## Лабораторная работа № 1.2
## «Алгоритмы вычисления значений степенных рядов (спецфункций)»

В предыдущей лабораторной работе мы изучили и реализовали алгоритмы вычисления конечных $S = \sum^{N} a_k$ и бесконечных $S = \sum^{\infty} a_k$ сумм для числовых рядов.

Во многих расчетных задачах прикладной математики и физики для вычислений значений высших трансцендентных функций приходится пользоваться их представлениями в виде бесконечных степенных рядов, например, $f(x) = \sum_{k=k_o}^{\infty} \alpha_k x^k$, где $\alpha_k = \alpha_k(k)$ – некоторые числовые коэффициенты, $k_o$ – начальное значение индекса суммирования.

Очевидно, что при непосредственных компьютерных вычислениях количество $K$ членов степенного ряда, действительно участвующих в суммировании, ограничено и выбирается программным путем с учетом требуемой точности конечного результата, т.е. мы будем иметь итоговое значение $f(x) \approx \sum_{k=0}^{K} \alpha_k x^k$.

Если нам задана абсолютная погрешность вычислений $\varepsilon$, то в качестве критерия выбора наибольшего значения $K$, в частности, может быть принято условие $\left|\alpha_{K+1} x^{K+1}\right| < \varepsilon$.

Однако выполнение этого условия, или какого-либо другого аналогичного условия, отнюдь не гарантирует нам качество получаемого результата, и требует дополнительных тестовых вычислений для выяснения границ применимости разработанного алгоритма.

Продемонстрируем сказанное на примере достаточно простых трансцендентных функций: $\sin(x)$, $\cos(x)$, $e^x$, $\sinh(x)$ и $\cosh(x)$, для которых известны следующие представления в виде степенных рядов:

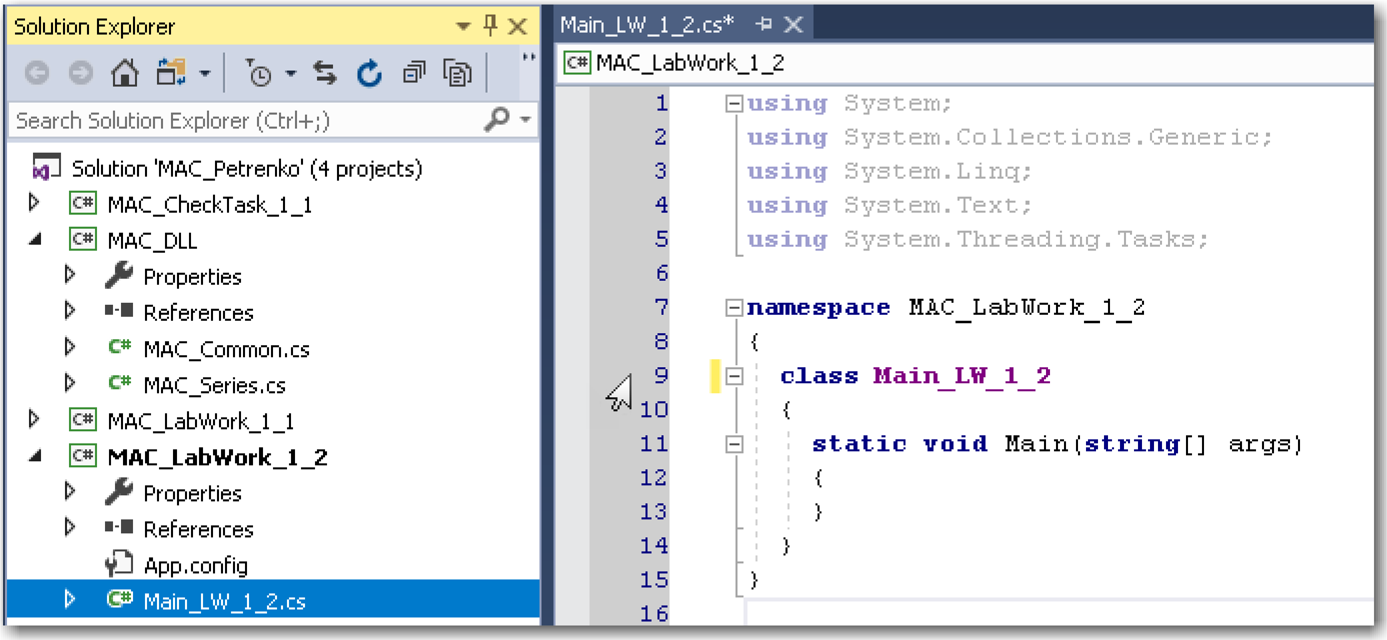
$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, \qquad \cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, \tag{1.2.1}$$

$$e^x = \exp(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots, \tag{1.2.2}$$

$$\sinh(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \ldots, \qquad \cosh(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \ldots \tag{1.2.3}$$

