E10-2004-136

Cs. Török*

VISUALIZATION AND DATA ANALYSIS IN THE MS .NET FRAMEWORK

*Technical University of Košice, Vysokoškolská 4, 04002, Košice, Slovakia, torokcs@tuke.sk

Торок Ч. E10-2004-136
Визуализация и анализ данных в платформе MS .NET

Платформа .NET является интегрирующей Windows-компонентой для построения и выполнения следующего поколения программных приложений. Она предлагает новую вычислительную модель, позволяющую быстро создавать стандартные приложения и соединять их. В работе дано описание новой .NET-компоненты, называемой `LinAlg`, которая упрощает визуализацию и анализ цифровых данных в среде .NET. Основной принцип `LinAlg` — это векторизация, позволяющая записывать векторно-матричные выражения и визуализировать данные в объектно-ориентированном стиле. В основу `LinAlg` заложены векторные и матричные классы. Описаны разнообразные пути создания и модификации этих классов, и показано, как их методы поддерживают получение информации о данных, а также их статистический, численный и графический анализ.

Работа выполнена в Лаборатории информационных технологий ОИЯИ.

Сообщение Объединенного института ядерных исследований. Дубна, 2004

Török Cs. E10-2004-136
Visualization and Data Analysis in the MS .NET Framework

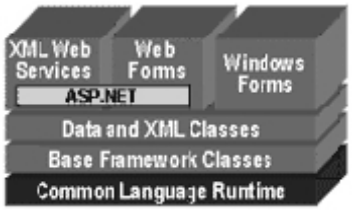
The .NET Framework is an integral Windows component for building and running the next generation of software applications. It provides a new computing model that enables a standard and rapid way of building applications and their connection. The paper presents a .NET component named `LinAlg` that makes visualization, signal and data analysis in .NET simpler. The main concept in `LinAlg` is vectorization that enables one to write vector/matrix expressions and visualize data in an object-oriented manner. At the center of `LinAlg` lie vector and matrix classes. The paper describes diverse ways of their creating and modification and illustrates how their methods support gaining information about data, their statistical, numerical, and graphical analysis.

The investigation has been performed at the Laboratory of Information Technologies, JINR.

Communication of the Joint Institute for Nuclear Research. Dubna, 2004

INTRODUCTION

The main benefit of systems SAS, Matlab or R is the support of visualization and vectorization (vectorial/matrix programming) with a rich set of functions and toolboxes. Everyone who got used to the services that are provided by these and similar systems and intends or is forced to write code in classic or new languages desires to use analogous services. The new component-oriented programming language MS Visual C# within the MS .NET Framework provides techniques and services that support the work with arrays and graphics, however, not at the level as the above-mentioned systems.



MS .NET Framework

The *.NET Framework* [1] is an integral Windows component for building and running the next generation of software applications and Web services. It is composed of the *Common Language Runtime* (CLR) and a set of class libraries. CLR is Microsoft's commercial implementation of the Common Language Infrastructure (CLI) specification (CLI is a framework for designing, developing, deploying, and executing distributed components and applications based on the Common Type System). CLR is responsible for run-time services such as language integration, security enforcement, and memory, process, and thread management. In addition, the CLR has a role in lifecycle management, strong type naming, cross-language exception handling, and dynamic binding.

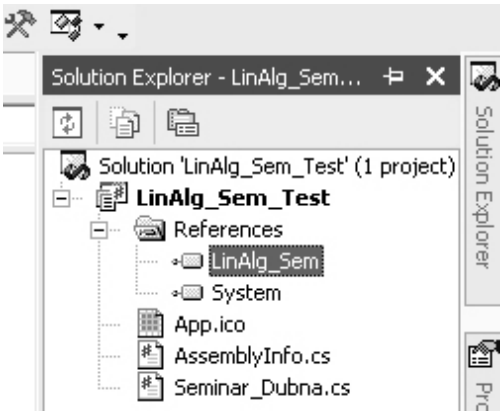
The paper [2] showed the process of vectorization in C# and introduced a simple vector class that later served as a basic building block for developing more advanced data models and classes [3]. This paper presents the `LinAlg` component library developed at the author's department. `LinAlg` is a set of types and classes that enable vectorial programming and incorporates a wide range of numerical, statistical, and graphical methods. Data analysis and visualization in the .NET framework gets more comfortable due to `LinAlg`.

Section 1 serves as a brief and fast introduction to how to use `LinAlg`. Section 2 presents the library as a whole and some different ways of getting information on its classes and members. The following section describes the vector and matrix classes, how to create and manipulate them. Section 4 introduces

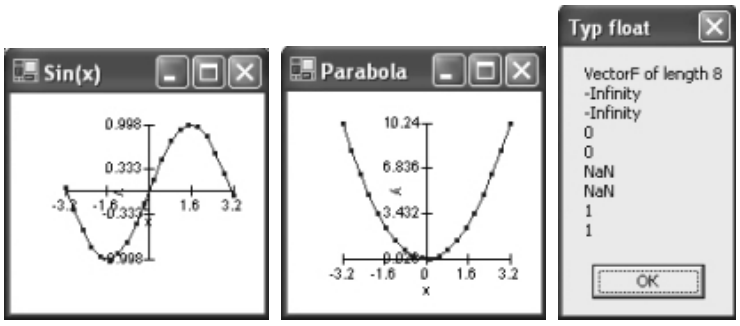
the common mathematical functions and operators for vector and matrix objects. Then we show the main visualization techniques included in `LinAlg`. Section 6 is devoted to data analysis techniques and presents varied statistical and numerical methods.

1. FAST INTRODUCTION TO LinAlg

This section serves as a fast introduction to how create, plot, and view vectors and matrices using the component `LinAlg`. You can find more thorough examples in the upcoming sections. The first example is a full program with every necessary code (referencing namespaces, a class declaration, and a `Main` static member). It creates four vectors (one without a variable) by three different ways, plots and shows them. The `z` vector illustrates the float calculus (the MS Intermediate Language's equivalent of `float` is `Single`). Notice that we must reference both the `LinAlg` dll component *LinAlg_Sem* in the solution explorer (on the right) and the `LinAlg` namespace in the code (on the left):




```
using System;
using System.Windows.Forms;
using LinAlg;

public class Seminar_Dubna
{
      static void Main()
      {
            sec_02_FastIntro();
      }
      static void sec_02_FastIntro()
      {
            VectorF x, y, z;
            x = VectorF.CreateFromTo(20, -3.2f, 3.2f);
            y = MathLA.Sin(x);
            y.PlotInWF(x, false, "Sin(x)");
            MathLA.Pow(x, 2).PlotInWF(x, true, "Parabola");
            float a = Single.PositiveInfinity;
            float b = Single.MaxValue, c = Single.MinValue;
            z = new VectorF(-a/0, -b/0, 1/a, 1/b, 0f/0f, a/a, b/b, c/c);
            z.MsgBox(3, "Typ float");
      }
}
```

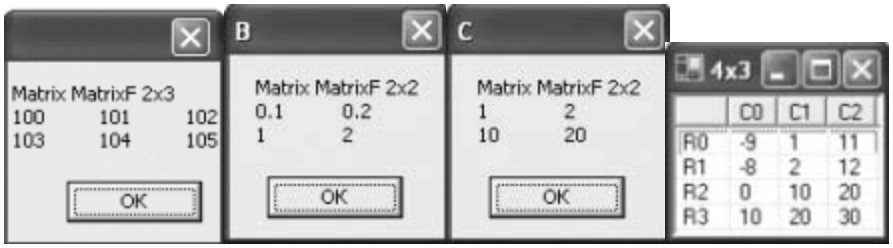


Setting the boolean `asApplicationRun` argument of `PlotInWF` to `true` runs the window in modal mode. If you want to display several windows at once, set `asApplicationRun` to `true` only for the last window (the same holds for the `ShowInWF` method).

