Differential Equations Computational Practicum Report

Agafonov Alexander (a.agafonov@innopolis.university)

1. Exact Solution

$$\begin{cases} y' = \frac{y}{x} + x\cos x \\ y(x_0) = 1 \\ x_0 = \pi \end{cases} \qquad Dix \in (-\infty; 0) \cup (0; +\infty)$$

1. Find the general solution

$y' = \frac{y}{x} + x\cos x$ - a linear non-homogeneous equation.

For such equations $y_g = y_h + y_p$.

1.1 Find y_h

$y' - \frac{y}{x} = 0$

$\frac{dy}{dx} = \frac{y}{x}$ Separable equation

$\frac{dy}{y} = \frac{dx}{x}$

$\int \frac{dy}{y} = \int \frac{dx}{x}$

$\ln|y| = \ln|x| + C$

$y_h = Cx$

1.2. Find y_p

$C = u(x)$

$y_p = u(x)x \qquad y_p' = u'(x)x + u(x)$

$u'(x)x + u(x) = \frac{u(x)x}{x} + x\cos x$

$u'(x)x = x\cos x$

$u'(x) = \cos x$ Separable equation

$\int du = \int \cos x\, dx$

$u(x) = \sin x + C$

$y_p = x\sin x + Cx$

1.3 Substituting y_p and y_h

$y_g = y_h + y_p = Cx + x\sin x + Cx = x(\sin x + C)$

2. Solve IVP

