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**

**«Суммирование рядов математических функций»**

**Выполнил**:

студент 3822Б1ПМ1

Кубышкин В. В.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2023

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 5](#_Toc26962564)

[Описание программной реализации 6](#_Toc26962565)

[Подтверждение корректности 7](#_Toc26962566)

[Результаты экспериментов 8](#_Toc26962567)

[Заключение 9](#_Toc26962568)

[Приложение 10](#_Toc26962569)

**Постановка задачи**

1. Реализовать функции sin x, cos x, e^x и ln(1+x) при помощи разложения в ряды Маклорена с указателем на функцию.
2. Реализовать обратную сумму данных функций в ряде Маклорена с указателем на функцию.
3. Многофайловость программы.
4. Сравнить написанные функции с библиотечными и найти погрешность.
5. Написать отчет о проделанной работе.

# Метод решения

1. Разложим функции sin x, cos x, e^x и ln(1+x) в ряд Маклорена и найдем зависимость i-го элемента и i+1- го элемента, так же заметим, что sin x, cos x, e^x – сходятся везде, а функция ln(1+x) имеет ограничения |x|<1. Затем реализуем прямую сумму данных функций. Функции обратной суммы выглядят почти так же, но ai элемент находится по-другому.
2. Мы реализуем функцию, которая считает каждый ai элемент и суммирует их, в итоге выводят общую сумму. Здесь нам и понадобится указатель на функцию. Далее делаем тоже самое, но для обратной функции, но сначала мы должны найти последний ненулевой элемент и далее суммировать его с предыдущими. Для обратной суммы воспользуемся указателем на прямую и обратную функцию sin x, cos x, e^x или ln(1+x).
3. Все эти функции мы пишем в файле core.cpp, далее создаем файл core.h с хедерами данных функций и в файле laba.cpp начинаем писать функцию main, где реализуем мини интерфейс для нашего проекта.
4. При помощи библиотеки math.h вызываем функции sin x, cos x, e^x и ln(1+x) и считаем погрешность, сравнивая их с нами вышенаписанными прямыми и обратными ходами. Пишем отчет о проделанной работе.

# Руководство пользователя

1. При запуске программы высвечивается надпись “Choose the sum: 1. Direct 2. Reverse”, где пользователь должен ввести число, которое соответствует прямому или обратному ходу.

2. Далее высвечивается надпись “Choose the fun: 1. Sin x 2. Cos x 3. Ln(1+x) 4. E^x”, где пользователь должен выбрать что и какую функцию он хочет посчитать и нажать enter.

3. Далее высвечивается надпись “Enter x and n:”, где пользователь должен ввести аргумент x и количество слагаемых, которое он хочет просуммировать и нажать enter.

4. После данных действий программа вычисляет ответ и выводит его на экран после слова “Result:”.



Рис.1 Выбор суммы



Рис.2 Выбор функции



Рис.3 Ввод аргументов



Рис.4 Результат

# Описание программной реализации

Для реализации данной программы мне пришлось воспользоваться такими директивам, как:

#include "stdio.h" – для использования основных функций языка C;

#include "math.h" – для использования функции модуля и функций sin x, cos x, e^x и ln(1+x);

#include “core.h” – для использования ранее написанных функций из файла core.h.

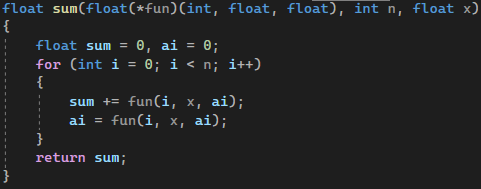


Рис.5 Функция суммирования

Изображение выглядит как текст

Автоматически созданное описание

Рис.6 Функция обратного суммирования

# Подтверждение корректности

Для подтверждения корректности в программе я воспользовался библиотечными функциями sin x. cos x, ln(1+x) и e^x, которые взял из библиотеки math.h и далее воспользовался формулой погрешности: |fun – sum| , где fun – это библиотечные sin x. cos x, ln(1+x) и e^x, а sum – это написанные мною эти же функции. В свою программу я дописал данную формулу и сделал так, чтобы погрешность высчитывалась и показывалась в конце программы:



Рис.7 Погрешность

Таким образом я могу протестировать любую функцию, при любом ее аргументе и сравнить ее точность с библиотечной.

