ГОУ ВПО «Донецкий национальный университет»

Физико-технический факультет

Кафедра компьютерных технологий

**ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА**

**ЛАБОРАТОРНАЯ РАБОТА №1**

**ВЫЧИСЛЕНИЯ С ПЛАВАЮЩЕЙ ТОЧКОЙ**

Выполнил:

Ермоленко Евгений

Студент 2 уск.курса

группы ИВТ-4

Проверил:

Пшеничный К.А.

**Ход работы**

# Задание 1

Определить машинный  и машинную бесконечность для стандартной и повышенной точности вычислений с вещественными числами. Соответствуют ли полученные значения действительности?

Машинное эпсилон — числовое значение, меньше которого невозможно задавать относительную точность для любого алгоритма, возвращающего вещественные числа.

Машинная бесконечность — число, большее по модулю некоторого фиксированного для данного типа ЭВМ числа.

Машинное эпсилон вычислял путем деления заданного числа (в данном случае 1) на 2 до тех пор, пока оно не будет максимально приближено к нулю или равное ему. Эпсилон вычислял для двух точностей: стандартной (float) и повышенной (double).

Для решения всех заданий был использован язык программирования C#.

**Стандартная точность (float):**

static float EpsFloat(float number)

{

float prev = 0;

while (true)

{

if (number == 0)

{

break;

}

prev = number;

number /= 2;

}

return prev;

}

**Повышенная точность (double):**

static double EpsDouble(double number)

{

double prev = 0;

while (true)

{

if (number == 0)

{

break;

}

prev = number;

number /= 2;

}

return prev;

}

Машинную бесконечность вычислял путем умножения заданного числа (в данном случае 1) на 2 до тех пор, пока оно не достигнет значения равное бесконечности, ситуацию, когда число достигает бесконечности отлавливал конструкцией try{} catch{}, таким образом машинная бесконечность будет равна предыдущему значению. Машинную бесконечность вычислял для двух точностей: стандартной (float) и повышенной (double).

**Стандартная точность (float):**

static float InfinityFloat(float number)

{

float current = number;

float prev = 0;

while (true)

{

try

{

prev = current;

current \*= 2;

if (double.IsInfinity(current))

{

throw new Exception();

}

}

catch

{

return prev;

}

}

}

**Повышенная точность (double):**

static double InfinityDouble(double number)

{

double current = number;

double prev = 0;

while (true)

{

try

{

prev = current;

current \*= 2;

if (double.IsInfinity(current))

{

throw new Exception();

}

}

catch

{

return prev;

}

}

}

**Результаты вычислений:**

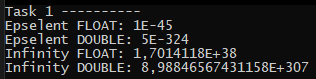


Рисунок 1.1 – Результаты вычислений машинной эпсилон и машинной бесконечности

Реальные машинные значения эпсилон и бесконечности для стандартной точности (float) для языка C# представлены ниже.



Рисунок 1.2 – Реальные машинные значения эпсилон и бесконечности для стандартной точности

Реальные машинные значения эпсилон и бесконечности для повышенной точности (double) для языка C# представлены ниже.



Рисунок 1.3 – Реальные машинные значения эпсилон и бесконечности для повышенной точности

**Вывод:**

Значения, которые были получены в результате вычислений соответствуют значениям для эпсилон и бесконечности, установленным в данном языке программирования, неточность возникла из-за алгоритма нахождения значений. Вычисления подтвердили, что точность вычислений с использованием double в разы выше, чем для float.

# Задание 2

Ряд Тейлора для функции ошибок имеет вид:

Этот ряд сходится для всех *x*.

Вычислить erf(3) по указанной формуле с точностью до нулевого члена ряда:

1. Без преобразования исходной формулы;

1.1. Расчёт факториала сделать с числом *n* целого типа;

1.2. Расчёт факториала сделать с числом *n* вещественного типа.

2. Повторить расчёты, изменив алгоритм вычислений очередного члена ряда через предыдущий член *an = kan-1,* т.е. избавиться от факториала.

## Без преобразований:

Расчёт факториала сделать с числом *n* целого типа.