2.1

$y = x(\sin x + C)$

$C = \frac{y}{x} - \sin x$

2.2 Substitute initial values

$y(x_0) = 1 \qquad x_0 = \pi$

$C = \frac{1}{\pi} - \sin\pi = \frac{1}{\pi}$

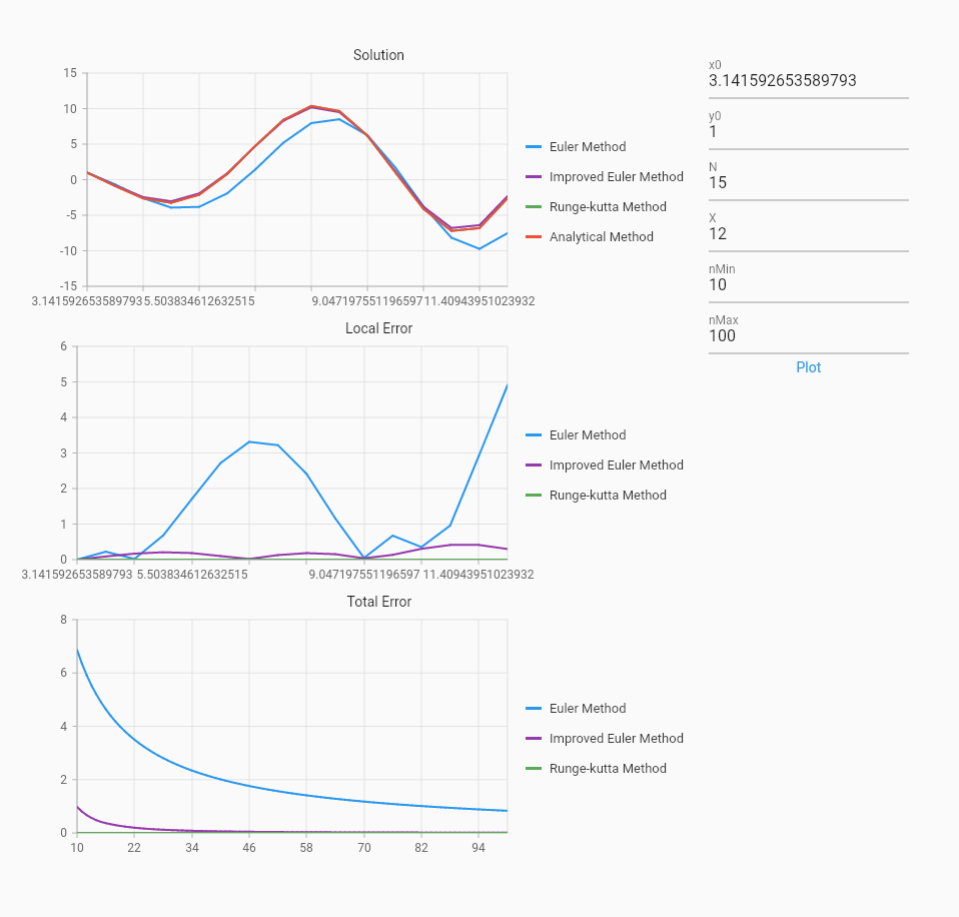
2.3 Insert C

$y = x\left(\sin x + \frac{1}{\pi}\right)$

2. Application

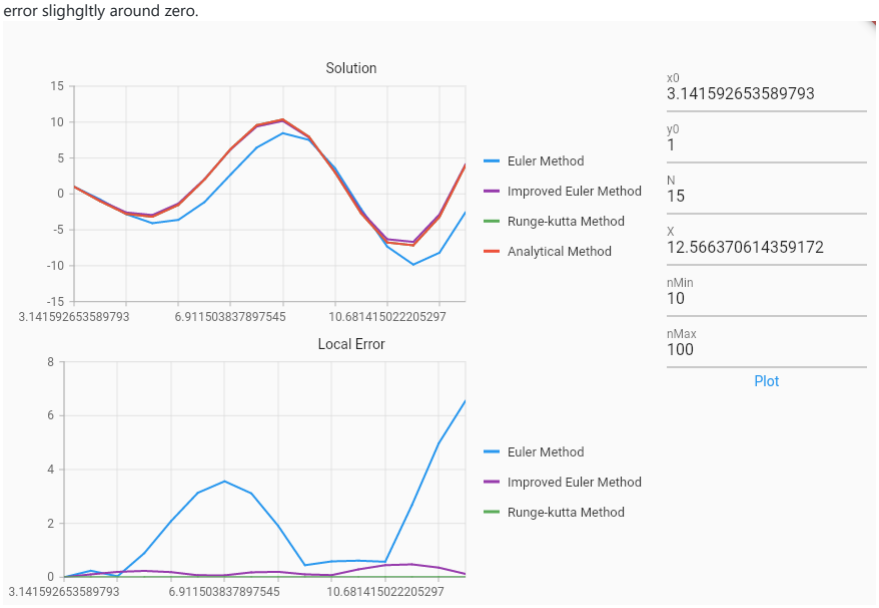
2.1 UI

The app UI is represented below. On the right there are text fields for defining the interval (**x0, X**) on which the solution will be calculated, initial value of **f(x0) = y0**, the number of steps **N**, and **nMin** and **nMax** for the minimal and maximum number of steps for calculating the total error. The left side contains three charts: chart of the solutions calculated using different methods, chart of local errors, and the chart of the total errors. Each chart on its left has the legend showing the names of the used methods. Clicking on any of them hides/reveals the according graph.

