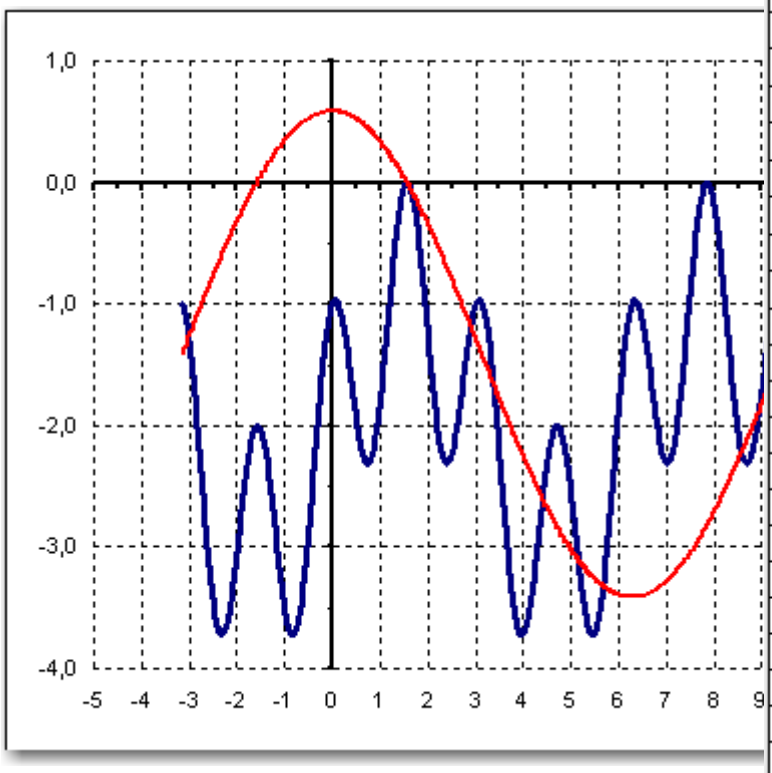
Лабораторная работа № 1.3

«Формирование наборов числовых данных в виде таблицы функции»

Анализ физических процессов и их математических моделей неразрывно связан с такими известными понятиями как функция, график функции, таблица функции.



x	y = f1(x)	y = f2(x)
-3,1416	-1,0000	-1,4142
-3,1227	-1,0217	-1,3953
-3,1039	-1,0491	-1,3765
-3,0850	-1,0821	-1,3576
-3,0661	-1,1206	-1,3388
-3,0473	-1,1646	-1,3199
-3,0284	-1,2138	-1,3011
-3,0095	-1,2680	-1,2822
-2,9906	-1,3272	-1,2634
-2,9718	-1,3910	-1,2446
-2,9529	-1,4591	-1,2258
-2,9340	-1,5313	-1,2070
-2,9152	-1,6073	-1,1883
-2,8963	-1,6868	-1,1695
-2,8774	-1,7693	-1,1508
-2,8586	-1,8545	-1,1321
-2,8397	-1,9420	-1,1135
-2,8208	-2,0315	-1,0948
-2,8020	-2,1225	-1,0762
-2,7831	-2,2145	-1,0576
-2,7642	-2,3072	-1,0391

Теоретические исследования, как правило, оперируют с функциями, для которых известно (или задается) аналитическое представление. Например, закон гармонического колебания материальной точки есть следующая функция времени: $x(t) = a \cdot \sin(\omega t + \alpha)$.

Другой пример – теоретическая зависимость вязкости μ моторного масла от его температуры θ выражается теоретической формулой (функцией) $\mu(\theta) = \mu_o \cdot \exp\left(-\delta_o \cdot \frac{\theta}{1+\theta}\right)$.

На основе этих аналитических зависимостей выполняется последующий математический анализ физических моделей, и делаются те или иные теоретические выводы и рекомендации.

Другая ситуация возникает, когда физическое явление (процесс) или его математическая модель изучаются посредством натурного (или компьютерного) эксперимента.

