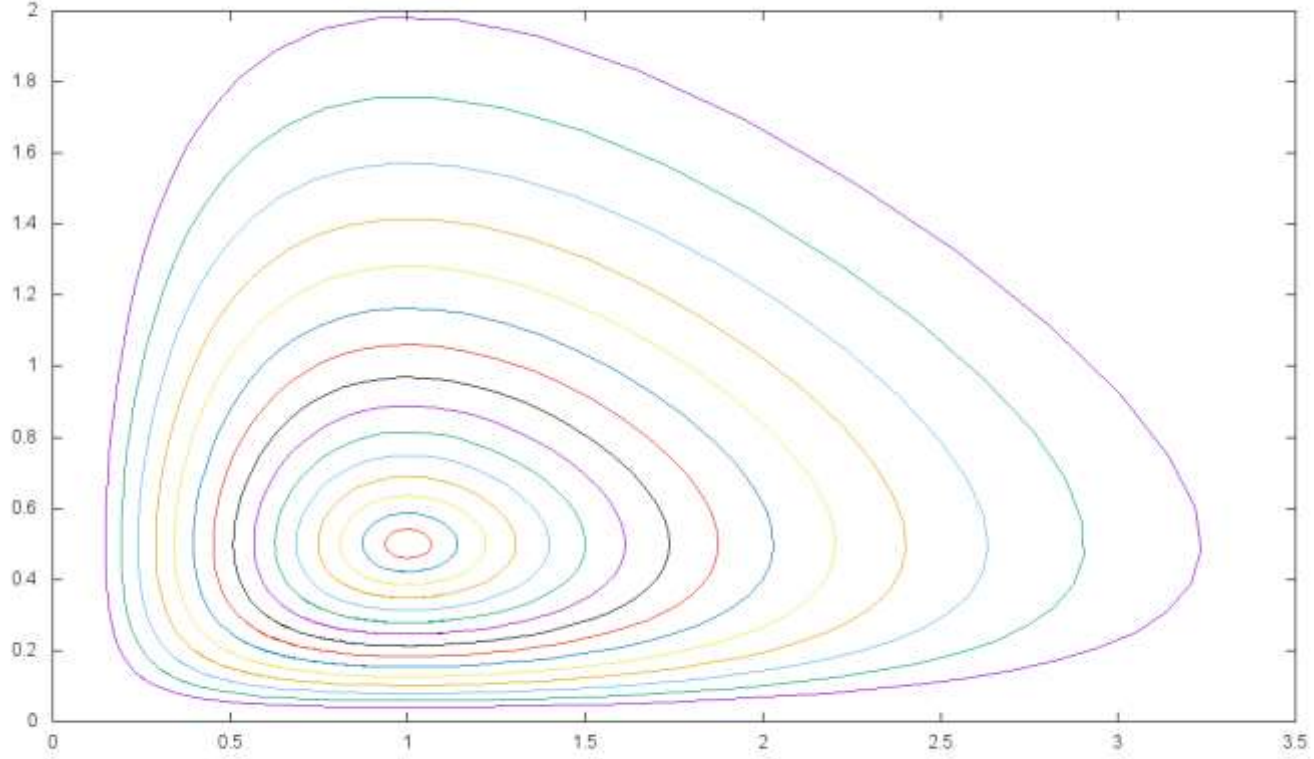
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5

```
using System.IO;
using rat = MMP.RationalNumbers.Rational;
using rpoly = MMP.Tools.Rational.Polynomial;
using static MMP.Tools.Rational.Polynomial;
using static MMP.Tools.General;
using static System.Console;
using static System.Math;
using vec = MathNet.Numerics.LinearAlgebra.Vector<double>;
using mat = MathNet.Numerics.LinearAlgebra.Matrix<double>;
using MathNet.Numerics.LinearAlgebra.Double;
using static MathNet.Numerics.LinearAlgebra.Double.DenseVector;
using static MMP.Tools.GnuPlot;
using MMP.Tools;
using System;

namespace MMP
{

    public static class ODE
    {
        public static void CodeGeneration()
        {
            int kmax = 12;
            var AB = new rat[kmax - 1][];
            var AM = new rat[kmax - 1][];
            var BDF = new rat[kmax - 1][];
            for (int k = 1; k < kmax; k++)
            {
                AB[k - 1] = new rat[k];
                for (int i = 0; i < k; i++)
```

```
{
    rpoly la = Monom(0);
    for (int j = 0; j < k; j++)
        if (i != j)
            la *= LinearFactor(-j);
    la = la / la.Eval(-i);
    AB[k - 1][i] = Integral(la).Eval(1);
}
AM[k - 1] = new rat[k];
for (int i = 0; i < k; i++)
{
    rpoly la = Monom(0);
    for (int j = 0; j < k; j++)
        if (i != j)
            la *= LinearFactor(1 - j);
    la = la / la.Eval(1 - i);
    AM[k - 1][i] = Integral(la).Eval(1);
}
BDF[k - 1] = new rat[k + 1];
for (int i = 0; i < k + 1; i++)
{
    rpoly la = Monom(0);
    for (int j = 0; j < k + 1; j++)
        if (i != j)
            la *= LinearFactor(1 - j);
    la = la / la.Eval(1 - i);
    if (i == 0)
    {
        BDF[k - 1][0] = 1 / Derivative(la).Eval(1);
    }
    else
```

```
                BDF[k - 1][i] = Derivative(la).Eval(1) * BDF[k - 1][0];
            }
        }

        using (StreamWriter outputFile = new StreamWriter("Adams.cs"))
        {
            outputFile.WriteLine("using System;");
            outputFile.WriteLine();
            outputFile.WriteLine("public class Adams");
            outputFile.WriteLine("{");
            outputFile.WriteLine(ToCode<rat, double>("Bashforth", AB));
            outputFile.WriteLine(ToCode<rat, double>("Moulton", AM));
            outputFile.WriteLine("}");
        }

        using (StreamWriter outputFile = new StreamWriter("BDF.cs"))
        {
            outputFile.WriteLine("using System;");
            outputFile.WriteLine();
            outputFile.WriteLine("public class BDF");
            outputFile.WriteLine("{");
            outputFile.WriteLine(ToCode<rat, double>("Table", BDF));
            outputFile.WriteLine("}");
