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 математической функции с помощью разложения»**

**Выполнил**:

Студент

группы 3822Б1ПМ1

Зазнобин П.В.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2022

**Содержание**

[Постановка задачи 3](#_Toc123230380)

[Метод решения 4](#_Toc123230381)

[Руководство пользователя 6](#_Toc123230382)

[Описание программной реализации 8](#_Toc123230383)

[Подтверждение корректности 12](#_Toc123230384)

[Результаты экспериментов 13](#_Toc123230385)

[Заключение 16](#_Toc123230386)

[Приложение 17](#_Toc123230387)

# Постановка задачи

Цель – научиться находить значение математических функций (sin(x), cos(x), ln(x), expx) с помощью разложения их в ряды Тейлора.

Для выполнения данной цели необходимо поставить и выполнить следующие задачи:

1. Подготовка тестовой программы, которая имеет пользовательский интерфейс, подсчитывает значения функций прямым и обратным способом.
2. Проверить корректность подсчета функций
3. Выявить погрешность вычислений для двух методов посчета в сравнении с заведомо правильными функциями.

# Метод решения

Нужно было написать функции подсчета для следующих математических функций

1. Sin(x)
2. Cos(x)
3. Ln(x)
4. Exp(x)

Для каждой математической функции были выведены формулы для прямого и обратного подсчета.

* Sin(x):

ai+1 =

ai-1 =

* Cos(x):

ai+1 =

ai-1 =

* Ln(x):

ai+1 =

ai-1 =

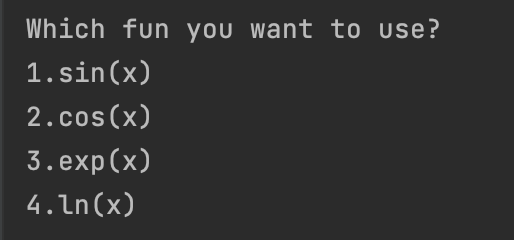
* Exp(x) :

ai+1 =

ai-1 =

Потом была создана функция суммирования всех элементов разложения. Для разных методов подсчета она оказалась разной.

# Руководство пользователя



Рисунок

На Рисунке 1 представлено начальное меню. Тут можно выбрать функцию, которую нужно посчитать

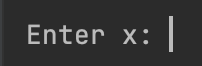


Рисунок 2

Далее требуется ввести значение **х**, для которого нужно посчитать значение функции

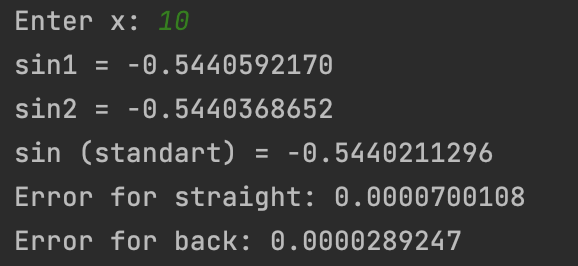


Рисунок 3

На выходе можно получить значения функции в веденной точке, полученное с помощью прямого и обратного способа подсчета. Также на экране выведена погрешность для обоих методов и значение, полученное с помощью встроенной функции.

# Описание программной реализации

Мой проект состоит из 11 функций, одна из которых функция main().

**Две функции для прямого и обратного подсчета синуса:**

float mysin(float x, int ind, float ap) {float ai;  
 ai = (-1) \* x \* x \* ap / (2 \* (float)ind \* (2 \* (float)ind + 1));  
 return ai;  
}  
  
float mysinR(float x, int ind, float an) {  
 float ai;  
 ai = (-an) \* (2 \* (float)ind + 2) \* (2 \* (float)ind + 3) / pow(x, 2);  
 return ai;  
}

**Две функции для прямого и обратного подсчета косинуса:**

float mycos(float x, int ind, float ap) { float ai;  
 ai = (-1) \* x \* x \* ap / (2 \* (float)ind \* (2 \* (float)ind - 1));  
 return ai;  
}  
  
float mycosR(float x, int ind, float an) {  
 float ai;  
 ai = (-1) \* ((2 \* (float)ind) + 1) \* ((2 \* (float)ind) + 2) \* an / (x \* x);  
 return ai;  
}

**Две функции для прямого и обратного подсчета экспоненты:**

float myexp(float x, int ind, float ap) { float ai;  
 ai = x \* ap / (float)ind;  
 return ai;  
}  
  
float myexpR(float x, int ind, float an) {  
 float ai;  
 ai = (float)(ind + 1) \* an / x;  
 return ai;  
}

**Две функции для прямого и обратного подсчета логарифма:**

float myln(float x, int ind, float ap) { float ai;  
 ai = (-1) \* x \* (float)ind \* ap / (float)(ind + 1);  
 return ai;  
}  
  
float mylnR(float x, int ind, float an) {  
 float ai;  
 ai = (-1) \* (float)(ind + 1) \* an / ((float)(ind)\*x);  
 return ai;  
}

