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 рядов»**

**Выполнил**:

студент группы 3824Б1ПМ1-1

Волков Н.Е.

**Проверил**:

преподаватель каф. ВВСП,

Волокитин В.Д.

Нижний Новгород

2025

# Цель работы

В этой лабораторной работе нам необходимо реализовать разложение в ряд Тейлора некоторые математические функции (sin x, cos x, ex, ln(1+x)), подсчитать с помощью рядов значения функций а также сравнить полученные результаты со значениями встроенных библиотечных функций из math.h.

# Методы и способы решения

В этой лабораторной работе рассматривается три подхода к подсчету слагаемых рядов. Для каждой из функций, мы рассмотрим все эти три метода. Для оптимизации нашего кода и для более быстрого и простого вычисления мы будем использовать рекуррентные формулы функций. Значения первых слагаемых для функций и будет 1, для остальных значение первого слагаемого – x.Аргумент для подсчета функций и количество слагаемых вводит пользователь, потом в тригонометрических функциях проводится преобразование аргумента путем деления поступившего на вход аргумента на 2π. Это в дальнейшем позволяет нам считать тригонометрические функции от больших аргументов без каких-либо ошибок. За счет того, что мы считаем каждое слагаемое рекуррентно, это позволяет нам обойтись без вычисления факториалов, которые присутствуют в изначальной формуле разложения в ряд Тейлора. Поступившие аргумент и длину ряда мы обозначим x и i соответственно.

Формулы для вычисления функций:

:

:

:

:

**Прямое суммирование**

Этот метод вычисления является самым простым и стандартным для понимания и самым удобным для вычисления, но в свою очередь он имеет некоторые изъяны. Если слагаемые имеют большие отличия между собой, то это негативно влияет на погрешность вычисления и погрешность становится больше. Суть метода заключается в том, чтобы сумма начиналась с первого по порядку слагаемого, и шла по возрастанию до конечного слагаемого.

**Обратное суммирование**

Этот метод отличается от прошлого тем, что здесь порядок суммирования идет полностью противоположно первому методу. Это помогает снизить погрешность вычисления отличных между собой слагаемых.

В этом методе подсчет начинается с последнего слагаемого, и идет по каждому слагаемому с убыванием на один индекс до самого первого. Этот метод полезен для подсчета монотонно убывающих членов ряда или для большого количества слагаемых в ряду.

**Попарное суммирование**

Метод попарного суммирования отличается от всех других методов тем, что он не складывает элементы последовательно, а складывает их по парам. Он позволяет уменьшить погрешность в частности для тех функций, у которых знак меняется поочередно.

Если в массиве четное количество элементов, то весь массив будет разбит а пары, если нечетное, то последний элемент который остался без пары просто добавляется к результату. Способ суммы элементов является рекурсивным, и с каждой итерацией пара чисел складывается, и полученный результат будет состоять в новом массиве, проходя тот же цикл за итерацию.

# Руководство пользователя

На вход пользователю подается выбор между 4-мя математическими функциями, из которых пользователь должен выбрать одну функцию, которая будет раскладываться по Тейлору. Далее пользователю предоставляется выбор метода суммирования слагаемых в ряде. Затем пользователь должен ввести значение аргумента и размерность ряда. После этого, когда пользователь ввел все данные, то программа выводит сначала все слагаемые из ряда по очереди, затем результат по Тейлору, результат библиотечных функций из math.h и абсолютную разницу между ними (см. рис. 1).

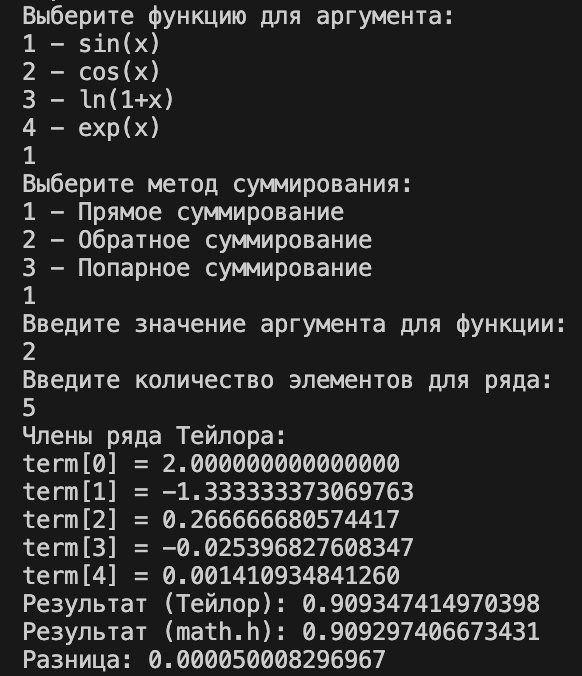


Рисунок 1. Интерфейс программы.

# Описание программной реализации

Это программа реализует подсчет математических функций с помощью раскладывания этой функции в ряд Тейлора. Полученный результат сравнивается с результатом вычислений библиотечных функций и также находится абсолютная разница между этими вычислениями. Вся программа состоит из 9 функций, 2 указателей и главной функции main, где содержится основное тело всей программы. В самом начале главной функции оформлена часть создания интерфейса программы для консоли, чтобы пользователь мог вводить данные. Далее, все основная структура программы оформлена dв функции main, где происходит выбор функции и метода суммирования от пользователя, указатели принимают выбор функции и выбор метода суммирования. Далее функция compute\_teylor принимает данные от указателя и все остальные данные которые выбрал пользователь, производит суммирование выбранных функций выбранным методом, и возвращает результат. Функции reduce\_sinx и reduce\_cosx отвечают за редукцию значения x и приводит значения x к диапазону допустимому для синуса и косинуса. Функции sinx, cosx, expx, lnx принимает три параметра(пустой массив, значение аргумента и размерность массива), заполняют массив рядом Тейлора и возвращает в конце уже заполненный рядом Тейлора массив. Функции sum\_pryam, sum\_obr и sum\_popar принимают на вход заполненный рядом массив и его размер. Каждая из этих функций подсчитывает разными способами значение всего поданного нам массива и возвращает значение суммы нашего ряда. После вызова всех функций, программа выводит в консоль все члены из ряда по порядку, значение подсчитанное путем разложения функции в ряд, значение посчитанное библиотечной функцией и абсолютная разница между ними.

**Все функции программы:**

* float reduce\_sinx(float x) – функция, которая производит редукцию аргумента x для функции
* float reduce\_cosx(float x) – функция, которая производит редукцию аргумента x для функции
* void sinx(float x, int size, float\* chleni\_ryada) – функция, отвечающая за создание массива состоящего из элементов, которыми являются члены от разложения функции в ряд;
* void cosx(float x, int size, float\* chleni\_ryada) – функция, смысл которой аналогичен функции синуса, но только разложние идет не синуса а косинуса;
* void expx(float x, int size, float\* chleni\_ryada) – функция, отвечающая за разложение экспоненциальной функции в ряд Тейлора таким же методом как в тригонометрических функциях;
* void lnx(float x, int size, float\* chleni\_ryada) – функция, отвечающая за разложение натурального логарфима в ряд Тейлора, аналогичным методом как во всех 3-х других;
* float sum\_pryam(float\* chleni\_ryada, int size) – функция, отвечающая за суммирование всех элементов нового созданного массива, путем прямого суммирования;
* float sum\_obr(float\* chleni\_ryada, int size) – функция, которая осуществляет суммирование элементов массива начиная с конца;
* float sum\_popar(float\* chleni\_ryada, int size) – функция, которая осуществляет суммирование элементов массива попарно, рекурсивно вызывая саму себя;
* float compute\_taylor(float x, int size, float\* chleni\_ryada, SumFunction sumFunction, FuncChoice func) – функций, которая принимает указатели на функции, принимает выбранную функцию и производит подсчет ее суммы.
* int main() – основная функция, которая содержит основную часть кода и в которой происходит вызов все остальных функций;

# Подтверждение корректности

Для подтверждения корректности в программе использовались математические функции из библиотеки math.h. В структуре switch-case вызываются библиотечные функции, и результат от них запоминается в одну переменную, с которой в конце программы сравнивается значение, которое было найдено по разложению в ряд.

