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**

**«Разложение функций в ряд Тейлора»**

**Выполнил**:

студент группы 3824Б1ПМ1

Ушаков С.Д.

**Проверила**:

Бусько П.В.

Нижний Новгород

2025

**Содержание**

[Введение 3](#_Toc193417085)

[Постановка задачи 4](#_Toc193417086)

[Руководство пользователя 5](#_Toc193417087)

[Описание программной реализации 8](#_Toc193417088)

[**Структура проекта** 8](#_Toc193417089)

[**Описание всех функций** 8](#_Toc193417090)

[**header.h 1. Использование** #pragma once 8](#_Toc193417091)

[**2. Подключение библиотек** 8](#_Toc193417092)

[**3. Определение структуры** Series 9](#_Toc193417093)

[**4. Прототипы функций для вычисления рядов** 9](#_Toc193417094)

[**5. Прототипы функций для вычисления в обратном порядке** 10](#_Toc193417095)

[**6. Вспомогательные функции** 10](#_Toc193417096)

[Результаты экспериментов 20](#_Toc193417097)

[Заключение 25](#_Toc193417098)

[Литература 26](#_Toc193417099)

[Приложение 27](#_Toc193417100)

# Введение

Ряды Тейлора играют ключевую роль в современной математике и её приложениях. Они позволяют аппроксимировать сложные функции с помощью бесконечных сумм степенных выражений, что особенно важно в областях, где точные аналитические решения недоступны или слишком сложны для вычисления. В эпоху цифровых технологий ряды Тейлора активно используются в компьютерных вычислениях, физике, инженерии, экономике и даже машинном обучении, где требуется высокая точность при ограниченных ресурсах.

Например, в компьютерной графике и моделировании ряды Тейлора применяются для вычисления тригонометрических функций, что позволяет создавать реалистичные визуализации. В финансовой аналитике они помогают оценивать сложные модели рисков. Кроме того, ряды Тейлора лежат в основе многих численных методов, таких как решение дифференциальных уравнений или оптимизация алгоритмов.

В данной работе рассматривается реализация вычисления функций

 с использованием разложения в ряд Тейлора. Уделяется внимание и на то, как выбор метода суммирования (обратный и прямой) будет влиять на результат.

# Постановка задачи

Цель данной работы: реализовать вычисление четырех функций  
( ( ) в рядах Тейлора, реализовать два метода суммирования (прямой и обратный) и сравнить эти два метода суммирования.

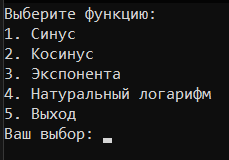
Для достижения поставленной цели были составлены соответствующие задачи:

1. Написать программу, которая реализует все перечисленное в цели работы, а так же которая будет взаимодействовать с пользователем с помощью интерфейса.
2. Составить графики полученных результатов с осями: погрешность, число слагаемых.
3. Сравнить методы суммирования и сделать вывод.

# Руководство пользователя

При запуске программы, перед пользователем открывается диалоговое окно. Будет предложено 4 различных функции для разложения и выход.

*Программа не перестанет работать, пока пользователь сам не напишет цифру 5 или не закроет диалоговое окно.*

  
*(рис. 1 – Меню программы)*

Если пользователь написал число от 1 до 4, то следующий шаг будет написание значение x.

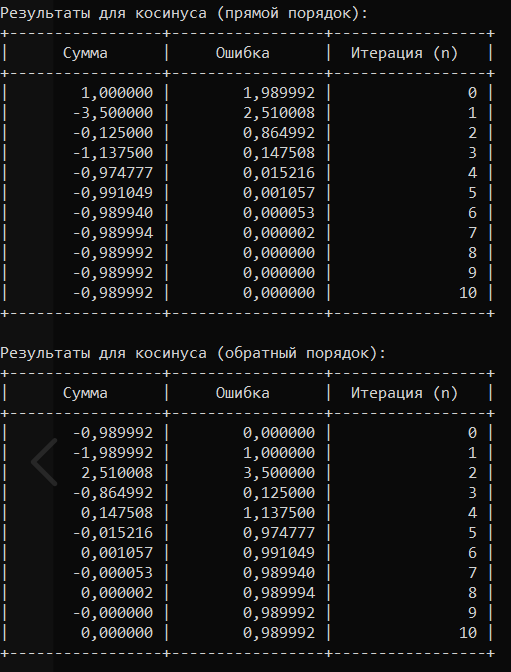
  
*(рис. 2 – Ввод X)*

Следующим шагом, после введения значения x, будет введение значения количества итераций.

