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

**Отчёт по лабораторной работе**

**«Математические функции»**

**Выполнил**:

студент группы 3823Б1ПМ1

Болтенков С. С.

**Проверил**:

преподаватель каф. ВВСП,

Волокитин В.Д.

Нижний Новгород

2024

**Содержание**

[Постановка задачи 3](#_7zt7dbrq9he3)

[Метод решения 4](#_fib6j129whi9)

[Руководство пользователя 6](#_1fob9te)

[Описание программной реализации 7](#_iktba7k35lxj)

[Подтверждение корректности 10](#_v772imsp5391)

[Результаты экспериментов 11](#_qv3e9b9b2nmg)

[Заключение 23](#_tntbfpufgvp)

[Список литературы 24](#_z017r5pi58fc)

[Приложение 25](#_19ewwx5i2atl)

# **Постановка задачи**

В данной лабораторной работе были реализованы следующие математические функции:

1. Синус;
2. Косинус;
3. Экспонента;
4. Натуральный логарифм аргумента ;

Для вычисления значения функций использовалась аппроксимация рядом Маклорена. Было проведено сравнение точности вычисления различных способов суммирования.

# **Метод решения**

В данной главе будут рассмотрены ряды, используемые для аппроксимации и методы суммирования [1].

Ряды Маклорена:

Ряды в общем виде:

В данной лабораторной работы было реализовано четыре метода суммирования:

1. Прямая сумма.

Сумма вычисляется последовательно, начиная с 0 члена ряда.

1. Обратная сумма.

Сумма вычисляется последовательно в обратном порядке, начиная с n-го члена.

1. Прямая попарная сумма.

Складываются и члены суммы и добавляются в общую сумму. Вычисляется с начала ряда.

1. Обратная попарная сумма.

Складываются и члены суммы и добавляются в общую сумму. Вычисляется с конца ряда.

# **Руководство пользователя**

При запуске программы вам необходимо ввести одно число - номер функции, которую хотите протестировать.

0 - синус

1 - косинус

2 - экспонента

3 - натуральный логарифм

Далее выбранная функция будет тестироваться, а результаты каждого теста будут выводиться на экран в следующей форме: аргумент, абсолютная ошибка прямой суммы, попарной, обратной и попарной обратной соответственно.

# **Описание программной реализации**

В данной главе будут рассмотрены программные реализации алгоритмов суммирования.

Первый из них - прямая сумма. Сигнатура этой функции выглядит следующим образом.

float sum\_service(float a0, float x, float(\*next)(float, int), int n);

Реализованная функция вычисляет сумму любого ряда, достаточно передать первый член (a0), аргумент (x), указатель на функцию получения следующего члена ряда (из текущего и его номера) и количество элементов в ряде (n).

Теперь пару слов о том, как получить следующий элемент ряда, зная его номер и предыдущий элемент. В ходе несложных вычислений получаем.

Например, для будет равен . Для каждой из функций можно получить свой . В реализации эти функции названы sin\_next, cos\_next (и так далее). Реализацию этих и других функций можно посмотреть в приложении.

Вернёмся к функции прямого суммирования, её реализация крайне проста. В цикле от до вычисляются элементы ряда, с помощью вышеописанного метода, и добавляются в результирующую сумму.

Следующая на очереди - обратная сумма, её сигнатура выглядит следующим образом.

float sumr\_service(float a0, float x, float(\*next)(float, int), int n);

Сигнатура абсолютна аналогична функции рассмотренной выше.

Реализация данной функции идейна похожа на предыдущую, только в этот раз мы вычисляем последний член ряда и начиная с него выполняем суммирование, находя с его помощью предшествующие члены.

Преимущество данного метода суммирования заключается в том, что он старается минимизировать ошибку, возникающую при сложении вещественных чисел, чьи порядки сильно различаются. К примеру, каждый меньше . И в какой-то момент общая сумма станет достаточно большой, а достаточно мал, что при сложение общая сумма не поменяется. Этот метод сначала складывать с конца, то есть потенциально от самых маленьких к большим.

Следующие метод методы - попарная прямая и обратная суммы.

float double\_sum\_service(float a0, float x, float(\*next)(float, int), int n);

float double\_sumr\_service(float a0, float x, float(\*next)(float, int), int n)

Эти два метода суммирования идейно ничем не отличаются друг от друга, поэтому можно рассмотреть их вместе. По своей сути они являются небольшим усовершенствованием предыдущих двух методов суммирования. Их задача также минимизировать ошибку, возникающую при сложение двух вещественных чисел сильно различного порядка.

Идея проста, сначала сложить и друг с другом и только после этого добавить их в общую сумму. Попарная сумма выполняет эти действия в прямом порядке, а обратная начиная с конца.

На данном этапе можно выдвинуть предположение, которое потом проверим на практике, что метод обратного попарного суммирования окажется самым точным.

Теперь вычисление, например, функции в коде выглядит очень лаконично.

float sin\_(float x, float (\*sum)(float, float, float(\*)(float, int), int))

{

return sum(x, x, sin\_next, 5);

}

Реализацию остальных функций также можно посмотреть в приложении.

# **Подтверждение корректности**

В данной главе будет рассказано, какие методы использовались для проверки корректности реализованных алгоритмов.

Реализованные математические функции сравнивались с аналогичными из стандартной библиотеки. Вблизи точки , как и ожидалось, значения этих функций в точности совпадают. Можно сделать выводы, что алгоритмы реализован корректно.

# **Результаты экспериментов**

В данной главе сравним виды суммирования на практике, посмотрим как изменяется ошибка при возрастании или убывании и сделаем выводы какой из методов суммирования является наиболее точным по сравнению с другими.

Начнём с синуса. Будем проводить тестирование на отрезке с шагом .

