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 3824Б1-ПМ1

Ерин.Е.С.

**Проверила**:

Бусько П.В.

Нижний Новгород

2025

**Содержание**

[Введение 3](#_Toc193300921)

[Постановка задачи 4](#_Toc193300922)

[Описание программной реализации 5](#_Toc193300923)

[Результаты экспериментов 7](#_Toc193300924)

[Заключение 12](#_Toc193300925)

[Литература 13](#_Toc193300926)

[Приложение 14](#_Toc193300927)

# Введение

Важной задачей вычислительной математики является расчет математических функций с определенной точностью. Среди популярных подходов аппроксимации выделяются разложения в ряды Тейлора и Маклорена. Эти методы дают возможность разложить сложные функции на бесконечные ряды элементарных слагаемых, что существенно облегчает их численное вычисление.

Цель исследования — анализ влияния количества членов ряда на погрешность расчетов для базовых математических функций: тригонометрических (синус, косинус), экспоненты и натурального логарифма. Практическая составляющая работы включает разработку алгоритмов на языке программирования C и сопоставление полученных результатов со встроенными математическими функциями стандартной библиотеки.

# Постановка задачи

Основная задача —исследование на практике точность вычисления функций с использованием рядов Тейлора. Этапы работы:

1. Реализация структуры ряда Тейлора и функций через прямое и обратное суммирование с помощью рядов Тейлора
2. Сравнение полученных значений функций со значениями библиотечных функций в math.h.

# Описание программной реализации

Проект состоит из одного файла “Lab2.cpp”:

**Подключаемые библиотеки**:

1. **Stdio.h** – стандартная библиотека С для ввода и вывода информации.
2. **Stdlib.h** – библиотека для работы с памятью.
3. **Math.h** – библиотека со стандартными функциями.

**Структуры**:

struct Xn – Структура последовательности сумм ряда Тейлора (*см. приложение* *1*). Состоит из:

* Количество элементов ряда (int)
* Массив сумм для n элементов(double)
* Массив погрешностей для n элементов(double)

**Функции:**

* fact (*см. приложение* 2) — вычисление факториала рекурсией для натуральных чисел.
* Msin, Mcos, Mexp, Mln (*см. приложение* *3*)  — реализация прямого суммирования ряда для стандартных функций по Маклорену. Принимают в себя структуру последовательности сумм ряда Тейлора, значение для которого просчитывается значение, эталонное значение функции.
* MsinR, McosR, MexpR, MlnR (*см. приложение* *4*) — альтернатива функциям Msin, Mcos, Mexp, Mln, где реализована обратная сумма.
* Main (*см. приложение* 5) — функция, которая создает и заполняет структуру данными. Принимает в себя функцию, которая вычисляет суммы членов последовательности, эталонную функцию из библиотеки “math.h”, точку в которой будет вычисляться ряд, начальное значение последовательности, количество членов последовательности, булевую переменную определяющую способ подсчёта.
* print (*см. приложение* 6) — функция выводящая в консоль значения последовательности. Принимает в себя структуру последовательности сумм ряда Тейлора.
* printR (*см. приложение* *7*) — функция выводящая в обратном порядке в консоль значения последовательности. Принимает в себя структуру последовательности сумм ряда Тейлора.
* Free (*см. приложение* 8) — освобождает память выделенную для структуры последовательности сумм ряда Тейлора. Принимает в себя структуру последовательности сумм ряда Тейлора.
* init (*см. приложение* 9) — выделяет память для массивов и инициализирует структуру. Принимает в себя ссылку на структуру последовательности сумм ряда Тейлора, эталонное значение, первый элемент, размер массивов.
* initR (*см. приложение* 10) — выделяет память для массивов и инициализирует структуру. Принимает в себя ссылку на структуру последовательности сумм ряда Тейлора, эталонное значение, последний элемент, размер массивов.