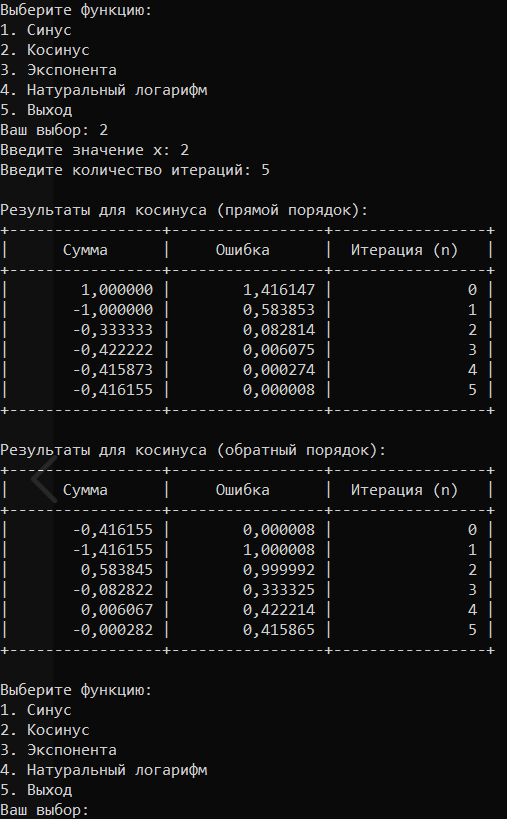
  
*(рис. 3 – Ввод кол-во итераций)*

После перед пользователем появятся две таблицы с суммированием двумя методами (прямым и обратным), а также ошибкой – погрешность между вычисленным значением и значением, находящимся в библиотеке math.h.

*Уточнение: Для прямого порядка следует смотреть результаты, начиная с 1 итерации, а также сверху вниз. Для обратного порядка с самого конца и до верху, то есть снизу в верх.*

  
*(рис. 4 – Таблицы результатов)*

Для более ясного понимания посмотрим на пример выполнения кода. Входные данные были: 2, 2, 5.

  
*(рис. 5 – Пример запуска программы)*

Как мы можем заметить: меню выбора функции остается, после вывода таблицы. Пользователь может дальше работать с программой, либо закрыть ее, получив нужные данные.

# Описание программной реализации

Программа представляет собой модульное приложение для вычисления четырёх функций (синус, косинус, экспонента, натуральный логарифм) с использованием разложения в ряды Тейлора. Входная точка программы находится в файле main.c, где пользователь выбирает одну из предложенных функций для вычисления. Программа предоставляет возможность выбора количества итераций (членов ряда) и выводит результаты в виде таблицы, включая частичные суммы и ошибки на каждом шаге.

## **Структура проекта**

Программа состоит из трёх основных файлов:

* header.h — заголовочный файл с объявлениями структур и функций.
* functions.c — файл с реализацией функций для вычисления рядов Тейлора.
* main.c — основной файл программы, содержащий точку входа main.

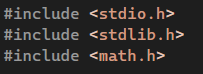
## **Описание всех функций**

## **header.h 1. Использование** #pragma once

  
*(рис. 6 – pragma once)*

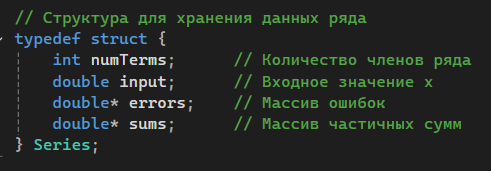
* #pragma once — это современная и удобная альтернатива #ifndef / #define.
* Она предотвращает повторное включение файла в проекте.
* Эта конструкция более лаконична и может быть быстрее обрабатываемой компилятором.

