Computational Practicum
Differential Equations

Dvoryanov Daniil, BS17-06

November 2018

1 Problem Statement

In this assignment we are asked to implement three numerical methods, namely: Euler's, Improved Euler's, and Runge-Kutta methods, to find approximated solutions to the given differential equation. We then need to calculate and plot the solution and two types of errors: local error and maximum (global) error. My variant is 9, the IVP is as follows:

$$\begin{cases} y' = \frac{y}{x} + \frac{x}{y} \\ y(1) = 1 \\ x \in [1; 2.3] \end{cases}$$

2 Exact Solution

2.1 General Solution

As we see, it is nonlinear homogeneous ordinary 1st order d.e. Moreover, it is a Bernoulli equation. The solution is as follows:

$$\frac{dy}{dx} = \frac{x}{y} + \frac{y}{x}$$
$$\frac{dy}{dx} - \frac{y}{x} = xy^{-1} \qquad | * y^{-1}$$
$$y\frac{dy}{dx} - \frac{y^2}{x} = x$$

Substitution: $z = y^2$, $dz = 2ydy$

$$\frac{1}{2} * \frac{dz}{dx} - \frac{z}{x} = x$$

Let's solve the complementary equation:

$$\frac{1}{2}z' = \frac{z}{x}$$
$$\frac{dz}{z} = 2\frac{dx}{x}$$
$$z = C_1x^2 = z_1$$
$$z = u * z_1,\ z' = u'z_1 + uz_1'$$
$$\frac{1}{2}u'z_1 + u(\frac{1}{2}z_1' - \frac{z_1}{x}) = x$$
$$\frac{1}{2}u' * (C_1x^2) = x$$
$$du = \frac{2}{C_1}\frac{dx}{x}$$
$$u = \frac{2}{C_1}\ln|x| + C_2$$
$$z = u * z_1 = 2x^2\ln|x| + Cx^2$$
$$y = \sqrt{z} = x\sqrt{2\ln|x| + C}$$

2.2 IVP Solution

$$\begin{cases} y = \sqrt{z} = x\sqrt{2\ln|x| + C} \\ y(1) = 1 \end{cases} \implies \begin{cases} 1 = \sqrt{C} \\ y(1) = 1 \end{cases} \implies C = 1$$

$$y = x\sqrt{2\ln|x| + 1}$$

3 Technology Stack

In order to complete this task, I used several of libraries and tools. Here is there list and brief description.

- As a programming language, I decided to use **Python** as it is best suited for this kind of tasks.
- As a library for plotting data, I use **Plotly**, as it is very convenient and has good-looking plots (it is not very fast though, unfortunately).
- To store and easily manipulate different parameters, e.g. initial value, step, etc., I use configuration file and library **configparser** to easily read it.
- In order to be able to effectively create different types of arrays, I use **numpy** library.
- As a mean of increasing the performance of the application, I decided to use **lru_cache** function wrapper from **functools** library. It allows to avoid calculating the same function twice (providing the same arguments).

4 Implementation

To provide better extendability and readability, I have two different files (and two different classes) responsible for most of the work. One another file has just several lines and serves as a container for the driver code. Class Calculator is further divided into two classes, **Values** and **Errors**, for improved readability and structural benefits. All three parts (Plotter and two subclasses of Calculator) have a dedicated section in the config file.

In this section of the report I will provide a general information about all the parts and most important functions in my project (you can see the detailed description in the documentation of the code).

4.1 Values

Values class is used for calculating the results of different numeric methods for the given IVP on the given interval, as well as calculating the exact solution on the given interval. The first four methods of this subclass calculate the result of a specific numerical method. The other two represent the given function for y' and it's analytical solution.
The section in the config file for this class is called *VALUES* and has the following parameters:

- *x0*: A beginning of the interval and x coordinate of the initial value.
- *y0*: y coordinate of the initial value.
- *xf*: An end of the interval of calculation.
- *h*: A grid step.