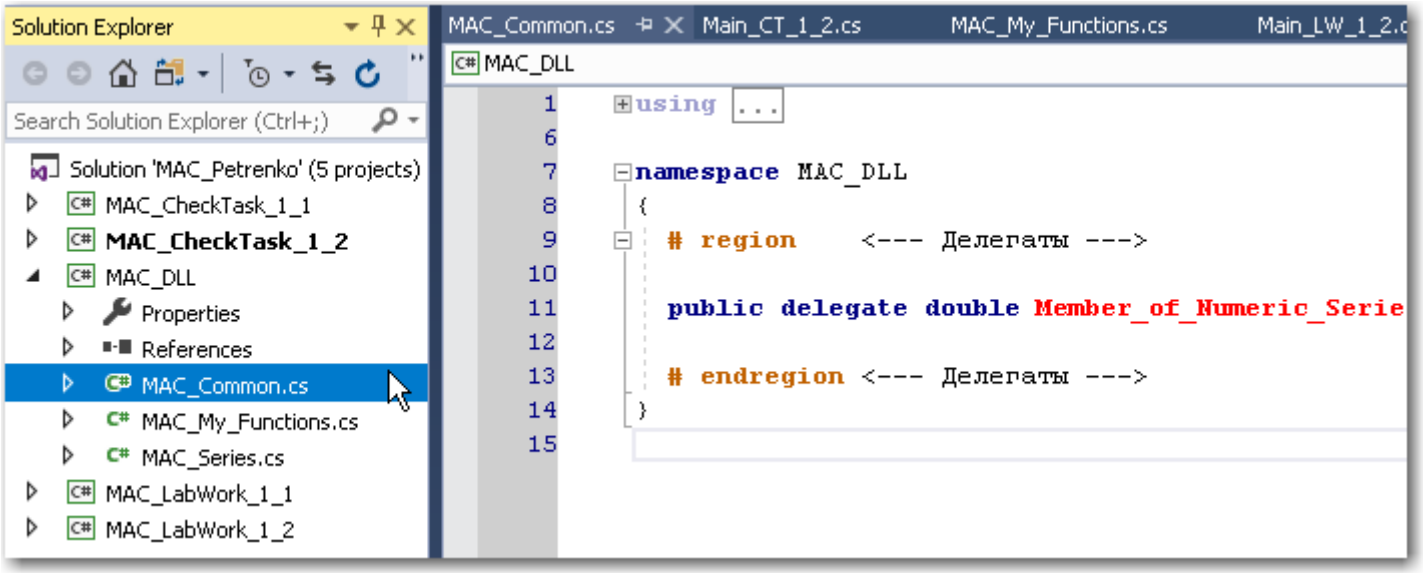
В этом случае в распоряжении исследователя имеются только числовые данные (результаты эксперимента), описывающие явление, и эти данные ему предстоит обработать для достижения конечных целей исследования. Как правило, такие числовые данные сохраняются в виде таблиц, которые дискретным образом определяют те или иные физические зависимости.

Методы приближенных вычислений должны нас обеспечить возможностью «обработки» функций в обеих указанных формах: в виде аналитической зависимости $f = f(x)$, и в форме дискретно заданной функции, записанной в виде упорядоченной таблицы данных.

Мы постараемся для этого использовать все преимущества **ООП**, которые нам предоставляет современная система разработки приложений **MS Visual Studio** и язык программирования **C#**.

Очевидно, что нам предстоит определенная подготовительная работа, непосредственно не связанная с численными методами, но обеспечивающая нам комфортное программирование и ясность в понимании основ и самой сути вычислительной математики.

В имеющемся рабочем пространстве **MAC_Petrenko** в проекте **MAC_DLL** библиотеки классов **(.dll)** уже сформирован файл с именем **MAC_Common**:



В этом файле мы и планировали размещать определения пользовательских типов, делегатов, интерфейсов и т.д., общего предназначения.

Класс `Point_xf`

Приступим к формированию первого пользовательского типа (класса).

Будем исходить из того, что основным исследуемым объектом численного анализа является заданная таблица функции $f = f(x)$, см. рисунок на предыдущей странице.

Под таблицей функции мы будем понимать упорядоченное конечное множество пар (x, f) действительных чисел, где x – аргумент, а f – соответствующее ему значение функции.

Каждая такая пара чисел «занимает» одну строчку в таблице функции.

Строки таблицы нумеруются от нуля и следуют друг за другом в соответствии с правилом возрастания значений аргумента x.

Таким образом, нулевая строка содержит наименьшее значение аргумента x, а последняя строка таблицы – наибольшее значение аргумента x.

Пару чисел (x, f), которую еще называют узлом таблицы, мы «объединим» в рамках класса `Point_xf`, который разместим в уже имеющемся файле **`MAC_Common`**:

```
1   using ...
6
7   namespace MAC_DLL
8   {
9       <--- Делегаты --->
14
15      #region     <--- Классы --->
16
17      public class Point_xf
18      {
19          public double x, f;
20          public Point_xf()
21          {
22              x = double.NaN; f = double.NaN;
23          }
24          public Point_xf(double x, double f)
25          {
26              this.x = x; this.f = f;
27          }
28          public string ToPrint()
29          {
30              return "  ("
31                     + string.Format("{0:F10},", x).PadLeft(16)
32                     + string.Format("{0:F10}", f).PadLeft(16)
33                     + "  )";
34          }
35      }
36
37      # endregion <--- Классы --->
38  }
```

Класс **Point_xf** имеет два конструктора и метод **ToPrint()**.