## **2. Подключение библиотек**

  
*(рис. 7 – Библиотеки)*

* Стандартные библиотеки для работы с вводом/выводом, выделением памяти и математическими операциями.

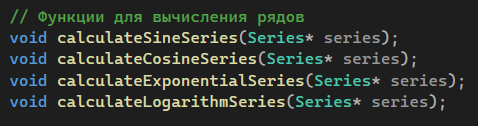
## **3. Определение структуры** Series

  
*(рис. 8 – структура Series)*

* typedef struct — создание пользовательского типа Series.
* **numTerms** — количество членов ряда, используемых при вычислении.
* **input** — входное значение x, для которого будет рассчитан ряд.
* **errors** — динамический массив ошибок, отражающих разницу между частичной суммой и точным значением.
* **sums** — массив частичных сумм ряда, необходимых для анализа сходимости.

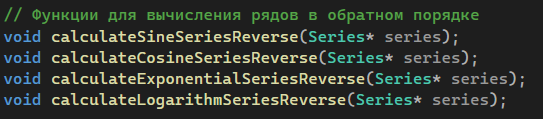
## 

## **4. Прототипы функций для вычисления рядов**

  
*(рис. 9 – Функции для вычисления рядов)*

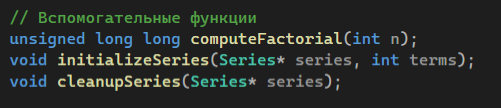
* Функции для вычисления значений синуса, косинуса, экспоненты и натурального логарифма с использованием разложения в ряд Тейлора.
* Принимают указатель на структуру Series, что позволяет передавать и изменять данные напрямую.

## **5. Прототипы функций для вычисления в обратном порядке**

  
*(рис. 10 – Функции, обратный порядок)*

* Эти функции выполняют те же вычисления, что и предыдущие, но в **обратном порядке** — от последнего члена ряда к первому.

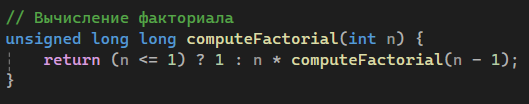
**6. Вспомогательные функции**

  
*(рис. 11 – Вспомогательные функции)*

* **computeFactorial** — функция для вычисления факториала n!n!n! с использованием unsigned long long для больших чисел.
* **initializeSeries** — функция для инициализации структуры Series, выделяющая память для массивов errors и sums.
* **cleanupSeries** — функция для освобождения памяти, чтобы предотвратить утечки.

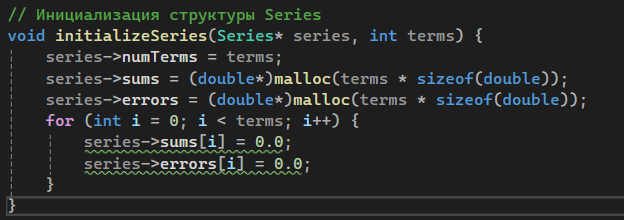
**functions.c**

1. unsigned long long computeFactorial(int n)

  
*(рис. 12 – Вычисление факториала)*

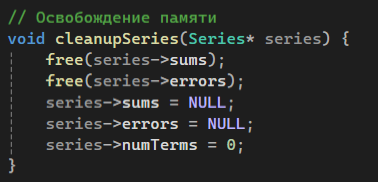
* Назначение: Вычисляет факториал числа nn.
* Принимает: Целое число nn.
* Возвращает: Факториал числа nn (тип unsigned long long).

2. void initializeSeries(Series\* series, int terms)

  
*(рис. 13 –Инициализация структуры)*

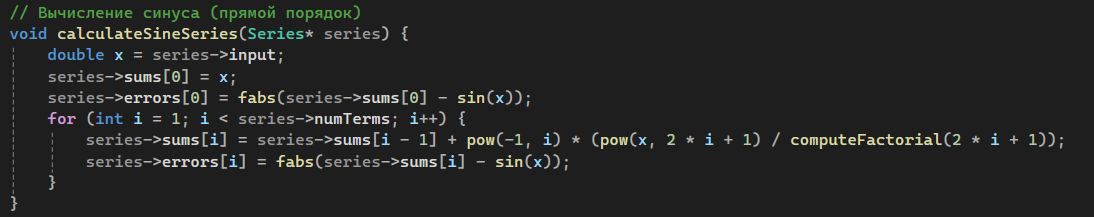
* Назначение: Инициализирует структуру Series, выделяя память для массивов sums и errors.
* Принимает:
* Указатель на структуру Series.
* Количество членов ряда termsterms.
* Действие:
* Выделяет память для массивов sums и errors.
* Инициализирует массивы нулями.

