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

«Программная реализация стандартных математических функций через ряды Тейлора на языке Си»

Выполнил:
студент группы 3824Б1ПМ1
Яковлев Р.О.

Проверила:

Бусько П.В.

Нижний Новгород
2025

Содержание.

Введение

В области математики очень часто используются разные функции, в частности функции синуса, косинуса, логарифма и экспоненты. Если взять первые две функции, то существуют табличные значения, которые позволяют посчитать значение функции в некоторых известных точках. Однако далеко не всегда можно вычислить точное значение этих функций в какой-то случайной точке. Поэтому для облегчения вычислений Брук Тейлор придумал свой способ, который позволяет разложить стандартные математические функции в бесконечную сумму степенных функций. Ряды Тейлора позволяют вычислять значения функций в некоторой точке с необходимой нам точностью. Цель моей исследовательской работы - проверить, действительно ли при помощи рядов Тейлора можно вычислять с нужной нам точностью значения функций синуса, косинуса, логарифма и экспоненты в случайной точке.

Постановка задачи

Основная задача - убедиться в том, что при помощи рядов Тейлора можно вычислять значение стандартных функций с нужной нам точностью относительно эталонных значений из библиотеки “math.h”.

Подзадачи:

1. Создание проекта на языке C с реализацией структуры ряда Тейлора, подсчёта через прямое и обратное суммирование синуса, косинуса, натурального логарифма и экспоненты в некоторой точке через ряды Тейлора
2. Написание алгоритмов для сравнения суммы k-ого количества членов ряда с эталонным значением и вычисления на основе этого k-ой погрешности.
3. Составление графиков для наглядного подтверждения гипотезы о том, что сумма k-ых членов ряда приближается к эталонному значению

Описание программной реализации

Проект состоит из одного файла *“Lab2.cpp”*.

Подключенные библиотеки:

- <iostream>, используется для ввода и вывода данных в консольном приложении;
- <math.h>, используется для подсчёта эталонного значения функций синуса, косинуса, логарифма и экспоненты, а также для возведения в степень, нахождения модуля и т.д.;
- <stdlib.h>, используется для рандомизации точки, в которой будут вычисляться математические функции;

Реализованная структура:

`struct Xn` *(см. Приложение, рис. 1)*. Состоит из таких переменных:

- Количество членов ряда (int)
- Эталонное значение функцци в точке (float)
- Массив double сумм для k элементов
- Массив double погрешностей для k элементов

Реализованные функции:

- `void Test(Xn* series, void(func)(Xn, int, double, double), double ideal, double x, double x0, int count, bool isReversed)`, в аргумент принимает структуру последовательности, в которую будут записываться значения, функцию, по которой будет вычислять сумму членов, точку, в которой будет считаться ряд, начальное значение функции в точке, количество членов, булевую переменную, которая определяет каким способом будет суммироваться ряд
- `void Init(Xn* series, int count),` в аргумент принимает структуру последовательности и количество членов ряда
- `void UnInit(Xn* series),` в аргумент принимает структуру последовательности
- `void InitStation(Xn* series, double ideal, double x0, int count)`, в аргумент принимает структуру последовательности, эталонное значение функции в точке, начальное значение функции в точке, количество членов ряда

- `void InitStationR(Xn* series, double ideal, double x0, int count)`, в аргумент принимает структуру последовательности, эталонное значение функции в точке, начальное значение функции в точке, количество членов ряда
- `void Print(Xn* series, int count),` в аргумент принимает структуру последовательности и количество членов ряда
- `void PrintR(Xn* series, int count),` в аргумент принимает структуру последовательности и количество членов ряда
- `double Factorial(int num)`, в аргумент принимает число, для которого будет считать факториал
- `void Sin(Xn* series, int count, double x, double ideal)`, в аргумент принимает структуру последовательности, количество членов ряда, точку, в которой будет считаться ряд, эталонное значение функции в точке
- `void SinR(Xn* series, int count, double x, double ideal)`, в аргумент принимает структуру последовательности, количество членов ряда, точку, в которой будет считаться ряд, эталонное значение функции в точке
- `void Cos(Xn* series, int count, double x, double ideal)`, в аргумент принимает структуру последовательности, количество членов ряда, точку, в которой будет считаться ряд, эталонное значение функции в точке
- `void CosR(Xn* series, int count, double x, double ideal)`, в аргумент принимает структуру последовательности, количество членов ряда, точку, в которой будет считаться ряд, эталонное значение функции в точке
- `void Ln(Xn* series, int count, double x, double ideal)`, в аргумент принимает структуру последовательности, количество членов ряда, точку, в которой будет считаться ряд, эталонное значение функции в точке
- `void LnR(Xn* series, int count, double x, double ideal)`, в аргумент принимает структуру последовательности, количество членов ряда, точку, в которой будет считаться ряд, эталонное значение функции в точке
- `void Exp(Xn* series, int count, double x, double ideal)`, в аргумент принимает структуру последовательности, количество членов

