

* Количества слагаемых в ряде.
* Значения

3) Тестирование на различных значенияx

Для проверки корректности реализации проведены тесты на различных значениях :

Для и :

Для :

Для : x = 0.1, 0.5, 0.9 (ряд сходится только при )

4) Примеры тестовых данных:

Пример 1 на рис.5:

, количество слагаемых .

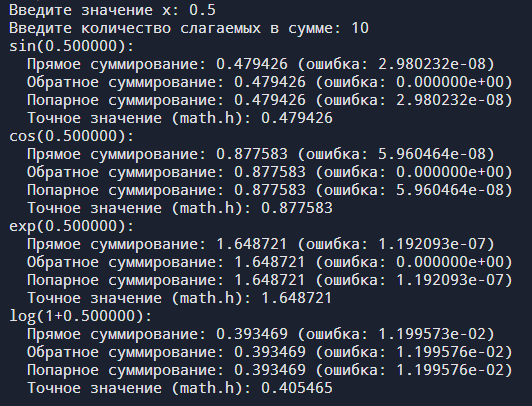


Рис. 5. Пример 1, когда x = 0.5, количество слагаемых = 10

Пример 2 на рис.6:

, количество слагаемых .

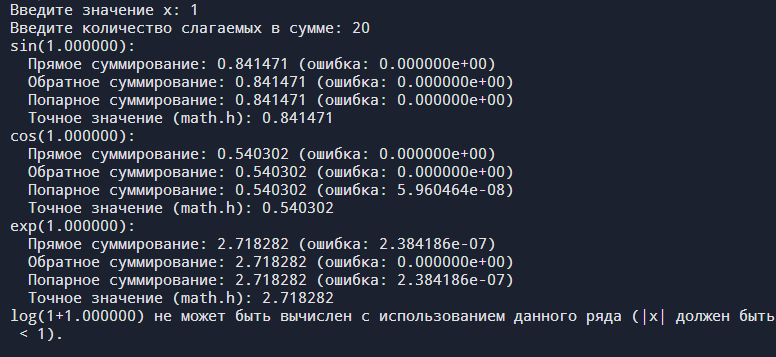


Рис. 6. Пример 2, когда x = 1, количество слагаемых = 20

Для повышения точности рекомендуется:

* Увеличивать количество слагаемых в ряде.
* Использовать обратное или попарное суммирование для уменьшения ошибок округления.

# Результаты экспериментов

Анализ результатов будет проходить следующим образом:

* Сравнить результаты, полученные разными методами, с точными значениями функций.
* Определить наилучшее число слагаемых в ряде
* Оценить погрешность для каждого метода:

* Взять фиксированное число слагаемых в суммировании ряда, изменяя значение , обращать на ошибку (погрешность значения суммирования полученного значения с значением из библиотеки math.h)

Так как ошибка (погрешность значения суммирования полученного значения с значением из библиотеки math.h) незначительна, будем анализировать погрешность вычисления суммирования ряда с значением . Для этого зададим фиксированное число слагаемых.

Чтобы определить, какое число слагаемых даёт наибольшую точность для прямого, обратного и попарного суммирования ряда, проведем эксперимент с разным значением слагаемых в ряде Тейлора для функций , , и . Для этого возьмём фиксированное значение 5. Например, возьмём такое число слагаемых: 5, 10, 20, 50, 100.

Начнём с количества слагаемых в ряде, равного 5 на рис. 7.

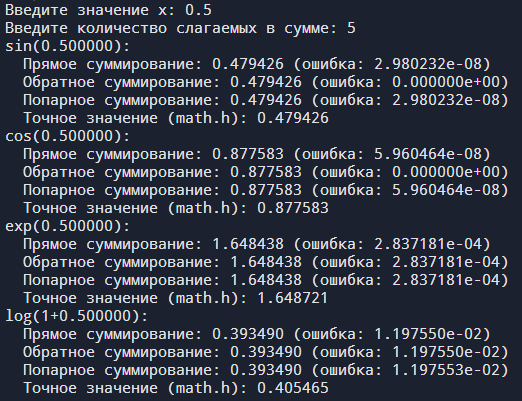


Рис. 7. Пример, когда x = 0.5, количество слагаемых = 5.

