NMath Visualization Using the Microsoft Chart Controls

CenterSpace™ Software

Introduction

CenterSpace Software's **NMath**™ numerical library provides object-oriented components for mathematical, engineering, scientific, and financial applications on the .NET platform. **NMath** contains vector classes, matrix classes, complex number classes, random number generators, and other high-performance functions for object-oriented numerics.

NMath can be easily combined with the free Microsoft Chart Controls for .NET to create a complete data analysis and visualization solution. The `NMathChartMicrosoft.dll` assembly included with **NMath** provides convenience methods for plotting **NMath** types using the Microsoft Chart Controls. This document describes how to use these methods.

Microsoft Chart Controls for .NET

The Microsoft Chart Controls for .NET are available as a separate download for .NET 3.5.
`http://www.microsoft.com/downloads/en/details.aspx?FamilyId=130F7986-BF49-4FE5-9CA8-910AE6EA442C`

Beginning in .NET 4.0, the Chart controls are part of the .NET Framework.

To use the Chart controls, add a reference to `System.Windows.Forms.DataVisualization` and `using` statements

```
using System.Drawing;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;
```

Data Model

Although the **NMathChart** adapter described in the next section enables you create Microsoft Chart Controls from **NMath** types in just a few lines of code, let's begin by creating a chart manually, to review the data model.

In the Microsoft Chart Controls for .NET library, a **Chart** object contains one or more **ChartArea**s, each of which contain one or more data **Series**. Each **Series** has an associated chart type, and a **DataPoint** collection. **DataPoint**s can be manually appended or inserted into the collection, or added automatically when a series is bound to a datasource using either the `DataBindY()` or `DataBindXY()` method.

Since any **IEnumerable** can act as a datasource, it's easy to use an **NMath** vector or vector view (of a matrix row or column, for example) as a datasource. For example, suppose we want to create a scatter plot of the first two columns of a `20 x 5` **DoubleMatrix** (that is, column `0` vs. column `1`).

```
DoubleMatrix data =
  new DoubleMatrix( 20, 5, new RandGenUniform() );
DoubleVector x = data.Col( 0 );
DoubleVector y = data.Col( 1 );
```

Begin by creating a new **Chart** object, and optionally adding a **Title**.

```
Chart chart = new Chart()
{
  Size = new Size( 500, 500 ),
};

Title title = new Title()
{
  Name = chart.Titles.NextUniqueName(),
  Text = "My Data",
  Font = new Font( "Trebuchet MS", 12F, FontStyle.Bold ),
};
chart.Titles.Add( title );
```

Next, add a **ChartArea**.

```
ChartArea area = new ChartArea()
{
  Name = chart.ChartAreas.NextUniqueName(),
};
area.AxisX.Title = "Col 0";
area.AxisX.TitleFont =
  new Font( "Trebuchet MS", 10F, FontStyle.Bold );
area.AxisX.MajorGrid.LineColor = Color.LightGray;
area.AxisX.RoundAxisValues();
area.AxisY.Title = "Col 1";
area.AxisY.TitleFont = new Font( "Trebuchet MS", 10F,
FontStyle.Bold );
area.AxisY.MajorGrid.LineColor = Color.LightGray;
area.AxisY.RoundAxisValues();
chart.ChartAreas.Add( area );
```

Finally, add a new data **Series**, and bind the datasource to the **NMath** *x*, *y* vectors.

```
Series series = new Series()
{
  Name = "Points",
  ChartType = SeriesChartType.Point,
  MarkerStyle = MarkerStyle.Circle,
  MarkerSize = 8,
};
series.Points.DataBindXY( x, y );
chart.Series.Add( series );
```

To display the chart, you can use a utility function like this, which shows the given chart in a default form running in a new thread.

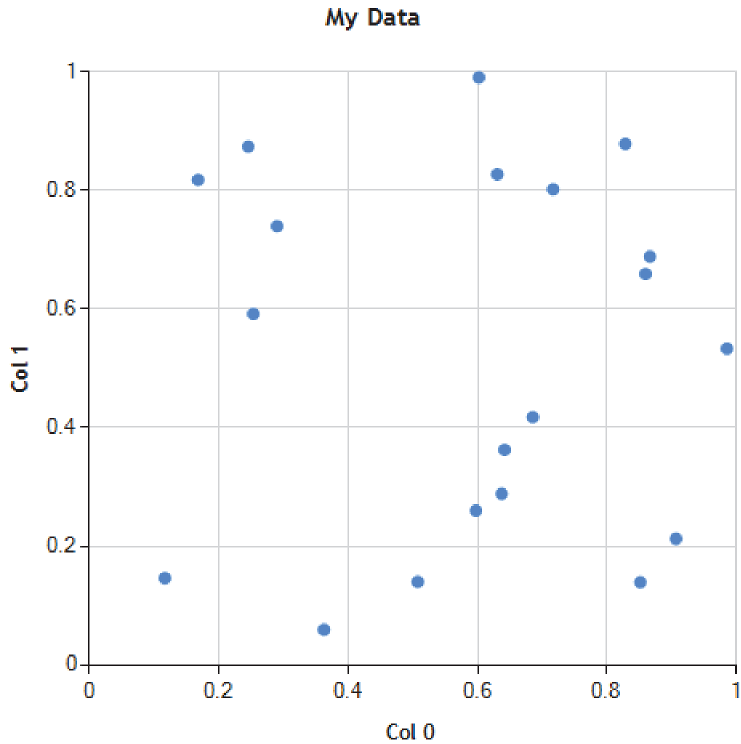
```
public static void Show( Chart chart )
{
  Form form = new Form();
  form.Size =
    new Size( chart.Size.Width + 20, chart.Size.Height + 40 );
  form.Controls.Add( chart );
  Thread t = new Thread( () => Application.Run( form ) );
  t.Start();
}
```

After calling

```
Show( chart );
```

the result looks like Figure 1.

Figure 1 – Chart control



NMath Chart Adapter

As an alternative to creating your own **Chart** controls from scratch, as shown above, **NMath** provides a convenient adapter class for the Microsoft Chart Controls for .NET. To use the adapter, add a reference to `NMathChartMicrosoft.dll` and `using` statement

```
using CenterSpace.NMath.Charting.Microsoft;
```

Class **NMathChart** provides static methods for plotting **NMath** types using the Microsoft Chart Controls. For example, this code reproduces the chart created manually in Figure 1.

```
DoubleMatrix data =
  new DoubleMatrix( 20, 5, new RandGenUniform() );

int xColIndex = 0;
int yColIndex = 1;
Chart chart = NMathChart.ToChart( data, xColIndex, yColIndex );

chart.Titles[0].Text = "My Data";
chart.Series[0].Name = "Points";
chart.Series[0].MarkerSize = 8;

NMathChart.Show( chart );
```

`ToChart()` returns an instance of `System.Windows.Forms.DataVisualization.Charting.Chart`. Overloads are provided for common **NMath** types. The returned **Chart** can be customized as desired. For prototyping and debugging console applications, the `Show()` function shows a given chart in a default form.

NOTE—When the window is closed, the chart is disposed.