3. void cleanupSeries(Series\* series)

  
*(рис. 14 – Освобождение памяти)*

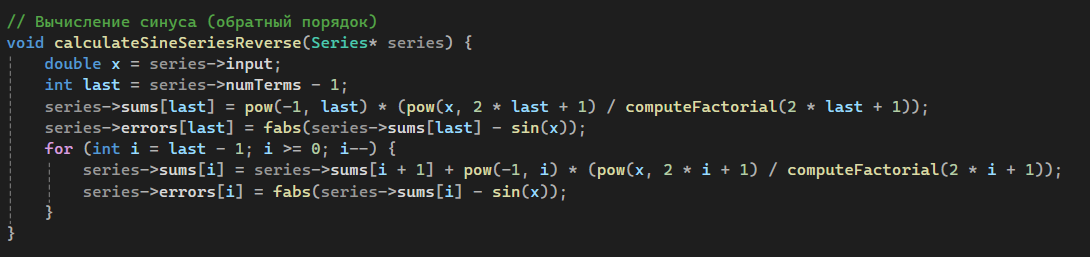
* Назначение: Освобождает память, выделенную для массивов sums и errors.
* Принимает: Указатель на структуру Series.
* Действие:
* Освобождает память.
* Устанавливает указатели на NULL и обнуляет numTerms.

4. void calculateSineSeries(Series\* series)

  
*(рис. 15 –Синус, прямой порядок)*

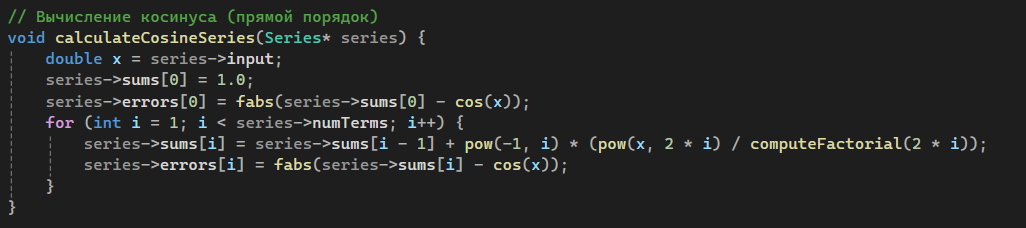
* Назначение: Вычисляет синус через ряд Тейлора в прямом порядке.
* Принимает: Указатель на структуру Series.
* Действие:
* Вычисляет частичные суммы и ошибки для синуса.
* Сохраняет результаты в массивы sums и errors.

5. void calculateSineSeriesReverse(Series\* series)

  
*(рис. 16 – Синус, обратный порядок)*

* Назначение: Вычисляет синус через ряд Тейлора в обратном порядке.
* Принимает: Указатель на структуру Series.
* Действие: Вычисляет частичные суммы и ошибки, начиная с последнего члена ряда.

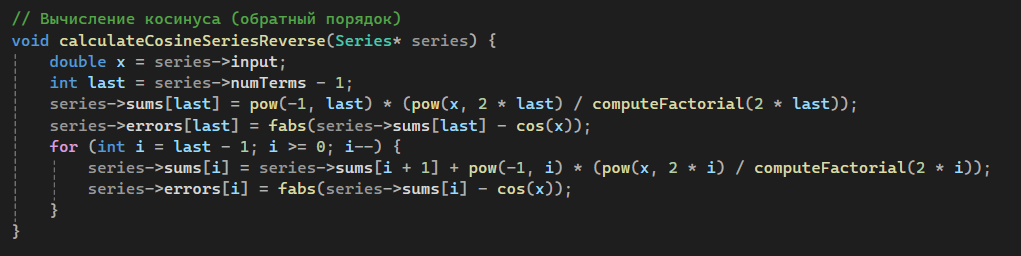
6. void calculateCosineSeries(Series\* series)



*(рис. 17 – Косинус, прямой порядок)*

* Назначение: Вычисляет косинус через ряд Тейлора в прямом порядке.
* Принимает: Указатель на структуру Series.
* Действие: Вычисляет частичные суммы и ошибки для косинуса.