4.2 Errors

Errors class is dedicated to calculating two types of errors:

- *Local error:* a list of absolute values of differences between the result of a particular numerical method and the exact solution (i.e., for the list of results of the exact solution y and list of results of a numerical method t for the given grid step, local error $e_i = |y_i - t_i|$ for each i in the grid.
  Methods **euler_local**, **improved_euler_local**, and **range_kutta_local** are dedicated to the calculation of this error for the corresponding numerical methods.

- *Global (maximum) error:* a list of maximum local local errors for different grid sizes.

The section in the config file for this class is called *ERRORS* and has the following parameters:

- *n_err_steps*: Number of different grid steps for the calculation of global error.
- *err0*: Smallest step for the global error.
- *errf*: Largest step for the global error.

4.3 Plotter

Plotter class is dedicated to creating plots of values and errors. It uses *Plotly* API to create html files. Graph objects are used to represent and prepair data for the plotting. The section in the config file for this class is called *ERRORS* and has the following parameters:

- *values_filename*: The desired filename for the file containing plot of the values.
- *local_errors_filename*: The desired filename for the file containing plot of the local errors.
- *global_errors_filename*: The desired filename for the file containing plot of the global errors.
- *values_mode*: Display mode for the values graph.
- *local_errors_mode*: Display mode for the local errors graph.
- *global_errors_mode*: Display mode for the global errors graph.

5 Results

In this section I would like to present the results of my work: graphs of values, local errors and global errors. For the demonstration purposes I will use the following config settings: *x0*, *y0*, and *xf* are set accordingly to the given IVP; h is set to 10^{-4}; *err0* and *errf* are set to 10^{-4} and 10^{-2} respectively, and *n_err_steps* (number of steps for global error) is set to 200.

5.1 Values

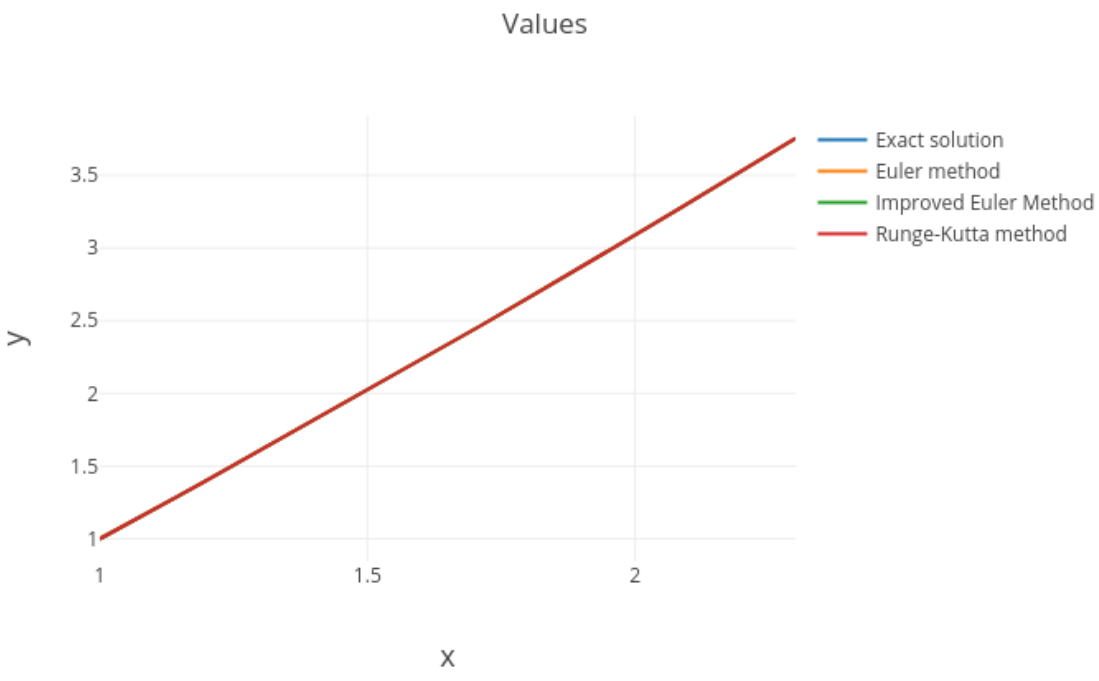
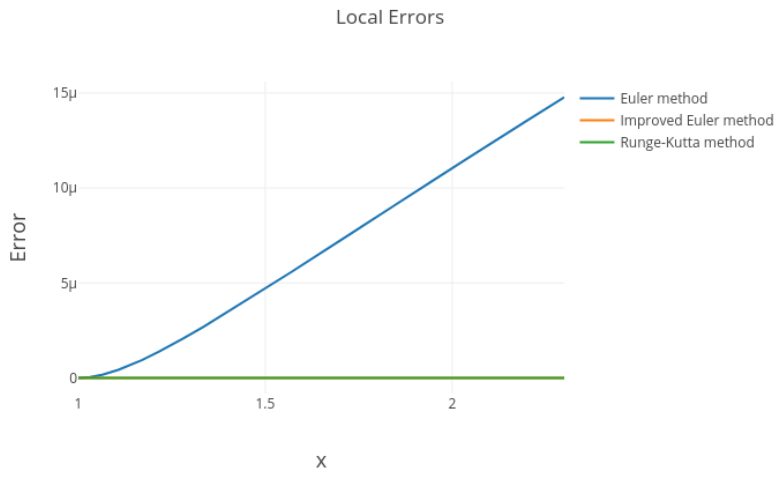