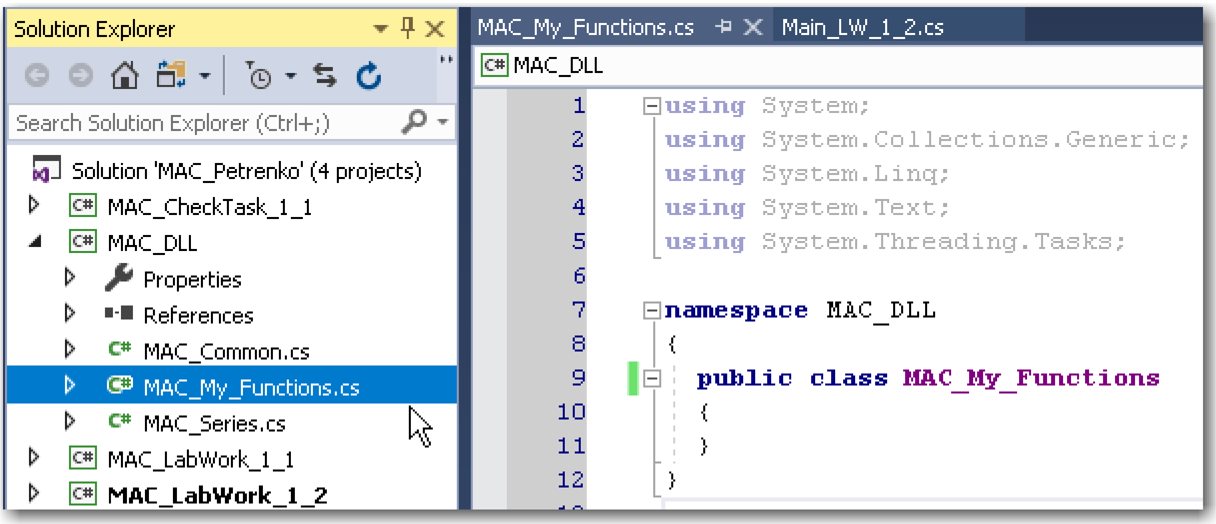
Одновременно мы разберем «технологию» составления алгоритма вычисления значений для степенных рядов (степенных представлений трансцендентных функций). Эта технология Вам непосредственно понадобится при выполнении последующего контрольного задания.

Итак, в рабочем пространстве **PetrenkoIN** генерируем проект **MAC_LabWork_1_2** консольного приложения. Переименовываем его основной класс в **Main_LW_1_2**.




Разработку алгоритмов вычисления функций (1.2.1)–(1.2.3) будем выполнять в отдельном классе **MAC_My_Functions** имеющейся библиотеки **MAC_DLL** (проект **MAC_DLL**).

Добавим новый класс **MAC_My_Functions** в библиотеку **MAC_DLL**.




Стандартным способом добавим библиотеку **MAC_DLL** в раздел **References** нового проекта **MAC_LabWork_1_2**, что позволит упоминать библиотеку **MAC_DLL** в директиве **using** консольного приложения **MAC_LabWork_1_2**, а также вызывать ее различные методы.

Прежде чем непосредственно программировать алгоритм вычисления тригонометрического синуса запишем выражение для общего члена ряда (1.2.1) и условимся о нумерации последовательности членов этого степенного ряда:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots + (-1)^{k+1}\frac{x^{2k-1}}{(2k-1)!} + \ldots = \sum_{k=1}^{\infty}(-1)^{k+1}\frac{x^{2k-1}}{(2k-1)!}. \quad (1.2.4)$$

Представление (1.2.4) подразумевает, что индекс $k$ имеет инкремент, равный 1.

Теперь запишем аналогичное представление для тригонометрического косинуса:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots + (-1)^{k}\frac{x^{2k}}{(2k)!} + \ldots = \sum_{k=0}^{\infty}(-1)^{k}\frac{x^{2k}}{(2k)!}. \quad (1.2.5)$$

Для удобства вычисления членов степенных рядов (1.2.4) и (1.2.5) можно построить соответствующие рекуррентные формулы.

Если для (1.2.4) ввести обозначения:

$$p_1 = x,\ p_2 = -\frac{x^3}{3!},\ p_3 = \frac{x^5}{5!},\ p_4 = -\frac{x^7}{7!}\ldots,\quad p_k = (-1)^{k+1}\frac{x^{2k-1}}{(2k-1)!},$$

то рекуррентная последовательность будет следующей:

$$p_1 = x,\quad p_k = -p_{k-1}\times\frac{x^2}{(2k-2)\cdot(2k-1)},\quad k > 1. \quad (1.2.6)$$

Преимущество такого способа вычисления значений для членов $(-1)^{k+1}\dfrac{x^{2k-1}}{(2k-1)!}$ степенного ряда (1.2.4) очевидно.

Во-первых, мы избегаем явного возведения аргумента $x$ в большие степени $(2k-1)$.

Во-вторых, нам не приходится явно вычислять значения факториала для больших значений индекса $k$.

Помимо этого, если ввести обозначение $\tilde{x} = x / 2$, можно повысить качество вычислений по алгоритму (1.2.6):

$$p_1 = x,\quad p_k = -p_{k-1}\times\left(\frac{\tilde{x}}{k-1.0}\right)\times\left(\frac{\tilde{x}}{k-0.5}\right),\quad k > 1. \quad (1.2.7)$$

Построим аналогичные рекуррентные формулы для тригонометрического косинуса.