Mention must be made that 1/0 would occur in a compilation *Division by constant zero* error.

The creation and viewing of matrices are illustrated by the codelines:

```
MatrixF A, B, C;
A = MatrixF.CreateFromBy1(2, 3, 100f); A.MsgBox();
B = MatrixF.CreateFromVector(2, 2, 0.1f, 0.2f, 1f, 2f); B.MsgBox("B");
VectorF x = 10*VectorF.CreateFromMatrix(B, true); // true => by rows
C = MatrixF.CreateFromVector(2, 2, x); C.MsgBox("C");
D = MatrixF.ConcatAfter(x-10, x, x+10); D.ShowInWF(true);
```



The `MsgBox` method serves for viewing small vectors and matrices. For viewing large ones use the `ShowInWF` method, that is equipped with scrollbars. At last we mention here without giving any example that the most frequently used properties `Length`, `RowsNumbers`, and `ColumnNumbers` return the number of elements in a vector and the number of the rows and columns of a matrix, respectively.

2. GENERAL INFORMATION ON LinAlg

`LinAlg` consists of nineteen classes and two enums:

Class FormPlot	class MatrixD	class Time
class GraphicsLA	class MatrixF	class VectorB
class GridView	class MatrixS	class VectorD
class InfoTypes	class Numerics	class VectorF
class LinRegression	class Point3D	class VectorS
class MathLA	class Point3DF	enum PointType
class MatrixB	class TimeCount	enum PlotType

We give here four different ways of getting information on classes and their public members (variables, methods, functions, and properties):

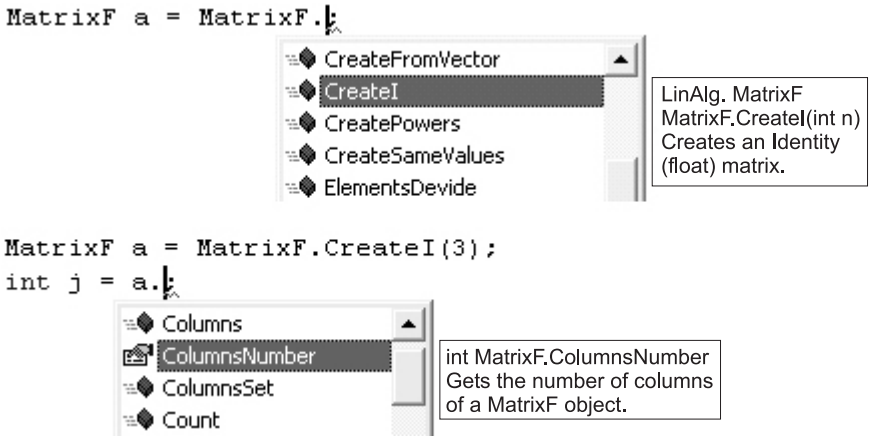
1) IntelliSense of Visual studio;

2) the class `InfoTypes`;

3) the object browser of Visual studio;

4) the XML-based NDoc application.

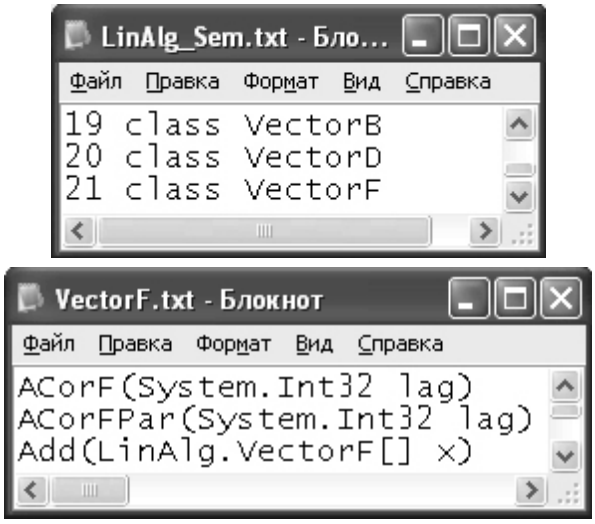
1) In MS Visual Studio the built-in IntelliSense help appears when writing a dot after a name of a class or its object. You must realize that a class can contain two different types of members: static and object members. The first ones are related to the class itself, while the second ones, to its objects. The following picture shows IntelliSense in work: `CreateI` is a static method of the `MatrixF` class, whereas `ColumnsNumber` is an object property (IntelliSense every time shows the appropriate members and does not mix them):



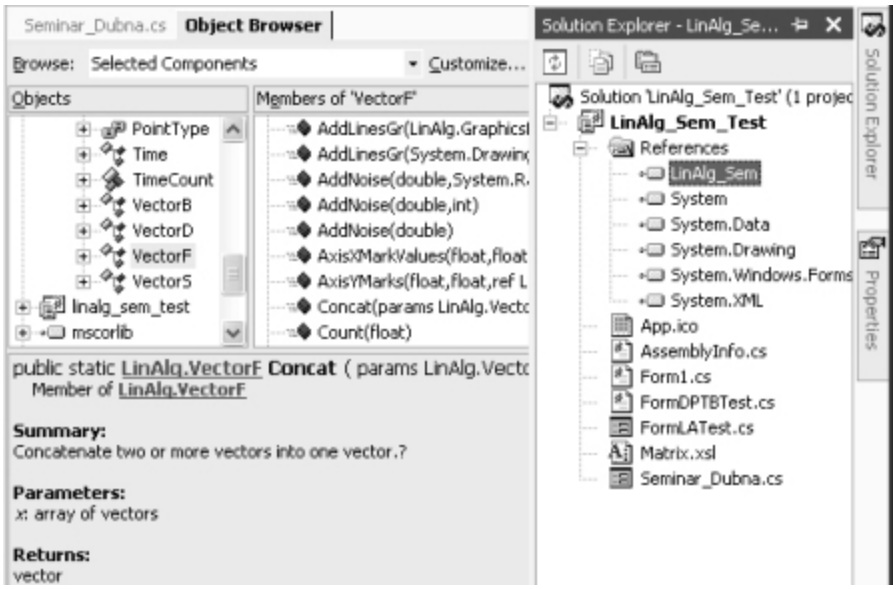
2) Executing the codelines

```
string ass = "LinAlg_Sem.dll";
string dirTo = "Info_LinAlg";
InfoTypes.FileWrite(ass, dirTo);
```

you get a list of the classes, enumerations, and interfaces contained in `LinAlg` (InfoLinalg\LinAlg_Sem.txt) and a thorough information on their members in the corresponding text files:



3) Selecting *View – Object Browser* (Ctrl+Alt+J) and clicking *linalg_sem – LinAlg* you can easy get the signatures of the class members with some other pieces of information:



4) The free XML-based NDoc application enables one to view also examples provided through attributes.

