Differential Equation Computational Assignment Report

Maksudov Rishat

BS17-05

Variant #5 (previous #22)

$$y' = \cos(x) - y$$

$$y(0) = 1$$

$$\mathrm{x} \in [1; 9.5]$$

General information:

In this project we had to implement and use of method, called Euler, Improved Euler and Runge-Kuffa in order to plot the graph of ordinary differential equation. Function is written above. I use Windows forms GUI and C# in implementing this application. Also there are two graphs for Local error with comparison to Analytical solution and Total approximation with dependence number of steps between Initial X and Final X.

$$h = \frac{X - x_0}{2}$$

As it is shown, the smaller h is precisely draw graph. There are some buttons and textboxes used for changing conditions.

Analytical Solution:

$$y' = \cos(x) - y$$

$$y' + y = \cos(x)$$

Method of variation of parameter

$$y' + y = 0$$

$$\int \frac{\mathrm{dy}}{\mathrm{dx}} = -\mathrm{y}$$

$$\int \frac{\mathrm{dy}}{\mathrm{y}} = -\int \mathrm{dx}$$

$$\ln|\mathrm{y}| = -\mathrm{x} + \mathrm{C}$$

$$y = e^{-x} * C_2(x)$$

$$y' = (e^{-x})' * C_2(x) + e^{-x} * C_2{}'(x)$$

$$y' = -e^{-x} * C_2(x) + e^{-x} * C_2{}'(x)$$

$$-e^{-x} * C_2(x) + e^{-x} * C_2'(x) + e^{-x} * C_2(x) = \cos(x)$$

$$e^{-x} * C_2'(x) = \cos(x)$$

$$C_2'(x) = e^{x} * \cos(x)$$

$$C_2(x) = \int e^x * \cos(x)\,\mathrm{dx}$$

$$C_2(x) = \frac{1}{2} * e^x * (\sin(x) + \cos(x)) + C_3$$

$$y = e^{-x} * \frac{1}{2} * e^x * (\sin(x) + \cos(x)) + e^{-x} * C_3$$

$$y = \frac{1}{2} * (\sin(x) + \cos(x)) + e^{-x} * C_3$$

As we have initial point, we can find C_3

$$y(0) = 1$$

$$1 = \frac{1}{2} * (\sin(0) + \cos(0)) + e^0 * C_3$$

$$C_3 = \frac{1}{2}$$

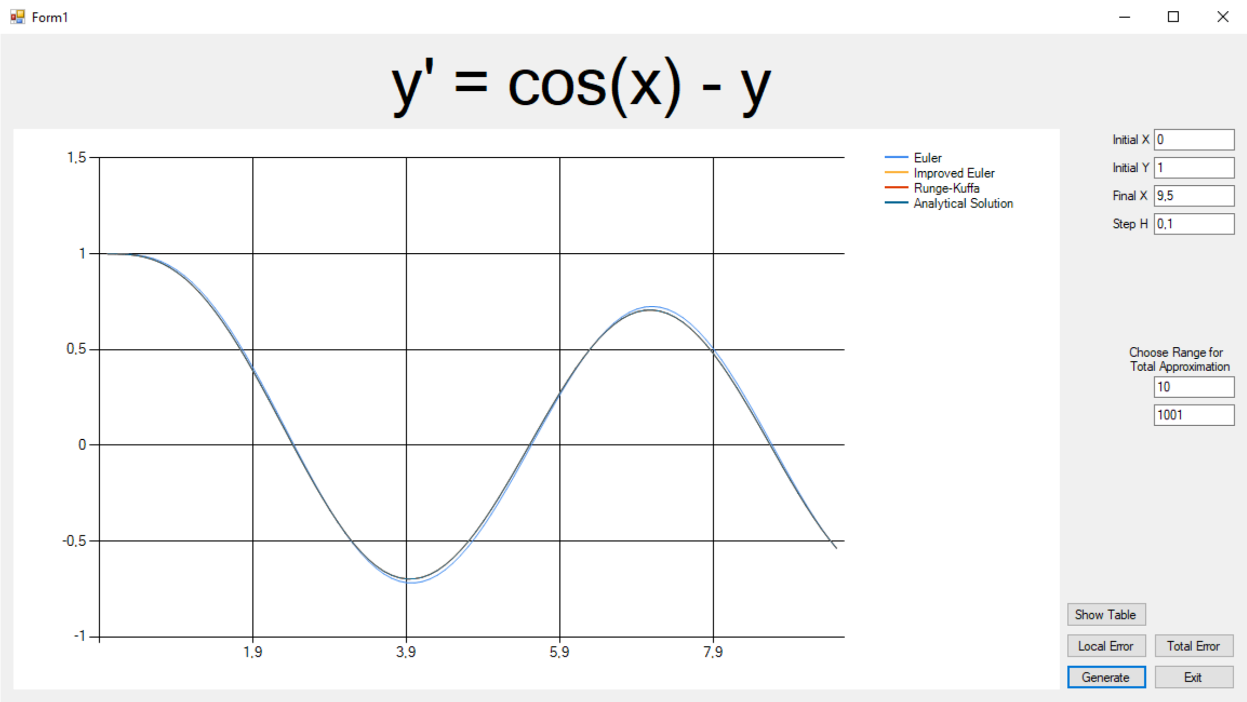
$$y = \frac{1}{2} * (\sin(x) + \cos(x)) + e^{-x} * \frac{1}{2}$$