Из формулы (1.2.5) имеем:

$$p_0 = 1,\ p_1 = -\frac{x^2}{2!},\ p_2 = \frac{x^4}{4!},\ p_3 = -\frac{x^6}{6!}\ldots,\quad p_k = (-1)^k \frac{x^{2k}}{(2k)!}.$$

Тогда рекуррентная последовательность для функции $\cos(x)$ будет следующей:

$$p_0 = 1.0,\quad p_k = -p_{k-1} \times \frac{x^2}{(2k-1)\cdot(2k)},\ k > 0, \tag{1.2.8}$$

или

$$p_0 = 1.0,\quad p_k = -p_{k-1} \times \left(\frac{\tilde{x}}{k-0.5}\right) \times \left(\frac{\tilde{x}}{k}\right),\ k > 0. \tag{1.2.9}$$

Таким образом, для вычисления тригонометрического синуса и косинуса у нас подготовлены следующие алгоритмы:

$$\sin(x) = p_1 + \sum_{k=2}^{K} p_k,\quad \text{где для индекса } K \text{ выполняется условие } |p_K| < \varepsilon, \tag{1.2.10}$$

$$\text{и}\quad p_1 = x,\quad p_k = -p_{k-1} \times \left(\frac{\tilde{x}}{k-1.0}\right) \times \left(\frac{\tilde{x}}{k-0.5}\right),\ k > 1,\ \tilde{x} = x/2.$$

$$\cos(x) = p_0 + \sum_{k=1}^{K} p_k,\quad \text{где для индекса } K \text{ выполняется условие } |p_K| < \varepsilon, \tag{1.2.11}$$

$$\text{и}\quad p_0 = 1.0,\quad p_k = -p_{k-1} \times \left(\frac{\tilde{x}}{k-0.5}\right) \times \left(\frac{\tilde{x}}{k}\right),\ k > 0,\ \tilde{x} = x/2.$$

Прежде чем приступать к непосредственному программированию (1.2.10) и (1.2.11), дополним эти алгоритмы очевидными выражениями:

$$\sin(0) = 0 \text{ и } \cos(0) = 1. \tag{1.2.12}$$

Абсолютная погрешность $\varepsilon$ вычислений будет второй переменной во всех разрабатываемых функциях, что будут размещаться в классе **`MAC_My_Functions`**.

Таким способом (при необходимости) мы сможем изменять это значение.

Сказанное реализуем следующим образом:

```
using ...

namespace MAC_DLL
{
  public class MAC_My_Functions
  {
    public static double MySin(double x, double eps)
    {
      if (x == 0) return 0.0;                                      // (1.2.12)
      double sin = x, pk = x, x2 = 0.5 * x;
      for (int k = 2; Math.Abs(pk) > eps; k++)
      {
        pk = -pk * (x2 / (k - 1.0)) * (x2 / (k - 0.5)); sin += pk; // (1.2.10)
      }
      return sin;
    }
    public static double MyCos(double x, double eps)
    {
      if (x == 0) return 1.0;                                      // (1.2.12)
      double cos = 1.0, pk = 1.0, x2 = 0.5 * x;
      for (int k = 1; Math.Abs(pk) > eps; k++)
      {
        pk = -pk * (x2 / (k - 0.5)) * (x2 / k); cos += pk;         // (1.2.11)
      }
      return cos;
    }
  }
}
```

Теперь нам следует протестировать разработанные функции **`MySin()`** и **`MyCos()`** с использованием известного тождественного равенства:

$$\forall x \in (-\infty, +\infty) \text{ имеет место } \cos^2(x) + \sin^2(x) \equiv 1. \qquad (1.2.13)$$

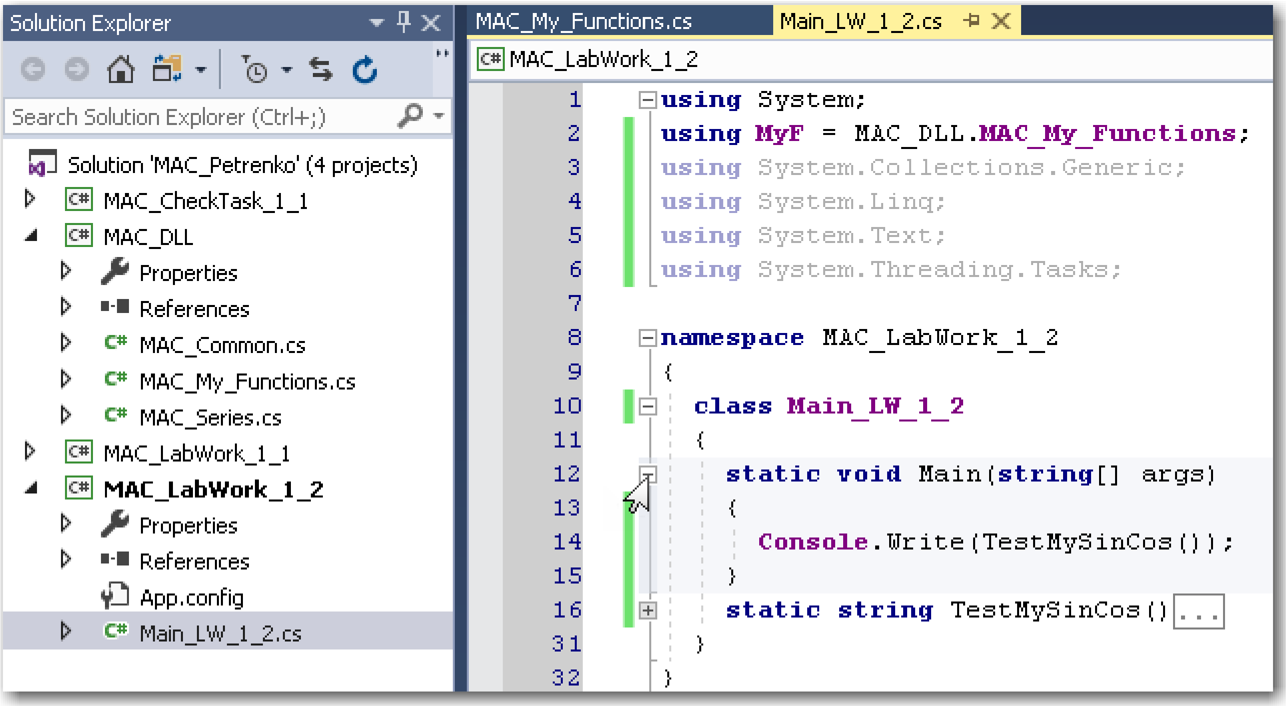
С этой целью создадим отдельную функцию в классе **`Main_LW_1_2`** проекта **`MAC_LabWork_1_2`** консольного приложения данной лабораторной работы:

```
    static string TestMySinCos()
    {
      string txt = " TestMySinCos() \r\n";
      double unit, error, e = 1.0E-20;
      for (double x = 1.0; x <= 40.0; x += 1.0)
      {
        unit = MyF.MyCos(x, e) * MyF.MyCos(x, e)          // (1.2.13)
             + MyF.MySin(x, e) * MyF.MySin(x, e);
        error = Math.Abs(1.0 - unit);
        txt += string.Format("{0:F1}", x).PadLeft(7)
             + string.Format("{0:F16}", unit).PadLeft(20)
             + string.Format("{0:F16}", error).PadLeft(20) + "\r\n";
      }
      return txt;
    }
```

