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](#_6y7cyxzgfhfi)

[Постановка задачи 4](#_2eyatxhakbk)

[Руководство пользователя 5](#_xqudlspe7d2n)-6

[Описание программной реализации](#_5us374373koc) 7-8

[Результаты экспериментов](#_k772tkkjm7wp) 9-12

[Заключение](#_j29r1c6xjhhn) 13

[Литература](#_lym28g5fwqv) 14

[Приложение](#_75xfxg8osmva) 15-16

# 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!

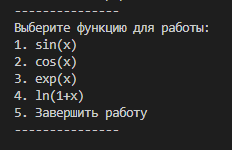
Для корректной работы программы пользователь должен запустить файл main.c (при наличии компилятора для языка c/c++:  
 

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

При отсутствии компилятора для языка c/c++ main.exe:

*Рисунок 2* (стартовый файл main.exe.)

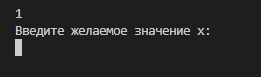
Далее, пользователь сможет выбрать функция для проверки:



*Рисунок 3* (стартовое меню программы)

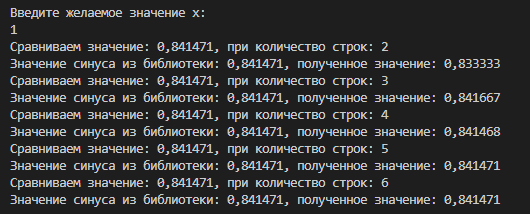
Нажмите нужную цифру, указанную в столбце выбора, для выбора функция, с которой будет производится проверка.

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



*Рисунок 4* (пример работы программы при выборе 1.)

После выбора числа, пользователь получает значения при разных количествах строк в разложении Тейлора:



*Рисунок 5* (меню программы после выбора 1 для функции sin(x).)

После завершения расчетов пользователь получает стартовое меню

При нажатии на 5 - программа будет завершена.

# Описание программной реализации

Проект состоит из:

1. 2 файлов .cpp: main, function.
2. 1 файла .h: function.

Файл **main.cpp** представляет из себя главный файл, в котором содержатся функции:

1. **int main():**

Функция, в которой создаются:

1. Два объекта структуры: test , p\_tes

2. Неизменяемый массив c длинами: int count\_string\_teilor\_test[10]

3. Переменная для выбора функции: int choice\_func

4. Переменная итогового значения вычислений: double rez\_val

В функции реализовано основное меню и взаимодействие пользователя и программы.

2. **struct Values**

Структура значений точки X и длины строчек в ряде Тейлора.

Файл **function.cpp** представляет из себя файл, в котором содержатся функции:

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.

2. double teilor\_sin(double x, int count\_string) - функция, которая раскладывает синус, используя ряд Тейлора

a. Функция получает:

1. значения x: double x.

2. длину строки, до которой нужно раскладывать:int count\_string

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

c. Возвращает значение типа double.

3. double teilor\_cos(double x, int count\_string) - функция, которая раскладывает косинус, используя ряд Тейлора

a. Функция получает:

1. значения x: double x.

2. длину строки, до которой нужно раскладывать:int count\_string

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

c. Возвращает значение типа double.

4. double teilor\_exp (double x, int count\_string) - функция, которая раскладывает e\*\*x, используя ряд Тейлора

a. Функция получает:

1. значения x: double x.

2. длину строки, до которой нужно раскладывать:int count\_string

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

c. Возвращает значение типа double.

5. double teilor\_ln (double x, int count\_string) - функция, которая раскладывает ln(x+1), используя ряд Тейлора

a. Функция получает:

1. значения x: double x.

2. длину строки, до которой нужно раскладывать:int count\_string

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

c. Возвращает значение типа double.

# Результаты экспериментов

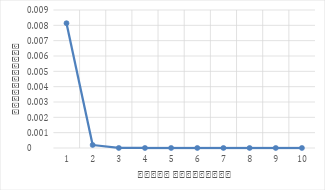
Данные эксперимента при запуске программы и вводе числа x = 1.

Все данные представлены ниже в таблицах и на графиках.

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 |



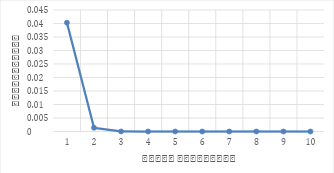
*График 1*

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

2) 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 |



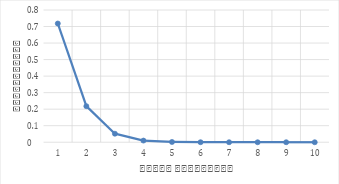
*График 2*

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 |



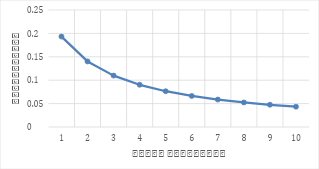
*График 3*

В случае с функцией e\*\*x ситуация заметно меняется. На  *Графике 3* 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.

# 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.

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 "function.h"

int factorial( int n)

{

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

return n \* factorial(n - 1);

}

double teilor\_sin(double x, int count\_string)

{

int k = 0;

double s=0.0;

while (k<count\_string)

{

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

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

double m = c/b;

s+=m;

k++;

}

return s;

}

double teilor\_cos(double x, int count\_string)

{

int k = 0;

double s=0.0;

while (k<count\_string)

{

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

int b = factorial(2\*k);

double m = c/b;

s+=m;

k++;

}

return s;

}

double teilor\_exp (double x, int count\_string)

{

int k = 0;

double s=0.0;

while (k<count\_string)

{

double c = pow(x,k);

int b = factorial(k);

double m = c/b;

s+=m;

k++;

}

return s;

}

double teilor\_ln (double x, int count\_string)

{

int k = 0;

double s=0.0;

while (k<count\_string)

{

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

int b = k+1;

double m = c/b;

s+=m;

k++;

}

return s;

}