Дальше рассмотрим, количество слагаемых в ряде, равного 10 на рис. 8.

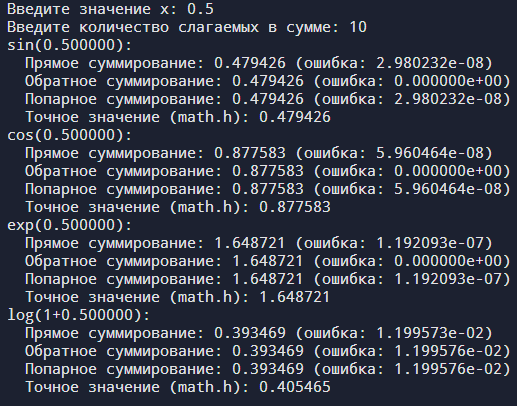


Рис. 8. Пример, когда x = 0.5, количество слагаемых = 10.

Затем - количество слагаемых в ряде, равного 20 на рис. 9.

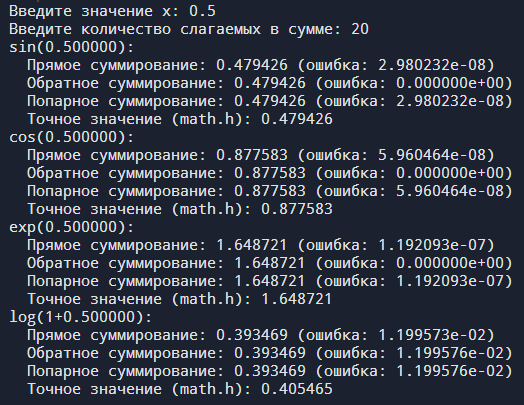


Рис. 9. Пример, когда x = 0.5, количество слагаемых = 20.

Потом количество слагаемых в ряде, равного 50 на рис. 10.

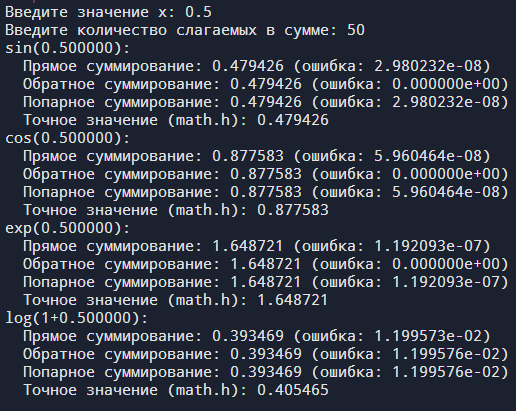


Рис. 10. Пример, когда x = 0.5, количество слагаемых = 50.

И наконец рассмотрим, когда количество слагаемых в ряде, равного 100 на рис. 11.

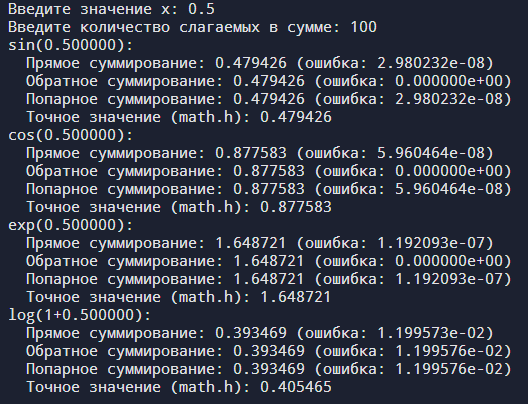


Рис. 11. Пример, когда x = 0.5, количество слагаемых = 100.

На основе предоставленного кода и данных экспериментов можно сделать следующие выводы:

1. Зависимость ошибки от количества слагаемых:

* При увеличении количества слагаемых max\_terms ошибка прямого суммирования может увеличиваться, в то время как обратное и попарное суммирование остаются более стабильными.
* Для функций, таких как sin(x), cos(x), exp(x), log1p(x), ошибка уменьшается с увеличением количества слагаемых, но только до определённого предела, после которого накопление ошибки округления может снова увеличить общую ошибку.

Таким образом, выбор метода суммирования может существенно повлиять на точность вычислений, особенно при работе с большим количеством слагаемых. Наиболее точное значение суммирования рядов будет выдавать большее число слагаемых в ряде из представленных, то есть 100.