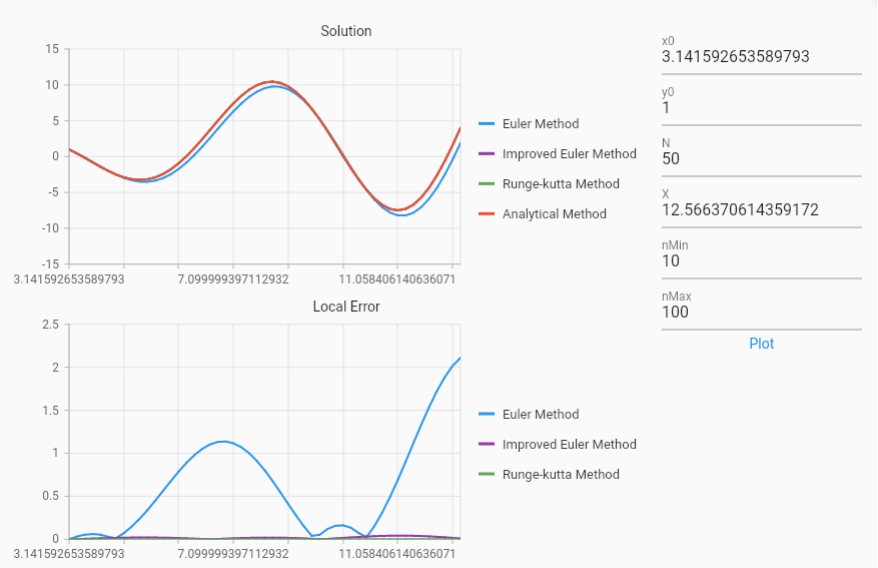


2.2 Results

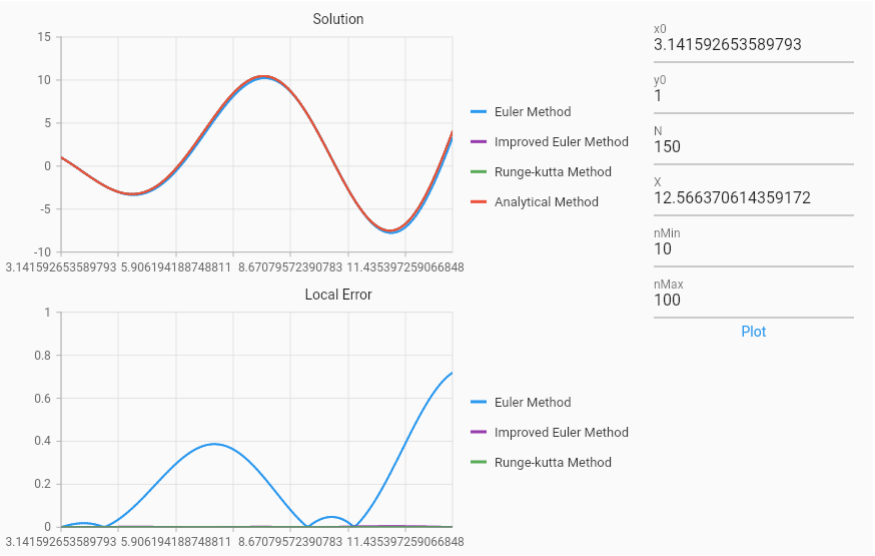
1. With small number of steps Euler method has a big error, Improved Euler method gets almost close to the exact solution, and Runge-Kutta method gets really close to the exact solution and has the error slighgltly around zero.



2. Increasing the number of steps reduces the local error of all methods.

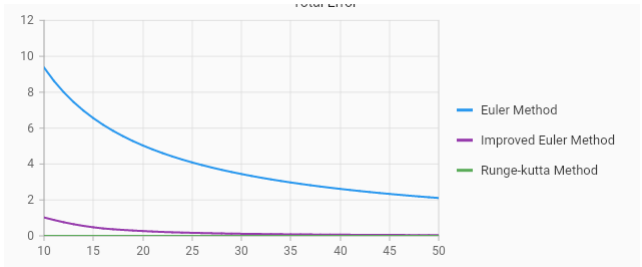


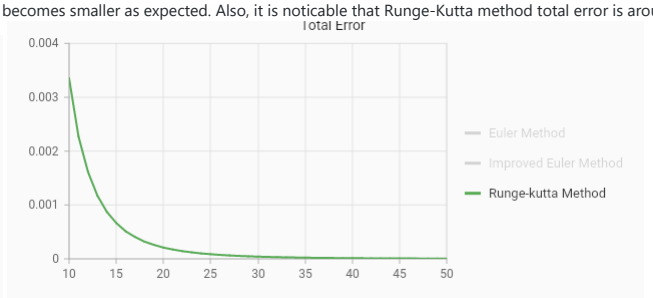
3. Incresing the number of steps further makes all graphs converge.