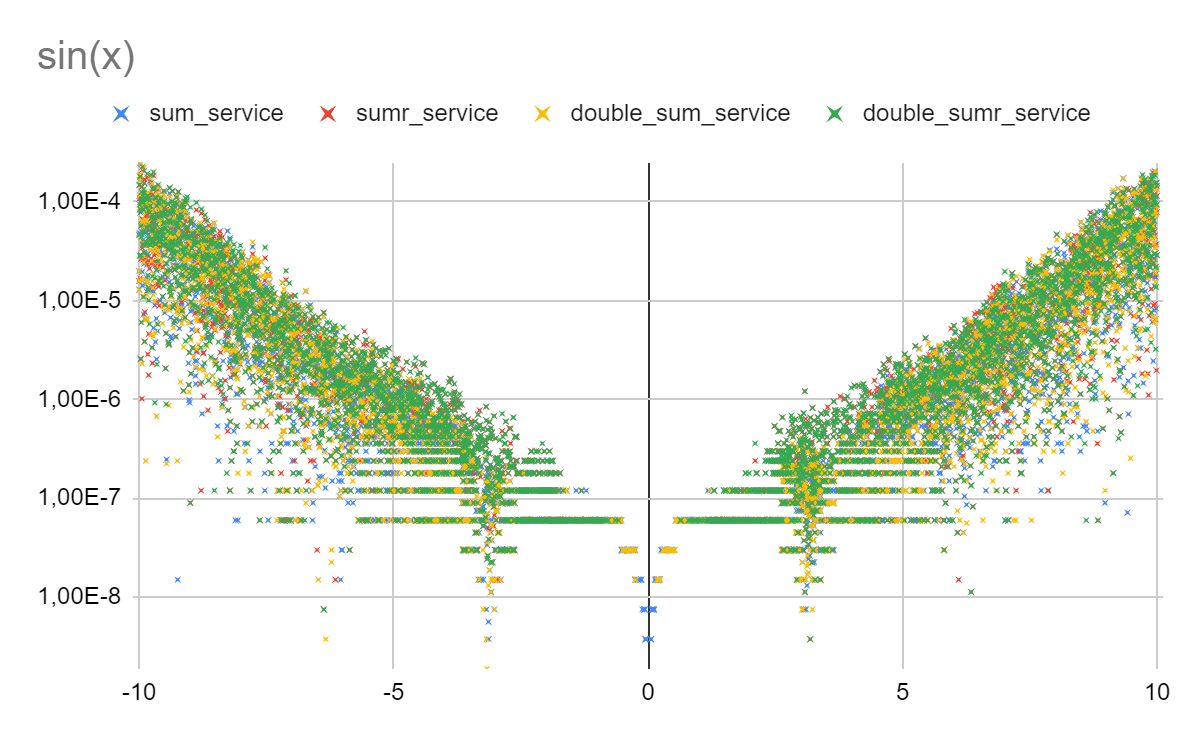


Рисунок 1. График ошибки синуса.

Теперь увеличим длину ряда, уменьшим шаг до и проведём ещё тестирование.

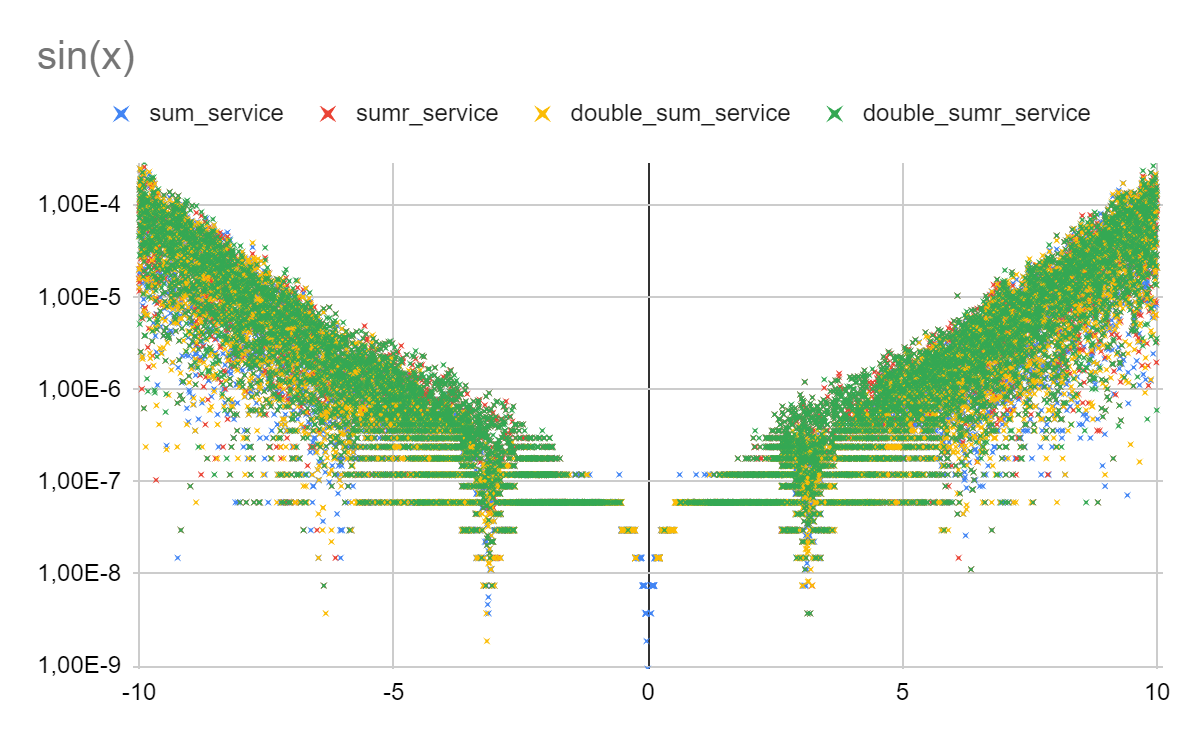


Рисунок 2. График ошибки синуса.

В проводимых эксперементах, ошибка вычислялась, как разница между значениями реализованного синуса и библиотечного. Разными цветами обозначены различные методы суммирования. Как видно из графика, чем дальше значение x от 0, тем больше становится ошибка.

Чтобы всё-таки выяснить точно какой из методов суммирования более точный был применен следующий подход. Вычислялось значение синуса в произвольной точке каждым из методов суммирования. Метод, который по модулю давал наименьшую ошибку, зарабатывал один балл. Проведя тестирование на том же отрезке , и суммируя баллы для каждого метода, были получены следующие данные.

