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**

**«Подсчёт математический функций»**

**Выполнил**:

студент группы 3822Б1ПМ-1

Ковалев К.И.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2023

**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)

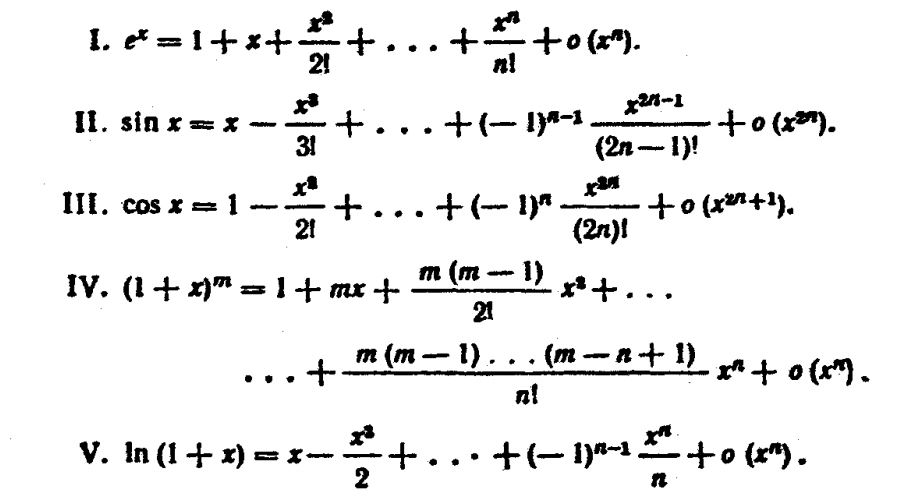
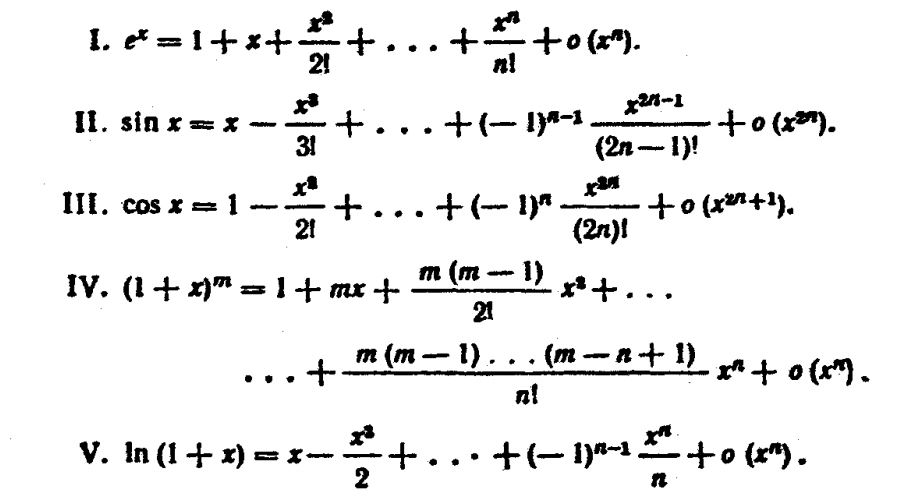
[Приложение 10](#_Toc26962569)

# Постановка задачи

Поставлена задача: реализовать подсчёт математических функций: **sin(x), cos(x), e^x, ln(1+x)** *(для -1<x<1)*для аргумента, вводимого пользователем, типа данных **float**. Подсчёт функций должен осуществляться через разложение в ряды Маклорена с количеством слагаемых, вводимым пользователем. Для каждой функции необходимо реализовать прямое (от первого элемента до последнего) и обратное (от последнего до первого) суммирование. Необходимо описать функционирование программы, убедиться в верности полученных результатов, сравнить точность прямого и обратного суммирования.

# Метод решения

Разложения в ряд Маклорена выбранных функций имеют вид:



*Рис. 1: разложения в ряд Маклорена выбранных функций*

Таким образом, каждое следующее слагаемое s в ряду может быть выражено через предыдущее и это выражение имеет вид:

* для **sin(x): si+1= - si (x2/(2i(2i+1)));**
* для **cos(x): si+1= - si (x2/(2i(2i-1)));**
* для **e^x: si+1= si (x/i);**
* для **ln(1+x): si+1=- si (x\*i/(i+1));**

Зная зависимость следующего слагаемого от предыдущего, нетрудно

произвести подсчёт ряда из необходимого n количества элементов.