Код расчета факториала для целого типа представлен ниже.

public static int Factorial(int n)

{

if (n == 0)

{

return 1;

}

else

{

return n \* Factorial(n - 1);

}

}

public static double TaylorSeries(int n)

{

double S = 0;

double result = 1;

double max = 1;

while (result != 0)

{

result = (Math.Pow(-1, n) \* Math.Pow(x, (2 \* n + 1))) / (Factorial(n) \* (2 \* n + 1));

if (max < Math.Abs(result))

{

max = Math.Abs(result);

}

S += result;

n += 1;

}

Console.WriteLine("Max S n - int: " + max);

return (2 / Math.Sqrt(Math.PI)) \* S;

}

Расчёт факториала сделать с числом *n* вещественного типа.

Код расчета факториала для целого типа представлен ниже.

public static double Factorial(double n)

{

if (n == 0)

{

return 1;

}

else

{

return n \* Factorial(n - 1);

}

}

public static double TaylorSeries(double n)

{

double S = 0;

double result = 1;

double max = 1;

while (result != 0)

{

result = (Math.Pow(-1, n) \* Math.Pow(x, (2 \* n + 1))) / (Factorial(n) \* (2 \* n + 1));

if (max < Math.Abs(result)) max = result;

S += result;

Console.WriteLine("При n равное " + n + "\t член ряда равен " + result + " | сумма на тек.итерации = " + S);

n += 1;

}

Console.WriteLine("\nMax S n - double: " + max);

return (2 / Math.Sqrt(Math.PI)) \* S;

}

**Результаты вычислений:**



Рисунок 1.4 – Результат вычисления суммы ряда без преобразований формулы для n целого типа



Рисунок 1.5 – Максимальное и минимальное значение для переменных типа Int



Рисунок 1.6 – Результат вычисления суммы ряда без преобразований формулы для n вещественного типа

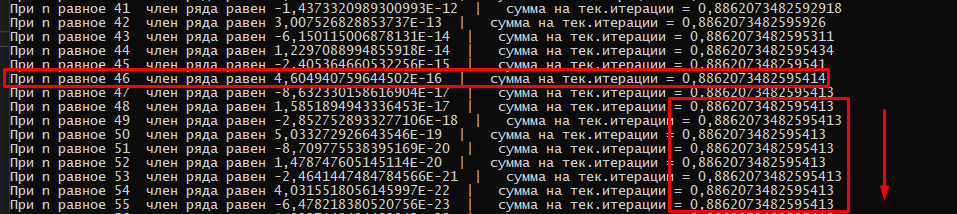


Рисунок 1.7 – Итерация при которой сумма ряда не изменялась

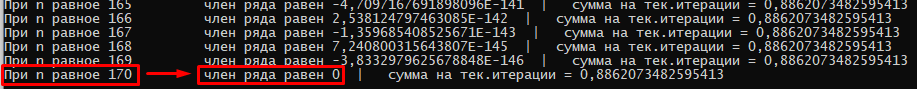


Рисунок 1.8 – Итерация при которой член ряда равен 0

**Вывод:**

Так как максимальное значение для типа Int (рис. 1.5), который принимала переменная n равна 2 147 483 647 при вычислении факториала происходит **переполнение стека** (рис.1.4).

При вычислении суммы ряда удалось обнаружить, что при достижении 46 (n=46) итерации сумма не изменяется (рис.1.7). Также стоит отметить, что при вычислении удалось заметить, что ряд сходится, то есть член ряда равняется нулю при n равное 170 (рис.1.8).

**Задание 2.2**

Алгоритм:

Для того, чтобыповторить расчёты, изменив алгоритм вычислений очередного члена ряда через предыдущий член *an = kan-1,* т.е. избавиться от факториала, необходимо произвести преобразования:

Далее для расчета по формуле необходимо определить нулевой член ряда ( ). При n =0.

