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»**

**Выполнила**:

студентка группы 3824Б1ПМ-1

Каргалёва С. С.

**Проверила**:

Бусько П.В.

Нижний Новгород

2025

**Содержание**

[Введение 3](#_bfo2ldmwn8jt)

[Постановка задачи 4](#_wrqkwzvusrs1)

[Руководство пользователя 5](#_d9vweoft1lcd)-6

[Описание программной реализации](#_zf88npiwfund) 7-9

[Результаты экспериментов](#_tdlzlwokonvd) 10-17

[Заключение](#_6g8d5euzada4) 18

[Литература](#_bia477m0oii) 19

[Приложение](#_9ynvyfikw7yr) 20-23

# Введение

В современном программировании важную роль играют приближённые вычисления значений функций, особенно когда аналитическое решение оказывается сложным или невозможным. Одним из мощных инструментов для достижения этой цели является ряд Тейлора – математическая конструкция, позволяющая представлять сложные функции в виде бесконечных сумм, что значительно упрощает их вычисление в окрестности заданной точки.

# Постановка задачи

Целью данной лабораторной работы является реализация программы для вычисления значений математических функций sin(x), cos(x), e\*\*x, ln(1+x) с использованием их разложений в ряды Тейлора.

# Руководство пользователя

Пользователь должен вводить только данные, которые предлагает ему программа!

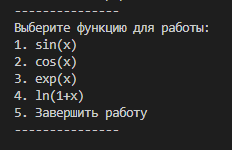
Для корректной работы программы пользователь должен запустить файл labs1.c (при наличии компилятора для языка c/c++:



*Рисунок 1* (стартовый файл labs1.cpp.)

При отсутствии компилятора для языка c/c++ labs1.exe.

Далее, пользователь сможет выбрать функция для проверки:



*Рисунок 2* (стартовое меню программы)

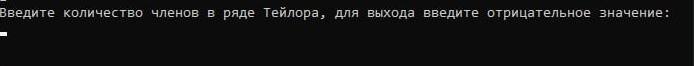
Нажмите нужную цифру, указанную в столбце выбора, для выбора функция, с которой будет производится проверка.

При выборе цифры пользователь может ввести число х, которое будет использоваться:



*Рисунок 3* (пример работы программы при выборе 1.)

После выбора числа x, пользователь должен ввести количество членов в разложении Тейлора, также пользователь может выйти из этого меню, введя отрицательное значение и выбрать другую функцию:



*Рисунок 4* (меню программы после выбора 1 для функции sin(x).)

После введение количества строк пользователь получит два измерения функции, используя прямое и обратное суммирование, и сможет сравнить их с требуемым:

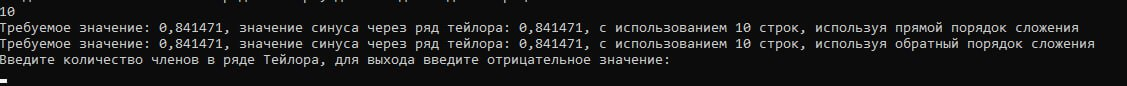


Рисунок 5 (введение количества строк 10 в ряде тейлора для функции синус и значения x=1)

\*При нажатии на 5 в главном меню - программа будет завершена.

# Описание программной реализации

Проект состоит из:

1. 2 файлов .cpp: labs1, functions.
2. 1 файла .h: Functions. - заголовочный файл, связывающий два cpp файла.

Файл **labs1.cpp** представляет из себя главный файл, в котором содержится функция:

1. **int main():**

Функция, в которой создаются:

1. объекты структуры: object , value, len;

2. Переменные для работы программы:

a. int function - выбор функции из предложенного списка;

b. double rez\_teilor - сохранение результатов измерений, используя ряд тейлора;

c. double value\_x - сохранение значения x.

d. int len\_teilor - количество членов в разложении тейлора.

В функции реализовано основное меню и взаимодействие пользователя и программы, используя циклы while и структуры switch.

2. **struct Values**

Структура значений точки X и длины строчек в ряде Тейлора.

Содержит:  
 1. double x;

2. int len;

Файл **functions.cpp** представляет из себя файл, в котором содержатся функции для реализации подсчета количества значения, используя ряд Тейлора:

1. int factorial( int n) - функция возведения числа в факториал.

a. Функция получает:

1. число, факториал которого нужен - int n;

b. Использует дополнительные библиотеки: math.h s и stdio.h.

c. Возвращает целочисленное значение факториала, используя при этом рекурсивный метод.