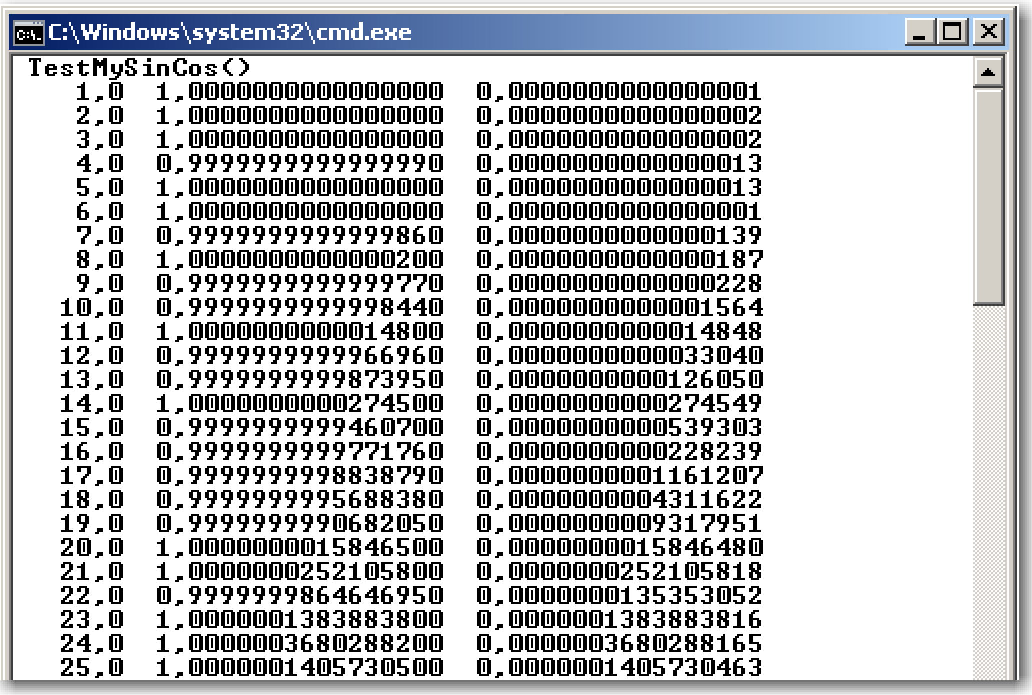
Эта функция строит таблицу значений выражения (1.2.13) – переменная **unit** – для последовательности значений аргумента $x = 1, 2, \ldots, 40$.

Кроме этого, определяется ошибка вычислений – переменная **error**, которая характеризует абсолютную погрешность вычисления значения **unit** по сравнению с ее «эталонным» значением $1.0$.

Вызов метода **TestMySinCos()** выполняем в основной исполняемой функции **Main**:




Имеем следующие результаты:



```
   26,0  0,9999981727821730  0,0000018272178274
   27,0  0,9999912771265990  0,0000087228734014
   28,0  1,0000034526206300  0,0000034526206274
   29,0  0,9999248202738160  0,0000751797261841
   30,0  1,0000368001933100  0,0000368001933146
   31,0  0,9995016843927710  0,0004983156072287
   32,0  1,0002522420209800  0,0002522420209836
   33,0  1,0006986760814700  0,0006986760814740
   34,0  1,0087068073139800  0,0087068073139804
   35,0  0,9977554988075770  0,0022445011924234
   36,0  0,9308503904004300  0,0691496095995703
   37,0  1,0846078397828100  0,0846078397828092
   38,0  1,0306121336205100  0,0306121336205103
   39,0  0,5246812624366070  0,4753187375633930
   40,0  3,0834366752911700  2,0834366752911700
Для продолжения нажмите любую клавишу . . .
```

Приведенная таблица хорошо демонстрирует область применимости разработанных алгоритмов для тригонометрических функций $\sin(x)$ и $\cos(x)$, когда они вычисляются в виде степенных рядов.

При увеличении абсолютного значения аргумента $x$ погрешность вычислений быстро возрастает и при $x \approx 20$ составляет $\sim 10^{-9}$. И это не смотря на то, что в методы **`MySin()`** и **`MyCos()`** передавалось значение $\varepsilon = 10^{-20}$.

При значениях же аргумента $x > 30$ результаты вычислений можно вообще считать неприемлемыми математически.

Анализ данной ситуации и выявление ее причин мы проведем несколько позже.

По аналогии с тригонометрическими функциями, выполним алгоритмизацию гиперболических функций $\sinh(x)$ и $\cosh(x)$, а также экспоненциальной функции $e^x$.