Ввиду непоследовательного суммирования (сначала большие величины, затем малые) переполнение типа данных при прямом порядке суммирования (i=1, 2, 3 … n) может привести к потере значащих разрядов и, соответственно, снижению точности. С целью сохранения большей точности реализуется и обратный порядок суммирования: сначала подсчитываются все значения слагаемых ряда в прямом порядке, последнее слагаемое передаётся в функцию обратного суммирования и для него по формулам:

* для **sin(x): si= - si+1 ((2i(2i+1))/x2);**
* для **cos(x): si= - si+1 ((2i(2i-1))/ x2);**
* для **e^x: si= si+1 (i/x);**
* для **ln(1+x): si=- si+1 ((i+1)/(x\*n));**

вычисляются предшествующие элементы. Элементы суммируются.

В функции обратного суммирования необходимо реализовать сравнение i-го элемента с величиной 10-8, т.к. если n-е слагаемое окажется меньше 10-8 или представлено нулём, все слагаемые ряда (получаемые только умножением и делением на величины) не занулялись и ряд имел ненулевое (максимальное по точности для этого типа данных и этого алгоритма подсчёта) значение.

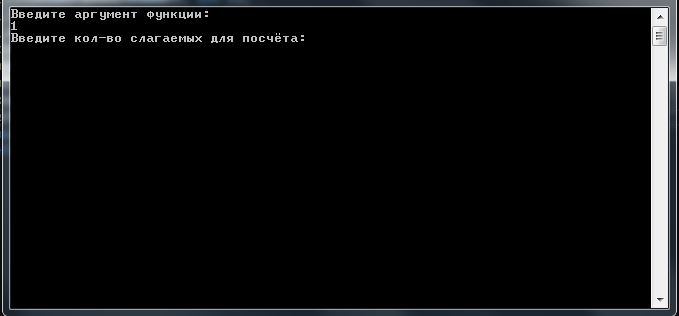
# Руководство пользователя

Интерфейс программы интуитивно понятен пользователю, и всё же следует рассмотреть его отдельно.



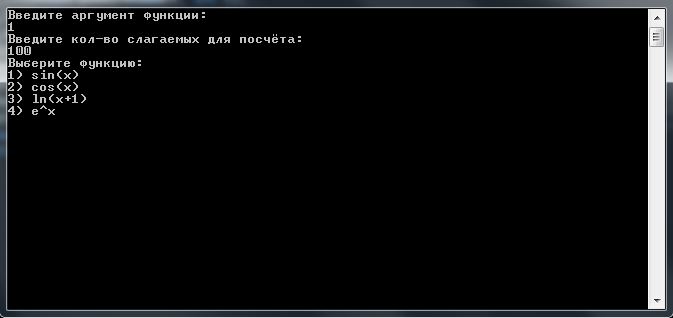
*Рис. 2: ввод аргумента функции.*

1. При запуске программы пользователь должен ввести с клавиатуры аргумент функции.



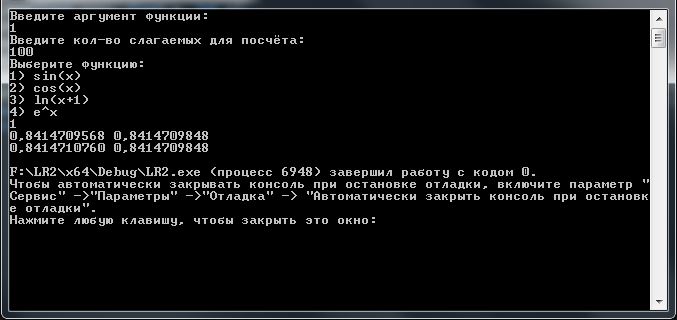
*Рис. 3: выбор количества слагаемых.*

1. После этого на экран будет выведен запрос количества слагаемых в разложении функции. Далее пользователю предлагается выбрать математическую функцию. Для этого необходимо ввести номер, под которым в списке располагается выбранная функция.



*Рис. 4: выбор дальнейшего действия.*

1. Будет произведён подсчёт прямым и обратным суммированием. Результаты будут выведены на экран в разных строках: сначала результат для прямого суммирования, затем – для обратного. В каждой из строк вывода после значения, рассчитанного программой будет выведено значение, рассчитанное стандартной функцией из библиотеки math.h.



