**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 функций с использованием формулы Маклорена и рекуррентных соотношений**»**

**Выполнил:**  
 студент группы 3824Б1ПМ1-2  
 Симонов Д. М.

**Проверила:**

Бусько П. В.

П.

**Нижний Новгород**  
2024

**Содержание**

[**Введение** 3](#_Toc192469385)

[**Постановка задачи** 4](#_Toc192469386)

[**Описание программной реализации** 5](#_Toc192469387)

[**Программа написана на языке C и состоит из следующих частей** 5](#_Toc192469388)

[**Функции вычисления с использованием ряда Маклорена** 5](#_Toc192469389)

[**Функции вычисления с использованием рекуррентных соотношений** 6](#_Toc192469390)

[**Структуры данных** 7](#_Toc192469391)

[**Функции инициализации и хранения данных (смотреть в приложение)** 7](#_Toc192469392)

[**Побочные функции** 7](#_Toc192469393)

[**Результаты экспериментов** 8](#_Toc192469394)

[**Заключение** 10](#_Toc192469395)

[**Литература** 11](#_Toc192469396)

[**Приложение** 12](#_Toc192469397)

# **Введение**

Численное вычисление элементарных функций является важной задачей в математическом моделировании и вычислительной математике. В данной работе исследуется применение ряда Маклорена и рекуррентных соотношений для вычисления функций: sin(x), cos(x), exp(x), ln(1+x). Основное внимание уделено сравнению точности двух подходов при различных значениях.

# **Постановка задачи**

Целью данной работы является исследование сходимости приближенных значений функций, полученных с помощью разложения в ряд Маклорена и рекуррентных соотношений.

Для каждого метода необходимо:

1. Реализовать вычисление функций sin(x), cos(x), exp(x), ln(1+x) , и с использованием разложения в ряд Маклорена.
2. Реализовать вычисление этих же функций с использованием рекуррентных формул.
3. Провести анализ точности вычислений в зависимости от количества членов в разложении.
4. Построить графики зависимости ошибки приближения от для обоих методов.

# **Описание программной реализации**

**Программа написана на языке C и состоит из следующих частей:**

* Вычисление значений функций с использованием ряда Маклорена.
* Вычисление значений функций с использованием рекуррентных соотношений.
* Хранение данных в структуре xn, содержащей количество членов ряда, массив значений x, массив сумм sum и массив ошибок err.
* Функции инициализации и хранения данных:  
   init выделяющая память для массивов значений.

free\_memory очищает память которую выделяет init.  
 init\_station заполняющая массивы начальными значениями вычисленных функций.

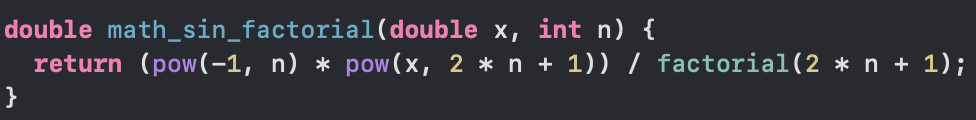
fill заполняет массивы полностью значениями вычисленных функций.

* Побочные функции.

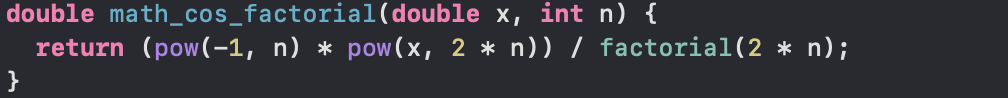
Нахождение факториала числа.

**Функции вычисления с использованием ряда Маклорена:**

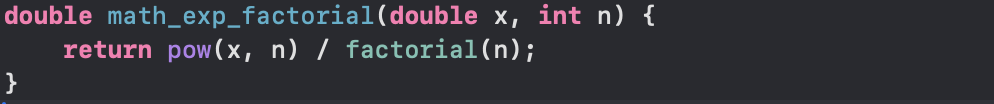
math\_sin\_factorial(x, n);



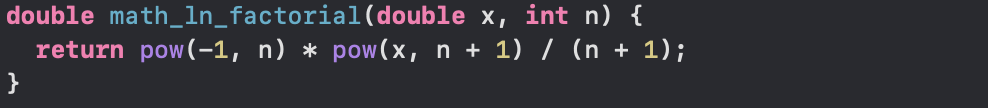
math\_cos\_factorial(x, n);



math\_exp\_factorial(x, n);

