```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;
using System.Drawing;

namespace ConsoleApplication7
{
    class Program
    {
        double RHS(double x)
        {
            return -16;
        }

////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
        // Basis Functions

////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
        double pwl(double x, int I, int nn, double[] xn)
        {
            double rs = 0;
            if (I == 0)
            {
                if ((x >= xn[I]) && (x <= xn[I + 1]))
                {
                    rs = 1 * ((x - xn[I + 1]) / (xn[I] - xn[I + 1]));
                }
                else
                    rs = 0;
            }
            else if (I == nn - 1)
            {
                if ((x <= xn[I]) && (x >= xn[I - 1]))
                {
                    rs = 1 * ((x - xn[I - 1]) / (xn[I] - xn[I - 1]));
                }
                else
                    rs = 0;
            }
            else
            {
                if ((x <= xn[I]) && (x >= xn[I - 1]))
                {
                    rs = 1 * ((x - xn[I - 1]) / (xn[I] - xn[I - 1]));
                }
                else if ((x >= xn[I]) && (x <= xn[I + 1]))
                {
                    rs = 1 * ((x - xn[I + 1]) / (xn[I] - xn[I + 1]));
                }
                else
                    rs = 0;
            }
```

```
            return rs;
        }

////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
        // Basis Derivative

////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
        double pwld(double x, int I, int nn, double[] xn)
        {
            double rs = 0;
            if (I == 0)
            {
                if ((x >= xn[I]) && (x <= xn[I + 1]))
                {
                    rs = (1 / (xn[I] - xn[I + 1]));
                }
                else
                    rs = 0;
            }
            else if (I == nn - 1)
            {
                if ((x <= xn[I]) && (x >= xn[I - 1]))
                {
                    rs = (1 / (xn[I] - xn[I - 1]));
                }
                else
                    rs = 0;
            }
            else
            {
                if ((x <= xn[I]) && (x >= xn[I - 1]))
                {
                    rs = (1 / (xn[I] - xn[I - 1]));
                }
                else if ((x >= xn[I]) && (x <= xn[I + 1]))
                {
                    rs = (1 / (xn[I] - xn[I + 1]));
                }
                else
                    rs = 0;
            }
            return rs;
        }

////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
        // Integration

////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
        double[] pointsf(int nq)
        {
            if (nq == 1)
            {
                double[] p = { 0 };
                return p;
```

```
    }
    else if (nq == 2)
    {
        double[] p = { -0.5773502, 0.5773502 };
        return p;
    }
    else if (nq == 3)
    {
        double[] p = { -0.7745966, 0, 0.7745966 };
        return p;
    }
    else
    {
        return null;
    }
}
double[] weightsf(int nq)
{
    if (nq == 1)
    {
        double[] w = { 2 };
        return w;
    }
    else if (nq == 2)
    {
        double[] w = { 1, 1 };
        return w;
    }
    else if (nq == 3)
    {
        double[] w = { 0.555555, 0.88888888, 0.55555555 };
        return w;
    }
    else
    {
        return null;
    }
}
double Integrand(double x,int[] pair,double[] xn)
{
    double tbr;
    if (pair[1] != 0)
    {
        tbr = pwld(x, pair[0], xn.Length, xn) * pwld(x, pair[1], xn.Length, xn);
    }
    else
    {
        tbr = pwl(x, pair[0], xn.Length, xn) * RHS(x);
    }
    return tbr;
}
double[] intervalf(int s, double[] interval)
{
    double step = (interval[1] - interval[0]) / s;
    double[] inte = new double[s + 1];
    for (int i = 0; i < s + 1; i++)
    {
        inte[i] = interval[0] + i * step;
```

```
            }
            return inte;
        }
        double integral(int nq, double[] interval,int[] pair,double[] xn)
        {
            double xp;
            double sum = 0;
            double[] weights = weightsf(nq);
            double[] points = pointsf(nq);
            for (int i = 0; i < nq; i++)
            {
                xp = (interval[0] + interval[1] + (-interval[0] + interval[1]) *
points[i]) / 2;
