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**

**«Суммирование рядов в алгебраических функциях»**

**Выполнил**:

студент группы 3822Б1ПМ1

Леонтьев Н.С.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2023

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 6](#_Toc26962564)

[Описание программной реализации 7](#_Toc26962565)

[Подтверждение корректности 9](#_Toc26962566)

[Результаты экспериментов 10](#_Toc26962567)

[Заключение 16](#_Toc26962568)

[Приложение 17](#_Toc26962569)

# Постановка задачи

В лабораторной работе необходимо реализовать функции sin(x), cos(x), ln(1+x) и exp(x), которые работают с типом данных float и вычисляют значение функции в точке через ряд Тейлора. Должна быть реализована прямая и обратная сумма, со сравнением точности: несколько значений функций необходимо сравнить с более точными.

Исходный код программы выложить на GitHub, предоставить в данном отчёте руководство пользователю, описание программной реализации. Сделать вывод о написании алгебраических функций через ряды Тейлора.

# Метод решения

Для разработки функций необходимо было вывести рекуррентные формулы для каждого следующего слагаемого:

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

Выражение через , в свою очередь, приведёт к делению двух чисел с плавающей запятой, отчего оно не будет использовано в функциях. Прямая сумма подразумевает сложение от первого элемента до последнего, а обратная – сложение от последнего к первому. Соответственно сохранение каждого слагаемого в массив из n элементов хоть и потребляет память, но приведёт к одинаковой сложности алгоритма в прямой и обратной сумме и большей точности в обратной. Стоит также помнить, что первое слагаемое синуса и логарифма – x, косинуса и экспоненты – 1.

# Руководство пользователя

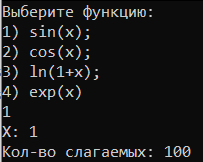
Необходимо запустить программу, код которой представлен внутри репозитория GitHub. Она запросит ввод от пользователя:

рис. 1. Ввод

После выбора функции, аргумента и кол-ва слагаемых программа выведет значение через прямую сумму, обратную, и встроенной функцией. Помимо этого, будет выведена погрешность прямой, обратной суммы относительно встроенной.

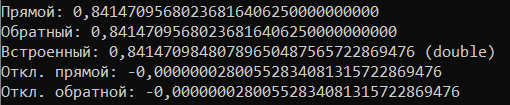


рис. 2. Итог

# Описание программной реализации

Программа представлена в виде трёх файлов: funcs.h, funcs.cpp, main.cpp.

Funcs.h содержит прототипы функций, а также директивы #include, funcs.cpp содержит реализацию функций, main.cpp же основной код при запуске программы.

Для начала были реализованы короткие функции вычисления следующего слагаемого, они лишь используют формулы, выведенные ранее. Они обязаны иметь одинаковые аргументы, чтобы было возможным одним указателем сослаться на любую. Соответствующие прототипы были написаны в funcs.h.

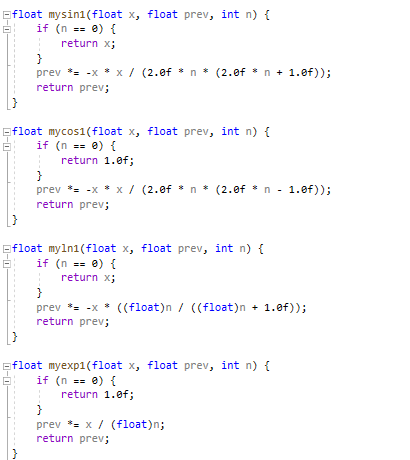


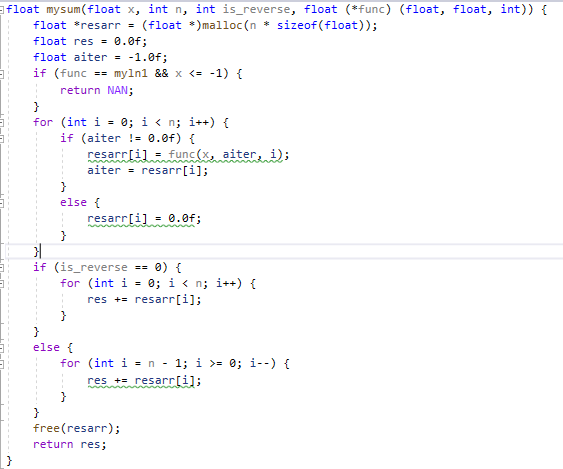
рис. 3. Функции следующего слагаемого

n – номер слагаемого

(funcs.cpp)

