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
Лазарев. Н.П.

Проверила:
преподаватель каф. ВВСП,
Бусько П.В.

Нижний Новгород
2025

Содержание

Постановка задачи

Вычислить погрешность в рядах Маклорена для функций sin(x), cos(x), e^x, ln(x+1) до n-го элемента при различных n. При расчётах сравнить три метода суммирования чисел с плавающей запятой: прямой, обратный и попарный.

Метод решения

Разработаны функции для определения n-го члена разложения sin(x), cos(x), e^x, ln(x+1) с использованием предыдущего члена и значения x. Каждая из этих функций передаётся в одну из трёх процедур суммирования, возвращающих итоговую сумму ряда. Результатом работы является абсолютная разница между полученным значением и результатом стандартных математических функций.

Руководство пользователя

Пользователь задаёт значение X и выбирает метод суммирования. Программа выводит абсолютное отклонение между вычисленным рядом и эталонным значением для $n = 1,2,4,8 \ldots$ и т.д.

Для логарифма ln(x+1) погрешность отображается только при $-1 \leq X \leq 1$

```
Select the X value that you would like to approximate Sin, Cos, Exponent and Log with via the Taylor series: -0.5
Select the summing method you would like to use for the approximation:

1. 1 to N
2. N to 1.
3. Pairs
2


N = 1: Sin diff = 0.0205744613957970, Cos diff = 0.1224174381096272, Exp diff = 0.3934693402873666, Log diff = 0.1931471805599453

N = 2: Sin diff = 0.0002588719375363, Cos diff = 0.0025825618903728, Exp diff = 0.1065306597126334, Log diff = 0.0681471805599453

N = 4: Sin diff = 0.0000000053700760, Cos diff = 0.0000000966125950, Exp diff = 0.0023639930459667, Log diff = 0.0108555138932787

N = 8: Sin diff = 0.0000000000000000, Cos diff = 0.0000000000000000, Exp diff = 0.0000000917562842, Log diff = 0.0003969945480405

N = 16: Sin diff = 0.0000000000000000, Cos diff = 0.0000000000000000, Exp diff = 0.0000000000000000, Log diff = 0.0000008522949418

N = 32: Sin diff = 0.0000000000000000, Cos diff = 0.0000000000000000, Exp diff = 0.0000000000000000, Log diff = 0.0000000000068588

N = 64: Sin diff = 0.0000000000000000, Cos diff = 0.0000000000000000, Exp diff = 0.0000000000000000, Log diff = 0.0000000000000000

N = 128: Sin diff = 0.0000000000000000, Cos diff = 0.0000000000000000, Exp diff = 0.0000000000000000, Log diff = 0.0000000000000000

N = 256: Sin diff = 0.0000000000000000, Cos diff = 0.0000000000000000, Exp diff = 0.0000000000000000, Log diff = 0.0000000000000000
```

Описание программной реализации

Проект mathFuncs.cpp разработан в среде Visual Studio с компилятором MSVC.

- Функции NthTaylorSin, NthTaylorCos, NthTaylorExp, NthTaylorLog вычисляют n-й член ряда для соответствующих функций.

- OneToN, NToOne, PairSum реализуют прямое, обратное и попарное суммирование.

- PairSumFunc — вспомогательная функция для рекурсии в PairSum.

- main обрабатывает ввод X и метода суммирования, выводит погрешности.

Подтверждение корректности

Для верификации выводятся эталонные значения стандартных функций и отклонения от них.

Результаты экспериментов

По данным экспериментов видно, что:

- Для значений $N \geq 128$ все функции "сходятся" к одному и тому же значению, и ошибка перестаёт убывать.

