DE Computation practicum V1

Valentin Chernyshov

02.06.2001

IVP solution:

$y' = 1 + 2\frac{y}{x}$, $y(1) = 2, x \in [1, 10]$

$y' - 2\frac{y}{x} = 1$, $x \neq 0$ - linear first order DE

Solve using method "Variation of parameters"

$y = y_1 u(x)$

$y_1' - 2\frac{y_1}{x} = 0$ - complementary equation

$y_1 = e^{\int \frac{2}{x} dx} = x^2$, $y \neq 0$

$y = y_1 u(x) => u' = \frac{f(x)}{y_1(x)} => u = \int \frac{1}{x^2} dx = -\frac{1}{x} + C$, $C \in R$

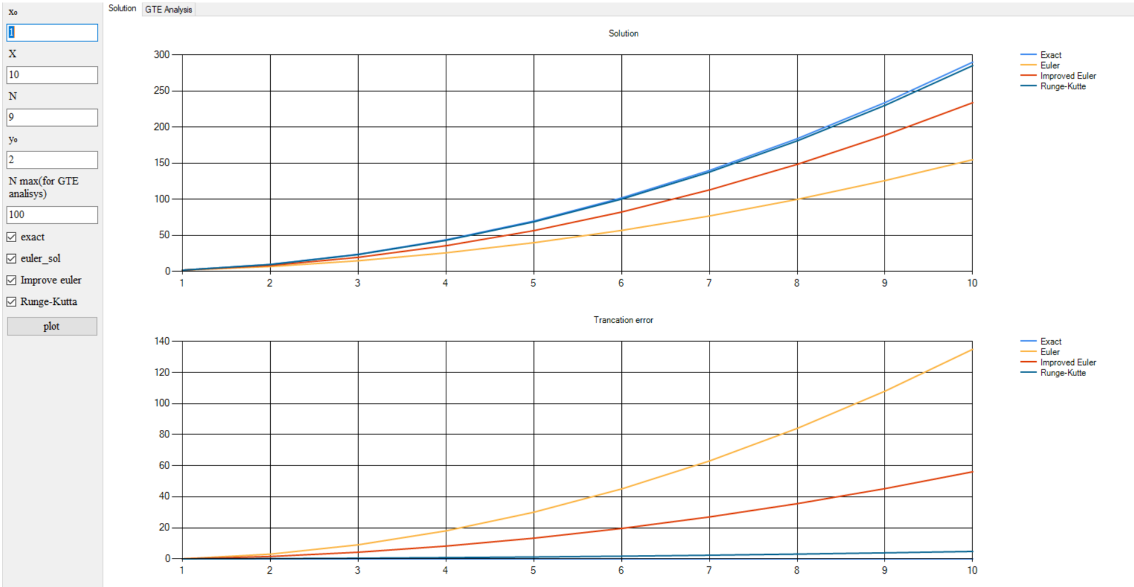
$y = x^2(C - \frac{1}{x}) = x(Cx - 1)$

$y(1) = 2 = C - 1 => C = 3$

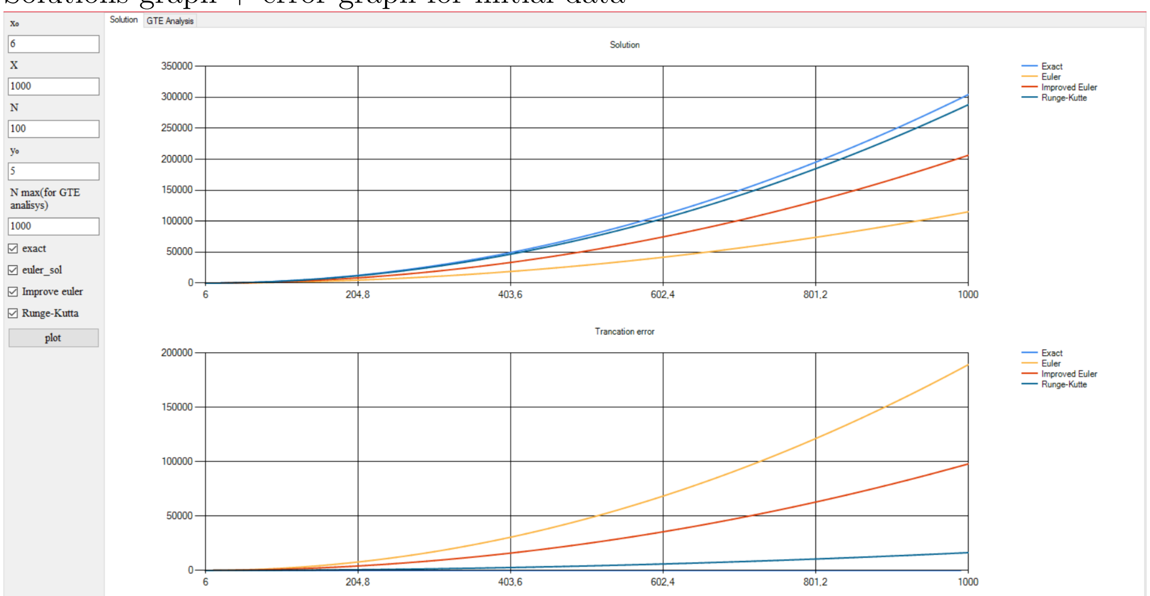
if y = 0:
$0 = 1 + 0 => y \neq 0$

Ans: $y = x(3x - 1), y \neq 0$.

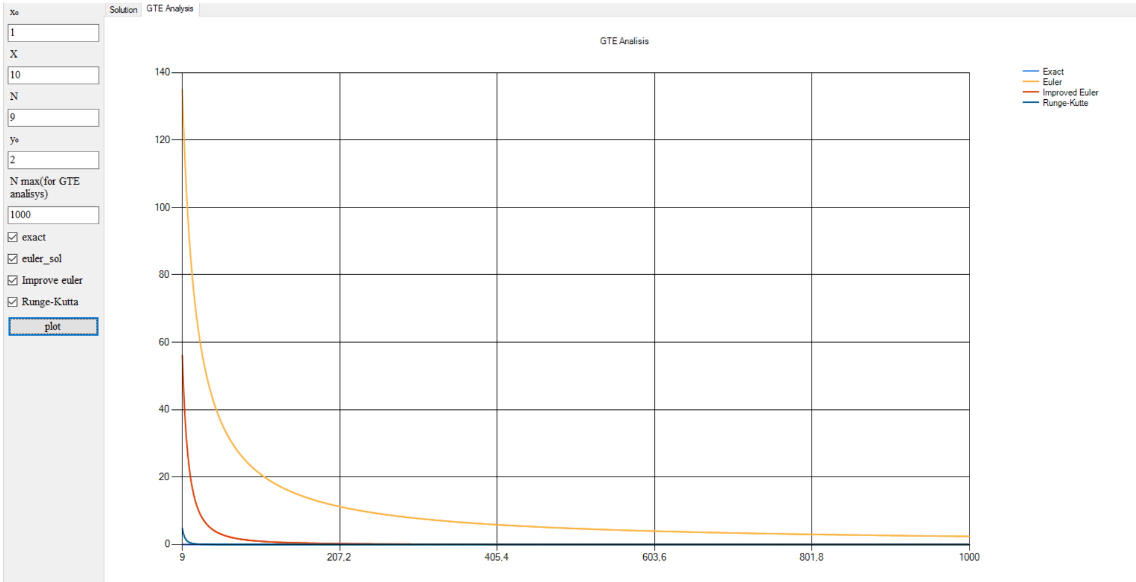
Graphs:



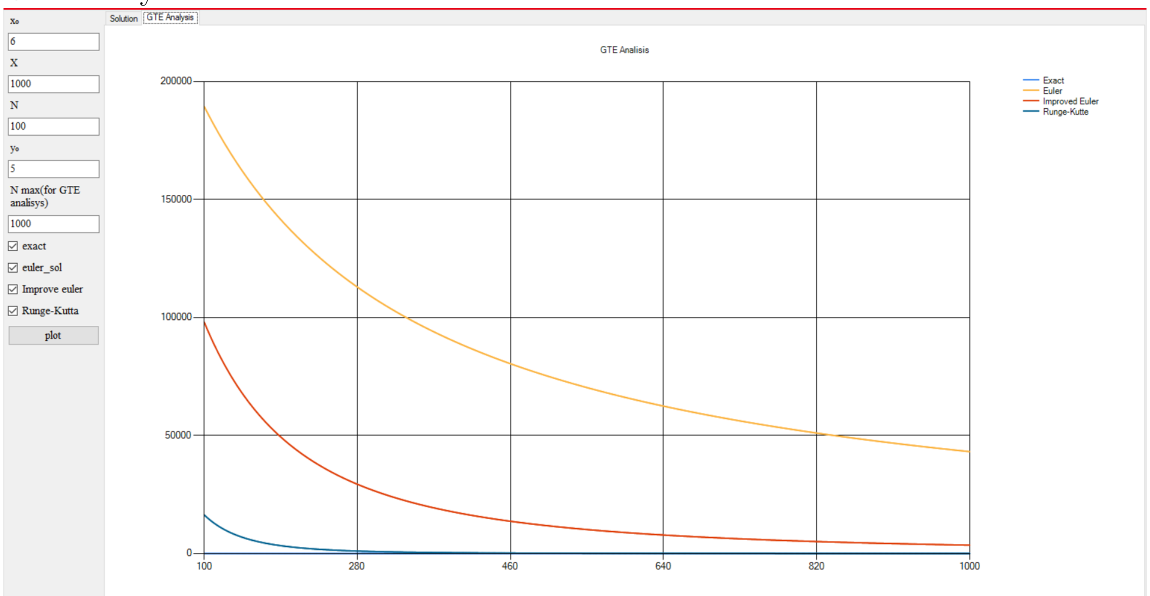
Solutions graph + error graph for initial data



