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**

**«Вычисления функций с помощью Ряда Тейлора»**

**Выполнил**:

студент группы 3824Б1ПМ1

Андреев Артём Михайлович

**Проверил**:

преподаватель каф. ВВСП,

Волокитин В.Д.

Нижний Новгород

2025

**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

Написание программы, которая считает математические функции sin(x), cos(x), ex, ln(1+x) с помощью рядов Тейлора и сравнить разные методы подсчёта со значениями функций в библиотеке <cmath>.

# Метод решения

Написали функцию func\_, которая считает (an+1/an), где an – n-ное слагаемое в ряде. Затем написали три функции sumd [прямая сумма, с первого члена], sumr [обратная сумма, с последнего члена], sumdr [парная сумма, одновременно с первого и последнего]. В процессе суммирования первый по порядку в функции член умножается на func\_ и прибавляется к первому. Второй снова умножается на func и прибавляем к результату сложения первых двух и т. д. Таким образом, мы посчитаем значение функции.

# Руководство пользователя

При запуске программы сначала ввести букву соответствующую выбранной функции( синус – „s“, косинус – „c“, экспонента – „e“, логарифм – „l“ ), затем аргумент функции x, и наконец число слагаемых в ряде (Обратное суммирование будет происходить до члена != 0 в типе данных float). На экран выведется 4 значения — функция через прямое суммирование, обратное, парное и значение функции в библиотеке cmath.

# Описание программной реализации

Были написаны функции(не включая main):

1) sin\_: возвращает -(x2)/(2n(2n+1).

2) cos\_: возвращает -(x2)/(2n(2n-1).

3) exp\_: возвращает x/n.

4) ln\_: возвращает -xn/(n+1).

5) sumd: создаёт переменные res и mem, где оба равны первому члену ряда. Потом, с помощью цикла в каждый проход умножает mem на func\_ (где func – соответствующая функция) и прибавляет его к res. По окончании цикла получен ответ — res и возвращает его.

6) sumr: создаёт переменные res и mem, где оба равны первому члену ряда. Потом, с помощью цикла в каждый проход умножает mem на func\_ (где func – соответствующая функция), пока mem\*func\_ != 0, затем приравнивает n к i ( где i – проход цикла по счёту), получает последний член ряда. Затем запускает цикл, который продолжает, пока mem не станет равен второму члену ряда: {res прибавляется mem и делится на func\_, тем самым получаем предыдущий член}. По окончании цикла получен ответ — res и возвращает его.

7) sumr: создаёт переменные res, memd, и memr, где все равны первому члену ряда. Потом, с помощью цикла в каждый проход умножает memr на func\_ (где func – соответствующая функция), пока memr\*func\_ != 0, затем приравнивает n к i ( где i – проход цикла по счёту), получает последний член ряда. Затем запускает цикл, который продолжает, пока i < n/2:{res прибавляется memd и memr, и memr делится на func\_, тем самым получаем предыдущий член}. По окончании цикла получен ответ — res и возвращает его.

# Подтверждение корректности

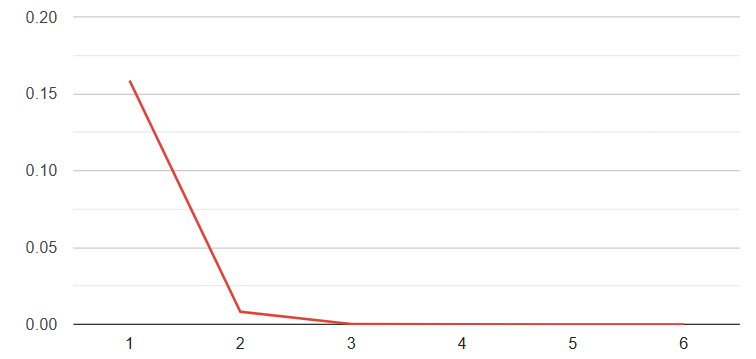
Для подтверждения корректности в программе ведется сравнение со значением функции в библиотеке cmath.

