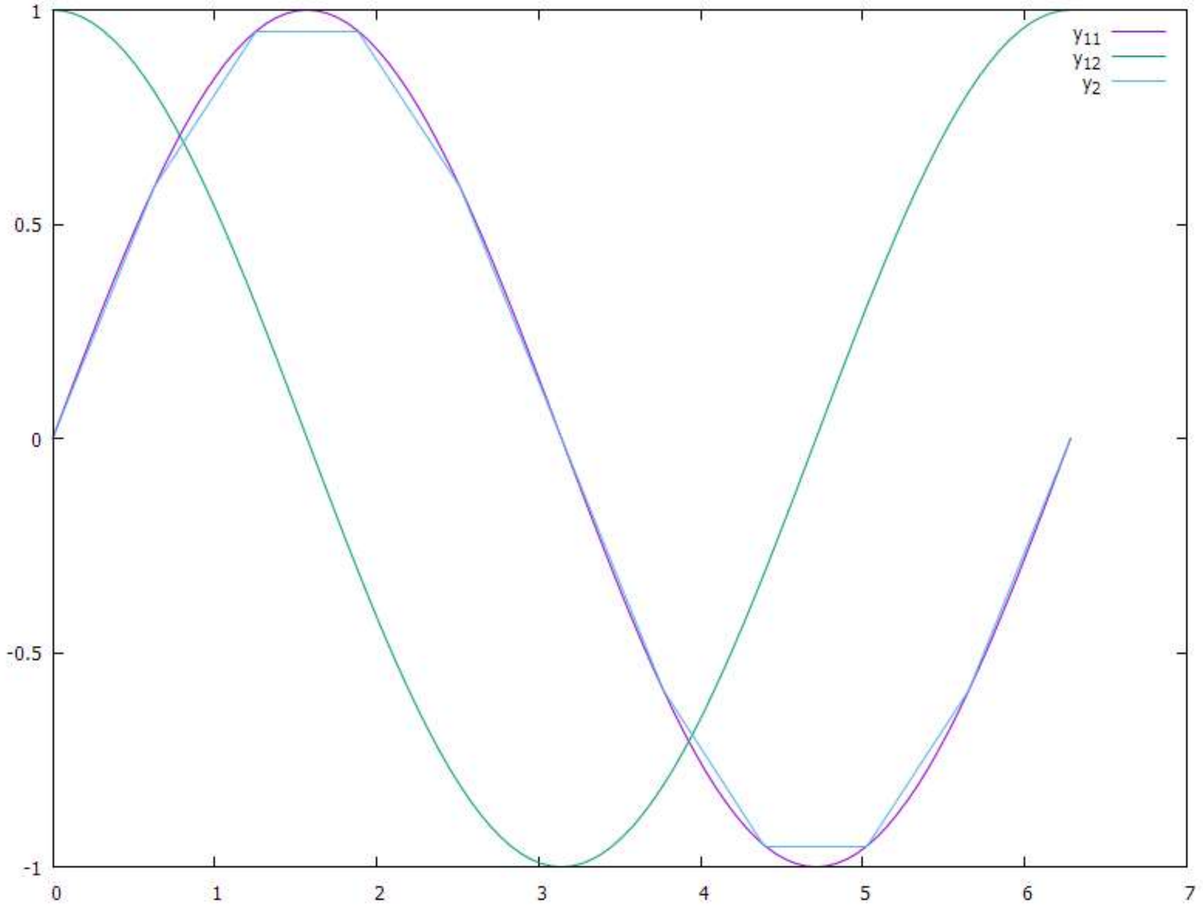
```
using System.Numerics;
using Real = System.Double;
//using Real = System.Single;
using static System.Math;
using static System.Numerics.Complex;
using static MMP.Tools;
using MMP;

namespace L2
{
    class Program
    {
        static Real[] r = { 1, 0.1, 0.3 };
        static int[] omega = { -1, 1, 2 };
        static Complex dft(Real t)
        {
            Complex z = Zero;
            Complex j = ImaginaryOne;
            int n = r.Length;
            for (int i = 0; i < n; i++)
            {
                z += r[i] * Exp(j * omega[i] * t);
            }
            return z;
        }
        static Real Real(Complex z)
        {
            return (Real)(z.Real);
        }
        static Real Imaginary(Complex z)
        {
            return (Real)(z.Imaginary);
        }

        static void test1()
        {
            Real[] t = LinSpace(0, (Real)(2 * PI), 1000);
            Complex[] cft = Apply(dft, t);
            Real[] x = Apply(Real, cft);
            Real[] y = Apply(Imaginary, cft);

            Table<Real> tab = new Table<Real>(x, y);
            tab.AddNames("x", "y");

            GNUPlot<Real> gplot = new GNUPlot<Real>(tab);

            gplot.Plot();

        }

        static void test2()
        {
            Real[] x1 = LinSpace(0, (Real)(2 * PI), 100);
            Real[] y11 = Apply(Sin, x1);
            Real[] y12 = Apply(Cos, x1);

            Table<Real> tab1 = new Table<Real>(x1, y11,y12);
            tab1.AddNames("x_1", @"y_{11}",@"y_{12}");

            Real[] x2 = LinSpace(0, (Real)(2 * PI), 10);
            Real[] y2 = Apply(Sin, x2);
            Table<Real> tab2 = new Table<Real>(x2, y2);
            tab2.AddNames("x_2", "y_2");

            GNUPlot<Real> gplot = new GNUPlot<Real>(tab1,tab2);

            gplot.Plot();

        }


        static void Main(string[] args)
        {
            //test1();
            test2();
        }
    }
}
```