Метод **ToPrint()** предназначен для визуального представления числовых данных, выражающих в интуитивно понятной текстовой форме смысловое содержание экземпляра данного класса.

Для самой таблицы функции мы подготовим абстрактный класс **MyTable**, который затем будут наследовать другие классы, описывающие конкретные таблицы, в основе генерации которых будут либо аналитические зависимости $f = f(x)$, либо определенные наборы числовых данных, полученных из эксперимента.

Поскольку Вы только начинаете знакомство с дисциплиной «Методы приближенных вычислений», самостоятельно создать такой абстрактный класс будет весьма затруднительно.

Поэтому Вам предлагается готовое решение, которое следует перенести в свое рабочее пространство (проект **MAC_DLL**) и разобраться с предназначением и спецификациями членов этого класса.

По ходу разработки программных компонентов будут даваться соответствующие пояснения, а строки кода будут сопровождаться краткими комментариями.

Абстрактный класс `MyTable`.

Раздел свойств абстрактного класса **`MyTable`**:

```
 7   namespace MAC_DLL
 8   {
 9       <--- Делегаты --->
14
15       #region    <--- Классы --->
16
17       public class Point_xf...
36
37       public abstract class MyTable
38       {
39           #region    <--- Основные Свойства MyTable --->
40
41           // Основное свойство - Массив узлов (x,f) данной таблицы
42           protected Point_xf[] Points { get; set; }
43
44           // Свойство - возвращает количество узлов в данной таблице
45           public int Length
46           { get { if (Points == null) return 0; else return Points.Length; } }
47
48           // Свойство - возвращает ссылку на узел с наибольшим значением функции
49           public Point_xf Maximum { get; protected set; }
50
51           // Свойство - возвращает ссылку на узел с наименьшим значением функции
52           public Point_xf Minimum { get; protected set; }
53
54           // Свойство - возвращает длину области определения для данной функции
55           public double Region_x
56           { get { return Points[Length - 1].x - Points[0].x; } }
57
58           // Свойство - возвращает длину области значений для данной функции
59           public double Region_f
60           { get { return Maximum.f - Minimum.f; } }
61
62           // Свойство - определяет погрешность математических операций
63           public double Epsilon { get; set; }
64
65           // Свойство - определяет заголовок таблицы
66           public string Title { get; protected set; }
67
68           #endregion <--- Основные Свойства MyTable --->
69
```

Свойство **`Points[]`** собственно является носителем основной числовой информации – таблицы некоторой зависимости $f = f(x)$.

Очевидно, что все числовые данные, которые однозначно определяют математическую суть таблицы, после ее непосредственного формирования изменяться не должны.

Поэтому такие свойства нами отмечены как **`protected`**.

Мы сможем изменять эти данные только в методах класса-наследника.

Смысл представленных выше свойств можно понять из комментариев в коде.

Раздел методов абстрактного класса **`MyTable`**:

```
70
71      #region    <--- Основные Методы MyTable --->
72
73      // Метод - возвращает значение x для строки таблицы с номером index
74      public double X(int index)
75      {
76        if ((index >= 0) && (index < Length)) return Points[index].x;
77        else return double.NaN;
78      }
79
80      // Метод - возвращает значение f для строки таблицы с номером index
81      public double F(int index)
82      {
83        if ((index >= 0) && (index < Length)) return Points[index].f;
84        else return double.NaN;
85      }
86
87      // Метод, формирующий таблицу функции в текстовой форме
88      public virtual string Table_of_Function()
89      {
90        string txt = "\r\n  Таблица функции " + Title + " : \r\n";
91
92        for (int i = 0; i < Length; i++)
93        {
94          txt += string.Format("{0}", i).PadLeft(4)
95               + Points[i].ToPrint() + "\r\n";
96        }
97
98        txt += "\r\n  x = ["
99             + string.Format("{0:F12}", Points[0].x).PadLeft(17) + " :"
100            + string.Format("{0:F12}", Points[Length - 1].x).PadLeft(17)
101            + " ] \r\n";
102
103       txt += "  x_Reg = "
104            + string.Format("{0:F12}", Region_x).PadLeft(18) + "\r\n";
105
106       txt += "\r\n Min  ("
107            + string.Format("{0:F12},", Minimum.x).PadLeft(18)
108            + string.Format("{0:F12}", Minimum.f).PadLeft(18) + "  )";
109
110       txt += "\r\n Max  ("
111            + string.Format("{0:F12},", Maximum.x).PadLeft(18)
112            + string.Format("{0:F12}", Maximum.f).PadLeft(18) + "  )";
113
114       txt += "\r\n  f_Reg = "
115            + string.Format("{0:F12}", Region_f).PadLeft(18) + "\r\n";
116
117       return txt ;
118     }
```

```
119
120         // Метод - возвращает два массива со значениями аргумента x и функции f
121         public void ToArrays(out double[] x, out double[] f)
122         {
123             x = new double[Length]; f = new double[Length];
124             for (int i = 0; i < Length; i++)
125             { x[i] = Points[i].x; f[i] = Points[i].f; }
126         }
127
128         // Дополнительный переопределяемый метод, предназначенный для операций
129         //  вывода в текстовой форме различной информации по данной таблице
130
131         public virtual string ToPrint(string comment)
132         {
133             return comment;
134         }
135
136         #endregion <--- Основные Методы MyTable --->
137
138     }
139     # endregion <--- Классы --->
140 }
```