Так как по услови*ю* , то нулевой член ряда равен .

public static double koeff(double n)

{

return ((-(Math.Pow(x, 2))) \* (2 \* n - 1)) / (n \* (2 \* n + 1));

}

public static double TSModified()

{

double a = 3;

double S = a;

double n = 1;

while (a != 0)

{

a = a \* koeff(n);

S += a;

n += 1;

}

return (2 / Math.Sqrt(Math.PI)) \* S;

}

**Результаты вычислений:**

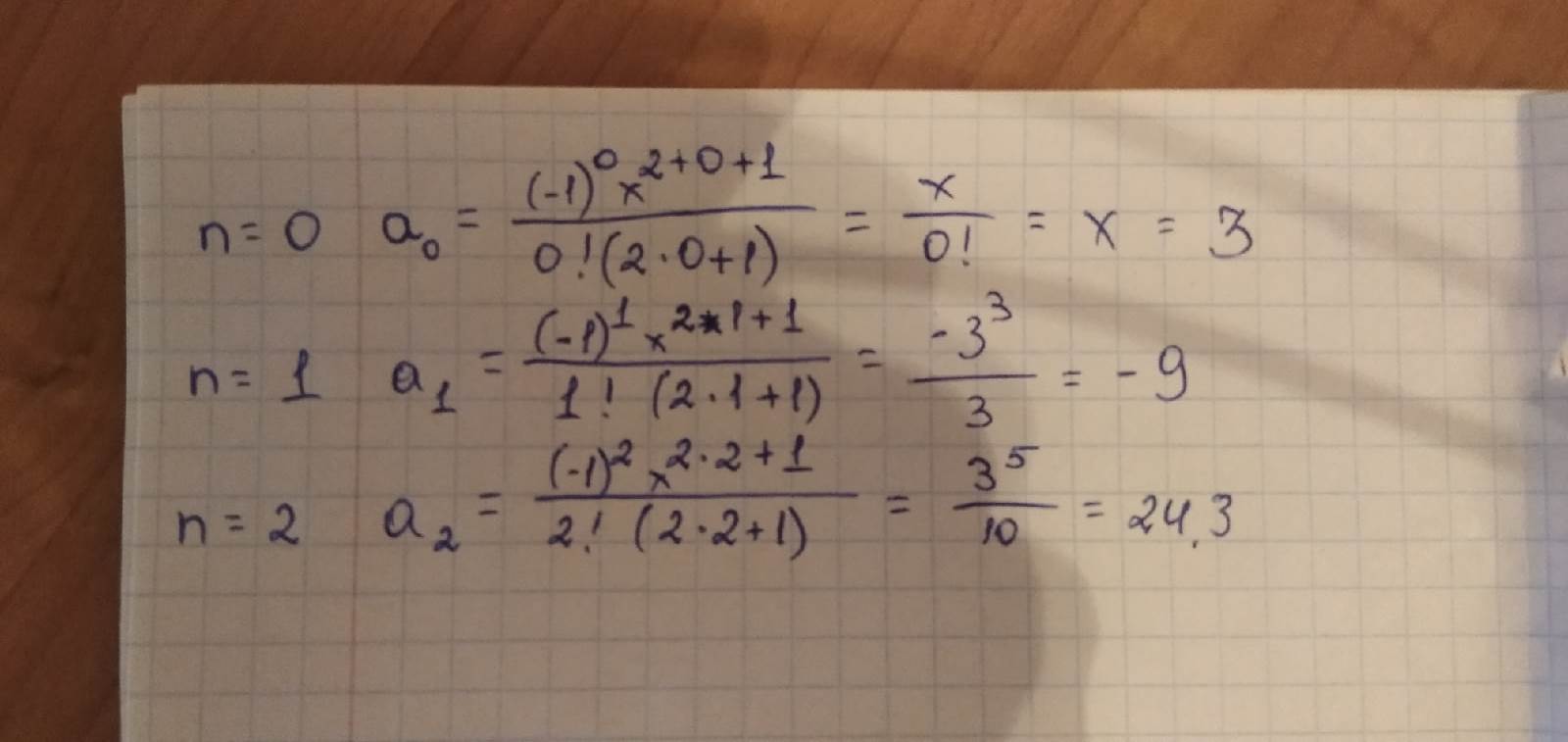


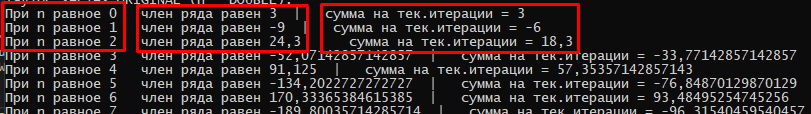
Рисунок 1.9 – Результат вычисления суммы ряда с преобразованием формулы для n вещественного типа