Что касается гиперболических функций $\sinh(x)$ и $\cosh(x)$ – то тут все очевидно, т.к. их представления в виде степенных рядов схожи с представлениями для тригонометрических функций $\sin(x)$ и $\cos(x)$, с той лишь разницей, что эти ряды являются знакопостоянными.

Поэтому алгоритмы для $\sinh(x)$ и $\cosh(x)$ получаем сразу из (1.2.10)–(1.2.11):

$$\sinh(x) = p_1 + \sum_{k=2}^{K} p_k\,, \quad \text{где для индекса } K \text{ выполняется условие } \left|p_K\right| < \varepsilon\,, \qquad (1.2.14)$$

и $$p_1 = x\,, \quad p_k = p_{k-1} \times \left(\frac{\tilde{x}}{k-1.0}\right) \times \left(\frac{\tilde{x}}{k-0.5}\right),\ k > 1\,,\ \tilde{x} = x\,/\,2\,, \quad \sinh(0) = 0\,.$$

$$\cosh(x) = p_0 + \sum_{k=1}^{K} p_k\,, \quad \text{где для индекса } K \text{ выполняется условие } \left|p_K\right| < \varepsilon\,, \qquad (1.2.15)$$

$$\text{и} \qquad p_0 = 1.0\,, \quad p_k = p_{k-1} \times \left(\frac{\tilde{x}}{k-0.5}\right) \times \left(\frac{\tilde{x}}{k}\right),\ k > 0\,,\ \ \tilde{x} = x\,/\,2\,, \quad \cosh(0) = 1\,.$$

Алгоритм для экспоненциальной функции будет следующим:

$$\exp(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^k}{k!} + \ldots = 1 + \sum_{k=1}^{\infty} \frac{x^k}{k!}, \text{ откуда}$$

$$\exp(x) = p_0 + \sum_{k=1}^{K} p_k\,, \quad \text{где для индекса } K \text{ выполняется условие } \left|p_K\right| < \varepsilon\,, \qquad (1.2.16)$$

$$\text{и} \quad p_0 = 1.0\,, \ \ p_k = p_{k-1} \times \frac{x}{k},\ k > 0\,, \quad \exp(0) = 1\,.$$

Самостоятельно дополните класс **MAC_My_Functions** библиотеки **MAC_DLL** соответствующими методами (функциями) **MySinh()**, **MyCosh()** и **MyExp()**:

```
using ...

namespace MAC_DLL
{
    public class MAC_My_Functions
    {
        public static double MySin(double x, double eps)...
        public static double MyCos(double x, double eps)...

        public static double MyExp(double x, double eps)...

        public static double MySinh(double x, double eps)...
        public static double MyCosh(double x, double eps)...
    }
}
```

Для тестирования алгоритмов (1.2.14)–(1.2.15) гиперболических функций $\sinh(x)$ и $\cosh(x)$ воспользуемся известным тождеством:

$$\forall x \in (-\infty, +\infty) \text{ имеет место } \cosh^2(x) - \sinh^2(x) \equiv 1\,. \qquad (1.2.17)$$

По аналогии с предыдущей тестовой функцией **TestMySinCos()** основного приложения **MAC_LabWork_1_2** создадим отдельную функцию **TestMySinhCosh()**:

```
static string TestMySinhCosh()
{
  string txt = " TestMySinhCosh() \r\n";
  double unit, error, e = 1.0E-20;
  for (double x = 0.0; x <= 20.0; x += 1.0)
  {
    unit = (MyF.MyCosh(x, e) - MyF.MySinh(x, e)) *     // (1.2.17)
           (MyF.MyCosh(x, e) + MyF.MySinh(x, e));
    error = Math.Abs(1.0 - unit);
    txt += string.Format("{0:F1}", x).PadLeft(7)
         + string.Format("{0:F16}", unit).PadLeft(23)
         + string.Format("{0:F16}", error).PadLeft(23) + "\r\n";
  }
  return txt;
}
```

Эта функция строит таблицу значений выражения (1.2.17) – переменная **unit** – для последовательности значений аргумента $x = 1, 2, \ldots, 20$. Здесь также определяется ошибка вычислений – переменная **error**, которая характеризует абсолютную погрешность вычисления значения **unit** по сравнению с её «эталонным» значением $1.0$.

Вызов метода **TestMySinhCosh()** выполняем в основной исполняемой функции **Main()**:

```
static void Main(string[] args)
{
  //Console.Write(TestMySinCos());
  Console.Write(TestMySinhCosh());
}
```

Имеем следующие результаты:

```
C:\Windows\system32\cmd.exe
 TestMySinhCosh()
    0,0      1,0000000000000000      0,0000000000000000
    1,0      1,0000000000000000      0,0000000000000002
    2,0      1,0000000000000100      0,0000000000000075
    3,0      0,9999999999999760      0,0000000000000243
    4,0      1,0000000000001500      0,0000000000001477
    5,0      0,9999999999993590      0,0000000000006412
    6,0      0,9999999999904670      0,0000000000095327
    7,0      1,0000000003376900      0,0000000003376890
    8,0      1,0000000011844900      0,0000000011844934
    9,0      1,0000000068180000      0,0000000068179984
   10,0      0,9999998262328400      0,0000001737671603
   11,0      0,9999994433309850      0,0000005566690152
   12,0      1,0000045997156800      0,0000045997156788
   13,0      0,9999957889371090      0,0000042110628915
   14,0      1,0001711787275500      0,0001711787275465
   15,0      0,9978380462413040      0,0021619537586963
   16,0      1,0013760747215300      0,0013760747215268
   17,0      0,8098579003743000      0,1901420996257010
   18,0      1,4676146761476400      0,4676146761476370
   19,0      5,3191870738025700      4,3191870738025700
   20,0    -28,9180991297835000     29,9180991297835000
Для продолжения нажмите любую клавишу . . .
```

Приведенная таблица отражает область применимости разработанных алгоритмов для гиперболических функций $\sinh(x)$ и $\cosh(x)$, когда они вычисляются в виде степенных рядов.

