NPlot Problems:
Over the weekend I spent a lot of time trying to generate a simple step plot. As mentioned in the previous notebook entry it was not quite working. Declaring the variable for the plot surface without the new command

```
private NPlot.Windows.PlotSurface2D plot;
```

was not working, as C# needs the variable to be initialized by their class constructor so the command needed was:

```
private NPlot.Windows.PlotSurface2D plot = new
Nplot.Windows.PlotSurface2D();
```

However, every example code from the NPlot declared it the first way, I looked at last year's code and it was done like the first method. After looking at all possibilities, I decided to write the code the second way and it compiled correctly, but the plotting surface wasn't added to the window. I tried looking for NPlot documentation.

NPlot Documentation:
There seems to be a steep learning curve, trying to use the DLL and the functions as intended. I am still trying to figure out what sort of initializations are mandatory, and what provide added features. Documentation available for it is rather sketchy, the examples packaged with the library are too robust so it is hard to glean basic information from it. I noticed that the in-code documentation was prepared in a way that Doxygen could extract most of the functional information. I generated the documentation using Doxygen, and after some tweaking it is finally useful.
After all that, I did find an online documentation for the NPlot API.

Soliciting outside help:
Even after looking at the documentation which provided function calls and such, but provided no instructions on how to use them, I still wasn't able to plot anything. I called Jason Nielsen (worked on code last year). After looking at everything he said, it all looked fine and he had no idea why it was not plotting anything. He vaguely remembered dragging and dropping the plot surface instead of writing code to include it. He could not remember how that was achieved. He said I should look into design toolbox.

Design Toolbox:
Visual Studio .NET 2003 provides for a graphical IDE in building windows applications. The design toolbox has objects that can be dragged and dropped onto the application window, instead of having to generate code for all objects the code is automatically created to reduce the programming burden and allowing the programmer to place the objects precisely where they need to be graphically.
I searched high and low through the toolbox to find where the NPlot plot surface was, but apparently it was no where. I checked again to make sure that the NPlot DLL was referenced correctly, and it appeared under references.
Finally, I figured out that the DLL needed to be separately added to the Toolbox so the toolbox could then display the plot surface as an object. There is a lot of hidden code initializations that occur and that is why my previous attempt at writing all the code wasn't very successful, since I was not initializing everything, only what was stated by the API.

Victory at last:
After adding the DLL to the toolbox and dragging the plot surface onto the application window, all the code I had written worked correctly to produce Figure 2.1: Simple Step Plot.