3. CREATING AND MANIPULATING VECTORS AND MATRICES

`LinAlg` contains four vector classes and four matrix ones

```
VectorF, VectorD, VectorS, VectorB
MatrixF, MatrixD, MatrixS, MatrixB
```

that hold `float`, `double`, `string`, and `bool` elements, respectively (F, D, S, B). Both the `float` and `double` classes can be used for numerical computations. However, plot can be created only for float vectors, whereas inverse matrix can be computed just for double matrix. It is not a strong constraint, for, by casting float, vectors/matrices can be converted to double vectors/matrices and vice versa.

This section shows different ways of creating and manipulating vectors and matrices.

New vectors/matrices can be created in `LinAlg` in five ways:

1) using the `new` keyword;

2) using one of the `Create*` functions;

3) reading data from a file due to `FileRead` function;

4) combining several vectors/matrices by `Concat*` functions;

5) casting vectors/matrices of another type.

Notice that some other functions and properties return also vectors/matrices, such as `MovingAverage`, `SortDown`, `MeanOfRows`, etc.

1) Instantiating objects with the `new` keyword is the standard way of creating class objects based on constructors. The vector and matrix classes contain a varied number of constructors:

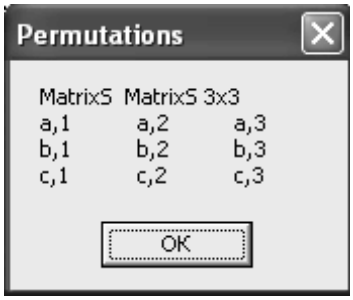
```
VectorS s = new VectorS("JINR", "LIT", "Dubna");
s.MsgBox(); // "JINR", "LIT", "Dubna"

VectorB B = new VectorB(3);
B.MsgBox(); // false, false, false

MatrixF A = new MatrixF(2,3);
A.MsgBox(); // 2x3 matrix with 0 elements


ArrayList aL = new ArrayList();
aL.Add(1.0/4); aL.Add(1/2.0);
VectorD a = new VectorD(aL);
a.MsgBox(); // 0.25, 05
```

Only `VectorD` can be instantiated based on an `ArrayList`. Unlike vectors, matrices cannot be instantiated from a list of values. For this purpose use the static `CreateFromVector`.

A vector of zero length `VectorD d = new VectorD(0)`; is a valid object in `LinAlg` and it can be sometimes useful. Remember that `d` does not have any element and `d.Length` returns 0.

2) The number of `VectorF` constructors in the earlier versions of `LinAlg` was more than ten; however, it led to code that was difficult to read, mainly in the case of several arguments. In later versions some constructors were replaced with static `Create*` functions. From the viewpoint of readability it was a proper decision.

```
static void sec_04_Vectors_Create()
{
   VectorD d = VectorD.CreateFromTo(5, 1, 3);
   d.MsgBox(); // 1, 1.5, 2, 2.5, 3

   VectorD d2 = VectorD.CreatePowers(2, -1, 4); // of 2 from to
   d2.MsgBox(); // 0.5, 0, 1, 2, 4, 8, 16

   VectorF f = VectorF.CreateRndNormal(2, 0, 1, 1397); // mean,var,kernel
   f.MsgBox(); //-1.07070744, -1.65021265

   float[,] a = {{1,2,3}, {4,5,6}};
   VectorF g = VectorF.CreateFromMatrix(a, false); // by columns
   g.MsgBox(); // 1, 4, 2, 5, 3, 6

   MatrixF F = MatrixF.CreateFromByH(2, 3, 1, 0.1f);
   F.MsgBox(); // MatrixF 2x2: 1, 1.1, 1.2 ; 1.3, 1.4, 1.5

   VectorS u = VectorS.CreateSameValues(3, "Hi");
   u.MsgBox(); // Hi, Hi, Hi

   VectorB b = new VectorB(true, true, false, false);
   MatrixB B = MatrixB.CreateFromVector(2, 2, b);
   B.MsgBox(); // 2x2: true, true ; false, false

   VectorS x = new VectorS("a", "b", "c");
   VectorS y = new VectorS("1", "2", "3");
   MatrixS S = MatrixS.CreatePermutation(x,y);
   S.MsgBox("Permutations"); //see the picture

   MatrixD D = MatrixD.CreateRndUniform(2, 2, 1, 10, 1307);
   D.MsgBox(); // 5, 3 ; 4, 5
}
```



Mention must be made that, unlike `CreateRndNormal` and `CreateRndUniform`, `AddNoise` adds pseudorandom numbers to an existing vector based on a signal-to-noise ratio (SNR), see Sec. 6.

3) Data processing is unimaginable without reading/writing measurements from/to files:

```
static void sec_04_Vectors_IO()
{
    VectorF f = VectorF.CreateFromBy1(2, 5);
    f.FileOverWrite("ja.txt");
    f.FileAppend("ja.txt");
    VectorF.FileRead("ja.txt").MsgBox(); // 5,6,5,6

    MatrixF F = MatrixF.CreateI(2);
    F.FileOverWrite("ha.txt",'\t'); // tab delimiter
    MatrixF.FileRead("ha.txt",'\t').MsgBox(); //
    MatrixF 2x2: 1,0 ;  0,1
}
```

As you can see it is necessary to supply matrices with a column delimiter `char`. Notice also that `FileRead` is always a static method, whereas `FileWrite` is an object one. A `MatrixS` object can be also read using a dialog window due to the static method `FileDialogRead` (see the casting example below).

4) Use the `Concat*` functions if you want to create a vector or matrix from several ones:

```
static void sec_04_Vectors_Concat()
{
    VectorF x = VectorF.CreateFromBy1(3, 1);
    VectorF y = VectorF.Concat(x, x);
    y.MsgBox(); // 1,2,3,1,2,3

    MatrixF a = MatrixF.ConcatBellow(x,x);
    MatrixF b = MatrixF.ConcatAfter(a, new MatrixF(2,2));
    b.MsgBox(); // MatrixF 2x5: 1,2,3,0,0 ; 1,2,3,0,0
}
```

5) Casting is an important concept in C#. A double array or a float vector can be implicitly casted to a double vector; however, in accordance with the MS recommendations a double vector should be casted to a float vector explicitly:

```
static void sec_04_Vectors_Cast()
{
    double[] d1 = {1, 2};
    VectorF f = new VectorF(3);
    VectorD d2 = d1; // implicit casting
    d2 = f; // implicit casting
    f = (VectorF)d2; // explicit casting
    MatrixD D = (MatrixD)MatrixS.FileDialogRead(';');
    D.MsgBox();
}
```

If a dialog is cancelled, a 0×0 matrix is returned that does not contain any element, and `D.RowsNumber` and `D.ColumnsNumber` return 0. The `CastToMatrixS` method of the `MatrixF` class rounds the matrix elements to the given decimal places and casts the matrix to `MatrixS`.