Ситуация здесь значительно хуже, чем для тригонометрических функций.

Погрешность вычислений делает данные алгоритмы бесполезными уже при $x > 10$.

Причина такого падения точности вычислительного алгоритма кроется в знакопостоянстве членов степенных рядов для функций $\sinh(x)$ и $\cosh(x)$.

Ничем другим (в данном контексте) тригонометрические функции $\sin(x)$ и $\cos(x)$ не отличаются от гиперболических функций $\sinh(x)$ и $\cosh(x)$.

Укажем две основные причины данной вычислительной проблемы.

Во-первых, это ограниченность представления действительных чисел типом **`double`**, которая обеспечивает 16 цифр в мантиссе числа с плавающей точкой.

Во-вторых, это «катастрофическая» потеря значащих цифр результата, когда из одного большого действительного числа вычитается другое большое действительное число, приблизительно равное первому по абсолютной величине.

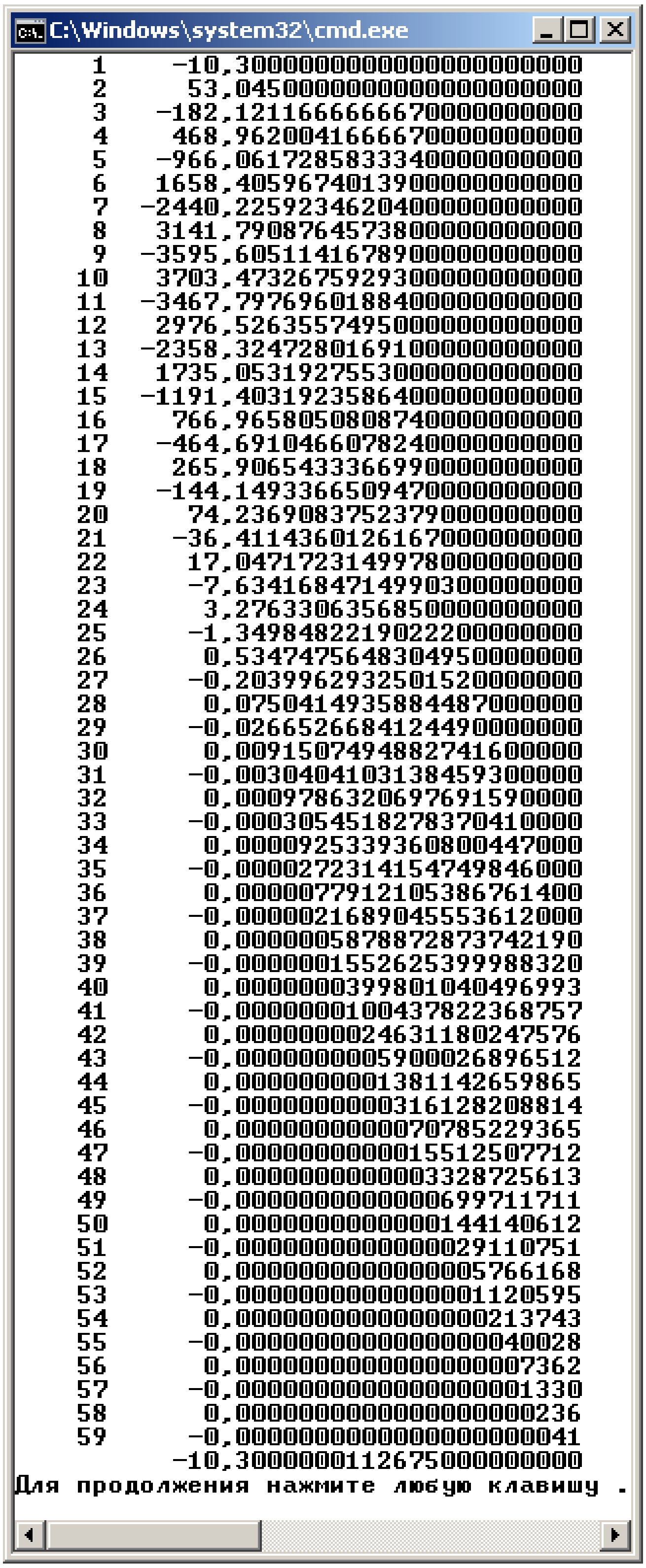
Сказанное можно проследить на следующем наглядном примере.

«Проверим» известное свойство:

$$\ln(\exp(-10.3)) = -10.3.$$

При этом для вычисления экспоненты будем использовать собственную функцию **`MyExp()`**, в которую дополнительно поместим оператор вывода значения члена степенного ряда.

