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**

**«Использование ряда Тейлора для вычисления значения функции»**

**Выполнил**:

Студент группы 3824Б1ПМ1

Кутьин Артём Павлович

**Проверила**:

Бусько П.В.

Нижний Новгород

2025

**Содержание**

[Введение 3](#_Toc193393866)

[Постановка задачи 4](#_Toc193393867)

[Руководство пользователя 5](#_Toc193393868)

[Описание программной реализации 6](#_Toc193393869)

[Результаты экспериментов 9](#_Toc193393870)

[Заключение 15](#_Toc193393871)

[Литература 15](#_Toc193393872)

[Приложение 1. 16](#_Toc193393873)

# Введение

Формула Тейлора — это способ разложения функции в бесконечный ряд, используя её значения и значения её производных в заданной точке. Она используется для приблизительного вычисления значения функции в точке. Разложение в ряд Тейлора удобно использовать для некоторых типов функций. Из-за аппроксимации сложных функций она широко используется в моделировании и инженерии.

# Постановка задачи

Основная цель лабораторной работы – реализовать разложение четырёх функций – sin x, cos x, ex, ln(1+x) – в ряд Тейлора. Также, необходимо посмотреть разницу прямого и обратного суммирования, сравнить их с истинным значением функции. Составить график зависимости погрешности истинного значения и суммы Тейлора от количества слагаемых и сделать выводы.

# Руководство пользователя

Основная концепция программы – сравнивание истинного значения c суммой Тейлора четырёх функций: sin x, cos x, ex, ln(1+x).

Программа состоит из 3 этапов:

1. Выбор функции
2. Ввод данных для разложения
3. Вывод полученных результатов

На первом этапе пользователь выбирает функцию для разложения.

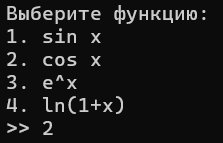


Рис. 1 – Выбор функции

Далее программа получает на вход значения аргумента x и количество слагаемых для разложения.

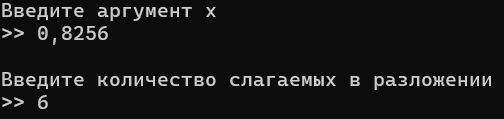


Рис. 2 – Ввод данных для разложения

Также программа получает на вход тип суммирования – прямое и обратное.

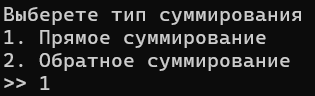


Рис. 3 – Выбор типа суммирования

При вводе неверного типа данных выводится сообщение об ошибке.

В итоге программа после вычисления суммы выводит на экран сначала истинное значение функции (используя модуль “math.h”), затем – значение вычисленной суммы Тейлора, а также погрешность суммы к истинному значению. Программа выводит двадцать знаков после запятой.

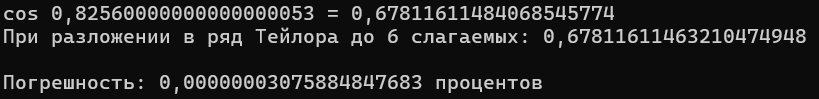


Рис. 4 – Вывод

# Описание программной реализации

Проект состоит из трёх файлов:

* “Лабораторная работа 2. Ряд Тейлора.cpp” – основной файл.

В начале подключаются библиотеки: “stdio.h”, “stdib.h”, “locale.h”, “math.h”, а также вызов зависимого файла “functions.h”

После ввода количества слагаемых для разложения создаётся пустой массив arr, длина которого равна количеству слагаемых. Позже в этот массив будут записаны значения слагаемых при разложении.

* “functions.cpp” – файл с четырьмя функциями для разложения, функции прямого и обратного суммирования и основная функция decision для ввода и вывода всех данных.

Функция разложения sin x:

Прототип: void sinus(long double x, int n, long double \*arr);

Из формулы разложения синуса следует рекуррентная формула вычисления i-ого слагаемого для более быстрого выполнения программы.

s0=x;

Соответственно, в нашем изначально пустом массиве arr длины n (то есть n слагаемых) мы задаём значение arr[0] равное x, а остальные элементы массива вычисляем по формуле выше. Аналогично будут вычислятся элементы для всех 4 функций.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 5 – Функция синуса

Функция разложения cos x:

Прототип: void cosinus(long double x, int n,long double\* arr);

Разложение косинуса:

Рекуррентная формула:

s0=1;

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 6 – Функция косинуса

Функция разложения ex:

Прототип: void e\_x(long double x, int n,long double\* arr);

Разложение ex:

Рекуррентная формула:

s0=1;

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 7 – Функция ex

Функция разложения ln(1+x):

