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**

**«Вычисление математических функций с использованием формулы Тейлора»**

**Выполнил**:

студент группы 3824Б1ПМ1

Тончаров В.В.

**Проверила**:

Бусько П.В.

Нижний Новгород

2024

**Содержание**

[Введение 3](#_Toc182419046)

[Постановка задачи 4](#_Toc182419047)

[Руководство пользователя 5](#_Toc182419048)

[Описание программной реализации 6](#_Toc182419049)

[Результаты экспериментов](#_Toc182419050) 9

[Заключение](#_Toc182419051) 18

[Литература](#_Toc182419052) 19

[Приложение](#_Toc182419053) 20

# Введение

Данная лабораторная работа направлена на исследование свойств формулы Тейлора как численного метода, в том числе влияния метода суммирования последовательности на точность результата. Исследование поведения численных методов на простом примере позволяет улучшить понимание практических проблем при выполнении вычислений с плавающей точкой и способов их решения.

# Постановка задачи

Требовалось в контексте вычисления функций sin(𝑥), cos(𝑥), exp(𝑥) и ln(1 + 𝑥) в 64-битных числах с плавающей точкой binary64 по формуле Тейлора исследовать два способа суммирования последовательности членов: прямой и обратный, сравнив с эталонным значением, вычисляемым libm, и оценив, как отличается неточность двух методов суммирования.

# Руководство пользователя

Программа taylor вызывается либо без аргументов (в этом случае она печатает краткую справку), либо следующим образом: taylor xstart xend xstep series num\_terms, где:

* xstart – начальное значение 𝑥
* xend – конечное значение 𝑥
* xstep – шаг 𝑥
* series – идентификатор последовательности (функции)
* num\_terms – число членов (заведомо нулевые не учитываются)

Вывод программы аналогичен формату CSV, но разделителями столбцов и строк являются предназначенные для этого управляющие символы ASCII (36₈ – разделитель строк, 37₈ – разделитель столбцов). Для перевода в CSV следует передать вывод программы команде  
tr '\037\036' ',\n'.

# Описание программной реализации

Исходный код программы разделён на множество заголовочных файлов .h и исходных файлов .c. В корне находятся скрипт компиляции build.sh, вспомогательные файлы для clangd и скрипт для запуска вычислений на предопределённых диапазонах test.sh.

Файлы утилит util.{c,h} содержат по большей части отстранённый от предметной области программы код, включая представленные макросами сокращения, удобную конвертацию строк в числа и очистка строк от невидимых символов (пробелов и т.п.) в начале и в конце:

* const char \* trim\_nongraph\_start(const char \* str) – возвращает указатель на первый видимый символ строки, или на нуль-окончание, если их в строке нет.
* char \* trim\_nongraph\_end(char \* str) – вставляет в строку нуль-окончание так, чтобы символ перед ним был видимым, если это возможно; иначе вставляет нуль-окончание в самом начале строки.
* char \* trim\_nongraph(char \* str) – совмещает trim\_nongraph\_start и trim\_nongraph\_end.
* bool str\_to\_ulong(char \* str, unsigned long \* restrict out, const char \*\* restrict oerr) – конвертирует строку, содержащую десятичное неотрицательное целое в тип unsigned long, записывая результат или сообщение об ошибке по соответствующим указателям и возвращая успешность конвертации для удобного ветвления.
* bool str\_to\_double(char \* str, double \* restrict out, const char \*\* restrict oerr) – аналогично str\_to\_ulong, но для double.

Файлы randarr.{c,h} отвечают за заполнение тестовых массивов.

* inline unsigned align\_to\_lane(unsigned x) – возвращает наименьшее число, которое больше или равно x и делится на 4.

Файл tune.h содержит определение порога использования alloca и количества знаков после точки при выводе чисел с плавающей точкой.

Файлы argv0.{c,h} отвечают за глобальный указатель на строку, содержащую argv[0] для печати справки. Это выполняется отдельно от main при помощи .init\_array.