Information about application:

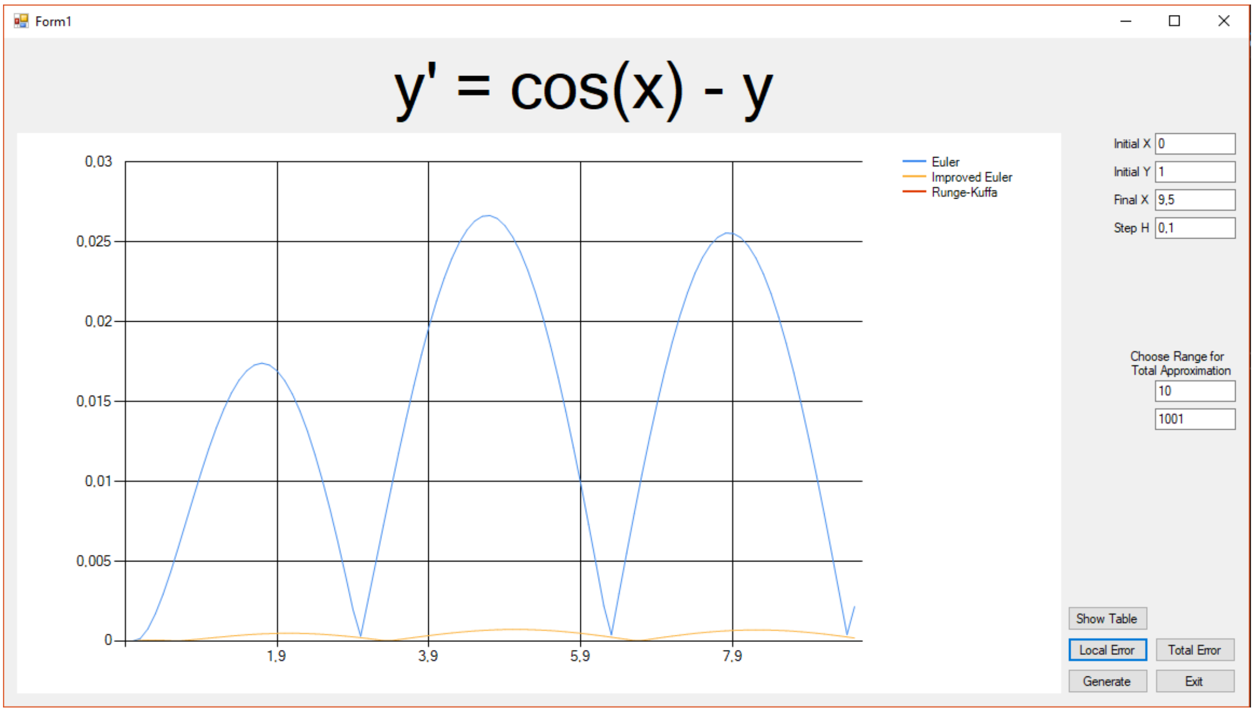
It is simple executable file of Windows form. Everything is written in the same class of Form1.