Figure 1: Values Graph

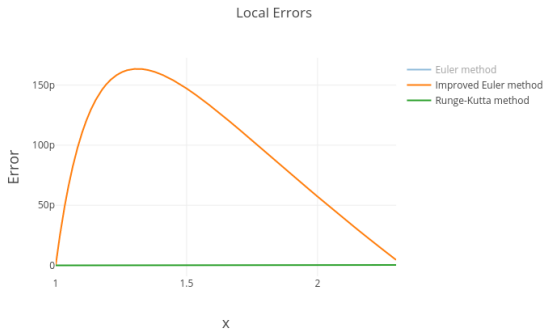
Above you can see the graph for the results of all three numerical methods and the exact values (values graph). It is hard to see on such a small scale, but if you try to zoom in on the line, it will still be quite hard to spot different lines. It may be not obvious now, but when we get to the errors graphs, we will see that they (errors) are very small (for Euler method, for example, you can spot the difference only on the sixth and further number after the decimal point; for others methods it is even less noticeable).

5.2 Local Error

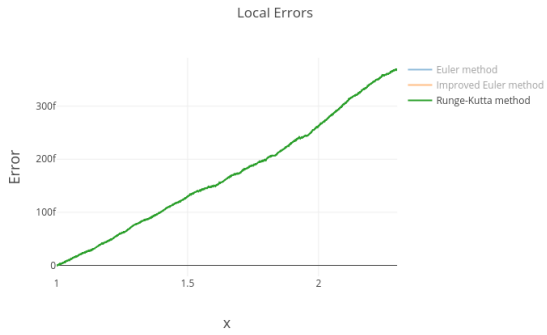
As mentioned earlier in section *Implementation*, local error is a list of absolute values of differences between the exact value and a value yielded by some numerical method in a point of the grid. In our configuration, local error graph looks like this:



(a) All local errors



(b) Without Euler error



(c) Only Runge-Kutta error

Figure 2: Local errors graph with some of graphs consecutively disabled.