Файл double4.h содержит определение типа double4, используемый для векторизации.

Файлы output.{c,h} отвечают за вывод вычисленных значений.

* int print\_header(void) – печатает заголовок таблицы в стандартный вывод.
* int print\_per\_row(double x, double ref) – печатает часть строки, ширина которой не зависит от количества алгоритмов сложения.
* int print\_per\_sumfn(double sum) – печатает часть строки, которая повторяется один раз для каждого алгоритма сложения.

Файлы print\_usage.{c,h} отвечают за вывод справки при вызове программы с неправильным количеством аргументов.

* int print\_usage(FILE \* file) – печатает справку в переданный файл.
* int print\_header(void) – печатает заголовок таблицы в стандартный вывод.
* int print\_per\_row(double x, double ref) – печатает часть строки, ширина которой не зависит от количества алгоритмов сложения.
* int print\_per\_sumfn(double sum) – печатает часть строки, которая повторяется один раз для каждого алгоритма сложения.

Файлы series.{c,h} содержат вспомогательные определения для инъекции зависимости от функции и её разложению по формуле Тейлора и определения самих функций. В заголовочном файле они не упоминаются напрямую – доступ остальным файлам даётся только через массив series\_plural.

* unsigned short str2idx\_series(const char \* str) – возвращает индекс функции с данным идентификатором, или USHRT\_MAX, если такой функции нет.
* void fill\_terms(double x, const series \*restrict ser, double4 \*restrict terms, unsigned num\_terms) – заполняет массив заданным числом членов формулы Тейлора для заданной функции в данной точке.
* static void ser\_{sin,cos,exp,ln1p}(double x, double \* restrict out, unsigned num\_terms) – реализуют формулы Тейлора для конкретных функций.
* static double ln1p(double x) – функция ln(1 + 𝑥), реализованная на основе log из libm.

Файлы sumfn.{c,h} содержат вспомогательные определения для инъекции зависимости от алгоритма суммирования и определения самих алгоритмов. В заголовочном файле они не упоминаются напрямую – доступ остальным файлам даётся только через массив sum\_algs.

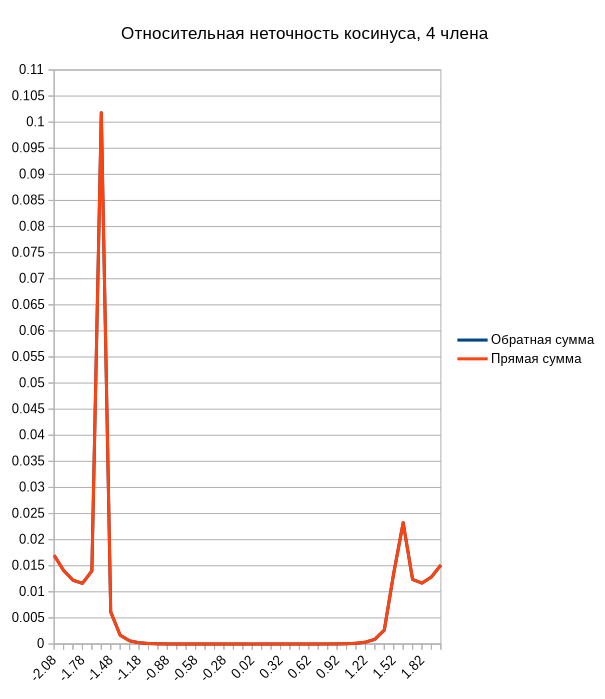
* unsigned short str2idx\_sum\_alg(const char \* str) – возвращает индекс алгоритма суммирования с данным идентификатором, или USHRT\_MAX, если такого алгоритма нет.
* double calc\_sum(const double4 \*restrict terms, unsigned num\_terms, const sum\_alg \*restrict sumfn) – суммирует заданный массив с заданным числом членом заданным алгоритмом суммирования.