# Результаты экспериментов

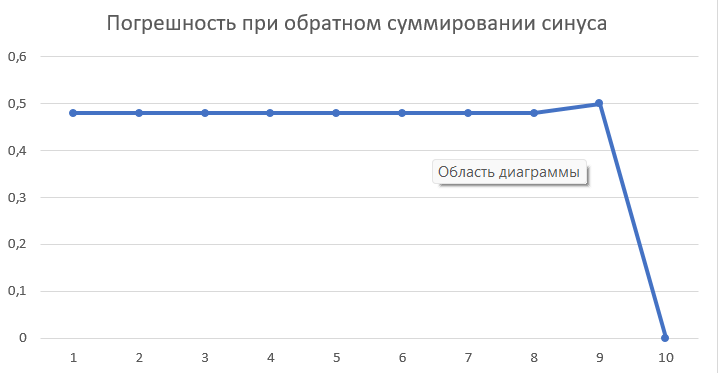
Вычисления проведены для x = 0,5. Все необходимые данные представлены в таблицах и на графиках. Ось абсцисс отвечает за число слагаемых, а ось ординат – за погрешность.

1. **Синус:**



(Рис.1 – график погрешности при обратном суммировании синуса)

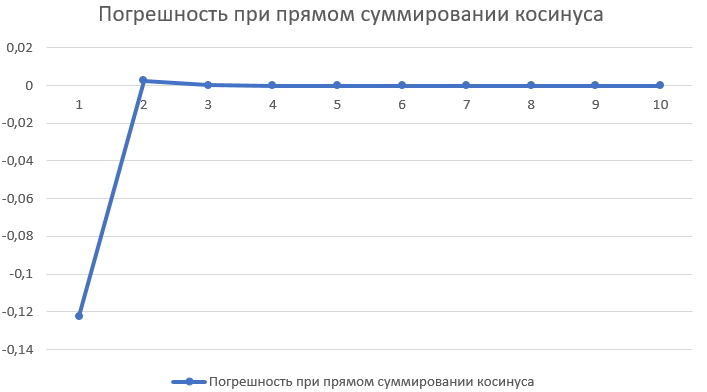
На рис.3 представлен график зависимости погрешности от числа слагаемых для функции . Видно, что уже при 3 слагаемых точность очень высока и практически не отличается от “эталонной”.



(Рис.2 – график погрешности при обратном суммировании синуса)

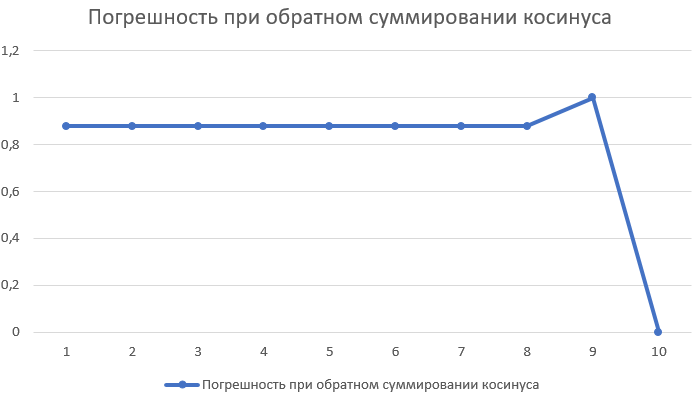
На рис. 2 видно, что при 10 слагаемых значение ряда абсолютно не отличается от истинного значения.

1. **Косинус:**



(Рис.3 – график погрешности при прямом суммировании косинуса)

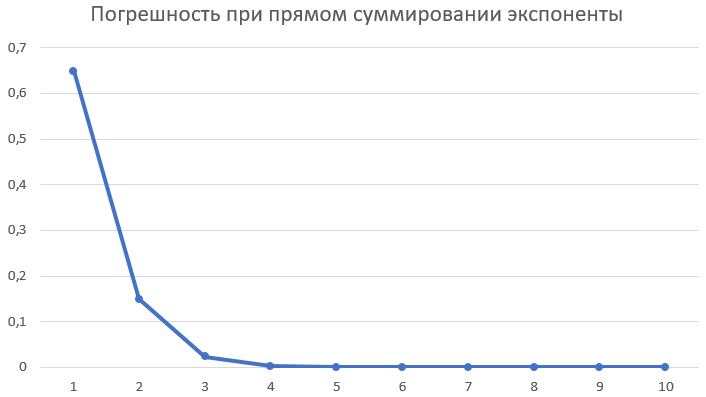
Также, как и с синусом, ряд с косинусом очень быстро сходится.