If you do not need to customize the chart, overloads of `Show()` are also provided for common **NMath** types.

```
NMathChart.Show( data, xColIndex, yColIndex );
```

This is equivalent to calling:

```
NMathChart.Show(
  NMathChart.ToChart( data, xColIndex, yColIndex ) );
```

Controlling the Look of Charts

The design philosophy of **NMathChart** is to generate basic charts which simplify the binding of **NMath** data to chart controls. It is expected that chart consumers will want to customize the returned charts to meet their needs.

However, the default look of generated charts can also be controlled by static properties on **NMathChart**:

- `DefaultSize` gets and sets the default size for new charts using an instance of `System.Drawing.Size`.
- `DefaultTitleFont` gets and sets the default primary title font for new charts using an instance of `System.Drawing.Font`.

- `DefaultAxisTitleFont` gets and sets the default axis title font for new charts using an instance of `System.Drawing.Font`.
- `DefaultMajorGridLineColor` gets and sets the default major grid line color for new charts using an instance of `System.Drawing.Color`.
- `DefaultMarker` gets and sets the default marker for new charts using an instance of `System.Windows.Forms.DataVisualization.Charting.MarkerStyle`.

Units

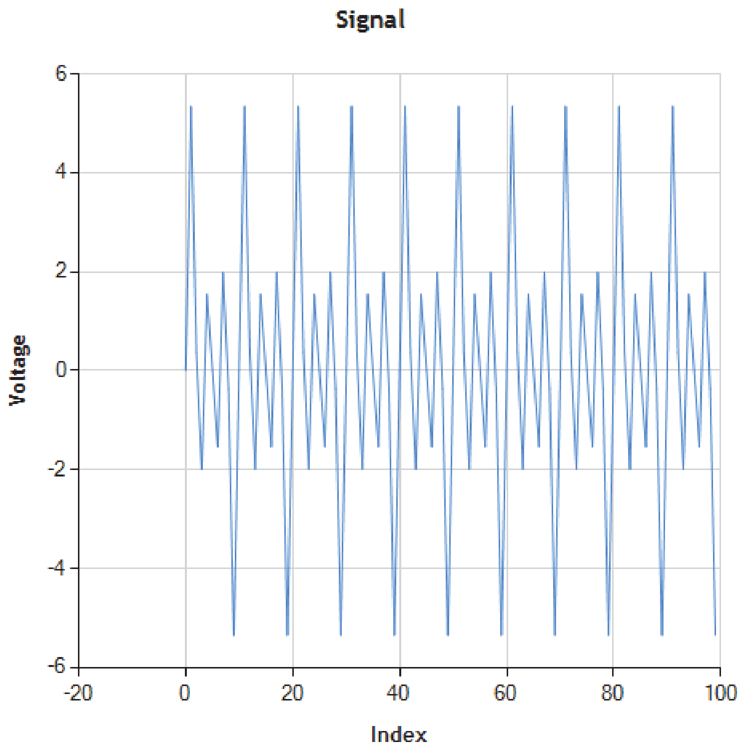
Class **NMathChart.Unit** represents a unit of physical quantity, and can be used specify axis units in cases where the **NMath** object does not provide this information.

A **NMathChart.Unit** is defined by a starting value, step size, and name. For example, suppose you are plotting a vector of signal data.

```
int n = 100;
DoubleVector t = new DoubleVector( n, 0, 0.1 );
DoubleVector signal = new DoubleVector( n );
for( int i = 0; i < n; i++ )
{
  signal[i] = Math.Sin( 2 * Math.PI * t[i] ) +
              2 * Math.Sin( 2 * Math.PI * 2 * t[i] ) +
              3 * Math.Sin( 2 * Math.PI * 3 * t[i] );
}

Chart chart = NMathChart.ToChart( signal );
chart.Titles[0].Text = "Signal";
chart.ChartAreas[0].AxisY.Title = "Voltage";
NMathChart.Show( chart );
```

The *y*-values are taken from the vector values, but how should the *x*-axis be labelled? By default, the vector indices are used, as shown in Figure 2.

Figure 2 – Vector data with default units

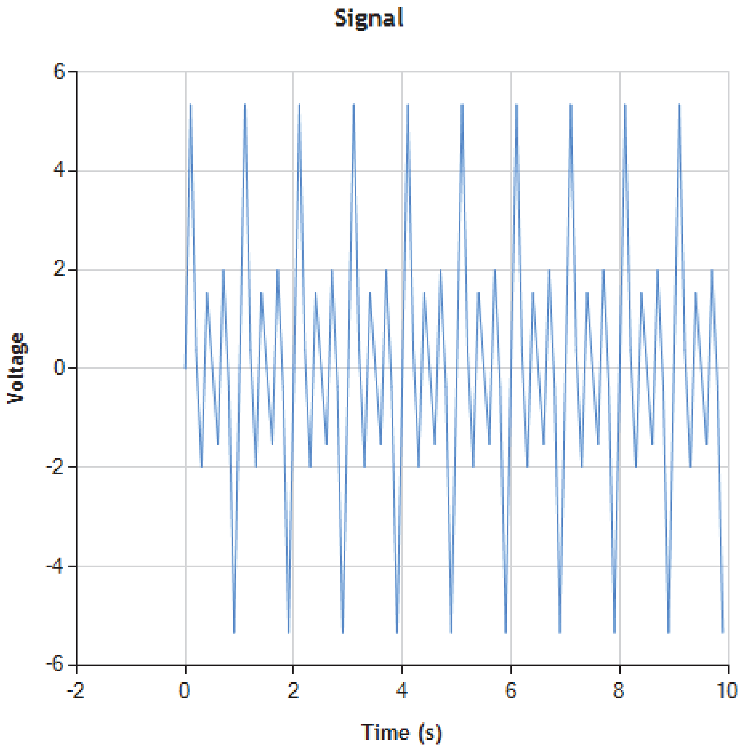


To set the units on the *x*-axis, provide an **NMathCart.Unit** instance to `ToChart()`.

```
NMathChart.Unit seconds =
  new NMathChart.Unit( 0, 0.1, "Time (s)" );

Chart chart = NMathChart.ToChart( signal, seconds );
chart.Titles[0].Text = "Signal";
chart.ChartAreas[0].AxisY.Title = "Voltage";
NMathChart.Show( chart );
```

The result is shown in Figure 3.