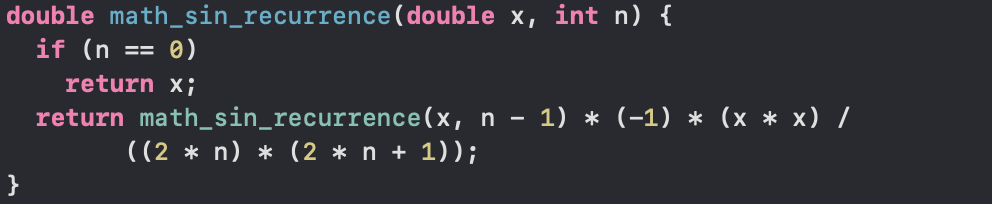


math\_ln\_factorial(x, n);

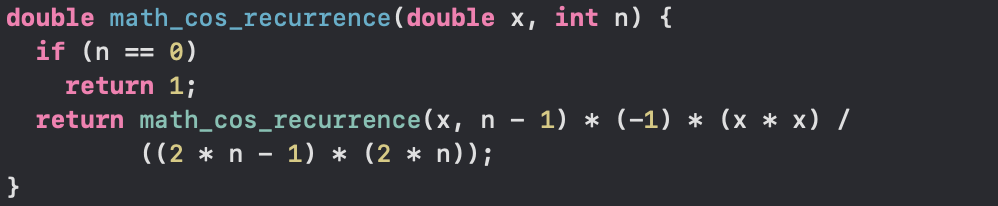


**Функции вычисления с использованием рекуррентных соотношений:**

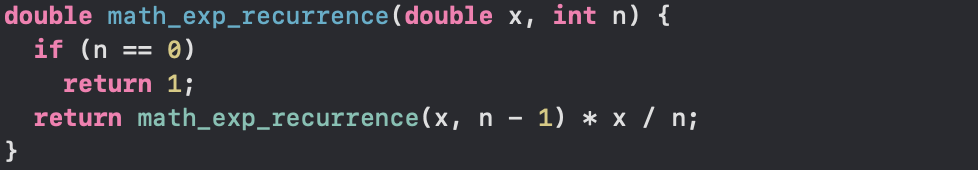
math\_sin\_recurrence(x, n);



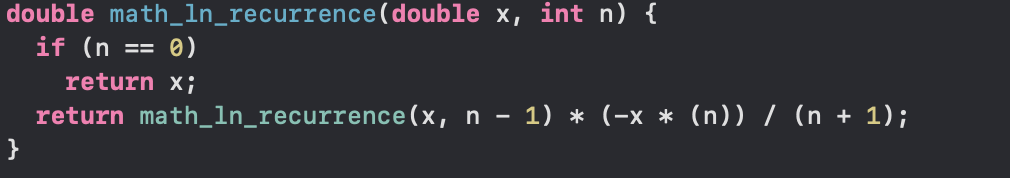
math\_cos\_recurrence(x, n);



math\_exp\_recurrence(x, n);

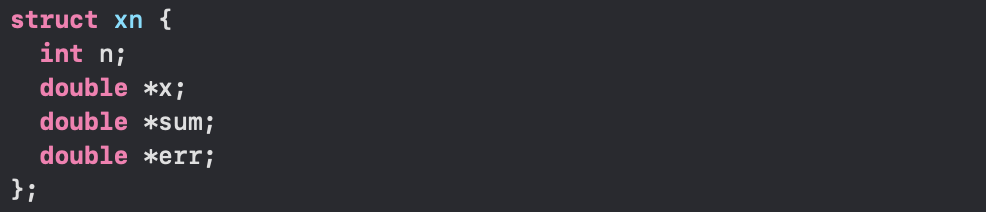


math\_ln\_recurrence(x, n);



**Структуры данных:**

struct xn



**Функции инициализации и хранения данных (смотреть в приложение):**

init(struct xn \*data, int n);

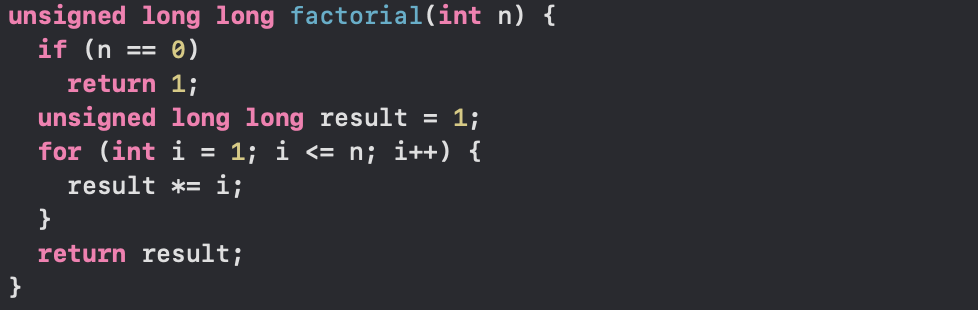
free\_memory(struct xn \*data);

init\_station(struct xn \*data, double (\*func)(double, int), double x);

fill(struct xn \*data, double (\*func)(double, int), double x);