# Результаты экспериментов

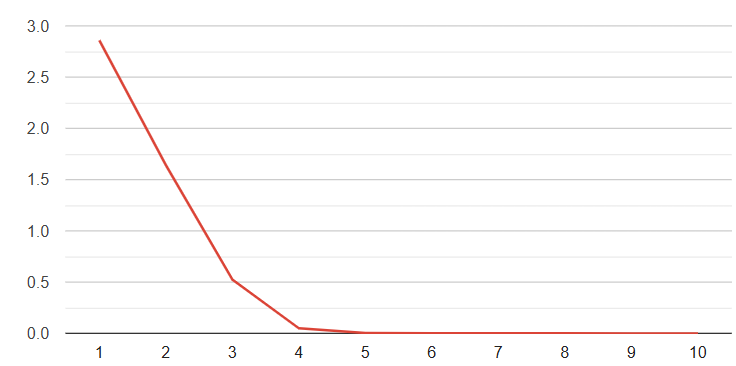
По данным экспериментов видно, что:

1. Прямое суммирование:

1.1. Синус.

При x = 1.

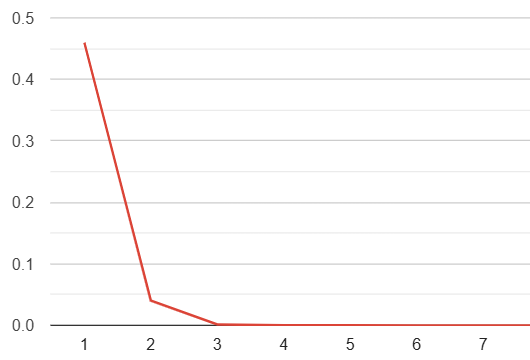
|  |  |
| --- | --- |
| n | |sumd-sin(x)| |
| 1 | 0.1585290431 |
| 2 | 0.0081376433 |
| 3 | 0.0001956820 |
| 4 | 0.0000027418 |
| 5-10 | 0.0 |

При x = 3:

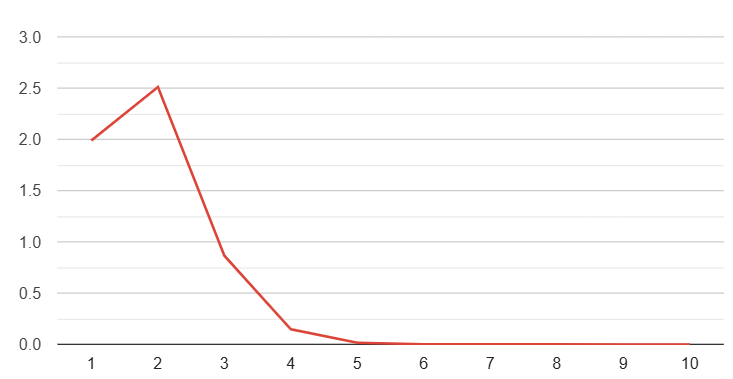
|  |  |
| --- | --- |
| n | |sumd-sin(x)| |
| 1 | 2.8588799983 |
| 2 | 1.6411200016 |
| 3 | 0.5249998569 |
| 4 | 0.0500486940 |
| 5 | 0.0041923671 |
| 6 | 0.0002455413 |
| 7 | 0.0000104904 |
| 8 | 0.0000004768 |
| 9 | 0.0000001192 |
| 10 | 0.0000001341 |

Как видно из примеров, чем больше аргумент функции, тем меньше точность вычислений. Для x = 10 точность падает до 0.0000198483 для 20 членов, 16.2678451538 для 10.

1.2. Косинус.

При x = 1:

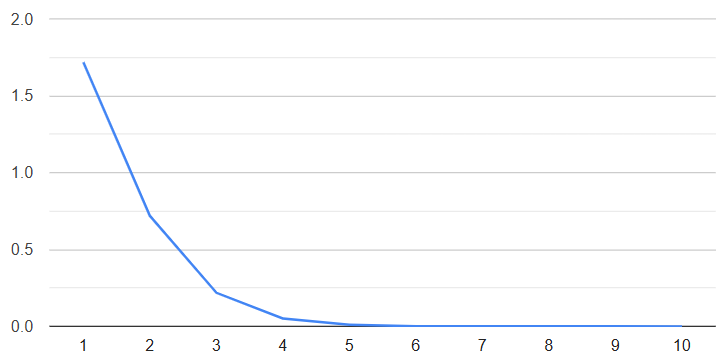
|  |  |
| --- | --- |
| n | |sumd-cos(x)| |
| 1 | 0.4596977233 |
| 2 | 0.0403022766 |
| 3 | 0.0013644099 |
| 4 | 0.0000244975 |
| 5 | 0.0000244975 |
| 6-10 | 0.0 |