2. double sin\_teilor\_pr(double value\_x, int len\_teilor) - функция, которая раскладывает синус, используя ряд Тейлора, при этом происходит прямое суммирование значений (от 1 до n).

a. Функция получает:

1. значения x: double value\_x;

2. количество членов в разложении:int len\_teilor;

b. Использует дополнительные библиотеки: math.h s и stdio.h.

c. Возвращает значение типа double.

Реализаци представляет из себя цикл while с условием, пока переменная счетчик меньше длины ряда Тейлора, в этот момент ведется подсчет каждого многочлена в разложении и суммирование этих значений, при этом счетчик увеличивается каждый проход цикла. Как только нарушается условие, поставленное в цикле while, функция завершает свою работу.

3. double sin\_teilor\_obr(double value\_x, int len\_teilor) - функция, которая раскладывает синус, используя ряд Тейлора, при этом происходит обратное суммирование значений (от n до 1)

a. Функция получает:

1. значения x: double value\_x;

2. длину строки, до которой нужно раскладывать:int len\_teilor;

b. Использует дополнительные библиотеки: math.h и stdio.h.

c. Возвращает значение типа double.

Реализаци представляет из себя цикл while с условием, пока переменная счетчик больше или равна 0, с условием, что переменная счетчик принимает значение длины ряда Тейлора, в этот момент ведется подсчет каждого многочлена в разложении и суммирование этих значений, при этом счетчик уменьшается каждый проход цикла. Как только нарушается условие, поставленное в цикле while, функция завершает свою работу.

4. double cos\_teilor\_pr(double value\_x, int len\_teilor) - функция, которая раскладывает cos, используя ряд Тейлора, при этом происходит прямое суммирование значений (от 1 до n).

a. Функция получает:

1. значения x: double value\_x;

2. длину строки, до которой нужно раскладывать:int len\_teilor;

b. Использует дополнительные библиотеки: math.h и stdio.h.

c. Возвращает значение типа double.

Аналогичная реализация, что и в случаи синуса, меняется лишь систематика вычисления ряда.

5. double cos\_teilor\_obr(double value\_x, int len\_teilor) - функция, которая раскладывает cos, используя ряд Тейлора, при этом происходит обратное суммирование значений (от n до 1)

a. Функция получает:

1. значения x: double value\_x;

2. длину строки, до которой нужно раскладывать:int len\_teilor;

b. Использует дополнительные библиотеки: math.h и stdio.h.

c. Возвращает значение типа double.

Аналогичная реализация, что и в случаи синуса, меняется лишь систематика вычисления ряда.

6. double exp\_teilor\_pr(double value\_x, int len\_teilor) - функция, которая раскладывает , используя ряд Тейлора, при этом происходит прямое суммирование значений (от 1 до n).

a. Функция получает:

1. значения x: double value\_x;

2. длину строки, до которой нужно раскладывать:int len\_teilor;

b. Использует дополнительные библиотеки: math.h и stdio.h.

c. Возвращает значение типа double.

Аналогичная реализация, что и в случаи синуса, меняется лишь систематика вычисления ряда.

7. double exp\_teilor\_obr(double value\_x, int len\_teilor) - функция, которая раскладывает , используя ряд Тейлора, при этом происходит обратное суммирование значений (от n до 1)

a. Функция получает:

1. значения x: double value\_x;

2. длину строки, до которой нужно раскладывать:int len\_teilor;

b. Использует дополнительные библиотеки: math.h и stdio.h.

c. Возвращает значение типа double.

Аналогичная реализация, что и в случаи синуса, меняется лишь систематика вычисления ряда.

8. double log\_teilor\_pr(double value\_x, int len\_teilor) - функция, которая раскладывает log(x+1), используя ряд Тейлора, при этом происходит прямое суммирование значений (от 1 до n).

a. Функция получает:

1. значения x: double value\_x;

2. длину строки, до которой нужно раскладывать:int len\_teilor;

b. Использует дополнительные библиотеки: math.h и stdio.h.

c. Возвращает значение типа double.

Аналогичная реализация, что и в случаи синуса, меняется лишь систематика вычисления ряда.