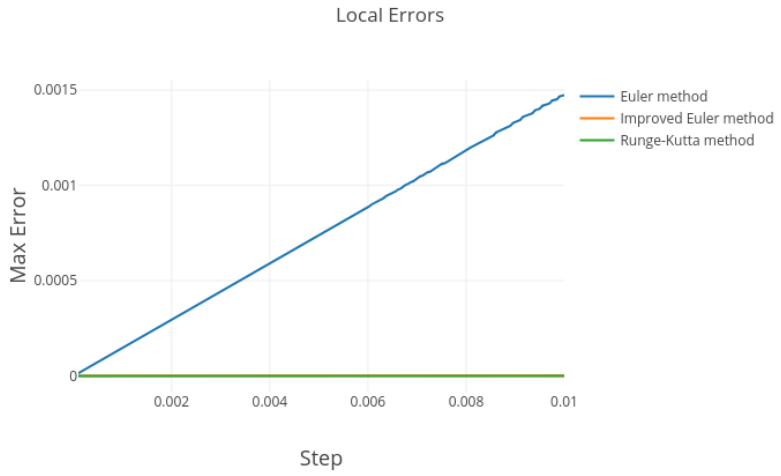
On these pictures we can see some strange coefficients on y axis, here's their values:

- $\mu = 10^{-6}$
- $n = 10^{-3}\mu = 10^{-9}$
- $p = 10^{-3}n = 10^{-12}$
- $f = 10^{-3}p = 10^{-15}$

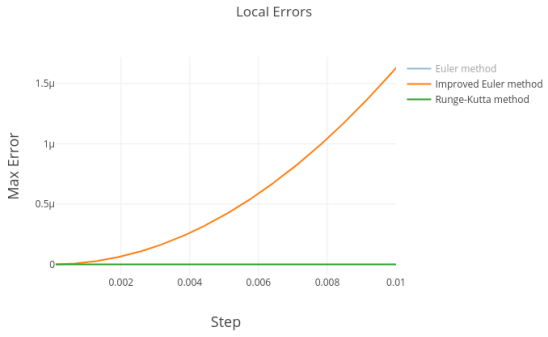
As we see on these pictures (and will see it further in the next subsection), *Euler error* > *Improved Euler Error* > *Runge Kutta Error*; in other words, Euler method has the worst approximation, Improved Euler is much better, and Runge-Kutta method gives the best approximation.

5.3 Global (Maximum) Error

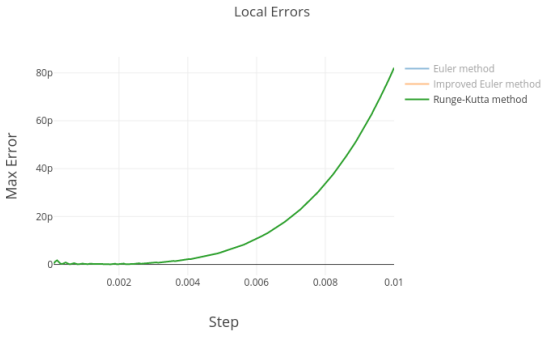
This error is given by maximum local error for different steps. In other words, I take some list of steps (number of steps, initial and final step can be set in *config* file), and for each step find the maximum local error. The plot depicts the dependency of maximum local error on step. In this example, as mentioned earlier, I took the number of steps to be 200, and set the initial step and final step to be 10^{-4} and 10^{-2}. Here is the graph:



(a) All global errors



(b) Without Euler error



(c) Only Runge-Kutta error

Figure 3: Global errors graph with some of graphs consecutively disabled.

As we have already seen in the local errors section, three graphs have very different y-values (each subsequent error is about a million times lower at average). Moreover, as the step increases, we can observe different behaviour of functions: euler error grows linearly, while two other error have a much more gradual increase.

6 Summary

As the result of this project, I implemented three different numerical methods for approximating the solution of the given d.e. I observed the local and global (maximum local) errors for these methods and come to obvious conclusion: the best approximation method is *Runge-Kutta* and the worst is *Euler's* method out of the three methods. The Improved Euler's method also gives reasonable approximation, and much better than Euler's method.

To avoid using GUI (as it requires a lot of time to get acquainted with a new framework) I decided to use a configuration file, where one can easily specify all the parameters.

Lastly, I discovered that *Plotly* (and underlying *Matplotlib* is not the best option for plotting big amounts of data (for example, in this assignment one should not specify a step less than 10^{-6}, as it would take a huge amount of time to calculate).