Далее была написана функция вычисления прямой/обратной суммы слагаемых ряда Тейлора. Для этого был выделен динамический float массив размера n, была встроена проверка на x <= -1 при функции ln(1+x). При x > 1 же функция некорректно рассчитывает, но точно не возвращает NAN.

Под конец функция считает сумму всех слагаемых, либо начиная с начала, либо начиная с конца.



Использование всех написанных функций в main() сделало возможным создание программы, подходящей задачам лабораторной работы.

рис. 4. Функция суммирования

(funcs.cpp)

# Подтверждение корректности

Для подтверждения корректности вычисленных значений применялось их сравнение со значениями встроенной функции sin(x). Она возвращает double, но так как значения float из наших функций легко привести к double без потери данных, будет логично сравнивать наши значения с более точными, ведь даже в sinf(float x) существует погрешность, не сильно отличная от нашей.

# Результаты экспериментов

Необходимо узнать, насколько значения наших функций совпадают с реальными значениями (используя встроенные функции).

Стоит заметить, что начиная с определённого значения, наш sin(x) перестаёт выдавать точные значения из-за погрешности, накапливающейся в результате округления. Аналогично с cos(x). Наглядно это видно на графике ниже:

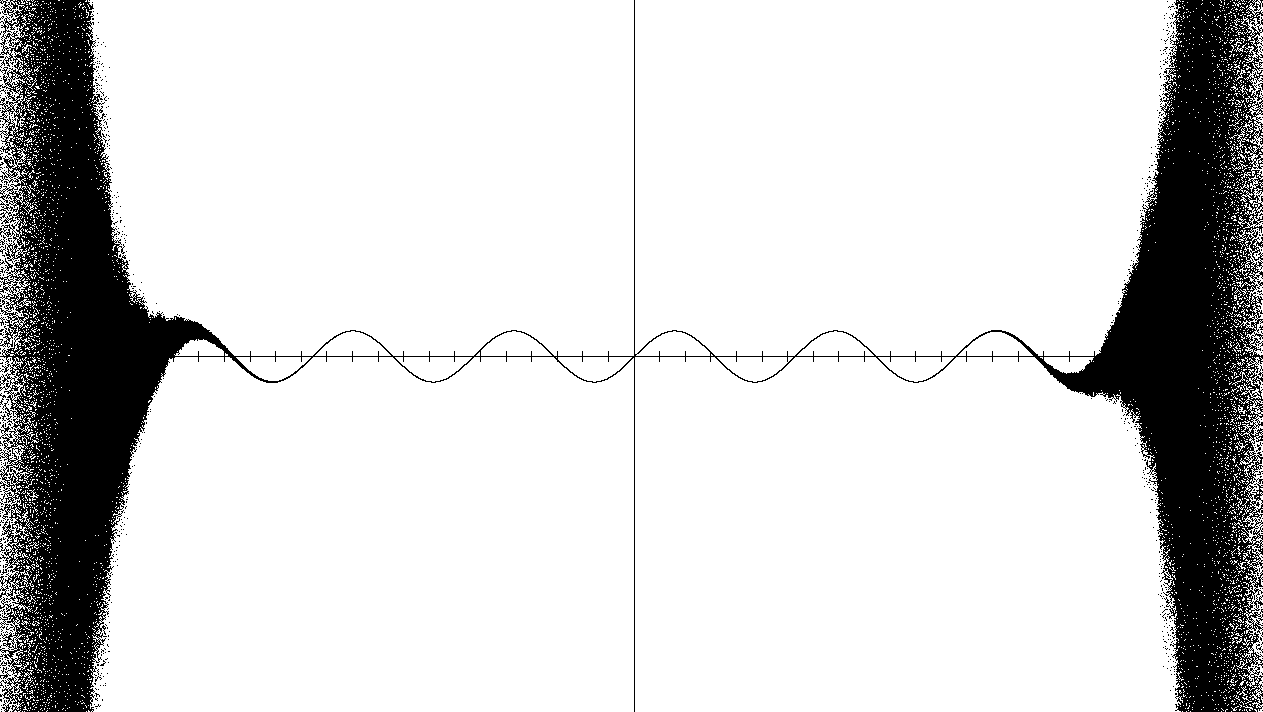


рис. 5. График по точкам

Следовательно, лучше всего выбирать значения x примерно от -12 до 12.

Ln(x) в свою очередь придётся выбирать от -1 до 1.

**SIN(X)**

1. x=-6

Прямой: 0,27941766381263732910156250000000

Обратный: 0,27941894531250000000000000000000

Встроенный: 0,27941549819892586015157576184720 (double)

Откл. прямой: 0,00000216561371146894998673815280

Откл. обратной: 0,00000344711357413984842423815280

1. x=-1

Прямой: -0,84147095680236816406250000000000

Обратный: -0,84147095680236816406250000000000

Встроенный: -0,84147098480789650487565722869476 (double)

Откл. прямой: 0,00000002800552834081315722869476

Откл. обратной: 0,00000002800552834081315722869476