**Вывод:**

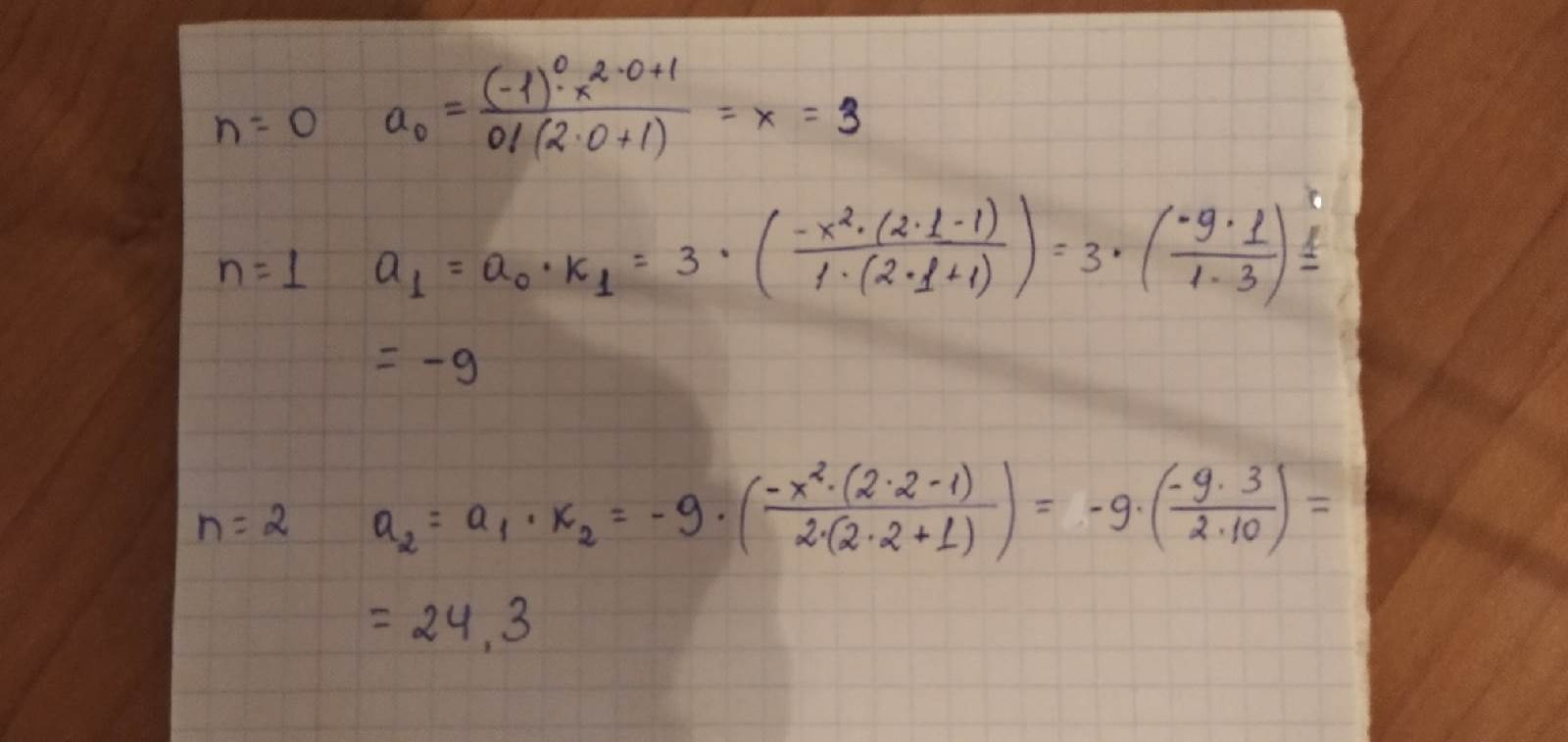
Первые три члена обоих алгоритмов, были просчитаны вручную и результаты программного вычисления с ними сошлись из этого следует, что алгоритмы были запрограммированы правильно.