        }

    }

    public static vec f(double t, vec y)
    {
        return new DenseVector(new double[] { y[1], -y[0] });
    }
```

```
public static void sys()
{
    vec y0 = new DenseVector(2);
    y0[0] = 1;
    Adams am = new Adams(f, y0, 0, PI / 100, 6);
    vec t;
    mat Y;
    am.Solution(PI / 2, out t, out Y);
    WriteLine(t);
    WriteLine(Y.Transpose().ToString("G"));
    WriteLine(t.Map(x => Cos(x)).ToString("G"));
}

public static void single()
{
    vec y0 = new DenseVector(1);
    y0[0] = 1;
    Adams am = new Adams((x, y) => y, y0, 0, 0.05, 6);
    vec t;
    mat Y;
    am.Solution(1, out t, out Y);
    WriteLine(t);
    WriteLine(Y.Transpose().ToString("G"));
    WriteLine(t.Map(x => Exp(x)).ToString("G"));
}

public static void LotkaVolterra()
{
    vec v = OfArray(new double[] { 0.05, 0.025 });
    double a = 2f / 3, b = 4f / 3, c = 1, d = 1;
```

```
        Func<double, vec, vec> f = (t, y) =>
        OfArray(new double[] {
            a * y[0] - b * y[0] * y[1],
            d * y[0] * y[1] - c * y[1] });
        Adams am;
        vec ts;
        int n = 15;
        mat[] Y = new mat[n]; ;
        for (int i = 0; i < n; i++)
        {
            vec y0 = (i + 5) * v;
            am = new Adams(f, y0, 0, 0.1, 6);
            am.Solution(9.5, out ts, out Y[i]);
            Y[i] = Y[i].Transpose();
        }
        Plot(Y);
        WaitForKey();
    }

    static Terminal win = new Terminal(1200, 800);
    static Terminal png = new Terminal(1200, 800, Terminal.Enum.png);
    static void Main()
    {
        //CodeGeneration();

        Set = "size ratio - 1";
        Terminals = new Terminal[] { win, png };
        LotkaVolterra();
    }
}
```

```
}


using System;
using System.Collections.Generic;
using static System.Math;
using MathNet.Numerics.LinearAlgebra.Double;
using static MathNet.Numerics.LinearAlgebra.Double.DenseMatrix;
using static MathNet.Numerics.LinearAlgebra.Double.DenseVector;
using vec = MathNet.Numerics.LinearAlgebra.Vector<double>;
using mat = MathNet.Numerics.LinearAlgebra.Matrix<double>;

public class Adams
{
    public static Double[][] Bashforth =
{
      new Double[] { 1 },
      new Double[] { 3.0/2, -1.0/2 },
      new Double[] { 23.0/12, -4.0/3, 5.0/12 },
      new Double[] { 55.0/24, -59.0/24, 37.0/24, -3.0/8 },
      new Double[] { 1901.0/720, -1387.0/360, 109.0/30, -637.0/360, 251.0/720 },
      new Double[] { 4277.0/1440, -2641.0/480, 4991.0/720, -3649.0/720, 959.0/480, -95.0/288 },
      new Double[] { 198721.0/60480, -18637.0/2520, 235183.0/20160, -10754.0/945, 135713.0/20160, -
5603.0/2520, 19087.0/60480 },
      new Double[] { 16083.0/4480, -1152169.0/120960, 242653.0/13440, -296053.0/13440,