# Результаты экспериментов

Для проведения экспериментов рассмотрим каждый вид суммирования для каждой функции по отдельности, рассматривая каждую функцию со значениями аргумента  для функций и для функции соответственно. Для того чтобы более детальнее рассмотреть разницу в погрешности и сделать вывод, какое количество элементов в ряду для каждой функции необходимо, рассмотрим разное количество элементов.

1. **Прямое суммирование**

Для функции рассмотрим длину ряда в 20 элементов.

Все результаты будут представлены на графиках и таблицах. Ось абсцисс графика будет отвечать за количество элементов в ряду, а ось ординат за разницу между значениями(погрешность).

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 1.783341169357300 |
| 2 | 0.618873834609985 |
| 3 | 0.092488110065460 |
| 4 | 0.007823228836060 |
| 5 | 0.000428140163422 |
| 6 | 0.000016152858734 |
| 7 | 0.000000715255737 |
| 8 | 0.000000238418579 |
| 9 | 0.000000238418579 |
| 10 | 0.000000238418579 |
| 11 | 0.000000238418579 |
| 12 | 0.000000238418579 |
| 13 | 0.000000238418579 |
| 14 | 0.000000238418579 |
| 15 | 0.000000238418579 |
| 16 | 0.000000238418579 |
| 17 | 0.000000238418579 |
| 18 | 0.000000238418579 |
| 19 | 0.000000238418579 |
| 20 | 0.000000238418579 |

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 1.759687900543213 |
| 2 | 1.201587200164795 |
| 3 | 0.259937763214111 |
| 4 | 0.028594076633453 |
| 5 | 0.001920998096466 |
| 6 | 0.000087082386017 |
| 7 | 0.000003039836884 |
| 8 | 0.000000119209290 |
| 9 | 0.000000178813934 |
| 10 | 0.000000178813934 |
| 11 | 0.000000178813934 |
| 12 | 0.000000178813934 |
| 13 | 0.000000178813934 |
| 14 | 0.000000178813934 |
| 15 | 0.000000178813934 |
| 16 | 0.000000178813934 |
| 17 | 0.000000178813934 |
| 18 | 0.000000178813934 |
| 19 | 0.000000178813934 |
| 20 | 0.000000178813934 |

Таблица 1. Прямое суммирование для sin(x) при x=15. Таблица 2. Прямое суммирование для cos(x) при x=15.

На рис. 2 представлен график зависимости погрешности от числа слагаемых для функции . Можно заметить, что начиная с 5 слагаемых в ряду, погрешность стремится к нулю, и точность наиболее высока.

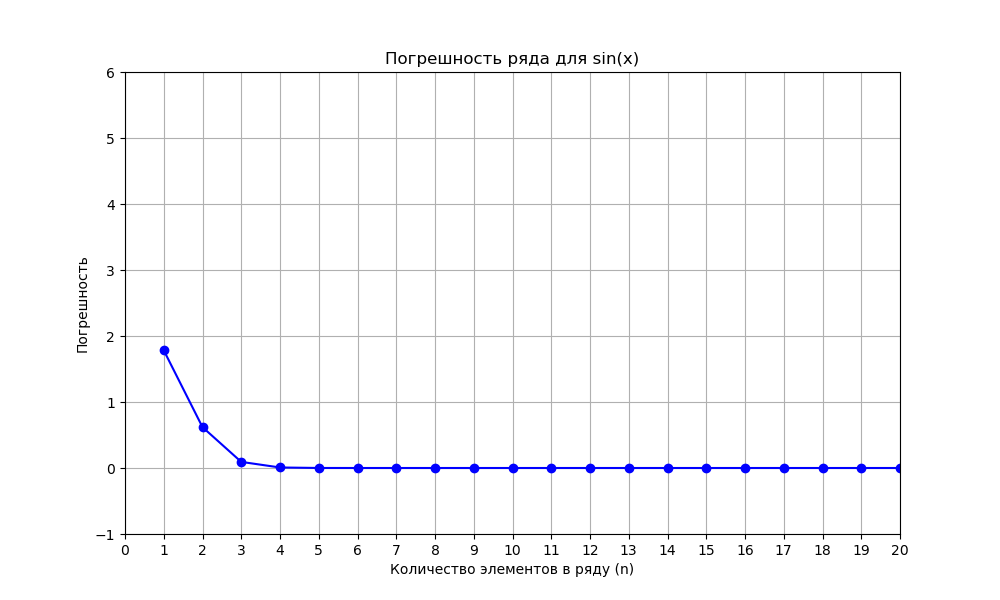


Рисунок 2. График прямого суммирования для sin(x) при x=15

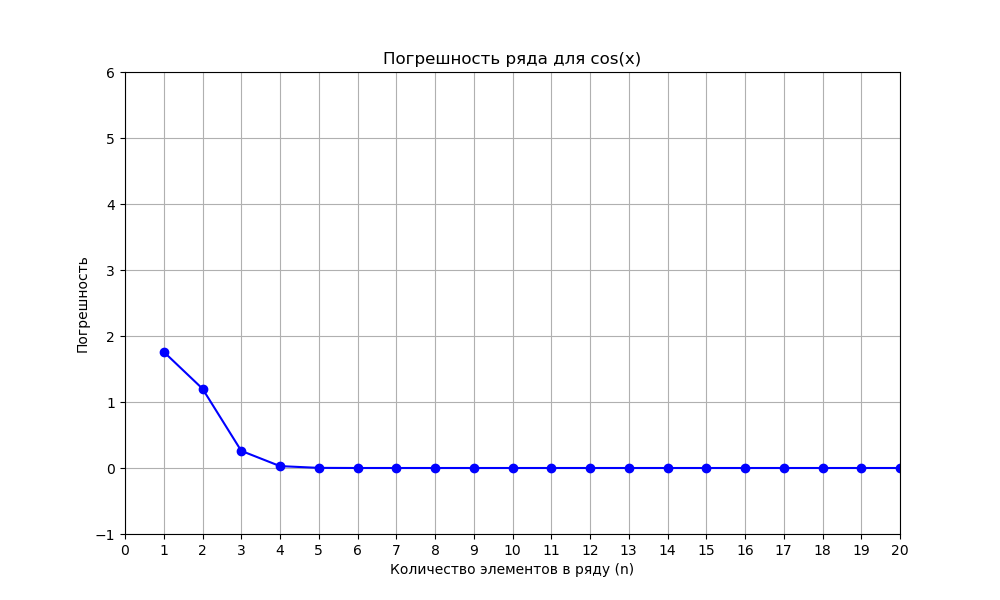


Рисунок 3. График прямого суммирования для cos(x) при x=15

С косинусом такая же ситуация, начиная с 5 элементов в ряду значения стремятся к значениям от библиотечной функции(см рис. 3), тем самым показывая, что прямое суммирование действует для косинуса и синуса одинаково.