1. x=5

Прямой: -0,95892477035522460937500000000000

Обратный: -0,95892524719238281250000000000000

Встроенный: -0,95892427466313845396683746002964 (double)

Откл. прямой: -0,00000049569208615540816253997036

Откл. обратной: -0,00000097252924435853316253997036

1. x=8

Прямой: 0,98934167623519897460937500000000

Обратный: 0,98935699462890625000000000000000

Встроенный: 0,98935824662338178736575855509727 (double)

Откл. прямой: -0,00001657038818281275638355509727

Откл. обратной: -0,00000125199447553736575855509727

1. x=-12

Прямой: 0,53604632616043090820312500000000

Обратный: 0,53588867187500000000000000000000

Встроенный: 0,53657291800043493967820040779770 (double)

Откл. прямой: -0,00052659184000403147507540779770

Откл. обратной: -0,00068424612543493967820040779770

**COS(X)**

1. x=-7:

Прямой: 0,75390493869781494140625000000000

Обратный: 0,75390625000000000000000000000000

Встроенный: 0,75390225434330460085874392461847 (double)

Откл. прямой: 0,00000268435451034054750607538153

Откл. обратной: 0,00000399565669539914125607538153

1. x=-2

Прямой: -0,41614681482315063476562500000000

Обратный: -0,41614675521850585937500000000000

Встроенный: -0,41614683654714240690353221907571 (double)

Откл. прямой: 0,00000002172399177213790721907571

Откл. обратной: 0,00000008132863654752853221907571

1. x=4

Прямой: -0,65364366769790649414062500000000

Обратный: -0,65364360809326171875000000000000

Встроенный: -0,65364362086361194048578227011603 (double)

Откл. прямой: -0,00000004683429455365484272988397

Откл. обратной: 0,00000001277035022173578227011603

1. x=9

Прямой: -0,91113346815109252929687500000000

Обратный: -0,91113281250000000000000000000000

Встроенный: -0,91113026188467693966543947681203 (double)

Откл. прямой: -0,00000320626641558963143552318797

Откл. обратной: -0,00000255061532306033456052318797

1. x=12

Прямой: 0,84379321336746215820312500000000

Обратный: 0,84375000000000000000000000000000

Встроенный: 0,84385395873249213760658449245966 (double)