Figure 3 – Vector data with custom units



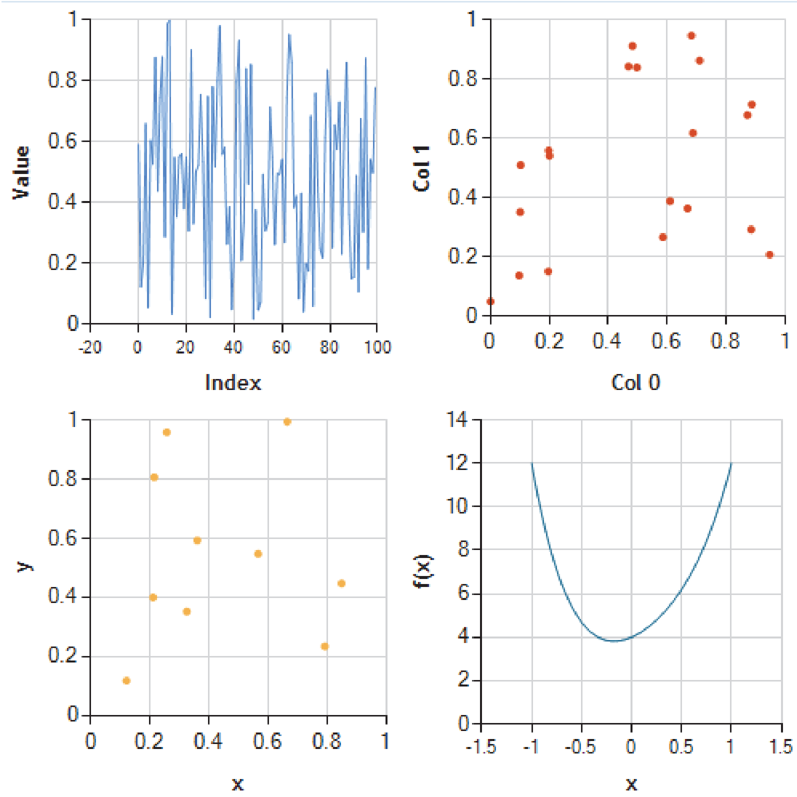
Composite Charts

Multiple charts can be combined in a composite **Chart** control, consisting of an *n* x *m* grid of component charts. The charts are supplied as an **IEnumerable<Chart>**, and the layout order is governed by a value from the `NMathChart.AreaLayoutOrder` enumeration.

```
RandGenUniform rnd = new RandGenUniform();
List<Chart> charts = new List<Chart>()
{
  NMathChart.ToChart( new DoubleVector( 100, rnd ) ),
  NMathChart.ToChart( new DoubleVector( 10, rnd ),
    new DoubleVector( 10, rnd ) ),
  NMathChart.ToChart( new DoubleMatrix( 20, 4, rnd ), 0, 1 ),
  NMathChart.ToChart( new Polynomial(
    new DoubleVector( 4, 2, 5, -2, 3 ) ), -1, 1, 100 )
};
```

```
Chart chart = NMathChart.Compose( charts, 4, 4,
  NMathChart.AreaLayoutOrder.ColumnMajor );

NMathChart.Show( chart );
```

Figure 4 – Composite chart



NOTE—Composite charts suppress component chart titles and legends.

Saving Charts

The `Save()` function saves a chart to a file or stream.

```
NMathChart.Save( chart, "chart.png", ChartImageFormat.Png );
```

`ChartImageFormat` is an enumeration in the `System.Windows.Forms.DataVisualization.Charting` namespace which enumerates many common image formats.

Using the Designer

If you are developing a Windows Forms application using the Designer, add a Microsoft Chart control to your form, then update it with an **NMath** object using the appropriate `Update()` function after initialization. For instance:

```
public Form1()
{
  InitializeComponent();

  Polynomial poly =
    new Polynomial( new DoubleVector( 4, 2, 5, -2, 3 ) );
  NMathChart.Update( ref this.chart1, poly, -1, 1 );
}
```

This has the following effect on your existing **Chart** object:

- a new, default **ChartArea** is added if one does not exist, otherwise `chart.ChartAreas[0]` is used
- axis titles, and `DefaultAxisTitleFont` and `DefaultMajorGridLineColor`, only have an effect if a new **ChartArea** is added
- titles are added only if the given **Chart** does not already contain any titles
- `chart.Series[0]` is replaced, or added if necessary

Using Charts in WPF

Class `System.Windows.Forms.DataVisualization.Charting.Chart` is a Windows Forms control. A **Chart** can still be displayed on a WPF page, however, by hosting it within a `System.Windows.Forms.Integration.WindowsFormsHost`.

1. Add references to:

   ```
   System.Windows.Forms
   System.Windows.Forms.Integration
   ```

2. In the XAML markup for a window (`MainWindow.xaml`, for example), add a **WindowsFormsHost** to the desired location element.

   ```
   <Grid>
       <WindowsFormsHost Name="ChartHost"/>
   </Grid>
   ```

3. In the code-behind for the window (`MainWindow.xaml.cs`, for example), set the `Child` property of the **WindowsFormsHost** to a **Chart**. For example:

```
public MainWindow() {

    InitializeComponent();

    double xmin = -Math.PI;
    double xmax = Math.PI;
    int numInterpolatedValues = 100;
    var chart = NMathChart.ToChart(
        NMathFunctions.CosFunction, xmin, xmax,
        numInterpolatedValues );

    ChartHost.Child = chart;
}
```

Plotting Vectors

NMath vector classes—**FloatVector**, **DoubleVector**, **FloatComplexVector**, and **DoubleComplexVector**—can be plotted in several ways.

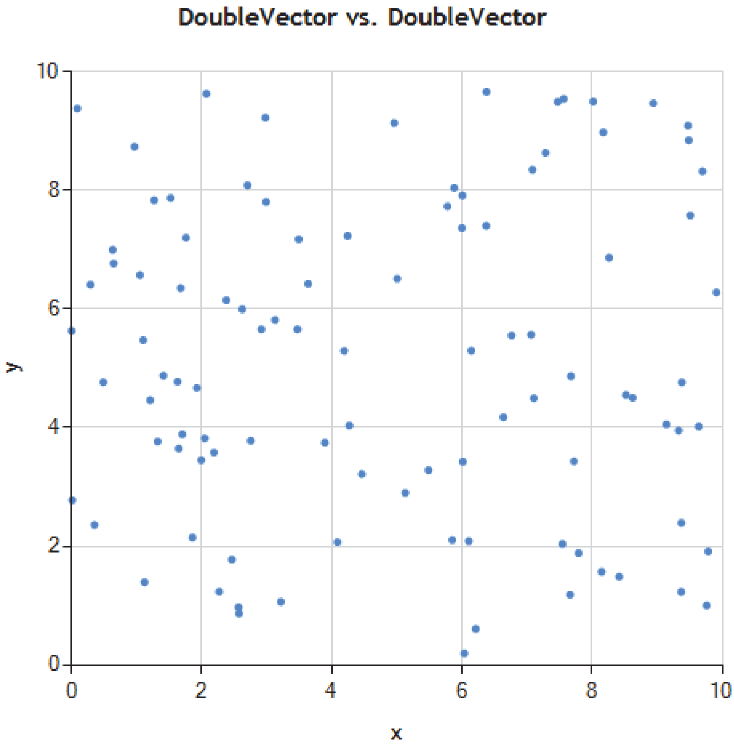
A single vector can be plotted as line data series, with the *x*-values taken from the vector indices (Figure 2) or a specified **NMathChart.Unit** object (Figure 3).

Two vectors can also be plotted against one another in a scatter plot, as shown in Figure 5. For example:

```
RandGenUniform rnd = new RandGenUniform( 0, 10 );
DoubleVector x = new DoubleVector( 100, rnd );
DoubleVector y = new DoubleVector( 100, rnd );
NMathChart.Show( x, y );
```

NOTE—Complex vector values are plotted as the absolute value. To plot complex values in the complex plane, plot the real and imaginary values against one another as a scatter plot.