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 3269016.250000000000000 |
| 2 | 3269001.250000000000000 |
| 3 | 3268888.750000000000000 |
| 4 | 3268326.250000000000000 |
| 5 | 3266217.000000000000000 |
| 6 | 3259888.750000000000000 |
| 7 | 3244068.500000000000000 |
| 8 | 3210167.750000000000000 |
| 9 | 3146604.000000000000000 |
| 10 | 3040664.500000000000000 |
| 11 | 2881755.000000000000000 |
| 12 | 2665060.500000000000000 |
| 13 | 2394192.000000000000000 |
| 14 | 2081651.750000000000000 |
| 15 | 1746787.125000000000000 |
| 16 | 1411922.500000000000000 |
| 17 | 1097987.000000000000000 |
| 18 | 820985.000000000000000 |
| 19 | 590150.000000000000000 |
| 20 | 407911.750000000000000 |

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0.094534903764725 |
| 2 | 0.030465096235275 |
| 3 | 0.011201560497284 |
| 4 | 0.004423439502716 |
| 5 | 0.001826554536819 |
| 6 | 0.000777602195740 |
| 7 | 0.000338464975357 |
| 8 | 0.000149816274643 |
| 9 | 0.000067204236984 |
| 10 | 0.000030457973480 |
| 11 | 0.000013917684555 |
| 12 | 0.000006437301636 |
| 13 | 0.000002950429916 |
| 14 | 0.000001400709152 |
| 15 | 0.000000625848770 |
| 16 | 0.000000327825546 |
| 17 | 0.000000119209290 |
| 18 | 0.000000089406967 |
| 19 | 0.000000000000000 |
| 20 | 0.000000059604645 |

Таблица 3. Прямое суммирование для exp(x) при x=15. Таблица 4. Прямое суммирование для ln(x+1) при x=0.5

Для экспоненты ситуация меняется. Она сильно не сходится со значениями из библиотечных функций, и даже ряд в 20 с трудом может дать минимальную погрешность. Рассмотрим поведение графика уже с 50 элементами в ряду. Как можно видеть(см. рис 5), начиная с 19 элемента, погрешность начинается стремится к нулю, что означает что для более точного вычисления экспоненциальной функции, необходимо большое количество элементов в ряде.

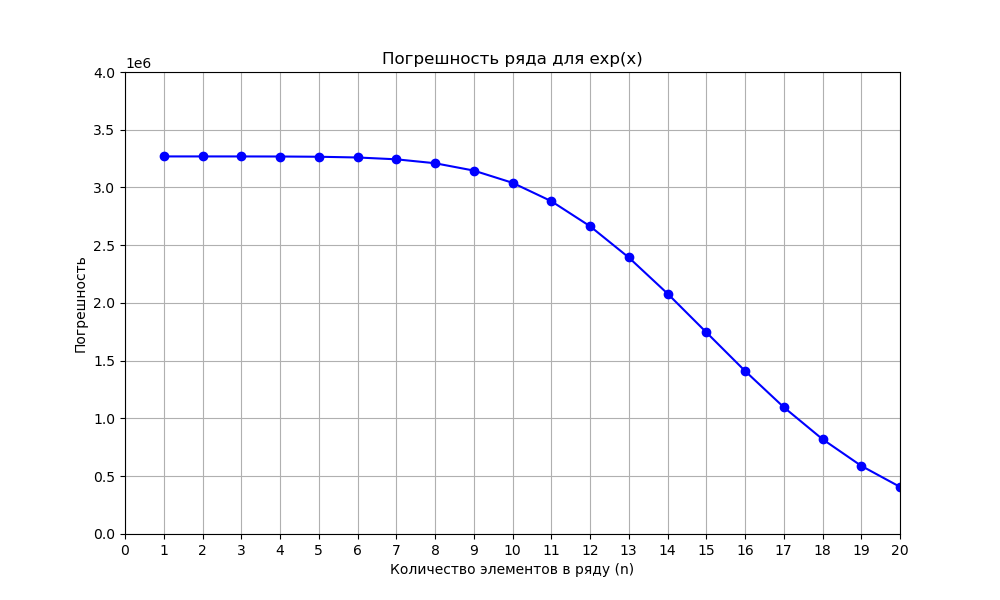


Рисунок 4. График прямого суммирования для exp(x) при x=15

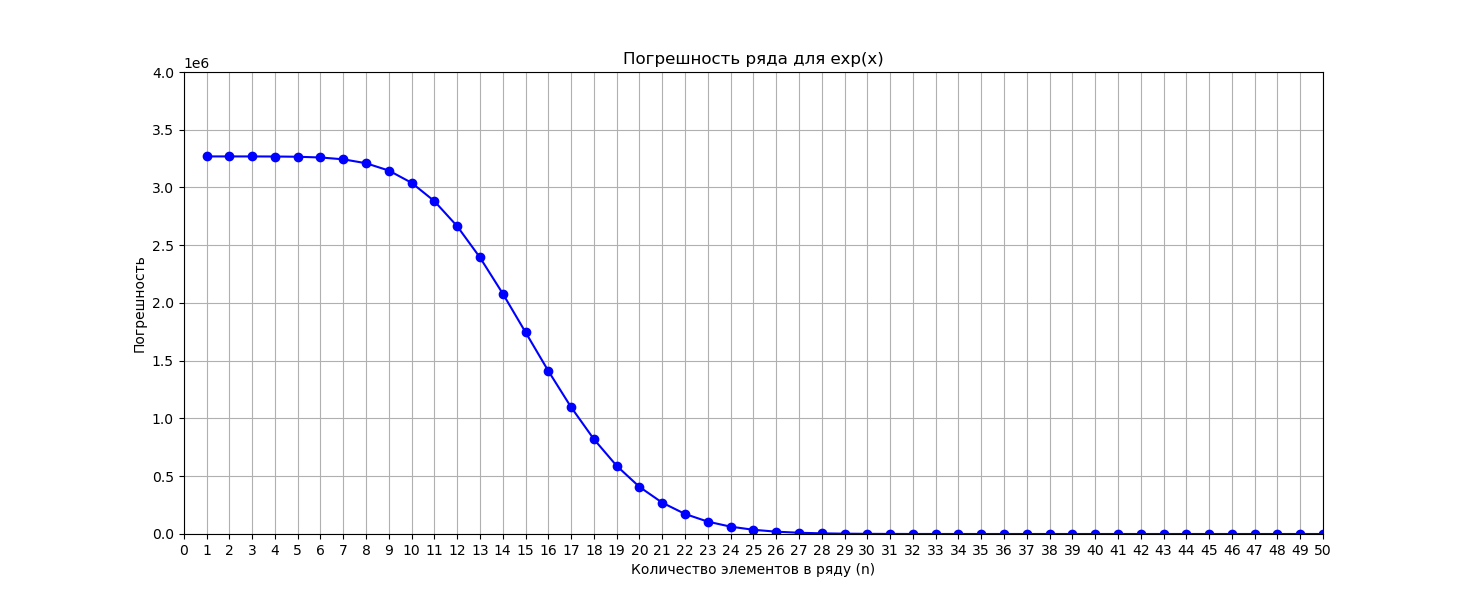


Рисунок 5. График прямого суммирования для exp(x) при x=15 при 50 элементах в ряду

Далее рассмотрим функцию логарифма. Кривая очень плавно опускается к значению ноль и достигает нулевой погрешности(см. рис 6).