2102243.0/120960, -115747.0/13440, 32863.0/13440, -5257.0/17280 },
      new Double[] { 14097247.0/3628800, -21562603.0/1814400, 47738393.0/1814400, -
69927631.0/1814400, 862303.0/22680, -45586321.0/1814400, 19416743.0/1814400, -4832053.0/1814400,
1070017.0/3628800 },
```

```
      new Double[] { 4325321.0/1036800, -104995189.0/7257600, 6648317.0/181440, -28416361.0/453600,
269181919.0/3628800, -222386081.0/3628800, 15788639.0/453600, -2357683.0/181440, 20884811.0/7257600,
-25713.0/89600 },
      new Double[] { 2132509567.0/479001600, -2067948781.0/119750400, 1572737587.0/31933440, -
1921376209.0/19958400, 3539798831.0/26611200, -82260679.0/623700, 2492064913.0/26611200, -
186080291.0/3991680, 2472634817.0/159667200, -52841941.0/17107200, 26842253.0/95800320 }
   };

    public static Double[][] Moulton =
    {
      new Double[] { 1 },
      new Double[] { 1.0/2, 1.0/2 },
      new Double[] { 5.0/12, 2.0/3, -1.0/12 },
      new Double[] { 3.0/8, 19.0/24, -5.0/24, 1.0/24 },
      new Double[] { 251.0/720, 323.0/360, -11.0/30, 53.0/360, -19.0/720 },
      new Double[] { 95.0/288, 1427.0/1440, -133.0/240, 241.0/720, -173.0/1440, 3.0/160 },
      new Double[] { 19087.0/60480, 2713.0/2520, -15487.0/20160, 586.0/945, -6737.0/20160,
263.0/2520, -863.0/60480 },
      new Double[] { 5257.0/17280, 139849.0/120960, -4511.0/4480, 123133.0/120960, -88547.0/120960,
1537.0/4480, -11351.0/120960, 275.0/24192 },
      new Double[] { 1070017.0/3628800, 2233547.0/1814400, -2302297.0/1814400, 2797679.0/1814400, -
31457.0/22680, 1573169.0/1814400, -645607.0/1814400, 156437.0/1814400, -33953.0/3628800 },
      new Double[] { 25713.0/89600, 9449717.0/7257600, -1408913.0/907200, 200029.0/90720, -
8641823.0/3628800, 6755041.0/3628800, -462127.0/453600, 335983.0/907200, -116687.0/1451520,
8183.0/1036800 },
      new Double[] { 26842253.0/95800320, 164046413.0/119750400, -296725183.0/159667200,
12051709.0/3991680, -33765029.0/8870400, 2227571.0/623700, -21677723.0/8870400, 23643791.0/19958400,
-12318413.0/31933440, 9071219.0/119750400, -3250433.0/479001600 }
   };

    Func<double, vec, vec> f;
```

```
int neq;
Queue<vec> Qf;
List<vec> F;
List<vec> Y;
List<double> T;
vec y;

int order;
double tau;
vec fv;

public Adams(Func<double, vec, vec> f, vec y0, double t0, double tau, int order = 11)
{
    if (order > Bashforth.Length + 1)
        throw new IndexOutOfRangeException();

    this.f = f;
    neq = y0.Count;

    F = new List<vec>();
    Y = new List<vec>();
    T = new List<double>();
    double t = t0;
    y = y0;

    Y.Add(y0);
    fv = f(t, y0);
    F.Add(fv);
    T.Add(t0);

    int n = (int)Ceiling(1 / tau);
```

```