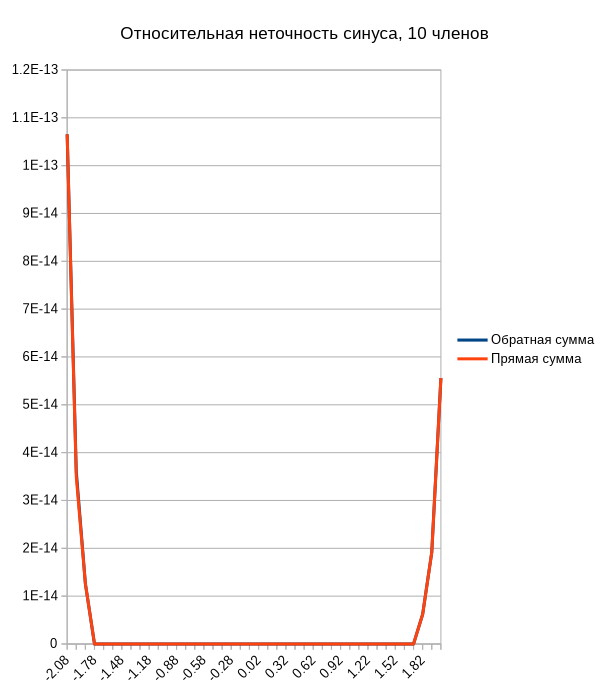
Файл main.c содержит точку входа программы, выполняющий роль высокоуровневой схемы действия программы код, и обработку ошибок верхнего уровня (печать ошибки и возврат ненулевого кода завершения).

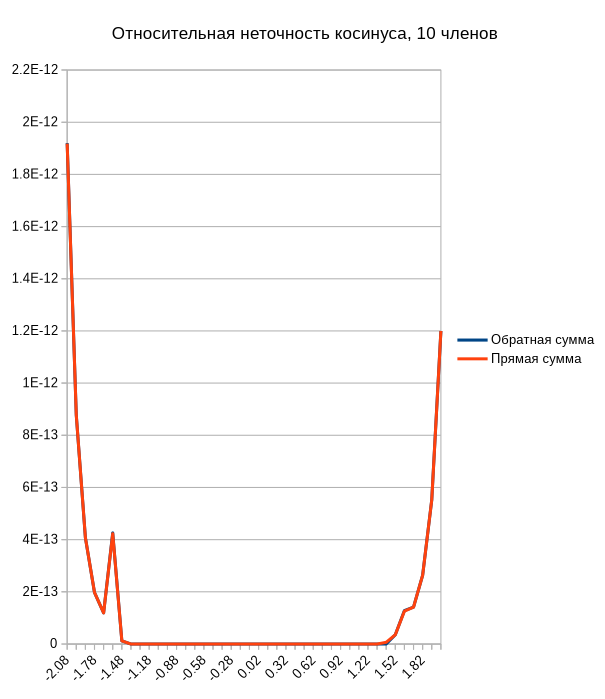
* int main(int argc, char \*\* argv) – точка входа программы.
* static int main\_inner(double xstart, double xend, double xstep, const series \*restrict ser, unsigned num\_terms) – тело программы, получающее от main обработанный ввод.
* static int for\_x(double x, const series \*restrict ser, const double4 \*restrict terms, unsigned num\_terms) – выполняется для каждого значения 𝑥; вычисляет эталонное значение по libm и разложение по формуле Тейлора, затем складывает разложение при помощи разных алгоритмов сложения и печатает строку таблицы вывода.