**Функция для суммирования элементов, с помощью метода нахождения значения математической функции прямым методом:**

float sumstr(float (\*anything)(float, int, float), float x, float ap) {  
 float s = ap;  
 float ai;  
 for (int i = 1; i < N; i++) {  
 ai = anything(x, i, ap);  
 ap = ai;  
 s += ai;  
 }  
 return s;  
}

**Функция для суммирования элементов, с помощью метода нахождения значения математической функции прямым методом:**

float sumback(float (\*revers)(float, int, float), float (\*straight)(float, int, float), float x, float ap) {  
 float ai;  
 float an;  
 int ind\_a;  
 for (int i = 1; i < N; i++) {  
 ai = straight(x, i, ap);  
 if (fabsf(ai) - pow(10, -28) < 0) {  
 break;  
 }  
 else {  
 an = ai;  
 ind\_a = i;  
 }  
 ap = ai;  
 }  
  
 float s = an;  
 for (int i = ind\_a - 1; i > 0; i--) {  
 ai = revers(x, i, an);  
 an = ai;  
 s += ai;  
 }  
 return s;  
}

**Главная функция, в которой реализован пользовательский интерфейс, и в которой можно выбирать математическая функцию для подсчета значения в ней. Эта функция находит погрешности методов и сравнивает их со стандартной аналогичной математической функцией:**

int main() {  
 float sin0, cos0, exp0, ln0, x, sinl, cosl, expl, lnl, sin1, sin2, cos1, cos2, exp1, exp2, ln1, ln2;  
 int c; *//choice* printf("Which fun you want to use?\n1.sin(x)\n2.cos(x)\n3.exp(x)\n4.ln(x)\n");  
 scanf("%d", &c);  
 printf("Enter x: ");  
 scanf("%f", &x);  
 sin0 = x;  
 cos0 = 1;  
 exp0 = 1;  
 ln0 = x;  
 float err1;  
 float err2;  
 switch (c) {  
 case 1:  
 if (x == 0) {  
 x = 2 \* (float)**M\_PI**;  
 }  
 sin1 = sumstr(mysin, x, sin0);  
 sin2 = sumback(mysinR, mysin, x, sin0) + sin0;  
 printf("sin1 = %.10f\n", sin1);  
 printf("sin2 = %.10f\n", sin2);  
 printf("sin (standart) = %.10f\n", sinf(x));  
 err1 = fabsf(sinf(x) - sin1) / fabsf(sinf(x));  
 err2 = fabsf(sinf(x) - sin2) / fabsf(sinf(x));  
 printf("Error for straight: %.10f\nError for back: %.10f", err1, err2);  
 break;  
 case 2:  
 if (x == 0) {  
 x = 2 \* (float)**M\_PI**;  
 }  
 cos1 = sumstr(mycos, x, cos0);  
 cos2 = sumback(mycosR, mycos, x, cos0) + cos0;  
 printf("cos1 = %f.10\n", cos1);  
 printf("cos2 = %f.10\n", cos2);  
 printf("cos (standart) = %.10f\n", cosf(x));  
 fabsf(cosf(x) - cos1) / fabsf(cosf(x));  
 err1 = fabsf(cosf(x) - cos1) / fabsf(cosf(x));  
 err2 = fabsf(cosf(x) - cos2) / fabsf(cosf(x));  
 printf("Error for straight: %.10f\nError for back: %.10f", err1, err2);  
 break;  
 case 3:  
 if (x == 0) {  
 x = 0;  
 }  
 exp1 = sumstr(myexp, x, exp0);  
 exp2 = sumback(myexpR, myexp, x, exp0) + exp0;  
 printf("exp1 = %.10f\n", exp1);  
 printf("exp2 = %.10f\n", exp2);  
 printf("exp (standart) = %.10f\n", expf(x));  
 err1 = fabsf(expf(x) - exp1) / fabsf(expf(x));  
 err2 = fabsf(expf(x) - exp1) / fabsf(expf(x));  
 printf("Error for straight: %.10f\nError for back: %.10f", err1, err2);  
 break;  
 case 4:  
 if (x <= 0) {  
 printf("ERROR!");  
 break;  
 }  
 ln1 = sumstr(myln, x, ln0);  
 ln2 = sumback(mylnR, myln, x, ln0) + ln0;  
 printf("ln1 = %f\n", ln1);  
 printf("ln2 = %f\n", ln2);  
 printf("ln (standart) = %.10f\n", logf(x+1));  
 err1 = fabsf(logf(x + 1) - ln1) / fabsf(logf(x + 1));  
 err2 = fabsf(logf(x + 1) - ln2) / fabsf(logf(x + 1));  
 printf("Error for straight: %.10f\nError for back: %.10f", err1, err2);  
 break;  
 }  
}

# Подтверждение корректности

Для подтверждения корректности, используются встроенные функции нахождения аналогичных математических функций. Также в проекте реализованы нахождения погрешностей для каждого метода каждой функции. Так например, для нахождения ошибки в прямом нахождении синуса в сравнении с встроенной функцией, используется формула: , где sinf(x) – это стандартная функция нахождения синуса, встроенная в “math.h”, а sin1(x) – это реализованная в проекте функция нахождения синуса прямым разложения этой функции по Тейлору.