При x = 3:

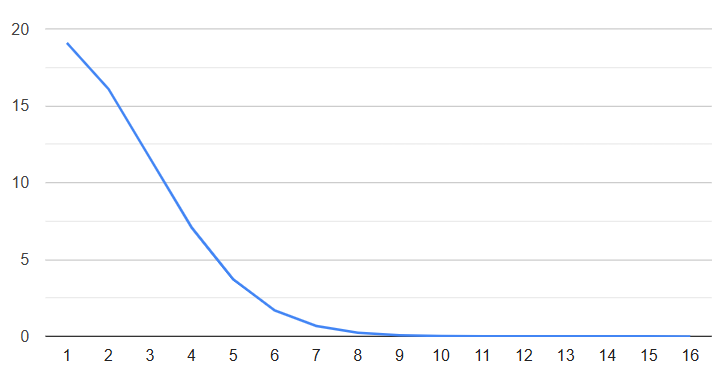
|  |  |
| --- | --- |
| n | |sumd-cos(x)| |
| 1 | 1.9899924993 |
| 2 | 2.5100073814 |
| 3 | 0.8649924993 |
| 4 | 0.1475075483 |
| 5 | 0.0152156352 |
| 6 | 0.0010566711 |
| 7 | 0.0000528097 |
| 8 | 0.0000020265 |
| 9 | 0.0000000596 |
| 10 | 0.0 |

Также как для синуса, для косинуса с большими аргументами увеличится погрешность. Решением этой проблемы будет вместо больших аргументов вводить малые, при которых их функции эквивалентны, то есть вычесть 2πk.

1.3. Экспонента.

При x = 1:

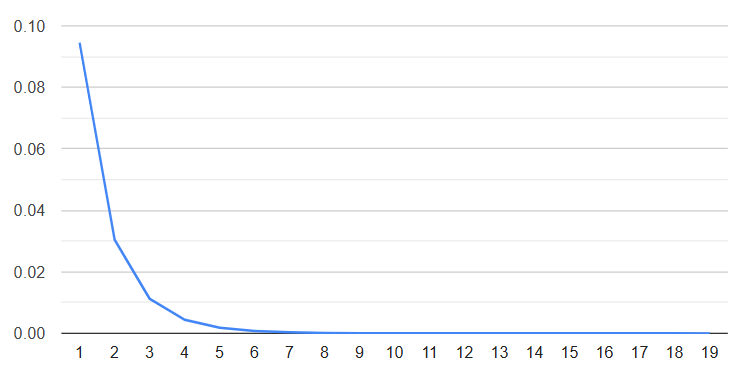
|  |  |
| --- | --- |
| n | |sumd-ex| |
| 1 | 1.7182817459 |
| 2 | 0.7182817459 |
| 3 | 0.2182817459 |
| 4 | 0.0516149997 |
| 5 | 0.0099482536 |
| 6 | 0.0016148090 |
| 7 | 0.0002260208 |
| 8 | 0.0000276565 |
| 9 | 0.0000028610 |
| 10 | 0.0000000000 |

При x = 3:

|  |  |
| --- | --- |
| n  Как видно на графиках, sumd определяет значения экспоненты достаточно точно. | |sumd-ex| |
| 1 | 19.0855369567 |
| 2 | 16.0855369567 |
| 3 | 11.5855369567 |
| 4 | 7.08553695678 |
| 5 | 3.71053695678 |
| 6 | 1.68553733825 |
| 7 | 0.67303657531 |
| 8 | 0.23910713195 |
| 9 | 0.07638359069 |
| 10 | 0.02214241027 |
| 11 | 0.00587081909 |
| 12 | 0.00143241882 |
| 13 | 0.00032234191 |
| 14 | 0.00006675720 |
| 15 | 0.00001144409 |
| 16 | 0.0 |