Solutions graph + error graph for changed data

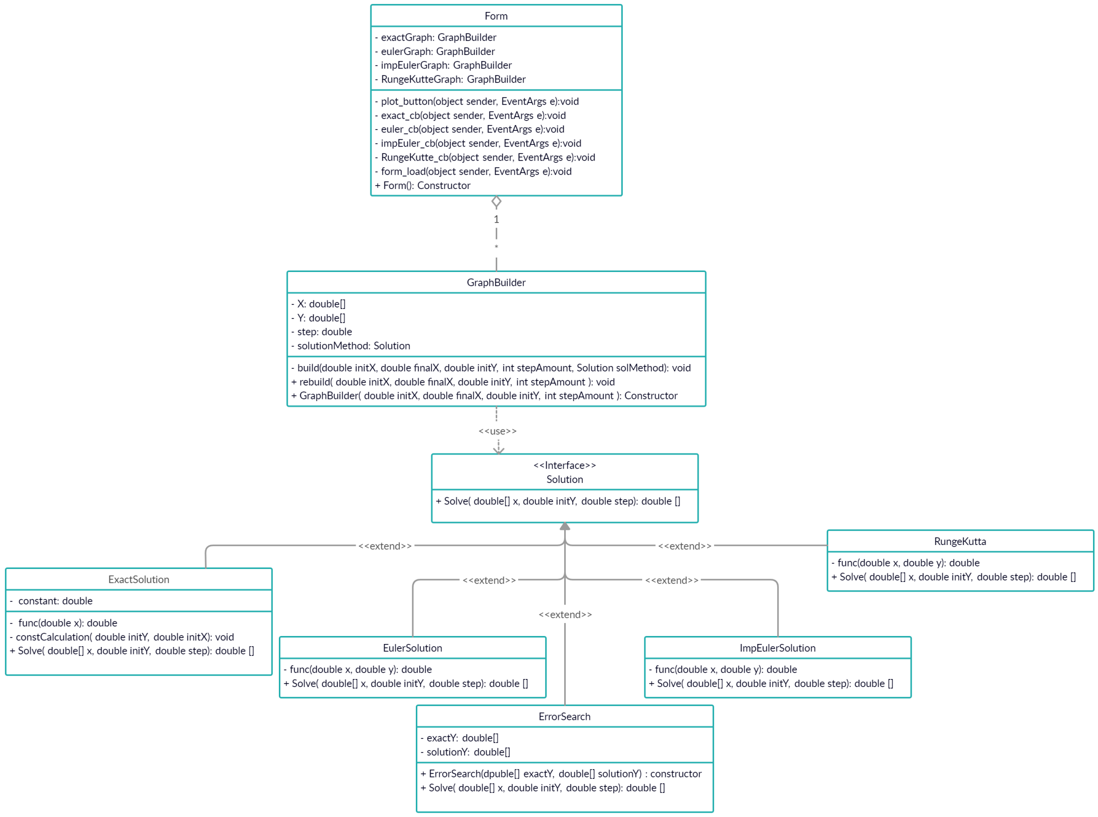


GTE analysis for initial data



GTE analysis for changed data

Code Review:



UML diagram

Mostly all computation happens inside class "GraphBuild". As an arguments it takes all initial values that was inserted in UI as well as "method of a solution". it can be rebuild in any time using "rebuild" method where you also should provide all initial values but without "method of a solution".

"Solution" is an interface that must be implemented by class that was privided to "GraphBuild" as a "method of a solution". Class that implements this interface should implement "Solve" function that returns array of "y" values that correspond to provided array of "x".

ExactSolution, EulerSolution¡ ImpEulerSolution, RungeKutta and ErrorSearch are just specific implementations of "Solution" interface

"Form" class describe whole UI and responsible for all interactions and setting up all data for backend.

```
using System;
using solutions;

namespace DE_comp_pract
{
    public class GraphBuilder
    {
        private double [] _xArray;
        private double [] _yArray;
        private double _step;
        private Solution _solMethod;

        public double[] XArray => _xArray;
        public double[] YArray => _yArray;


        public GraphBuilder(double initialX, double finaleX, double
            initialY, int stepAmount, Solution solMethod){
            Build(initialX, finaleX, initialY, stepAmount, solMethod);
        }

        public GraphBuilder(Solution solMethod) : this(1, 10, 2, 9,
            solMethod){}

        private void Build(double initialX, double finaleX, double
            initialY, int stepAmount, Solution solMethod)
        {
            _xArray = new double[stepAmount + 1];
            _yArray = new double[stepAmount + 1];
            _yArray[0] = initialY;
            _xArray[0] = initialX;
            _solMethod = solMethod;
            if (stepAmount < 0) throw new Exception("N cannot be negative");
            if (stepAmount == 0) throw new Exception("N cannot be equal to
                zero");
            _step = (finaleX - initialX) / stepAmount;
            if(_step <= 0) throw new Exception("Initial value if \"x\"
                cannot be greater than finale one");
            for (int i = 1; i < stepAmount + 1; i++)
            {
                _xArray[i] = _xArray[i - 1] + _step;
            }
            _yArray = _solMethod.Solve(_xArray, initialY, _step);
        }

        public void Rebuild(double initialX, double finaleX, double
            initialY, int stepAmount)
        {
```

```
            Build(initialX, finaleX, initialY, stepAmount, _solMethod);
        }
    }
}
```

Solution interface

```
namespace solutions
{
    public interface Solution
    {
        double[] Solve(double[] xArray, double initY, double step);
    }
}
```