Теперь у нас имеется основа абстрактного класса **`MyTable`**, на основе которой мы можем проектировать классы-наследники, и которые будут учитывать особенности формирования таблиц функций.

По мере расширения круга решаемых задач вычислительной математики мы будем дополнять новыми свойствами и методами, как сам абстрактный класс **`MyTable`**, так и классы его наследников.

Класс `MyTableOfData`

Итак, пусть результаты некоторого численного эксперимента, реализованного в виде самостоятельного **`Windows`**–приложения, сохраняются в виде именованного файла неупорядоченных числовых данных (на жестком диске компьютера).

Для хранения больших объемов числовых данных, как правило, используются текстовые форматные файлы либо файлы бинарного типа.

Первые удобны тем, что позволяют исследователю быстро просматривать результаты и делать предварительные выводы на основе их непосредственного визуального анализа.

Однако такие файлы не удобны для дальнейшего использования в рамках других приложений, поскольку требуют дополнительных усилий по организации считывания текстовой информации с последующим ее конвертированием в наборы данных числового типа, над которыми следует выполнять определенные математические действия.

Бинарные файлы хранят в себе информацию во «внутреннем» представлении, определяемом архитектурой данного процессора.

Поэтому без каких-либо дополнительных конвертаций данные из этих файлов можно загружать непосредственно в оперативную память компьютера (для их последующей обработки другими приложениями).

Кроме того, при одном и том же объеме полезной математической информации бинарные файлы имеют размеры в несколько раз меньшие, чем файлы текстовые.

Поскольку, как уже отмечалось выше, файл результатов формируется отдельным самостоятельным **`Windows`**–приложением, в разрабатываемой нами программе должен быть предусмотрены способы «открывать» файлы обоих указанных типов.

Введем следующее правило идентификации файлов результатов численного эксперимента, которое по мере необходимости можно будет легко модифицировать.

Текстовый файл данных должен иметь расширение «**`.txt`**».

При этом пара числовых данных (x, f) должна разделяться между собой в строке символами `'пробел'` или `';'`.

Такое расширение позволяет открывать и просматривать указанный файл любой стандартной **`Windows`**–программой, например, «**`Блокнот`**» или **`MS Word`**.

Файл бинарного типа должен иметь расширение «**.bin**».

На данном этапе разработки будем требовать, чтобы одна запись в файле результатов содержала в качестве информации два числа (тип **double**): значение независимой переменной и соответствующее ему значение искомой функции.

Например, это может быть момент времени t и координата материальной точки x, или же x – координата и y – координата точки, движущейся на плоскости Oxy.

Разрабатываемый алгоритм должен выполнять следующие действия:

- открыть требуемый файл (который предварительно размещен в директории приложения);
- выделить память под список значений «пара вещественных чисел» (x, f) – для независимой переменной x и исследуемой функции $f = f(x)$;
- прочитать все данные из файла в указанный список;
- упорядочить полученный список по возрастанию аргумента x;
- определить дополнительные характеристики упорядоченной таблицы;
- сохранить упорядоченную таблицу данных в виде текстовой информации, удобной для визуального анализа.

Подведем итог вышесказанному.

Числовые данные, представляющие собой искомую таблицу функции, размещены в некотором файле (переменная **path**) и должны быть нами приведены к объекту определенного типа, что наследует основные элементы класса **MyTable**.

Алгоритм непосредственного преобразования информации из файла в объект «таблица данных» будет нами реализован в конструкторе нового класса-наследника, который мы назовем **MyTableOfData**.

Этот алгоритм включает в себя несколько этапов.

Первый этап. Чтение будущей таблицы из файла во «временный список» ее узлов.