Таблица средних ошибок каждого алгоритма при $N \geq 128$ и разных X

X	Обычная	Обратная	Попарная	Вывод
$\lvert X \rvert \leq 0.5$	S – 0, C – 1e-15, E – 2e-15, L – 1e-15	S – 0, C – 0, E – 0, L – 1e-15	S – 0, C – 0, E – 0, L – 1e-15	Обратный и попарный методы точнее
$\lvert X \rvert \leq 1$	S – 1e-15, C – 1e-15, E – 3e-15, L – 2e-15	S – 0, C – 0, E – 0, L – 1e-15	S – 0, C – 0, E – 1e-15, L – 1e-15	Обратный лучше, попарный превосходит прямой
$\lvert X \rvert \leq 5$	S – 5e-15, C – 5e-15, E – 5e-14	S – 7e-15, C – 3e-15, E – 2e-14	S – 7e-15, C – 6e-15, E – 4e-14	Обычная для синуса, обратная для остальных.
$\lvert X \rvert \leq 10$	S – 3e-13, C – 1e-12, E – 9e-12	S – 2e-13, C – 5e-13, E – 5e-12	S – 2e-13, C – 5e-13, E – 7e-12	Обратная точнее, попарная лучше обычной
$\lvert X \rvert \leq 15$	S – 3e-11, C – 2e-11, E – 1e-9	S – 5e-11, C – 6e-11, E – 3e-9	S – 4e-11, C – 7e-11, E – 4e-9	Обычная точнее для всех функций, попарная лучше обратной для синуса и косинуса
$\lvert X \rvert \leq 20$	S – 3e-8, C – 5e-9, E – 9e-7	S – 5e-9, C – 5e-10, E –2e-7	S – 7e-8, C – 4e-8, E –1e-6	Обратная точнее, обычная лучше попарной
$\lvert X \rvert \leq 30$	S – 4e-5, C – 8e-5, E – 6e-3	S – 3e-6, C – 9e-6, E –4e-3	S – 8e-6, C – 1e-5, E –5e-3	Обратная точнее, попарная лучше обычной

Вывод: Обратный метод чаще обеспечивает наибольшую точность. Для sin(x) иногда предпочтителен прямой метод. Попарное суммирование — компромисс между двумя подходами.

Заключение

Разработана программа для вычисления погрешностей рядов Маклорена функций $sin(x)$, $cos(x)$, e^x, $ln(x+1)$ при различных n. Проведено сравнение точности прямого, обратного и попарного методов суммирования.

Приложение

```cpp
#include "pch.h"
#include <cstdio>
#include <cstdlib>
#include <cmath>

class TaylorSeries {
public:
	static double nthSin(double x, double prev, int n) {
		return n == 0 ? x : (-(prev * x * x) / ((2 * n)*(2 * n + 1)));
	}

	static double nthCos(double x, double prev, int n) {
		return n == 0 ? 1 : (-(prev * x * x) / ((2 * n) * (2 * n - 1)));
	}

	static double nthExp(double x, double prev, int n) {
		return n == 0 ? 1 : prev * x / n;
	}

	static double nthLog(double x, double prev, int n) {
		return n == 0 ? x : -prev * x * n / (n + 1);
	}
};

class SummationStrategy {
public:
	virtual ~SummationStrategy() = default;
	virtual double compute(int n, double(*termFunc)(double, double, int), double x) = 0;
};

class OneToNStrategy : public SummationStrategy {
public:
	double compute(int n, double(*termFunc)(double, double, int), double x) override {
		double ret = 0, val = 0;
		for (int i = 0; i < n; i++) {
			val = termFunc(x, val, i);
			ret += val;
		}
		return ret;
	}
};

class NToOneStrategy : public SummationStrategy {
public:
	double compute(int n, double(*termFunc)(double, double, int), double x) override {
		double ret = 0;
		double* arr = (double*)malloc(n * sizeof(double));
		arr[0] = termFunc(x, 0, 0);
		for (int i = 1; i < n; i++) {
			arr[i] = termFunc(x, arr[i - 1], i);
		}
		for (int i = n - 1; i >= 0; i--) {
			ret += arr[i];
		}
		free(arr);
		return ret;
	}
};

class PairSumStrategy : public SummationStrategy {
private:
	double pairSumFunc(double* arr, int l, int r) {
		double ret = 0;
		if (r - l <= 5) {
```

```cpp
				for (int i = r - 1; i >= l; i--) ret += arr[i];
				return ret;
			}
			return pairSumFunc(arr, l, (l + r) / 2) + pairSumFunc(arr, (l + r) / 2, r);
		}

public:
		double compute(int n, double(*termFunc)(double, double, int), double x) override {
			double* arr = (double*)malloc(n * sizeof(double));
			arr[0] = termFunc(x, 0, 0);
			for (int i = 1; i < n; i++) {
				arr[i] = termFunc(x, arr[i - 1], i);
			}
			double ret = pairSumFunc(arr, 0, n);
			free(arr);
			return ret;
		}
};
```