*Рис. 5: меню после вывода значений функции.*

1. После вывода после вывода значений программа завершится

# Описание программной реализации

Программа состоит из четырёх файлов – **Main.cpp**, в котором реализована основная функция **int main()**, **Prototypes.h,** который содержит прототипы функций подсчёта слагаемых, функций прямого и обратного суммирования, **Summands.cpp,** который содержит реализацию функций подсчёта слагаемого и **Sums.cpp,** который содержит реализацию функций суммирования**.**

1. **float slag\_sin(float prev, float x, int n) –** функция подсчёта следующего слагаемого в разложении функции **sin(x)**. Принимает значение предыдущего в разложении слагаемого и аргумент функции типа **float**, номер предыдущего слагаемого типа **int**. Возвращает значение слагаемого типа **float**. Аналогичная её функция - **float r\_slag\_sin(float last, float x, int n) –** функция подсчёта предыдущего слагаемого в ряду.
2. **float slag\_cos(float prev, float x, int n) -** функция подсчёта следующего слагаемого в разложении функции **cos(x)**. Принимает значение предыдущего в разложении слагаемого и аргумент функции типа **float**, номер предыдущего слагаемого типа **int**. Возвращает значение слагаемого типа **float.** Аналогичная её функция - **float r\_slag\_cos(float prev, float x, int n) –** функция подсчёта предыдущего слагаемого в ряду.
3. **float slag\_exp(float prev, float x, int n) -** функция подсчёта следующего слагаемого в разложении функции **e^x**. Принимает значение предыдущего в разложении слагаемого и аргумент функции типа **float**, номер предыдущего слагаемого типа **int**. Возвращает значение слагаемого типа **float.** Аналогичная её функция - **float r\_slag\_exp(float prev, float x, int n) –** функция подсчёта предыдущего слагаемого в ряду.
4. **float slag\_ln(float prev, float x, int n) -** функция подсчёта следующего слагаемого в разложении функции **ln(1+x)**. Принимает значение предыдущего в разложении слагаемого и аргумент функции типа **float**, номер предыдущего слагаемого типа **int**. Возвращает значение слагаемого типа **float.** Аналогичная её функция - **float r\_slag\_ln(float prev, float x, int n) –** функция подсчёта предыдущего слагаемого в ряду.
5. **float sum(float (\*slag)(float, float, int), float x, int quan) -** функция суммирования в прямом порядке. Принимает функцию подсчёта следующего слагаемого и аргумент типа **float,** количество слагаемых типа **int.**
6. **float r\_sum(float (\*r\_slag)(float, float, int), float x, float last, int quan) –** функция суммирования в обратном порядке. Принимает функцию подсчёта предыдущего слагаемого, аргумент и начальный элемент типа **float,** количество слагаемых типа **int.**
7. **float the\_last\_of\_us(float (\*slag)( float, float, int), float x, int\* quan) –** функция подсчёта ряда Маклорена, возвращающая n-е слагаемое ряда (тип **double)**, которое впоследствии передаётся в функцию обратного суммирования. Принимает функцию подсчёта следующего слагаемого и аргумент типа **float,** количество слагаемых типа **int**. В функции реализовано сравнение **i**-го слагаемого с величиной 10-8. Если **i**-е слагаемое меньше этой величины, возвращается (**i**-1)-е слагаемое и значение quan изменяется на (**i**-1). Это необходимо для того, чтобы в случае, если n-е слагаемое окажется меньше 10-8 или представлено нулём, все слагаемые ряда (получаемые только умножением и делением на величины) не занулялись и ряд имел ненулевое (максимальное по точности для этого типа данных и этого алгоритма подсчёта) значение.