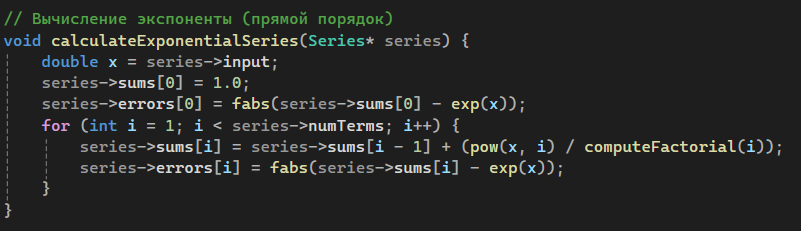
7. void calculateCosineSeriesReverse(Series\* series)



*(рис. 18 – Косинус, обратный порядок)*

* Назначение: Вычисляет косинус через ряд Тейлора в обратном порядке.
* Принимает: Указатель на структуру Series.
* Действие: Вычисляет частичные суммы и ошибки, начиная с последнего члена ряда.

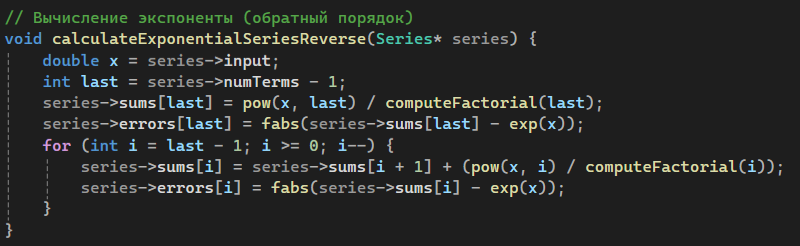
8. void calculateExponentialSeries(Series\* series)



*(рис. 19 –Экспонента, прямой порядок)*

* Назначение: Вычисляет экспоненту через ряд Тейлора в прямом порядке.
* Принимает: Указатель на структуру Series.
* Действие: Вычисляет частичные суммы и ошибки для экспоненты.

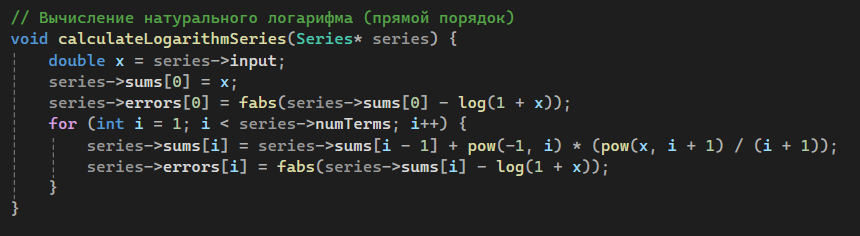
9. void calculateExponentialSeriesReverse(Series\* series)



*(рис. 20 –Экспонента, обратный порядок)*

* Назначение: Вычисляет экспоненту через ряд Тейлора в обратном порядке.
* Принимает: Указатель на структуру Series.
* Действие: Вычисляет частичные суммы и ошибки, начиная с последнего члена ряда.

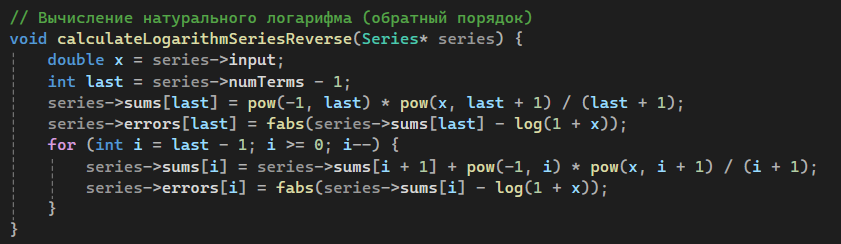
10. void calculateLogarithmSeries(Series\* series)



*(рис. 21 – Логарифм, прямой порядок)*

* Назначение: Вычисляет натуральный логарифм через ряд Тейлора в прямом порядке.
* Принимает: Указатель на структуру Series.
* Действие: Вычисляет частичные суммы и ошибки для натурального логарифма.