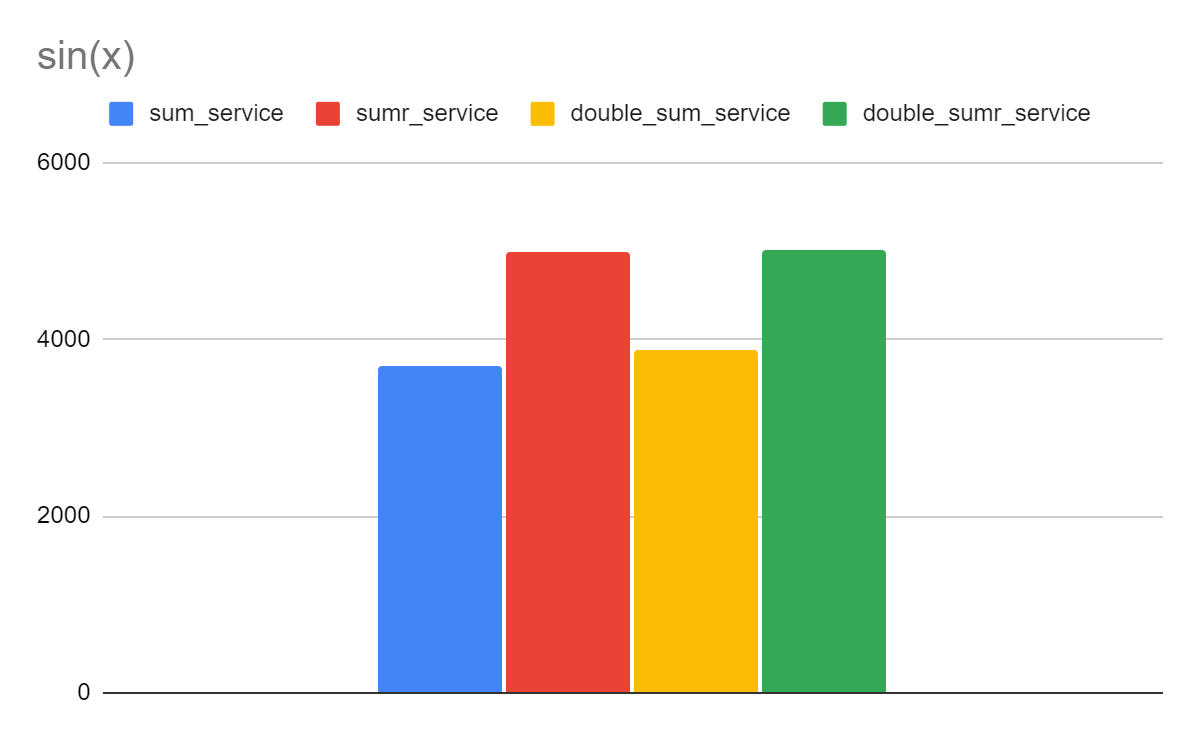


Рисунок 3. Точность методов суммирования для синуса в первом эксперименте.

Как видно из получившегося графика, значение синуса вычисляется немного точнее с помощью метода попарного обратного суммирования.

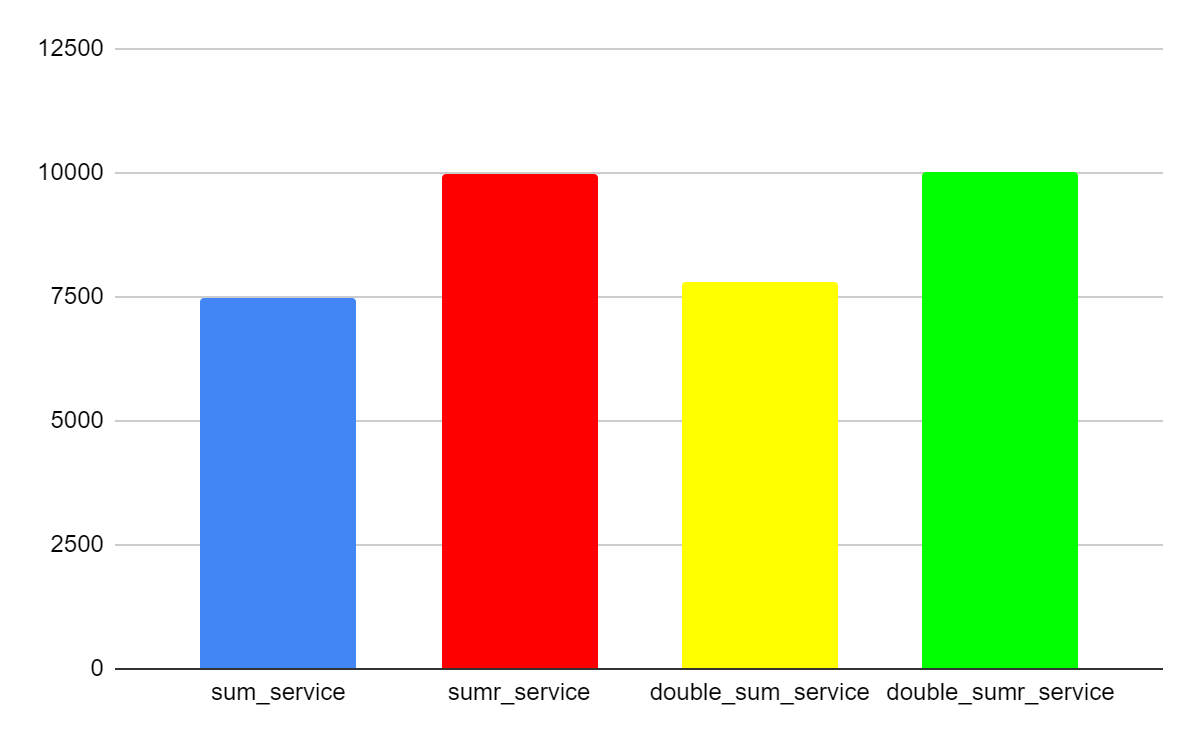


Рисунок 4. Точность методов суммирования для синуса во втором эксперименте.

А в данном случае методы обратного и попарного попарного суммирования оказались равны по точности.

Перейдем к косинусу. Для него проведем аналогичное тестирование сначала на отрезке с шагом .



Рисунок 5. График ошибки косинуса.

Во втором эксперименте также увеличим длину ряда и уменьшим шаг до .

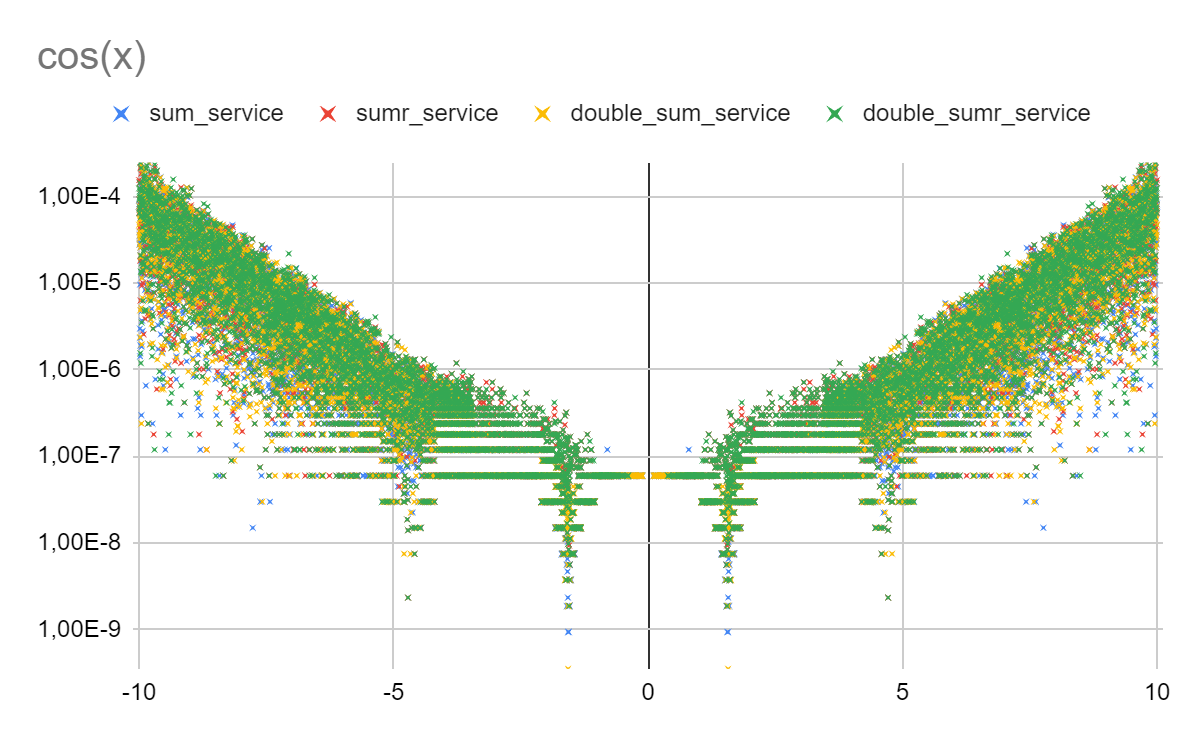


Рисунок 6. График ошибки косинуса.