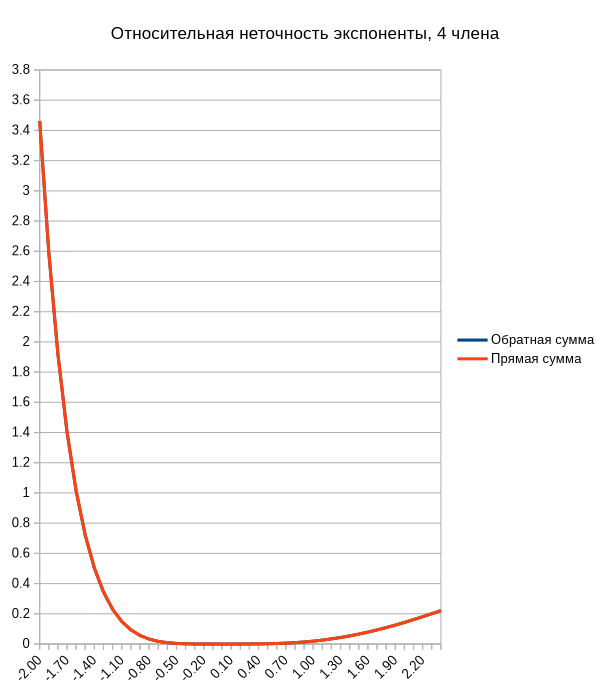
# Результаты экспериментов

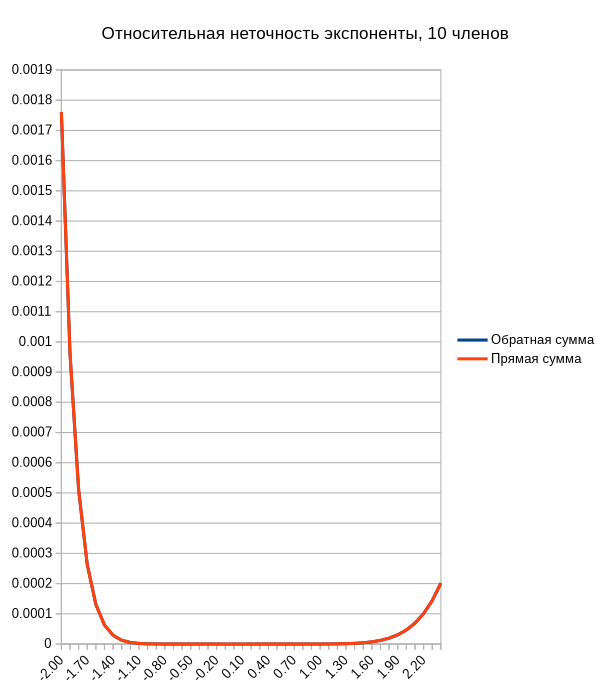
Относительная неточность разложения по формуле Тейлора на отрезке сходимости, рассчитанная по эталонному значению из libm, приведена ниже.