Перейдём к основному эксперименту. Мы выяснили, что наибольшую точность, будет выдавать: 100. Сделаем это значение фиксированным. Теперь выберем значения для . Для ), и возьмём следующие значения: .

При вычислении , на отрезке от -20 до 20 для двухсот равномерно распределённых точек при схождении ста слагаемых (членов формулы Тейлора), средняя ошибка прямого суммирования - , обратного - , попарного - (рис. 12)

При , прямое суммирование даст большую точность. Средняя ошибка прямого суммирования - , обратного - , попарного - .



Рис. 12. Ошибка вычисления , при суммировании 100 слагаемых ряда Тейлора

Теперь рассмотрим , возьмём следующие значения: . При вычислении , на отрезке от -20 до 20 для двухсот равномерно распределённых точек при схождении ста слагаемых (членов формулы Тейлора), средняя ошибка прямого суммирования - , обратного - , попарного - (рис. 13)

При , прямое суммирование даст большую точность. Средняя ошибка прямого суммирования - , обратного - , попарного - .

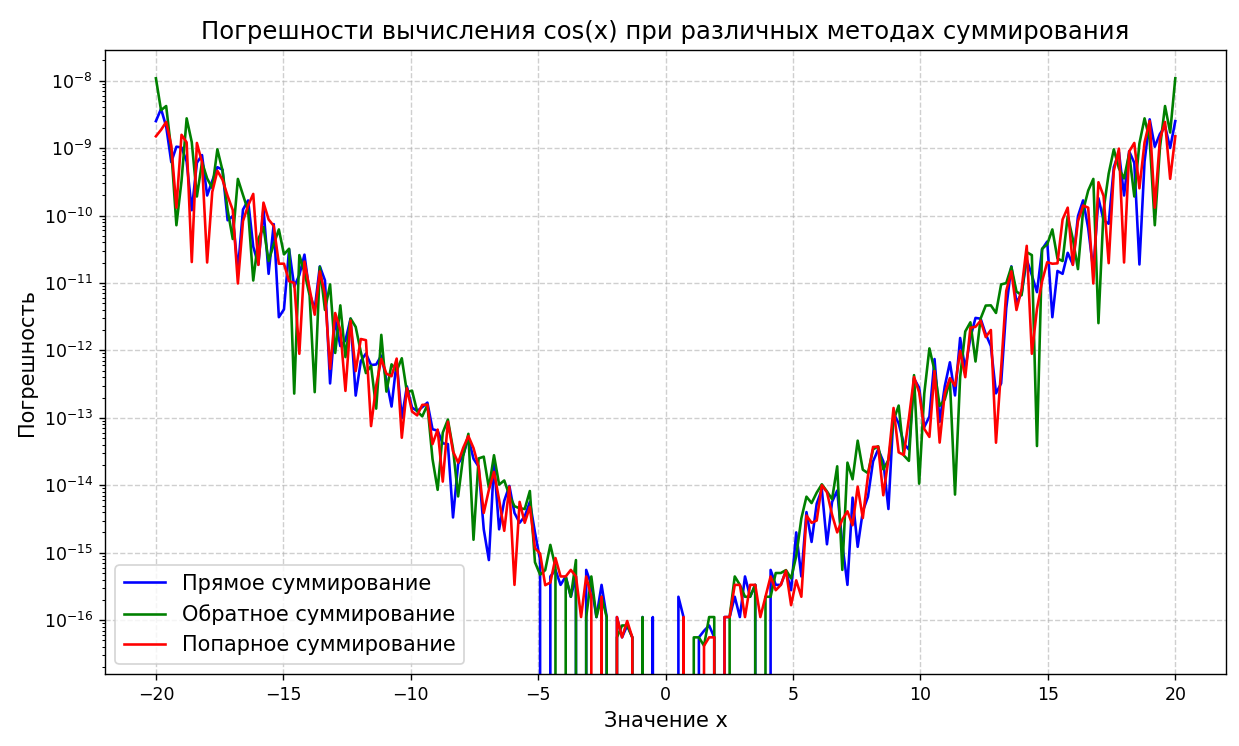


Рис. 13. Ошибка вычисления , при суммировании 100 слагаемых ряда Тейлора

Дальше будем рассматривать , возьмём следующие значения: . При вычислении , на отрезке от -20 до 20 для двухсот равномерно распределённых точек при схождении ста слагаемых (членов формулы Тейлора), средняя ошибка прямого суммирования - , обратного - , попарного - (рис. 14)