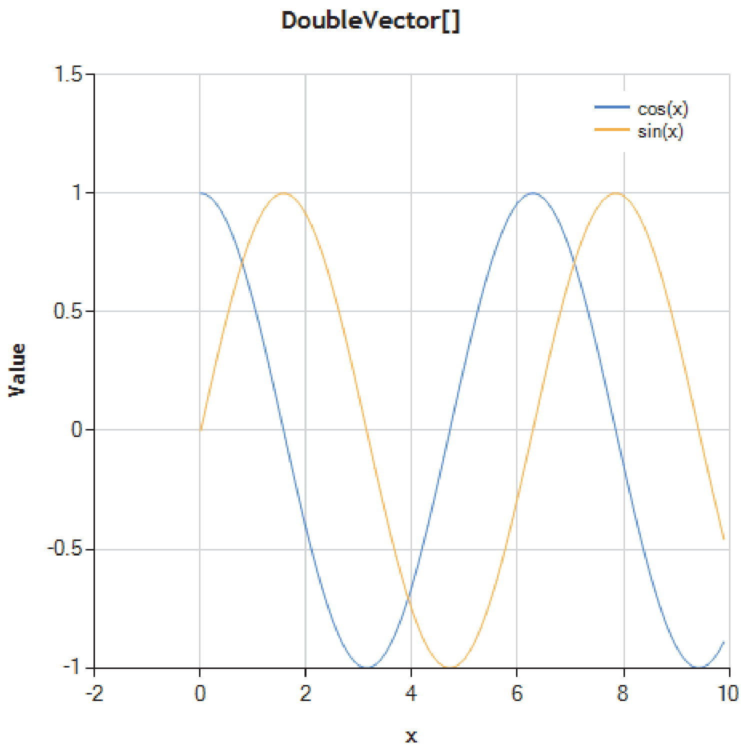
Figure 5 – Scatter plot



Finally, an array of vectors can be plotted as multiple data series.

```
DoubleVector x = new DoubleVector( 100, 0, 0.1 );
DoubleVector cos = x.Apply( NMathFunctions.CosFunction );
DoubleVector sin = x.Apply( NMathFunctions.SinFunction );

DoubleVector[] data = new DoubleVector[] { cos, sin };
NMathChart.Unit unit = new NMathChart.Unit( 0, 0.1, "x" );
Chart chart = NMathChart.ToChart( data, unit );

chart.Series[0].Name = "cos(x)";
chart.Series[1].Name = "sin(x)";
NMathChart.Show( chart );
```

Figure 6 – Multiple vector series

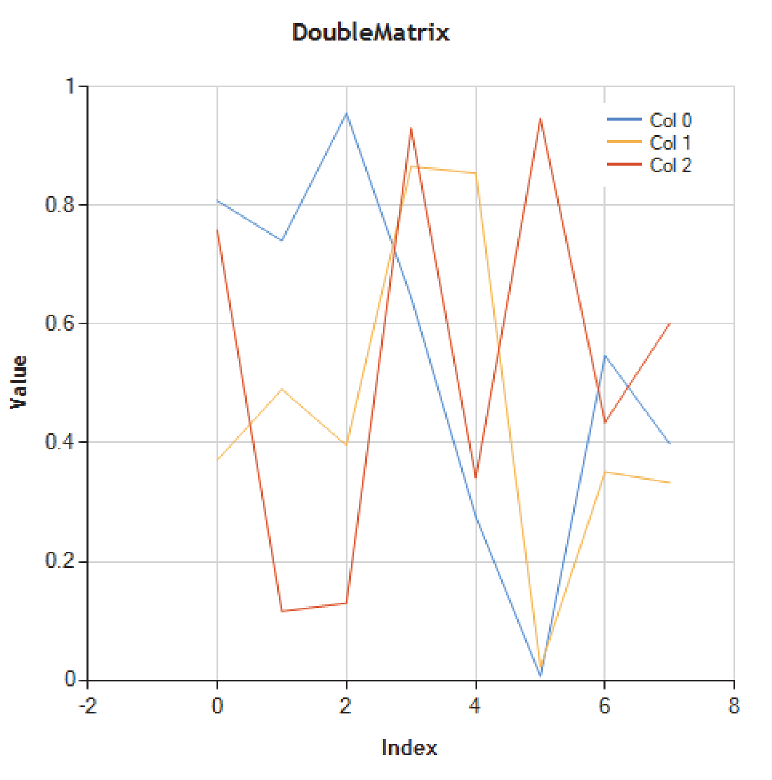


Plotting Matrices

NMath matrix classes—**FloatMatrix**, **DoubleMatrix**, **FloatComplexMatrix**, **DoubleComplexMatrix**—can also be plotted in many ways. By default, each matrix column is plotted as a separate data series. (To plot the matrix rows, simply `Transpose()` first.)

```
RandGenUniform rnd = new RandGenUniform();
DoubleMatrix A = new DoubleMatrix( 8, 3, rnd );
NMathChart.Show( A )
```

Figure 7 – Matrix columns

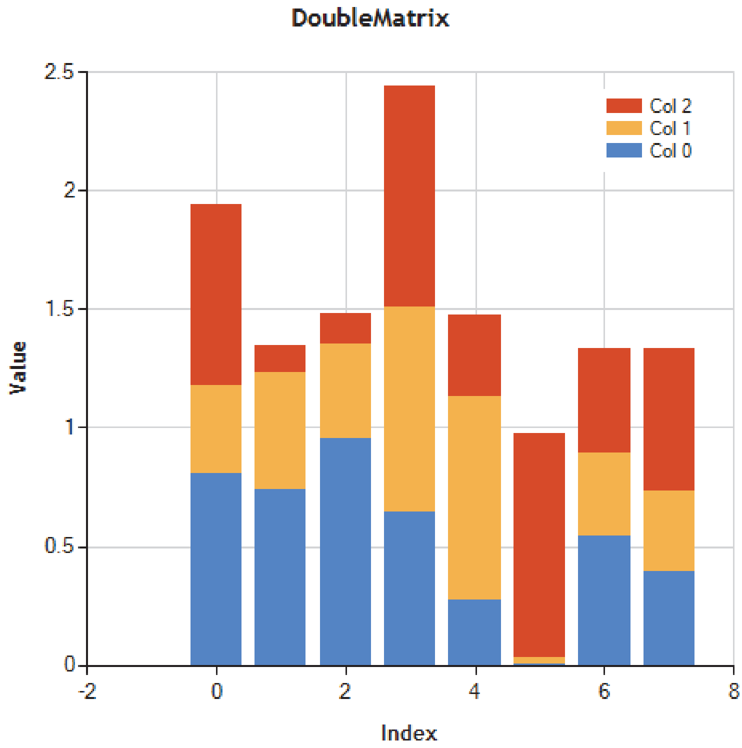


NOTE—Complex matrix values are always plotted as the absolute value.