Также как и с синусом по графику ошибки нельзя точно сказать, какой из методов суммирования оказался наиболее точным. Но видно, что при увеличение аргумента растет и абсолютная ошибка.

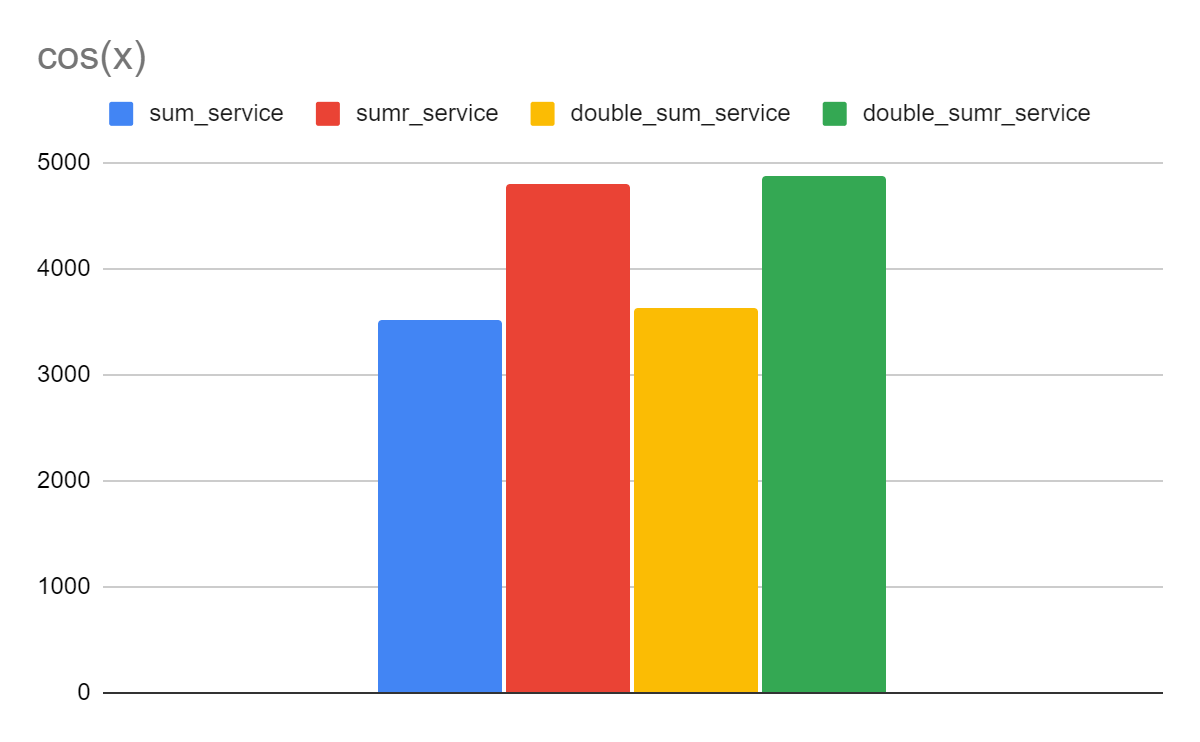


Рисунок 7. Точность методов суммирования для косинуса в первом эксперименте.

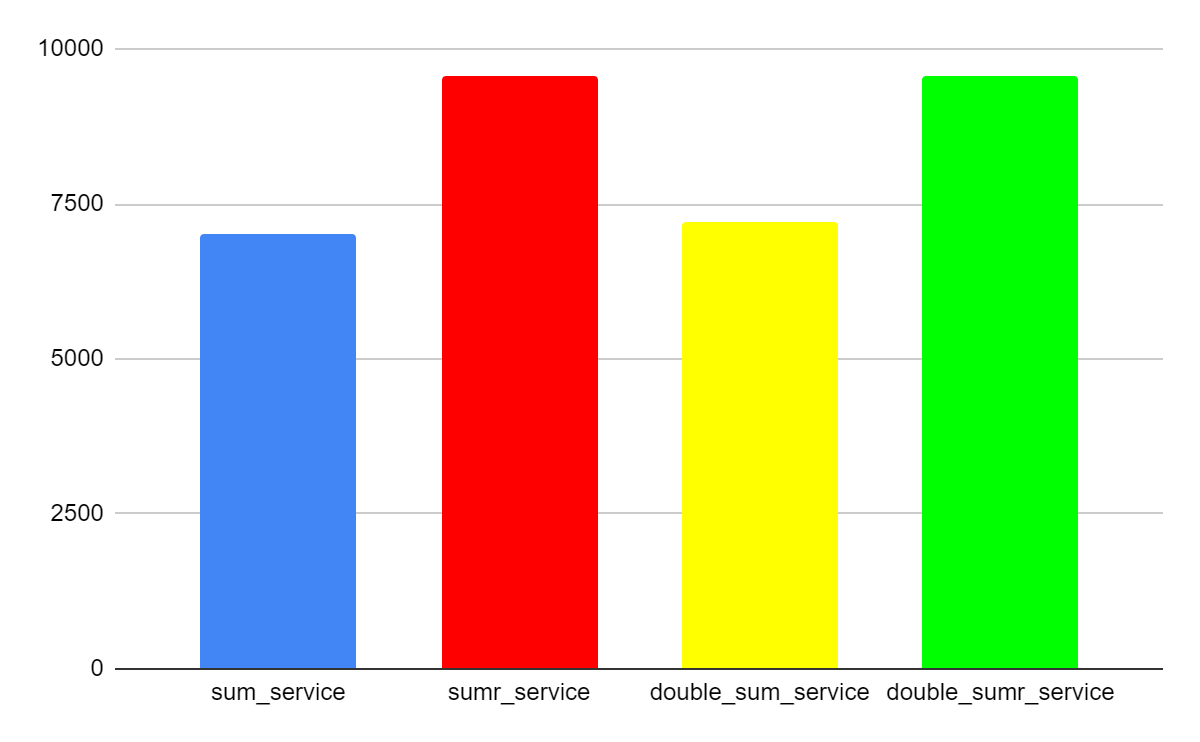


Рисунок 8. Точность методов суммирования для косинуса в первом эксперименте.

Для косинуса также самыми точным методами суммирования оказалась обратная и попарная обратная сумма.