Откл. прямой: -0,00006074536502997940345949245966

Откл. обратной: -0,00010395873249213760658449245966

**LN(1+X)**

1. x=-0,03

Прямой: -0,03045920841395854949951171875000

Обратный: -0,03045920655131340026855468750000

Встроенный: -0,03045920679341756337477065130770 (double)

Откл. прямой: -0,00000000162054098612474106744230

Откл. обратной: 0,00000000024210416310621596380770

1. x=0,01

Прямой: 0,00995033048093318939208984375000

Обратный: 0,00995033048093318939208984375000

Встроенный: 0,00995033063186370921038292891581 (double)

Откл. прямой: -0,00000000015093051981829308516581

Откл. обратной: -0,00000000015093051981829308516581

1. x=0,05

Прямой: 0,04879016801714897155761718750000

Обратный: 0,04879016429185867309570312500000

Встроенный: 0,04879016487901110393776704654556 (double)

Откл. прямой: 0,00000000313813786761985014095444

Откл. обратной: -0,00000000058715243084206392154556

**EXP(X)**

1. x=1

Прямой: 2,71828198432922363281250000000000

Обратный: 2,71828174591064453125000000000000

Встроенный: 2,71828182845904509079559829842765 (double)

Откл. прямой: 0,00000015587017854201690170157235

Откл. обратной: -0,00000008254840055954559829842765

1. x=2

Прямой: 7,38905668258666992187500000000000

Обратный: 7,38905620574951171875000000000000

Встроенный: 7,38905609893065040694182243896648 (double)

Откл. прямой: 0,00000058365601951493317756103352

Откл. обратной: 0,00000010681886131180817756103352

1. x=-2

Прямой: 0,13533526659011840820312500000000

Обратный: 0,13533520698547363281250000000000

Встроенный: 0,13533528323661270231781372785917 (double)

Откл. прямой: -0,00000001664649429411468872785917

Откл. обратной: -0,00000007625113906950531372785917

Из 16 выполнений:

7 – прямая точнее обратной

7 – обратная точнее прямой

2 – прямая и обратная равны

# Заключение

В итоге поставленные задачи были выполнены. Были написаны все необходимые функции (код см. в Приложении), описаны взаимодействия программы с пользователем, структура программы, способ проверки корректности, приведены результаты экспериментов. В итоге выяснилось, что при типе данных float при разных аргументах может быть точнее и прямая, и обратная сумма.

В пределах приведённых выше промежутков значения вычисляются с точностью до 3-4 знака после запятой. Я выяснил, что работать с числами с плавающей запятой не так уж просто.

Цели и задачи лабораторной работы выполнены.

# Приложение

https://github.com/NeiroYT/mp1-3822B1PM1

float mysin1(float x, float prev, int n) {

if (n == 0) {

return x;

}

prev \*= -x \* x / (2.0f \* n \* (2.0f \* n + 1.0f));

return prev;

}

float mycos1(float x, float prev, int n) {

if (n == 0) {

return 1.0f;

}

prev \*= -x \* x / (2.0f \* n \* (2.0f \* n - 1.0f));

return prev;

}

float myln1(float x, float prev, int n) {

if (n == 0) {

return x;

}

prev \*= -x \* ((float)n / ((float)n + 1.0f));

return prev;

}

float myexp1(float x, float prev, int n) {

if (n == 0) {

return 1.0f;

}

prev \*= x / (float)n;

return prev;

}

float mysum(float x, int n, int is\_reverse, float (\*func) (float, float, int)) {

float \*resarr = (float \*)malloc(n \* sizeof(float));

float res = 0.0f;

float aiter = -1.0f;

if (func == myln1 && x <= -1) {

return NAN;

}

for (int i = 0; i < n; i++) {

if (aiter != 0.0f) {

resarr[i] = func(x, aiter, i);

aiter = resarr[i];

}

else {

resarr[i] = 0.0f;

}

}

if (is\_reverse == 0) {

for (int i = 0; i < n; i++) {

res += resarr[i];

}

}

else {

for (int i = n - 1; i >= 0; i--) {

res += resarr[i];

}

}

free(resarr);

return res;

}