The generated chart uses a line chart to display each series, but this can easily be customized. For instance, this code uses a stacked column chart.

```
Chart chart = NMathChart.ToChart( A );
foreach ( Series series in chart.Series )
{
  series.ChartType = SeriesChartType.StackedColumn;
}
NMathChart.Show( chart );
```

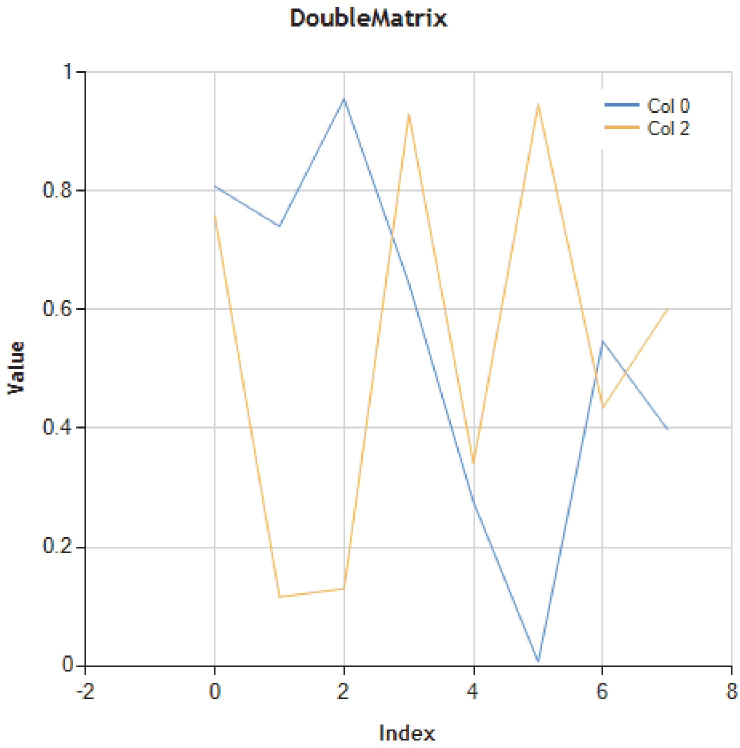
Figure 8 – Matrix columns with custom chart type



By default, all matrix columns are plotted, but you can optionally specify an array of column indices to plot.

```
int[] colIndices = new int[] { 0, 2 };
NMathChart.Show( A, colIndices );
```

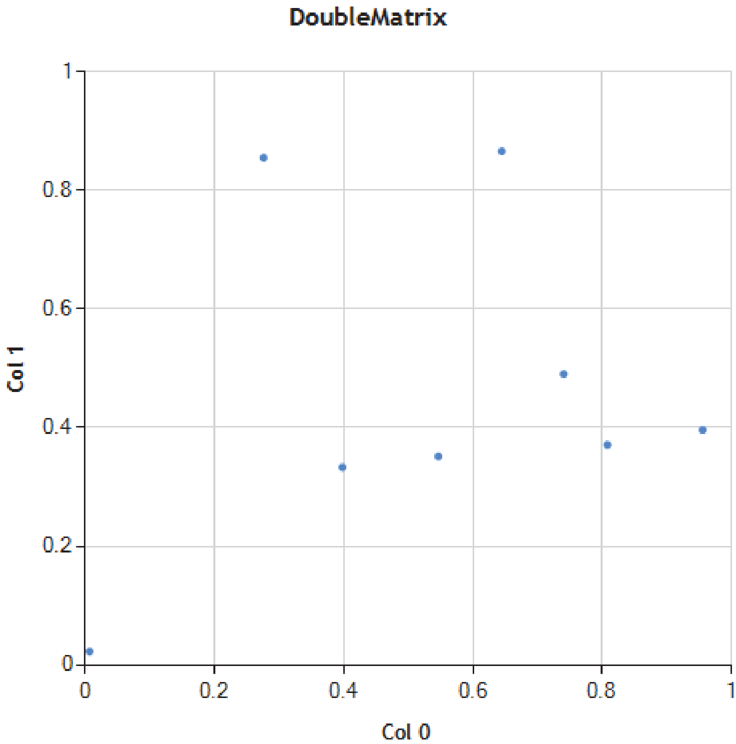
Figure 9 – Matrix columns by index



Lastly, you can specify two column indices to plot versus one another in a scatter plot.

```
int xColIndex = 0;
int yColIndex = 1;
NMathChart.Show( A, xColIndex, yColIndex );
```

Figure 10 – Matrix column scatter plot



Plotting Functions

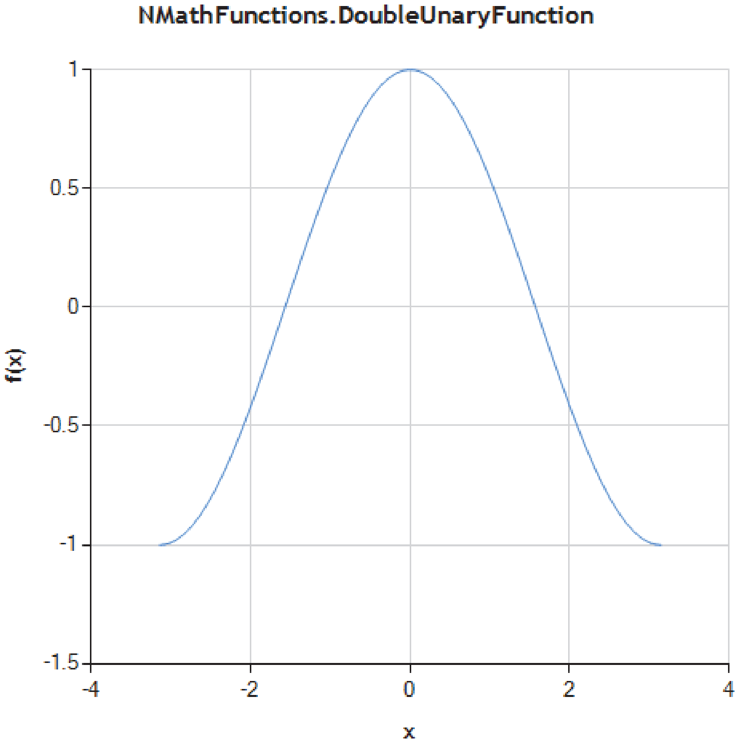
NMath provides various classes and delegates for encapsulating functions of one variable:

- Delegate `Func<double, double>`, a functor that takes a double-precision floating point number and returns a double-precision floating point number.
- Class **OneVariableFunction**, and derived type **Polynomial**
- Classes **DoubleParameterizedFunction** and **DoubleParameterizedDelegate**, and delegate `NMathFunctions.GeneralizedDoubleUnaryFunction`, for representing parameterized functions.