Перейдем к экспоненте. Для нее проведем аналогичное тестирование сначала на отрезке с шагом .

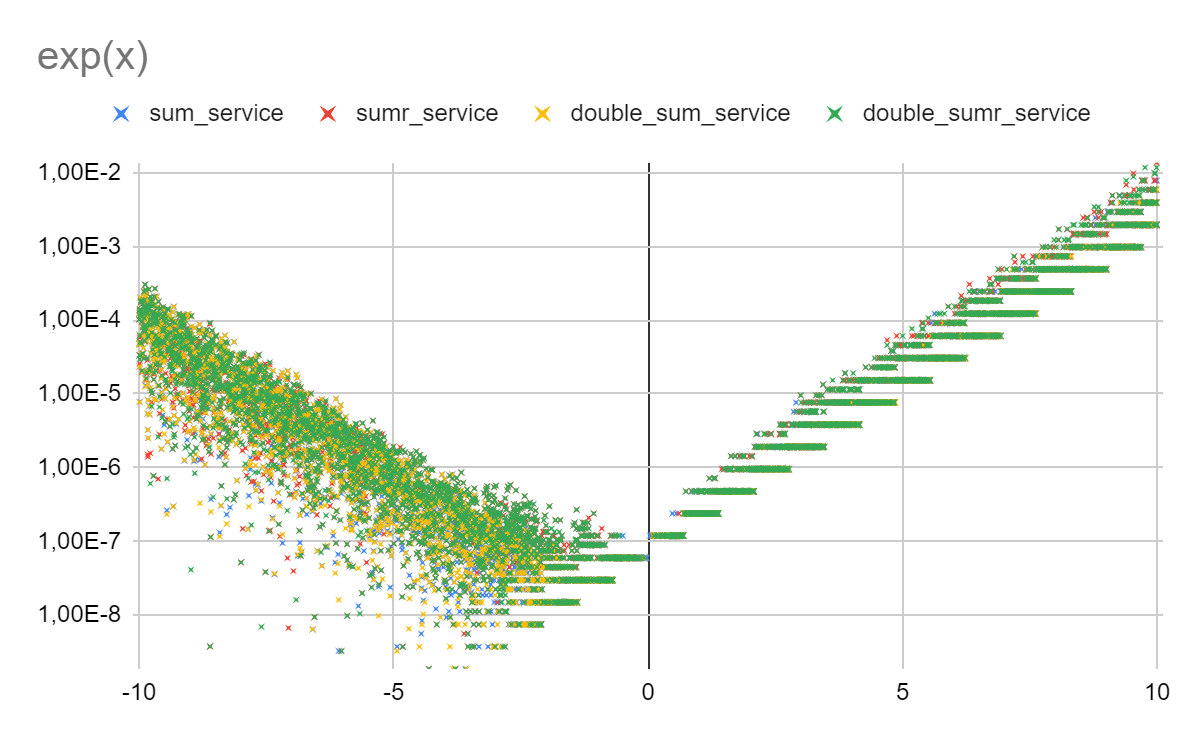


Рисунок 9. График ошибки экспоненты.

Аналогично эксперимента с синусом и косинусом увеличиваем длину ряда и уменьшаем шаг.

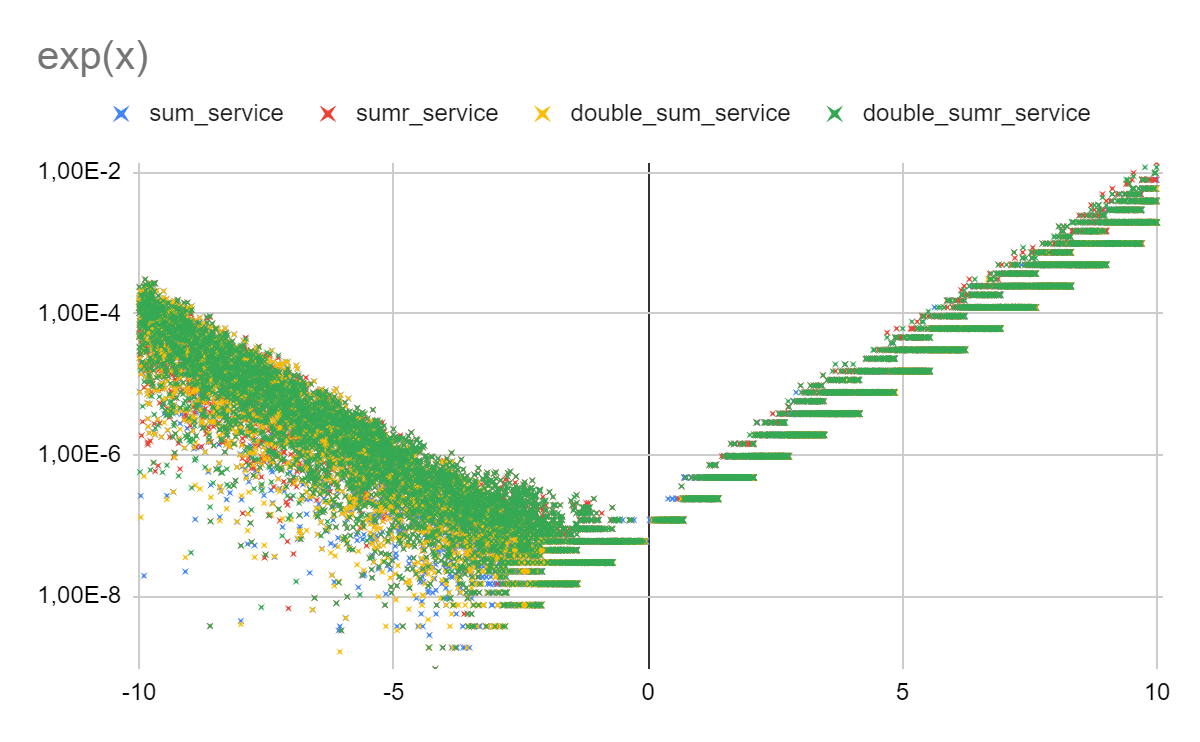


Рисунок 10. График ошибки экспоненты.