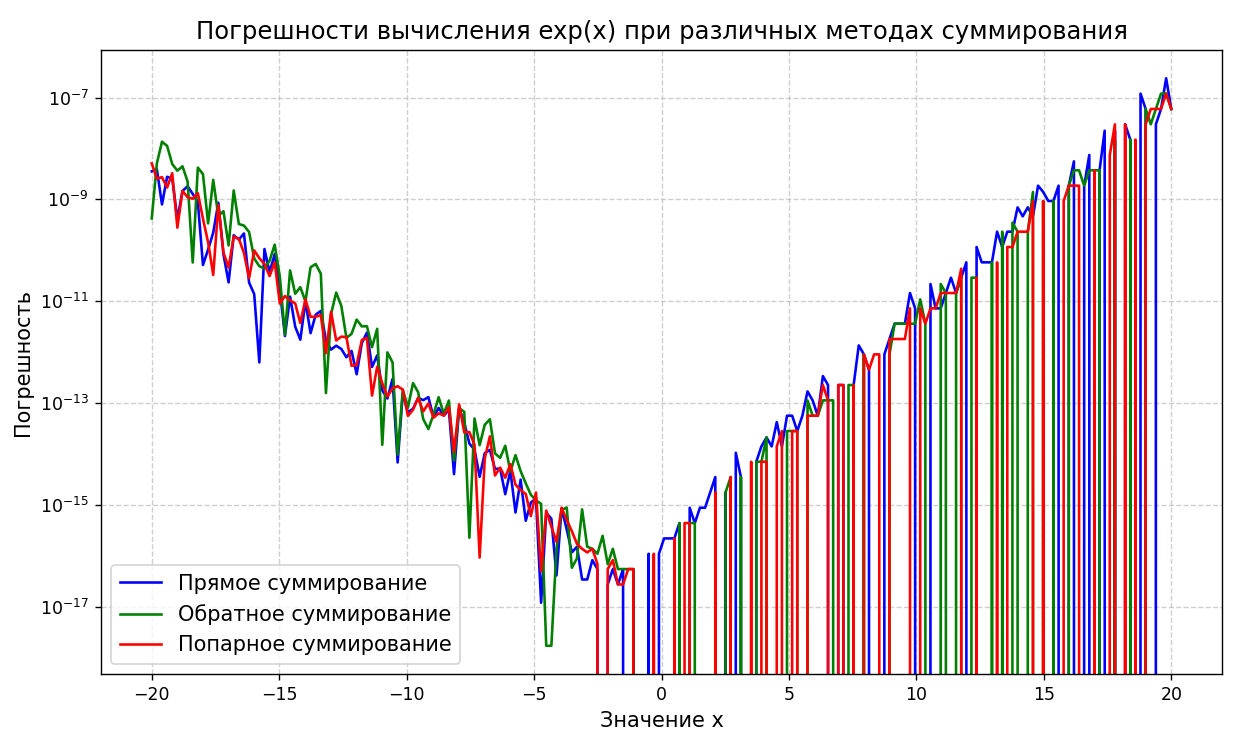


Рис. 14. Ошибка вычисления , при суммировании 100 слагаемых ряда Тейлора

И наконец рассмотрим . Теперь выберем значения для этой функции значения . Возьмём, например: -0.9, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 0.9. Для более точного результата возьмём фиксированное число слагаемых (членов ряда Тейлора), равное 1000.