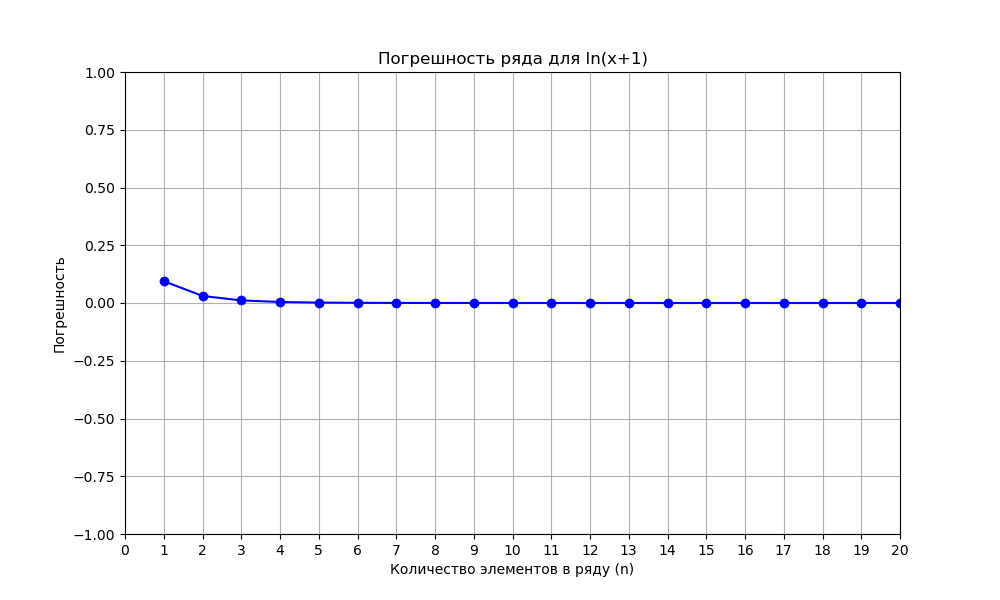
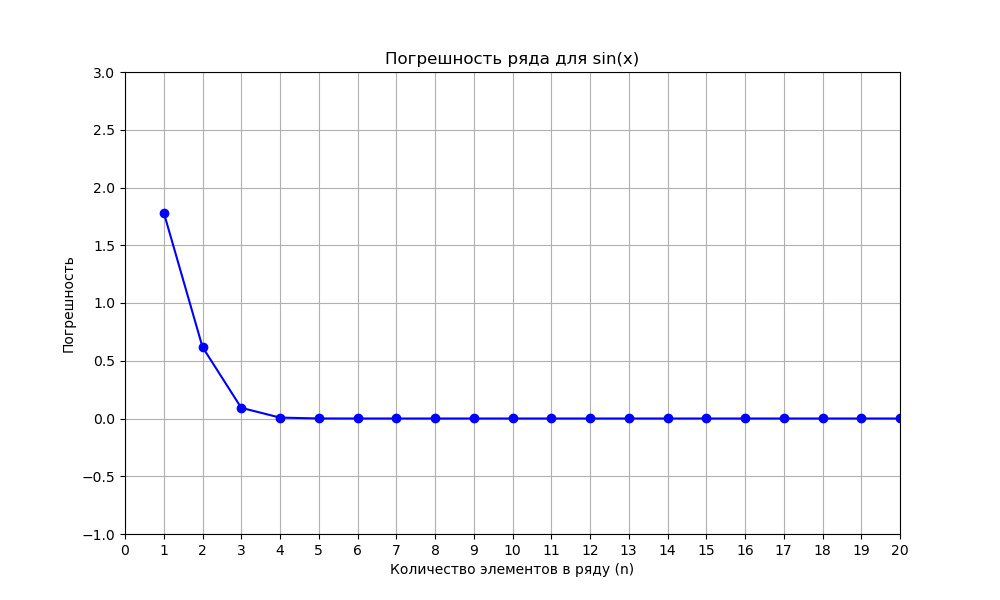


Рисунок 6. График прямого суммирования для ln(x+1) при x=15

1. **Обратное суммирование**

****

**Рисунок 7. График обратного суммирования для sin(x) при x=15**

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 1.783341169357300 |
| 2 | 0.618873834609985 |
| 3 | 0.092488050460815 |
| 4 | 0.007823228836060 |
| 5 | 0.000428199768066 |
| 6 | 0.000016093254089 |
| 7 | 0.000000715255737 |
| 8 | 0.000000238418579 |
| 9 | 0.000000238418579 |
| 10 | 0.000000238418579 |
| 11 | 0.000000238418579 |
| 12 | 0.000000238418579 |
| 13 | 0.000000238418579 |
| 14 | 0.000000238418579 |
| 15 | 0.000000238418579 |
| 16 | 0.000000238418579 |
| 17 | 0.000000238418579 |
| 18 | 0.000000238418579 |
| 19 | 0.000000238418579 |
| 20 | 0.000000238418579 |

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 1.783341169357300 |
| 2 | 0.618873834609985 |
| 3 | 0.092488050460815 |
| 4 | 0.007823228836060 |
| 5 | 0.000428199768066 |
| 6 | 0.000016093254089 |
| 7 | 0.000000715255737 |
| 8 | 0.000000238418579 |
| 9 | 0.000000238418579 |
| 10 | 0.000000238418579 |
| 11 | 0.000000238418579 |
| 12 | 0.000000238418579 |
| 13 | 0.000000238418579 |
| 14 | 0.000000238418579 |
| 15 | 0.000000238418579 |
| 16 | 0.000000238418579 |
| 17 | 0.000000238418579 |
| 18 | 0.000000238418579 |
| 19 | 0.000000238418579 |
| 20 | 0.000000238418579 |

Таблица 5. Обратное суммирование для sin(x) . Таблица 6. Обратное суммирование для cos(x).

На графике видно(см рис.7) что погрешность для синуса становится нулевой при 4 членах в ряду. Начиная с 2 элементов, она плавна стремится к нулю.