После достижения конца соответствующего файла мы будем знать общее количество узлов таблицы и сможем выделить память под массив узлов **Points[]**.

Второй этап. Нет никакой гарантии, что во «временном списке» узлы упорядочены по возрастанию аргумента. Поэтому следует выполнить соответствующую сортировку элементов «временного списка», после чего преобразовать его в требуемый массив узлов **Points[]**.

После того, как таблица узлов упорядочена, следует определить те ее узлы, которые содержат наибольшее и наименьшее значения функции на всем интервале изменения аргумента.

Кроме того, очень важно нам иметь представление о числовой информации, записанной непосредственно в файле данных.

Для этого мы добавим в класс **`MyTableOfData`** свойство **`Table_in_File`**.

Общая структура класса **`MyTableOfData`**:

```
1    using System;
2    using System.Collections.Generic;
3    using System.IO;
4    // using System.Linq; ...
7
8    namespace MAC_DLL
9    {
10     public class Point_xf...
29
30     public abstract class MyTable...
130
131
132    public class MyTableOfData : MyTable
133    {
134      // Свойство - Таблица данных непосредственно в файле
135      public string Table_in_File { get; }
136
137      // <--- Конструктор экземпляра --->
138      public MyTableOfData(string path, string title)
139      {
140        int i = -1;
141        List<Point_xf> Temp = new List<Point_xf>();
142        FileInfo file = new FileInfo(path);
143        Table_in_File = "\r\n Таблица " + title +
144                        " в файле " + file.Name + " :\r\n";
145
146        <--- Чтение бинарного файла в список данных --->
166
167        <--- Чтение форматного файла в список данных --->
195
196        <--- Формирование Таблицы Данных --->
215      }
216
217      #region    <--- Переопределение методов класса MyTable --->
218
219      public override string ToPrint(string Comment)
220      {
221        return Comment + "\r\n" + Table_in_File + Table_of_Function();
222      }
223
224      #endregion <--- Переопределение методов класса MyTable --->
225    }
226  }
```

Алгоритмы этапов конструктора следует «разобрать» самостоятельно:

```
146     #region    <--- Чтение бинарного файла в список данных --->
147
148     if (file.Extension == ".bin")
149     {
150       BinaryReader bin_rdr = new BinaryReader(file.OpenRead());
151       try
152       {
153         while (true)
154         {
155           Temp.Add(new Point_xf(bin_rdr.ReadDouble(), bin_rdr.ReadDouble()));
156           i++;
157           Table_in_File += string.Format("{0}", i).PadLeft(4)
158                          + Temp[i].ToPrint() + "\r\n";
159         }
160       }
161       catch (IOException) { }
162       bin_rdr.Close();
163     }
164
165     #endregion <--- Чтение бинарного файла в список данных --->
```

```
167     #region    <--- Чтение форматного файла в список данных --->
168
169     if (file.Extension == ".txt")
170     {
171       bool key_point;
172       try { double x = Convert.ToDouble("0.1"); key_point = true; }
173       catch (FormatException) { key_point = false; };
174       StreamReader txt_rdr = new StreamReader(file.OpenRead());
175
176       // Читаем записи из текстового файла и редактируем их
177       string[] txt; string line;
178       while (!txt_rdr.EndOfStream)
179       {
180         i++; line = txt_rdr.ReadLine();
181         if (key_point)
182           line = (line.Replace(",", ".")).Trim();
183         else
184           line = (line.Replace(".", ",")).Trim();
185         txt = line.Split(new char[] { ' ', ';' },
186                     StringSplitOptions.RemoveEmptyEntries);
187         Temp.Add(new Point_xf(Convert.ToDouble(txt[0]),
188                               Convert.ToDouble(txt[1])));
189         Table_in_File += string.Format("{0}", i).PadLeft(4)
190                                  + Temp[i].ToPrint() + "\r\n";
191       }
192       txt_rdr.Close();
193     }
194     #endregion <--- Чтение форматного файла в список данных --->
```

```
196         #region    <--- Формирование Таблицы Данных --->
197
198         if (Temp != null)
199         {
200           Temp.Sort((point_i, point_j) => point_i.x.CompareTo(point_j.x));
201           Points = Temp.ToArray();
202           Minimum = new Point_xf(double.NaN, double.MaxValue);
203           Maximum = new Point_xf(double.NaN, double.MinValue);
204           for (i = 0; i < Length; i++)
205           {
206             if (Minimum.f > Points[i].f) { Minimum = Points[i]; }
207             if (Maximum.f < Points[i].f) { Maximum = Points[i]; }
208           }
209         }
210         else { Points = null; Minimum = null; Maximum = null; }
211
212         Title = title;
213
214         #endregion <--- Формирование Таблицы Данных --->
215       }
216
```

После того, как Вы закончили формирование всех перечисленных выше программных компонентов, следует выполнить компиляцию проекта **`MAC_DLL`** и убедиться в отсутствии ошибок (исправить ошибки).