При вычислении , на отрезке от -0.9 до 0.9 для двухсот равномерно распределённых точек при схождении ста слагаемых (членов формулы Тейлора), средняя ошибка прямого суммирования - , обратного -, попарного - (рис. 15)

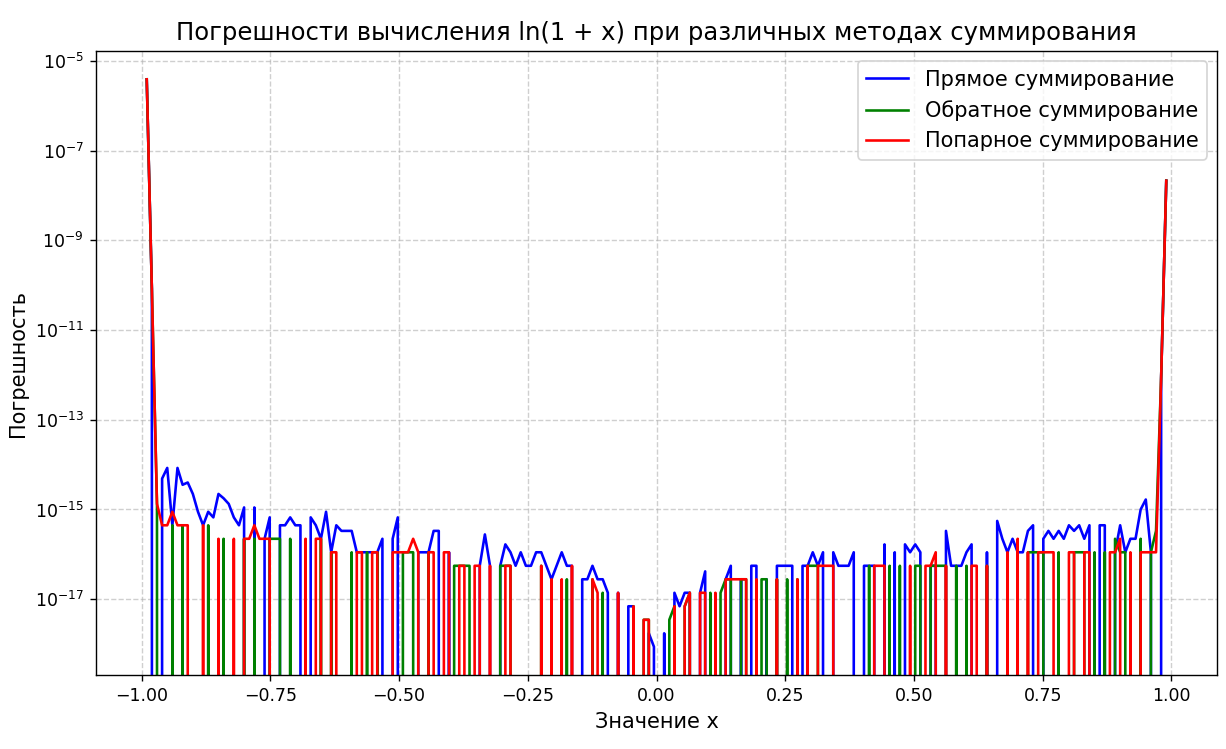


Рис. 15. Ошибка вычисления , при суммировании 1000 слагаемых ряда Тейлора

По результатам экспериментов сделаем следующие выводы:

1. Точность методов суммирования:

* Прямое суммирование накапливает ошибку из-за потери точности при сложении чисел с разными порядками. Это особенно заметно при большом количестве слагаемых. Ошибка увеличивается с увеличением при больших значениях .
* Обратное суммирование показывает лучшую точность, так как складывает числа от меньших к большим, что уменьшает потерю точности. Ошибка значительно меньше, чем при прямом суммировании.
* Попарное суммирование также демонстрирует хорошую точность, так как оно минимизирует потерю значимых разрядов при сложении. Ошибка сравнима с обратным суммированием, иногда даже меньше.

2. Сравнение функций:

1) ) и :

* + Ошибки для ) и ведут себя схожим образом.
  + При больших значениях ошибка прямого суммирования увеличивается, в то время как обратное и попарное суммирование остаются стабильными.

2):

* Для экспоненты ошибка прямого суммирования также увеличивается с ростом .
* Обратное и попарное суммирование показывают высокую точность даже при больших .

3):

* Ряд для сходится при ()
* Ошибка прямого суммирования увеличивается при приближении x к границам
* Обратное и попарное суммирование показывают стабильную точность на всём интервале.