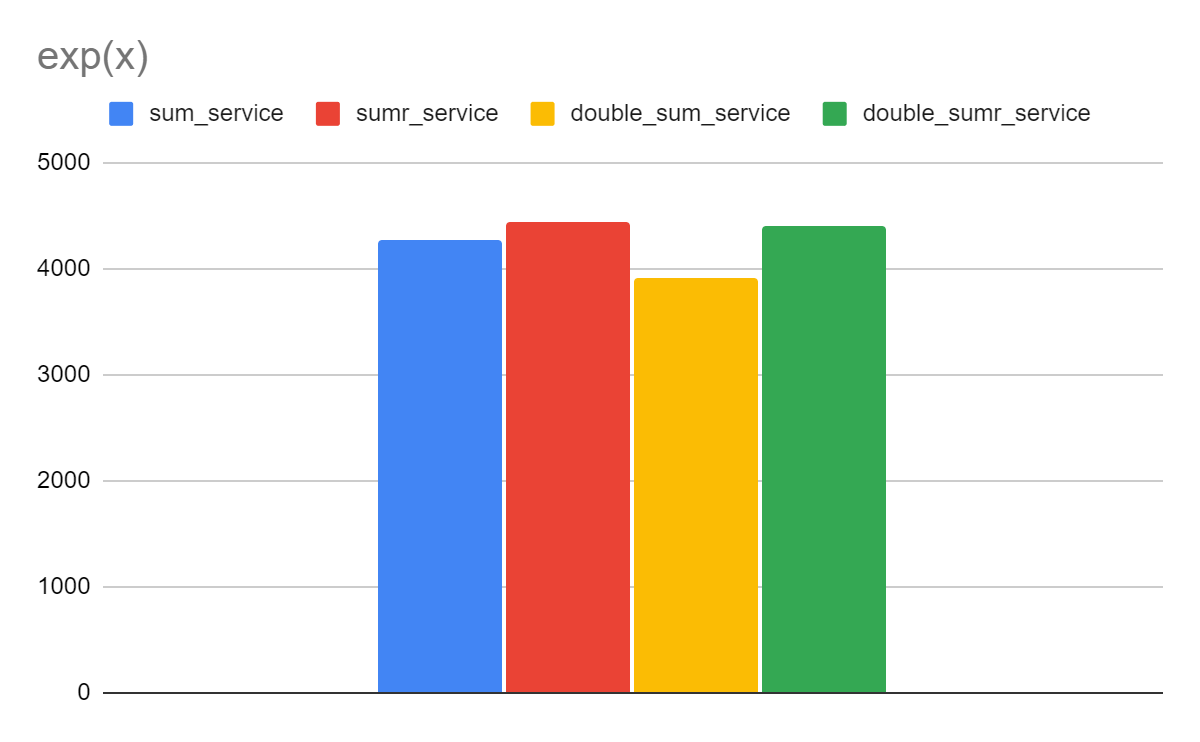


Рисунок 11. Точность методов суммирования для экспоненты в первом эксперименте.

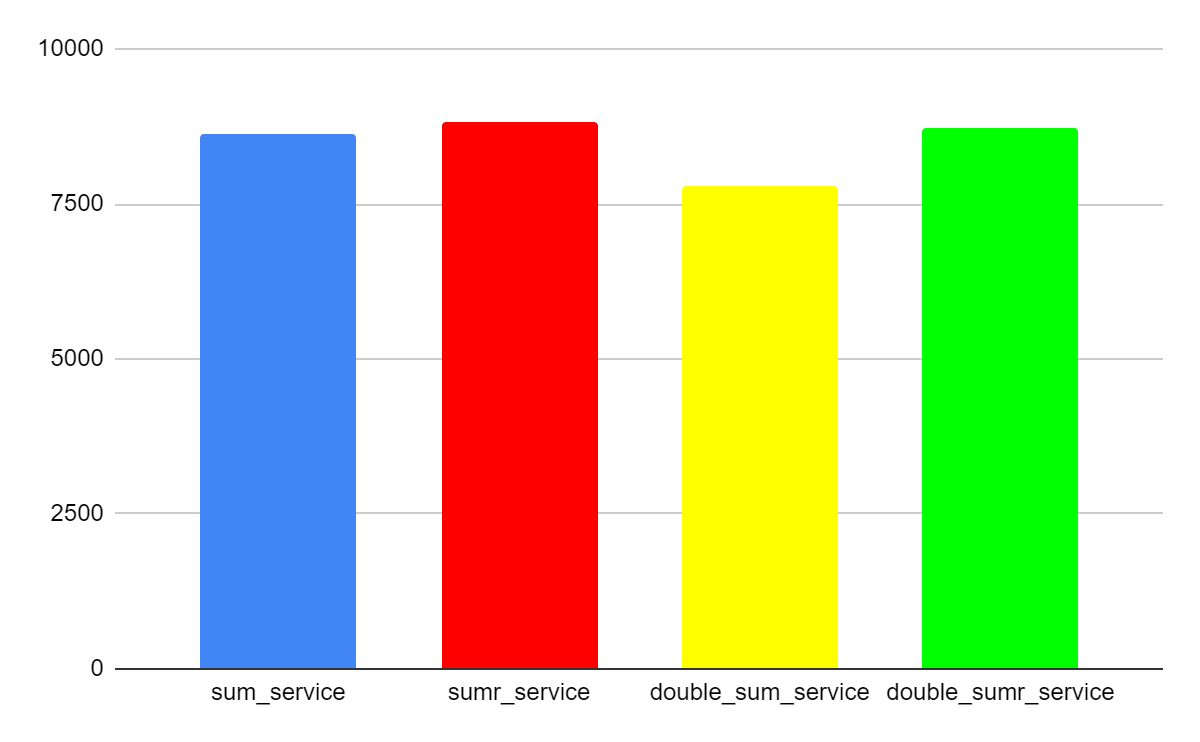


Рисунок 12. Точность методов суммирования для экспоненты во втором эксперименте.

Для экспоненты наиболее точным методом суммирования оказалась обратная сумма, хотя и попарная обратная сумма также не сильно отстаёт.

Для натурального логарифма аргумента проведём тестирование на отрезке с шагом .