# Подтверждение корректности

Для подтверждения корректности реализован подсчёт абсолютной и относительной погрешности. В качестве эталонных используются функции **sinf(x), cosf(x), powf(M\_E,x), logf(1+x)**  стандартной библиотеки **math.h**. Значения погрешностей выводятся на экран. При вводе аргументов, по абсолютной величине меньших 15, выводится не менее 4-х значащих знаков после запятой. Более подробный анализ результатов приведён в разделе *«Результаты экспериментов»*

# Результаты экспериментов

Был проведён ряд экспериментальных запусков программы, в ходе которых одинаковый объем данных сортировался разными алгоритмами, зафиксировано среднее время работы программы:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Аргумент функции | Вывод:  Значение  Абсолютная погрешность  Относительная погрешность | | | | | |
| Sin(x) | | Cos(x) | | E^x | |
| ПС | ОС | ПС | ОС | ПС | ОС |
| 1 | *0,84147096*  *0,00000003*  *0,00000003* | *0,84147108*  *0,00000009*  *0,00000011* | *0,54030228*  *0,00000003*  *0,00000005* | *0,54030240*  *0,00000009*  *0,00000017* | *2,71666694*  *0,00161481*  *0,00059409* | *2,71666670*  *0,00161513*  *0,00059417* |
| 2 | *0,90929741*  *0,00000002*  *0,00000002* | *0,90929759*  *0,00000016*  *0,00000017* | *-0,41614681*  *0,00000002*  *0,00000005* | *-0,41614687*  *0,00000004*  *0,00000009* | *7,38095284*  *0,00810289*  *0,00109666* | *7,38095236*  *0,00810374*  *0,00109672* |
| 3 | *0,14112009*  *0,00000008*  *0,00000059* | *0,14111996*  *0,00000005*  *0,00000036* | *-0,98999244*  *0,00000006*  *0,00000006* | *-0,98999286*  *0,00000036*  *0,00000036* | *20,00915337*  *0,07638168*  *0,00380291* | *20,00915146*  *0,07638546*  *0,00380301* |
| 5 | *-0,95892501*  *0,00000073*  *0,00000077* | *-0,95892477*  *0,00000050*  *0,00000052* | *0,28366211*  *0,00000007*  *0,00000026* | *0,28366256*  *0,00000037*  *0,00000131* | *148,11357117*  *0,29956055*  *0,00201861* | *148,11357117*  *0,29958794*  *0,00201861* |
| 7,5 | *0,93801582*  *0,00001584*  *0,00001689* | *0,93799686*  *0,00000311*  *0,00000332* | *0,34662843*  *0,00000689*  *0,00001988* | *0,34663022*  *0,00000510*  *0,00001472* | *1799,71020508*  *8,33178711*  *0,00460841* | *1799,71044922*  *8,33196545*  *0,00460828* |
| 10 | *-0,54412025*  *0,00009914*  *0,00018224* | *-0,54418850*  *0,00016739*  *0,00030769* | *-0,83918649*  *0,00011496*  *0,00013701* | *-0,83899736*  *0,00007417*  *0,00008839* | *21950,38085938*  *76,0781250*  *0,00345425* | *21950,38281250*  *76,08298492*  *0,00345416* |
| 13 | *0,41788149*  *0,00228555*  *0,00543962* | *0,41893959*  *0,00122745*  *0,00292133* | *0,90618533*  *0,00126145*  *0,00139011* | *0,91136944*  *0,00392266*  *0,00432275* | *440656,18750*  *1757,031250*  *0,00397186* | *440656,093750*  *1757,29821777*  *0,00397207* |
| 15 | *0,67294276*  *0,02265492*  *0,03483829* | *0,67230415*  *0,02201631*  *0,03385626* | *-0,75503147*  *0,00465645*  *0,00612942* | *-0,74000537*  *0,01968254*  *0,02590872* | *3258190,750*  *10825,250*  *0,00331189* | *3258189,50*  *10827,87207031*  *0,00331227* |

*Табл. 1: результаты экспериментов*

*Примечание:*

1. *в таблице «*-*» означает, что время сортировки этого объёма данных велико, а ранее собранных данных достаточно для того, чтобы сделать вывод о вычислительной сложности сортировки;*
2. *значение «0,000000» означает, что время сортировки этого объёма данных мало для того, чтобы быть измеренным использованным методом;*

Вычислим фактическую вычислительную сложность алгоритмов сортировки. Для этого пошагово будем делить время работы конкретной сортировки на **N**, **N\*log2N**, **N2** и строить графики зависимости полученной величины от размера сортируемого массива:

Отметим, что для поразрядной сортировки график зависимости **T/N** от **N** стремится к константе, что свидетельствует о том, что вычислительная сложность поразрядной сортировки в среднем – **O(N)**. Это подтверждает указанную в главе «Метод решения» вычислительную сложность.

