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**

**«Математические функции»**

**Выполнил**:

студент группы 3823Б1ПМ1

Бабадаев Р. Б.

**Проверил**:

преподаватель каф. ВВСП,

Волокитин В.Д.

Нижний Новгород

2024

**Содержание**

[Постановка задачи 3](#_Toc162291807)

[Метод решения 4](#_Toc162291808)

[Руководство пользователя 6](#_Toc162291809)

[Описание программной реализации 7](#_Toc162291810)

[Подтверждение корректности 9](#_Toc162291811)

[Результаты экспериментов 10](#_Toc162291812)

[Заключение 11](#_Toc162291813)

[Список литературы 12](#_Toc162291814)

[Приложение 13](#_Toc162291815)

Постановка задачи

1. Реализовать функции суммирования рядов Маклорена тремя различными способами: прямое суммирование, попарное суммирование и обратное суммирование. Каждая из функций принимает аргумент, нулевой член, указатель на функцию для вычисления следующего множителя, на который необходимо домножить текущий член, чтобы получить следующий, а также количество слагаемых. Также в каждую из функций был добавлен указатель на функцию, преобразующую аргумент таким образом, чтобы данный ряд Маклорена сходился для любого значения аргумента.

2. С помощью данных функций реализовать функции основных математических функций (синуса, косинуса, экспоненты и натурального логарифма);

3. Написать программу с понятным интерфейсом, позволяющую получить значение функции по заданному аргументу;

4. Проверить программу на корректность;

5. Исходный код программ загрузить в указанный репозиторий;

6. Провести ряд экспериментов, в ходе которых выяснить, какой из способов суммирования наиболее точно вычисляет значение заданной функции, сравнивая полученные значения с значениями тех же функций, реализованных в библиотеке cmath.

Метод решения

В данной главе будут рассмотрены ряды, используемые для аппроксимации и методы суммирования [1].

Ряды Маклорена:

Ряды в общем виде:

В данной лабораторной работы было реализовано четыре метода суммирования:

1. Прямая сумма.

Сумма вычисляется последовательно, начиная с 0 члена ряда.

1. Обратная сумма.

Сумма вычисляется последовательно в обратном порядке, начиная с n-го члена.

1. Прямая попарная сумма.

Складываются и члены суммы и добавляются в общую сумму. Вычисляется с начала ряда.

1. Обратная попарная сумма.

Складываются и члены суммы и добавляются в общую сумму. Вычисляется с конца ряда.

Руководство пользователя

При запуске программы вам необходимо ввести два числа: первое - это точка, в которой вы хотите посчитать значение синуса, косинуса, экспоненты, натурального логарифма (1+х), второе - это количество членов ряда. (Напоминание: чем больше членов ряда, тем точнее идет подсчет функции)

Далее выбранная функция будет тестироваться, а результаты теста будут выводиться на экран сразу в нескольких вариантах подсчета: прямой, обратной и попарной суммы.

Описание программной реализации

В данной главе будут рассмотрены программные реализации алгоритмов суммирования.

Первый из них - прямая сумма. Сигнатура этой функции выглядит следующим образом.

**float sum\_function(((float,а0, float x, float(\*next)((float, int )int n);**

Реализованная функция вычисляет сумму любого ряда, достаточно передать первый член (a0), аргумент (x), указатель на функцию получения следующего члена ряда (из текущего и его номера) и количество элементов в ряде (n).

Например, для будет равен . Для каждой из функций можно получить свой .В реализации эти функции названы next\_sin, next\_cos, next\_exp, next\_ln. Реализацию этих и других функций можно посмотреть в приложении.

Вернёмся к функции прямого суммирования, её реализация крайне проста. В цикле от до вычисляются элементы ряда, с помощью вышеописанного метода, и добавляются в результирующую сумму.

Следующая на очереди - обратная сумма, её сигнатура выглядит следующим образом.

**float sumreverse\_function(float a0, float x, float(\*next)(float, int), int n);**

Сигнатура абсолютна аналогична функции рассмотренной выше.

Реализация данной функции идейна похожа на предыдущую, только в этот раз мы вычисляем последний член ряда и начиная с него выполняем суммирование, находя с его помощью предшествующие члены. Для начала мы создаем массив и заполняем его вычисленными значениями, далее мы разворачиваем его и начинаем складывать.

Преимущество данного метода суммирования заключается в том, что он старается минимизировать ошибку, возникающую при сложении вещественных чисел, чьи порядки сильно различаются.

Следующий метод - попарная сумма.

**float pairwisesum\_function(float a0, float x, float(\*next)(float, int), int n);**

Этот метод по сути является усовершенствованием метода прямой суммы. Его задача также минимизировать ошибку, возникающую при сложении двух вещественных чисел сильно различного порядка.