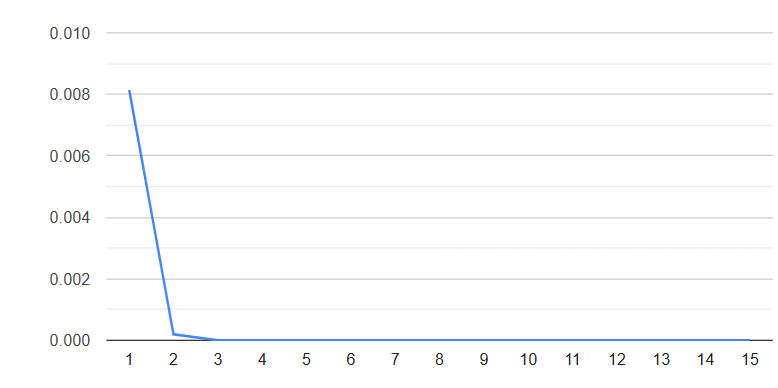
1.4. Логарифм.

Значения аргумента x при разложении логарифма в ряд Тейлора должны быть по модулю < 1, т. к. иначе ряд будет расходится. Но это так же можно использовать, т. к. когда (1+x)<1 можно получить любое число >1 возведя в -1 степень, соответственно умножив логарифм на -1

При x = 0.5:

|  |  |
| --- | --- |
| n  Из-за малого значения аргумента логарифма, его с самого первого члена можно достаточно точно определить, но чтобы точность была до 10 знака нужно больше слагаемых, чем для других функций. | |sumd-ln(1+x)| |
| 1 | 0.0945349037 |
| 2 | 0.0304650962 |
| 3 | 0.0112015604 |
| 4 | 0.0044234395 |
| 5 | 0.0018265545 |
| 6 | 0.0007776021 |
| 7 | 0.0003384649 |
| 8 | 0.0001498162 |
| 9 | 0.0000672042 |
| 10 | 0.0000304579 |
| 11 | 0.0000139176 |
| 12 | 0.0000064373 |
| 13 | 0.0000029504 |
| 14 | 0.0000014007 |
| 15 | 0.0000006258 |
| 16 | 0.0000003278 |
| 17 | 0.0000001192 |
| 18 | 0.0000000894 |
| 19 | 0.0 |

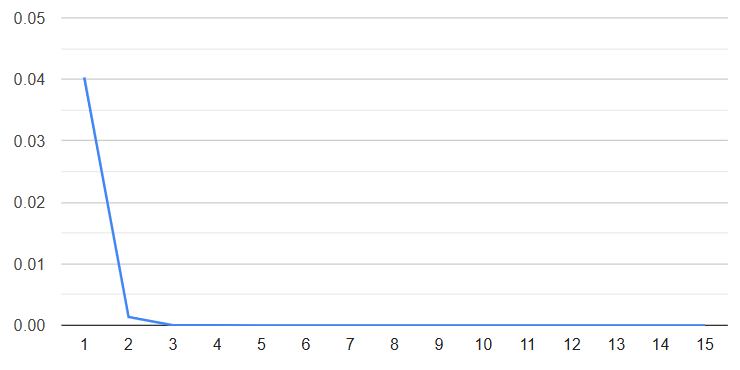
2. Обратное суммирование.

Все значения будем брать для x = 1 (Для логарифма x = 0.5), т. к. уже известно, что точность зависит от значения аргумента.

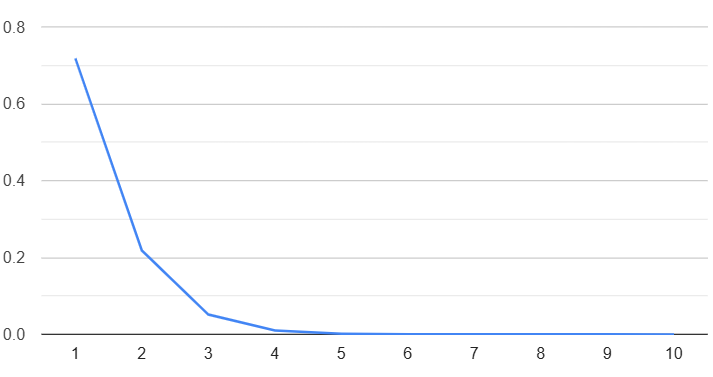
2.1. Синус.

