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 3824Б1ПМ1-2
Теселкин К. И.

Проверил:
Бусько П. В.

Нижний Новгород
2025

Содержание

Постановка задачи

Требуется выполнить вычисление математических функций с использованием рядов Тейлора. Также необходимо провести сравнение значений функций, полученных различными методами суммирования, с результатами библиотечных функций из math.h.

Метод решения

Для подсчета членов ряда используются формулы Маклорена.

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^{n+1} \frac{x^{2n-1}}{(2n-1)!}$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots + (-1)^{n} \frac{x^{2n}}{(2n)!}$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!}$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots + (-1)^{n-1} \frac{x^n}{n}$$

Прямое суммирование - последовательное сложение элементов ряда в порядке их вычисления. Для формулы Тейлора это также означает сложение в порядке их убывания(каждое следующее слагаемое меньше предыдущего). Для прямого суммирования используется функция direct_sum.

Обратное суммирование - сложение элементов ряда в порядке их убывания, начиная с последнего элемента и переходя к первому. В случае формулы Тейлора это означает, что мы суммируем слагаемые в порядке их возрастания. Для обратного суммирования используется функция reverse_sum.

Руководство пользователя

Для работы с программой сначала нужно выбрать функцию (рис. 1). Далее необходимо выбрать метод суммирования (рис. 2). Далее вводится значение аргумента функции (рис. 3). После ввода данных, на экран выведется итоговые значения функции при различном количестве слагаемых (от 1 до 10) и значение посчитанное с помощью библиотеки math.h (рис. 4).

Рис. 1 Выбор функции

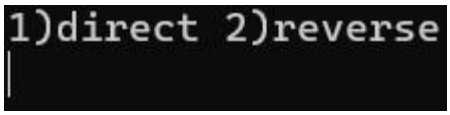

Рис. 2 Выбор метода суммирования

Рис. 3 Ввод значения аргумента функции

Рис. 4 Вывод

Описание программной реализации

Проект состоит из 3 файлов. В файле “Вычисление математических функций с использованием рядов.cpp” содержится основная функция main, где объявлены переменные, массив и указатели на функции, а также вызывается функция для расчёта и выведения результатов. Также в main написан интерфейс программы. Файл “Header.h” содержит прототипы функций. В файле “Source.cpp” находятся реализации прототипов.

Функции Sin, Cos, Exp, Log вычисляют слагаемые ряда для соответствующей функции. Они принимают 2 параметра: номер слагаемого типа int и значение x типа double. Возвращают значение типа double.

Функции direct_sum и reverse_sum изменяют массив так что каждый x[i] элемент становится равен сумме элементов от x[0] до x[i]. Они принимают 2 параметра: массив типа double и его длину. Функции ничего не возвращают.

Подтверждение корректности

Для подтверждения корректности в программе использовались математические функции из библиотеки math.h. В результатах выводится значение, полученное math.h и моими функциями.

Результаты экспериментов

Чтобы получить результаты экспериментов, я записал погрешность для запусков с каждым n от 1 до 10, x = 2.0 для sin(x), cos(x), e^x и ln(x).

Число слагаемых	Погрешность при прямом суммировании	Погрешность при обратном суммировании
1	1,090702573	1,090702573
2	0,24263076	0,24263076
3	0,024035907	0,024035907
4	0,001360919	0,001360919
5	5,00159E-05	5,00159E-05
6	1,29086E-06	1,29086E-06
7	2,4694E-08	2,4694E-08
8	3,64234E-10	3,64234E-10
9	4,26992E-12	4,26992E-12
10	4,00791E-14	4,00791E-14

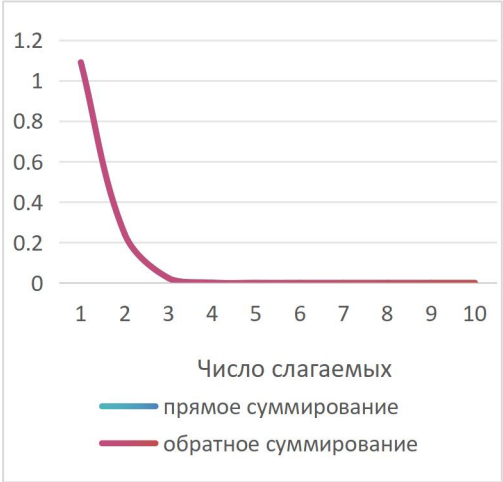

Рис. 5 данные для sin(x)

На рис. 5 представлен график зависимости модуля погрешности от числа слагаемых для функции sin(x). Ряд быстро сходится.

Число слагаемых	Погрешность при прямом суммировании	Погрешность при обратном суммировании
1	1,416146837	
2	0,583853163	
3	0,082813503	
4	0,006075386	
5	0,000273821	
6	8,36627E-06	
7	1,84845E-07	
8	3,09176E-09	
9	4,0518E-11	
10	4,26992E-13	

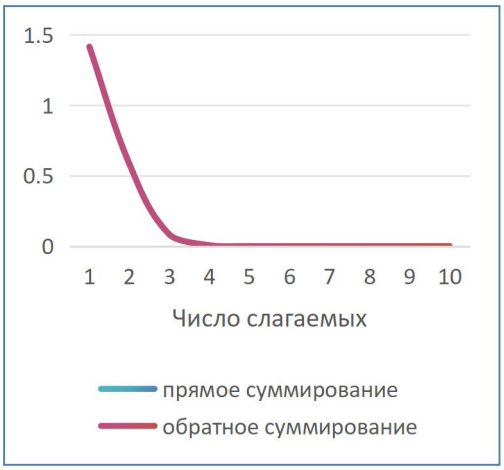

Рис. 6 данные для cos(x)

Также, как и с синусом, ряд с косинусом очень быстро сходится (см. рис. 6).