В качестве результата будем иметь следующую таблицу числовых данных (приводится справа от текста):



C:\Windows\system32\cmd.exe

| | |
|---|---|
| 1 | -10,3000000000000000000000 |
| 2 | 53,0450000000000000000000 |
| 3 | -182,1211666666670000000000 |
| 4 | 468,9620041666670000000000 |
| 5 | -966,0617285833340000000000 |
| 6 | 1658,4059674013900000000000 |
| 7 | -2440,2259234620400000000000 |
| 8 | 3141,7908764573800000000000 |
| 9 | -3595,6051141678900000000000 |
| 10 | 3703,4732675929300000000000 |
| 11 | -3467,7976960188400000000000 |
| 12 | 2976,5263557495000000000000 |
| 13 | -2358,3247280169100000000000 |
| 14 | 1735,0531927553000000000000 |
| 15 | -1191,4031923586400000000000 |
| 16 | 766,9658050808740000000000 |
| 17 | -464,6910466078240000000000 |
| 18 | 265,9065433366990000000000 |
| 19 | -144,1493366509470000000000 |
| 20 | 74,2369083752379000000000 |
| 21 | -36,4114360126167000000000 |
| 22 | 17,0471723149978000000000 |
| 23 | -7,6341684714990300000000 |
| 24 | 3,2763306356850000000000 |
| 25 | -1,3498482219022200000000 |
| 26 | 0,5347475648304950000000 |
| 27 | -0,2039962932501520000000 |
| 28 | 0,0750414935884487000000 |
| 29 | -0,0266526684124490000000 |
| 30 | 0,0091507494882741600000 |
| 31 | -0,0030404103138459300000 |
| 32 | 0,0009786320697691590000 |
| 33 | -0,0003054518278370410000 |
| 34 | 0,0000925339360800447000 |
| 35 | -0,0000272314154749846000 |
| 36 | 0,0000077912105386761400 |
| 37 | -0,0000021689045553612000 |
| 38 | 0,0000005878872873742190 |
| 39 | -0,0000001552625399988320 |
| 40 | 0,0000000399801040496993 |
| 41 | -0,0000000100437822368757 |
| 42 | 0,0000000024631180247576 |
| 43 | -0,0000000005900026896512 |
| 44 | 0,0000000001381142659865 |
| 45 | -0,0000000000316128208814 |
| 46 | 0,0000000000070785229365 |
| 47 | -0,0000000000015512507712 |
| 48 | 0,0000000000003328725613 |
| 49 | -0,0000000000000699711711 |
| 50 | 0,0000000000000144140612 |
| 51 | -0,0000000000000029110751 |
| 52 | 0,0000000000000005766168 |
| 53 | -0,0000000000000001120595 |
| 54 | 0,0000000000000000213743 |
| 55 | -0,0000000000000000040028 |
| 56 | 0,0000000000000000007362 |
| 57 | -0,0000000000000000001330 |
| 58 | 0,0000000000000000000236 |
| 59 | -0,0000000000000000000041 |
| | -10,3000000112675000000000 |

Для продолжения нажмите любую клавишу . . .

Анализ таблицы свидетельствует о том, что для аргумента $x = -10.3$ абсолютная погрешность вычисления экспоненциальной функции по алгоритму (1.2.15) (с использованием $\varepsilon \sim 10^{-20}$) составила `0,0000000112675`, т.е. всего $\sim 10^{-8}$.

```
public static double MyExp(double x, double eps)
{
  if (x == 0) return 1.0;
  double exp = 1.0, pk = 1.0; string txt;
  for (int k = 1; Math.Abs(pk) > eps; k++)
  {
    pk = pk * (x / k); exp += pk;
    txt = string.Format("{0}", k).PadLeft(6) +
          string.Format("{0:F22}", pk).PadLeft(30);
    Console.WriteLine(txt);
  }
  return exp;
}
```

```
using System;
using MyF = MAC_DLL.MAC_My_Functions;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MAC_LabWork_1_2
{
  class Main_LW_1_2
  {
    static void Main(string[] args)
    {
      //Console.Write(TestMySinCos());
      //Console.Write(TestMySinhCosh());
      double d = Math.Log(MyF.MyExp(-10.3, 1.0e-20));
      Console.WriteLine(string.Format("{0:F22}", d).PadLeft(36));
    }
    static string TestMySinhCosh()...
    static string TestMySinCos()...
  }
}
```

Теперь нам предстоит проверить качество разработанных алгоритмов для тригонометрических функций с помощью известных аналитических формул и библиотечных математических функций языка программирования **C#**.

Например, известна трехпараметрическая формула для функции синус:

$$\begin{aligned}\sin(A+B+C) = \sin A\cdot\cos B\cdot\cos C + \cos A\cdot\sin B\cdot\cos C + \\ +\cos A\cdot\cos B\cdot\sin C - \sin A\cdot\sin B\cdot\sin C\end{aligned} \tag{1.2.18}$$

Определим абсолютную ошибку вычисления данного выражения для некоторых заданных значений величин $A$, $B$ и $C$, когда в вычислениях участвуют либо математические функции класса **Math** языка **C#**, либо методы, разработанные нами в ходе выполнения данной лабораторной работы, и входящие в состав класса **MAC_My_Functions** библиотеки **MAC_DLL**.