To show graphs I used charts with representation of line series. It shows graphs of each methods, graphs of local errors for each method with comparison to analytical solution, and graph of total approximation. Below you can see some screenshots of application.

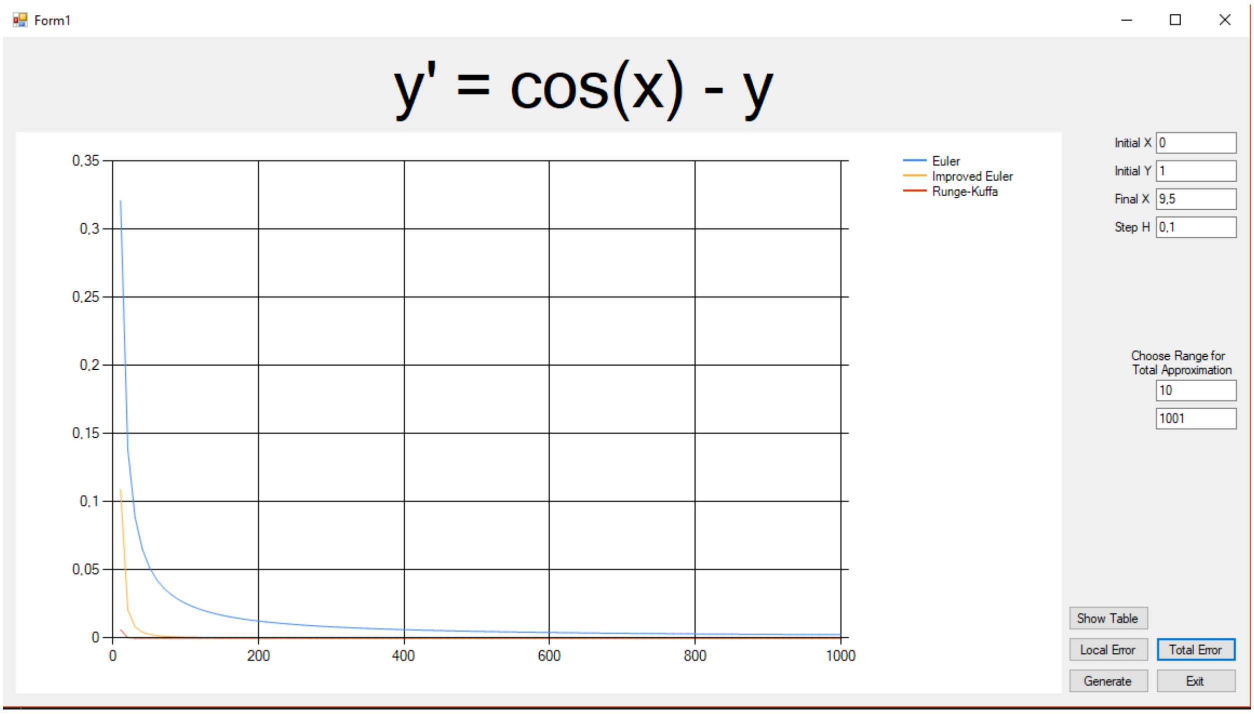
The graph of 3 different methods and analytical solution function