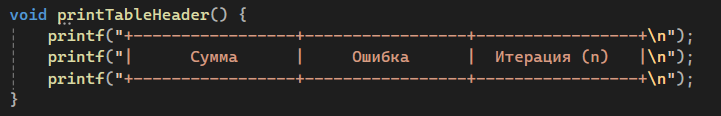
11. void calculateLogarithmSeriesReverse(Series\* series)



*(рис. 22 – Логарифм, обратный порядок)*

* Назначение: Вычисляет натуральный логарифм через ряд Тейлора в обратном порядке.
* Принимает: Указатель на структуру Series.
* Действие: Вычисляет частичные суммы и ошибки, начиная с последнего члена ряда.

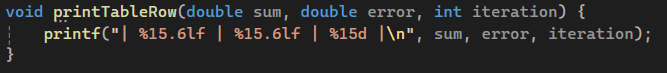
12. void printTableHeader()



*(рис. 23 – Начало таблицы)*

* Назначение: Выводит заголовок таблицы.
* Действие: Печатает шапку таблицы с колонками "Сумма", "Ошибка" и "Итерация (n)".

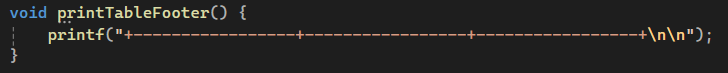
13. void printTableRow(double sum, double error, int iteration)



*(рис. 24 – Вывод строки)*

* Назначение: Выводит строку таблицы.
* Принимает:
* sum: Частичная сумма.
* error: Ошибка.
* iteration: Номер итерации.
* Действие: Печатает строку с данными.

14. void printTableFooter()



*(рис. 25 – Конец таблицы)*

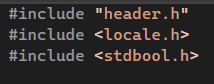
* Назначение: Выводит нижнюю границу таблицы.
* Действие: Печатает разделительную линию.

15. void runProgram()  
*см. Приложение 1.*

* Назначение: Основная функция программы, реализующая взаимодействие с пользователем.
* Действие:
* Выводит меню выбора функции.
* Запрашивает у пользователя значение xx и количество итераций.
* Вызывает соответствующую функцию для вычисления ряда.
* Выводит результаты в виде таблицы.
* Рекурсивно вызывает себя для повторного выбора функции.

**Main.c**

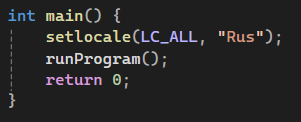
1. Подключение заголовочных файлов



*(рис. 26 – main.c, библиотеки)*

* #include "header.h": Подключает заголовочный файл header.h, который содержит объявления структур и функций, необходимых для работы программы.
* #include <locale.h>: Подключает библиотеку для работы с локалями (настройка языка и региональных стандартов).
* #include <stdbool.h>: Подключает библиотеку для использования типа bool (логические значения true и false).

2. Функция main



*(рис. 25 – функция main)*

* Назначение: Основная функция программы, которая запускает выполнение.
* Действие:
* setlocale(LC\_ALL, "Rus");  
  Устанавливает локаль на русский язык, чтобы программа могла корректно работать с кириллицей (например, выводить русский текст в консоль).
* runProgram();  
  Вызывает функцию runProgram, которая реализует основную логику программы (меню выбора функции, ввод данных, вычисление и вывод результатов).
* return 0;  
  Завершает выполнение программы с кодом возврата 0, что означает успешное завершение.
* Роль main.c в проекте
* Точка входа:  
  Функция main является стартовой точкой программы. Она запускает выполнение всех остальных функций.
* Настройка локали:  
  Устанавливает локаль для поддержки кириллицы, что важно для корректного отображения текста на русском языке.
* Вызов основной логики:  
  Функция runProgram (реализованная в functions.c) содержит всю основную логику программы, включая меню, ввод данных и вывод результатов.