Прототип:

void nat\_log(long double x, int n,long double\* arr);

Разложение ln (1+x): (Для |x|<1)

Рекуррентная формула:

s0=x;

Изображение выглядит как текст, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 8 – Функция ln (1+x)

Функции прямого и обратного суммирования:

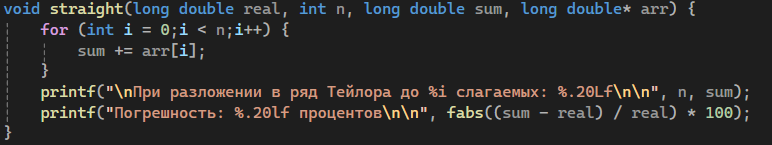
Прототипы:

void straight(long double real, int n, long double sum, long double\* arr);

void reverse(long double real, int n, long double sum, long double\* arr);

На вход функции прямого и обратного суммирования получает число real – истинное значение функции, количество слагаемых, sum и массив arr. Функции суммирования прибавляют к изначально нулевой сумме sum элементы уже заполненного массива arr. В прямом суммировании – от 0го до n-1 члена, в обратном – с n-1 до 0го члена.

Также функции straight и reverse выводят значение погрешности, которое считается по формуле |(значение суммы – истинное значение)/истинное значение|\*100.



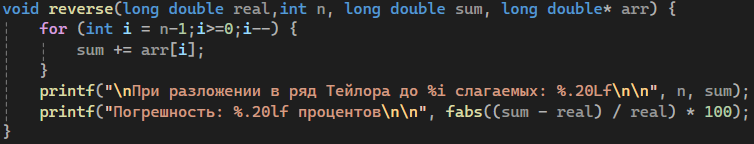


Рис. 9 и 10 - Функции прямого и обратного суммирования

Функция decision:

Decision (см. Приложение 1) отвечает за выбор функции для разложения, вызова соответствующих функций разложения и суммирования и вывода (в функциях straight и reverse).

* “functions.h” – заголовочный файл всех функций.

# Результаты экспериментов

Проведём четыре пары экспериментов соответственно для 4 раскладываемых в сумму функций. Возьмём произвольное значение аргумента, и посмотрим, как ведёт себя погрешность при увеличении количества слагаемых, а также выясним, существует ли разница в значениях между прямым и обратным суммированием.

Разложение sin x при x=0,924

*Прямое суммирование:*

|  |  |
| --- | --- |
| Количество слагаемых | Погрешность |
| 1 | 15,78678513879880000000 |
| 2 | 0,68921123931165300000 |
| 3 | 0,01412927487423690000 |
| 4 | 0,00016823109813762100 |
| 5 | 0,00000130872768428734 |
| 6 | 0,00000000717262547092 |
| 7 | 0,00000000002920180476 |
| 8 | 0,00000000000008347348 |
| 9 | 0,00000000000001391225 |
| 10 | 0,00000000000001391225 |

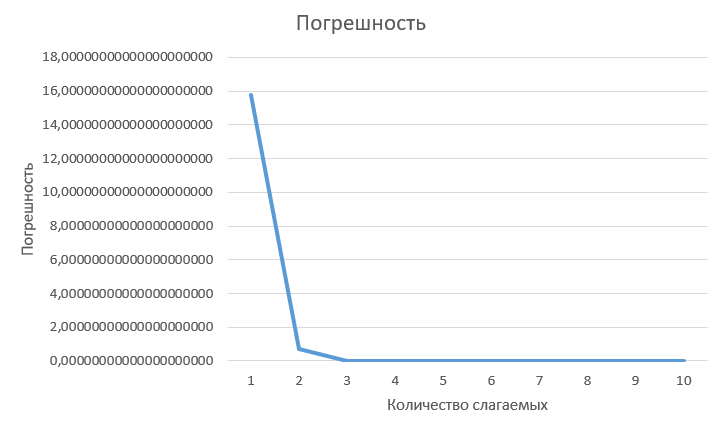


Таблица 1, Рис. 11 - Прямое суммирование, разложение sin x

*Обратное суммирование:*

|  |  |
| --- | --- |
| Количество слагаемых | Погрешность |
| 1 | 15,78678513879880000000 |
| 2 | 0,68921123931165300000 |
| 3 | 0,01412927487423690000 |
| 4 | 0,00016823109813762100 |
| 5 | 0,00000130872768428734 |
| 6 | 0,00000000717262547092 |
| 7 | 0,00000000002918789251 |
| 8 | 0,00000000000008347348 |
| 9 | 0,00000000000000000000 |
| 10 | 0,00000000000000000000 |

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Таблица 2, Рис. 12 - Обратное суммирование, разложение sin x