ряда, точку, в которой будет считаться ряд, эталонное значение функции в точке

- `void ExpR(Xn* series, int count, double x, double ideal)`, в аргумент принимает структуру последовательности, количество членов ряда, точку, в которой будет считаться ряд, эталонное значение функции в точке
- `int main()`

Краткое описание работы функций (принимаемые значения см. *Реализованные функции*):

- *Test (см. Приложение, рис. 2)*. Вызывает функцию *Init* для структуры, на основе булевой переменной isReversed вызывает или функцию *InitStation* или *InitStationR*, заполняет остальные элементы в массивах в структуре по переданной функции, на основе булевой переменной isReversed вызывает функцию *Print* или *PrintR*, освобождает память в структуре функцией *UnInit*
- *Init (см. Приложение, рис. 3)*, выделяет память для массивов в структуре
- *UnInit (см. Приложение, рис. 4)*, освобождает память в структуре
- *InitStation (см. Приложение, рис. 5)*, инициализирует начальные значения в структуре для прямого суммирования
- *InitStationR (см. Приложение, рис. 6)*, инициализирует начальные значения в структуре для обратного суммирования
- *Print (см. Приложение, рис. 7)*, выводит массивы суммы и погрешности для прямого суммирования в консоль
- *PrintR (см. Приложение, рис. 8)*, выводит массивы суммы и погрешности для обратного суммирования в консоль
- *Factorial (см. Приложение, рис. 9)*, считает факториал для переданного в аргумент числа
- *Sin (см. Приложение, рис. 10)*, рекуррентно высчитывает n-ую сумму по формуле подсчёта n-ого члена ряда Тейлора для синуса *(см. Приложение, рис. 19)* и для подсчитанной суммы вычисляет n-ую погрешность через разность эталонного значения и суммы соответственно
- *SinR (см. Приложение, рис. 11)*, рекуррентно высчитывает n-ую сумму по формуле подсчёта n-ого члена ряда Тейлора для синуса и для подсчитанной суммы вычисляет n-ую погрешность через разность эталонного значения и суммы соответственно, но в отличие от функции Sin вычисляет с последнего элемента до первого

- *Cos (см. Приложение, рис. 12)*, рекуррентно высчитывает n-ую сумму по формуле подсчёта n-ого члена ряда Тейлора для косинуса *(см. Приложение, рис. 20)* и для подсчитанной суммы вычисляет n-ую погрешность через разность эталонного значения и суммы соответственно
- *CosR (см. Приложение, рис. 13)*, рекуррентно высчитывает n-ую сумму по формуле подсчёта n-ого члена ряда Тейлора для косинуса и для подсчитанной суммы вычисляет n-ую погрешность через разность эталонного значения и суммы соответственно, но в отличие от функции Cos вычисляет с последнего элемента до первого
- *Ln (см. Приложение, рис. 14)*, рекуррентно высчитывает n-ую сумму по формуле подсчёта n-ого члена ряда Тейлора для логарифма *(см. Приложение, рис. 21)* и для подсчитанной суммы вычисляет n-ую погрешность через разность эталонного значения и суммы соответственно
- *LnR (см. Приложение, рис. 15)*, рекуррентно высчитывает n-ую сумму по формуле подсчёта n-ого члена ряда Тейлора для логарифма и для подсчитанной суммы вычисляет n-ую погрешность через разность эталонного значения и суммы соответственно, но в отличие от функции Ln вычисляет с последнего элемента до первого
- *Exp (см. Приложение, рис. 16)*, рекуррентно высчитывает n-ую сумму по формуле подсчёта n-ого члена ряда Тейлора для экспоненты *(см. Приложение, рис. 22)* и для подсчитанной суммы вычисляет n-ую погрешность через разность эталонного значения и суммы соответственно
- *ExpR (см. Приложение, рис. 17)*, рекуррентно высчитывает n-ую сумму по формуле подсчёта n-ого члена ряда Тейлора для экспоненты и для подсчитанной суммы вычисляет n-ую погрешность через разность эталонного значения и суммы соответственно, но в отличие от функции Exp вычисляет с последнего элемента до первого
- main (см. Приложение, рис. 18), переменные: 8 структур ряда (4 для прямого суммирования и 4 для обратного), число членов ряда (int), точка, в которой будут вычисляться функции (double), 4 эталонных значения для функций в точке (double). Вызывает функцию Test для каждой функции дважды (для прямого и обратного суммирования).