3. Сравнение с точными значениями:

* Для всех функций (sin(x), cos(x), exp(x), log1p(x)) обратное и попарное суммирование обычно дают меньшую ошибку по сравнению с прямым суммированием.
* Ошибка для может быть значительной, если близок к границам интервала сходимости ряда ()

4. Наблюдения:

1) Обратное и попарное суммирование:

* Показывают значительно меньшую ошибку по сравнению с прямым суммированием.
* Рекомендуются для повышения точности вычислений, особенно при большом количестве слагаемых или при больших значениях .

2) Прямое суммирование:

* Просто в реализации, но менее точное.
* Подходит для задач, где высокая точность не требуется.

3) Зависимость от x:

* Для функций , , ошибка увеличивается с ростом
* Для ошибка увеличивается при приближении к границам интервала

5. Рекомендации:

* Для достижения максимальной точности рекомендуется использовать обратное или попарное суммирование.
* При работе с рядами, сходящимися только на определённом интервале (например, ), важно проверять входные данные на соответствие условиям сходимости.
* Для больших значений max\_terms следует учитывать накопление ошибок округления, особенно при прямом суммировании.

# Заключение

В ходе выполнения лабораторной работы была реализована программа для вычисления значений математических функций , , и с использованием разложения этих функций в ряды Тейлора. Для каждой функции были применены три метода суммирования: прямое, обратное и попарное. Программа также сравнивает результаты с точными значениями, полученными с использованием стандартных библиотечных функций, и вычисляет погрешность.

Программа демонстрирует эффективное использование рядов Тейлора для вычисления значений математических функций. Она позволяет сравнивать различные методы суммирования и оценивать их точность. Реализация на языке C обеспечивает высокую производительность и может быть адаптирована для работы с другими функциями и рядами.

Эксперименты показали, что **обратное** и **попарное** суммирование являются более точными методами по сравнению с прямым суммированием. Они обеспечивают стабильную точность даже при большом количестве слагаемых и больших значениях . Для задач, требующих высокой точности, рекомендуется использовать именно эти методы.

В заключение, проделанная работа продемонстрировала эффективность использования рядов Тейлора для вычисления математических функций и важность выбора метода суммирования для обеспечения высокой точности. Полученные результаты могут быть использованы для дальнейшего изучения численных методов и их применения в практических задачах.

# Приложение

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

float next\_sin\_term(float x, int n, float prev\_term) {

return -prev\_term \* x \* x / ((2 \* n) \* (2 \* n + 1));

}

float next\_cos\_term(float x, int n, float prev\_term) {

return -prev\_term \* x \* x / ((2 \* n - 1) \* (2 \* n));

}

float next\_exp\_term(float x, int n, float prev\_term) {

return prev\_term \* x / n;

}

float next\_log\_term(float x, int n, float prev\_term) {

return -prev\_term \* x / (n + 1);

}

// Прямое суммирование

float direct\_sum(float x, float (\*next\_term)(float, int, float), float first\_term, int max\_terms) {

float sum = first\_term;

float term = first\_term;

for (int n = 1; n < max\_terms; n++) {

term = next\_term(x, n, term);

sum += term;

}

return sum;

}

// Обратное суммирование

float reverse\_sum(float x, float (\*next\_term)(float, int, float), float first\_term, int max\_terms) {

// Выдение память для массива terms

float\* terms = (float\*)malloc(max\_terms \* sizeof(float));

if (terms == NULL) {

// Обработка ошибки выделения памяти

return 0.0;

}

terms[0] = first\_term;

for (int n = 1; n < max\_terms; n++) {

terms[n] = next\_term(x, n, terms[n - 1]);

}

float sum = 0.0;

for (int n = max\_terms - 1; n >= 0; n--) {

sum += terms[n];

}

// Освобождение выделенную память

free(terms);

return sum;

}

// Попарное суммирование

float pairwise\_sum(float x, float (\*next\_term)(float, int, float), float first\_term, int max\_terms) {

float\* terms = (float\*)malloc(max\_terms \* sizeof(float));

if (terms == NULL) {

return 0.0;

}

// Заполняем массив terms

terms[0] = first\_term;

for (int n = 1; n < max\_terms; n++) {

terms[n] = next\_term(x, n, terms[n - 1]);