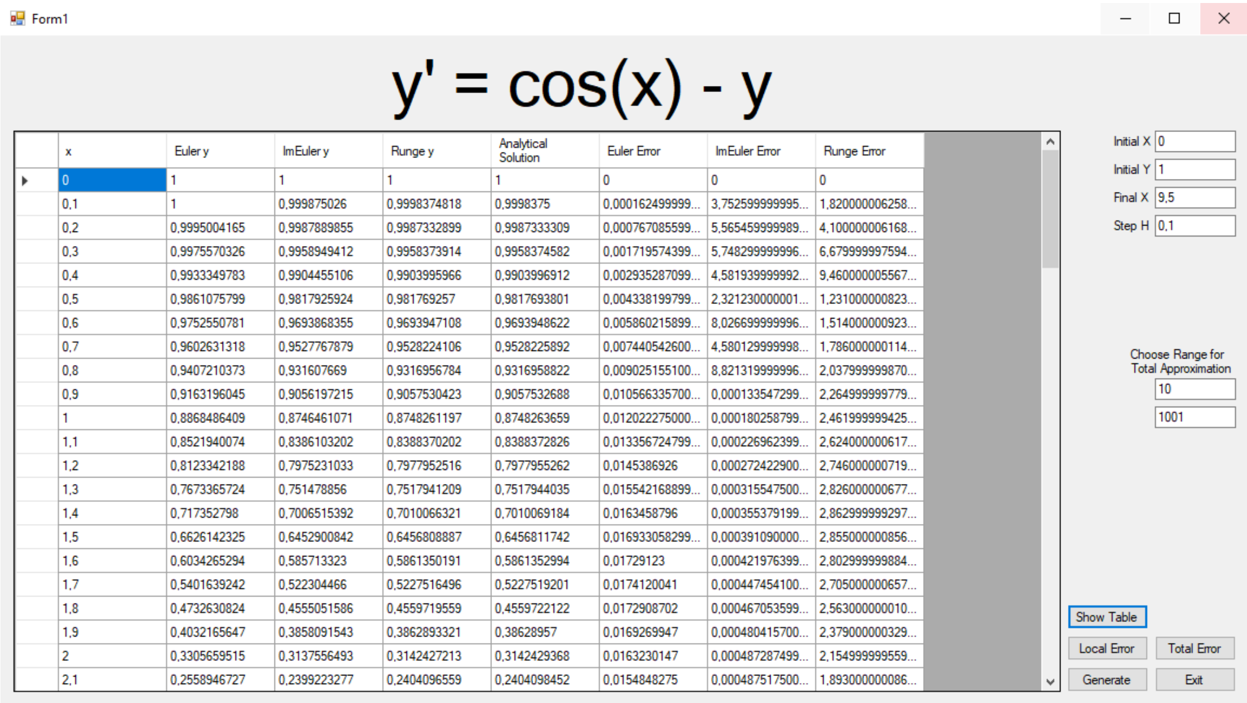
The graph of local errors with comparison to analytical solution