9. double exp\_teilor\_obr(double value\_x, int len\_teilor) - функция, которая раскладывает log(1+x), используя ряд Тейлора, при этом происходит обратное суммирование значений (от n до 1)

a. Функция получает:

1. значения x: double value\_x;

2. длину строки, до которой нужно раскладывать:int len\_teilor;

b. Использует дополнительные библиотеки: math.h и stdio.h.

c. Возвращает значение типа double.

Аналогичная реализация, что и в случаи синуса, меняется лишь систематика вычисления ряда.

# Результаты экспериментов

Данные эксперимента при запуске программы и вводе числа x = 1.

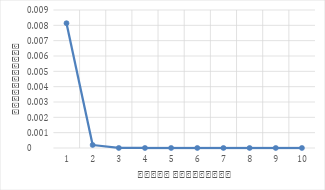
Все данные представлены ниже в таблицах и на графиках.

**А) Прямое суммирование.**

1. Функция sin(x) (*Таблица 1,График 1*).

*Таблица 1*

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0,008138 |
| 2 | 0,000196 |
| 3 | 0,000003 |
| 4 | 0,000000 |
| 5 | 0,000000 |
| 6 | 0,000000 |
| 7 | 0,000000 |
| 8 | 0,000000 |
| 9 | 0,000000 |
| 10 | 0,000000 |



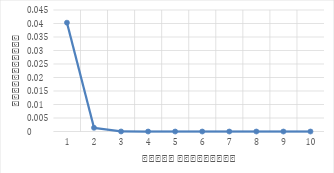
*График 1*

Заметим, что уже при 3 слагаемых точность высока и практически не отличается от заданной.

1. Функция cos(x) (*Таблица 2, График 2*).

*Таблица 2*

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0,040302 |
| 2 | 0,001364 |
| 3 | 0,000025 |
| 4 | 0,000000 |
| 5 | 0,000000 |
| 6 | 0,000000 |
| 7 | 0,000000 |
| 8 | 0,000000 |
| 9 | 0,000000 |
| 10 | 0,000000 |



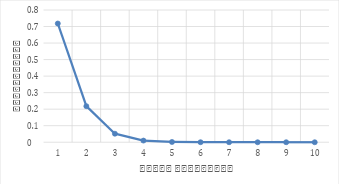
*График 2*

Не очень трудно заметить, что в случае с косинусом, ряд сходится к нужному значению так же быстро, как с синусом.

3) Функция (*Таблица 3, График 3*).

*Таблица 3*

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0,718282 |
| 2 | 0,218282 |
| 3 | 0,051615 |
| 4 | 0,009948 |
| 5 | 0,001615 |
| 6 | 0,000226 |
| 7 | 0,000028 |
| 8 | 0,000003 |
| 9 | 0,000000 |
| 10 | 0,000000 |



*График 3*

В случае с функцией e\*\*x ситуация заметно меняется. Необходимо заметить, что кривая заметно позже приближается к желаемому значению, нежели чем косинус и синус.

4) Функция (*Таблица 4, График 4*).

*Таблица 4*

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0,193147 |
| 2 | 0,140186 |
| 3 | 0,109814 |
| 4 | 0,090186 |
| 5 | 0,076481 |
| 6 | 0,066377 |
| 7 | 0,058623 |
| 8 | 0,052488 |
| 9 | 0,047512 |
| 10 | 0,043397 |



*График 4*

На данном графике мы можем отследить плавное достижение желаемого результата.

**Б) Обратное суммирование.**

1. Функция sin(x) (*Таблица 5, График 5*).

Таблица 5

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0,841471 |
| 2 | 0,841471 |
| 3 | 0,841471 |
| 4 | 0,841471 |
| 5 | 0,841471 |
| 6 | 0,841468 |
| 7 | 0,841667 |
| 8 | 0,833333 |
| 9 | 1,000000 |
| 10 | 0,000000 |

График 5

Необходимо заметить, что при 10 слагаемых значение истинно.

2) Функция cos(x) (*Таблица 6, График 6*).

Таблица 6

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0,540302 |
| 2 | 0,540302 |
| 3 | 0,540302 |
| 4 | 0,540302 |
| 5 | 0,540303 |
| 6 | 0,540278 |
| 7 | 0,541667 |
| 8 | 0,500000 |
| 9 | 1,000000 |
| 10 | 0,000000 |

График 6

Необходимо заметить, что 10 слагаемым значение истинно. (см. *График 6*).

3) Функция (*Таблица 7, График 7*).