}

// Выполняем попарное суммирование

int size = max\_terms;

while (size > 1) {

int new\_size = (size + 1) / 2;

for (int i = 0; i < new\_size; i++) {

if (2 \* i + 1 < size) {

terms[i] = terms[2 \* i] + terms[2 \* i + 1];

}

else {

terms[i] = terms[2 \* i];

}

}

size = new\_size;

}

float result = terms[0];

free(terms);

return result;

}

int main() {

float x;

int max\_terms;

printf("Введите значение x: ");

scanf("%f", &x);

printf("Введите количество слагаемых в сумме: ");

scanf("%d", &max\_terms);

// Точные значения из библиотеки math.h

float exact\_sin = sinf(x);

float exact\_cos = cosf(x);

float exact\_exp = expf(x);

float exact\_log = log1pf(x); // log1pf(x) = log(1 + x)

float sin\_direct = direct\_sum(x, next\_sin\_term, x, max\_terms);

float sin\_reverse = reverse\_sum(x, next\_sin\_term, x, max\_terms);

float sin\_pairwise = pairwise\_sum(x, next\_sin\_term, x, max\_terms);

printf("sin(%f):\n", x);

printf(" Прямое суммирование: %f (ошибка: %e)\n", sin\_direct, fabsf(sin\_direct - exact\_sin));

printf(" Обратное суммирование: %f (ошибка: %e)\n", sin\_reverse, fabsf(sin\_reverse - exact\_sin));

printf(" Попарное суммирование: %f (ошибка: %e)\n", sin\_pairwise, fabsf(sin\_pairwise - exact\_sin));

printf(" Точное значение (math.h): %f\n", exact\_sin);

float cos\_direct = direct\_sum(x, next\_cos\_term, 1.0, max\_terms);

float cos\_reverse = reverse\_sum(x, next\_cos\_term, 1.0, max\_terms);

float cos\_pairwise = pairwise\_sum(x, next\_cos\_term, 1.0, max\_terms);

printf("cos(%f):\n", x);

printf(" Прямое суммирование: %f (ошибка: %e)\n", cos\_direct, fabsf(cos\_direct - exact\_cos));

printf(" Обратное суммирование: %f (ошибка: %e)\n", cos\_reverse, fabsf(cos\_reverse - exact\_cos));

printf(" Попарное суммирование: %f (ошибка: %e)\n", cos\_pairwise, fabsf(cos\_pairwise - exact\_cos));

printf(" Точное значение (math.h): %f\n", exact\_cos);

float exp\_direct = direct\_sum(x, next\_exp\_term, 1.0, max\_terms);

float exp\_reverse = reverse\_sum(x, next\_exp\_term, 1.0, max\_terms);

float exp\_pairwise = pairwise\_sum(x, next\_exp\_term, 1.0, max\_terms);

printf("exp(%f):\n", x);

printf(" Прямое суммирование: %f (ошибка: %e)\n", exp\_direct, fabsf(exp\_direct - exact\_exp));

printf(" Обратное суммирование: %f (ошибка: %e)\n", exp\_reverse, fabsf(exp\_reverse - exact\_exp));

printf(" Попарное суммирование: %f (ошибка: %e)\n", exp\_pairwise, fabsf(exp\_pairwise - exact\_exp));

printf(" Точное значение (math.h): %f\n", exact\_exp);

if (x > -1 && x < 1) { // Ряд для log(1+x) сходится только при |x| < 1

float log\_direct = direct\_sum(x, next\_log\_term, x, max\_terms);

float log\_reverse = reverse\_sum(x, next\_log\_term, x, max\_terms);

float log\_pairwise = pairwise\_sum(x, next\_log\_term, x, max\_terms);

printf("log(1+%f):\n", x);

printf(" Прямое суммирование: %f (ошибка: %e)\n", log\_direct, fabsf(log\_direct - exact\_log));

printf(" Обратное суммирование: %f (ошибка: %e)\n", log\_reverse, fabsf(log\_reverse - exact\_log));

printf(" Попарное суммирование: %f (ошибка: %e)\n", log\_pairwise, fabsf(log\_pairwise - exact\_log));

printf(" Точное значение (math.h): %f\n", exact\_log);

}

else {

printf("log(1+%f) не может быть вычислен с использованием данного ряда (|x| должен быть < 1).\n", x);

}

return 0;

}