|  |  |
| --- | --- |
| n | |sumr-sin(x)| |
| 1 | 0.0081376433 |
| 2 | 0.0001956820 |
| 3 | 0.0000026822 |
| 4-15 | 0.0000000596 |

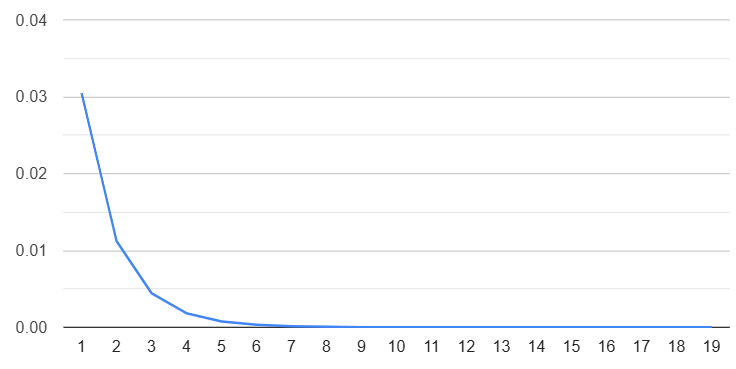
Очевидно, что есть максимум числа слагаемых для обратного суммирования, т. к. при большем их числе, последний член ряда будет равен 0 в типе данных float и, после вычисления остальных членов ряда, они тоже будут равны 0. Поэтому в написанной программе, есть условие, которое ограничивает n до этого максимума.

2.2. Косинус.

|  |  |
| --- | --- |
| n | |sumr-cos(x)| |
| 1 | 0.0403022766 |
| 2 | 0.0013643503 |
| 3 | 0.0000245571 |
| 4-15 | 0.0000002384 |

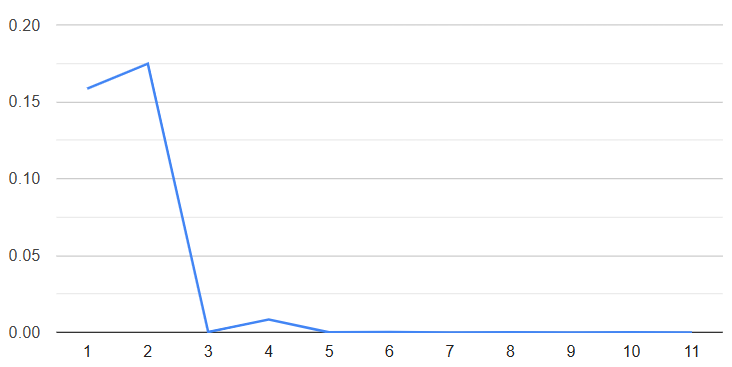
2.3. Экспонента.

|  |  |
| --- | --- |
| n | |sumr-ex| |
| 1 | 0.7182817459 |
| 2 | 0.2182817459 |
| 3 | 0.0516152381 |
| 4 | 0.0099484920 |
| 5 | 0.0016150474 |
| 6 | 0.0002262592 |
| 7 | 0.0000278949 |
| 8 | 0.0000030994 |
| 9 | 0.0000004768 |
| 10 | 0.0 |

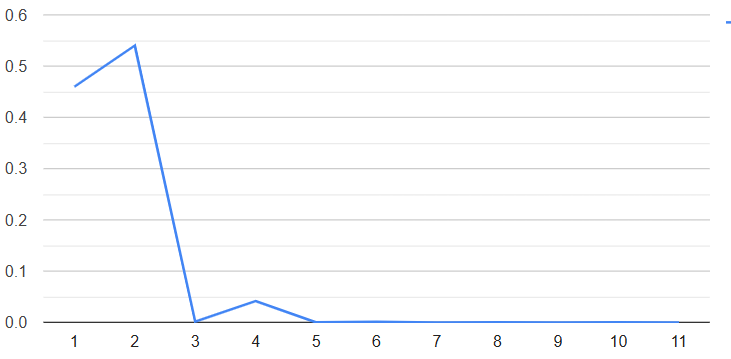
2.4. Логарифм.