Результаты экспериментов

Для облегчения проведения исследований, как я уже написал ранее, функция *main* тестирует сразу все 4 функции по два раза (с прямым и обратным суммированием). Я выводил результаты вычислений для 10-и членов ряда. Вычисления проводил для x = 0.5. Посмотрим на полученные результаты.

1. Синус.

Таблица 1. Суммы и погрешности для вычисления синуса при прямом и обратном суммировании

Сумма	Погрешность		Сумма (обр.)	Погрешность (обр.)
0.500000	-0.020574		0.000000	0.479426
0.479167	0. 000259		0.000000	0.479426
0.479427	0.000002		0.000000	0.479426
0.479426	0.000000		0.000000	0.479426
0.479426	0.000000		0.000000	0.479426
0.479426	0.000000		0.000000	0.479426
0.479426	0.000000		0.000002	0. 479427
0.479426	0.000000		0.000259	0.479167
0.479426	0.000000		-0.020574	0.500000
0.479426	0.000000		0. 479426	0.000000

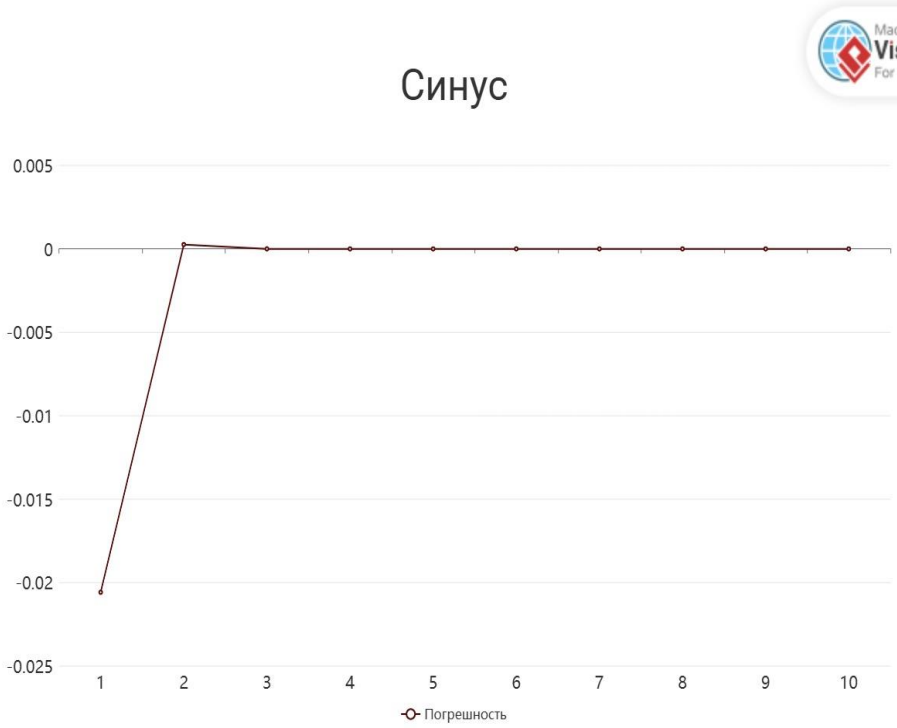

Рис. 23. График погрешности для синуса при прямом суммировании

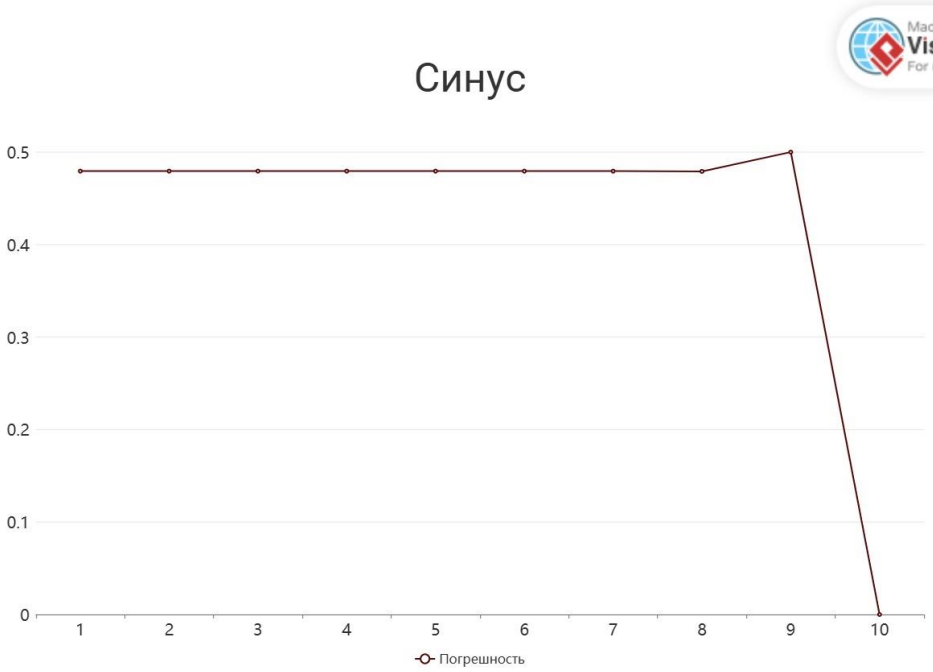

Рис. 24. График погрешности для синуса при обратном суммировании

2. Косинус

Таблица 2. Суммы и погрешности для вычисления косинуса при прямом и обратном суммировании

Сумма	Погрешность		Сумма (обр.)	Погрешность (обр.)
1.000000	-0.122417		0.000000	0.877583
0.875000	0.002583		0.000000	0.877583
0.877604	0.000022		0.000000	0.877583
0.877582	0.000000		0.000000	0.877583
0.877583	0.000000		0.000000	0.877583
0.877583	0.000000		0.000000	0.877582
0.877583	0.000000		0.000022	0.877604
0.877583	0.000000		0.002583	0.875000
0.877583	0.000000		-0.122417	1.000000
0.877583	0.000000		0. 877583	0.000000

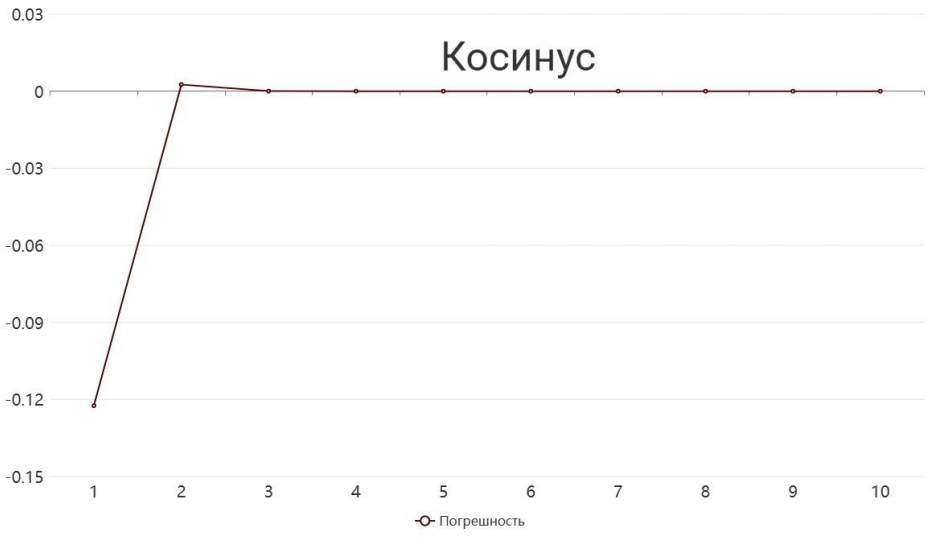

Рис. 25. График погрешности для косинуса при прямом суммировании

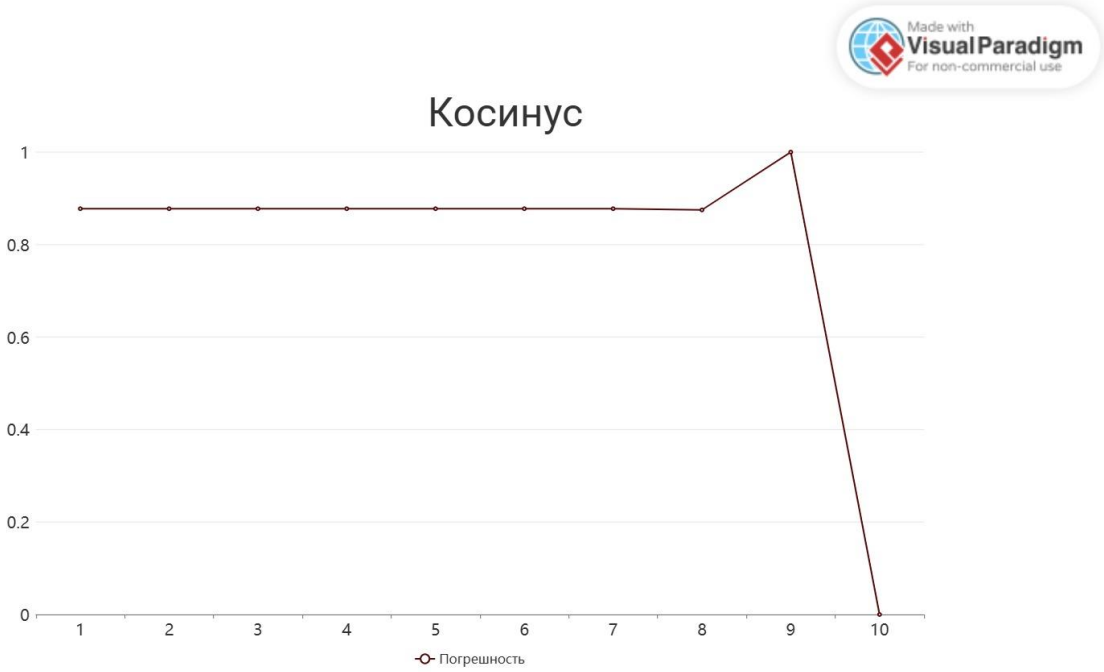

Рис. 26. График погрешности для косинуса при обратном суммировании

3. Натуральный логарифм

Таблица 3. Суммы и погрешности для вычисления логарифма при прямом и обратном суммировании

Сумма	Погрешность		Сумма (обр.)	Погрешность (обр.)
0.500000	-0.094535		-0.000067	0.405532
0.375000	0.030465		0.000150	0.405315
0.416667	0.011202		-0.000338	0.405804
0.401042	0.004423		0.000778	0.404688
0.407292	0.001827		-0.001827	0.407292

0.404687	0.000778		0.004423	0.401042
0.405804	0.000150		0.011202	0.416667
0.405532	0.000067		0.030465	0.375000
0.405435	0.000030		-0.094535	0.500000
0.405479	0.000014		0.405465	0.000000

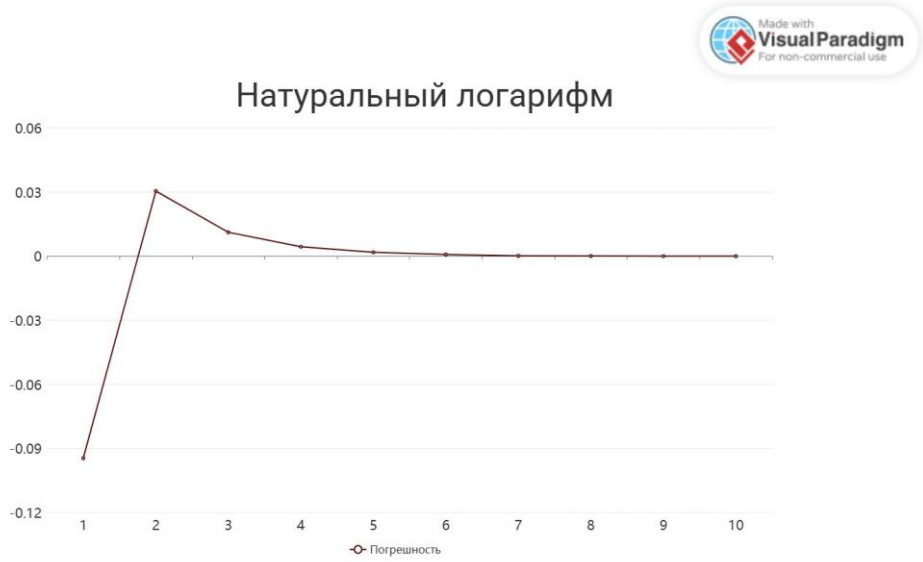

Рис. 27. График погрешности для натурального логарифма при прямом суммировании

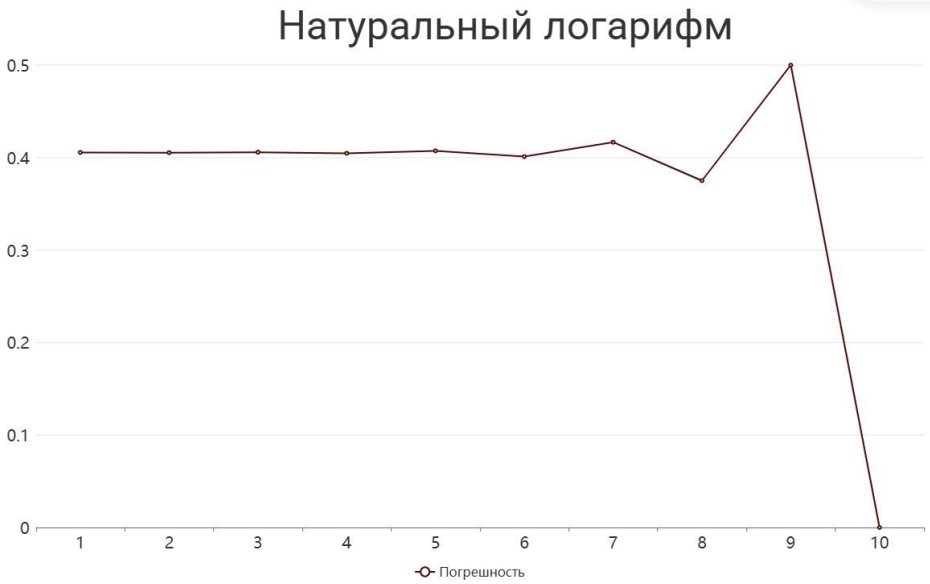

Рис. 28. График погрешности для натурального логарифма при обратном суммировании

4. Экспонента

Таблица 4. Суммы и погрешности для вычисления экспоненты при прямом и обратном суммировании

Сумма	Погрешность		Сумма (обр.)	Погрешность (обр.)
1.000000	0.648721		0.000000	1.648721
1.500000	0.148721		0.000000	1.648721
1.625000	0.023721		0.000000	1.648720
1.645833	0.002888		0.000002	1.648698
1.648438	0.000284		0.000284	1.648438

1.648698	0.000002		0.002888	1.645833
1.648720	0.000000		0.023721	1.625000
1.648721	0.000000		0.148721	1.500000
1.648721	0.000000		0.648721	1.000000
1.648721	0.000000		1.48721	0.000000