The graph of total approximation with dependence on number of steps



The table with results of points



x	Euler y	ImEuler y	Runge y	Analytical Solution	Euler Error	ImEuler Error	Runge Error
0	1	1	1	1	0	0	0
0,1	1	0,999875026	0,9998374818	0,9998375	0,000162499999...	3,752599999995...	1,820000006258...
0,2	0,9995004165	0,9987889855	0,9987332899	0,9987333309	0,000767085599...	5,565459999989...	4,100000006168...
0,3	0,9975570326	0,9958949412	0,9958373914	0,9958374582	0,001719574399...	5,748299999996...	6,679999997594...
0,4	0,9933349783	0,9904455106	0,9903995966	0,9903996912	0,002935287099...	4,581939999992...	9,460000005567...
0,5	0,9861075799	0,9817925924	0,981769257	0,9817693801	0,004338199799...	2,321230000001...	1,231000000823...
0,6	0,9752550781	0,9693868355	0,9693947108	0,9693948622	0,005860215899...	8,026699999996...	1,514000000923...
0,7	0,9602631318	0,9527767879	0,9528224106	0,9528225892	0,007440542600...	4,580129999998...	1,786000000114...
0,8	0,9407210373	0,931607669	0,9316956784	0,9316958822	0,009025155100...	8,821319999996...	2,037999999870...
0,9	0,9163196045	0,9056197215	0,9057530423	0,9057532688	0,010566335700...	0,000133547299...	2,264999999779...
1	0,8868486409	0,8746461071	0,8748261197	0,8748263659	0,012022275000...	0,000180258799...	2,461999999425...
1,1	0,8521940074	0,8386103202	0,8388370202	0,8388372826	0,013356724799...	0,000226962399...	2,624000000617...
1,2	0,8123342188	0,7975231033	0,7977952516	0,7977955262	0,0145386926	0,000272422900...	2,746000000719...
1,3	0,7673365724	0,751478856	0,7517941209	0,7517944035	0,015542168899...	0,000315547500...	2,826000000677...
1,4	0,717352798	0,7006515392	0,7010066321	0,7010069184	0,0163458796	0,000355379199...	2,862999999297...
1,5	0,6626142325	0,6452900842	0,6456808887	0,6456811742	0,016933058299...	0,000391090000...	2,855000000856...
1,6	0,6034265294	0,585713323	0,5861350191	0,5861352994	0,01729123	0,000421976399...	2,802999999884...
1,7	0,5401639242	0,522304466	0,5227516496	0,5227519201	0,0174120041	0,000447454100...	2,705000000657...
1,8	0,4732630824	0,4555051586	0,4559719559	0,4559722122	0,0172908702	0,000467053599...	2,563000000010...
1,9	0,4032165647	0,3858091543	0,3862893321	0,38628957	0,0169269947	0,000480415700...	2,379000000329...
2	0,3305659515	0,3137556493	0,3142427213	0,3142429368	0,0163230147	0,000487287499...	2,154999999559...
2,1	0,2558946727	0,2399223277	0,2404096559	0,2404098452	0,0154848275	0,000487517500...	1,893000000086...

Also we can see that there is no discontinuity. And the most accurate method is Rung-Kuffa, the most inaccurate is Euler.

UML diagram of the application:

```
Form1
```

```
double x0, y0, xF, h  //initial variables
int n0, nF  //initial variables

List<double> xPoints, yEulerPoints, yImEulerPoints, yRungePoints, yAnSolutionPoints, yEulerErrorPoints,
yImEulerErrorPoints, yRungeErrorPoints, xEulerTotalErrorPoints, yEulerTotalErrorPoints,
xImEulerTotalErrorPoints, yImEulerTotalErrorPoints, xRungeTotalErrorPoints, yRungeTotalErrorPoints
//lists for y and x points

Bool not_calculated //to know if needed to calculate again
```

```
void Form1_Load
void Exit_Click
void GenerateButton_Click
void TotalErrorButton_Click
void LocalErrorButton_Click
void TableButton_Click
void EulerTotalErrorSol(double finalX, double initX)
void ImEulerTotalErrorSol(double finalX, double initX)
void RungeTotalError(double finalX, double initX)
List<double> EulerErrorSol(List<double> yInputAn, List<double> yInputEu, List<double> xInput)
List<double> ImEulerErrorSol(List<double> yInputAn, List<double> yInputImEu, List<double> xInput)
List<double> RungeErrorSol(List<double> yInputAn, List<double> yInputRun, List<double> xInput)
List<double> EulerSol(List<double> xValues, double step, double initY)
List<double> ImprovedEuler(List<double> xValues, double step, double initY)
List<double> RungeKuffaSol(List<double> xValues, double step, double initY)
List<double> AnalyticalSol(List<double> xValues, double step, double initY, double initX)
double deFunct(double x, double y)
double findMaxDifference(List<double> first, List<double> second)
```