Зависимость **T/N\*log2N** от **N** для сортировки расчёской стремится к константе. Это подтверждает среднюю вычислительную сложность **O(N\*logN)**.

Зависимость **T/N2** от **N** для сортировки пузырьком стремится к константе. Это подтверждает среднюю вычислительную сложность **O(N2)**.

# Заключение

Итак, написана программа, реализующая сортировку массива случайных чисел типа **double** по возрастанию тремя различными алгоритмами. Интерфейс, эффективность и скорость работы программы удовлетворяют обозначенным условиям. Описаны принципы функционирования программы, проведена серия экспериментов, подтверждающих соответствие программы теоретическим данным.

# Приложение

int main()

{

setlocale(LC\_ALL, "Rus");

int action, quan;

float x, rez1, rez2;

printf("Введите аргумент функции: \n");

scanf\_s("%f", &x);

printf("Введите кол-во слагаемых для посчёта: \n");

scanf\_s("%d", &quan);

printf("Выберите функцию: \n1) sin(x)\n2) cos(x)\n3) ln(x+1)\n4) e^x\n");

scanf\_s("%d", &action);

switch (action)

{

case 1:

{

printf("%.10f %.10f \n", sum(slag\_sin, x, quan), sin(float(x)));

printf("%.10f %.10f \n", r\_sum(r\_slag\_sin, x, the\_last\_of\_us(slag\_sin, x, &quan), quan), sin(float(x)));

printf("%.10f %.10f \n", sum(slag\_sin, x, quan), sin(x));

printf("%.10f %.10f \n", r\_sum(r\_slag\_sin, x, the\_last\_of\_us(slag\_sin, x, &quan), quan), sin(x));

break;

}

case 2:

{

printf("%.10f %.10f \n", sum(slag\_cos, x, quan), cos(float(x)));

printf("%.10f %.10f \n", r\_sum(r\_slag\_cos, x, the\_last\_of\_us(slag\_cos, x, &quan), quan), cos(float(x)));

printf("%.10f %.10f \n", sum(slag\_cos, x, quan), cos(x));

printf("%.10f %.10f \n", r\_sum(r\_slag\_cos, x, the\_last\_of\_us(slag\_cos, x, &quan), quan), cos(x));

break;

}

case 3:

{

printf("%.10f %.10f \n", sum(slag\_ln, x, quan), log(float(x) + 1));

printf("%.10f %.10f \n", r\_sum(r\_slag\_ln, x, the\_last\_of\_us(slag\_ln, x, &quan), quan), log(float(x) + 1.0f));

printf("%.10f %.10f \n", sum(slag\_ln, x, quan), log(x + 1.0f));

printf("%.10f %.10f \n", r\_sum(r\_slag\_ln, x, the\_last\_of\_us(slag\_ln, x, &quan), quan), log(x + 1.0f));

break;

}

case 4:

{

printf("%.10f %.10f \n", sum(slag\_exp, x, quan), pow(M\_E, float(x)));

printf("%.10f %.10f \n", r\_sum(r\_slag\_exp, x, the\_last\_of\_us(slag\_exp, x, &quan), quan), pow(M\_E, float(x)));

printf("%.10f %.10f \n", sum(slag\_exp, x, quan), pow(M\_E, x));

printf("%.10f %.10f \n", r\_sum(r\_slag\_exp, x, the\_last\_of\_us(slag\_exp, x, &quan), quan), pow(M\_E, x));

}

}

return 0;

}

float r\_sum(float (\*r\_slag)(float, float, int), float x, float last, int quan)

{

float rez = 0;

for (; quan > 0; quan--)

{

rez += last;

last = r\_slag(last, x, quan - 1);

}

return rez;

}

float sum(float (\*slag)(float, float, int), float x, int quan)

{

float rez = 0, prev = 0;

for (int i = 1; i <= quan; i++)

{

prev = slag(prev, x, i - 1);

rez += prev;

}

return rez;

}

float the\_last\_of\_us(float (\*slag)(float, float, int), float x, int\* quan)

{

float prev = 0;

for (int i = 1; i <= \*quan; i++)

{

if (fabsf(slag(prev, x, i - 1)) <= 1E-8)

{

\*quan = i - 1;

return prev;

}

prev = slag(prev, x, i - 1);

}

return prev;

}