Число слагаемых	Погрешность при прямом суммировании	Погрешность при обратном суммировании
1	6,389056099	6,389056099
2	4,389056099	4,389056099
3	2,389056099	2,389056099
4	1,055722766	1,055722766
5	0,389056099	0,389056099
6	0,122389432	0,122389432
7	0,033500543	0,033500543
8	0,008103718	0,008103718
9	0,001754512	0,001754512
10	0,000343577	0,000343577

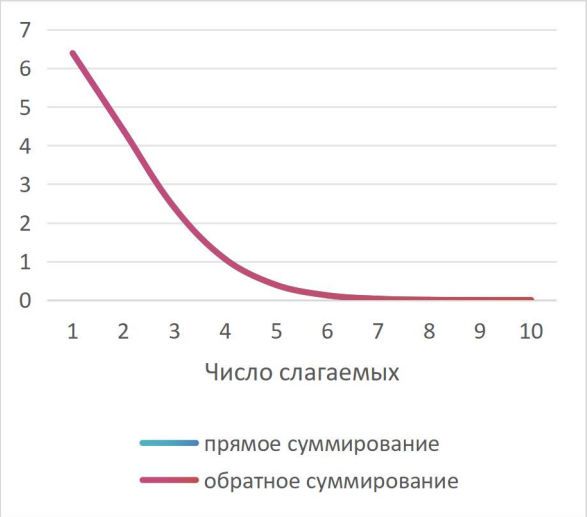

Рис. 7 данные для exp(x)

На рис. 7 видно, что кривая медленнее опускается , по сравнению с синусом и косинусом.

Число слагаемых	Погрешность при прямом суммировании	Погрешность при обратном суммировании
1	0,306852819	0,306852819
2	0,193147181	0,193147181
3	0,140186153	0,140186153
4	0,109813847	0,109813847
5	0,090186153	0,090186153
6	0,076480514	0,076480514
7	0,066376629	0,066376629
8	0,058623371	0,058623371
9	0,05248774	0,05248774
10	0,04751226	0,04751226

Рис. 8 данные для ln(x)

График для логарифма опускается медленнее все остальных (см. рис. 8). Точность достаточно высокая.

Вывод

Рассмотрев каждый метод суммирования я пришёл к выводу, что они практически не отличаются друг от друга. Число слагаемых в сумме играет большую роль чем метод суммирования.

Заключение

В процессе выполнения этой лабораторной работы было осуществлено вычисление функций и с применением разложения в ряд Тейлора. Также были сопоставлены результаты, полученные различными методами суммирования, с данными, полученными при помощи стандартных функций из библиотеки math.h.

Приложение

Вычисление математических функций с использованием рядов.cpp

```cpp
#include <math.h>
#include <iostream>
#include "Header.h"
#include <iomanip>

int main()
{
    double Xn[10];
    double (*MaclaurinXn)(int, double);
    void (*Type_sum)(double*, int);
    double (*standardFn)(double);
    double standard;

    std::cout << "1)Sin 2)Cos 3)Exp 4)Log\n";
    int w;
    std::cin >> w;
    switch(w) {
    case 1:MaclaurinXn = Sin; standardFn = sin; break;
    case 2:MaclaurinXn = Cos; standardFn = cos; break;
    case 3:MaclaurinXn = Exp; standardFn = exp; break;
    default:MaclaurinXn = Log; standardFn = log; break;
    }
    std::cout << "1)direct 2)reverse\n";
    std::cin >> w;
    switch (w) {
    case 1:Type_sum = direct_sum; break;
    default:Type_sum = reverse_sum; break;
    }
    std::cout << "x = ";
    double r;
    std::cin >> r;

    std::cout << std::fixed << std::setprecision(16);

    for (int i = 0; i < 10;i++) { Xn[i] = MaclaurinXn(i,r); }
    Type_sum(Xn, 10);
    standard = standardFn(r);
    std::cout << "standard = " << standard;
}
```

Header.h

```
#pragma once
double Sin(int n, double x);
double Cos(int n, double x);
double Exp(int n, double x);
double Log(int n, double x);
void direct_sum(double* x, int l);
void reverse_sum(double* x, int l);
```

Source.cpp

```
#include "math.h"
#include "Header.h"
#include <iostream>

double Sin(int n, double x) {
	double r=1;
	for (int i = 1; i <= 2*n+1; i++) { r *= i; }
	int k = -1;
	if (n % 2 == 0) { k = 1; }
	return k * pow(x, n * 2 + 1) / r;
}
double Cos(int n, double x){
	double r = 1;
	for (int i = 1; i <= 2 * n; i++) { r *= i; }
	int k = -1;
	if (n % 2 == 0) { k = 1; }
	return (k * pow(x, 2 * n)) / r;
}
double Exp(int n, double x){
	double r = 1;
	for (int i = 1; i <= n; i++) { r *= i; }
	return pow(x, n) / r;
}
double Log(int n, double x){
	int k = -1;
	if (n % 2 == 0) { k = 1; }
	return (k * pow(x-1, n+1)) / (n+1);
}

void direct_sum(double* x, int l) {
	std::cout << x[0] << "\n";
	for (int i = 1; i < l;i++) {
```

```
            x[i] = x[i - 1] + x[i];
            std::cout << x[i] << "\n";
      }
}

void reverse_sum(double* x, int l) {
      double w=0;
      for (int i=l-1; i >= 0; i--) {
            for (int j = i; j >= 0; j--) { w += x[j]; }
            x[i] = w;
            w = 0;
      }
      for (int i = 0; i <l;i++) { std::cout << x[i] << "\n"; }
}
```