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**

**«Подсчет рядов Маклорена»**

**Выполнил**:

студентка группы 3821Б1ПМ2

Храмова А.А.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

**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)

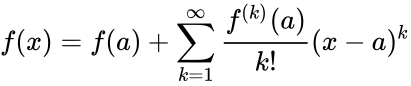
[Приложение 10](#_Toc26962569)

# Постановка задачи

Задачей данной лабораторной работы является реализация программы, которая выполняет подсчет рядов Маклорена: sin(x), cos(x), e^x, ln(1+x). Для выполнения этого задания используется 3 алгоритма суммирования: прямое, обратное и попарное суммирование. После чего необходимо проанализировать ошибки суммирования путем подсчета точности с помощью абсолютной и относительной погрешностей, поставить эксперименты с целью выявить самый точный способ подсчета суммы. Опираясь на собранные данные, сравнить результаты суммирования с библиотечными стандартными функциями.

# Метод решения

Данное задание выполняем с помощью частного случая для рядов Тейлора – рядов Маклорена (при = 0). Формула ряда Тейлора получается из полинома степени n, который представляется в виде суммы по k = 0 до n коэффициентов, умноженных на . Последовательно берем 1-ю, 2-ю и так далее производные и подставляем точку вместо x. Получаем, что коэффициенты полинома выражаются через производную по номеру коэффициента, взятую в точке . Отсюда получаем формулу Тейлора алгебраического полинома



Эти формулы нужны для того, чтобы заменить данные нам функции на полиномы, которые будут с каждым слагаемым все точнее приближаться к исходным функциям, этот процесс называется аппроксимацией функции многочленами.

Формулы Маклорена получаются отсюда при

Изображение выглядит как текст

Автоматически созданное описание

Бесконечное число слагаемых посчитать невозможно, тип данных double позволяет нам посчитать 15 знаков.

Для подсчета суммы будем использовать три способа суммирования: прямое суммирование, обратное суммирование и попарное суммирование.

# Руководство пользователя

При запуске программы перед пользователем появляется выбор функции, для которой будет посчитана сумма:

Изображение выглядит как текст

Автоматически созданное описание

Далее запрашивается х:

Изображение выглядит как текст

Автоматически созданное описание

Потом выводится 3 вида суммы и абсолютная и относительная ошибки для каждой из сумм:

Изображение выглядит как текст

Автоматически созданное описание

# Описание программной реализации

void first(double x, int choice\_func, double\* mas) – функция задает первое слагаемое в массиве слагаемых

double next\_member\_sin(double prev, int i, double x) – функция высчитывает последующее слагаемое в массиве слагаемых. На вход подается предыдущий член этой последовательности, по которому и строится следующий.

Аналогичные функции для косинуса, экспоненты и натурального логарифма:

double next\_member\_cos(double prev, int i, double x)

double next\_member\_exp(double prev, int i, double x)

double next\_member\_ln(double prev, int i, double x)

void summf(double\* mas, int n, double \* summ\_pr, double \* summ\_obrat, double \* summ\_twice) – функция суммирования. В этой функции реализованы все 3 метода суммирования: прямое, обратное, попарное. На вход функция принимает указатель на прямую сумму, обратную и попарную (а также указатель на массив слагаемых). В ходе вычисления по данным трем адресам записываются значения этих трех сумм соответственно, после чего при вызове данной функции эти значения возвращаются также через параметры – получаем значения посчитанных сумм.

Прямое суммирование реализовано в функции как обычная сумма n слагаемых.

Обратное суммирование – сумма n слагаемых в обратном цикле, начиная с последнего.

Попарное суммирование – складываем члены последовательности попарно – 1-й со 2-м, 3-й с 4-м и т. д. и новые значения записываем в новую последовательность, а количество слагаемых при этом уменьшается вдвое, так как сумма первых двух слагаемых стала 1-м слагаемых, сумма 3-го и 4-го стала 2-м слагаемым и так далее.

void create\_mas(double\* mas, double x, int n, function next) – функция с помощью функции нахождения следующего элемента суммы создает тот самый массив слагаемых.

# Подтверждение корректности

Корректность исполнения программы проверяется за счет стандартных функций библиотеки <math.h>, а также во время подсчета относительной и абсолютной ошибок, где можно увидеть точность подсчета.

# Результаты экспериментов

Программа считает относительную и абсолютную погрешность вычисления.

Для sin(x):

Изображение выглядит как стол

Автоматически созданное описание

Попарное суммирование самое точное для синуса, на втором месте по точности прямое суммирование, на последнем – обратное.

Для cos(x):

Изображение выглядит как стол

Автоматически созданное описание

Прямое суммирование самое точное для косинуса, на втором месте по точности попарное суммирование, на последнем – обратное.

Для :

Изображение выглядит как стол

Автоматически созданное описание

Обратное суммирование самое точное для экспоненты, на втором месте по точности попарное суммирование, на последнем – прямое.

Для ln(1+x):

Изображение выглядит как стол

Автоматически созданное описание

Прямое суммирование самое точное для натурального логарифма, на втором месте по точности обратное суммирование, на последнем – попарное.

# Заключение

Я реализовала алгоритм подсчета прямой, обратной и попарной суммы для стандартных функций: синуса, косинуса, экспоненты и натурального логарифма. Каждый из методов подсчета имеет свои особенности и поэтому будет целесообразно рассматривать различные алгоритмы, выбирая под конкретную функцию тот алгоритм, который будет наиболее точным. Также был проведен анализ погрешности, что позволило наглядно увидеть различия результата подсчета при использовании различных схем подсчета.

# Приложение

Изображение выглядит как стол

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание