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

«Поиск значений функций при помощи рядов Тейлора».

Выполнила:
студентка группы 3824Б1ПМ1
Терешина Е.Д.

Проверила:

Бусько П.В.

Нижний Новгород
2025

Содержание

Введение

В математике и программировании часто нужно вычислять значения функций, но не все функции легко посчитать напрямую. Для таких случаев используют приближенные методы, например, ряды Тейлора. Ряд Тейлора — это способ представить функцию в виде суммы простых слагаемых, что позволяет находить её значения с нужной точностью.

Ряды Тейлора применяются во многих областях, например, в физике, инженерии и компьютерных расчетах. Они помогают работать с такими функциями, как синус, косинус, экспонента, логарифм и другими, которые сложно вычислить точно.

В этой работе мы познакомимся с тем, как работают ряды Тейлора, и как их можно использовать для вычисления значений функций и насколько представление функции с помощью рядов Тейлора приближенно к реальному значению функции в данной точке.

Постановка задачи

Задача состоит в том, чтобы найти значение функции с помощью рядов Тейлора двумя способами и проверить погрешность этих способов. Для этого нужно вычислить члены ряда Тейлора функций sin(x), cos(x), exp(x), ln(1+x). Потом просуммировать их двумя способами: прямым и обратным. Посчитать погрешность и построить графики зависимости этой погрешности от количества членов ряда Тейлора.

Подробнее про способы суммирования членов ряда Тейлора:

1. Прямой: члены ряда Тейлора складываются от первого (наибольшего) к последнему (наименьшему), поэтому получается, что к числу прибавляется число много меньшее, поэтому первое число не меняется и теряется точность;
2. Обратный: члены ряда Тейлора складываются от последнего (наименьшего) к первому (наибольшему), поэтому сохраняются все знаки, значит точность должна быть выше.

Чтобы вычислить i-ый член ряда Тейлора, будем использовать рекуррентную формулу для каждой функции (рис. 1).

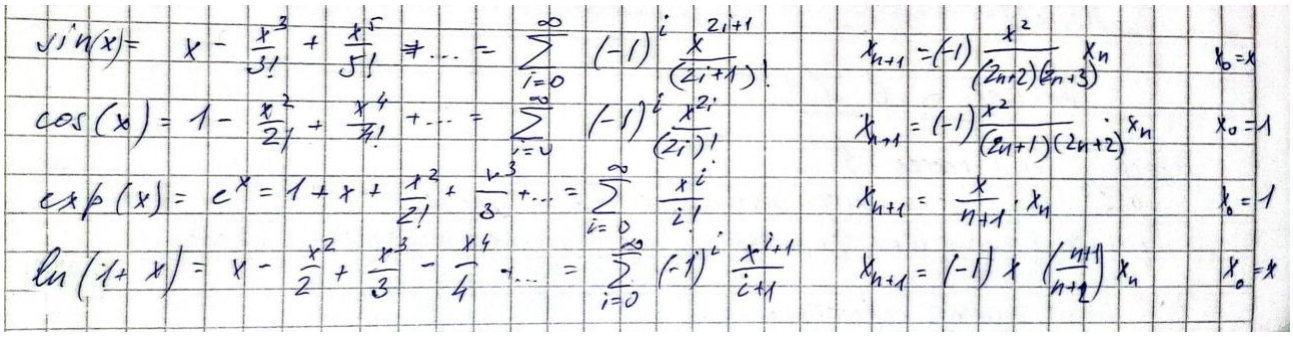


Рисунок 1. Формулы для вычисления i-того члена ряда Тейлора для всех функций

Руководство пользователя

Запустите программу. Введите число от 1 до 8, согласно указаниям на экране (см. рис.2), чтобы выбрать функцию. (Обратная функция означает, что при подсчёте погрешности будут складываться с последнего элемента). Например, ‘1’. Далее нажмите Enter. Если всё сделано верно, то на экран выведется таблица (см. рис.3) с результатами погрешности выбранной функции от количества элементов времени сортировки массивов с разным количеством элементов.

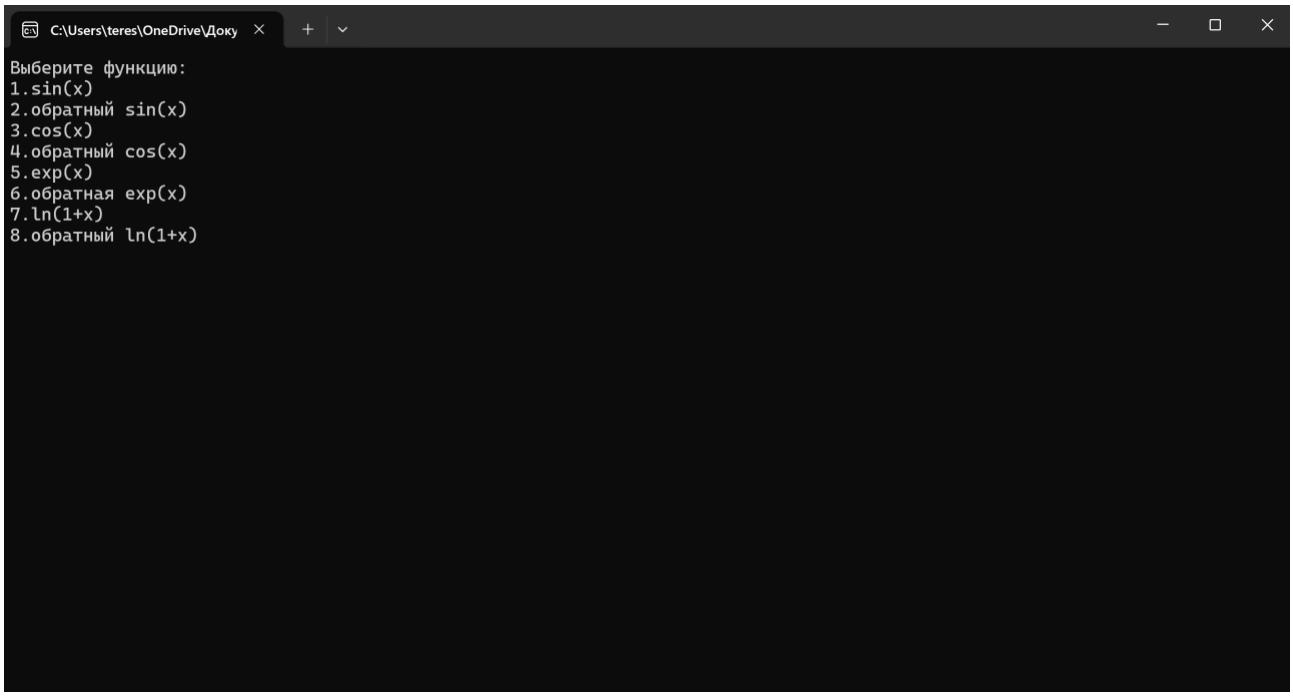


Рисунок 2. Экран с указаниями, который увидит пользователь, запустив программу.

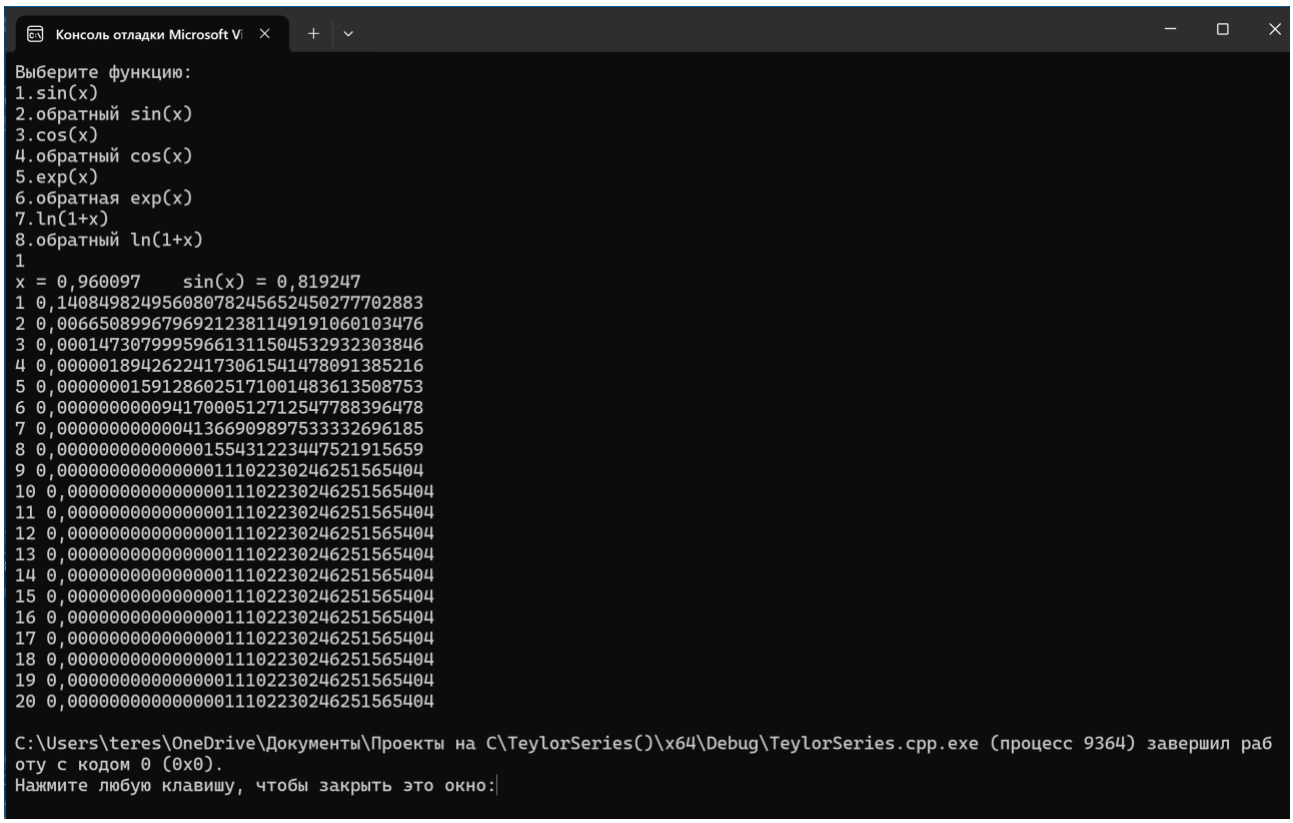


Рисунок 3. Экран, который увидит пользователь, после окончания работы программы.

Описание программной реализации

Проект состоит из 3 файлов: main.cpp (содержит вызовы функций и считывание одной переменной, вводимых пользователем), header.h (содержит список всех функций и объявление структуры), header.cpp (содержит реализацию всех функций, используемых в проекте).

Рассмотрим структуру, все функции и их принцип работы:

1. `struct x_n` – структура (см. рис. 4), в которой хранится массив из последовательных членов ряда Тейлора, массив из суммы этих членов, массив из погрешностей для каждой суммы и количество этих членов;

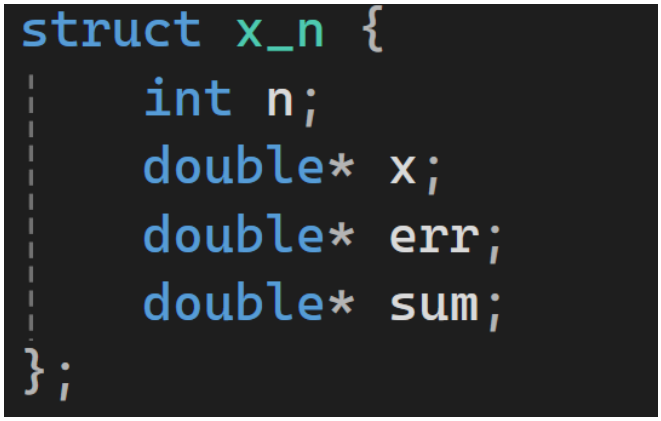


Рисунок 4. Структура x_n.

2. `void locat(void)` – функция (см. рис. 5) для локализации, чтобы можно было выводить в консоль русский текст;

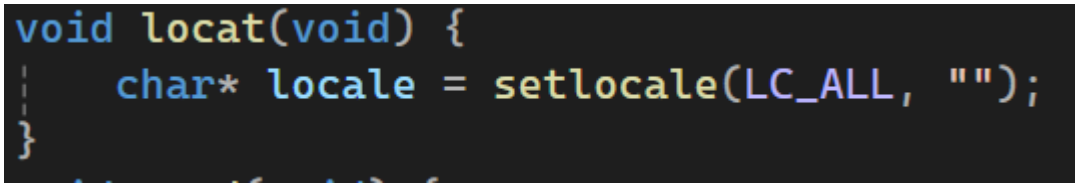


Рисунок 5. Функция locat.

3. `void vvod(void)` – функция (см. рис. 6) выводит в консоль указания для пользователя;

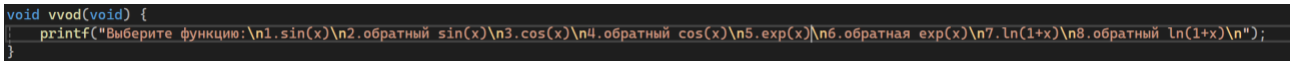


Рисунок 6. Функция vvod.

4. `double point(void)` – функция (см. рис. 7) возвращает сгенерированное число от 0,5 до 1;

```
double point(void) {
    return ((double)(rand())) / RAND_MAX * 0.5 + 0.5;
}
```

Рисунок 7. Функция point.

5. `struct x_n init(x_n a, int n)` – функция (см. рис. 8) возвращает структуру x_n, для каждого массива которой выделена память;

```
struct x_n init(x_n a, int n) {
    a.x = (double*)malloc(sizeof(double) * n);
    a.sum = (double*)malloc(sizeof(double) * n);
    a.err = (double*)malloc(sizeof(double) * n);
    return a;
}
```

Рисунок 8.Функция init.

6. `void free_x(x_n a)` – функция (см. рис. 9), которая освобождает память всех массивов структуры x_n;

```
void free_x(x_n a) {
    free(a.x);
    free(a.err);
    free(a.sum);
}
```

Рисунок 9. Функция free_x.

7. `void copymas(double masa[], double masb[], int n)` – функция (см. рис. 10), которая копирует поэлементно один массив в другой;

```
void copymas(double masa[], double masb[], int n) {
    for (int i = 0; i < n; i++)
        masa[i] = masb[i];
}
```

Рисунок 10. Функция copymas.

8. `struct x_n sinx(int n, double x)` – функция (см. рис. 11), которая возвращает структуру x_n, получая на вход количество членов ряда Тейлора и точку, в которой нужно вычислить значение, сначала она инициализирует все массивы структуры, затем

создает дополнительный массив, в который записывает сначала все i-ые члены ряда Тейлора для sin(x), потом их сумму и погрешность, в процессе записывая промежуточные массивы в возвращаемую структуру;

```
struct x_n sinx(int n, double x) {
    struct x_n a;
    a = init(a, n);
    a.n = n;
    double* mas = (double*)malloc(sizeof(double) * n);
    mas[0] = x;
    for (int i = 0; i < a.n - 1; i++) {
        mas[i + 1] = ((-1) * x * x * mas[i]) / ((2 * i + 3) * (2 * i + 2));
    }
    copymas(a.x, mas, n);
    for (int i = 1; i < a.n; i++) {
        mas[i] = mas[i - 1] + mas[i];
    }
    copymas(a.sum, mas, n);
    for (int i = 0; i < a.n; i++)
        mas[i] = fabs(sin(x) - a.sum[i]);
    copymas(a.err, mas, n);
    free(mas);
    return a;
    free_x(a);
}
```

Рисунок 11. Функция sinx.

9. `struct x_n resinx(int n, double x)` – функция (см. рис. 12), которая возвращает структуру x_n, получая на вход количество членов ряда Тейлора и точку, в которой нужно вычислить значение, сначала она инициализирует все массивы структуры, затем создает дополнительный массив, в который записывает сначала все i-ые члены ряда Тейлора для sin(x), потом их обратную сумму и погрешность, в процессе записывая промежуточные массивы в возвращаемую структуру;

```
struct x_n resinx(int n, double x) {
    struct x_n a;
    a = init(a, n);
    a.n = n;
    double* mas = (double*)malloc(sizeof(double) * n);
    mas[0] = x;
    for (int i = 0; i < a.n - 1; i++) {
        mas[i + 1] = ((-1) * x * x * mas[i]) / ((2 * i + 3) * (2 * i + 2));
    }
    copymas(a.x, mas, n);
    for (int i = a.n-1; i>=0; i--) {
        for (int j=i-1;j>=0;j--){
            mas[i] += mas[j];
        }
    }
    copymas(a.sum, mas, n);
    for (int i = 0; i < a.n; i++)
        mas[i] = fabs(sin(x) - a.sum[i]);
    copymas(a.err, mas, n);
    free(mas);
    return a;
    free_x(a);
}
```

Рисунок 12. Функция resinx.

10. `struct x_n cosx(int n, double x)` – функция (см. рис. 13), которая возвращает структуру x_n, получая на вход количество членов ряда Тейлора и точку, в которой нужно вычислить значение, сначала она инициализирует все массивы структуры, затем создает дополнительный массив, в который записывает сначала все i-ые члены ряда Тейлора для cos(x), потом их сумму и погрешность, в процессе записывая промежуточные массивы в возвращаемую структуру;

```
struct x_n cosx(int n, double x) {
    struct x_n a;
    a = init(a, n);
    a.n = n;
    double* mas = (double*)malloc(sizeof(double) * n);
    mas[0] = 1;
    for (int i = 0; i < a.n - 1; i++) {
        mas[i + 1] = ((-1) * x * x * mas[i]) / ((2 * i + 1) * (2 * i + 2));
    }
    copymas(a.x, mas, n);
    for (int i = 1; i < a.n; i++) {
        mas[i] = mas[i - 1] + mas[i];
    }
    copymas(a.sum, mas, n);
    for (int i = 0; i < a.n; i++)
        mas[i] = fabs(cos(x) - a.sum[i]);
    copymas(a.err, mas, n);
    free(mas);
    return a;
    free_x(a);
}
```

Рисунок 13. Функция cosx.

11. `struct x_n recosx(int n, double x)` – функция (см. рис. 14), которая возвращает структуру x_n, получая на вход количество членов ряда Тейлора и точку, в которой нужно вычислить значение, сначала она инициализирует все массивы структуры, затем создает дополнительный массив, в который записывает сначала все i-ые члены ряда Тейлора для cos(x), потом их обратную сумму и погрешность, в процессе записывая промежуточные массивы в возвращаемую структуру;

```c
struct x_n recosx(int n, double x) {
    struct x_n a;
    a = init(a, n);
    a.n = n;
    double* mas = (double*)malloc(sizeof(double) * n);
    mas[0] = 1;
    for (int i = 0; i < a.n - 1; i++) {
        mas[i + 1] = ((-1) * x * x * mas[i]) / ((2 * i + 1) * (2 * i + 2));
    }
    copymas(a.x, mas, n);
    for (int i = a.n - 1; i >= 0; i--) {
        for (int j = i - 1; j >= 0; j--) {
            mas[i] += mas[j];
        }
    }
    copymas(a.sum, mas, n);
    for (int i = 0; i < a.n; i++)
        mas[i] = fabs(cos(x) - a.sum[i]);
    copymas(a.err, mas, n);
    free(mas);
    return a;
    free_x(a);
}
```

Рисунок 14. Функция recosx.

12. `struct x_n expx(int n, double x)` – функция (см. рис. 15), которая возвращает структуру x_n, получая на вход количество членов ряда Тейлора и точку, в которой нужно вычислить значение, сначала она инициализирует все массивы структуры, затем создает дополнительный массив, в который записывает сначала все i-ые члены ряда Тейлора для exp(x), потом их сумму и погрешность, в процессе записывая промежуточные массивы в возвращаемую структуру;

```
struct x_n expx(int n, double x) {
    struct x_n a;
    a = init(a, n);
    a.n = n;
    double* mas = (double*)malloc(sizeof(double) * n);
    mas[0] = 1;
    for (int i = 0; i < a.n - 1; i++) {
        mas[i + 1] = (x * mas[i]) / (i + 1);
    }
    copymas(a.x, mas, n);
    for (int i = 1; i < a.n; i++) {
        mas[i] = mas[i - 1] + mas[i];
    }
    copymas(a.sum, mas, n);
    for (int i = 0; i < a.n; i++)
        mas[i] = fabs(exp(x) - a.sum[i]);
    copymas(a.err, mas, n);
    free(mas);
    return a;
    free_x(a);
}
```

Рисунок 15. Функция expx.

13. `struct x_n reexpx(int n, double x)` – функция (см. рис. 16), которая возвращает структуру x_n, получая на вход количество членов ряда Тейлора и точку, в которой нужно вычислить значение, сначала она инициализирует все массивы структуры, затем создает дополнительный массив, в который записывает сначала все i-ые члены ряда Тейлора для sin(x), потом их обратную сумму и погрешность, в процессе записывая промежуточные массивы в возвращаемую структуру;

```c
struct x_n reexpx(int n, double x) {
    struct x_n a;
    a = init(a, n);
    a.n = n;
    double* mas = (double*)malloc(sizeof(double) * n);
    mas[0] = 1;
    for (int i = 1; i < a.n; i++) {
        mas[i] = (x * mas[i - 1]) / i;
    }
    copymas(a.x, mas, n);
    for (int i = a.n - 1; i >= 0; i--) {
        for (int j = i - 1; j >= 0; j--) {
            mas[i] += mas[j];
        }
    }
    copymas(a.sum, mas, n);
    for (int i = 0; i < a.n; i++)
        mas[i] = fabs(exp(x) - a.sum[i]);
    copymas(a.err, mas, n);
    free(mas);
    return a;
    free_x(a);
}
```

Рисунок 16. Функция reexpx.

14. `struct x_n lnx (int n, double x)` – функция (см. рис. 17), которая возвращает структуру x_n, получая на вход количество членов ряда Тейлора и точку, в которой нужно вычислить значение, сначала она инициализирует все массивы структуры, затем создает дополнительный массив, в который записывает сначала все i-ые члены ряда Тейлора для ln(1+x), потом их сумму и погрешность, в процессе записывая промежуточные массивы в возвращаемую структуру;

```
struct x_n lnx(int n, double x) {
    struct x_n a;
    a = init(a, n);
    a.n = n;
    double* mas = (double*)malloc(sizeof(double) * n);
    mas[0] = x;
    for (int i = 0; i < a.n - 1; i++) {
        mas[i + 1] = ((-1.0) * x * mas[i] * (i + 1)) / (i + 2);
    }
    copymas(a.x, mas, n);
    for (int i = 1; i < a.n; i++) {
        mas[i] = mas[i - 1] + mas[i];
    }
    copymas(a.sum, mas, n);
    for (int i = 0; i < a.n; i++)
        mas[i] = fabs(log(x + 1) - a.sum[i]);
    copymas(a.err, mas, n);
    free(mas);
    return a;
    free_x(a);
}
```

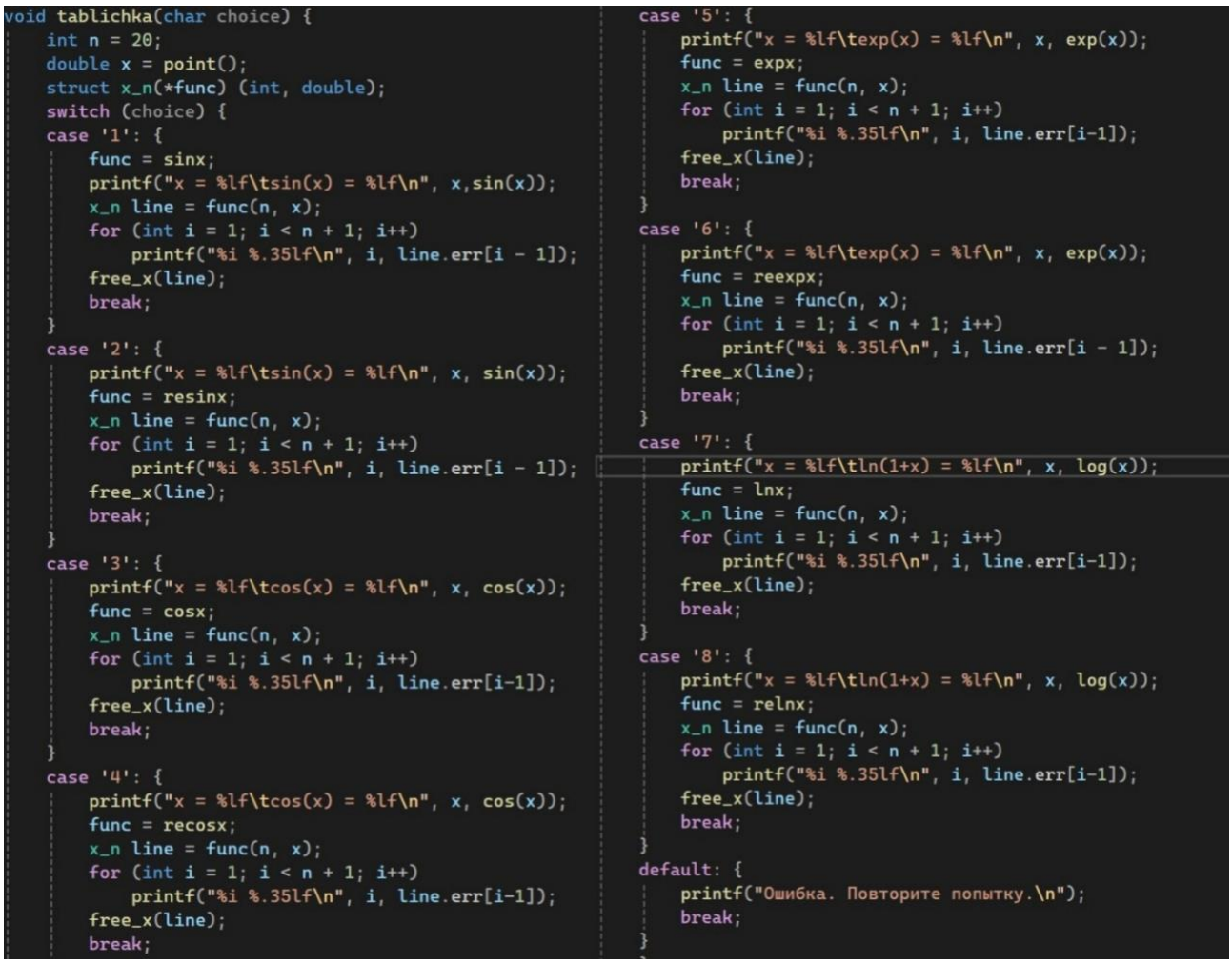
Рисунок 17. Функция lnx.

15. `struct x_n relnx(int n, double x)` – функция (см. рис. 18), которая возвращает структуру x_n, получая на вход количество членов ряда Тейлора и точку, в которой нужно вычислить значение, сначала она инициализирует все массивы структуры, затем создает дополнительный массив, в который записывает сначала все i-ые члены ряда Тейлора для ln(1+x), потом их обратную сумму и погрешность, в процессе записывая промежуточные массивы в возвращаемую структуру;

```
struct x_n relnx(int n, double x) {
    struct x_n a;
    a = init(a, n);
    a.n = n;
    double* mas = (double*)malloc(sizeof(double) * n);
    mas[0] = x;
    for (int i = 0; i < a.n - 1; i++) {
        mas[i + 1] = ((-1) * x * mas[i] * (i + 1)) / (i + 2);
    }
    copymas(a.x, mas, n);
    for (int i = a.n - 1; i >= 0; i--) {
        for (int j = i - 1; j >= 0; j--) {
            mas[i] += mas[j];
        }
    }
    copymas(a.sum, mas, n);
    for (int i = 0; i < a.n; i++)
        mas[i] = fabs(log(x + 1) - a.sum[i]);
    copymas(a.err, mas, n);
    free(mas);
    return a;
    free_x(a);
}
```

Рисунок 18. Функция relnx.

16. `void tablichka(char choice)` – функция (см. рис. 19), которая выводит в консоль таблицу с результатами, получая на вход номер функции, выбранной пользователем, выводится сгенерированный x, значение функции в данной точке и погрешность, зависящая от количества членов ряда Тейлора.



```c
void tablichka(char choice) {
    int n = 20;
    double x = point();
    struct x_n(*func) (int, double);
    switch (choice) {
    case '1': {
        func = sinx;
        printf("x = %lf\tsin(x) = %lf\n", x,sin(x));
        x_n line = func(n, x);
        for (int i = 1; i < n + 1; i++)
            printf("%i %.35lf\n", i, line.err[i - 1]);
        free_x(line);
        break;
    }
    case '2': {
        printf("x = %lf\tsin(x) = %lf\n", x, sin(x));
        func = resinx;
        x_n line = func(n, x);
        for (int i = 1; i < n + 1; i++)
            printf("%i %.35lf\n", i, line.err[i - 1]);
        free_x(line);
        break;
    }
    case '3': {
        printf("x = %lf\tcos(x) = %lf\n", x, cos(x));
        func = cosx;
        x_n line = func(n, x);
        for (int i = 1; i < n + 1; i++)
            printf("%i %.35lf\n", i, line.err[i-1]);
        free_x(line);
        break;
    }
    case '4': {
        printf("x = %lf\tcos(x) = %lf\n", x, cos(x));
        func = recosx;
        x_n line = func(n, x);
        for (int i = 1; i < n + 1; i++)
            printf("%i %.35lf\n", i, line.err[i-1]);
        free_x(line);
        break;
    case '5': {
        printf("x = %lf\texp(x) = %lf\n", x, exp(x));
        func = expx;
        x_n line = func(n, x);
        for (int i = 1; i < n + 1; i++)
            printf("%i %.35lf\n", i, line.err[i-1]);
        free_x(line);
        break;
    }
    case '6': {
        printf("x = %lf\texp(x) = %lf\n", x, exp(x));
        func = reexpx;
        x_n line = func(n, x);
        for (int i = 1; i < n + 1; i++)
            printf("%i %.35lf\n", i, line.err[i - 1]);
        free_x(line);
        break;
    }
    case '7': {
        printf("x = %lf\tln(1+x) = %lf\n", x, log(x));
        func = lnx;
        x_n line = func(n, x);
        for (int i = 1; i < n + 1; i++)
            printf("%i %.35lf\n", i, line.err[i-1]);
        free_x(line);
        break;
    }
    case '8': {
        printf("x = %lf\tln(1+x) = %lf\n", x, log(x));
        func = relnx;
        x_n line = func(n, x);
        for (int i = 1; i < n + 1; i++)
            printf("%i %.35lf\n", i, line.err[i-1]);
        free_x(line);
        break;
    }
    default: {
        printf("Ошибка. Повторите попытку.\n");
        break;
    }
```

Рисунок 19. Функция tablichka.

Результаты экспериментов

Проведём эксперименты для каждой из функций.

1. sin(x)

1.1. Прямое суммирование:

На рис. 20 можно увидеть таблицу значений погрешностей для каждого количества элементов. На рис. 21 можно увидеть график зависимости погрешности от количества членов ряда Тейлора. (x = 0,844234 sin(x) = 0,000000)

Число слагаемых	Погрешность
1	0,09677136680481750000
2	0,00351376194833585000
3	0,00006005029193933970
4	0,00000059647207351077
5	0,00000000387213794362
6	0,00000000001771183200
7	0,00000000000006006307
8	0,00000000000000022204
9	0,00000000000000011102
10	0,00000000000000011102
11	0,00000000000000011102
12	0,00000000000000011102
13	0,00000000000000011102
14	0,00000000000000011102
15	0,00000000000000011102
16	0,00000000000000011102
17	0,00000000000000011102
18	0,00000000000000011102
19	0,00000000000000011102
20	0,00000000000000011102

Рисунок 20. Таблица погрешности sin(x) от числа слагаемых при прямом суммировании.

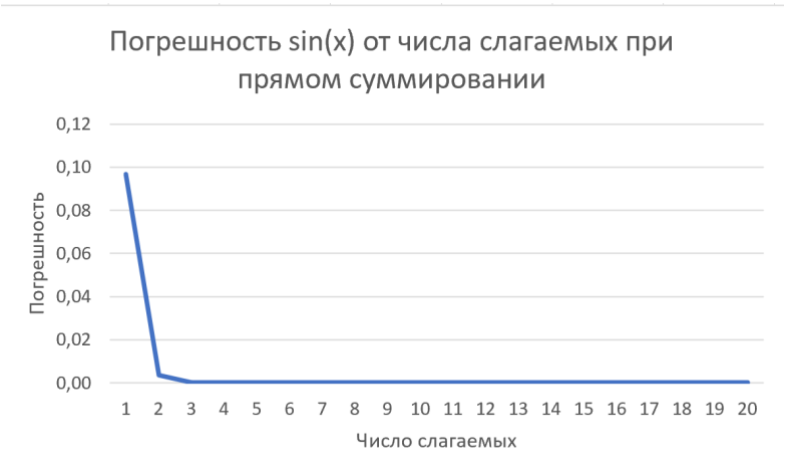


Рисунок 21. График погрешности sin(x) от числа слагаемых при прямом суммировании.

1.2. Обратное суммирование:

На рис. 22 можно увидеть таблицу значений погрешностей для каждого количества элементов. На рис. 23 можно увидеть график зависимости погрешности от количества членов ряда Тейлора. (x = 0,931196 sin(x) = 0,802334)

Число слагаемых	Погрешность
1	0,12886163138705000000
2	0,00571575308029215000
3	0,00011902456487744500
4	0,00000143942478814107
5	0,00000001137328020651
6	0,00000000006330858060
7	0,00000000000026167957
8	0,00000000000000077716
9	0,00000000000000000000
10	0,00000000000000000000
11	0,00000000000000000000
12	0,00000000000000000000
13	0,00000000000000000000
14	0,00000000000000000000
15	0,00000000000000000000
16	0,00000000000000000000
17	0,00000000000000000000
18	0,00000000000000000000
19	0,00000000000000000000
20	0,00000000000000000000

Рисунок 22. График погрешности sin(x) от числа слагаемых при обратном суммировании.

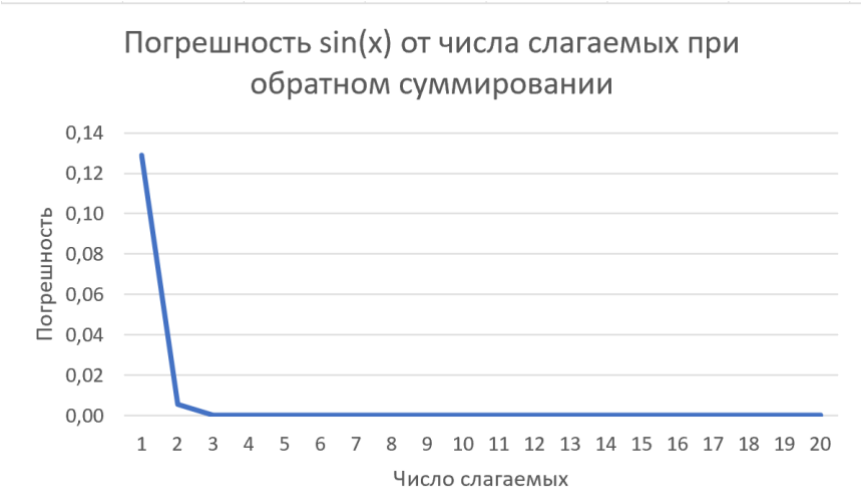


Рисунок 23. График погрешности sin(x) от числа слагаемых при обратном суммировании.

2. cos(x)

2.1. Прямое суммирование:

На рис. 24 можно увидеть таблицу значений погрешностей для каждого количества элементов. На рис. 25 можно увидеть график зависимости погрешности от количества членов ряда Тейлора. (x = 0,966063 cos(x) = 0,568542)

Число слагаемых	Погрешность
1	0,43145762666001700000
2	0,03518163661982920000
3	0,00111039705256388000
4	0,00001862213782366500
5	0,00000019374372028924
6	0,00000000137248201693
7	0,00000000000704647451
8	0,00000000000002764455
9	0,00000000000000011102
10	0,00000000000000022204
11	0,00000000000000022204
12	0,00000000000000022204
13	0,00000000000000022204
14	0,00000000000000022204
15	0,00000000000000022204
16	0,00000000000000022204
17	0,00000000000000022204
18	0,00000000000000022204
19	0,00000000000000022204
20	0,00000000000000022204

Рисунок 24. Таблица погрешности cos(x) от числа слагаемых при прямом суммировании.

Рисунок 25. График погрешности cos(x) от числа слагаемых при прямом суммировании.

2.2. Обратное суммирование:

На рис. 26 можно увидеть таблицу значений погрешностей для каждого количества элементов. На рис. 27 можно увидеть график зависимости погрешности от количества членов ряда Тейлора. (x = 0,986358 cos(x) = 0,551731)

Число слагаемых	Погрешность
1	0,12886163138705000000
2	0,00571575308029215000
3	0,00011902456487744500
4	0,00000143942478814107
5	0,00000001137328020651
6	0,00000000006330858060
7	0,00000000000026167957
8	0,00000000000000077716
9	0,00000000000000000000
10	0,00000000000000000000
11	0,00000000000000000000
12	0,00000000000000000000
13	0,00000000000000000000
14	0,00000000000000000000
15	0,00000000000000000000
16	0,00000000000000000000
17	0,00000000000000000000
18	0,00000000000000000000
19	0,00000000000000000000
20	0,00000000000000000000

Рисунок 26. График погрешности cos(x) от числа слагаемых при обратном суммировании.

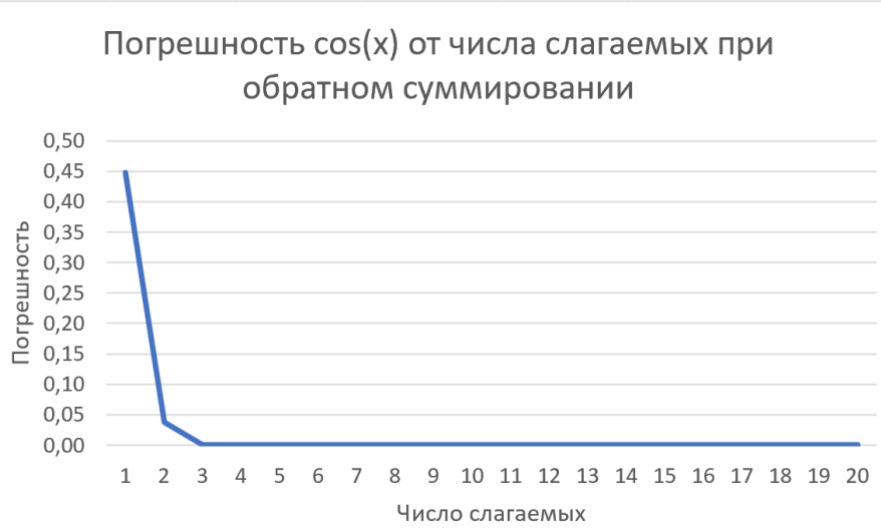


Рисунок 27. График погрешности cos(x) от числа слагаемых при обратном суммировании.

3. exp(x)

3.1. Прямое суммирование:

На рис. 28 можно увидеть таблицу значений погрешностей для каждого количества элементов. На рис. 29 можно увидеть график зависимости погрешности от количества членов ряда Тейлора. (x = 0,532579 exp(x) = 1,703319)

Число слагаемых	Погрешность
1	0,70331867334829000000
2	0,17074016448266300000
3	0,02892023042989650000
4	0,00374348076148178000
5	0,00039133181235984000
6	0,00003427531459609940
7	0,00000258187840285195
8	0,00000017055797596477
9	0,00000001003079619366
10	0,00000000053153770274
11	0,00000000002562772217
12	0,00000000000113353771
13	0,00000000000004640732
14	0,00000000000000177636
15	0,00000000000000000000
16	0,00000000000000000000
17	0,00000000000000000000
18	0,00000000000000000000
19	0,00000000000000000000
20	0,00000000000000000000

Рисунок 28. Таблица погрешности exp(x) от числа слагаемых при прямом суммировании.

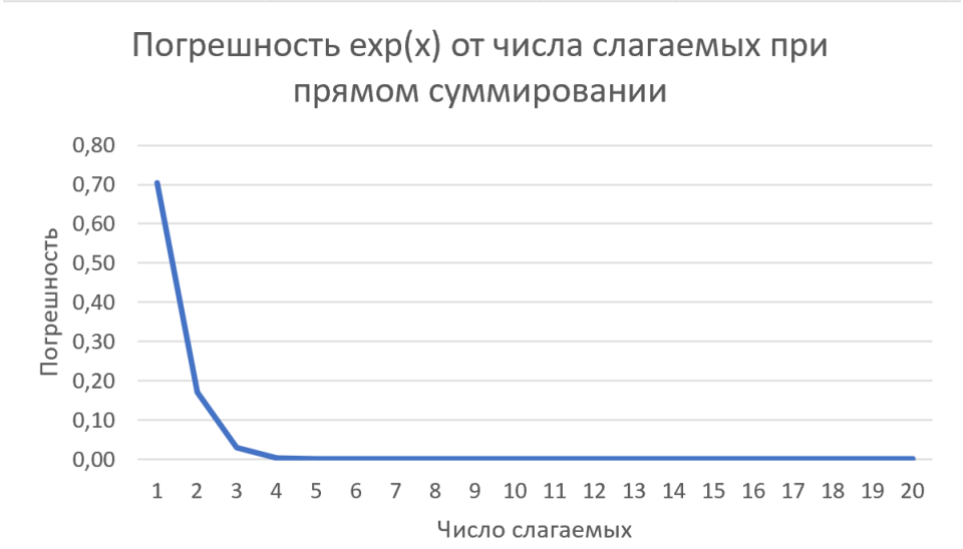


Рисунок 29. График погрешности exp(x) от числа слагаемых при прямом суммировании.

3.2. Обратное суммирование:

На рис. 30 можно увидеть таблицу значений погрешностей для каждого количества элементов. На рис. 31 можно увидеть график зависимости погрешности от количества членов ряда Тейлора. (x = 0,540147 exp(x) = 1,716259)

Число слагаемых	Погрешность
1	0,71625930401491900000
2	0,17611220479924500000
3	0,03023276040369180000
4	0,00396730749520779000
5	0,00042050544568184000
6	0,00003734647797304360
7	0,00000285277714873188
8	0,00000019110965698843
9	0,00000001139816019347
10	0,00000000061253313532
11	0,00000000002995070858
12	0,00000000000134336986
13	0,00000000000005551115
14	0,00000000000000222045
15	0,00000000000000000000
16	0,00000000000000000000
17	0,00000000000000000000
18	0,00000000000000000000
19	0,00000000000000000000
20	0,00000000000000000000

Рисунок 30. График погрешности exp(x) от числа слагаемых при обратном суммировании.

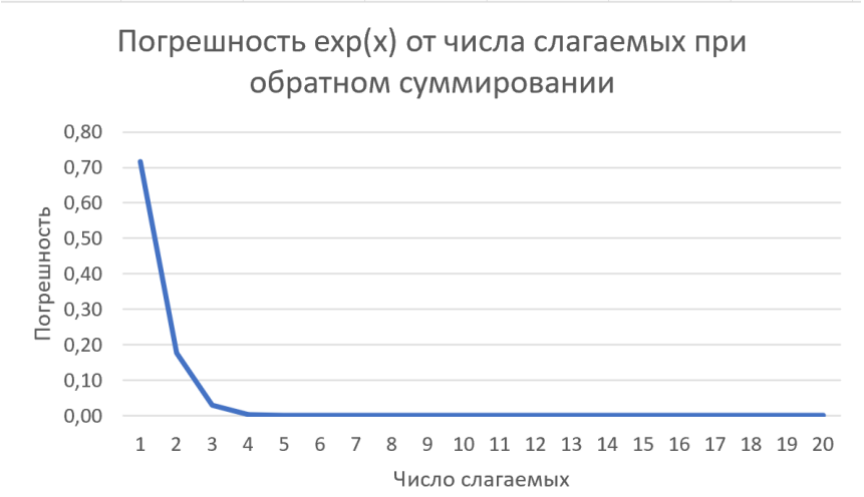


Рисунок 31. График погрешности exp(x) от числа слагаемых при обратном суммировании.