                sum = sum + weights[i] * Integrand(xp,pair,xn);
            }
            sum = ((-interval[0] + interval[1]) / 2) * sum;
            return sum;
        }
        double CI(int nq, double[] interval, int s,int[] pair,double[] xn)
        {
            double sum = 0;
            double[] intervals = intervalf(s, interval);
            for (int j = 0; j < s; j++)
            {
                double[] ninterval = { intervals[j], intervals[j + 1] };
                double integrals = integral(nq, ninterval,pair,xn);
                sum = sum + integrals;
            }
            return sum;
        }

////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
        // Solver ///

////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
        double[,] SwapRow(double[,] M, int i, int j, int N)
        {
            double[] temp = new double[N];
            for (int z = 0; z < N; z++)
                {
                    temp[z] = M[i, z]; M[i, z] = M[j, z];
                    M[j, z] = temp[z];
                }
            return M;
        }
        double[,] AddRowInstance(double T, int RI, int RA, double[,] M, int N)
        {
            for (int i = 0; i < N; i++)
            {
                M[RI, i] = M[RI, i] + T * M[RA, i];
            } return M;
        }
        double[] FB(double[,] M, double[] F, int N)
        {
            // Forward Elimination
            for (int i = 0; i < N - 1; i++)
```

```
            {
                double B = M[0, i]; int k = i;
                for (int j = i; j < N; j++)
                {
                    if (B < M[j, i])
                    {
                        B = M[j, i]; k = j;
                    }
                }
                if (i != k)
                {
                    M = SwapRow(M, i, k, N);
                }
                for (int z = i + 1; z < N; z++)
                {
                    double T = -M[z, i] / M[i, i]; M = AddRowInstance(T, z, i, M, N);
                    F[z] = F[z] + T * F[i];
                }
            }
            // Backward Substitution
            double[] C = new double[N];
            for (int i = N - 1; i >= 0; i--)
            {
                C[i] = F[i];
                for (int j = i + 1; j < N; j++)
                {
                    C[i] = C[i] - M[i, j] * C[j];
                }
                C[i] = C[i] / M[i, i];
            }
            return C;
        }
        double Approximate(double x, double[] c,double[] xn)
        {
            int size = c.Length;
            double sum = 0;
            for (int i = 0; i < size; i++ )
            {
                sum = sum + c[i] * pwl(x, i, size, xn);
            }
            return sum;
        }
        double True(double x)
        {
            return 8 * (x * x) - 10 * x + 3;
        }

////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

        static void Main(string[] args)
        {
```

```
////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
            // Building Nodes ////

////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
            Program p = new Program();
            System.IO.FileStream osteam;
            System.IO.StreamWriter writer;
            System.IO.TextWriter oldOut = Console.Out;
            try
            {
                osteam = new System.IO.FileStream("./AssemblyOutput.txt",
System.IO.FileMode.OpenOrCreate, System.IO.FileAccess.Write);
                writer = new System.IO.StreamWriter(osteam);
            }
            catch (Exception e)
            {
                Console.WriteLine("Cannot open Redirect.txt for writing");
                Console.WriteLine(e.Message);
                return;
            }
            Console.SetOut(writer);
            // First and Last Message
            DateTime now = DateTime.Now;
            string[] firstmessage = { "FEM1", "C sharp version", "One dimensional finite
element code" };
            foreach (string line in firstmessage)
                Console.WriteLine(line);
            Console.WriteLine();
            string[] lastmessage = { "FEM1", "Normal end of exccution" };
            // Prblem Parameters
            int EN = 4;
            int EO = 1;
            double XL = 0.0;
            double XR = 1.0;
            double UL = 3.0;
            double UR = 1.0;
            int NN = EN * EO + 1;
            Console.WriteLine("Number of Elements [EN] = \t{0,8:N}", EN);