Тестирование классов `MyTable` и `MyTableOfData`.

Теперь переходим к проектированию исполняемого консольного приложения, в котором будут использоваться спроектированные нами программные компоненты.

В имеющееся рабочее пространство **`MAC_Petrenko`** добавляется новый проект консольного приложения **`MAC_LabWork_1_3`**.

Отладку его кода будем производить с использованием ***тестовых*** файлов данных **`MAC_LW_1_3_v00.txt`** и **`MAC_LW_1_3_v00.bin`**.

Эти файлы должны быть Вами предварительно скопированы в папку **`bin\Debug\`** проекта консольного приложения **`MAC_LabWork_1_3`**.

Результаты тестовых вычислений приводятся ниже.

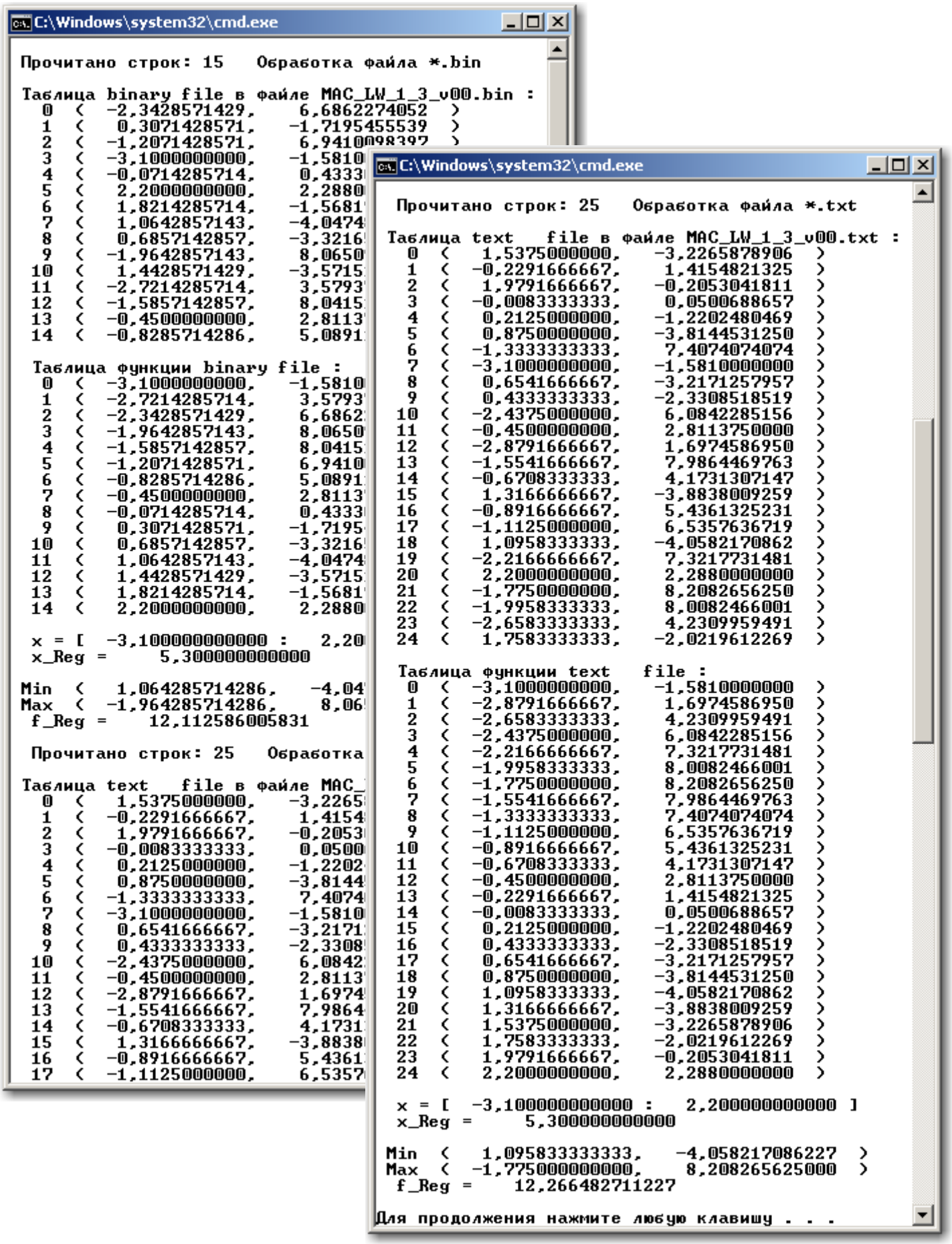
Код основной программы **`MAC_LabWork_1_3`** должен выглядеть так:

```
1   using System;
2   using MyTD = MAC_DLL.MyTableOfData;
3   // using System.Collections.Generic; ...
7
8   namespace MAC_LabWork_1_3
9   {
10    class Main_LW_1_3
11    {
12      static void Main(string[] args)
13      {
14        MyTD T1 = new MyTD("MAC_LW_1_3_v00.bin", "binary file");
15        string txt = "\r\n";
16        txt += string.Format(" Прочитано строк: {0}", T1.Length);
17        txt += T1.ToPrint("   Обработка файла *.bin");
18        Console.WriteLine(txt);
19
20        MyTD T2 = new MyTD("MAC_LW_1_3_v00.txt", "text   file");
21        txt = "\r\n";
22        txt = string.Format("  Прочитано строк: {0}", T2.Length);
23        txt += T2.ToPrint("   Обработка файла *.txt");
24        Console.WriteLine(txt);
25      }
26    }
27  }
```