NMathChart plots **NMath** functions by interpolating over a given function within a specified range. For instance:

```
double xmin = -Math.PI;
double xmax = Math.PI;
int numInterpolatedValues = 100;
NMathChart.Show( NMathFunctions.CosFunction, xmin, xmax,
    numInterpolatedValues );
```

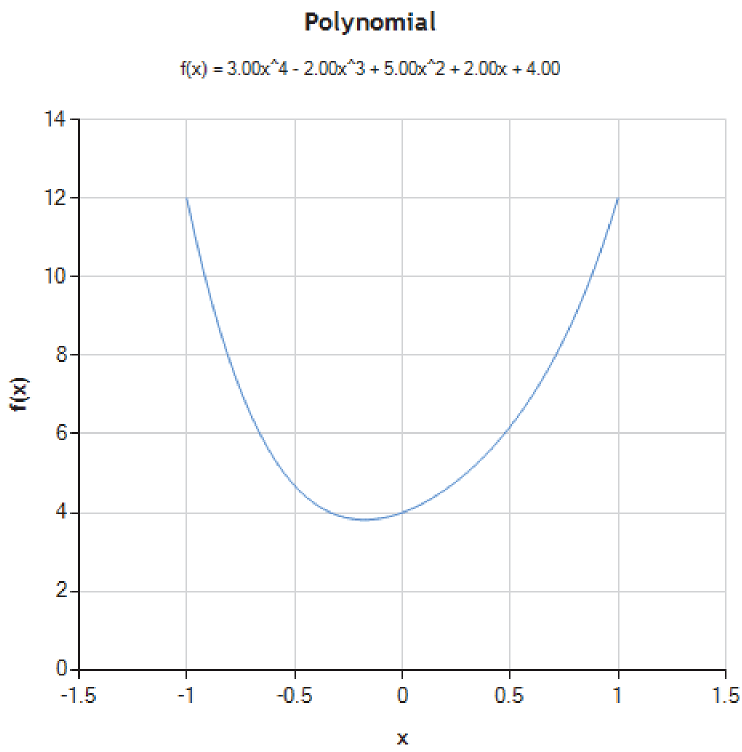
Figure 11 – Interpolated function plot



In cases where the analytic form of the function is known, such as in a **Polynomial** instance, the function is displayed in a subtitle.

```
Polynomial poly =
    new Polynomial( new DoubleVector( 4, 2, 5, -2, 3 ) );
NMathChart.Show( poly, xmin, xmax, numInterpolatedValues );
```

Figure 12 – Polynomial

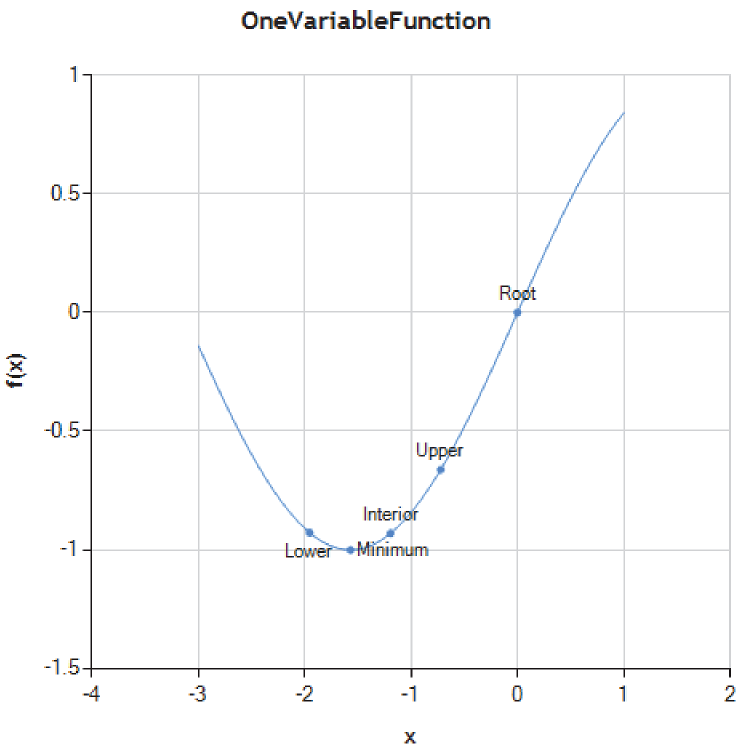


Optionally, you can also provide a **Dictionary** of function point labels. For example, this code uses a **BrentMinimizer** to find a function minimum within a given bracketed range, and a **RiddersRootFinder** to find a function root, then labels those points in the generated chart.

```
OneVariableFunction f = new OneVariableFunction(
NMathFunctions.SinFunction );

double xmin = -3.0;
double xmax = 1.0;
int numInterpolatedValues = 100;

Bracket bracket = new Bracket( f, 0.01, 0.02 );
BrentMinimizer minimizer = new BrentMinimizer();
double min = minimizer.Minimize( bracket );

RiddersRootFinder finder = new RiddersRootFinder();
double root = finder.Find( f, xmin, xmax );
```

```
Dictionary<double, string> pointLabels =
  new Dictionary<double, string>()
{
  { min, "Minimum" },
  { root, "Root" },
  { bracket.Lower, "Lower" },
  { bracket.Interior, "Interior" },
  { bracket.Upper, "Upper" },
};

NMathChart.Show( f, xmin, xmax, numInterpolatedValues,
  pointLabels );
```

Figure 13 – Function point labels



Plotting Fitted Functions

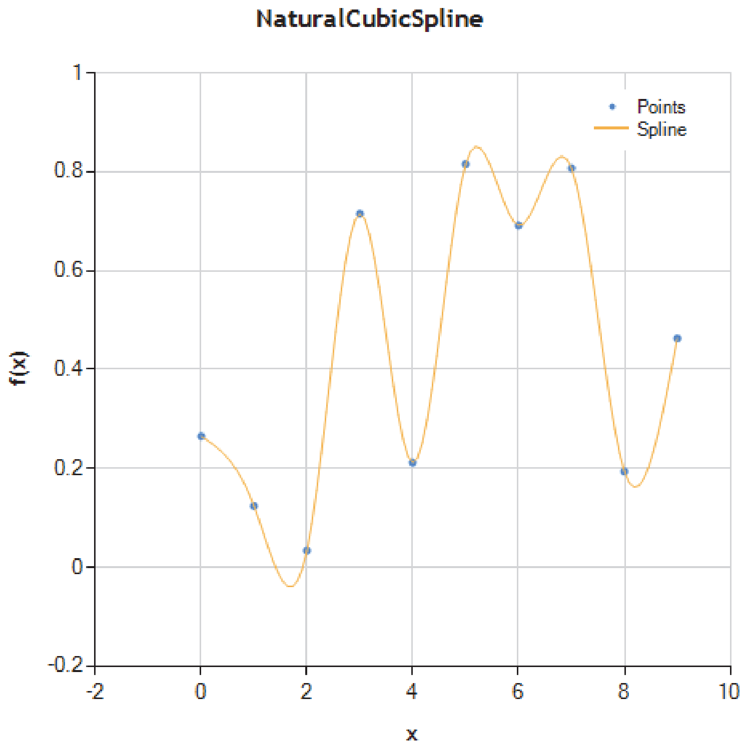
NMath provides various classes for fitting functions to data:

- Classes **LinearSpline**, **ClampedCubicSpline**, and **NaturalCubicSpline** for spline interpolation
- Class **PolynomialLeastSquares** for polynomial fitting.
- Classes **OneVariableFunctionFitter** and **BoundedOneVariableFunctionFitter** for non-linear least squares fitting of arbitrary functions.