Разложение cos x при x=0,553

*Прямое суммирование:*

|  |  |
| --- | --- |
| Количество слагаемых | Погрешность |
| 1 | 17,51535502850970000000 |
| 2 | 0,45327157444699400000 |
| 3 | 0,00464240328830688000 |
| 4 | 0,00002540389893992330 |
| 5 | 0,00000008641263349144 |
| 6 | 0,00000000020033400233 |
| 7 | 0,00000000000032617063 |
| 8 | 0,00000000000001304683 |
| 9 | 0,00000000000001304683 |
| 10 | 0,00000000000001304683 |

Изображение выглядит как текст, снимок экрана, число, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Таблица 3, Рис. 13 - Прямое суммирование, разложение cos x

*Обратное суммирование:*

|  |  |
| --- | --- |
| Количество слагаемых | Погрешность |
| 1 | 17,51535502850970000000 |
| 2 | 0,45327157444699400000 |
| 3 | 0,00464240328830688000 |
| 4 | 0,00002540389893992330 |
| 5 | 0,00000008641263349144 |
| 6 | 0,00000000020032095550 |
| 7 | 0,00000000000032617063 |
| 8 | 0,00000000000000000000 |
| 9 | 0,00000000000000000000 |
| 10 | 0,00000000000000000000 |

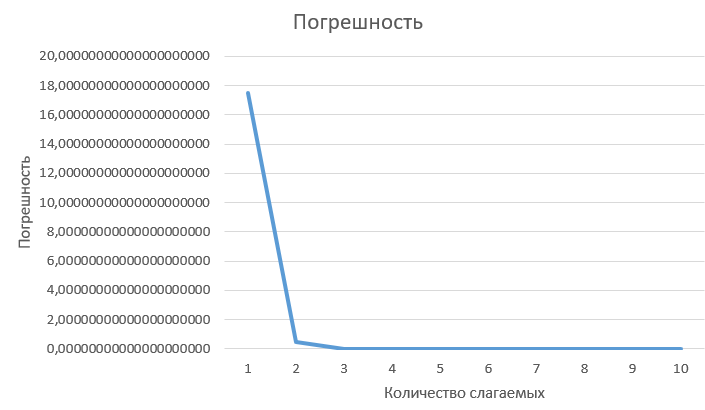


Таблица 4, Рис. 14 - Обратное суммирование, разложение cos x

Разложение ex при x=0,367

*Прямое суммирование:*

|  |  |
| --- | --- |
| Количество слагаемых | Погрешность |
| 1 | 30,71903549556080000000 |
| 2 | 5,29292152243164000000 |
| 3 | 0,62722960836244900000 |
| 4 | 0,05645996420798860000 |
| 5 | 0,00409184935682192000 |
| 6 | 0,00024802972674395400 |
| 7 | 0,00001291609270891090 |
| 8 | 0,00000058942074563250 |
| 9 | 0,00000002393467074391 |
| 10 | 0,00000000087541142470 |

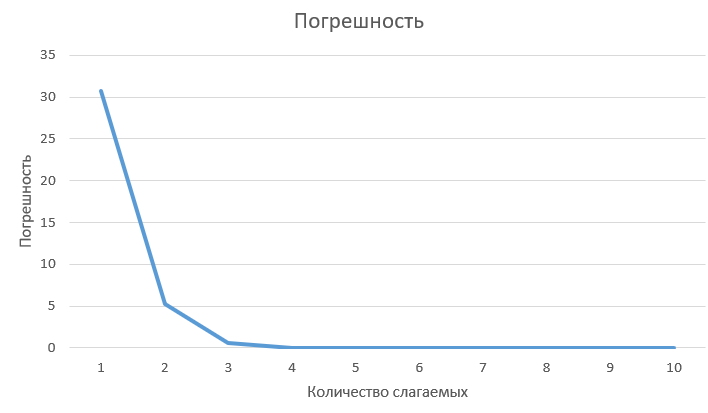


Таблица 5, Рис. 15 - Прямое суммирование, разложение ex

*Обратное суммирование:*

|  |  |
| --- | --- |
| Количество слагаемых | Погрешность |
| 1 | 30,71903549556080000000 |
| 2 | 5,29292152243164000000 |
| 3 | 0,62722960836244900000 |
| 4 | 0,05645996420797330000 |
| 5 | 0,00409184935680653000 |
| 6 | 0,00024802972672857100 |
| 7 | 0,00001291609269352750 |
| 8 | 0,00000058942073024903 |
| 9 | 0,00000002393465536045 |
| 10 | 0,00000000087539604124 |