At the end of this section we demonstrate the use of some vector and matrix manipulation methods, such as `ElementsSet`, `RowsSet (ColumnsSet)`, `Reverse`, and `SortUp`:

```
static void sec_04_Vectors_Manipulate()
{
    VectorF x = new VectorF(1f, 2f, 3f, 7f, 7f, 6f);
    VectorF y = new VectorF(60f, 50f, 40f);

    x.ElementsSet(3, y);
    x.MsgBox(); // 1,2,3,60,50,40

    x.Reverse.MsgBox(); // 40,50,60,3,2,1
    x.SortUp.MsgBox(); // 1,2,3,40,50,60

    MatrixF a = MatrixF.CreateFromBy1(2, 3, 1);
    a.RowsSet(0, y);
    a.MsgBox(); // 60,50,40 ; 4,5,6
    a.ElementsSet(0,1, new MatrixF(2,2));
    a.MsgBox(); // 60,0,0 ; 4,0,0
}
```

The `Length` of the new vector should remain the same, so you cannot do this:

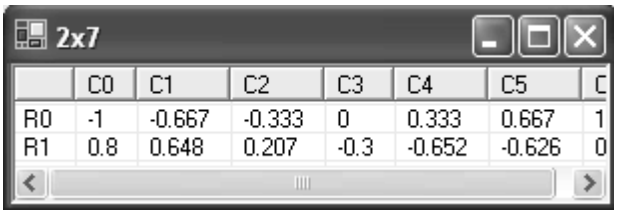
```
x.ElementsSet(5, y);
```

4. COMPUTATION WITH VECTORS AND MATRICES

In the .NET Framework the `System.Math` class provides some common mathematical functions and constants. These functions are static and take value type arguments. Their counterparts in `LinAlg` with vector and matrix arguments are contained in the class `MathLA`. In addition to the well known basic mathematical functions, .NET Framework also provides arithmetic and comparison operators +, –, /, ==, !=. Their vector and matrix counterparts are embedded to the `LinAlg`'s appropriate vector and matrix classes through operator overloading (notice that we defined for string vectors and matrices only the + operator).

The next example illustrates the use of such `MathLA` functions as `PolynomialByRoots`, `Round`, the operator *, the `Transposed` property, and the `ElementsMultiply` method of the class `MatrixD`. `MathLA` in addition to the `PolynomialByRoots` function also has a `PolynomialByCoefs` function (they take the coefficients and the real roots of the polynomial as the second argument). Since the standard `Math` class does not have such functions, `MathLA` also contains overloads of these two functions for a scalar first argument:

```
static void sec_05_Computation()
{
    VectorD x, r, y;
    MatrixD a;
    x = VectorD.CreateFromTo(7, -1, 1);
    r = new VectorD(-1.5, -0.2, 1);
    y = MathLA.PolynomialByRoots(x, r);
    a = MatrixD.ConcatAfter(x,y); //7x2
    MathLA.Round(a.Transposed, 3).ShowInWF();
```




```
    MatrixD d, e, f1, f2;
    d = MatrixD.CreateFromBy1(2,2,-1);
    e = MatrixD.CreateFromBy1(2,2,1);
    f1 = d*e;
    f2 = MatrixD.ElementsMultiply(d,e);
    f1.MsgBox("f1"); f2.MsgBox("f2");
    (f1 == f2).MsgBox();
}
```

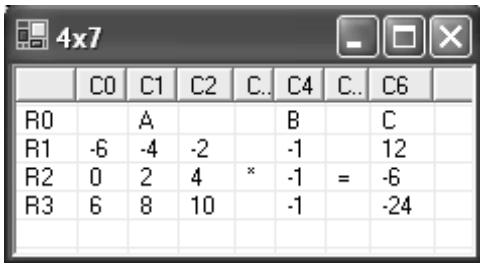
Notice that the comparison operators == and != return for vector/matrix operands a boolean vector/matrix.

We emphasize that in `LinAlg` the * operator always decreases the dimension and so for vectors it corresponds to the dot product `( z = x*y; )`. Consequently, to get a vector of the elements' square, use the `MathLA.Pow` function (see Sec. 1) or the `ElementsMultiply` function for vectors.

It is a proper place to make an important remark relating to shallow and deep copy. A simple reference type assignment `y = x;` always results in a shallow copy: changing `y` you also change `x`. However, `y = x + 0;` or `y = x*1;` presents a deep copy (changes in `y` do not effect `x`).

The second example demonstrates the * operator and the use of a string matrix for an Excel-like data viewing:

```
Static void sec_05_ExcelLike_DataViewing()
{
    // A) Compute:
    int n = 3; // 15
    MatrixF a = MatrixF.CreateFromByH(n, n, -2*n, 2f);
    MatrixF b = MatrixF.CreateSameValues(n, 1, -n/2);
    MatrixF c = a*b;
    MatrixS d = new MatrixS();
```



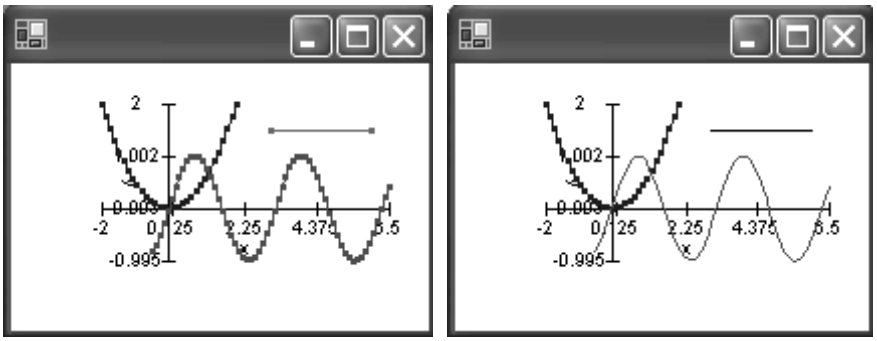

```
    // B) Show:
    string s = "";
    d.Add("A", 0, n/2, s);
    d.Add( a , 1, 0, s); d.Add("*", n/2+1, n , s);
    d.Add("B", 0, n+1, s);
    d.Add( b , 1, n+1, s); d.Add("=", n/2+1, n+2, s);
    d.Add("C", 0, n+3, s);
    d.Add( c , 1, n+3, s);
    d.ShowInWF(true);
}
```

5. DATA VISUALIZATION

Section 1 showed how easy it is to plot one vector in a separate window using `PlotInWF`. This section begins with showing how to handle such easy plotting of several vectors. However, plotting in a separate window does not enable one to adjust some settings, such as the font height, decimal places of the axes values and the place of markers along the axes, and neither permits it to draw verticals and horizontals in a simple way. This section presents the methods that make the process of creating enhanced plots simpler based on the `Paint` event handler. We underscore that plotting with `PlotInWF` and the `Paint` event handler present two diverse approaches: you should never use `PlotInWF` in an event handler.

