```
using System;
using MathNet.Numerics.LinearAlgebra;
using MathNet.Numerics.LinearAlgebra.Double;
using static System.Console;
using static System.Globalization.CultureInfo;
using static System.Math;
using static MathNet.Numerics.Generate;
using MMP.Tools;
using MMP.Tools.Double;
using MMP.Tools.DoubleDouble;
using static MMP.Tools.GnuPlot;

namespace MMP
{

    class Program
    {
        static Terminal win = new Terminal(1200, 600);
        static Terminal png = new Terminal(1200, 600, Terminal.Enum.png);
        static Terminal eps = new Terminal(12,6,Terminal.Enum.eps);
        static Terminal tex = new Terminal(Terminal.Enum.epslatex);
        static Terminal svg = new Terminal(Terminal.Enum.svg);
        static Terminal pdf = new Terminal(12,6,Terminal.Enum.pdf);

        static void test_poly()
        {
            var x = new Polynomial.Monom(1);
            AnnotatedFunc<double>[] T = new AnnotatedFunc<double>[10];
            T[0] = new AnnotatedFunc<double>(t=>Sin(t)/t,"sin(x)/x");
            for (int i = 1; i < T.Length; i++)
            {
```

```
            var tmp = Polynomial.Sin(2 * i - 1) / x;
            T[i] = new AnnotatedFunc<double>(tmp.Eval, $"T_{{{2*i-1}}}");
        }
        var u = LinearSpaced(51, 0, 2*PI);
        GraphList<double> glist = new GraphList<double>(u,T);
        Terminals = new Terminal[] { win };
        Set = "yrange [-0.5:1.5]";
        Plot(glist);
        WaitForKey();
    }
    static void test_expressions()
    {
        Set = "yrange [-0.5:1]";
        Plot("sin(x)/x,1,1-x**2/6,1-x**2/6+x**4/120");
        WaitForKey();
        Reset();
    }

    static void test_graphs()
    {
        var x = LinearSpaced(51, 0, 4 * PI);
        GraphList<double> glist =
            new GraphList<double>(x,
            new AnnotatedFunc<double>(Sin, "sin(x)"),
            new AnnotatedFunc<double>(t=>Sin(2*t), "sin(2*x)"));
        Plot(glist);
        WaitForKey();
    }

    static void test_curves()
    {
```

```
    var x = LinearSpaced(51, 0, 2 * PI);
    Ellipse circle1 = new Ellipse();
    Ellipse circle2 = new Ellipse(0.5, 0.5);
    Ellipse ellipse = new Ellipse(0.25, 0.25, 2, 0.25, PI / 4);

    CurveList<double> clist =
        new CurveList<double>(x,
        circle1.Curve("Circle_1"),
        circle2.Curve("Circle_2"),
        ellipse.Curve("Ellipse"));

    Set = "size ratio - 1";
    Terminal[] term = { eps, pdf, win };
    Terminals = term;
    Plot(clist);
    WaitForKey();
}

static void test_surface()
{
    double[] u = LinearSpaced(51,0,2*PI);
    double[] v = LinearSpaced(26, 0, PI);
    Terminal[] term = { eps, pdf, win };
    Terminals = term;
    Set = "xlabel 'x'";
    Set = "hidden3d";
    Plot((x, y) => Sin(x) * Cos(y), u, v );
    WaitForKey();
}

static void test_mesh()
```

```
        {
            double[] x = new LinSpace(0, 50, 51);
            double[] y = new LinSpace(0, 20, 21);
            MeshGrid<double> mgrid = new MeshGrid<double>(x, y);
            Set = "size ratio -1";
            UnSet = "border";
            UnSet = "xtics";
            UnSet = "ytics";
            Plot(mgrid.Triangulation);
            WaitForKey();
        }
        static void Main(string[] args)
        {
            DefaultThreadCurrentCulture = InvariantCulture;
            //test_expressions();
            //test_graphs();
            //test_curves();
            //test_surface();
            //test_mesh();
            test_poly();
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MMP.Tools
{
```

```
public class AnnotatedFunc<T>
{
    Func<T, T> f;
    public string Title;
    public AnnotatedFunc(Func<T, T> f, string title)
    {
        this.f = f;
        Title = title;
    }
    public T[] F(T[] t)
    {
        Func<T[], T[]> F = (Vectorize<T, T>)f;
        return F(t);
    }
}

public class AnnotatedCurve<T>
{
    Func<T, T> x, y;
    public string Title;
    public AnnotatedCurve(Func<T, T> x, Func<T, T> y, string title)
    {
        this.x = x;
        this.y = y;
        Title = title;
    }

    public T[] X(T[] t)
    {
        Func<T[], T[]> X = (Vectorize<T, T>)x;
        return X(t);
```

```
    }

    public T[] Y(T[] t)
    {
        Func<T[], T[]> Y = (Vectorize<T, T>)y;
        return Y(t);
    }
}

public class AnnotatedCurve3D<T>
{
    Func<T, T> x, y, z;
    public string Title;
    public AnnotatedCurve3D(Func<T, T> x, Func<T, T> y, Func<T, T> z, string title)
    {
        this.x = x;
        this.y = y;
        this.z = z;
        Title = title;
    }

    public T[] X(T[] t)
    {
        Func<T[], T[]> X = (Vectorize<T, T>)x;
        return X(t);
    }

    public T[] Y(T[] t)
    {
        Func<T[], T[]> Y = (Vectorize<T, T>)y;
        return Y(t);
```

```
        }

        public T[] Z(T[] t)
        {
            Func<T[], T[]> Z = (Vectorize<T, T>)y;
            return Z(t);
        }

    }
}


using System;


namespace MMP.Tools
{
    public class Vectorize<T1, T2>
    {
        Func<T1, T2> f;
        public Vectorize(Func<T1, T2> f)
        { this.f = f; }
        public T2[] F(T1[] x)
        {
            T2[] y = new T2[x.Length];
            for (int i = 0; i < x.Length; i++)
            {
                y[i] = f(x[i]);
            }
            return y;
        }
```

```
    public static implicit operator Vectorize<T1, T2>(Func<T1, T2> f)
    {
        return new Vectorize<T1, T2>(f);
    }
    public static implicit operator Func<T1[], T2[]>(Vectorize<T1, T2> v)
    {
        return v.F;
    }
}

public class Vectorize<T>
{
    Func<T, T, T> f;
    public Vectorize(Func<T, T, T> f)
    { this.f = f; }
    public T[,] F(T[] x, T[] y)
    {
        T[,] V = new T[y.Length, x.Length];
        for (int i = 0; i < x.Length; i++)
            for (int j = 0; j < y.Length; j++)
            {
                V[j, i] = f(x[i], y[j]);
            }
        return V;
    }
    public static implicit operator Vectorize<T>(Func<T, T, T> f)
    {
        return new Vectorize<T>(f);
    }
    public static implicit operator Func<T[], T[], T[,]>(Vectorize<T> v)
    {
```

```
            return v.F;
        }
    }
}
```