|  |  |
| --- | --- |
| n | |sumr-ln(1+x)| |
| 1 | 0.0304650962 |
| 2 | 0.0112015902 |
| 3 | 0.0044234097 |
| 4 | 0.0018266141 |
| 5 | 0.0007775723 |
| 6 | 0.0003385245 |
| 7 | 0.0001497566 |
| 8 | 0.0000672638 |
| 9 | 0.0000304281 |
| 10 | 0.0000139772 |
| 11 | 0.0000064074 |
| 12 | 0.0000030100 |
| 13 | 0.0000013411 |
| 14 | 0.0000006854 |
| 15 | 0.0000002682 |
| 16 | 0.0000002086 |
| 17 | 0.0000000298 |
| 18 | 0.0000000894 |
| 19-30 | 0.0000000298 |

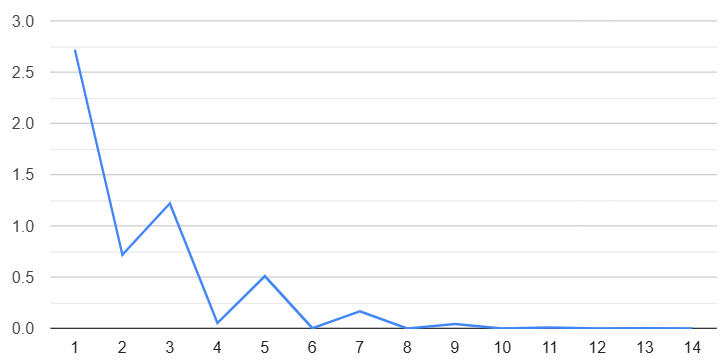
3. Парное суммирование.

3.1. Синус.

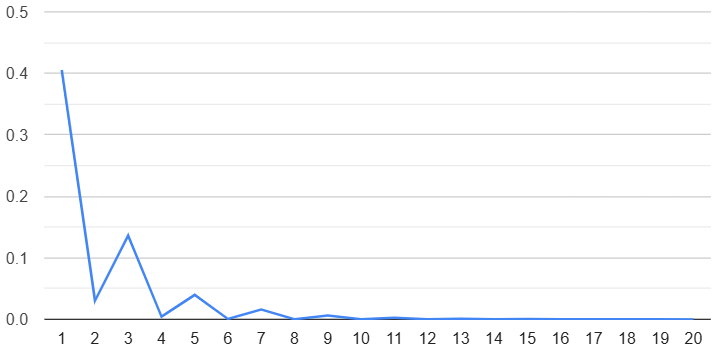
|  |  |
| --- | --- |
| n | |sumdr-sin(x)| |
| 1 | 0.1585290431 |
| 2 | 0.1748043298 |
| 3 | 0.0001957416 |
| 4 | 0.0083305835 |
| 5 | 0.0000000596 |
| 6 | 0.0001984238 |
| 7 | 0.0 |
| 8 | 0.0000027418 |
| 9 | 0 |
| 10 | 0.0000000596 |
| 11-20 | 0.0 |

3.2. Косинус.

|  |  |
| --- | --- |
| n | |sumdr-cos(x)| |
| 1 | 0.4596977233 |
| 2 | 0.5403022766 |
| 3 | 0.0013643503 |
| 4 | 0.0416421294 |
| 5 | 0.0000002980 |
| 6 | 0.0013888478 |
| 7 | 0.0 |
| 8 | 0.0000248551 |
| 9 | 0.0 |
| 10 | 0.0000002384 |
| 11-20 | 0.0 |

3.3. Экспонента.

|  |  |
| --- | --- |
| n | |sumdr-ex| |
| 1 | 2.7182817459 |
| 2 | 0.7182817459 |
| 3 | 1.2182817459 |
| 4 | 0.0516152381 |
| 5 | 0.5099484920 |
| 6 | 0.0016152858 |
| 7 | 0.1668930053 |
| 8 | 0.0000278949 |
| 9 | 0.0416698455 |
| 10 | 0.0000002384 |
| 11 | 0.0083334445 |
| 12 | 0.0000002384 |
| 13 | 0.0013885498 |
| 14 | 0.0 |