### **Побочные функции**

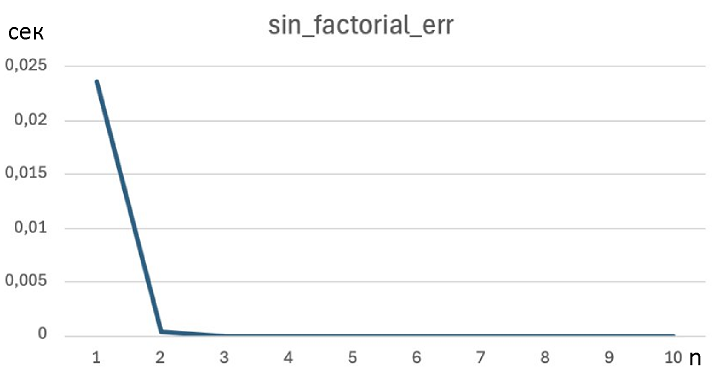
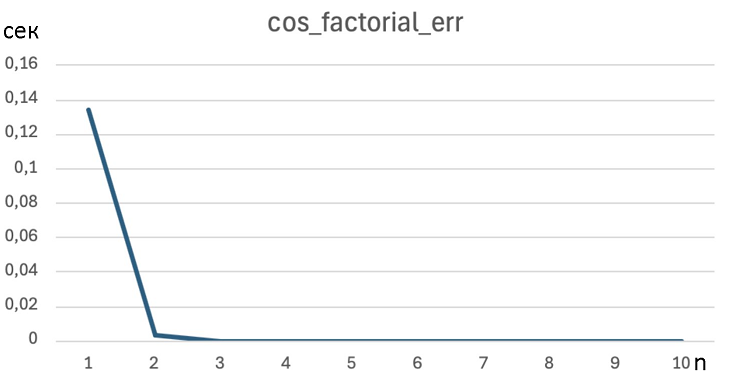
factorial(n);

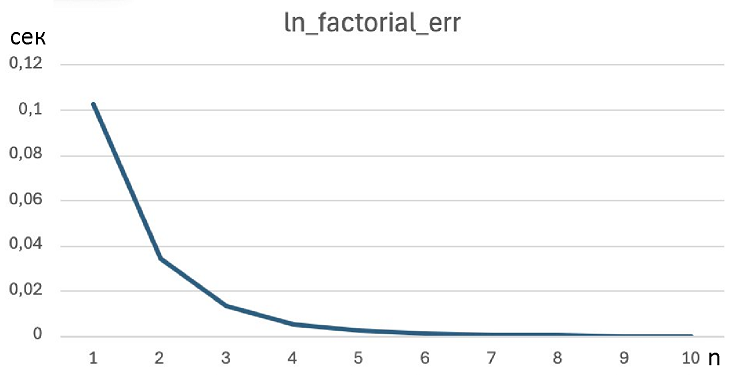
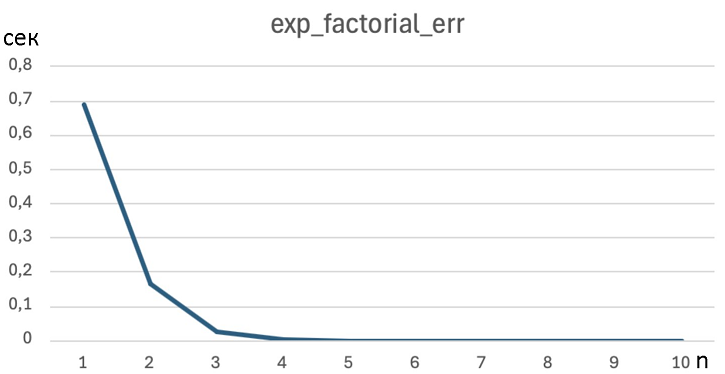