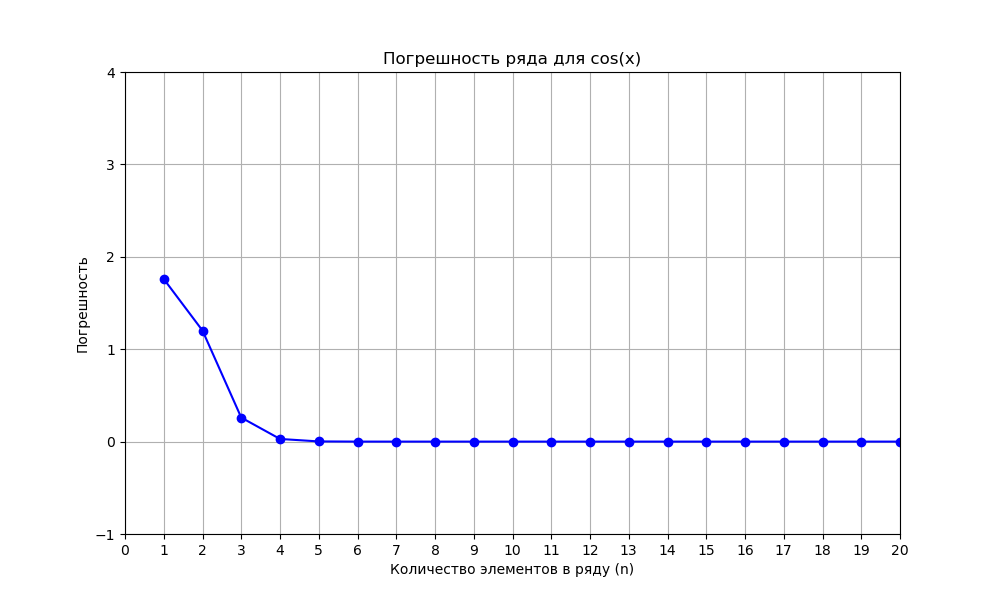
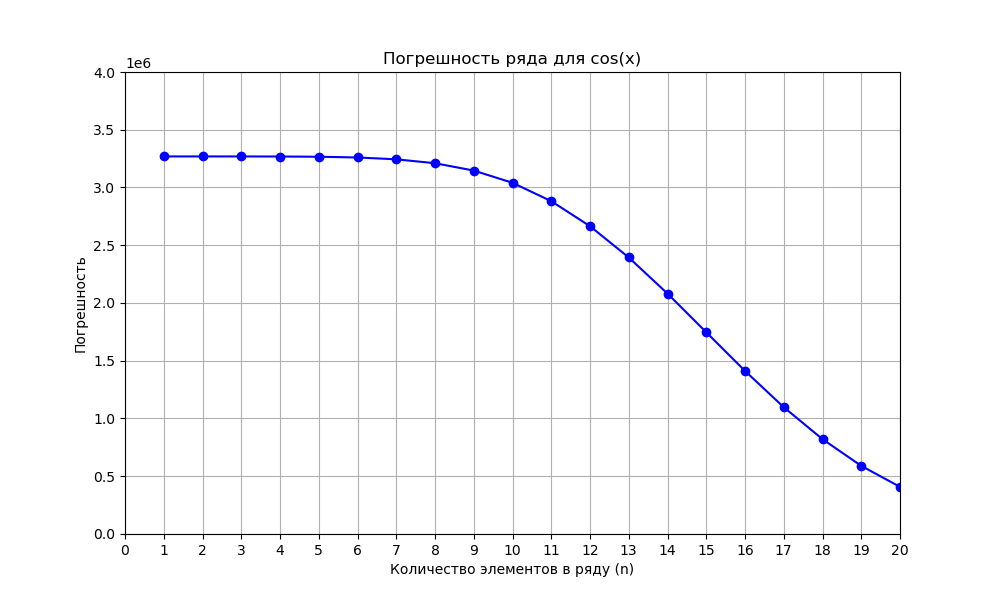
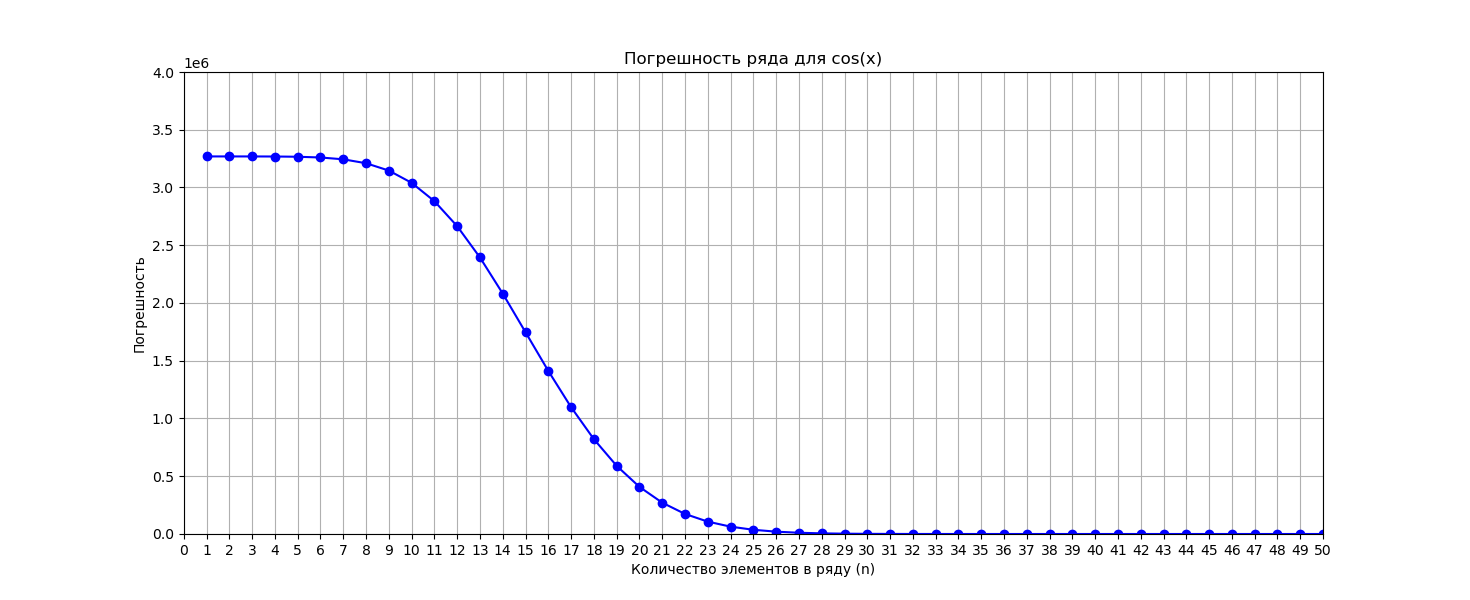


Рисунок 7. График обратного суммирования для cos(x)

По поводу косинуса можно сказать то же самое что и про синус, только погрешность становится нулевой при 5 элементах в ряду(см. рис7).

Рисунок 8. График обратного суммирования для exp(x) при 20 элементов в ряду

Рисунок 9. График обратного суммирования для exp(x) при 50 элементов в ряду

Для того чтобы определить где погрешность у экспоненциальной функции сходится к нулю, мы также как и в прямом суммировании должны рассмотреть график с 50 элементами. На этом графике, погрешность приближается к нулю с 25 элемента.

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 3269016.250000000000000 |
| 2 | 3269001.250000000000000 |
| 3 | 3268888.750000000000000 |
| 4 | 3268326.250000000000000 |
| 5 | 3266217.000000000000000 |
| 6 | 3259888.750000000000000 |
| 7 | 3244068.500000000000000 |
| 8 | 3210167.750000000000000 |
| 9 | 3146604.000000000000000 |
| 10 | 3040664.500000000000000 |
| 11 | 2881755.000000000000000 |
| 12 | 2665060.500000000000000 |
| 13 | 2394192.000000000000000 |
| 14 | 2081651.750000000000000 |
| 15 | 1746787.250000000000000 |
| 16 | 1411922.750000000000000 |
| 17 | 1097986.750000000000000 |
| 18 | 820985.250000000000000 |
| 19 | 590150.250000000000000 |
| 20 | 407911.750000000000000 |

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0.094534903764725 |
| 2 | 0.030465096235275 |
| 3 | 0.011201590299606 |
| 4 | 0.004423409700394 |
| 5 | 0.001826554536819 |
| 6 | 0.000777602195740 |
| 7 | 0.000338464975357 |
| 8 | 0.000149816274643 |
| 9 | 0.000067204236984 |
| 10 | 0.000030428171158 |
| 11 | 0.000013947486877 |
| 12 | 0.000006407499313 |
| 13 | 0.000002980232239 |
| 14 | 0.000001370906830 |
| 15 | 0.000000655651093 |
| 16 | 0.000000298023224 |
| 17 | 0.000000149011612 |
| 18 | 0.000000059604645 |
| 19 | 0.000000029802322 |
| 20 | 0.000000000000000 |

Таблица 8. Обратное суммирование для ln(1+x).

Таблица 7. Обратное суммирование для exp(x).