# Результаты экспериментов

Для проведения эксперимента рассмотрим погрешность между нашей и библиотечной реализациями значений функций в конкретной точке. Для функций возьмем . Рассмотрим все предложенные способы суммирования с разным числом слагаемых.

Все данные представлены в таблицах и на графиках. Ось абсцисс отвечает за число слагаемых, а ось ординат – за погрешность.

**1. Прямое суммирование**

**1.1 Синус**

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0,242631 |
| 2 | 0,024036 |
| 3 | 0,001361 |
| 4 | 0,000050 |
| 5 | 0,000001 |
| 6 | 0,000000 |
| 7 | 0,000000 |
| 8 | 0,000000 |

*Таблица 1 График 1*

Как мы видим, начиная с 6 слагаемого, погрешность стремиться к 0 и становится очень точной.

**1.2 Косинус**

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0,583853 |
| 2 | 0,082814 |
| 3 | 0,006075 |
| 4 | 0,000274 |
| 5 | 0,000008 |
| 6 | 0,000000 |
| 7 | 0,000000 |
| 8 | 0,000000 |

*Таблица 2 График 2*

График косинуса схож с графиком синуса, только погрешность на каждом шагу чуть-чуть побольше.

**1.3 Экспонента**

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 4,389056 |
| 2 | 2,389056 |
| 3 | 1,055723 |
| 4 | 0,389056 |
| 5 | 0,122389 |
| 6 | 0,033501 |
| 7 | 0,008104 |
| 8 | 0,001755 |

*Таблица 3 График 3*

У экспоненты погрешность на каждом шагу больше, чем у тригонометрических функций. С каждым новым шагом ее погрешность уменьшается, но не настолько быстро, как мы видели у синуса и косинуса.

**1.4 Логарифм**

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0,075628 |
| 2 | 0,038705 |
| 3 | 0,021320 |
| 4 | 0,012294 |
| 5 | 0,007314 |
| 6 | 0,004451 |
| 7 | 0,002755 |
| 8 | 0,001729 |

*Таблица 4 График 4*

График логарифма очень сильно прижат к оси “Погрешности”, но у тригонометрических функций точность выше. Логарифм же намного точен при вычислении первых слагаемых.

**2. Обратное суммирование**

**2.1 Синус**

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0,909297 |
| 2 | 0,909296 |
| 3 | 0,909347 |
| 4 | 0,907937 |
| 5 | 0,933333 |
| 6 | 0,666667 |
| 7 | 2,000000 |
| 8 | 0,000000 |

*Таблица 5 График 5*

Погрешность больше, чем в прямом, но все также маленькая, а начиная с 8 уже равна 0

**2.2 Косинус**

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0,416147 |
| 2 | 0,416155 |
| 3 | 0,415873 |
| 4 | 0,422222 |
| 5 | 0,333333 |
| 6 | 1,000000 |
| 7 | 1,000000 |
| 8 | 0,000000 |

*Таблица 6 График 6*

Все также точность быстро достигает абсолютного значения.

**2.3 Экспонента**

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 7,357310 |
| 2 | 7,268421 |
| 3 | 7,001755 |
| 4 | 6,335088 |
| 5 | 5,001755 |
| 6 | 3,001755 |
| 7 | 1,001755 |
| 8 | 0,001755 |

*Таблица 7 График 7*

Точность растет с увеличением слагаемых.

**2.4 Логарифм**

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0,533351 |
| 2 | 0,521586 |
| 3 | 0,541194 |
| 4 | 0,507580 |
| 5 | 0,567605 |
| 6 | 0,453271 |
| 7 | 0,698271 |
| 8 | 0,001729 |

*Таблица 8 График 8*

Логарифм показывает погрешность, которая ближе всех других функций к истинному значению, но только в начале суммы.

# Заключение

Цели, поставленные в начале работы, были выполнены: было реализовано вычисление четырех функций в рядах Тейлора, было реализовано два метода суммирования.

Анализируя два метода – можно прийти к такому выводу:

* Прямое суммирование лучше подходит для рядов с быстро убывающими членами.
* Обратное суммирование предпочтительно для рядов с медленно убывающими членами или при необходимости высокой точности.