NMathChart plots fitted functions by interpolating over the fitted function within a specified range, and displaying the original data in a separate data series. For instance:

```
DoubleVector x = new DoubleVector( 10, 0, 1 );
DoubleVector y = new DoubleVector( 10, new RandGenUniform() );
NaturalCubicSpline spline = new NaturalCubicSpline( x, y );
NMathChart.Show( spline, numInterpolatedValues );
```

Figure 14 – Tabulated values

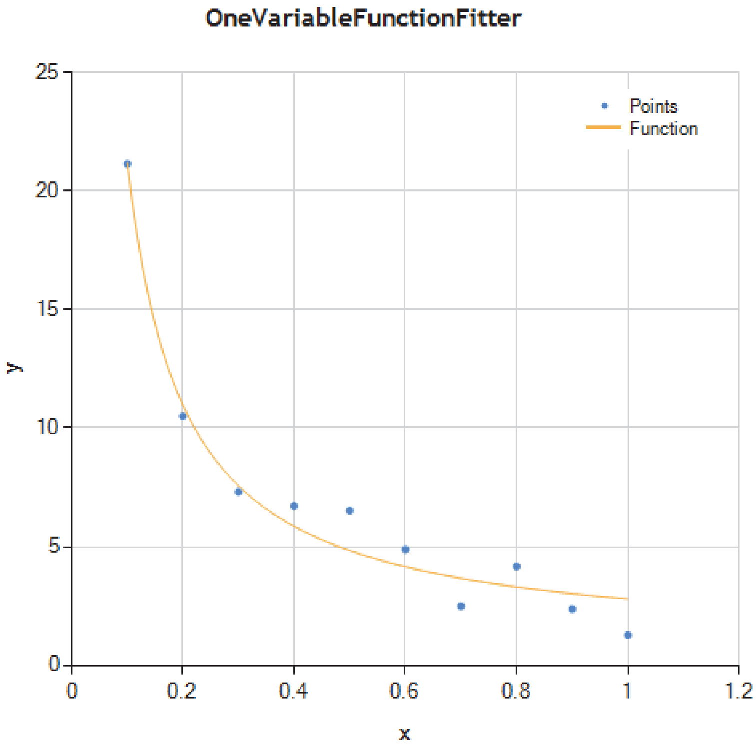


This code fits a two-parameter asymptotic ($y = a + \frac{b}{x}$) to a set of data points and plots the result.

```
DoubleParameterizedDelegate f =
  new DoubleParameterizedDelegate(
    AnalysisFunctions.TwoParameterAsymptotic );

DoubleVector x = new DoubleVector( 10, 0.1, 0.1 );
DoubleVector y = new DoubleVector( x.Length );
DoubleVector target_parameters = new DoubleVector( "1 2" );
RandGenUniform rnd = new RandGenUniform( -2, 2 );
for( int i = 0; i < y.Length; i++ )
{
  // add noise
  y[i] = f.Evaluate( target_parameters, x[i] ) + rnd.Next();
}
```

```
OneVariableFunctionFitter<TrustRegionMinimizer> fitter =
  new OneVariableFunctionFitter<TrustRegionMinimizer>( f );
DoubleVector start = new DoubleVector( "0.1 0.1" );
DoubleVector solution = fitter.Fit( x, y, start );

int numInterpolatedValues = 100;
NMathChart.Show( fitter, x, y, solution, numInterpolatedValues );
```

Figure 15 – Fitted asymptotic

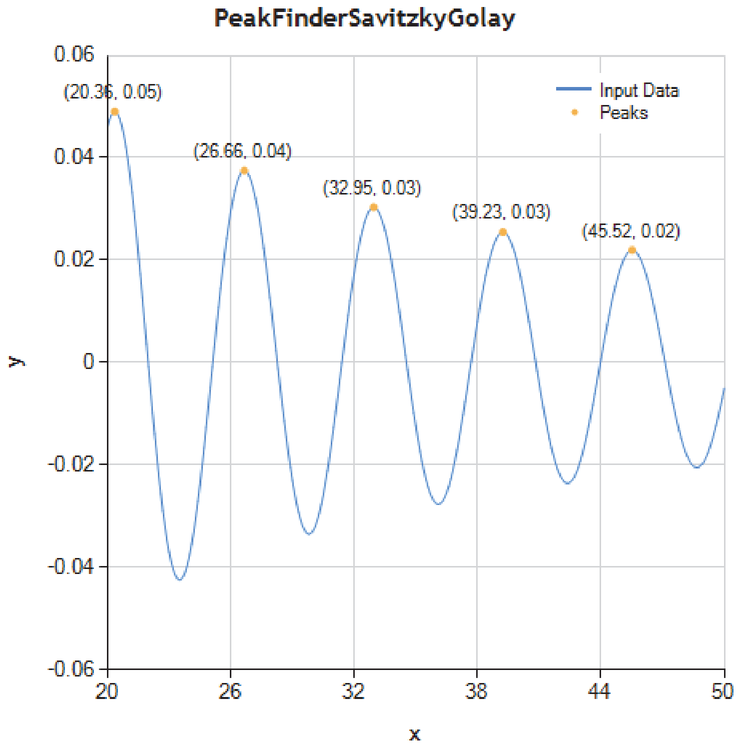


Plotting Function Peaks

Class **PeakFinderSavitzkyGolay** uses smooth Savitzky-Golay derivatives to find peaks in data. **NMathChart** supports plotting the found peaks, as shown in Figure 16.

```
double step_size = 0.1;
DoubleVector x = new DoubleVector( 1000, 0.01, step_size );
DoubleVector v = NMathFunctions.Sin( x ) / x;
PeakFinderSavitzkyGolay pf =
  new PeakFinderSavitzkyGolay( v, 5, 4 );

pf.AbscissaInterval = step_size;
pf.SlopeSelectivity = 0;
pf.RootFindingTolerance = 0.0001;
pf.LocatePeaks();

double xmin = 20;
double xmax = 50;
NMathChart.Show( pf, xmin, xmax );
```