        this.tau = Pow(tau, order);

        for (int k = 1; k < order; k++)
        {
            this.order = k;
            Qf = new Queue<vec>(F);
            F.RemoveRange(1, F.Count - 1);
            for (int ik = 0; ik < k; ik++)
            {
                int nj;
                if (ik == 0)
                    nj = n - k + 1;
                else
                    nj = n;
                for (int j = 0; j < nj; j++)
                    Step(ref t, ref y, ref fv);
                F.Add(fv);
                if (k == order - 1)
                {
                    Y.Add(y);
                    T.Add(t);
                }
            }
            this.tau *= n;
        }
        this.order = order;
        Qf = new Queue<vec>(F);
        //this.tau = tau;
    }

    public void Step(ref double t, ref vec y, ref vec fv)
```

```
    {
        vec v = new DenseVector(neq);
        int i = order - 1;
        foreach (var f in Qf)
        {
            v += f * Bashforth[order - 1][i--];
        }
        y += v * tau;
        t += tau;
        fv = f(t, y);
        Qf.Dequeue();
        Qf.Enqueue(fv);
    }

    public void Solution(double T, out vec t_all, out mat Y_all)
    {
        double t = this.T[this.T.Count - 1];
        vec y = Y[Y.Count - 1];
        vec fv = null;
        while (t < T)
        {
            Step(ref t, ref y, ref fv);
            this.T.Add(t);
            this.Y.Add(y);
        }
        t_all = OfEnumerable(this.T);
        Y_all = OfColumnVectors(Y.ToArray());
    }

}
```

```
public static void Plot<T>(params Matrix<T>[] Y)
            where T : struct , IEquatable<T>, IFormattable
        {
            GnuPlot.Write2(Set);
            using (StreamWriter outputFile = new StreamWriter("data"))
            {
                for (int i = 0; i < Y.Length; i++)
                {
                    DelimitedWriter.Write(outputFile, Y[i]);
                    outputFile.WriteLine();
                    outputFile.WriteLine();
                    outputFile.WriteLine();
                }

            }

            foreach (var terminal in Terminals)
            {
                GnuPlot.WriteLine2(terminal.ToString());
                string s1, s2;
                for (int i = 0; i < Y.Length; i++)
                {
                    if (i == 0)
                        s1 = "plot ";
                    else
                        s1= "";
                    if (i < Y.Length-1)
                        s2 = @", \";
                    else
                        s2 = "";
```

```
            GnuPlot.WriteLine2($"{s1}'data' i {i} u 1:2 w l t ''{s2}");
        }
        GnupStWr.Flush();
    }
}
```