y_{11}
y_{12}
y_2
1
0.5
0
-0.5
-1
0
1
2
3
4
5
6
7

```
using System;

namespace MMP
{
    using Real = System.Double;
    static partial class Tools
    {
        public static Real[] LinSpace(Real a, Real b, int n = 100)
        {
            Real[] t = new Real[n + 1];
            Real dt = (b - a) / n;
            for (int i = 0; i < n; i++)
            {
                t[i] = a;
                a += dt;
            }
            t[n] = b;
            return t;
        }
    }
}

namespace MMP.Single
{
    using Real = System.Single;
    static class Tools
    {
        public static Real[] LinSpace(Real a, Real b, int n = 100)
        {
            Real[] t = new Real[n + 1];
            Real dt = (b - a) / n;
            for (int i = 0; i < n; i++)
            {
                t[i] = a;
                a += dt;
            }
            t[n] = b;
            return t;
        }
    }
}

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

```
using System.Collections.Generic;
using System.IO;
using static System.Diagnostics.Process;


namespace MMP
{
    public struct Terminal
    {
        uint width, height;
        public Terminal(uint w, uint h)
        {
            width = w; height = h;
        }
        public string ToGnuplot()
        {
            return string.Format("set terminal wxt size {0}, {1}", width, height);
        }
    }
    public class GNUPlot<T>
    {
        List<Table<T>> tables;
        Terminal terminal = new Terminal(800,600);
        public GNUPlot(params Table<T>[] tables)
        {
            this.tables = new List<Table<T>>();
            for (int i = 0; i < tables.Length; i++)
                if (!tables[i].Jagged && tables[i].Columnwise)
                    this.tables.Add(tables[i]);