# Результаты экспериментов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Функция** | **Аргумент** | **Прямой подсчет** | **Обратный подсчет** | **Встроенная функция** | **Ошибка прямого подсчета** | **Ошибка обратного подсчета** |
| **Функция** | 1,56 | 0.9999417067 | 0.9999420643 | 0.9999417067 | 0.0000000000 | 0.0000003576 |
|  | 5 | -0.9589242935 | -0.9589252472 | -0.9589242935 | 0.0000000000 | 0.0000009945 |
|  | 15 | 0.6440424323 | 0.6723022461 | 0.6502878666 | 0.0096041067 | 0.0338532850 |
| **cos(x)** | 1,56 | 0.0107962126 | 0.0107960105 | 0.0107961744 | 0.0000035368 | 0.0000151825 |
|  | 5 | 0.2836621106 | 0.2836627960 | 0.2836622000 | 0.0000003152 | 0.0000021013 |
|  | 15 | -0.7550314665 | -0.7394332886 | -0.7596879005 | 0.0061294041 | 0.0266617537 |
| **ln(x)** | 0,3 | 0.2623642087 | 0.2230017781 | 0.2623642385 | 0.0000001136 | 0.1500298232 |
|  | 0,55 | 0.4382549524 | 0.3115798533 | 0.4382548928 | 0.0000001360 | 0.2890442014 |
|  | 0,999 | 0.6970975399 | 0.3183467984 | 0.6926470995 | 0.0064252638 | 0.5403910875 |
| **exp(x)** | 0,3 | 1.3498586416 | 1.3498587608 | 1.3498588800 | 0.0000001766 | 0.0000001766 |
|  | 10 | 22026.46875 | 22026.46875 | 22026.46484375 | 0.0000001773 | 0.0000001773 |
|  | 15 | 3269017.5 | 3269016.0 | 3269017.25 | 0.0000000765 | 0.0000000765 |

**\*Желтым помечены наименьшие погрешности для каждого эксперимента**

# Заключение

В результате проделанной работы изучены и реализованы алгоритмы вычисления значений функций синуса, косинуса, логарифма, экспоненты. Была найдена точность вычисления функций для двух методов (прямого и обратного подсчета функции). Вопреки ожиданиям, получилось, что метод прямого подсчета оказался точнее. Это можно объяснить тем, что в моем решении для нахождения значения функции методом обратного подсчета, сначала ищется последний элемент в разложении прямым подсчетом, а потом от этого элемента ищется предыдущие обратным подсчетом. Вероятно, можно повысить точность, проходя до меньшего элемента, при поиске начального для метода обратного подсчета.

# Приложение

#include <stdio.h>  
#include <stdlib.h>  
#define **\_USE\_MATH\_DEFINES**#include <math.h>  
  
const int N = 101;  
  
  
float mysin(float x, int ind, float ap) { *//ap = a past == a i-1* float ai;  
 ai = (-1) \* x \* x \* ap / (2 \* (float)ind \* (2 \* (float)ind + 1));  
 return ai;  
}  
  
float mysinR(float x, int ind, float an) {  
 float ai;  
 ai = (-an) \* (2 \* (float)ind + 2) \* (2 \* (float)ind + 3) / pow(x, 2);  
 return ai;  
}  
  
float mycos(float x, int ind, float ap) { *//ap = a past == a i-1* float ai;  
 ai = (-1) \* x \* x \* ap / (2 \* (float)ind \* (2 \* (float)ind - 1));  
 return ai;  
}  
  
float mycosR(float x, int ind, float an) {  
 float ai;  
 ai = (-1) \* ((2 \* (float)ind) + 1) \* ((2 \* (float)ind) + 2) \* an / (x \* x);  
 return ai;  
}  
  
float myexp(float x, int ind, float ap) { *//ap = a past == a i-1* float ai;  
 ai = x \* ap / (float)ind;  
 return ai;  
}  
  
float myexpR(float x, int ind, float an) {  
 float ai;  
 ai = (float)(ind + 1) \* an / x;  
 return ai;  
}  
  