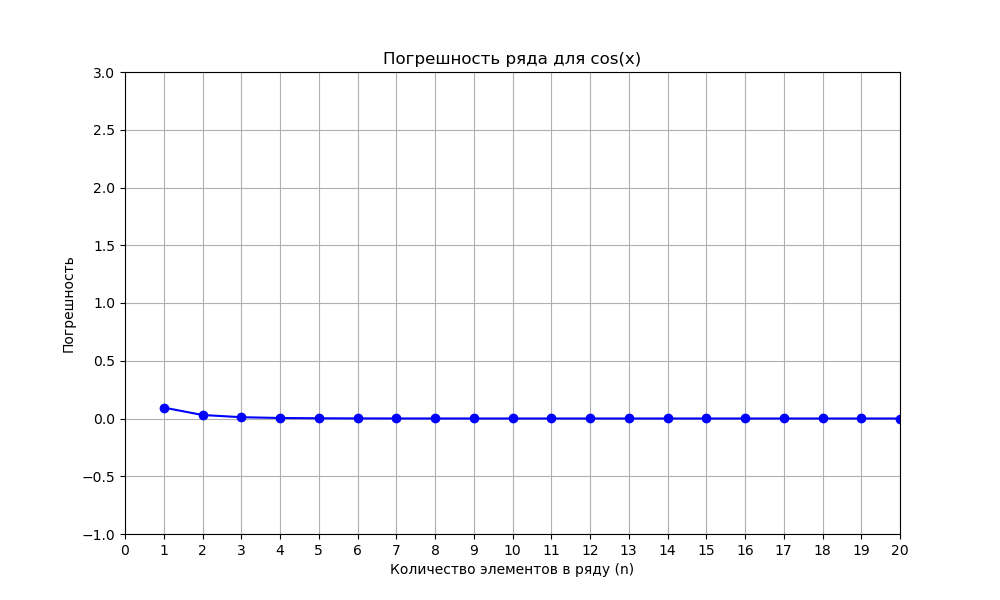


Рисунок 10. График обратного суммирования для ln(1+x)

У логарифмической функции кривая плавно сходится к нулю, начиная со второго элемента(см. рис 10).

1. **Попарное суммирование**

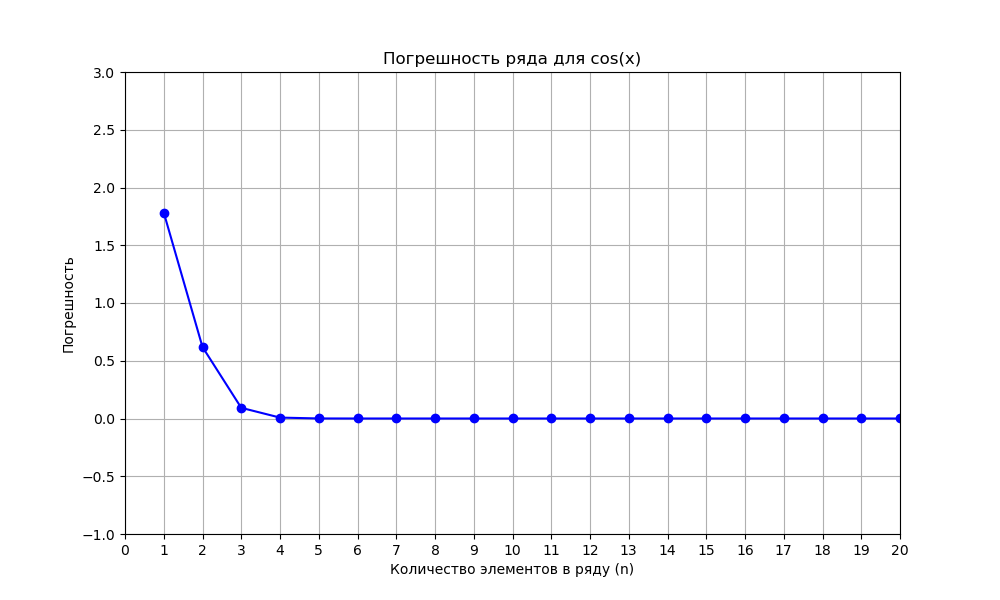
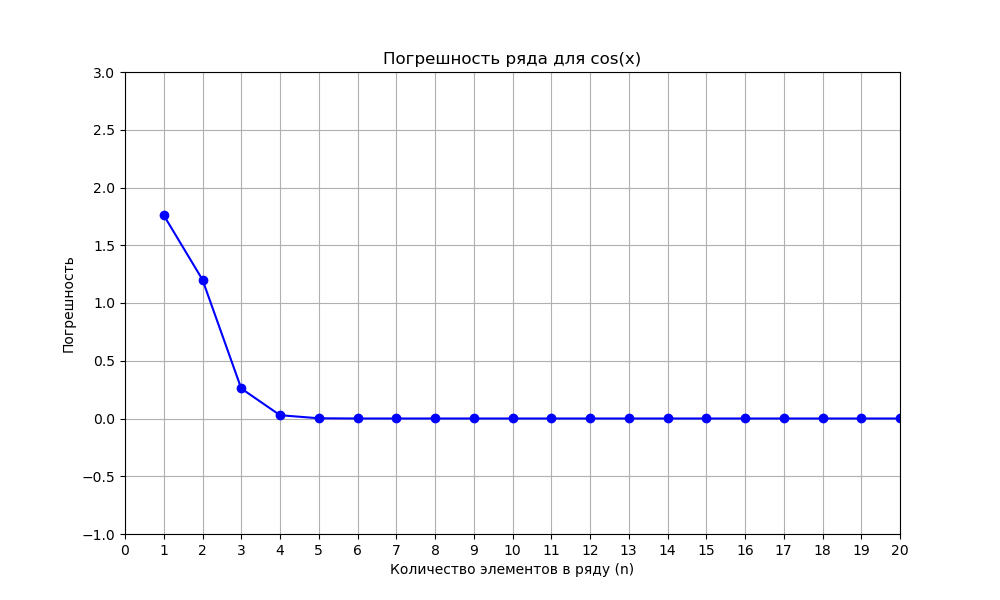
****

Рисунок 11.График попарного суммирования для sin(x).

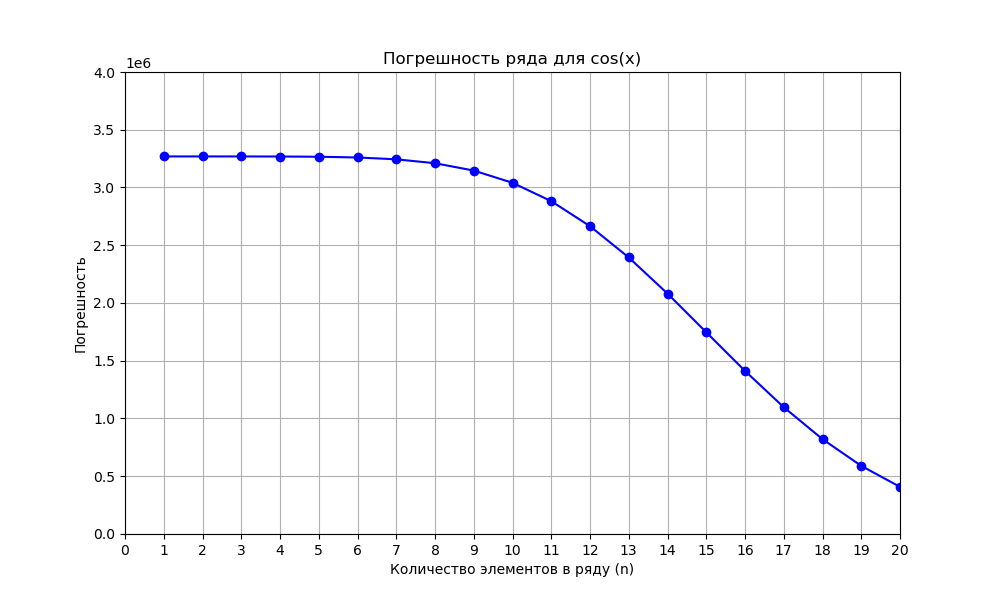
 Рисунок 12. График попарного суммирования для cos(x).