        }
        void GnuplotScript(string FileName)
        {
            using (StreamWriter output = new StreamWriter(FileName + ".gpl"))
            {
                output.WriteLine(terminal.ToGnuplot());
                //output.WriteLine("set yrange [-0.5:1.1]");
                output.Write("plot ");

                for (int itab = 0; itab < tables.Count; itab++)
                {
                    var tab = tables[itab];
                    for (int i = 1; i < tab.Length; i++)
                    {
                        output.Write(" '{2}' i {3} u 1:{0}  w l t '{1}'", i + 1, tab.GetName(i),
FileName,itab);
                        if (itab < tables.Count-1 || i < tab.Length - 1)
                            output.Write(@", \");
                        output.WriteLine();
                    }
                }
                output.WriteLine("pause 3");
            }
        }

        public void Plot()
        {
            string s = "tab";

            using (StreamWriter output = new StreamWriter(s))
            {
                for (int itab = 0; itab < tables.Count; itab++)
                {
                    var tab = tables[itab];
                    for (int irow = 0; irow < tab[0].Length; irow++)
                    {
                        for (int icol = 0; icol < tab.Length; icol++)
                            output.Write(tab.Format, tab[icol][irow]);
                        output.WriteLine();
                    }
                    if (itab < tables.Count - 1)
                    {
                        output.WriteLine(); output.WriteLine();
                    }

                }
            }


            GnuplotScript(s);
            Start(@"c:\Program Files (x86)\gnuplot\bin\gnuplot.exe", s + ".gpl");

            /*
            ProcessStartInfo pinfo = new ProcessStartInfo(@"c:\Program Files
(x86)\gnuplot\bin\gnuplot.exe", "s.gp");
            pinfo.WorkingDirectory = Environment.CurrentDirectory;
            pinfo.UseShellExecute = true;
            Process gp = new Process();
            gp.StartInfo = pinfo;
            bool b=gp.Start();
            gp.WaitForExit();
            */
```

```
                    }

          }
}
```

```csharp
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;

namespace MMP
{
    public class Table<T>
    {
        protected List<T[]> data;
        protected List<string> columnNames;
        protected string format="{0} ";
        public Table(params T[][] columns)
        {
            if (columns == null)
                throw new NullReferenceException();
            length = columns.Length;
            jagged = false;
            for (int i = 0; i < length; i++)
                if (columns[i] == null)
                    throw new NullReferenceException();
            data = new List<T[]>();
            columnNames = new List<string>();
            data.AddRange(columns);
            for (int i = 0; i < length; i++)
                if (data[i].Length != data[0].Length)
                {
                    jagged = true;
                    break;
                }
            Columnwise = true;
```

```
}

protected int length;
protected bool jagged;

public int Length { get { return length; } }
public bool Jagged { get { return jagged; } }

public bool Columnwise { get; set; }
public T[] this[int i] { get { return data[i]; } }

public string Format { get { return format; } }

public string GetName(int i) {  return columnNames[i];  }

public void AddNames(params string[] names)
{
    columnNames.AddRange(names);
}

public override string ToString()
{
    StringBuilder sb = new StringBuilder();
    if (Columnwise)
        if (jagged)
            sb.AppendLine("Columns must have equal length!");
        else
            for (int irow = 0; irow < data[0].Length; irow++)
            {
                for (int icol = 0; icol < length; icol++)
                    sb.AppendFormat(format, data[icol][irow]);
                sb.AppendLine();
            }
    else
        for (int icol = 0; icol < length; icol++)
        {
            for (int irow = 0; irow < data[icol].Length; irow++)
                sb.AppendFormat(format, data[icol][irow]);
            sb.AppendLine();
        }
    return sb.ToString();
}

public void Save(string FileName)
{
    using (StreamWriter output = new StreamWriter(FileName))
    {
        if (Columnwise)
            if (jagged)
                output.WriteLine("Columns must have equal length!");
            else
                for (int irow = 0; irow < data[0].Length; irow++)
                {
                    for (int icol = 0; icol < length; icol++)
                        output.Write(format, data[icol][irow]);
                    output.WriteLine();
                }
        else
            for (int icol = 0; icol < length; icol++)
            {
                for (int irow = 0; irow < data[icol].Length; irow++)
                    output.Write(format, data[icol][irow]);
                    output.WriteLine();
                }
        }
        //WriteAllText(FileName, ToString());
    }

}
}
```