4. ln(1+x)

4.1. Прямое суммирование:

На рис. 32 можно увидеть таблицу значений погрешностей для каждого количества элементов. На рис. 33 можно увидеть график зависимости погрешности от количества членов ряда Тейлора. (x = 0,525651 ln(1+x) = -0,643118)

Число слагаемых	Погрешность
1	0,10322972936162000000
2	0,03492465620104680000
3	0,01348931999311960000
4	0,00559731425707848000
5	0,00242900950342234000
6	0,00108686013297165000
7	0,00049724247768412300
8	0,00023135673631757800
9	0,00010907772143875500
10	0,00005197696133218960
11	0,00002498533321709790
12	0,00001209868468327040
13	0,00000589507895742968
14	0,00000288775482637504
15	0,00000142116859719321
16	0,00000070225866138918
17	0,00000034826487038675
18	0,00000017326541973395
19	0,00000008644882887321
20	0,00000004324422531177

Рисунок 32. Таблица погрешности ln(1+x) от числа слагаемых при прямом суммировании.

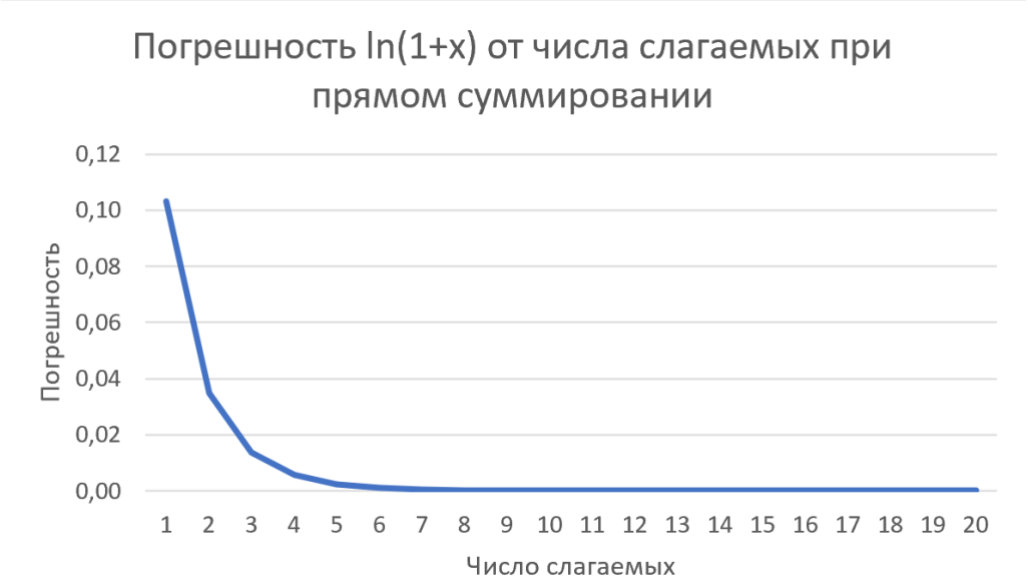


Рисунок 33. График погрешность ln(1+x) от числа слагаемых при прямом суммировании.

4.2. Обратное суммирование:

На рис. 34 можно увидеть таблицу значений погрешностей для каждого количества элементов. На рис. 35 можно увидеть график зависимости погрешности от

количества членов ряда Тейлора. (x = 0,506470 ln(1+x) = -0,680290)

Число слагаемых	Погрешность
1	0,09670080876939040000
2	0,03155508319340580000
3	0,01175008471065590000
4	0,00469948911251655000
5	0,00196548241084437000
6	0,00084752393995413100
7	0,00037365015666773700
8	0,00016752680126025300
9	0,00007610862341278500
10	0,00003494599010522840
11	0,00001618657504864890
12	0,00000755243914024594
13	0,00000354580397882164
14	0,00000167362763109357
15	0,00000079362515770232
16	0,00000037786484147206
17	0,00000018055817108120
18	0,00000008655381961153
19	0,00000004161014954818
20	0,00000002005548638273

Рисунок 34. График погрешности ln(1+x) от числа слагаемых при обратном суммировании.

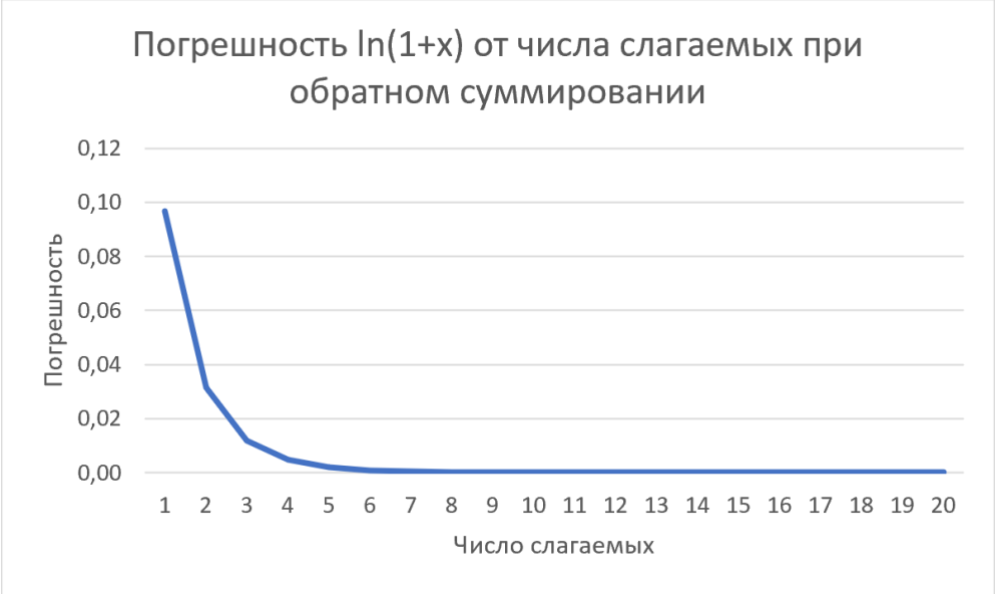


Рисунок 35. График погрешности ln(1+x) от числа слагаемых при обратном суммировании.

Заключение

В ходе данной лабораторной работы были посчитаны первые 20 элементов из рядов Тейлора для функций sin(x), cos(x), exp(x), ln(1+x). Выведена зависимость погрешности значений функций, посчитанных с помощью рядов Тейлора, от количества членов рядов Тейлора. Построены таблицы и графики, наглядно представляющие полученные результаты. Проведены эксперименты, показывающие, что способ прямого суммирования оказался менее точным, чем способ обратного суммирования, но незначительно. Все задачи успешно выполнены.

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.

Приложение

```
#define _CRT_SECURE_NO_WARNINGS
#include "header.h"
int main() {
    srand(time(NULL));
    locat();
    vvod();
    char choice;
    scanf(" %c", &choice);
    tablichka(choice);
}
```

Рисунок 36. Код файла main.cpp.