float myln(float x, int ind, float ap) { *//ap = a past == a i-1* float ai;  
 ai = (-1) \* x \* (float)ind \* ap / (float)(ind + 1);  
 return ai;  
}  
  
float mylnR(float x, int ind, float an) {  
 float ai;  
 ai = (-1) \* (float)(ind + 1) \* an / ((float)(ind)\*x);  
 return ai;  
}  
  
float sumstr(float (\*anything)(float, int, float), float x, float ap) {  
 float s = ap;  
 float ai;  
 for (int i = 1; i < N; i++) {  
 ai = anything(x, i, ap);  
 ap = ai;  
 s += ai;  
 }  
 return s;  
}  
  
float sumback(float (\*revers)(float, int, float), float (\*straight)(float, int, float), float x, float ap) {  
 float ai;  
 float an;  
 int ind\_a;  
 for (int i = 1; i < N; i++) {  
 ai = straight(x, i, ap);  
 if (fabsf(ai) - pow(10, -28) < 0) {  
 break;  
 }  
 else {  
 an = ai;  
 ind\_a = i;  
 }  
 ap = ai;  
 }  
  
 float s = an;  
 for (int i = ind\_a - 1; i > 0; i--) {  
 ai = revers(x, i, an);  
 an = ai;  
 s += ai;  
 }  
 return s;  
}  
  
int main() {  
 float sin0, cos0, exp0, ln0, x, sinl, cosl, expl, lnl, sin1, sin2, cos1, cos2, exp1, exp2, ln1, ln2;  
 int c; *//choice* printf("Which fun you want to use?\n1.sin(x)\n2.cos(x)\n3.exp(x)\n4.ln(x)\n");  
 scanf("%d", &c);  
 printf("Enter x: ");  
 scanf("%f", &x);  
 sin0 = x;  
 cos0 = 1;  
 exp0 = 1;  
 ln0 = x;  
 float err1;  
 float err2;  
 switch (c) {  
 case 1:  
 if (x == 0) {  
 x = 2 \* (float)**M\_PI**;  
 }  
 sin1 = sumstr(mysin, x, sin0);  
 sin2 = sumback(mysinR, mysin, x, sin0) + sin0;  
 printf("sin1 = %.10f\n", sin1);  
 printf("sin2 = %.10f\n", sin2);  
 printf("sin (standart) = %.10f\n", sinf(x));  
 err1 = fabsf(sinf(x) - sin1) / fabsf(sinf(x));  
 err2 = fabsf(sinf(x) - sin2) / fabsf(sinf(x));  
 printf("Error for straight: %.10f\nError for back: %.10f", err1, err2);  
 break;  
 case 2:  
 if (x == 0) {  
 x = 2 \* (float)**M\_PI**;  
 }  
 cos1 = sumstr(mycos, x, cos0);  
 cos2 = sumback(mycosR, mycos, x, cos0) + cos0;  
 printf("cos1 = %f.10\n", cos1);  
 printf("cos2 = %f.10\n", cos2);  
 printf("cos (standart) = %.10f\n", cosf(x));  
 fabsf(cosf(x) - cos1) / fabsf(cosf(x));  
 err1 = fabsf(cosf(x) - cos1) / fabsf(cosf(x));  
 err2 = fabsf(cosf(x) - cos2) / fabsf(cosf(x));  
 printf("Error for straight: %.10f\nError for back: %.10f", err1, err2);  
 break;  
 case 3:  
 if (x == 0) {  
 x = 0;  
 }  
 exp1 = sumstr(myexp, x, exp0);  
 exp2 = sumback(myexpR, myexp, x, exp0) + exp0;  
 printf("exp1 = %.10f\n", exp1);  
 printf("exp2 = %.10f\n", exp2);  
 printf("exp (standart) = %.10f\n", expf(x));  
 err1 = fabsf(expf(x) - exp1) / fabsf(expf(x));  
 err2 = fabsf(expf(x) - exp1) / fabsf(expf(x));  
 printf("Error for straight: %.10f\nError for back: %.10f", err1, err2);  
 break;  
 case 4:  
 if (x <= 0) {  
 printf("ERROR!");  
 break;  
 }  
 ln1 = sumstr(myln, x, ln0);  
 ln2 = sumback(mylnR, myln, x, ln0) + ln0;  
 printf("ln1 = %f\n", ln1);  
 printf("ln2 = %f\n", ln2);  
 printf("ln (standart) = %.10f\n", logf(x+1));  
 err1 = fabsf(logf(x + 1) - ln1) / fabsf(logf(x + 1));  
 err2 = fabsf(logf(x + 1) - ln2) / fabsf(logf(x + 1));  
 printf("Error for straight: %.10f\nError for back: %.10f", err1, err2);  
 break;  
 }  
}