ExactSolution implementation

```
public class ExactSolution : Solution
    {
        private double _constant;

        private double func(double x)
        {
            return x * (_constant * x - 1);
        }

        private void constCalculation(double initX, double initY)
        {
            _constant = (initY + initX) / (initX * initX);
        }

        public double[] Solve(double[] xArray, double initY, double step)
        {
            double[] yArray = new double[xArray.Length];
            constCalculation(xArray[0], initY);
            yArray[0] = initY;
            for (int i = 1; i < xArray.Length; i++)
            {
                yArray[i] = func(xArray[i]);
            }
            return yArray;
        }

    }
```

EulerSolution implementation(other numerical methods implemented in simular way)

```
public class EulerSolution : Solution
    {
```

```
        private double func(double x, double y)
        {
            if(x == 0) throw new Exception("X cannot be equal to zero");
            return 1 + 2 * y / x;
        }

        public double[] Solve(double[] xArray, double initY, double step)
        {
            double[] yArray = new double[xArray.Length];
            yArray[0] = initY;
            for (int i = 1; i < xArray.Length; i++)
            {
                yArray[i] = yArray[i-1] + step * func(xArray[i - 1],
                    yArray[i - 1]);
            }
            return yArray;
        }
    }
```

Example of plotting

```
public class EulerSolution
    {
        private GraphBuilder exactGraph = new GraphBuilder(new
            ExactSolution());

        private void plot(object sender, EventArgs e)
        {
            try
            {
                double initX = Double.Parse(x0_info.Text);
                double finaleX = Double.Parse(X_info.Text);
                double initY = Double.Parse(y0_info.Text);
                int stepAmount = Int32.Parse(N_info.Text);
                int finaleStepAmount = Int32.Parse(N_max_info.Text);
                //plot a solution graph
                exactGraph.Rebuild(initX, finaleX, initY, stepAmount);

                solution_chart.Series[0].Points.DataBindXY(exactGraph.XArray,
                    exactGraph.YArray);

                solution_chart.ChartAreas[0].AxisX.Minimum = initX;
                solution_chart.ChartAreas[0].AxisX.Maximum = finaleX;

                //plot an error graph
                GraphBuilder exactErrGraph = new GraphBuilder(initX,
                    finaleX, initY, stepAmount,
                    new ErrorSearch(exactGraph.YArray, exactGraph.YArray));
```

```
            error_chart.Series[0].Points.DataBindXY(exactErrGraph.XArray,
                exactErrGraph.YArray);

            error_chart.ChartAreas[0].AxisX.Minimum = initX;
            error_chart.ChartAreas[0].AxisX.Maximum = finaleX;
            //plot GTE analysis graph
            double[] GTEXArray = new double[finaleStepAmount -
                stepAmount + 1];
            double[] GTEYExactArray = new double[finaleStepAmount -
                stepAmount + 1];

            for (int i = stepAmount; i <= finaleStepAmount; i++)
            {
                GTEXArray[i - stepAmount] = i;
                exactGraph.Rebuild(initX, finaleX, initY, i);
                GraphBuilder exactGTEGraph = new GraphBuilder(initX,
                    finaleX, initY, i,
                    new ErrorSearch(exactGraph.YArray,
                        exactGraph.YArray));
                GTEYExactArray[i - stepAmount] =
                    exactGTEGraph.YArray.Max();

            }

            GTE_analisis_chart.Series[0].Points.DataBindXY(GTEXArray,
                GTEYExactArray);

            GTE_analisis_chart.ChartAreas[0].AxisX.Minimum = stepAmount;
            GTE_analisis_chart.ChartAreas[0].AxisX.Maximum =
                finaleStepAmount;
        }
        catch(Exception err)
        {
            MessageBox.Show(err.Message);
        }
    }
}
```

P.S. source code and .exe file can be found here: `https://github.com/zZzwat4er/DE_comp`