# Литература

1. Керниган Б., Ритчи Д., Фьюэр А. Язык программирования СИ //М.: Финансы и статистика. – 1992.
2. Кнут Д. Э. Искусство программирования: Сортировка и поиск. – Издательский дом Вильямс, 2000. – Т. 3.

# Приложение

**Приложение 1.**

void runProgram() {

Series data; // Используем новую структуру Series

int option;

double value;

int iterations;

printf("Выберите функцию:\n");

printf("1. Синус\n");

printf("2. Косинус\n");

printf("3. Экспонента\n");

printf("4. Натуральный логарифм\n");

printf("5. Выход\n");

printf("Ваш выбор: ");

scanf\_s("%d", &option);

if (option == 5) {

printf("Завершение работы программы.\n");

return;

}

if (option < 1 || option > 5) {

printf("Ошибка: выбран неверный вариант. Попробуйте снова.\n\n");

runProgram();

return;

}

printf("Введите значение x: ");

scanf\_s("%lf", &value);

if (option == 4 && fabs(value) >= 1) {

printf("Ошибка: для логарифма |x| должен быть меньше 1.\n\n");

runProgram();

return;

}

printf("Введите количество итераций: ");

scanf\_s("%d", &iterations);

if (iterations < 1) {

printf("Ошибка: количество итераций должно быть положительным числом.\n\n");

runProgram();

return;

}

data.input = value; // Используем новое поле input

initializeSeries(&data, iterations + 1); // Используем новое название функции

switch (option) {

case 1:

printf("\nРезультаты для синуса (прямой порядок):\n");

printTableHeader();

calculateSineSeries(&data); // Используем новое название функции

for (int i = 0; i < data.numTerms; i++) {

printTableRow(data.sums[i], data.errors[i], i);

}

printTableFooter();

printf("Результаты для синуса (обратный порядок):\n");

printTableHeader();

calculateSineSeriesReverse(&data); // Используем новое название функции

for (int i = 0; i < data.numTerms; i++) {

printTableRow(data.sums[i], data.errors[i], i);

}

printTableFooter();

break;

case 2:

printf("\nРезультаты для косинуса (прямой порядок):\n");

printTableHeader();

calculateCosineSeries(&data); // Используем новое название функции

for (int i = 0; i < data.numTerms; i++) {

printTableRow(data.sums[i], data.errors[i], i);

}

printTableFooter();

printf("Результаты для косинуса (обратный порядок):\n");

printTableHeader();

calculateCosineSeriesReverse(&data); // Используем новое название функции

for (int i = 0; i < data.numTerms; i++) {

printTableRow(data.sums[i], data.errors[i], i);

}

printTableFooter();

break;

case 3:

printf("\nРезультаты для экспоненты (прямой порядок):\n");

printTableHeader();

calculateExponentialSeries(&data); // Используем новое название функции

for (int i = 0; i < data.numTerms; i++) {

printTableRow(data.sums[i], data.errors[i], i);

}

printTableFooter();

printf("Результаты для экспоненты (обратный порядок):\n");

printTableHeader();

calculateExponentialSeriesReverse(&data); // Используем новое название функции

for (int i = 0; i < data.numTerms; i++) {

printTableRow(data.sums[i], data.errors[i], i);

}

printTableFooter();

break;

case 4:

printf("\nРезультаты для натурального логарифма (прямой порядок):\n");

printTableHeader();

calculateLogarithmSeries(&data); // Используем новое название функции

for (int i = 0; i < data.numTerms; i++) {

printTableRow(data.sums[i], data.errors[i], i);

}

printTableFooter();

printf("Результаты для натурального логарифма (обратный порядок):\n");

printTableHeader();

calculateLogarithmSeriesReverse(&data); // Используем новое название функции

for (int i = 0; i < data.numTerms; i++) {

printTableRow(data.sums[i], data.errors[i], i);

}

printTableFooter();

break;

default:

break;

}

cleanupSeries(&data); // Используем новое название функции

// Рекурсивный вызов для повторного выбора операции

runProgram();

}