Работу выполним в рамках консольного приложения **MAC_LabWork_1_2**.

```
namespace MAC_LabWork_1_2
{
    class Main_LW_1_2
    {
        static double A, B, C, e;

        static void Main(string[] args)
        {
            //Console.Write(TestMySinCos());
            //Console.Write(TestMySinhCosh());
            //double d = Math.Log(MyF.MyExp(-10.3, 1.0e-20));
            //Console.WriteLine(string.Format("{0:F22}", d).PadLeft(36));

            A = 15.0; B = 17.0; C = -11.0; e = 1.0E-20;
            Console.WriteLine(Test_DLL());
            Console.WriteLine(Test_Math());
        }

        static string Test_DLL()
        {
            double d0 = MyF.MySin(A + B + C, e);
            double d1 = MyF.MySin(A, e) * MyF.MyCos(B, e) * MyF.MyCos(C, e);
            double d2 = MyF.MyCos(A, e) * MyF.MySin(B, e) * MyF.MyCos(C, e);
            double d3 = MyF.MyCos(A, e) * MyF.MyCos(B, e) * MyF.MySin(C, e);
            double d4 = MyF.MySin(A, e) * MyF.MySin(B, e) * MyF.MySin(C, e);
            double error = Math.Abs(d0 - (d1 + d2 + d3 - d4));
            return (string.Format("  MAC = {0:F16}   error = {1:E2}", d0, error));
        }
        static string Test_Math()
        {
            double d0 = Math.Sin(A + B + C);
            double d1 = Math.Sin(A) * Math.Cos(B) * Math.Cos(C);
            double d2 = Math.Cos(A) * Math.Sin(B) * Math.Cos(C);
            double d3 = Math.Cos(A) * Math.Cos(B) * Math.Sin(C);
            double d4 = Math.Sin(A) * Math.Sin(B) * Math.Sin(C);
            double error = Math.Abs(d0 - (d1 + d2 + d3 - d4));
            return (string.Format(" Math = {0:F16}   error = {1:E2}", d0, error));
        }

        static string TestMySinhCosh()...
        static string TestMySinCos()...
    }
}
```

С результатом:

```
C:\Windows\system32\cmd.exe
  MAC = 0,8366556407693710   error = 2,39E-009
 Math = 0,8366556385360560   error = 1,11E-016
Для продолжения нажмите любую клавишу . . .
```

Очевидно, что для больших значений аргументов тригонометрических функций «качество» наших алгоритмов уступает качеству алгоритмов фирмы **`Microsoft`**.

Но если уменьшить значения входящих параметров, например, в 10 раз, то результат будет существенно лучше:

```
using System;
using MyF = MAC_DLL.MAC_My_Functions;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MAC_LabWork_1_2
{
    class Main_LW_1_2
    {
        static double A, B, C, e;

        static void Main(string[] args)
        {
            //Console.Write(TestMySinCos());
            //Console.Write(TestMySinhCosh());
            //double d = Math.Log(MyF.MyExp(-10.3, 1.0e-20));
            //Console.WriteLine(string.Format("{0:F22}", d).PadLeft(36));

            //A = 15.0; B = 17.0; C = -11.0; e = 1.0E-20;
            A = 1.50; B = 1.70; C = -1.10; e = 1.0E-20;
            Console.WriteLine(Test_DLL());
            Console.WriteLine(Test_Math());
        }

        static string Test_DLL()...
        static string Test_Math()...

        static string TestMySinhCosh()...
        static string TestMySinCos()...
    }
}
```

```
C:\Windows\system32\cmd.exe
  MAC = 0,8632093666488740    error = 2,22E-016
 Math = 0,8632093666488740    error = 1,11E-016
Для продолжения нажмите любую клавишу . . .
```

На этом общая часть Лабораторной работы 1.2. завершается и Вам предстоит выполнить самостоятельный этап, который заключается в проведении подобных тестовых вычислений с другой известной тригонометрической формулой (можно использовать **`MAC_LabWork_1_2`**).

Результаты этих вычислений заносятся в соответствующий бланк индивидуального задания и сдаются на проверку. Приведенные выше вычисления можно считать «примером выполнения» по индивидуальному варианту Лабораторной работы 1.2.

## Лабораторная Работа 1.2, Вариант 0

Тема: ***«Алгоритмы вычисления значений степенных рядов (спецфункций)»***

Задание: *Протестировать разработанные алгоритмы для тригонометрических функций (класс* **MAC_My_Functions** *библиотеки* **MAC_DLL***) с использованием заданной тригонометрической формулы:*

$$\Phi_1(A,B) = \sin(A+B) \quad = \quad \sin A \cdot \cos B + \cos A \cdot \sin B = \Phi_2(A,B)$$

*Выполнить дублирующие вычисления значений* $\Phi_1(A,B)$ *и* $\Phi_2(A,B)$ *с использованием библиотечных тригонометрических функций (класс* **Math***) языка программирования* **C#**.

*Результаты вычислений заносятся в таблицу по одному символу (цифре) в каждую клетку. Числовые результаты записываются с* ***пятнадцатью*** *цифрами после десятичной точки.*

*Значения ошибки вычислений* $\Delta = \left|\Phi_1 - \Phi_2\right|$ *записываются с* ***двумя*** *значащими цифрами в форме числа с плавающей точкой (смотри образец таблицы результатов).*

Значения параметров для тестовой формулы

| *Тест* | *A* | *B* |
|---|---|---|
| 1 | 15.30 | 12.70 |
| 2 | 2.13 | -4.85 |

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Класс | | MAC_My_Functions | | | | | | | | | | Тест | | | | 1 | |
| ***Функция*** | ***Значение*** | | | | | | | | | | | | | | | | | |
| $\Phi_1(A,B) =$ | + | 0 | . | 2 | 7 | 0 | 9 | 1 | 7 | 3 | 4 | 8 | 0 | 2 | 0 | 4 | 1 | 2 |
| $\Phi_2(A,B) =$ | + | 0 | . | 2 | 7 | 0 | 9 | 0 | 5 | 7 | 8 | 8 | 3 | 4 | 0 | 7 | 8 | 8 |
| $\left\|\Phi_1 - \Phi_2\right\| =$ | 1 | . | 1 | 2 | e | - | 0 | 6 | | | | | | | | | | |