Обратите внимание на то что, что бинарный и текстовый файлы содержат числовую информацию, соответствующую одной и той же математической функции $f(x)$.

Интервал изменения аргумента $x \in [-3.1\,,+2.2]$ у «Таблиц данных в файле» также совпадает, но количество строк в этих таблицах различное, а сами узлы в них расположены в произвольном порядке.

Результаты:


C:\Windows\system32\cmd.exe
Прочитано строк: 15 Обработка файла *.bin
Таблица binary file в файле MAC_LW_1_3_v00.bin :
0 (-2,3428571429, 6,6862274052)
1 (0,3071428571, -1,7195455539)
2 (-1,2071428571, 6,9410098392)
3 (-3,1000000000, -1,5810
4 (-0,0714285714, 0,4333
5 (2,2000000000, 2,2880
6 (1,8214285714, -1,5681
7 (1,0642857143, -4,0474
8 (0,6857142857, -3,3216
9 (-1,9642857143, 8,0650
10 (1,4428571429, -3,5715
11 (-2,7214285714, 3,5793
12 (-1,5857142857, 8,0415
13 (-0,4500000000, 2,8113
14 (-0,8285714286, 5,0891
Таблица функции binary file :
0 (-3,1000000000, -1,5810
1 (-2,7214285714, 3,5793
2 (-2,3428571429, 6,6862
3 (-1,9642857143, 8,0650
4 (-1,5857142857, 8,0415
5 (-1,2071428571, 6,9410
6 (-0,8285714286, 5,0891
7 (-0,4500000000, 2,8113
8 (-0,0714285714, 0,4333
9 (0,3071428571, -1,7195
10 (0,6857142857, -3,3216
11 (1,0642857143, -4,0474
12 (1,4428571429, -3,5715
13 (1,8214285714, -1,5681
14 (2,2000000000, 2,2880
x = [-3,1000000000000 : 2,20
x_Reg = 5,3000000000000
Min (1,064285714286, -4,04
Max (-1,964285714286, 8,06
f_Reg = 12,112586005831
Прочитано строк: 25 Обработка
Таблица text file в файле MAC_
0 (1,5375000000, -3,2265
1 (-0,2291666667, 1,4154
2 (1,9791666667, -0,2053
3 (-0,0083333333, 0,0500
4 (0,2125000000, -1,2202
5 (0,8750000000, -3,8144
6 (-1,3333333333, 7,4074
7 (-3,1000000000, -1,5810
8 (0,6541666667, -3,2171
9 (0,4333333333, -2,3308
10 (-2,4375000000, 6,0842
11 (-0,4500000000, 2,8113
12 (-2,8791666667, 1,6974
13 (-1,5541666667, 7,9864
14 (-0,6708333333, 4,1731
15 (1,3166666667, -3,8838
16 (-0,8916666667, 5,4361
17 (-1,1125000000, 6,5357
C:\Windows\system32\cmd.exe
Прочитано строк: 25 Обработка файла *.txt
Таблица text file в файле MAC_LW_1_3_v00.txt :
0 (1,5375000000, -3,2265878906)
1 (-0,2291666667, 1,4154821325)
2 (1,9791666667, -0,2053041811)
3 (-0,0083333333, 0,0500688657)
4 (0,2125000000, -1,2202480469)
5 (0,8750000000, -3,8144531250)
6 (-1,3333333333, 7,4074074074)
7 (-3,1000000000, -1,5810000000)
8 (0,6541666667, -3,2171257957)
9 (0,4333333333, -2,3308518519)
10 (-2,4375000000, 6,0842285156)
11 (-0,4500000000, 2,8113750000)
12 (-2,8791666667, 1,6974586950)
13 (-1,5541666667, 7,9864469763)
14 (-0,6708333333, 4,1731307147)
15 (1,3166666667, -3,8838009259)
16 (-0,8916666667, 5,4361325231)
17 (-1,1125000000, 6,5357636719)
18 (1,0958333333, -4,0582170862)
19 (-2,2166666667, 7,3217731481)
20 (2,2000000000, 2,2880000000)
21 (-1,7750000000, 8,2082656250)
22 (-1,9958333333, 8,0082466001)
23 (-2,6583333333, 4,2309959491)
24 (1,7583333333, -2,0219612269)
Таблица функции text file :
0 (-3,1000000000, -1,5810000000)
1 (-2,8791666667, 1,6974586950)
2 (-2,6583333333, 4,2309959491)
3 (-2,4375000000, 6,0842285156)
4 (-2,2166666667, 7,3217731481)
5 (-1,9958333333, 8,0082466001)
6 (-1,7750000000, 8,2082656250)
7 (-1,5541666667, 7,9864469763)
8 (-1,3333333333, 7,4074074074)
9 (-1,1125000000, 6,5357636719)
10 (-0,8916666667, 5,4361325231)
11 (-0,6708333333, 4,1731307147)
12 (-0,4500000000, 2,8113750000)
13 (-0,2291666667, 1,4154821325)
14 (-0,0083333333, 0,0500688657)
15 (0,2125000000, -1,2202480469)
16 (0,4333333333, -2,3308518519)
17 (0,6541666667, -3,2171257957)
18 (0,8750000000, -3,8144531250)
19 (1,0958333333, -4,0582170862)
20 (1,3166666667, -3,8838009259)
21 (1,5375000000, -3,2265878906)
22 (1,7583333333, -2,0219612269)
23 (1,9791666667, -0,2053041811)
24 (2,2000000000, 2,2880000000)
x = [-3,1000000000000 : 2,2000000000000]
x_Reg = 5,3000000000000
Min (1,095833333333, -4,058217086227)
Max (-1,775000000000, 8,208265625000)
f_Reg = 12,266482711227
Для продолжения нажмите любую клавишу . . .