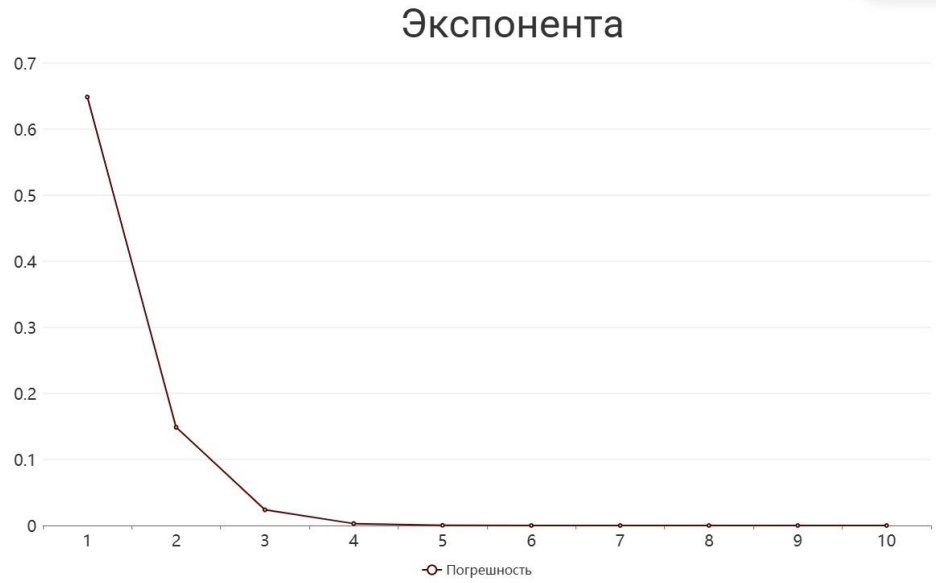

Рис. 29. График погрешности для экспоненты при прямом суммировании

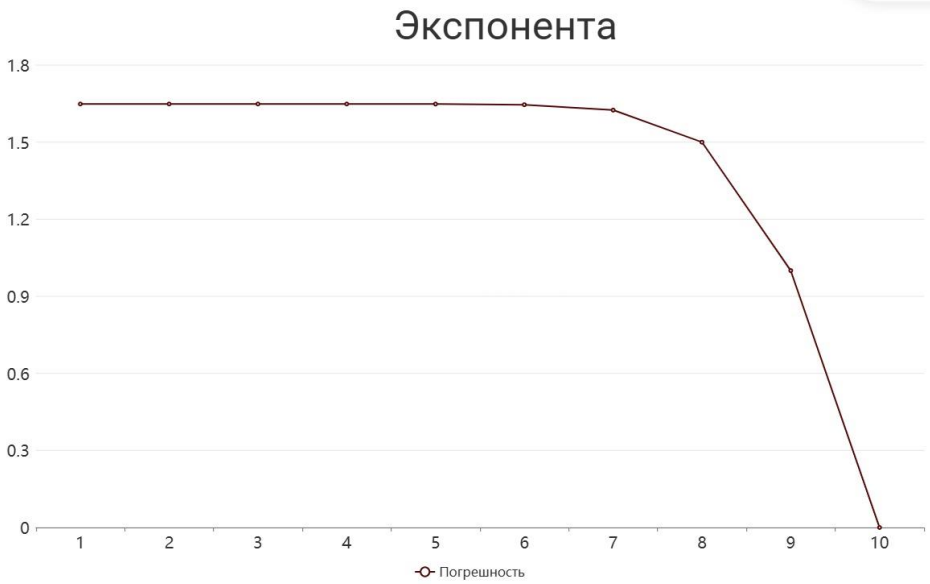

Рис. 30. График погрешности для экспоненты при обратном суммировании

Как видно из таблиц и графиков, погрешность при увеличивающемся числе членов ряда действительно стремится к нулю, более отчётливо это заметно у функции натурального логарифма и экспоненты, где погрешность постепенно убывает к нулю, тогда как у синуса или косинуса погрешность уже после 3-его члена становится равна нулю. Гипотеза подтверждена.

Заключение

В заключении хочется сказать, что наша гипотеза действительно оказалась подтверждена. При помощи рядов Тейлора мы можем вычислять значение стандартных математических функций в нужной нам точке с нужной точностью, что подтверждает огромную пользу данного метода вычисления и значимость вклада Брука Тейлора в развитие математики.

Литература

1. Керниган Б., Ритчи Д., Фьюэр А. Язык программирования СИ //М.: Финансы и статистика. – 1992.
2. Кнут Д. Э. Искусство программирования: Сортировка и поиск. – Издательский дом Вильямс, 2000. – Т. 3.
3. Вирт Н. Алгоритмы и структуры данных.
4. Дж. Клейнберг, Э. Тардос. Алгоритмы: разработка и применение.
5. Р. Мартин. Чистый код.

Приложение

```
struct Xn {
    int n;
    float ideal;
    double* accurance;
    double* ksum;
};
```

Рис. 1. Структура ряда Тейлора

```
void Test(Xn* series, void(*func)(Xn*, int, double, double), double ideal, double x, double x0, int count, bool isReversed) {
    Init(series, count);
    if (isReversed) InitStationR(series, ideal, x0, count);
    else InitStation(series, ideal, x0, count);
    func(series, count, x, ideal);
    if (isReversed) PrintR(series, count);
    else Print(series, count);