Figure 16 – Peaks in data

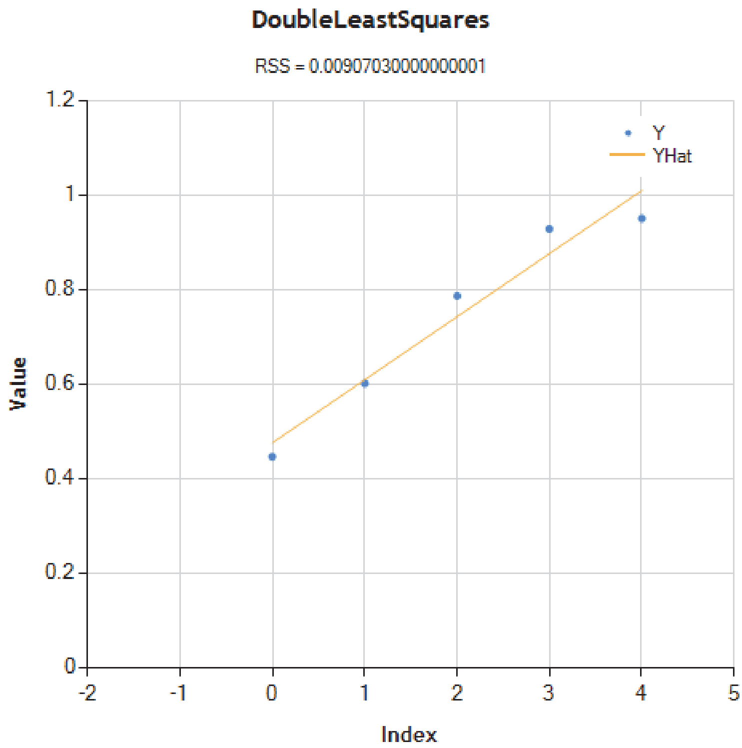


Plotting Least Squares Classes

NMath provides classes **FloatLeastSquares**, **DoubleLeastSquares**, **FloatComplexLeastSquares**, and **DoubleComplexLeastSquares** for computing the minimum-norm solution to a linear system `Ax = y`. **NMathChart** plots the fitted line and residual sum of squares (RSS).

```
DoubleMatrix A = new DoubleMatrix(
  " 5x2[1.0 20.0  1.0 30.0  1.0 40.0  1.0 50.0  1.0 60.0]" );
DoubleVector y = new DoubleVector( "[.446 .601 .786 .928 .950]" );
DoubleLeastSquares lsq = new DoubleLeastSquares( A, y );
NMathChart.Show( lsq, y );
```

Figure 17 – Least squares

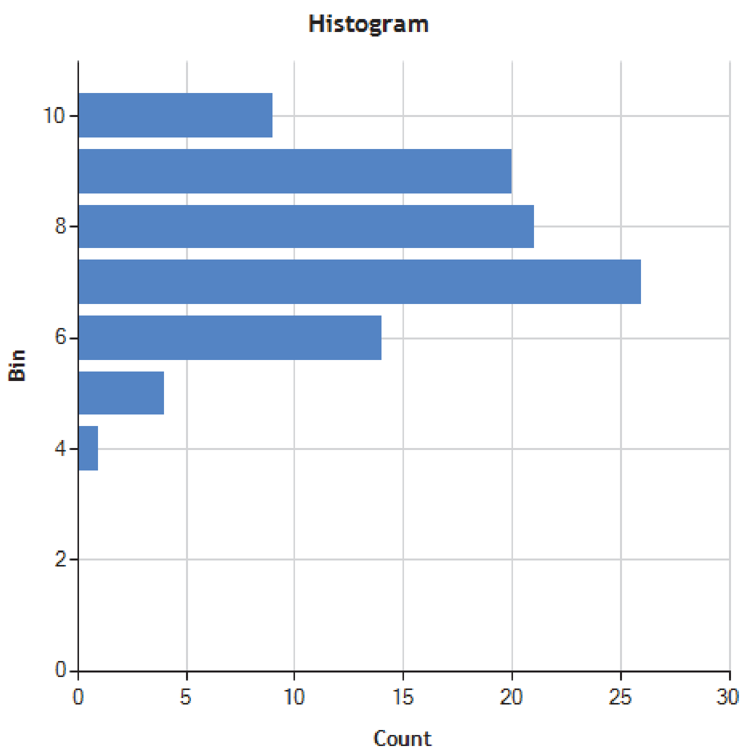


Plotting Histograms

Class **Histogram** constructs and maintains a histogram of input data. Input data is sorted into bins and a count is kept of how many data points fall into each bin. **NMathChart** supports plotting histograms, as shown in Figure 18.

```
Histogram histogram = new Histogram( 10, 0.0, 100.0 );
DoubleVector data =
  new DoubleVector( 100, new RandGenNormal( 70, 200 ) );
histogram.AddData( data );
NMathChart.Show( histogram );
```

Figure 18 – Histogram chart



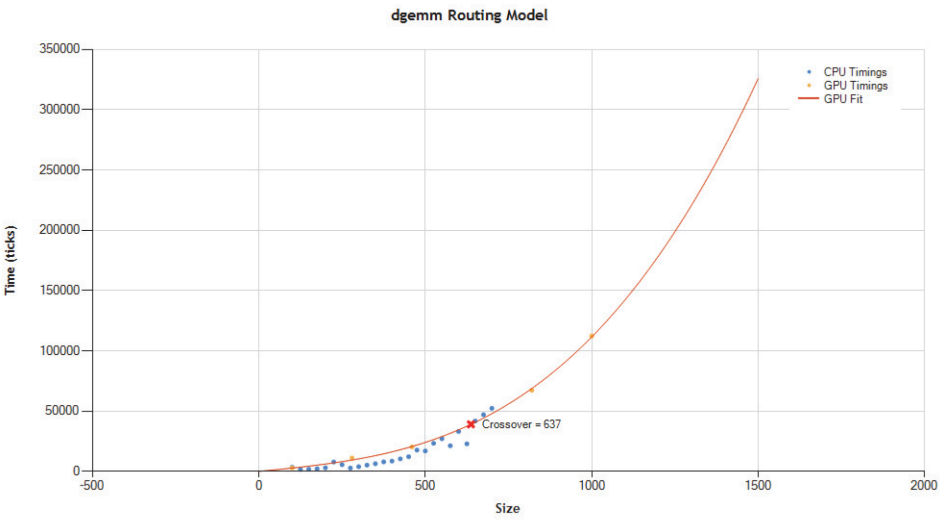
Plotting GPU Routing Models

The **Premium Edition** of **NMath** leverages the power of NVIDIA's CUDA™ architecture for accelerated performance. **NMath Premium**'s Adaptive Bridge™ technology provides automatic performance tuning of individual CPU–GPU routing to insure optimal hardware usage. After tuning a GPU-enabled function, you can visualize the recorded CPU and GPU benchmarks, and fitted GPU timing model, using the provided convenience functions on **NMathChart**, as shown in Figure 19.

```
var bmanager = BridgeManager.Instance;
var device = bmanager.GetComputeDevice( 0 );
var bridge = bmanager.GetBridge( device );
```

```
var f = BridgeFunctions.dgemm;
bridge.Tune( f, device, 1000 );

double xmin = 0;
double xmax = 1500;
int numInterpolatedValues = 250;
var chart = NMathChart.ToChart( bridge, f, device, xmin, xmax,
  numInterpolatedValues );

NMathChart.Show( chart );
```

Figure 19 – Routing model chart



Conclusions

NMath types can be easily plotted using the **NMathChart** adapter and the free Microsoft Chart Controls for .NET, creating a complete solution for data analysis and visualization.

NMATH VISUALIZATION USING THE MICROSOFT CHART CONTROLS

The correct bibliographic reference for this document is:
NMath Visualization Using the Microsoft Chart Controls, CenterSpace Software, Corvallis, OR.

Printed in the United States.
Printing Date: March, 2016

CENTERSPACE SOFTWARE

Address:	622 NW 32nd St., Corvallis, OR 97330 USA
Phone:	(541) 896-1301
Web:	`http://www.centerspace.net`
Technical Support:	`support@centerspace.net`