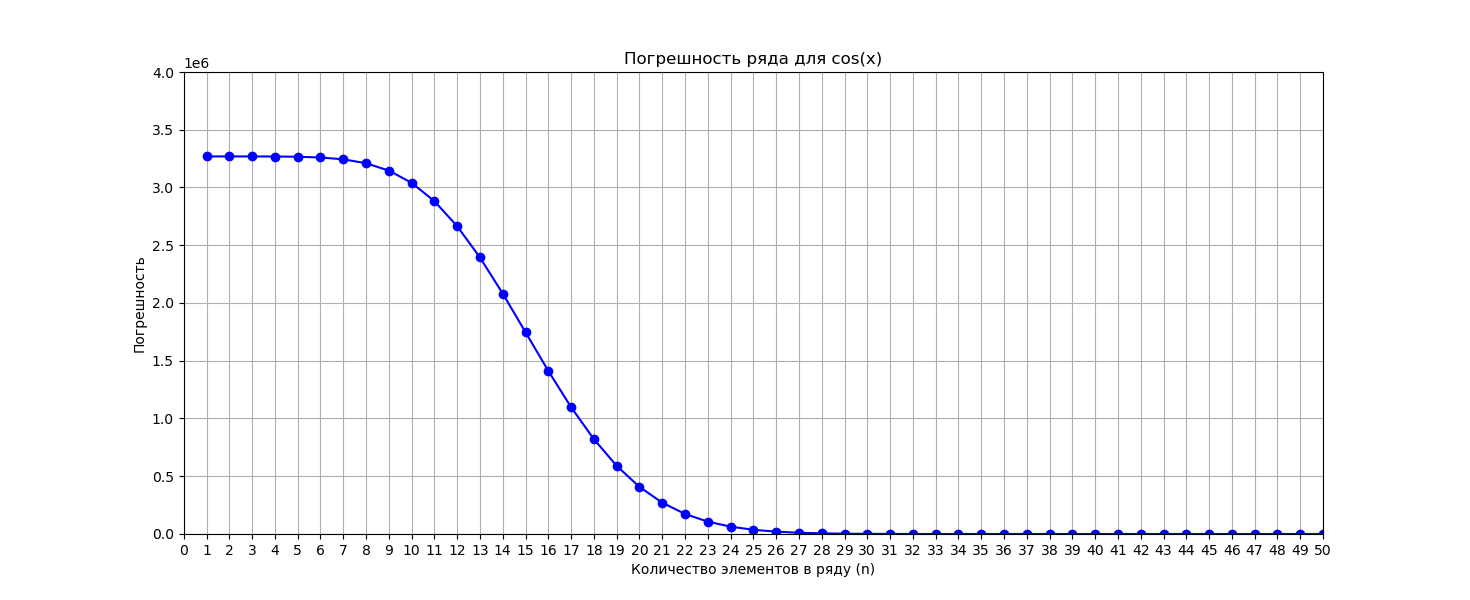
|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 1.759687900543213 |
| 2 | 1.201587200164795 |
| 3 | 0.259937763214111 |
| 4 | 0.028594017028809 |
| 5 | 0.001921057701111 |
| 6 | 0.000087022781372 |
| 7 | 0.000003099441528 |
| 8 | 0.000000119209290 |
| 9 | 0.000000178813934 |
| 10 | 0.000000178813934 |
| 11 | 0.000000178813934 |
| 12 | 0.000000178813934 |
| 13 | 0.000000178813934 |
| 14 | 0.000000178813934 |
| 15 | 0.000000178813934 |
| 16 | 0.000000178813934 |
| 17 | 0.000000178813934 |
| 18 | 0.000000178813934 |
| 19 | 0.000000178813934 |
| 20 | 0.000000178813934 |

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 1.783341169357300 |
| 2 | 0.618873834609985 |
| 3 | 0.092488110065460 |
| 4 | 0.007823228836060 |
| 5 | 0.000428140163422 |
| 6 | 0.000016093254089 |
| 7 | 0.000000774860382 |
| 8 | 0.000000298023224 |
| 9 | 0.000000298023224 |
| 10 | 0.000000298023224 |
| 11 | 0.000000298023224 |
| 12 | 0.000000298023224 |
| 13 | 0.000000298023224 |
| 14 | 0.000000298023224 |
| 15 | 0.000000298023224 |
| 16 | 0.000000298023224 |
| 17 | 0.000000298023224 |
| 18 | 0.000000298023224 |
| 19 | 0.000000298023224 |
| 20 | 0.000000298023224 |

Таблица 9. Попарное суммирование для sin(x). Таблица 10. Попарное суммирование для cos(x).

Для функций синуса и косинуса, кривые схожи с кривыми прямого суммирования. Погрешности также сходятся к нулю начиная с 3 элементов в ряду.

Рисунок 13. График попарного суммирования для exp(x) при 20 членов в ряду.

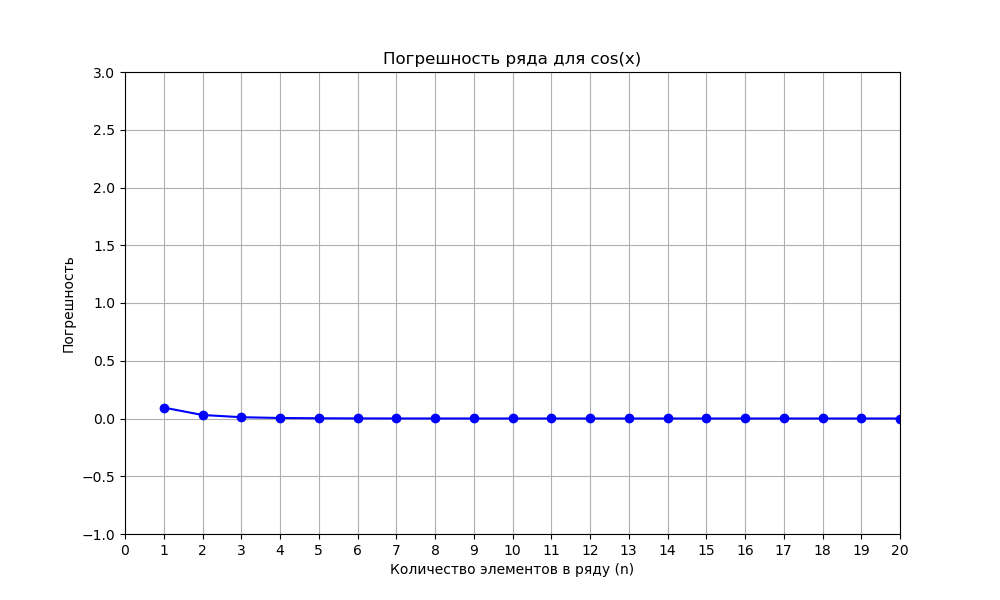
Рисунок 14. График попарного суммирования для exp(x) при 50 членов в ряду.

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0.094534903764725 |
| 2 | 0.030465096235275 |
| 3 | 0.011201560497284 |
| 4 | 0.004423439502716 |
| 5 | 0.001826554536819 |
| 6 | 0.000777602195740 |
| 7 | 0.000338464975357 |
| 8 | 0.000149816274643 |
| 9 | 0.000067204236984 |
| 10 | 0.000030457973480 |
| 11 | 0.000013917684555 |
| 12 | 0.000006407499313 |
| 13 | 0.000002980232239 |
| 14 | 0.000001370906830 |
| 15 | 0.000000655651093 |
| 16 | 0.000000298023224 |
| 17 | 0.000000149011612 |
| 18 | 0.000000059604645 |
| 19 | 0.000000029802322 |
| 20 | 0.000000000000000 |

Экспоненциальная функция никак не меняется, ее кривая все также плавно опускается и погрешность начинается сходится к нулю начиная с 25 элемента(см.рис. 14).