2.3 Total error

On thegraph of the total error we can see that with the increasing of the number of steps the total error becomes smaller as expected. Also, it is noticable that Runge-Kutta method total error is around zero for all N.

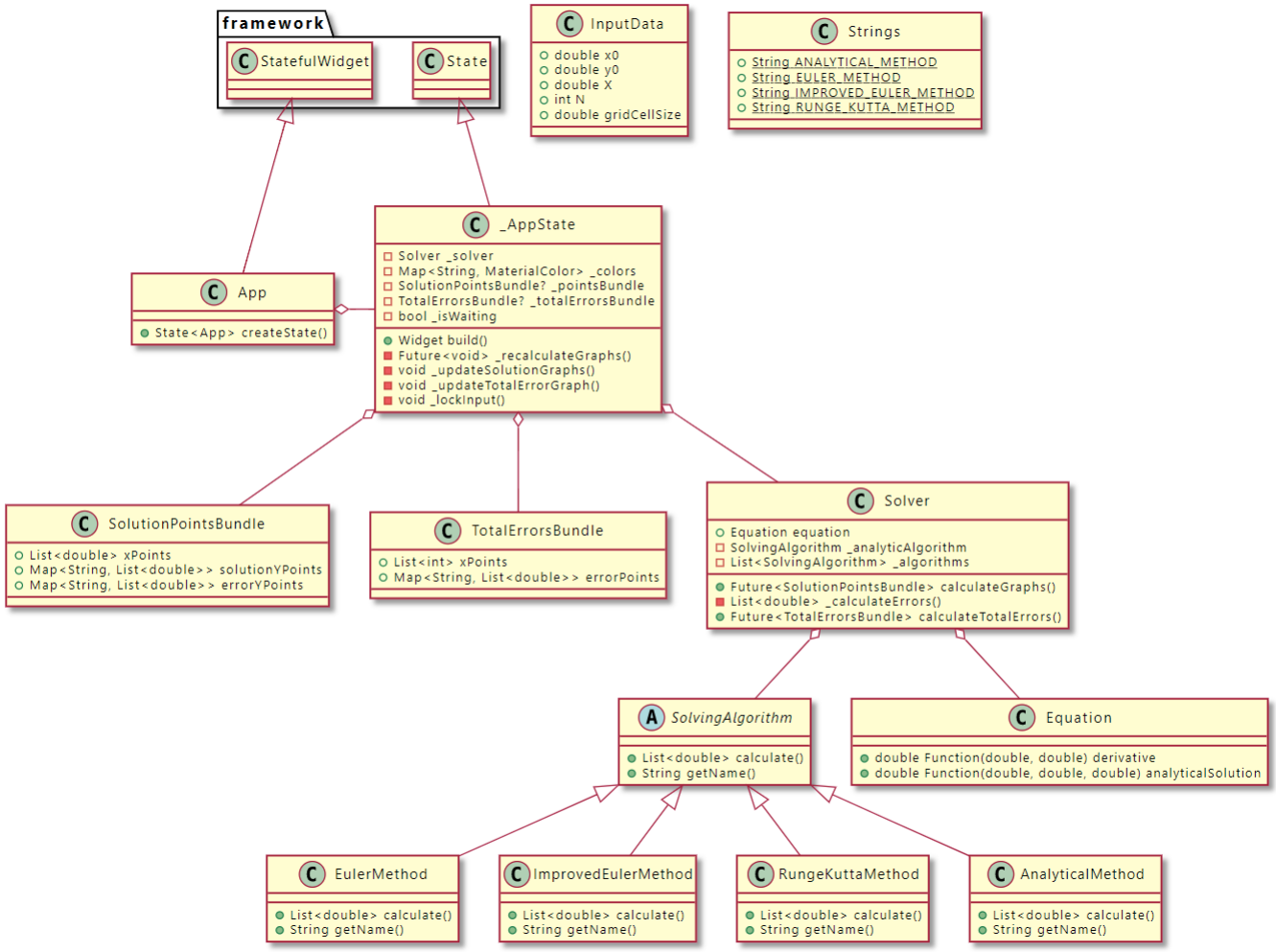




3. Code explanation

To create the app, I chose Flutter framework made by Google. For plotting charts I used the external syncfusion_flutter_charts library.