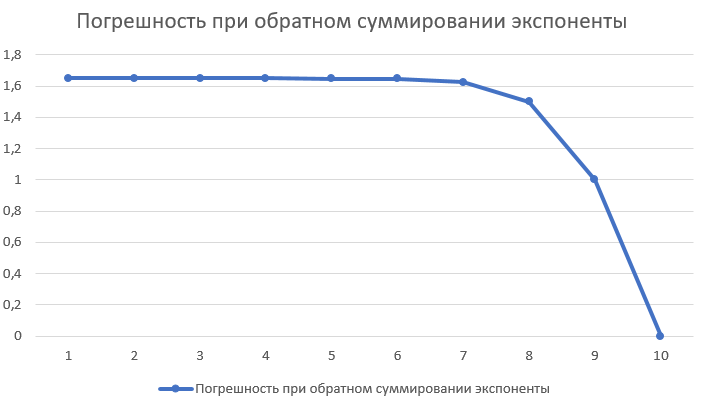
(Рис.4 – график погрешности при обратном суммировании косинуса)

Ряд с 10 слагаемыми недалек от истины.

1. **Экспонента:**

 (Рис.5 – график погрешности при прямом суммировании экспоненты)

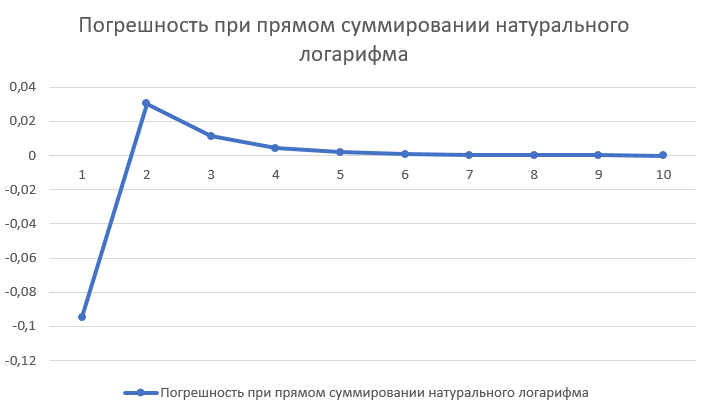
На рис. 5 видно, что кривая очень плавно опускается и прижимается к оси абсцисс, по сравнению с двумя другими функциями, которые мы уже рассмотрели



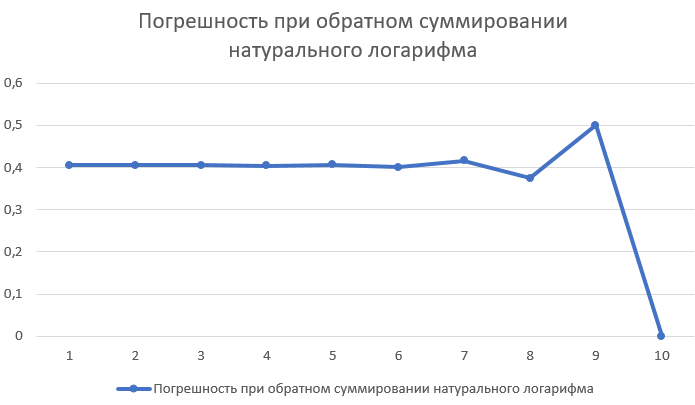
(Рис.5 – график погрешности при обратном суммировании экспоненты)

Видно, что тенденция плавного уменьшения сохраняется.

1. **Логарифм:**



(Рис.7 – график погрешности при прямом суммировании натурального логарифма)

График для логарифма также плавно опускается 

(Рис.8 – график погрешности при обратном суммировании натурального логарифма)

# Заключение

В рамках исследования были разработаны алгоритмы расчета функций через разложение в числовые ряды. Вычислительные эксперименты продемонстрировали обратную зависимость между количеством учитываемых членов ряда и величиной погрешности. Метод прямого суммирования (от младших к старшим степеням) показал повышенную точность на начальных этапах итерационного процесса. При использовании обратного порядка сложения выявлена особенность: при числе итераций свыше 5 начальные элементы ряда практически не оказывают влияния на конечный результат.

# Литература

1. Кнут Д. Э. Искусство программирования. Том 2.
2. Бахвалов Н.С. Численные методы.
3. Документация стандартной библиотеки C (math.h).