Таблица 7

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 2,718279 |
| 2 | 2,718254 |
| 3 | 2,718056 |
| 4 | 2,716667 |
| 5 | 2,708333 |
| 6 | 2,666667 |
| 7 | 2,500000 |
| 8 | 2,000000 |
| 9 | 1,000000 |
| 10 | 0,000000 |

График 7

По графику можно заметить плавное уменьшение к истинному значению при 10 слагаемых. Такая же ситуация рассматривается при прямом суммировании.

4) Функция (*Таблица 8, График 8*).

Таблица 8

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0,702238 |
| 2 | 0,591127 |
| 3 | 0,716127 |
| 4 | 0,573270 |
| 5 | 0,739937 |
| 6 | 0,539937 |
| 7 | 0,789937 |
| 8 | 0,456603 |
| 9 | 0,956603 |
| 10 | 0,043397 |

График 8

Необходимо заметить, что погрешность колеблется в определенных пределах, но при достижении 10 слагаемых близко к истинному значению.

# Заключение

В результате тестирования программы мы смогли проанализировать зависимость точности вычислений от количества членов ряда. Было установлено, что увеличение числа членов ряда значительно повышает точность результата, однако также увеличивает вычислительные затраты. Это подчеркивает важность выбора оптимального числа членов в зависимости от требований к точности и доступных ресурсов.

Рассмотрев два способа суммирования, можно заметить, что они практически не отличаются друг от друга. То есть метод суммирования не настолько важен, как число слагаемых в сумме.

Таким образом, лабораторная работа не только углубила наши знания о рядах Тейлора, но и развила навыки программирования, позволяя применять теоретические знания на практике.

**Литература**

1. Самарский А. А., Тихонов А. Н. Численные методы математической физики
2. Страуструп Б. Язык программирования С
3. Кнут Д. Э. Искусство программирования: Сортировка и поиск. – Издательский дом Вильямс, 2000. – Т. 3.

# Приложение

#include "Functions.h"

int factorial(int n)

{

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

return n \* factorial(n - 1);

}

double sin\_teilor\_pr(double value\_x, int len\_teilor)

{

int k = 0;

double s = 0.0;

while (k < len\_teilor)

{

double c = (pow(-1, k) \* pow(value\_x, 2 \* k + 1));

int b = factorial(2 \* k + 1);

double m = c / b;

s += m;

k++;

}

return s;

}

double sin\_teilor\_obr(double value\_x, int len\_teilor)

{

int k = len\_teilor;

double s = 0.0;

while (k >= 0)

{

double c = (pow(-1, k) \* pow(value\_x, 2 \* k + 1));

int b = factorial(2 \* k + 1);

double m = c / b;

s += m;

k--;

}

return s;

}

double cos\_teilor\_pr(double value\_x, int len\_teilor)

{

int k = 0;

double s = 0.0;

while (k < len\_teilor)

{

double c = (pow(-1, k) \* pow(value\_x, 2 \* k));

int b = factorial(2 \* k);

double m = c / b;

s += m;

k++;

}

return s;

}

double cos\_teilor\_obr(double value\_x, int len\_teilor)

{

int k = len\_teilor;

double s = 0.0;

while (k >=0)

{

double c = (pow(-1, k) \* pow(value\_x, 2 \* k));

int b = factorial(2 \* k);

double m = c / b;

s += m;

k--;

}

return s;

}

double exp\_teilor\_pr(double value\_x, int len\_teilor)

{

int k = 0;

double s = 0.0;

while (k < len\_teilor)

{

double c = pow(value\_x, k);

int b = factorial(k);

double m = c / b;

s += m;

k++;

}

return s;

}

double exp\_teilor\_obr(double value\_x, int len\_teilor)

{

int k = len\_teilor;

double s = 0.0;

while (k >=0)

{

double c = pow(value\_x, k);

int b = factorial(k);

double m = c / b;

s += m;

k--;

}

return s;

}

double log\_teilor\_pr(double value\_x, int len\_teilor)

{

int k = 0;

double s = 0.0;

while (k < len\_teilor)

{

double c = pow(-1, k) \* pow(value\_x, (k + 1));

int b = k + 1;

double m = c / b;

s += m;

k++;

}

return s;

}

double log\_teilor\_obr(double value\_x, int len\_teilor)

{

int k = len\_teilor;

double s = 0.0;

while (k >=0)

{

double c = pow(-1, k) \* pow(value\_x, (k + 1));

int b = k + 1;

double m = c / b;

s += m;

k--;

}

return s;

}