    UnInit(series);
}
```

Рис. 2. Функция Test

```
void Init(Xn* series, int count) {
    series->accurance = (double*)malloc(sizeof(double) * count);
    series->ksum = (double*)malloc(sizeof(double) * count);
}
```

Рис. 3. Функция Init

```
void UnInit(Xn* series) {
    free(series->accurance);
    free(series->ksum);
    series->accurance = NULL;
    series->ksum = NULL;
}
```

Рис. 4. Функция UnInit

```c
void InitStation(Xn* series, double ideal, double x0, int count) {
    series->n = count;
    series->ideal = ideal;
    series->ksum[0] = x0;
    series->accurance[0] = series->ideal - x0;
}
```

Рис. 5. Функция InitStation

```c
void InitStationR(Xn* series, double ideal, double x0, int count) {
    series->n = count;
    series->ideal = ideal;
    series->ksum[count - 1] = x0;
    series->accurance[count - 1] = series->ideal - x0;
}
```

Рис. 6. Функция InitStationR

```c
void Print(Xn* series, int count) {
    for (int i = 0; i < count; i++) {
        printf("Sum: %lf  Err: %lf\n", series->ksum[i], series->accurance[i]);
    }
    printf("\n");
}
```

Рис. 7. Функция Print

```c
void PrintR(Xn* series, int count) {
    for (int i = count - 1; i >= 0; i--) {
        printf("Sum: %lf  Err: %lf\n", series->ksum[i], series->accurance[i]);
    }
    printf("\n");
}
```

Рис. 8. Функция PrintR

```c
double Factorial(int num) {
    if (num == 0 || num == 1) return 1;
    else return num * Factorial(num - 1);
}
```

Рис. 9. Функция Factorial

```
void Sin(Xn* series, int count, double x, double ideal) {
    for (int i = 1; i < count; i++) {
        series->ksum[i] = series->ksum[i - 1] + pow(-1, i) * (pow(x, 2 * i + 1) / Factorial(2 * i + 1));
        series->accurance[i] = fabs(series->ksum[i] - ideal);
    }
}
```