# **Результаты экспериментов**

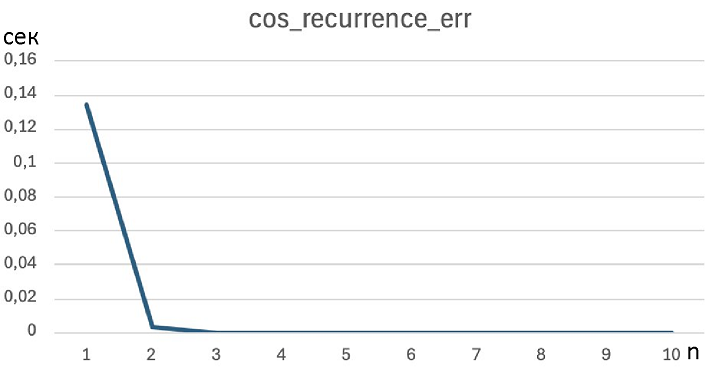
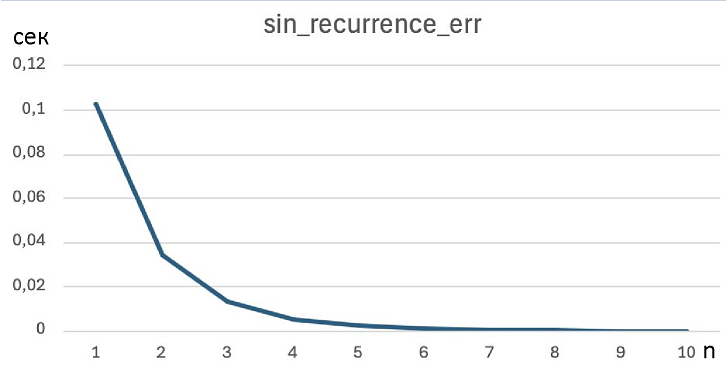
Для каждого метода были проведены вычисления значений функций при различных и проанализирована ошибка относительно стандартных библиотечных функций.

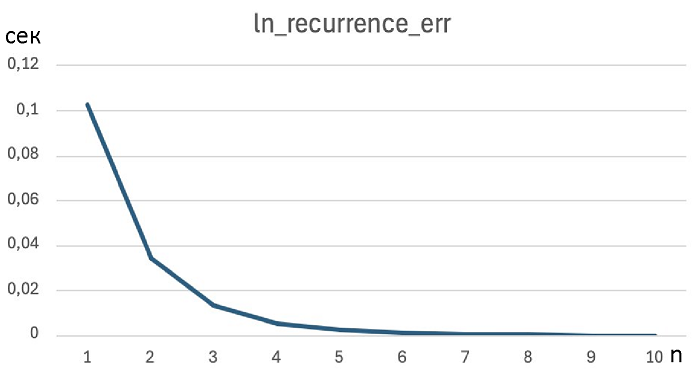
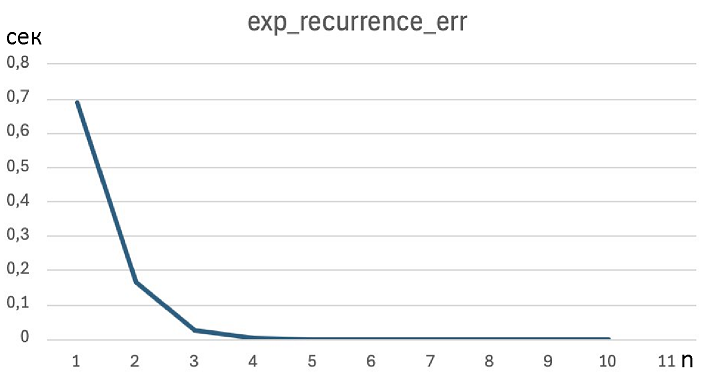
**Графики, демонстрирующие зависимость ошибки от количества членов n, рассчитанные по формулам Маклорена:**

****



**Графики, демонстрирующие зависимость ошибки от количества членов n, рассчитанные при помощи рекуррентных соотношений:**





Построены графики зависимости ошибки от количества членов ряда. В результате экспериментов было установлено:

* При увеличении точность вычислений возрастает.
* Оба метода работают примерно одинаково, но рекурсивный подход требует дополнительного выделения памяти. Поскольку функция вызывается рекурсивно, может произойти переполнение стека при больших n.
* При небольших значениях x ошибка довольно мала при любых n, чем при больших x.
* Для малых методы дают сопоставимые результаты, но при больших возможны ошибки округления.
* Методы на основе рекуррентных формул не имеют явного преимущества перед разложением в ряд Маклорена.
* Когда достигает примерно 10, ошибка становится очень маленькой и не критичной.