# Результаты экспериментов

Чтобы сравнить, какой же ход все-таки лучше, прямой или обратный, я решил составить такую таблицу, чтобы наглядно убедиться:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Sin x | Rev\_sin x | Cos x | Rev\_cos x | Ln(1+x) | Rev\_ln (1+x) | E^x | Rev\_e^x |
| 1 | 0 | 0 | 0 | 0 | 0.000009 | 1.3863 | 0 | 0.000002 |
| 2 | 0 | 0 | 0 | 0 |  |  | 0 | 0.000005 |
| 3 | 0 | 0 | 0 | 0 |  |  | 0.000002 | 0.00001 |
| 5 | 0 | 0.000001 | 0 | 0.000001 |  |  | 0 | 0.000137 |
| 10 | 0.00002 | 0.000001 | 0.000198 | 0.000211 |  |  | 0 | 0.025391 |
| 15 | 0.000777 | 0.00426 | 0.009402 | 0.008127 |  |  | 0 | 0.5 |
| 20 | 1.3 | 2.387 | 3.717 | 3.5167 |  |  | 32 | 800 |

Табл.1 Погрешности функций

В самой верхней строке написаны названия функций, в самом левом столбце написаны аргументы x, а число слагаемых всегда будет 100000, так как все равно на каком-то будет занулятся. На пересечении строк и столбцов будет написана погрешность.

Из-за того, что в натуральный логарифм можно подставлять только лишь числа меньшие или равные 1 по модулю, то произведем только 1 проверку при максимальном значении агрумента.

# Заключение

Итак, поставленная в начале работы задача выполнена: написана программа для подсчета данных функций, использована многофайловость, указатели на функции, а так же проверка функции и подсчет их погрешности. Не вооруженным взглядом из Табл.1 видно, что функции, написанные прямым ходом оказались точнее при любых аргументах, что следует из того, что в какой момент элементы просто зануляются и точность становится хуже.

# Приложение

Core.cpp:

#include <stdio.h>

#include <math.h>

float pr\_sin(int n, float x, float ai) //вычисляет синус (прямая сумма)

{

float sum;

if (n == 0)

{

return x;

}

sum = ((-1) \* (x \* x) / (2 \* n \* (2 \* n + 1))) \* ai;

return sum;

}

float ob\_sin(int n, float x, float ai) //вычисляет синус (обратная сумма)

{

float sum;

sum = (2 \* n \* (2 \* n + 1)) / ((-1) \* (x \* x)) \* ai;

return sum;

}

float pr\_cos(int n, float x, float ai) //вычисляет косинус (прямая сумма)

{

float sum;

if (n == 0)

{

return 1;

}

sum = ((-1) \* (x \* x) / (2 \* n \* (2 \* n - 1))) \* ai;

return sum;

}

float ob\_cos(int n, float x, float ai) //вычисляет косинус (обратная сумма)

{

float sum;

sum = (2 \* n \* (2 \* n - 1)) / ((-1) \* (x \* x)) \* ai;

return sum;

}

float pr\_ln(int n, float x, float ai) //вычисляет ln(1+x) (прямая сумма)

{

float sum;

if (n == 0)

{

return x;

}

sum = ((-1) \* (x \* n) / (n + 1)) \* ai;

return sum;

}

float ob\_ln(int n, float x, float ai) //вычисляет ln(1+x) (обратная сумма)

{

float sum;

if (n == 0)

{

return 0;

}

sum = ((-1) \* (n + 1) / (x \* n)) \* ai;

return sum;

}

float pr\_ex(int n, float x, float ai)//вычисляет e^x(прямая сумма)

{

float sum;

if (n == 0)

{

return 1;

}

sum = (x / n) \* ai;

return sum;

}

float ob\_ex(int n, float x, float ai) //вычисляет e^x(обратная сумма)

{

float sum;

sum = (n / x) \* ai;

return sum;

}

float sum(float(\*fun)(int, float, float), int n, float x)

{

float sum = 0, ai = 0;

for (int i = 0; i < n; i++)

{

sum += fun(i, x, ai);

ai = fun(i, x, ai);

}

return sum;

}

float reversesum(float(\*fun1)(int, float, float), float(\*fun2)(int, float, float), int n, float x)

{

float sum = 0.0, ai = 0.0, aj = 0.0;

int k = 0;

for (int i = 0; i < n; i++)

{

ai = fun2(i, x, ai);

if ((fabs(ai) - (3.4e-34)) < 0)

{

ai = aj;

k = i;

break;

}

aj = ai;

k = i;

}

sum += ai;

for (int i = k-1; i >= 0; i--)

{

sum += fun1(i, x, ai);

ai = fun1(i, x, ai);

}

return sum;

}

Core.h:

#pragma once

float pr\_sin(int n, float x, float ai);

float ob\_sin(int n, float x, float ai);

float pr\_cos(int n, float x, float ai);

float ob\_cos(int n, float x, float ai);

float pr\_ln(int n, float x, float ai);

float ob\_ln(int n, float x, float ai);

float pr\_ex(int n, float x, float ai);

float ob\_ex(int n, float x, float ai);

float sum(float(\*fun)(int, float, float), int n, float x);

float reversesum(float(\*fun1)(int, float, float), float(\*fun2)(int, float, float), int n, float x);

laba.cpp:

#include <stdio.h>

#include <math.h>

#include "core.h"

int main()

{

int a, b, n;

float res, x;

float funres;

printf("Choose the sum: 1.direct 2.reverse\n");

scanf\_s("%d", &b);

if (b == 1)

{

printf("Choose the fun: 1.sinx 2. cosx 3. ln(1+x) 4. e^x\n");

scanf\_s("%d", &a);

printf("Enter x and n:\n");

scanf\_s("%f %d", &x, &n);

if (a == 1)

{

res = sum(pr\_sin, n, x);

funres = (float)sin(x);

printf("Result: %f\n", res);

printf("Error: %f", fabs(funres - res));

}

if (a == 2)

{

res = sum(pr\_cos, n, x);

funres = (float)cos(x);

printf("Result: %f\n", res);

printf("Error: %f", fabs(funres - res));

}

if (a == 3)

{

res = sum(pr\_ln, n, x);

funres = (float)log(x+1);

printf("Result: %f\n", res);

printf("Error: %f", fabs(funres - res));

}

if (a == 4)

{

res = sum(pr\_ex, n, x);

funres = (float)exp(x);

printf("Result: %f\n", res);

printf("Error: %f", fabs(funres - res));

}

}

if (b == 2)

{

printf("Choose the fun: 1.sinx 2. cosx 3. ln(1+x) 4. e^x\n");

scanf\_s("%d", &a);

printf("Enter x and n:\n");

scanf\_s("%f %d", &x, &n);

if (a == 1)

{

res = reversesum(ob\_sin, pr\_sin, n, x);

funres = (float)sin(x);

printf("Result: %f\n", res);

printf("Error: %f", fabs(funres - res));

}

if (a == 2)

{

res = reversesum(ob\_cos, pr\_cos, n, x);

funres = (float)cos(x);

printf("Result: %f\n", res);

printf("Error: %f", fabs(funres - res));

}

if (a == 3)

{

res = reversesum(ob\_ln, pr\_ln, n, x);

funres = (float)log(x + 1);

printf("Result: %f\n", res);

printf("Error: %f", fabs(funres - res));

}

if (a == 4)

{

res = reversesum(ob\_ex, pr\_ex, n, x);

funres = (float)exp(x);

printf("Result: %f\n", res);

printf("Error: %f", fabs(funres - res));

}

}

return 0;

}