Идея проста, сначала сложить их друг с другом и только после этого добавить их в общую сумму.

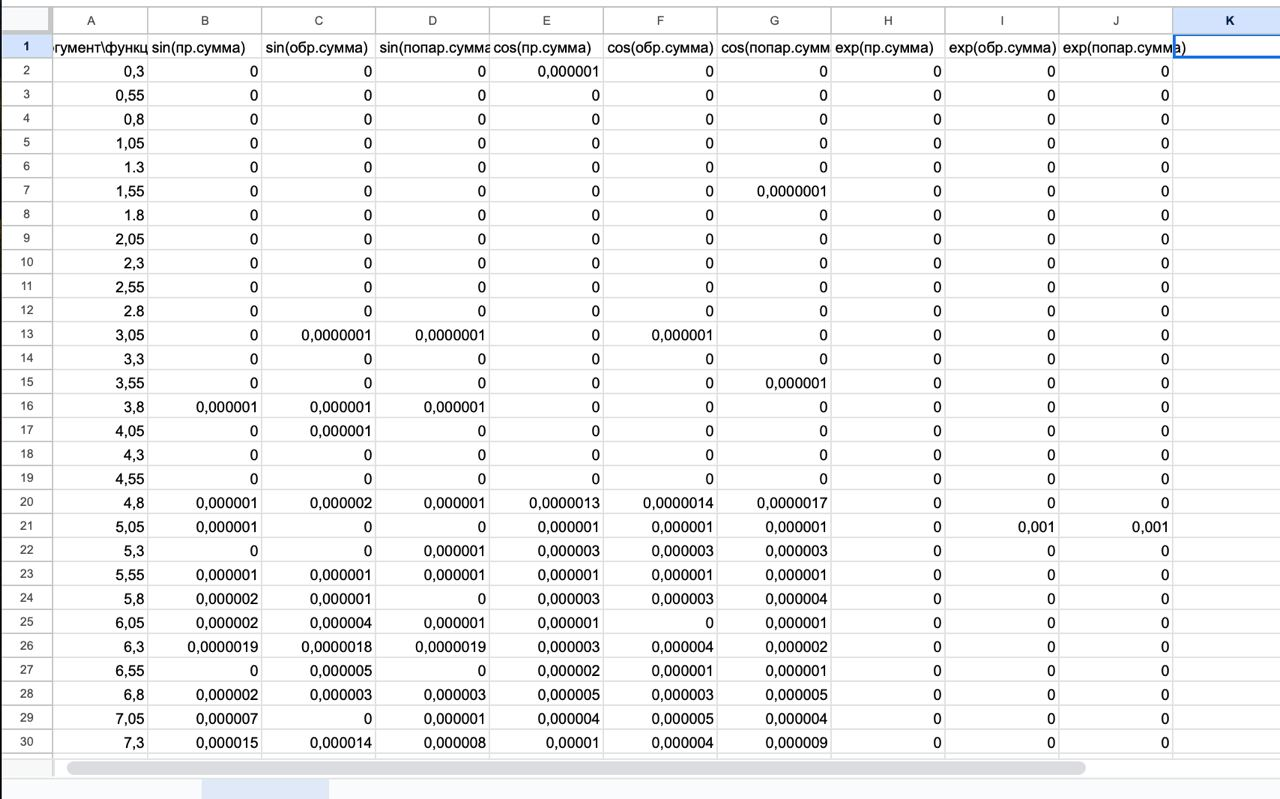
Подтверждение корректности

В данной главе будет рассказано, какие методы использовались для проверки корректности реализованных алгоритмов.

Реализованные математические функции сравнивались с аналогичными из стандартной библиотеки. Вблизи точки , как и ожидалось, значения этих функций в точности совпадают. Можно сделать выводы, что алгоритмы реализован корректно.

Результаты экспериментов

Для данных экспериментов была составлена таблица, в которой указана погрешность для аргументов всех функций (кроме натурального логарифма) от 0 до 8 с шагом 1/4.



По данной таблице можно сделать точный вывод о том, что при увеличении аргумента растет погрешность для любого метода суммирования. Для синуса наиболее точным оказалось попарно суммирование, так как в основном там погрешность в 0,000001. Для косинуса трудно сказать какой именно метод суммирования точен, но скорее всего это обратная сумма. Для экспоненты наиболее плохим методом является прямая сумма.

Отдельно разберем функцию ln(1+x). Для нее можно сделать особенный вывод о том, что при увеличении количества членов ряда погрешность растет(чего не скажешь для других функций), а при переходе за 30 и вовсе не считает. При экспериментах было трудно сделать вывод о том, какой метод суммирования точнее, поэтому функция не была включена в таблицу.