# **Заключение**

В ходе работы были реализованы и исследованы два подхода к вычислению элементарных функций. Было показано, что оба метода позволяют получать достаточно точные значения при увеличении числа членов ряда, однако рекуррентные формулы не имеют значительных преимуществ перед разложением в ряд Маклорена. Кроме того, рекуррентный подход требует дополнительной памяти, а при больших существует риск переполнения стека. При n примерно равным 10 ошибка вычислений становится незначительной, что делает оба метода практически равноценными для большинства практических применений.

# **Литература**

1. Кнут Д. Э. Искусство программирования – Издательский дом Вильямс, 2000. – Т. 3.
2. Керниган Б., Ритчи Д. Язык программирования СИ //М.: Финансы и статистика. – 1992.
3. Седжвик Р., Уэйн К. Алгоритмы на C++: краткий курс. 2013.
4. Лафоре Р. Объектно-ориентированное программирование в C++. — М.: Вильямс, 2009. — 864 с.
5. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. — М.: Вильямс, 2018.

# **Приложение**

void init(struct xn \*data, int n) {

data->n = n;

data->x = (double \*)malloc(n \* sizeof(double));

data->sum = (double \*)malloc(n \* sizeof(double));

data->err = (double \*)malloc(n \* sizeof(double));

}

**void** free\_memory(**struct** xn \*data) {

free(data->x);

free(data->sum);

free(data->err);

}

void init\_station(struct xn \*data, double (\*func)(double, int), double x) {

if (func == math\_sin\_factorial) {

data->sum[0] = func(x, 0);

data->err[0] = fabs(sin(x) - data->sum[0]);

}

if (func == math\_cos\_factorial) {

data->sum[0] = func(x, 0);

data->err[0] = fabs(cos(x) - data->sum[0]);

}

if (func == math\_exp\_factorial) {

data->sum[0] = func(x, 0);

data->err[0] = fabs(exp(x) - data->sum[0]);

}

if (func == math\_ln\_factorial) {

data->sum[0] = func(x, 0);

data->err[0] = fabs(log(x + 1) - data->sum[0]);

}

if (func == math\_sin\_recurrence) {

data->sum[0] = math\_sin\_recurrence(x, 0);

data->err[0] = fabs(sin(x) - data->sum[0]);

}

if (func == math\_cos\_recurrence) {

data->sum[0] = math\_cos\_recurrence(x, 0);

data->err[0] = fabs(cos(x) - data->sum[0]);

}

if (func == math\_exp\_recurrence) {

data->sum[0] = math\_exp\_recurrence(x, 0);

data->err[0] = fabs(exp(x) - data->sum[0]);

}

if (func == math\_ln\_recurrence) {

data->sum[0] = math\_ln\_recurrence(x, 0);

data->err[0] = fabs(log(x + 1) - data->sum[0]);

}

}

void fill(struct xn \*data, double (\*func)(double, int), double x) {

for (int n = 1; n < data->n; n++) {

if (func == math\_sin\_factorial) {

data->sum[n] = data->sum[n - 1] + func(x, n);

data->err[n] = fabs(sin(x) - data->sum[n]);

}

if (func == math\_cos\_factorial) {

data->sum[n] = data->sum[n - 1] + func(x, n);

data->err[n] = fabs(cos(x) - data->sum[n]);

}

if (func == math\_exp\_factorial) {

data->sum[n] = data->sum[n - 1] + func(x, n);

data->err[n] = fabs(exp(x) - data->sum[n]);

}

if (func == math\_ln\_factorial) {

data->sum[n] = data->sum[n - 1] + func(x, n);

data->err[n] = fabs(log(x + 1) - data->sum[n]);

}

if (func == math\_sin\_recurrence) {

data->sum[n] = data->sum[n - 1] + func(x, n);

data->err[n] = fabs(sin(x) - data->sum[n]);

}

if (func == math\_cos\_recurrence) {

data->sum[n] = data->sum[n - 1] + func(x, n);

data->err[n] = fabs(cos(x) - data->sum[n]);

}

if (func == math\_exp\_recurrence) {

data->sum[n] = data->sum[n - 1] + func(x, n);

data->err[n] = fabs(exp(x) - data->sum[n]);

}

if (func == math\_ln\_recurrence) {

data->sum[n] = data->sum[n - 1] + func(x, n);

data->err[n] = fabs(log(x + 1) - data->sum[n]);

}

}

}