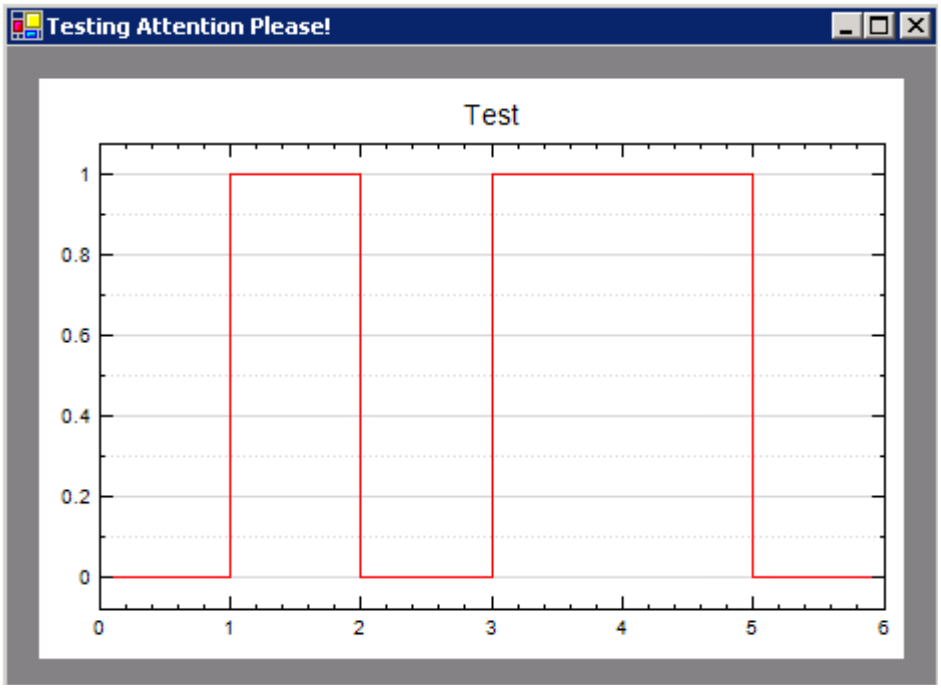

Figure 2.1 Simple Step Plot

The code will be included here since this is the simplest form using the NPlot library, and this will be a good reference for a bare bone basics. Code 5.1: Bare bone basic code for Step Plot will probably be the last time all the code is included for any part of the testing process. Code excerpts will assume full code for the program to function as shown below.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Reflection;

using NPlot;

namespace initialtest
{
 //! Test Class just creates an application window and attaches a
 //! plot surface area to the window and draws a step plot.
 public class test : System.Windows.Forms.Form
 {
      private System.ComponentModel.Container components = null;
      private NPlot.Windows.PlotSurface2D plot;
```

```
//! Constructor initializes components and calls function plotline()
//! on this instance of the of object.
public test()
{
      InitializeComponent();
      this.plotline();
}

//! Creates one stepplot of six values that have been hard coded.
public void plotline()
{
      // --- Plotting ---
      plot.Clear();

      // --- Grid Code ---
      Grid mygrid = new Grid();
      mygrid.HorizontalGridType = Grid.GridType.None;
      mygrid.VerticalGridType = Grid.GridType.Fine;

      plot.Add(mygrid);
      plot.Title = "Test";

      StepPlot line = new StepPlot();
      line.Pen = new Pen (Color.Red, 1);
      line.OrdinateData = new int [] {0, 1, 0, 1, 1, 0};
      line.HideVerticalSegments = false;
      plot.Add(line);
      plot.Refresh();

      return;
}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
      if( disposing )
      {
            if (components != null)
            {
                  components.Dispose();
            }
      }
      base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
```

```
      private void InitializeComponent()
      {
            this.plot = new NPlot.Windows.PlotSurface2D();
            this.SuspendLayout();
            //
            // plot
            //
            this.plot.AutoScaleAutoGeneratedAxes = false;
            this.plot.AutoScaleTitle = false;
            this.plot.BackColor =
System.Drawing.SystemColors.ControlLightLight;
            this.plot.DateTimeToolTip = false;
            this.plot.Legend = null;
            this.plot.LegendZOrder = -1;
            this.plot.Location = new System.Drawing.Point(16, 16);
            this.plot.Name = "plot";
            this.plot.Padding = 10;
            this.plot.RightMenu = null;
            this.plot.ShowCoordinates = true;
            this.plot.Size = new System.Drawing.Size(432, 288);
            this.plot.SmoothingMode =
System.Drawing.Drawing2D.SmoothingMode.None;
            this.plot.TabIndex = 0;
            this.plot.Text = "plot";
            this.plot.Title = "";
            this.plot.TitleFont = new System.Drawing.Font("Arial", 14F,

 System.Drawing.FontStyle.Regular,

 System.Drawing.GraphicsUnit.Pixel);
            this.plot.XAxis1 = null;
            this.plot.XAxis2 = null;
            this.plot.YAxis1 = null;
            this.plot.YAxis2 = null;
            //
            // test
            //
            this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
            this.BackColor = System.Drawing.SystemColors.AppWorkspace;
            this.ClientSize = new System.Drawing.Size(464, 317);
            this.Controls.Add(this.plot);
            this.Name = "test";
            this.Text = "Testing Attention Please!";
            this.ResumeLayout(false);

      }
      #endregion

      /// <summary>
      /// The application creates an instance of the class test.
      /// </summary>
      [STAThread]
      static void Main()
```

```
        {
                Application.Run(new test());
        }

    }
}
```

Code 5.1 Bare bone basic code for Step Plot

Documentation

I decided to start documenting the code the way I would like to document for the entire project, so that it is extractable by Doxygen. I ran Doxygen and it extracted the documentation correctly, that was encouragement enough to document more, and do a better job of documentation.

Multiple Lines

At this point I started expanding the code to handle an arbitrary number of lines specified by a variable called numlines. I am creating an array of plot step lines and plotting them one by one.