To plot several vectors in a separate window, create an array of `VectorF` objects and use the static `PlotInWF` method:

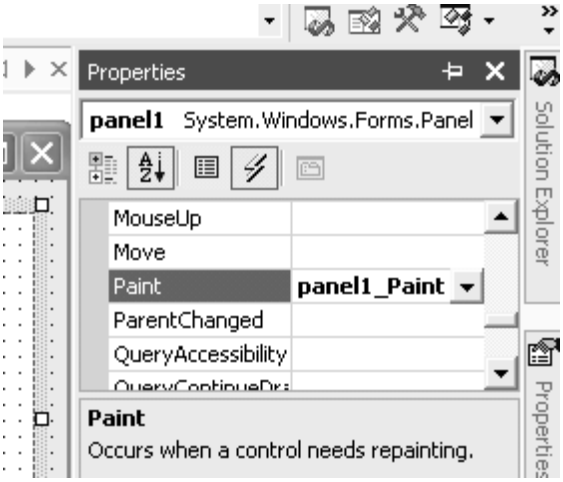
```
static void plot3VectorsInWF()
{
    VectorF x1 = VectorF.CreateFromTo(31, -2f, 2f);
    VectorF x2 = VectorF.CreateFromTo( 61, -.5f, 6.5f);
    VectorF x3 = new VectorF(3f, 6f);
    VectorF y1 = 0.5f*MathLA.Pow(x1, 2);
    VectorF y2 = MathLA.Sin(2*x2);
    VectorF y3 = new VectorF(1.5f, 1.5f);

    VectorF[] xV = new VectorF[]{x1, x2, x3};
    VectorF[] yV = new VectorF[]{y1, y2, y3};

    VectorF.PlotInWF(yV, xV, false);
}
```



To change the type of plot and its color, use the `PlotType` enumeration and the appropriate overload method of `PlotInWF` (see the right picture):

```
PlotType[] Pt = {PlotType.Both, PlotType.Line, PlotType.Line};
Color[] pC = {Color.Blue, Color.Red, Color.Black};
VectorF.PlotInWF(yV, xV, pT, pC, true);
```

If you want to place the points over the line, first draw the line and then the points.



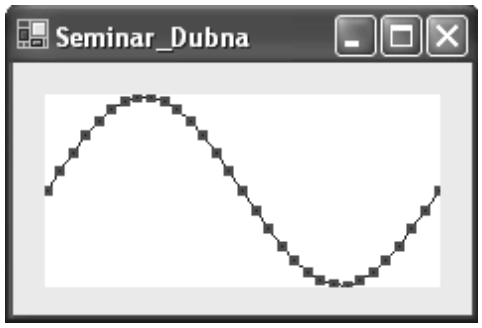
You can create plots not only in a separate window, but in the native window of your application too. GDI+ enables one to draw on the graph surface of the most controls due to their `OnPaint` event handler. The subsequent examples will be drawn on a panel. Add a panel control to the Windows Form in a design mode and set its `BackColor` to White in the Properties window (F4). Bounding the panel's Anchor to its every four edges will result in increasing and decreasing of the panel when resizing the application window. In design mode click the panel if it is not selected, in the properties window select the Event pane (see the yellow thunder) and double-click the Paint event. The system generates:

— a codeline that associates the Paint event with the `panel1_Paint` event handler by creating a `PaintEventHandler` delegate instance;

— an empty `panel1_Paint` method and opens the code window.

To draw something in GDI+ you, need a `Graphics` object (associate it with a surface or canvas). There are several ways of creating such `Graphics` objects (see [4]). We shall use the panel's `CreateGraphics` method. `LinAlg` also needs a `GraphicsLA` object to set (here implicitly) the World area (the real mathematical coordinates) and the Plot area (the pixel coordinates on the screen):

```
void panel1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{
    Graphics g1 = panel1.CreateGraphics();
    g1.Clear(panel1.BackColor);
    GraphicsLA g2 = new GraphicsLA(g1);
    plot1Vector(g2);
    g1.Dispose();
}
```



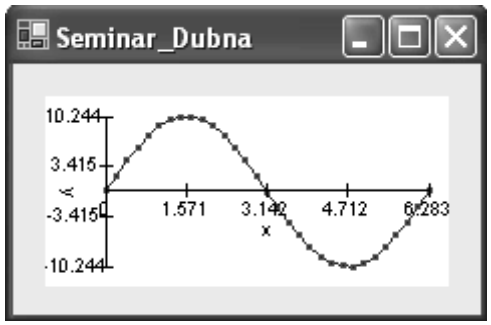

```
static void plot1Vector(GraphicsLA g)
{
    float b = 2*(float)Math.PI;
    VectorF x = VectorF.CreateFromTo(31,0f,b);s
    VectorF y = 10.3f*MathLA.Sin(x);

    y.PlotLines(g);
    y.PlotPoints(g);
}
```

In order the Windows resizing worked properly when the Windows is getting smaller, select in the design mode either the form or the panel, in the Events pane double-click the Resize event, and to the created and viewed `*_Resize` event handler add the codeline

`Refresh()`;

To make space for the axes, you can use the `PlotAreaIni` method. Its four parameters set the distance of the plot (plot area) from the right, left, bottom, and top edges of the panel (notice that `PlotAreaIni` has several overloads)

```
static void plot1VectorWithAxes(GraphicsLA g)
{
    VectorF x = VectorF.CreateFromTo(31, 0f, 2*(float)Math.PI);
    VectorF y = 10.3f*MathLA.Sin(x);
    g.PlotAreaIni(30f, 10f, 10f, 10f);
    y.PlotLines(g, x);
    y.PlotPoints(g, new Pen(Color.Red), 2, PointType.Rectangle, x);
    g.PlotAxisX(y, x);
    g.PlotAxisY(y, x); //=> see the picture
}
```



To run `plot1VectorWithAxes`, call it from the panel's paint event handler `panel1_Paint` after commenting the codeline `plot1Vector(g2)`;

Unlike the `PlotLines` and `PlotPoints` methods, `PlotAxisX` and `PlotAxisY` do not set implicitly the world area coordinates, so if you want to plot the vectors over the axes, change the order of commands and add before the axes drawing commands a codeline with the `WorkAreaIni` method:

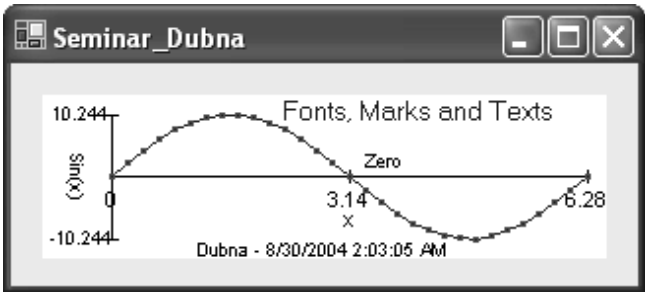
```
g.WorldAreaIni(x.Minimum, x.Maximum, y.Minimum, y.Maximum);
g.PlotAxisX(y, x);
g.PlotLines(y, x);
...
```

The four input arguments of the WorkAreaInimethod set the world area coordinates: the minimal and maximal values over the `x` and `y` axes.