3.4. Логарифм.

|  |  |
| --- | --- |
| n | |sumdr-ln(1+x)| |
| 1 | 0.4054650962 |
| 2 | 0.0304650962 |
| 3 | 0.1362015902 |
| 4 | 0.0044234097 |
| 5 | 0.0398401021 |
| 6 | 0.0007776021 |
| 7 | 0.0159634649 |
| 8 | 0.0001498162 |
| 9 | 0.0061827898 |
| 10 | 0.0000304877 |
| 11 | 0.0026181340 |
| 12 | 0.0000063776 |
| 13 | 0.0011131167 |
| 14 | 0.0000013709 |
| 15 | 0.0004889369 |
| 16 | 0.0000002682 |
| 17 | 0.0002169013 |
| 18 | 0.0000001192 |
| 19 | 0.0000976920 |
| 20 | 0.0000000298 |

# Заключение

Из написанного кода и проведённых тестирований, можно сделать вывод, что прямая сумма — наиболее эффективный и точный из трёх алгоритмов вычисления функций.

# Приложение

#include <iostream>

#include <cmath>

// Дефайны для ввода

#define DEF1(func) ({char \_f = (func); (\_f == 's' ? (sin\_) : (\_f == 'c' ? (cos\_) : (\_f == 'e' ? (exp\_) : (ln\_))));})

#define DEF2(func) ({char \_f = (func); (\_f == 's' ? (sin(x)) : (\_f == 'c' ? (cos(x)) : (\_f == 'e' ? (exp(x)) : (log(x+1)))));})

#define DEF3(func) ({char \_f = (func); (((\_f == 'e') || (\_f == 'c')) ? 1 : (x));})

using namespace std;

// Подсчёт членов ряда

float exp\_(float x, int n){

    return x/n;}

float sin\_(float x, int n){

    return -(x\*x)/(2\*n\*(2\*n+1));}

float cos\_(float x, int n){

    return -(x\*x)/(2\*n\*(2\*n-1));}

float ln\_(float x, int n){

    return (-1)\*(x\*n/(n+1));}

// Реализация суммирования

// Прямое суммирование

float sumd(float x0, float x, int n, float(\*f)(float, int)){

    float mem, res;

    mem=res=x0;

    for(int i=1; i<n; i++){

        mem\*=f(x, i);

        res+=mem;}

    return res;}

// Обратное суммирование

float sumr(float x0, float x, int n, float(\*f)(float, int)){

    float mem, res=0;

    mem=res=float(x0);

    for(int i=1; i<=n; i++){

        if(mem\*f(x, i)==0){

            n=i-1;

            break;}

        mem\*=f(x, i);}

    for(int i=0; i<n; i++){

        res+=mem;

        mem/=(f(x, n-i));}

    return res;}

// Парное суммирование

float sumsr(float x0, float x, int n, float(\*f)(float, int)){

    float memd, memr, res;

    memr=memd=res=float(x0);

    for(int i=1; i<=n; i=i+1){

        memr\*=f(x, i);}

    for(int i=1; i<=(int)n/2; i++){

        memd\*=f(x, i);

        memr/=f(x, n-i+1);

        res+=memr+memd;}

    return res;}

int main(){

    float x; int n;

    char func;

    cout<<"> sin(x) - s\n> cos(x) - c\n> e^x - e\n> ln(1+x) - l\n> Enter function: "; cin>>func;

    if(func!='s' && func!='c' && func!='e' && func!='l'){

        cout<<"  Input Error! Please, restart programm.";

        return 0;}

    cout<<"> Enter argument: "; cin>>x;

    cout<<"> Enter number of terms: "; cin>>n;

    if(n < 0){

        cout<<"  Input Error! Please, restart programm.";

        return 0;}

    printf("Straight sum = %.10f\n", sumd(DEF3(func), x, n, DEF1(func)), '\n');

    printf("Reverse sum = %.10f\n", sumr(DEF3(func), x, n, DEF1(func)), '\n');

    printf("Pair sum = %.10f\n", sumsr(DEF3(func), x, n, DEF1(func)), '\n');

    printf("Function in cmath = %.10f\n", DEF2(func));

    return 0;}