# Приложение

**Приложение №1**

struct Xn

{

int n;

double\* err;

double\* sum;

};

**Приложение №2**

double fact(int num) {

if (num == 0 || num == 1) return 1;

else return num \* fact(num - 1);

}

**Приложение №3**

void Msin(Xn\* xn, double x, double ideal)

{

for (int i = 1; i < xn->n; i++)

{

xn->sum[i] = xn->sum[i - 1] + pow(-1, i) \* (pow(x, 2 \* i + 1) / fact(2 \* i + 1));

xn->err[i] = fabs(xn->sum[i] - ideal);

}

}

void Mcos(Xn\* xn, double x, double ideal)

{

for (int i = 1; i < xn->n; i++)

{

xn->sum[i] = xn->sum[i - 1] + pow(-1, i) \* (pow(x, 2 \* i) / fact(2 \* i));

xn->err[i] = fabs(xn->sum[i] - ideal);

}

}

void Mexp(Xn\* xn, double x, double ideal)

{

for (int i = 1; i < xn->n; i++)

{

xn->sum[i] = xn->sum[i - 1] + (pow(x, i) / fact(i));

xn->err[i] = fabs(xn->sum[i] - ideal);

}

}

void Mln(Xn\* xn , double x, double ideal)

{

x = x - 1;

for (int i = 1; i < xn->n; i++)

{

xn->sum[i] = xn->sum[i - 1] + pow(-1, i) \* (pow(x, i + 1) / (i + 1));

xn->err[i] = fabs(xn->sum[i] - ideal);

}

}

**Приложение №4**

void MsinR(Xn\* xn, double x, double ideal)

{

for (int i = xn->n - 1; i >= 0; i--)

{

xn->sum[i] = xn->sum[i + 1] + pow(-1, i) \* (pow(x, 2 \* i + 1) / fact(2 \* i + 1));

xn->err[i] = fabs(xn->sum[i] - ideal);

}

}

void McosR(Xn\* xn, double x, double ideal)

{

for (int i = xn->n - 1; i >= 0; i--)

{

xn->sum[i] = xn->sum[i + 1] + pow(-1, i) \* (pow(x, 2 \* i) / fact(2 \* i));

xn->err[i] = fabs(xn->sum[i] - ideal);

}

}

void MexpR(Xn\* xn, double x, double ideal)

{

for (int i = xn->n - 1; i >= 0; i--)

{

xn->sum[i] = xn->sum[i + 1] + (pow(x, i) / fact(i));

xn->err[i] = fabs(xn->sum[i] - ideal);

}

}

void MlnR(Xn\* xn , double x, double ideal)

{

x = x - 1;

for (int i = xn->n - 1; i >= 0; i--)

{

xn->sum[i] = xn->sum[i + 1] + pow(-1, i) \* (pow(x, i + 1) / (i + 1));

xn->err[i] = fabs(xn->sum[i] - ideal);

}

}

**Приложение №5**

void Main(void(\*func)(Xn\*, double, double), double (\*ideal)(double), double num, double x0, int count, bool isReversed)

{

struct Xn xn;

if (isReversed)

{

initR(&xn, ideal(num), x0, count);

}

else

{

init(&xn, ideal(num), x0, count);

}

func(&xn, num, ideal(num));

if (isReversed)

{

printR(&xn);

}

else

{

print(&xn);

}

Free(&xn);

}

**Приложение №6**

void print(Xn\* xn )

{

for (int i = 0; i < xn->n; i++)

{

printf("sum %lf err %lf\n", xn->sum[i], xn->err[i]);

}

printf("\n");

}

**Приложение №7**

void printR(Xn\* xn )

{

for (int i = xn->n - 1; i >= 0; i--)

{

printf("sum %lf err %lf\n", xn->sum[i], xn->err[i]);

}

printf("\n");

}

**Приложение №8**

void Free(Xn\* xn)

{

free(xn->err);

free(xn->sum);

xn->err = NULL;

xn->sum = NULL;

}

**Приложение №9**

void init(Xn\* xn, double ideal, double x0, int count)

{

xn->n = count;

xn->err = (double\*)malloc(sizeof(double) \* count);

xn->sum = (double\*)malloc(sizeof(double) \* count);

xn->sum[0] = x0;

xn->err[0] = ideal - x0;

}

**Приложение №10**

void initR(Xn\* xn, double ideal, double x0, int count)

{

xn->n = count;

xn->err = (double\*)malloc(sizeof(double) \* count);

xn->sum = (double\*)malloc(sizeof(double) \* count);

xn->sum[count - 1] = x0;

xn->err[count - 1] = ideal - x0;

}