One of the overloaded methods of `PlotAxis*` can be used to alter the axes' default name. To change the default settings of the axes marks (the number of marks, their font heights and decimal places), use the `*MarksNB`, `*MarksDecimalPlaces`, and the `*FontHeight` methods, and to draw title, subtitle, and any text on the plot utilize the `PlotTitle` method. The overloaded versions of these methods enable one to create many effects on the plots. The next example demonstrates the use of these methods:

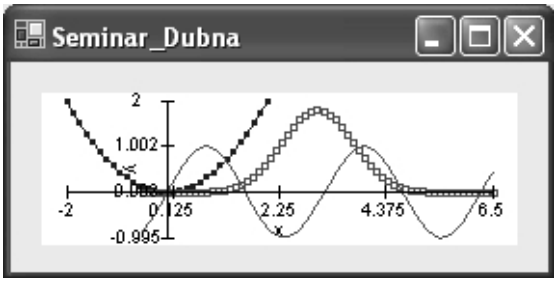
```
static void plotMarksFontsTitles(GraphicsLA g)
{
    VectorF x = VectorF.CreateFromTo(31, 0f, 2*(float)Math.PI);
    VectorF y = 10.3f*MathLA.Sin(x);

    g.PlotAreaIni(35f, 10f, 10f, 10f);
    g.WorldAreaIni(x.Minimum, x.Maximum, y.Minimum, y.Maximum);
```

```
    g.XMarksNB = 3; g.YMarksNB = 2;
    g.XMarksDecimalPlaces = 2; g.YMarksDecimalPlaces = 3;
    g.XFontHeight = 8; g.YFontHeight = 7;
    g.PlotAxisX(y, x);
    g.PlotAxisY(y, x, "Sin(x)");

    string title = "Fonts, Marks and Texts";
    Font fnt = new Font(FontFamily.GenericSansSerif,10);
    g.PlotTitle(title, fnt, Color.Blue, StringAlignment.Center,50,0);
    g.PlotTitle("Zero", g.PlotAreaGet[0]-5, g.PlotAreaGet[3]/2 - 8);
    string subTitle = "Dubna - "+DateTime.Now.ToString();
    g.PlotTitle(subTitle, 0, g.PlotAreaGet[3]);

    y.PlotLines(g, x);
    y.PlotPoints(g, new Pen(Color.Red), 2, PointType.Rectangle, x);
}
```



`WordAreaIni` is generally needed when plotting several charts at the same time. The following example plots three charts. To set the world area coordinates, we leveraged the `MiniMaxi` static function of the `VectorF` class:

```
static void plot3Vectors(GraphicsLA g)
{
    VectorF x1 = VectorF.CreateFromTo(31, -2f, 2f);
    VectorF x2 = VectorF.CreateFromTo( 61, -.5f, 6.5f);
    VectorF y1 = 0.5f*MathLA.Pow(x1, 2);
    VectorF y2 = MathLA.Sin(2*x2);
    VectorF y3 = 1.8f*MathLA.Exp(-MathLA.Pow(x2-3, 2));
    VectorF mimax = VectorF.MiniMaxi(x1, x2, x2);
    VectorF mimay = VectorF.MiniMaxi(y1, y2, y3);

    g.WorldAreaIni(mimax[0], mimax[1], mimay[0], mimay[1]);
    g.PlotAreaIni(5);

    y1.PlotLines(g, x1);
    y1.PlotPoints(g, new Pen(Color.Blue), 2, PointType.Rectangle, x1);
    y2.PlotLines(g, new Pen(Color.Red, 1), x2);
    y3.PlotPoints(g, new Pen(Color.Red, 1), 3, PointType.Rectangle, x2);

    VectorF xAx = new VectorF(mimax[0], mimax[1]);
    VectorF yAx = new VectorF(mimay[0], mimay[1]);

    g.PlotAxisX(yAx, xAx);
    g.PlotAxisY(yAx, xAx);
}
```

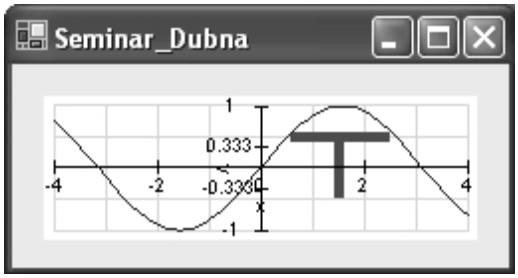


If you want to enhance any point on the chart, create two additional float vectors of length one that will hold the `x` and `y` coordinates of the given point, and based on them use the `PlotPoints` method.

To better locate the plot values, it is common to draw vertical and horizontal lines. `GraphicsLA` has four methods for this purpose: `PlotVerticals`, `PlotHorizontals`, `PlotVerticalLine`, and `PlotHorizontalLine`.

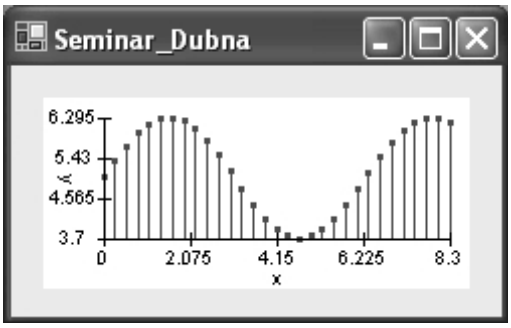
The following codelines illustrate their use:

```
static void plot_VerticalsHorizontals(GraphicsLA g)
{
    VectorF x;
    x = VectorF.CreateFromTo(100,-4f, 4f);
    VectorF y = MathLA.Sin(x);

    VectorF mimaX = VectorF.MiniMaxi(x);
    VectorF mimaY = VectorF.MiniMaxi(y);
    g.WorldAreaIni(mimaX[0], mimaX[1], mimaY[0], mimaY[1]);
    g.PlotAreaIni(5f, 5f, 5f, 5f);

    VectorF v = new VectorF(-4, -3, -2, -1, 1, 2, 3, 4);
    g.PlotVerticals(new Pen(Color.LightGray), mimaY, v);

    VectorF h = new VectorF(-1, -0.5f, .5f, 1);
    g.PlotHorizontals(new Pen(Color.LightGray), mimaX, h);

    g.PlotVerticalLine(new Pen(Color.Red,5), -.5f, .5f, 1.5f);
    g.PlotHorizontalLine(new Pen(Color.Red,5), .6f, 2.5f, .5f);

    g.PlotAxisX(y, x);
    g.PlotAxisY(y, x);

    y.PlotLines(g, x);
}
```



LinAlg in addition to the main plot methods PlotLines and PlotPoints has some special plot methods as well. This section's last example leverages one of them. PlotStems should be called before any plot command, since it sets the WorldAreaIni according to its second argument refValue:

```
static void plot_Stems(GraphicsLA g)
{
    VectorF x = VectorF.CreateFromTo(31,0f, 8.3f);
    VectorF y = 1.3f*MathLA.Sin(x)+5;

    g.PlotAreaIni(30f, 10f, 25f, 10f);
    float refVal = y.Minimum;
    y.PlotStems(g, refVal, new Pen(Color.Red), 2, PointType.Rectangle, x);

    g.PlotAxisX(y, x);
    g.PlotAxisY(y, x);
}
```



6. DATA ANALYSIS

LinAlg provides a wide range of statistical and numerical functions, properties, and methods that support the data analysis process. You can leverage for both (value type) vectors and matrices the properties:

Length, RowsNumbers, ColumnNumbers, Maximum, Minimum, Mean, Median, StandDev, Var, Sum, CumSum, Norm:

Matrices also support

MaximumOfColumns, MaximumOfRows, MeanOfColumns, MeanOfRows Inverse, Determinant, Transposed

The vector function IndexOf returns the index of a value, and CountOf returns the count of a given value either in a vector or matrix.