3.1 UML Diagram



3.2 Code Explanation

Class `Solver` acts like a controller that gets input data and calculates the solutions using the implementations of the `SolvingAlgorithm`. `Solver` has the field `Equation` that takes two lambdas: the derivative and analitical solution. Then `Solver` creates bundle with all the points of the solution and returns it to the `_AppState` that builds the UI.

3.3 Code Snippets

`Solver` calculating method:

```
Future<SolutionPointsBundle> calculateGraphs(InputData inputData) async {

    List<double> xPoints = List.generate(
        inputData.N + 1,
        (index) =>
            inputData.x0 + index * inputData.gridCellSize);

    List<double> exactSolution =
        _analyticAlgorithm.calculate(equation, xPoints, inputData);

    Map<String, List<double>> solutionYPoints = {
      for (var algorithm in _algorithms)
        algorithm.getName(): algorithm.calculate(equation, xPoints, inputData)
    };

    Map<String, List<double>> errorYPoints = {
      for (var algorithm in _algorithms)
        algorithm.getName(): _calculateErrors(
            exactSolution, solutionYPoints[algorithm.getName()] ?? [])
    };
    solutionYPoints[_analyticAlgorithm.getName()] = exactSolution;

    return SolutionPointsBundle(xPoints, solutionYPoints, errorYPoints);
```

Runge-Kutta algorithm:

```
List<double> calculate(
    Equation equation, List<double> xPoints, InputData inputData) {
  List<double> yPoints = [];

  xPoints.asMap().forEach((index, x) {
    if (index == 0) {
      yPoints.add(inputData.y0);
    } else {
      var k1 = equation.derivative(xPoints[index - 1], yPoints[index - 1]);
      var k2 = equation.derivative(
          xPoints[index - 1] + inputData.gridCellSize / 2,
          yPoints[index - 1] + inputData.gridCellSize * k1 / 2);
      var k3 = equation.derivative(
          xPoints[index - 1] + inputData.gridCellSize / 2,
          yPoints[index - 1] + inputData.gridCellSize * k2 / 2);
      var k4 = equation.derivative(
```

```
          xPoints[index], yPoints[index - 1] + inputData.gridCellSize * k3);
      yPoints.add(yPoints[index - 1] +
          inputData.gridCellSize * (k1 + 2 * k2 + 2 * k3 + k4) / 6);
    }
  });

  return yPoints;
}
```

`Solver` method for calculating total error:

```
Future<TotalErrorsBundle> calculateTotalErrors(int n0, int N, InputData inputData) async {

  var map = <String, List<double>>{
    for (var algo in _algorithms) algo.getName(): []
  };

  var xPoints = List.generate(N - n0 + 1, (index) => n0 + index);
  for (var i in xPoints) {
    var pointsBundle = await calculateGraphs(inputData.copyWith(N: i));
    for (var errorPoints in pointsBundle.errorYPoints.entries) {
      var maxError = 0.0;
      errorPoints.value.asMap().forEach((index, value) {
        maxError = max(value, maxError);
      });
      map[errorPoints.key]!.add(maxError);
    }
  }
  return TotalErrorsBundle(xPoints, map);
}
```

Method that is called when **Plot** button is clicked:

```
  Future<void> _recalculateGraphs(InputData inputData, int nMin, int nMax) async {
  _lockInput(true);


  Future.wait([
    _solver.calculateGraphs(inputData).then(_updateSolutionGraphs),
    _solver
        .calculateTotalErrors(nMin, nMax, inputData)
        .then(_updateTotalErrorGraph)
  ]).then((_) => _lockInput(false));
}
```