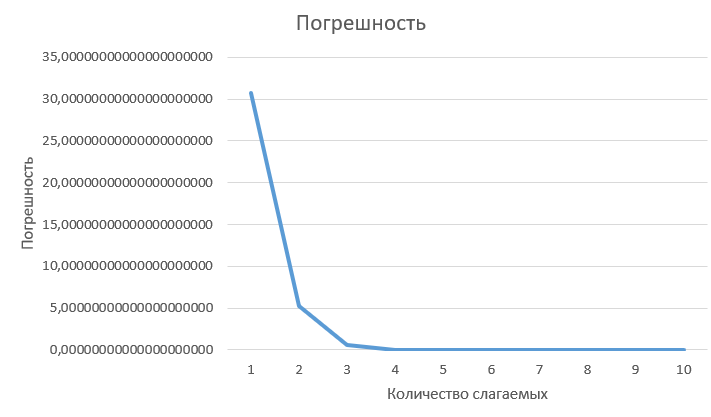


Таблица 6, Рис. 16 - Обратное суммирование, разложение ex

Разложение ln (1+x) при x=0,129

*Прямое суммирование:*

|  |  |
| --- | --- |
| Количество слагаемых | Погрешность |
| 1 | 6,31959978491143000000 |
| 2 | 0,53801440121534500000 |
| 3 | 0,05174041879155990000 |
| 4 | 0,00531836004410961000 |
| 5 | 0,00057010593173228900 |
| 6 | 0,00006290416067437530 |
| 7 | 0,00000708866954320409 |
| 8 | 0,00000081177116444835 |
| 9 | 0,00000009414604063041 |
| 10 | 0,00000001103094643430 |



Таблица 7, Рис. 17 - Прямое суммирование, разложение ln (1+x)

*Обратное суммирование:*

|  |  |
| --- | --- |
| Количество слагаемых | Погрешность |
| 1 | 6,31959978491143000000 |
| 2 | 0,53801440121534500000 |
| 3 | 0,05174041879154850000 |
| 4 | 0,00531836004410961000 |
| 5 | 0,00057010593172085100 |
| 6 | 0,00006290416067437530 |
| 7 | 0,00000708866954320409 |
| 8 | 0,00000081177116444835 |
| 9 | 0,00000009414602919258 |
| 10 | 0,00000001103095787213 |



Таблица 8, Рис. 18 - Обратное суммирование, разложение ln (1+x)

# Заключение

В процессе выполнения этой лабораторной работы была произведена оценка функций sin x, cos x, ex и ln(1+x) с помощью разложения в ряд Тейлора. Также было осуществлено сравнение полученных результатов, рассчитанных методами прямого и обратного суммирования, с результатами, полученными при использовании стандартных функций из библиотеки math.h (то есть истинным значением).

Благодаря проведённым экспериментам было доказано, что существует разница (но очень маленькая) в значениях при прямом и обратном суммировании у некоторых слагаемых при одинаковых данных. Очевидно, что для получения более точного значения функции нужно разложение на как можно больше слагаемых. Из четырёх исследуемых функций ex и ln (1+x) имеют бо́льшую погрешность, чем у синуса и косинуса. Для них желательнее разложение на бо́льшее количество слагаемых.

# Литература

1. Керниган Б., Ритчи Д., Фьюэр А. Язык программирования СИ //М.: Финансы и статистика. – 1992.
2. Кудрявцев Л. Д. Курс математического анализа (в двух томах) : Учебник для студентов университетов и втузов. – М.: Высш. школа, 1981, т. I. – 687 с., ил.

# Приложение 1.

void decision(int decision1, int decision2, long double x, int n, long double\* arr) {

long double sum = 0.0;

if (decision1 == 1) {

sinus(x, n, arr);

long double real = sin(x);

printf("\nsin %.20Lf = %.20Lf", x, sin(x));

if (decision2 == 1) {

straight(real,n, sum, arr);

}

else {

reverse(real,n, sum, arr);

}

}

if (decision1 == 2) {

cosinus(x, n, arr);

long double real = cos(x);

printf("\ncos %.20Lf = %.20Lf", x, cos(x));

if (decision2 == 1) {

straight(real,n, sum, arr);

}

else {

reverse(real,n, sum, arr);

}

}

if (decision1 == 3) {

e\_x(x, n, arr);

long double real = exp(x);

printf("\ne^ %.20Lf = %.20Lf", x, exp(x));

if (decision2 == 1) {

straight(real,n, sum, arr);

}

else {

reverse(real,n, sum, arr);

}

}

if (decision1 == 4) {

nat\_log(x, n, arr);

long double real = log(1 + x);

printf("\nln(1 + %.20Lf) = %.20Lf", x, log(1+x));

if (decision2 == 1) {

straight(real,n, sum, arr);

}

else {

reverse(real,n, sum, arr);

}

}

}