You can find the results of the next example in the comments:

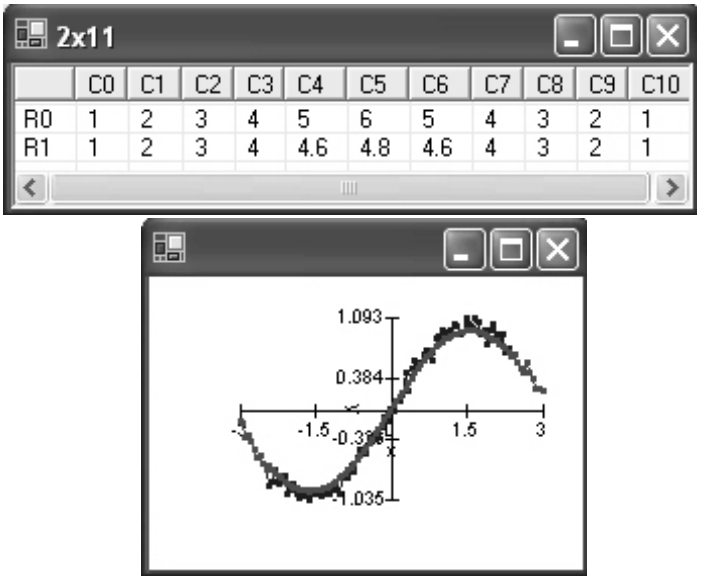
```
static void sec_07_Mean()
{
    VectorF x = new VectorF(1, 2, 3, 4, 20);
    VectorF z = new VectorF(x.Mean, x.Median);
    z.MsgBox(); // 6, 3

    MatrixD A = MatrixD.CreateFromBy1(2,3,1); // 2x3: 1,2,3 ; 4,5,6
    MatrixD B = A.MeanOfRows;
    B.MsgBox(); // 2x1: 2, 5

    A[0,0] = 6;
    MessageBox.Show(A.CountOf(6).ToString()); // 2
}
```

The vector classes offer further functions, such as

`MovingAverage, Differences, GetEveryKth`
`LinRegression,`
`ACorFPar, ACorF, P_A_CF, P_A_CF_Errors`

We illustrate each of the functions with an example.

`MovingAverage` returns the moving average of the order 2 * befAft + 1, where befAft is the function argument. The return vector has the same length as the input one, because the function does not delete the elements at the beginning and at the end of the vector:




```
static void sec_07_MovingAverage()
{
   VectorF a = VectorF.CreateFromBy1(6,1);
   VectorF b = VectorF.CreateFromByH(5,5,-1);
   VectorF c = VectorF.Concat(a,b);
   VectorF d;
```

```
   int befAft = 2;
   d = c.MovingAverage(befAft);
   MatrixF.ConcatAfter(c,d).Transposed.ShowInWF();

   VectorF x = VectorF.CreateFromTo(101,-3,3);
   VectorF y = MathLA.Sin(x);
   y = y.AddNoise(10);
   VectorF yEs = y.MovingAverage(10);
   VectorF[] u = new VectorF[]{x,x};
   VectorF[] v = new VectorF[]{y,yEs};
   VectorF.PlotInWF(v, u, true);
}
```

To make the checking easy, notice that (3+4+5+6+6)/5 = 4.6.

The signature of `Differences` is

```
public VectorF[] Differences(int order, int lag)
```

and it gets the differences x[i] - x[i-lag] of the given order:

```
   VectorF a;
   a = new VectorF(1, 2, 4, 7, 11, 16, 22); // maxorder 6
   int order = 3;
   VectorF[] ad = a.Differences(order, 1);
   ad[0].ShowInWF(); // 1,2,3,4,5,6
   ad[1].ShowInWF(); // 1,1,1,1,1
   ad[2].ShowInWF(); // 0,0,0,0
```

The function `GetEveryKth`, who's signature is

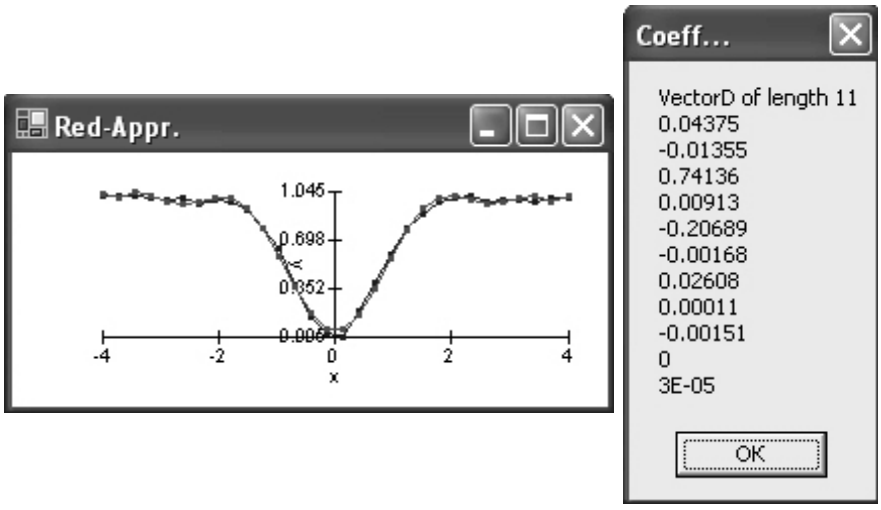
```
public VectorF GetEveryKth(int k,
int fromThisElement)
```

returns the every k-th ($\geqslant$1) element of a given vector from `fromThisElement` ($\geqslant$1):

```
   VectorF a = VectorF.CreateFromBy1(32,1);
   a = a.GetEveryKth(4, 20);
   a.ShowInWF(); // 20,24,28,32
```

The `LinRegression` class serves for performing linear and polynomial regression. In the following example a nonpolynomial function is fitted by a polynomial of degree ten:

```
static void sec_06_Regression ()
{
   VectorD x = VectorD.CreateFromTo(30, -4f, 4f);
   VectorD y;
   y = 1 - MathLA.Exp(-MathLA.Pow(x, 2));
   y = y.AddNoise(20, 5107701); // SNR=20, kernel=5107701

   LinRegression c = y.Regression(x, 12);
   c.Coefficients.MsgBox(5,"Coeff..."); //rounded to 5 decimals

   VectorF yf = (VectorF)y;
   VectorF yEst = (VectorF)c.YEstimation;
   VectorF[] yV = new VectorF[]{yf, yEst};
   VectorF.PlotInWF(yV, (VectorF)x, true, "Red-Appr.");
}
```

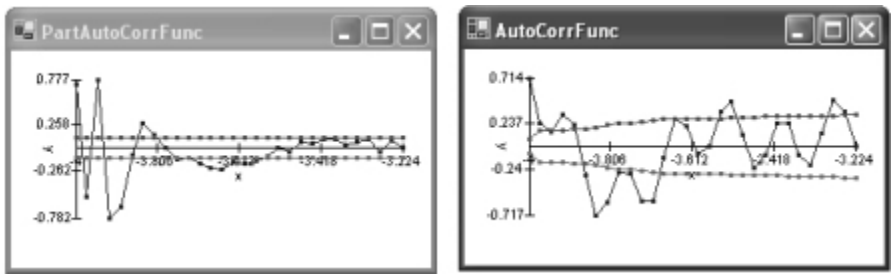


Notice that since the function is even the odd regression coefficient are zero. To view the estimations and residuals, write:

```
c.YEstimation.MsgBox("YEst...");
c.YResiduals.MsgBox("YRes...");
```