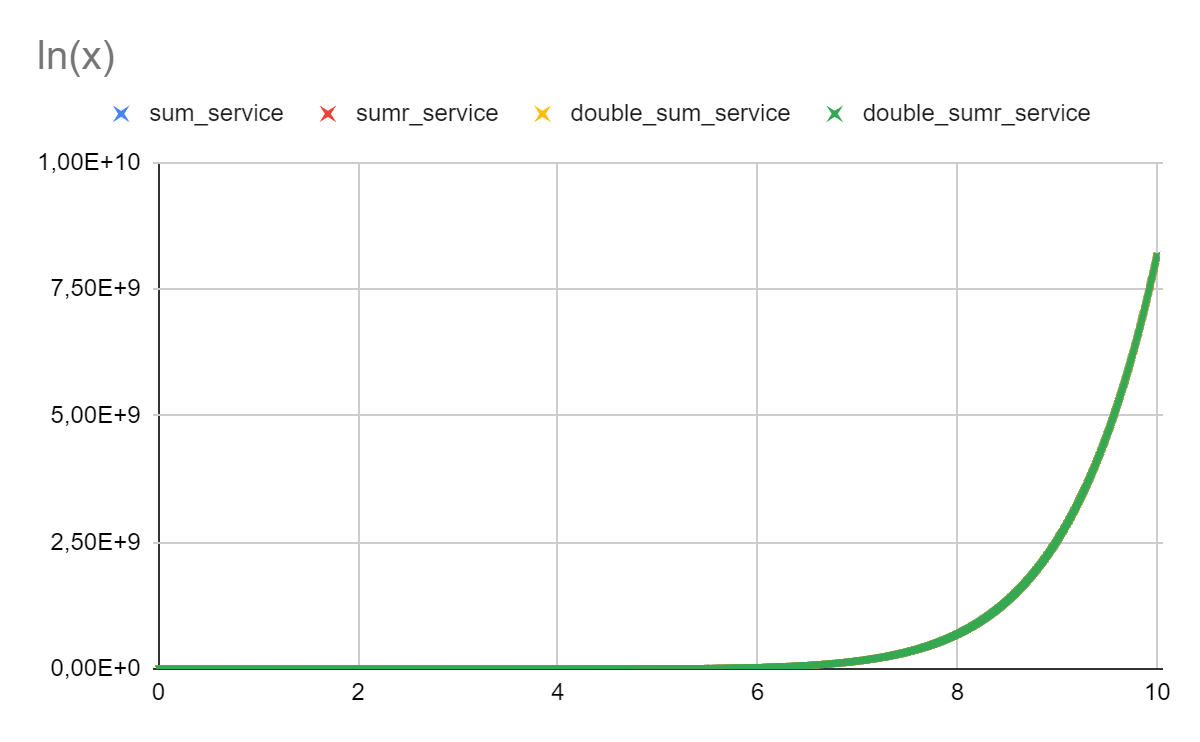


Рисунок 13. График ошибки натурального логарифма.

В ходе экспериментов выяснилось, что логарифм быстро расходится и чем больше длина ряда, тем абсолютная ошибка растёт быстрее после перехода через точку . Для того, чтобы выяснить какой всё-таки метод суммирования точнее, проведём тестирование на с шагом .

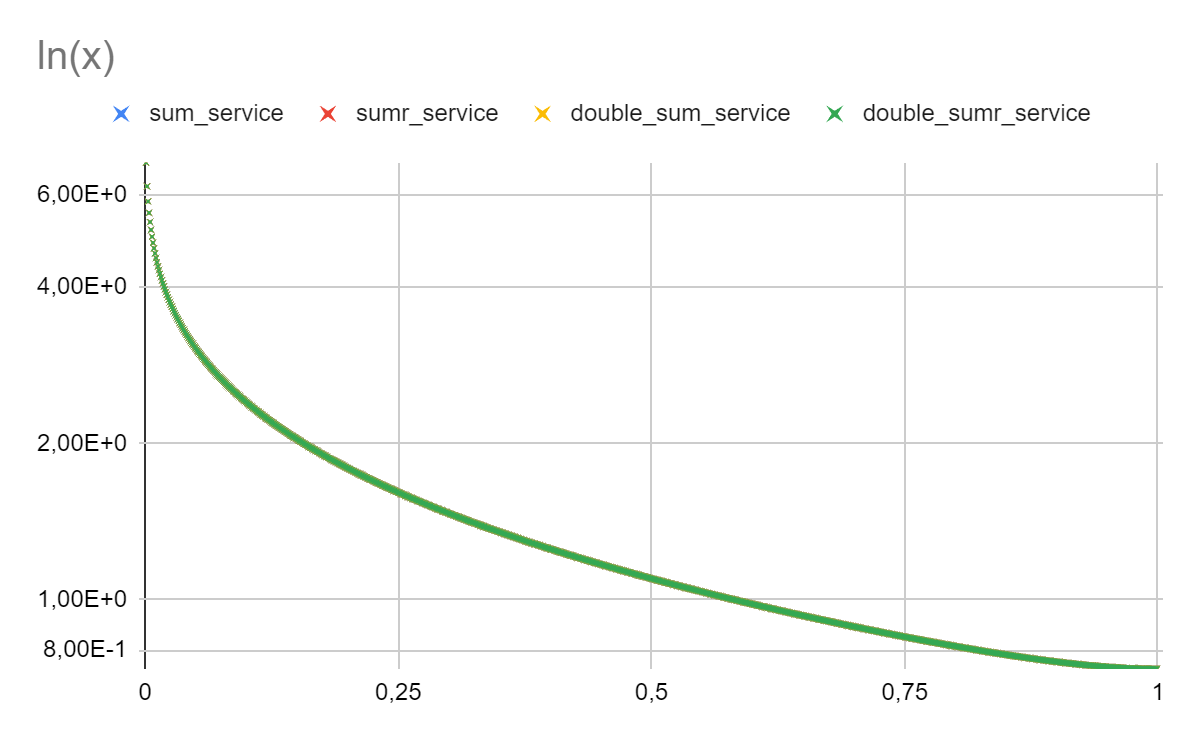


Рисунок 14. График ошибки натурального логарифма.

Как видно из получившегося графика ни один из методов суммирования сильно не выделился, по сравнению с рассмотренными ранее функциями, когда уже на графике были видны хоть какие-то визуальные различия между ними.

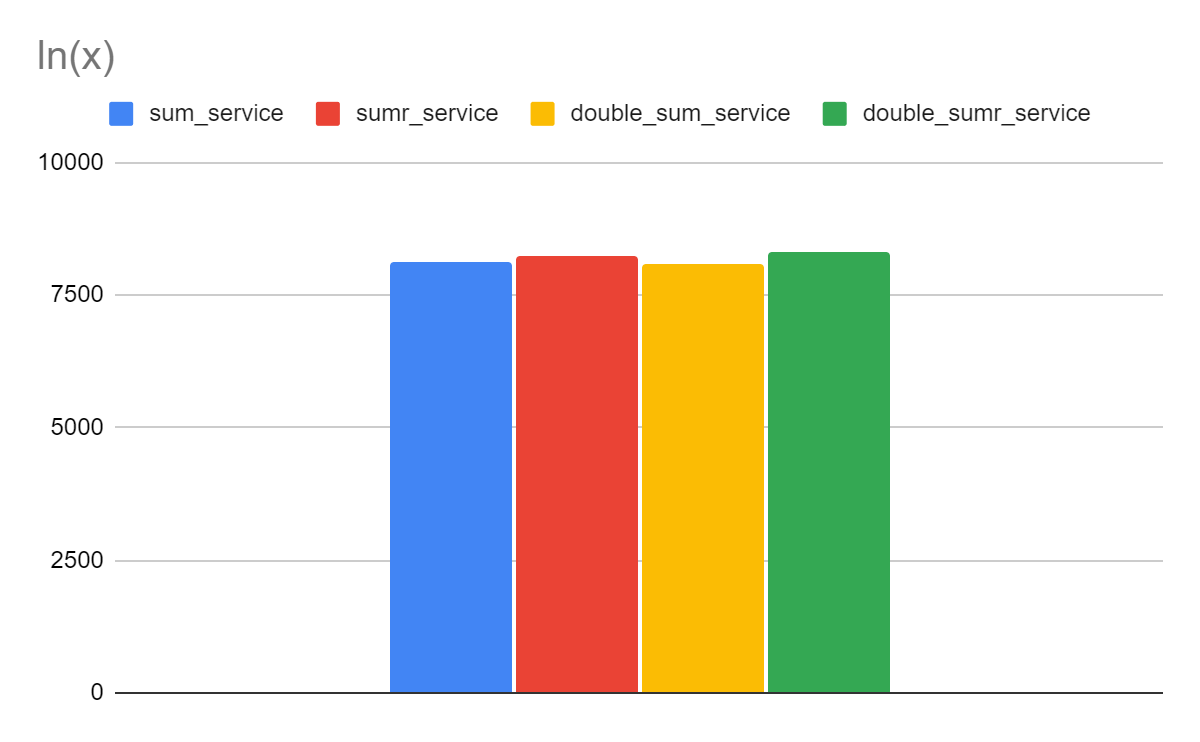


Рисунок 15. Точность методов суммирования для натурального логарифма в первом эксперименте.