Заключение

Подводя итоги и взяв за основу результаты наших экспериментов таблицу, указанную выше, можно сделать вывод о том, что при увеличении членов ряда функции подсчитывают наиболее точно(кроме ln). Также мы можем сказать о том, что для синуса наиболее точным оказался попарный метод суммирования, а для косинуса - обратный. Для натурального логарифма выявить наиболее точный метод не удалось.

Список литературы

1. http://mathprofi.ru/razlozhenie\_funkcij\_v\_stepennye\_ryady.html

Приложение

#include <iostream>

#include <cmath>

float next\_sin(float x, int i) {

return -x \* x / ((2\*i + 1)\*2\*i);

}

float next\_cos(float x, int i){

return -x \* x / ((2\*i)\*(2\*i-1));

}

float next\_exp(float x, int i){

return x / i;

}

float next\_ln ( float x, int i){

return (-x \* i) / (i + 1);

}

float sum\_function(float a0, float x, float(\*next)(float, int), int n){

float res, ai;

res = ai = a0;

for (int i = 1; i <= n; i++) {

ai \*= next(x, i);

res += ai;

}

return res;

}

float sumreverse\_function(float a0, float x, float(\*next)(float, int), int n) {

float res, ai;

int size = n + 1;

float\* arr = new float[size];

res = 0;

ai = a0;

arr[0] = a0;

for (int i = 1; i < size; i++) {

ai \*= next(x, i);

arr[i] = ai;

}

for (int i = 0; i < size / 2; i++) {

float tmp = arr[i];

arr[i] = arr[size - i - 1];

arr[size - i - 1] = tmp;

}

for (int i = 0; i < size; i++) {

res += arr[i];

}

delete[] arr;

return res;

}

float pairwisesum\_function(float a0, float x, float(\*next)(float, int), int n) {

float res, ai, tmp;

res = ai = a0;

for (int i = 1; i <= n; i++) {

ai \*= next(x, i);

tmp = ai;

if (i + 1 <= n)

{

i++;

ai \*= next(x, i);

tmp += ai;

}

res += tmp;

}

return res;

}

int main() {

setlocale(LC\_ALL, "Rus");

float x;

int n;

std::cout << "Введите значение x: ";

std::cin >> x;

std::cout << "Введите количество итераций n: ";

std::cin >> n;

std::cout << "Значения различных функций в точке x = " << x << ":" << std::endl;

float sin\_value = sum\_function(x, x, next\_sin, n);

std::cout << "Синус: " << sin\_value << std::endl;

float cos\_value = sum\_function(1.0, x, next\_cos, n);

std::cout << "Косинус: " << cos\_value << std::endl;

float exp\_value = sum\_function(1.0, x, next\_exp, n);

std::cout << "Экспонента: " << exp\_value << std::endl;

float ln\_value = sum\_function(x, x, next\_ln, n);

std::cout << "Логарифм: " << ln\_value << std::endl;

float reverse\_sin\_value = sumreverse\_function(x, x, next\_sin, n);

std::cout << "Сумма обратного ряда синусов: " << reverse\_sin\_value << std::endl;

float reverse\_cos\_value = sumreverse\_function(1.0, x, next\_cos, n);

std::cout << "Сумма обратного ряда косинусов: " << reverse\_cos\_value << std::endl;

float reverse\_exp\_value = sumreverse\_function(1.0, x, next\_exp, n);

std::cout << "Сумма обратного ряда экспоненты : " << reverse\_exp\_value << std::endl;

float reverse\_ln\_value = sumreverse\_function(x, x, next\_ln, n);

std::cout << "Сумма обратного ряда логарифма: " << reverse\_ln\_value << std::endl;

float pairwise\_exp\_value = pairwisesum\_function(1.0, x, next\_exp, n);

std::cout << "Попарная сумма экспоненциального ряда: " << pairwise\_exp\_value << std::endl;

float pairwise\_sin\_value = pairwisesum\_function(x, x, next\_sin, n);

std::cout << "Попарная сумма ряда синусов: " << pairwise\_sin\_value << std::endl;

float pairwise\_cos\_value = pairwisesum\_function(1.0, x, next\_cos, n);

std::cout << "Попарная сумма ряда косинусов: " << pairwise\_cos\_value << std::endl;

float pairwise\_ln\_value = pairwisesum\_function(x, x, next\_ln, n);

std::cout << "Попарная сумма ряда логарифма: " << pairwise\_ln\_value << std::endl;

std::cout << sin(x) <<std::endl;

std::cout << cos (x)<<std::endl;

std::cout << exp (x)<<std::endl;

std::cout << log(1 + x)<<std::endl;

return 0;

}