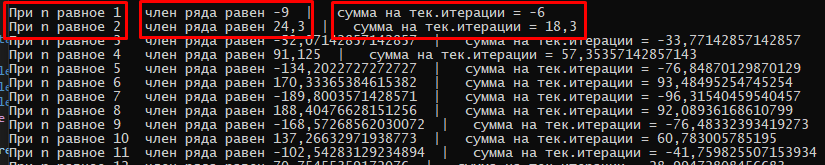
Для алгоритма с вычислением факториала:





Для алгоритма без вычисления факториала:





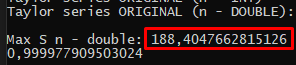
Привычислении суммы по предоставленной формуле происходит быстрый рост членов ряда, что приводит к выходу значений за границы диапазона. Чтобы этого избежать, необходимо было произвести ряд преобразований, чтобы избавиться от факториала, т.к. факториал становится причиной быстрого роста значения, что влечет за собой переполнения.

**Проведём анализ полученных результатов:**

Для вычислений без факториала произошла катастрофическая потеря верных знаков, что привело к округлению конечного значения. доверять можно лишь первым 14 верным знакам потому, что максимальное значение хранит 17 верных знаков.



В алгоритме с вычислением факториала получаем следующее:



Здесь ситуация аналогична предыдущему алгоритму.

В итоге можно заключить, что алгоритм с вычислением факториала производит вычисления точнее. Но, если для области, в которой применяется результат erf(x) функции, достаточно 1 верных знаков (учитывая округления и накопление ошибок в результате), то подойдут оба алгоритма.

**Контрольные вопросы**

1. Какова причина возникновения округлений при работе с вещественными числами?

Для записи приближенных чисел с верными цифрами применяется округление.

Точные числа также требуется округлить, если количество используемых разрядов ограничено.

Округлением (по дополнению) числа называется запись этого числа с меньшим количеством разрядов по следующему правилу: если первая отбрасываемая цифра больше или равна 5, то последнюю оставляемую цифру увеличивают на единицу. При округлении чисел возникает погрешность, которую также надо учитывать.

При сложении и вычитании приближенных чисел с различным числом верных цифр после запятой результат округляется по наименьшему числу верных цифр после запятой у исходных данных.

При умножении и делении приближенных чисел с различным числом верных цифр производится округление результата по минимальному числу верных цифр у исходных данных.

1. Что такое «катастрофическая потеря верных знаков»?

Значительное увеличение погрешности результата по сравнению с погрешностью аргумента называется катастрофической потерей точности.

1. Что такое и как проявляют себя исчезновение и переполнение вещественных чисел?

Переполнение происходит при перемножении очень больших чисел, а также при делении на очень малое число или ноль. Исчезновение порядка происходит при перемножении очень малых чисел. Результат обыкновенно заменяется нулем, что часто приводит к неожиданным последствиям.

1. Что такое накопление вычислительных погрешностей? Напишите пример алгоритма, накапливающего вычислительные погрешности.

При численном интегрировании в результате округления на каждом шаге в некоторой неточности формул происходит постепенное накопление погрешности с увеличением числа шагов. В общем, невозможно определить, какой величины ошибка накапливается при интегрировании, однако при помощи теории ошибок можно установить, какую вероятную ошибку следует ожидать после любого числа шагов. Общая теория накопления погрешностей при численном интегрировании указывает, что после N шагов вероятная ошибка двойного интеграла равна 0,1124 (в единицах последнего десятичного знака). Это означает, что при большом числе примеров около половины погрешностей будет больше этого значения, а половина — меньше.

1. Что такое чувствительность решения задачи к её параметрам, и к чему может приводить высокая чувствительность отдельных параметров?

Чувствительность к параметрам выражается в сильной зависимости порядка одного или нескольких параметров. Для примера возьмем формулу расчета **x** в квадратном уравнении https://refnew.ru/laboratornaya-rabota-1-vichisleniya-s-plavayushej-tochkoj-furs/3563_html_m34c09541.gif. Допустим, что **а** и **с** имеют порядок 1, а **b** имеет порядок 20. Таким образом можно заметить, что формула будет практически полностью зависеть от значения b, при этом будут игнорироваться значения a и c. В итоге это приводит к кардинально неправильному результату.