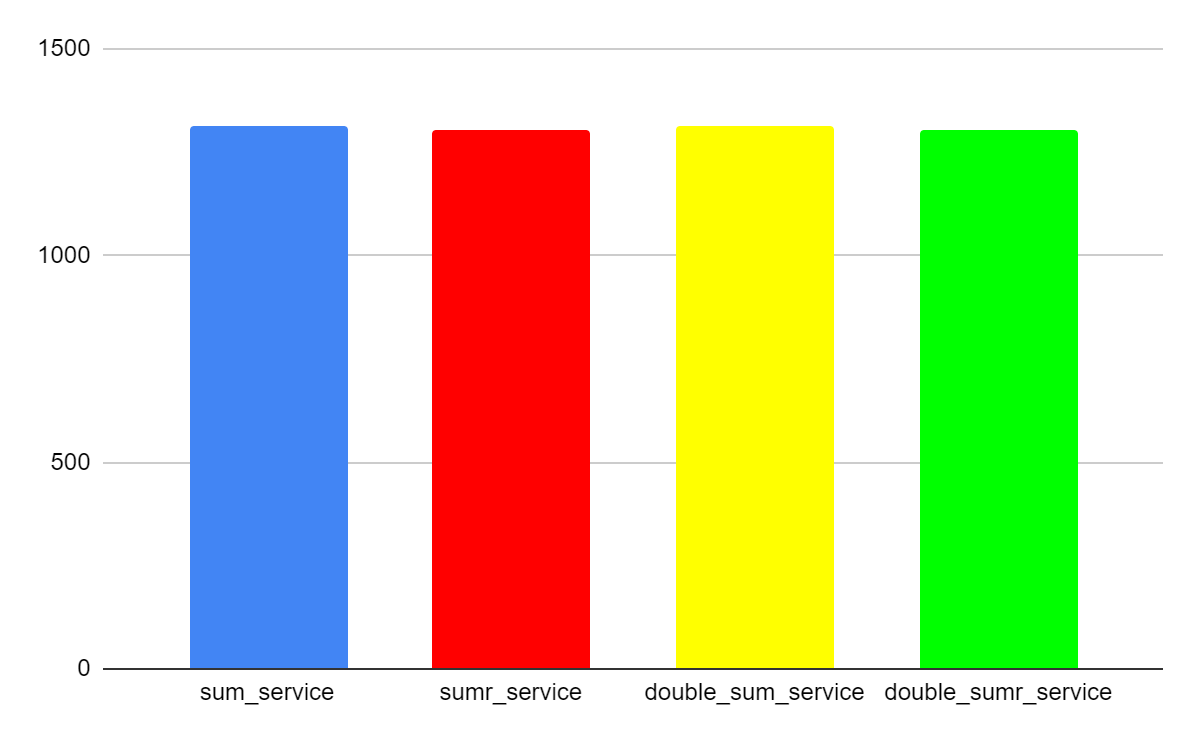


Рисунок 16. Точность методов суммирования для натурального логарифма в первом эксперименте.

Для натурального логарифма трудно сказать какой из методов суммирования является более точным. В первой таблице разница между ними минимальна, а во второй они почти все одинаковые.

# 

# **Заключение**

Подводя итог, можно сделать вывод, что синус, косинус и экспоненту наиболее точно вычисляют методы обратного и попарного обратного суммирования. Эти два метода показывали очень схожие результаты при тестирование. С натуральным логарифмом всё получилось немного сложнее. Выяснилось, что он достаточно быстро расходится даже при маленьких аргументах и достаточно большой длине ряда. Сложно сказать какой для него метод суммирования оказался точнее.

# **Список литературы**

1. Ссылка на online статью “ряд Тейлора”: <https://ru.wikipedia.org/wiki/%D0%A0%D1%8F%D0%B4_%D0%A2%D0%B5%D0%B9%D0%BB%D0%BE%D1%80%D0%B0> (дата обращения 20.02.2023)

# **Приложение**

float sin\_next(float x, int i)

{

return -x \* x / (float)(2 \* i \* (2 \* i + 1));

}

float cos\_next(float x, int i)

{

return -x \* x / (float)(2 \* i \* (2 \* i - 1));

}

float exp\_next(float x, int i)

{

return x / (float)i;

}

float ln\_next(float x, int i)

{

return -(float)x \* i / (float)(i + 1);

}

float sum\_service(float a0, float x, float(\*next)(float, int), int n)

{

float res, ai;

res = ai = a0;

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

{

ai \*= next(x, i);

res += ai;

}

return res;

}

float double\_sum\_service(float a0, float x, float(\*next)(float, int), int n)

{

float res, ai, at;

res = ai = a0;

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

{

ai \*= next(x, i);

at = ai;

if (i + 1 <= n)

{

ai \*= next(x, ++i);

at += ai;

}

res += at;

}

return res;

}

float double\_sumr\_service(float a0, float x, float(\*next)(float, int), int n)

{

float res, ai, at, t;

ai = a0;

res = 0;

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

{

t = next(x, i);

if (t == 0)

{

n = i - 1;

break;

}

ai \*= t;

}

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

{

at = ai;

ai /= next(x, i);

if (i - 1 >= 1)

{

at += ai;

ai /= next(x, --i);

}

res += at;

}

return res + a0;

}

float sumr\_service(float a0, float x, float(\*next)(float, int), int n)

{

float res, an, t;

an = a0;

res = 0;

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

{

t = next(x, i);

if (t == 0)

{

n = i - 1;

break;

}

an \*= t;

}

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

{

res += an;

an /= next(x, i);

}

return res + a0;

}

float sin\_(float x, float (\*sum)(float, float, float(\*)(float, int), int))

{

return sum(x, x, sin\_next, 5);

}

float cos\_(float x, float (\*sum)(float, float, float(\*)(float, int), int))

{

return sum(1, x, cos\_next, 5);

}

float exp\_(float x, float (\*sum)(float, float, float(\*)(float, int), int))

{

return sum(1, x, exp\_next, 10);

}

float ln\_(float x, float (\*sum)(float, float, float(\*)(float, int), int))//1 + x

{

return sum(x, x, ln\_next, 10);

}