`LinAlg` provides four methods for computing the correlation functions: `ACorFPar`, `ACorF`, `P_A_CF` and `P_A_CF_Errors`. The first and the second columns in the matrices returned by the last two methods refer to the partial auto correlation and the auto correlation, respectively.

```
static void plot_P_A_CF()
{
 VectorF x = VectorF.CreateFromTo(300, -4f, 4f);

 VectorF y;
 y = (VectorF)(1*MathLA.Sin(10*x)+1*MathLA.Sin(15*x)+1*MathLA.Sin(50*x));
 y = y.AddNoise(20);

 int lag = x.Length/10;
 MatrixF a = y.P_A_CF(lag);

 int what;
 what = 0;
 VectorF yEr = y.P_A_CF_Errors( a.Columns(1) ).Columns(what);
 VectorF[] yV = new VectorF[]{a.Columns(what), yEr, -yEr};
 VectorF.PlotInWF(yV, x.ElementsGet(0,lag-1), false, "PartAutoCorrFunc");

 what = 1;
 yEr = y.P_A_CF_Errors(a.Columns(1)).Columns(what);
 yV = new VectorF[]{a.Columns(what), yEr, -yEr};

 VectorF.PlotInWF(yV, x.ElementsGet(0,lag-1), true, "AutoCorrFunc");
}
```



To draw the error lines with the same color, you could write:

```
PlotType[] pt = new PlotType[3]{PlotType.Both, PlotType.Line,
PlotType.Line};
Color[] pc = new Color[3]{Color.Blue, Color.Magenta, Color.Magenta};
VectorF[] xV = new VectorF[]{x.ElementsGet(0,lag-1),
x.ElementsGet(0,lag-1),
x.ElementsGet(0,lag-1)};
VectorF.PlotInWF(yV, xV, pt, pc, true);
```

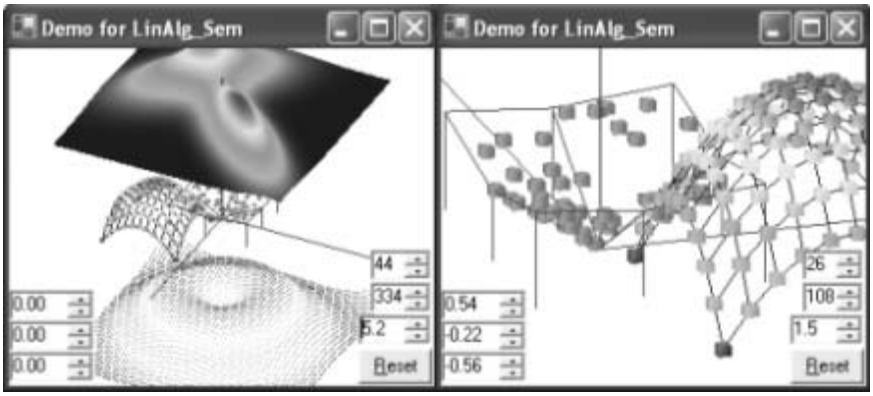
7. WHAT'S FURTHER?

To answer this question, we consider four topics: code management, code reuse, graphics, and applications.

Code management. `LinAlg` was created in 2001–2003; however, its enhancement has been continued. Managing such a system needs time and resources. The author is thankful to his PhD students for helping in code testing and writing documentation that is not complete even so. Thus, we have to improve the documentation.

Code reuse. Another problem is connected with code reuse. Let us take an example. To support the + operator in vector expressions, we had to implement the + operator for three classes VectorF, VectorD, and VectorS. With parametric polymorphism (also called as generics or templates in C++) it would be sufficient to implement the operator only once for a parametric class `Vector<T>` C# 1 does not provide parametric polymorphism; however, it will be included into the release 2. So our future plan is to redesign and reimplement `LinAlg` based on generics in the .NET 2005 version [5].

Graphics. The graphics system of `LinAlg` is flexible, it enables one to create plots in two different ways; however, it does not support interactive graphics (this question is not completely solved even in Maple or Matlab). Thus, one of the main tasks of the future will be creating interactive graphics. Leveraging the managed DirectX 9, we designed and built a component for plotting 3D charts. The last picture illustrates the plot types provided by this component. Our intention is to connect these two components.



Applications. So far `LinAlg` has played a key role in developing stand-alone Windows applications for wavelets [6], neural networks [7], polynomial approximation by DPT [8], adaptive piecewise cubic approximation [9], and time series analysis by ARI processes. There are many scientific problems that need solution and we intend to continue the sequence of applications created based on `LinAlg`. The third application was also converted to a Web application. Our plan is to develop some tools that will make the converting process simpler.

CONCLUSIONS

Visualization and data analysis are the key techniques for gaining information from measurements. `LinAlg` was created to support this process in the .NET framework. The paper presents the main classes of `LinAlg`. The use of the graphical, statistical, and numerical methods of these classes is illustrated with examples. Considerations on further enhancements of `LinAlg` are given too.

Acknowledgements. The author is thankful to V. V. Ivanov, whose interest in `LinAlg` stimulated the writing of this paper. He also thanks VEGA for supporting the project ***1/1006/04***.

REFERENCES

1. http://msdn.microsoft.com/netframework/technologyinfo/overview/
2. *Török Cs.* Vectorization and Operator Overloading in C# // Proc. of 7th Intern. Sci. Conf. «Applied Mathematics», Košice, 2002.
3. *Török Cs.* Matlab-like Programming in. NET Framework. Microsoft Research Acad. Conf. Budapest, 2003.

4. *Glynn J. et al.* Professional Windows GUI Programming Using C#. Wrox Press, 2002.

5. *Török Cs.* Generics-based Vectorization in MS. NET Rotor. JINR Preprint E10-2004-135. Dubna, 2004.

6. *Kepič T.* Descrete Wavelet Analysis // Intern. Seminar of Computing Statistics, Bratislava, Dec. 4–5, 2003. P. 199–202.

7. *Révayová M., Török Cs.* Analysis of Prediction with Neural Networks. Prastan, 2004. To be appeared.

8. *Matejčiková A.* On the Estimation of the Degree of Regression Polynomial // Ibid. P. 215–218.

9. *Török Cs., Dikoussar N. D.* MS .NET Components for Piecewise Cubic Approximation. To be appeared.

Received on August 31, 2004.

Редактор *Н. С. Скокова*
Макет *Н. А. Киселевой*

Подписано в печать 1.10.2004.
Формат 60 × 90/16. Бумага офсетная. Печать офсетная.
Усл. печ. л. 1,43. Уч.-изд. л. 2,21. Тираж 300 экз. Заказ № 54619.

Издательский отдел Объединенного института ядерных исследований
141980, г. Дубна, Московская обл., ул. Жолио-Кюри, 6.
E-mail: publish@pds.jinr.ru
www.jinr.ru/publish/