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 3269016.250000000000000 |
| 2 | 3269001.250000000000000 |
| 3 | 3268888.750000000000000 |
| 4 | 3268326.250000000000000 |
| 5 | 3266217.000000000000000 |
| 6 | 3259888.750000000000000 |
| 7 | 3244068.500000000000000 |
| 8 | 3210167.750000000000000 |
| 9 | 3146604.000000000000000 |
| 10 | 3040664.500000000000000 |
| 11 | 2881755.000000000000000 |
| 12 | 2665060.500000000000000 |
| 13 | 2394192.000000000000000 |
| 14 | 2081651.750000000000000 |
| 15 | 1746787.125000000000000 |
| 16 | 1411922.625000000000000 |
| 17 | 1097987.000000000000000 |
| 18 | 820985.250000000000000 |
| 19 | 590150.250000000000000 |
| 20 | 407912.000000000000000 |

Таблица 11. Попарное суммирование для exp(x) Таблица 12. Попарное суммирование для ln(1+x)

Рисунок 15. График попарного суммирования для ln(x+1).

Тут ситуация аналогичная, график погрешности и кривая никак не отличаются от графиков и кривых других методов суммирования(см. рис. 15).

Подводя итог, можно сказать что различия между методами суммирования крайне незначительны, погрешности почти что одинаковы, кроме графика синуса в обратном суммировании. Чтобы максимально точно посчитать функцию с помощью разложения, необходимо брать большое количество элементов в ряде. Для всех функций кроме экспоненциальной хватает около 15 для точного подсчета, для экспоненциальной же хватает 50 элементов чтобы посчитать значение с минимальной погрешностью.

# Заключение

В ходе данной лабораторной работы было реализовано вычисление функций , , с помощью их разложения в ряд Тейлора. Также было проведено сравнение результатов полученных с помощью библиотечных функций с результатами вычислений с помощью разложения.

# Приложение

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

typedef float (\*SumFunction)(float\*, int);

typedef void (\*FuncChoice)(float, int, float\*);

float reduce\_sinx(float x) {

const float pi\_over\_2=M\_PI/2.0f;

const float two\_pi=2.0f\*M\_PI;

x = fmodf(x, two\_pi);

if (x>=pi\_over\_2) {

x=M\_PI-x;

}

if (x>=pi\_over\_2) {

x-=M\_PI;

}

if (x>=pi\_over\_2) {

x=M\_PI-x;

}

return x;

}

float reduce\_cosx(float x) {

const float pi\_over\_2=M\_PI/2.0f;

const float two\_pi=2.0f\*M\_PI;

x = fmodf(x, two\_pi);

if (x>M\_PI) {

x=two\_pi-x;

}

if (x>pi\_over\_2) {

x=M\_PI-x;

}

return x;

}

void sinx(float x, int size, float\* chleni\_ryada) {

x = reduce\_sinx(x);

chleni\_ryada[0] = x;

for (int i = 1; i < size; i++) {

chleni\_ryada[i] = ((chleni\_ryada[i - 1]) \* (-x \* x)) / ((2 \* i + 1) \* (2 \* i));

}

}

void cosx(float x, int size, float\* chleni\_ryada) {

x=reduce\_cosx(x);

chleni\_ryada[0]=1.0;

for (int i=1; i<size; i++) {

chleni\_ryada[i] = ((chleni\_ryada[i - 1]) \* (-x \* x)) / ((2 \* i) \* (2 \* i - 1));

}

}

void expx(float x, int size, float\* chleni\_ryada) {

chleni\_ryada[0] = 1.0;

for (int i = 1; i < size; i++) {

chleni\_ryada[i] = chleni\_ryada[i - 1] \* x / i;

}

}

void lnx(float x, int size, float\* chleni\_ryada) {

chleni\_ryada[0] = x;

for (int i = 1; i < size; i++) {

chleni\_ryada[i] = chleni\_ryada[i - 1] \* (-(i \* x)) / (i + 1);

}

}

float sum\_pryam(float\* chleni\_ryada, int size) {

float summ = 0.0;

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

summ += chleni\_ryada[i];

}

return summ;

}

float sum\_obr(float\* chleni\_ryada, int size) {

float summ = 0.0;

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

summ += chleni\_ryada[i];

}

return summ;

}

float sum\_popar(float\* chleni\_ryada, int size) {

if (size == 1) {

return chleni\_ryada[0];

}

int new\_size = (size + 1) / 2;

float\* new\_chleni\_ryada = (float\*)malloc(new\_size \* sizeof(float));

if (new\_chleni\_ryada == NULL) {

printf("Ошибка выделения памяти.\n");

return 0;

}

for (int i = 0; i < size; i += 2) {

if (i + 1 < size) {

new\_chleni\_ryada[i / 2] = chleni\_ryada[i] + chleni\_ryada[i + 1];

} else {

new\_chleni\_ryada[i / 2] = chleni\_ryada[i];

}

}

float result = sum\_popar(new\_chleni\_ryada, new\_size);

free(new\_chleni\_ryada);

return result;

}

float compute\_taylor(float x, int size, float\* chleni\_ryada, SumFunction sumFunction, FuncChoice func) {

func(x, size, chleni\_ryada);

return sumFunction(chleni\_ryada, size);

}

int main() {

int choice\_Func, choice\_MethSum, size;

float x;

SumFunction sumFunctions[] = {sum\_pryam, sum\_obr, sum\_popar};

printf("Выберите функцию для аргумента:\n1 - sin(x)\n2 - cos(x)\n3 - exp(x)\n4 - ln(1+x)\n");

scanf("%d", &choice\_Func);

printf("Выберите метод суммирования:\n1 - Прямое суммирование\n2 - Обратное суммирование\n3 - Попарное суммирование\n");

scanf("%d", &choice\_MethSum);

printf("Введите значение аргумента для функции:\n");

scanf("%f", &x);

printf("Введите количество элементов для ряда:\n");

scanf("%d", &size);

if (choice\_Func < 1 || choice\_Func > 4 || choice\_MethSum < 1 || choice\_MethSum > 3) {

printf("Неверный выбор.\n");

return 1;

}

float teylor\_result = 0.0;

float math\_result = 0.0;

float\* chleni\_ryada = (float\*)malloc(size \* sizeof(float));

if (chleni\_ryada == NULL) {

printf("Ошибка выделения памяти. Похоже вы выбрали недопустимый размер.\n");

return 1;

}

switch (choice\_Func) {

case 1:

math\_result = sin(x);

break;

case 2:

math\_result = cos(x);

break;

case 3:

math\_result = exp(x);

break;

case 4:

if (x <= -1 || x > 1) {

printf("Ошибка: Ряд Тейлора для ln(1+x) сходится только при |x|<1\n");

free(chleni\_ryada);

return 1;

}

math\_result = log(1 + x);

break;

}

FuncChoice selectedFunc;

switch (choice\_Func) {

case 1:

selectedFunc = sinx;

break;

case 2:

selectedFunc = cosx;

break;

case 3:

selectedFunc = expx;

break;

case 4:

selectedFunc = lnx;

break;

}

SumFunction selectedSumFunction = sumFunctions[choice\_MethSum - 1];

teylor\_result = compute\_taylor(x, size, chleni\_ryada, selectedSumFunction, selectedFunc);

printf("Члены ряда Тейлора:\n");

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

printf("chleni\_ryada[%d] = %.15f\n", i, chleni\_ryada[i]);

}

printf("Результат (Тейлор): %.15f\n", teylor\_result);

printf("Результат (math.h): %.15f\n", math\_result);

printf("Разница: %.15f\n", fabs(teylor\_result - math\_result));

free(chleni\_ryada);

return 0;

}