Рис. 10. Функция Sin

```
void SinR(Xn* series, int count, double x, double ideal) {
    for (int i = count - 1; i >= 0; i--) {
        series->ksum[i] = series->ksum[i + 1] + pow(-1, i) * (pow(x, 2 * i + 1) / Factorial(2 * i + 1));
        series->accurance[i] = fabs(series->ksum[i] - ideal);
    }
}
```

Рис. 11. Функция SinR

```
void Cos(Xn* series, int count, double x, double ideal) {
    for (int i = 1; i < count; i++) {
        series->ksum[i] = series->ksum[i - 1] + pow(-1, i) * (pow(x, 2 * i) / Factorial(2 * i));
        series->accurance[i] = fabs(series->ksum[i] - ideal);
    }
}
```

Рис. 12. Функция Cos

```
void CosR(Xn* series, int count, double x, double ideal) {
    for (int i = count - 1; i >= 0; i--) {
        series->ksum[i] = series->ksum[i + 1] + pow(-1, i) * (pow(x, 2 * i) / Factorial(2 * i));
        series->accurance[i] = fabs(series->ksum[i] - ideal);
    }
}
```

Рис. 13. Функция CosR

```
void Ln(Xn* series, int count, double x, double ideal) {
    for (int i = 1; i < count; i++) {
        series->ksum[i] = series->ksum[i - 1] + pow(-1, i) * (pow(x, i + 1) / (i + 1));
        series->accurance[i] = fabs(series->ksum[i] - ideal);
    }
}
```

Рис. 14. Функция Ln

```
void LnR(Xn* series, int count, double x, double ideal) {
    for (int i = count - 1; i >= 0; i--) {
        series->ksum[i] = series->ksum[i + 1] + pow(-1, i) * (pow(x, i + 1) / (i + 1));
        series->accurance[i] = fabs(series->ksum[i] - ideal);
    }
}
```

Рис. 15. Функция LnR

```
void Exp(Xn* series, int count, double x, double ideal) {
    for (int i = 1; i < count; i++) {
        series->ksum[i] = series->ksum[i - 1] + (pow(x, i) / Factorial(i));
        series->accurance[i] = fabs(series->ksum[i] - ideal);
    }
}
```

Рис. 16. Функция Exp

```
void ExpR(Xn* series, int count, double x, double ideal) {
    for (int i = count - 1; i >= 0; i--) {
        series->ksum[i] = series->ksum[i + 1] + (pow(x, i) / Factorial(i));
        series->accurance[i] = fabs(series->ksum[i] - ideal);
    }
}
```

Рис. 17. Функция ExpR

```
int main()
{
    struct Xn sinXn {}, cosXn{}, logXn{}, expXn{}, sinXnR{}, cosXnR{}, logXnR{}, expXnR{};

    int count = 20;

    double x = (rand()/RAND_MAX) * 0.5 + 0.5;
    printf("%lf\n", x);
    double idealSin = sin(x), idealCos = cos(x), idealExp = exp(x), idealLn = log(x+1);

    Test(&sinXn, Sin, idealSin, x, x, count, false);
    Test(&sinXnR, SinR, idealSin, x, pow(-1, count - 1) * (pow(x, 2 * (count - 1) + 1) / Factorial(2 * (count - 1) + 1)), count, true);
    Test(&cosXn, Cos, idealCos, x, 1, count, false);
    Test(&cosXnR, CosR, idealCos, x, pow(-1, count - 1) * (pow(x, 2 * (count - 1)) / Factorial(2 * (count - 1))), count, true);
    Test(&logXn, Ln, idealLn, x, x, count, false);
    Test(&logXnR, LnR, idealLn, x, pow(-1, count - 1)* (pow(x, count - 1 + 1) / (count - 1 + 1)), count, true);
    Test(&expXn, Exp, idealExp, x, 1, count, false);
    Test(&expXnR, ExpR, idealExp, x, (pow(x, count - 1) / Factorial((count - 1))), count, true);
}
```

Рис. 18. Функция main

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!} + \ldots = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!},$$

Рис. 19. Формула Тейлора для разложения синуса

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots + (-1)^n \frac{x^{2n}}{(2n)!} - \ldots = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$$

Рис. 20. Формула Тейлора для разложения косинуса

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \ldots + (-1)^{n-1} \frac{x^n}{n} + \ldots$$

Рис. 21. Формула Тейлора для разложения натурального логарифма

$$e^x = 1 + x + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} + \ldots = \sum_{n=0}^{\infty} \frac{x^n}{n!},$$

Рис. 22. Формула Тейлора для разложения экспоненты