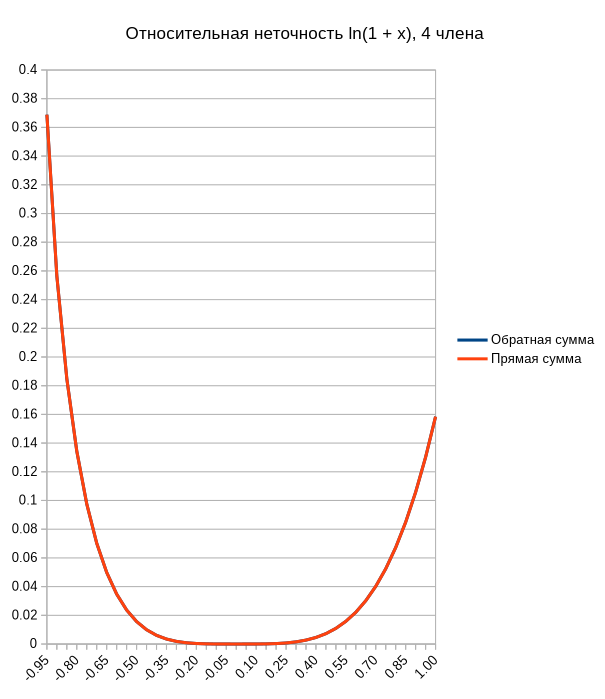


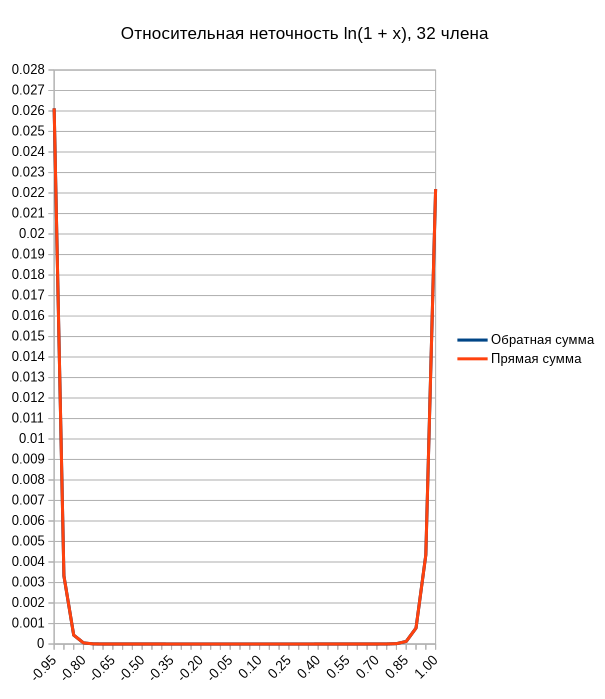












Как видно по графикам, при суммировании разложений тригонометрических функций, экспоненты и логарифма по формуле Тейлора в числовом формате binary64 порядок суммирования не влияет на точность. Это можно объяснить тем, что слагаемые убывают быстро, и инкрементальная неточность при прямом суммировании сходит на нет из-за малого количества ненулевых членов, даже при фильтрации заведомо нулевых из рядов для тригонометрических функций (чётные степени для синуса, нечётные для косинуса). Единственным исключением является одно значение в разложении косинуса по 10 членам, которое немного точнее при обратном суммировании.

На отрезке сходимости неточность совершенно незначительна при суммировании 10 или более членов. При суммировании 32 членов неточность составляет 0 на всём диапазоне для тригонометрических функций и экспоненты (график очевидной формы для этого случая построен не был), тогда как у ln(1 + 𝑥) ярко выражены границы отрезка сходимости, вблизи которых относительная неточность резко возрастает.

# Заключение

В рамках лабораторной работы были реализованы приближения некоторых трансцендентных функций по формуле Тейлора, и важность выбора такой детали реализации, как алгоритм суммирования, была оценена при помощи инъекции зависимости. Было установлено, что результат прямого и обратного суммирования практически идентичен для рассмотренных последовательностей.

# 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

Самый важный по мнению автора код:

static int for\_x(cdbl x, const series \*exc ser, const double4 \*exc terms, cuns num\_terms) {

cdbl ref = ser->ref(x);

try(print\_per\_row(x, ref));

for (const sum\_alg \*exc al = sum\_algs; al < sum\_algs\_end; al++) {

cdbl sum = calc\_sum(terms, num\_terms, al);

try(print\_per\_sumfn(sum));

}

try(putchar(\*RS));

return 0;

}

static int main\_inner(cdbl xstart, cdbl xend, cdbl xstep, const series \*exc ser, cuns num\_terms) {

cuns num\_terms\_align = align\_to\_lane(num\_terms);

double4 \*exc terms = NULL;

if (num\_terms\_align < ALLOCA\_CUTOFF\_TERMS) terms = alloca(num\_terms\_align \* sizeof(dbl));

else if (!(terms = reallocarray(NULL, num\_terms\_align, sizeof(dbl)))) return -1;

dbl x = xstart;

do {

fill\_terms(x, ser, terms, num\_terms);

#ifdef DEBUG\_SERIES

for (cdbl \*exc d = (cdbl \*) terms; d < (cdbl \*) terms + num\_terms\_align; d++)

fprintf(stderr, "%p " PRECDBL "\n", d, \*d);

putchar('\n');

#endif

for\_x(x, ser, terms, num\_terms);

x += xstep;

} while ((xstep > 0 && x < xend) || (xstep < 0 && x > xend));

return 0;

}