После успешного выполнения данной лабораторной работы с использованием тестовых файлов Вам следует выполнить заключительный этап (с индивидуальными наборами данных в соответствии с Вашим Вариантом).

Контрольный просчет следует выполнять с файлами данных **`MAC_LW_1_3_vNN.txt`** и **`MAC_LW_1_3_vNN.bin`**, где **`NN`** – номер Вашего персонального варианта.

Эти файлы данных следует скопировать в директорию **`bin\Debug\`**, в которой расположен соответствующий исполняемый файл **`MAC_Lab_Work_1_3.exe`**.

Ниже приводится «образец» заполнения таблиц результатов на бланке индивидуального задания. Эти таблицы соответствуют тестовым результатам, полученным и представленным на предыдущей странице.

1. Таблица результатов на базе файла **`MAC_LW_1_3_v00.bin`**:

всего строк в таблице	1	5	

узел «**Minimum**»	$x_{\min}$ =	+	1	.	0	6	4	2	8	5	7	1	4	3
	$f_{\min}$ =	-	4	.	0	4	7	4	8	9	4	3	1	5
узел «**Maximum**»	$x_{\max}$ =	-	1	.	9	6	4	2	8	5	7	1	4	3
	$f_{\max}$ =	+	8	.	0	6	5	0	9	6	5	7	4	3

2. Таблица результатов на базе файла **`MAC_LW_1_3_v00.txt`**:

всего строк в таблице	2	5	

узел «**Minimum**»	$x_{\min}$ =	+	1	.	9	5	8	3	3	3	3	3	3	3
	$f_{\min}$ =	-	4	.	0	5	8	2	1	7	0	8	6	2
узел «**Maximum**»	$x_{\max}$ =	-	1	.	7	7	5	0	0	0	0	0	0	0
	$f_{\max}$ =	+	8	.	2	0	8	2	6	5	6	2	5	0
верхний узел (x_0, f_0) таблицы функции	x_0 =	-	3	.	1	0	0	0	0	0	0	0	0	0
	f_0 =	-	1	.	5	8	1	0	0	0	0	0	0	0
нижний узел (x_n, f_n) таблицы функции	x_n =	+	2	.	2	0	0	0	0	0	0	0	0	0
	f_n =	+	2	.	2	8	8	0	0	0	0	0	0	0