            Console.WriteLine("Order of each element [ED] = \t{0,8:N}", EO);
            Console.WriteLine("Left Endpoint [XL] = \t{0,8:N}", XL);
            Console.WriteLine("Right Endpoint [XR] = \t{0,8:N}", XR);
            Console.WriteLine("Number of Nodes [NN] = \t{0,8:N}", NN);
            Console.WriteLine();
            // Node Cordinates
            double[] XN = new double[NN];
            Console.WriteLine("XN array:");
            for (int i = 0; i < NN; i++)
            {
                if (i == 0)
                {
                    XN[i] = XL;
                }
                else if (i == NN - 1)
                {
                    XN[i] = XR;
```

```
    }
    else
    {
        XN[i] = XL + i * ((XR - XL) / NN);
    }
    Console.WriteLine("{0,8:N} {1,8:N}", i, XN[i]);
    Console.WriteLine();
}
int[,] EC = new int[EN, EO + 1];
Console.WriteLine("EC array:");
for (int j = 0; j < EN; j++)
{
    for (int k = 0; k < EO + 1; k++)
    {
        EC[j, k] = j + k;
    }
    Console.WriteLine("{0,8:N} {1,8:N} {2,8:N}", j, EC[j, 0], EC[j, 1]);
}
/// Building linear system
double[,] A = new double[NN,NN];
double[] f = new double[NN];
double[] interval = {XL, XR};
for (int i = 0; i < NN; i++)
{
    for (int j = 0; j < NN; j++)
    {
        if ((i == 0) | (i == NN-1))
        {
            if ((j == 0) & (i == 0))
            {
                A[i, j] = 1;
            }
            else if ((j == NN-1) & (i == NN-1))
            {
                A[i, j] = 1;
            }
            else
            {
                A[i, j] = 0;
            }
        }
        else
        {
            if (j == i - 1 | j == i | j == i + 1)
            {
                int[] pair = { i, j };
                A[i, j] = p.CI(3, interval, 5, pair, XN);
            }
            else
                A[i, j] = 0;
        }
    }
}
Console.WriteLine("The Matrix A of the System");
for (int a = 0; a < NN; a++)
{
    for (int j = 0; j < NN; j++)
    {
```

```
            Console.Write(A[a, j]);
            Console.Write(" ");
        }
        Console.WriteLine();
    }
    Console.WriteLine();
    Console.WriteLine("The f Vector of the System");
    Console.WriteLine();
    for (int c = 0; c < NN; c++)
    {
        if ( c == 0)
        {
            f[c] = UL;
        }
        else if (c ==NN - 1)
        {
            f[c] = UR;
        }
        else
        {
            int[] pair = { c, 0 };
            f[c] = p.CI(3, interval, 5, pair, XN);
        }
        Console.WriteLine(f[c]);
    }
    double[] C = p.FB(A, f, 5);
    double[] x = new double[41];
    double[] Approximation = new double[41];
    double[] True = new double[41];
    for (int j = 0; j < 41; j++)
    {
        x[j] = ((XR - XL)/40) * j;
        Approximation[j] = p.Approximate(x[j], C, XN);
        True[j] = p.True(x[j]);
    }
    var chart = new Chart();
    chart.Size = new Size(600, 250);

    var chartArea = new ChartArea();
    chartArea.AxisX.MajorGrid.LineColor = Color.LightGray;
    chartArea.AxisY.MajorGrid.LineColor = Color.LightGray;
    chartArea.AxisX.LabelStyle.Font = new Font("Consolas", 8);
    chartArea.AxisY.LabelStyle.Font = new Font("Consolas", 8);
    chartArea.AxisX.Interval = 0.1;
    chartArea.AxisX.IntervalOffset = 0;
    chartArea.AxisX.Minimum = 0;
    chartArea.AxisX.Maximum = 1;
    chart.ChartAreas.Add(chartArea);

    var series1 = new Series();
    var series2 = new Series();
    series1.Name = "Approximation";
    series2.Name = "True";
    series1.ChartType = SeriesChartType.FastLine;
    series2.ChartType = SeriesChartType.FastLine;
    chart.Series.Add(series1);
    chart.Series.Add(series2);
    // bind the datapoints
```

```
            chart.Series["Approximation"].Points.DataBindXY(x, Approximation);
            chart.Series["True"].Points.DataBindXY(x, True);
            chart.Legends.Add(new Legend("Legend1"));

            // Set title
            chart.Legends["Legend1"].Title = "My legend";
            // Assign the legend to Series1.
            chart.Series["Approximation"].Legend = "Legend1";
            chart.Series["Approximation"].IsVisibleInLegend = true;
            // draw!
            chart.Invalidate();

            // write out a file
            chart.SaveImage("Solution.png", ChartImageFormat.Png);
            Console.WriteLine("End");
            Console.SetOut(oldOut);
            writer.Close();
            osteam.Close();
        }
    }
}
```