Computational Practicum Report

Sergey Bogdanik, BS2-8

November 4, 2018

Problem Statement

1. Variant: 2
2. Equation: $y\prime = -2y + 4x$
3. Initial values: $x_0 = 0$, $y_0 = 0$
4. Interval: $X = 3$

Part I

$$y\prime = -2y + 4x \tag{1}$$

$$y\prime + 2y = 4x \tag{2}$$

This is first-order linear nonhomogeneous ordinary differential equation. Complementary equation:

$$y_c\prime + 2y_c = 0 \tag{3}$$

$$y_c = e^{-2x} \tag{4}$$

Exact solution:

$$\begin{gathered} y = y_c \int \frac{f(x)}{y_c(x)}\mathrm{d}x = e^{-2x} \int \frac{4x}{e^{-2x}}\mathrm{d}x = 4e^{-2x} \int xe^{2x} = \\ 4e^{-2x}(\frac{e^{2x}}{4}(2x-1) + C) = Ce^{-2x} + 2x - 1 \end{gathered} \tag{5}$$

Constant:

$$C = (y_0 + 1 - 2x_0)e^{2x_0} \tag{6}$$

$$C = (0 + 1 - 2 \cdot 0)e^{2 \cdot 0} = 1 \tag{7}$$

Initial value problem's solution:

$$y = e^{-2x} + 2x - 1 \tag{8}$$

This function is continuous on $\mathbb{R}$.

Part II

The program for solving this differential equation can be found on this GitHub repository. This program can be configured to solve any simple (without discontinuity points) differential equation. There are 2 way to use it. Run the script "as is" or import it and reset the arguments of main function. The code is self-explanatory enough therefore I will not write a manual here. After finishing computations the program creates a folder named with timestamp and leaves there 3 png files.

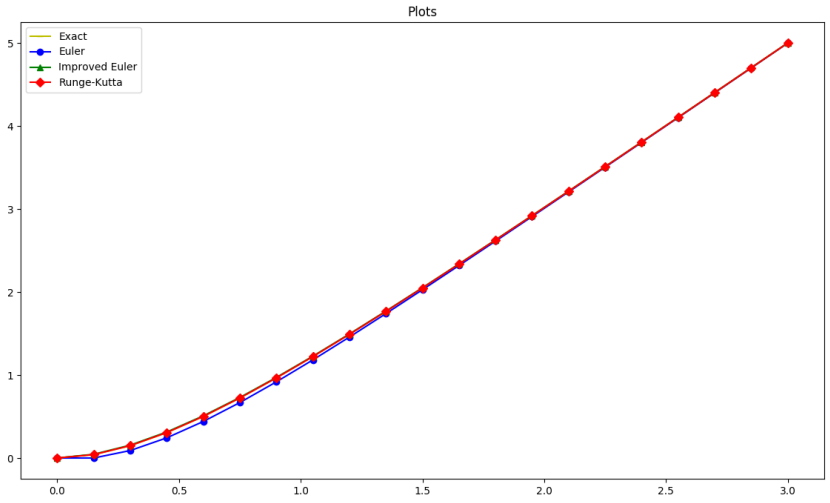

Figure 1: Approximations

Plots

The first file generated by the program is `plots.png`. It contains 4 plots:

1. Exact solution of the problem. It should be calculated manually and sent to the main function as an argument.

2. Euler's method approximation. It is calculated from the formula of derivative form the problem's statement and initial values points.

3. Improved Euler's method approximation. It is calculated almost the same as Euler's method approximation but with a greater amount of intermediate competitions.

4. Runge-Kutta's method of approximation. This method is even more accurate than Improved Euler's method and usually drawn above the Exact solution.

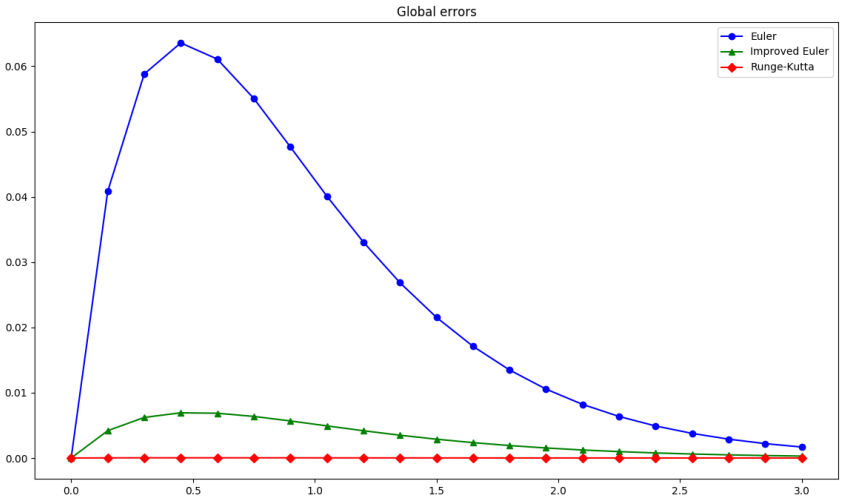

Figure 2: Global errors

Global errors

The second file generated by the program is `global_errors.png`. The global error is absolute difference between the exact solution and an approximation. The file contains 3 plots of global errors:

1. Euler's method error. The biggest one.
2. Improved Euler's method approximation. Moderate error.
3. Runge-Kutta's method of approximation. Very close to zero.

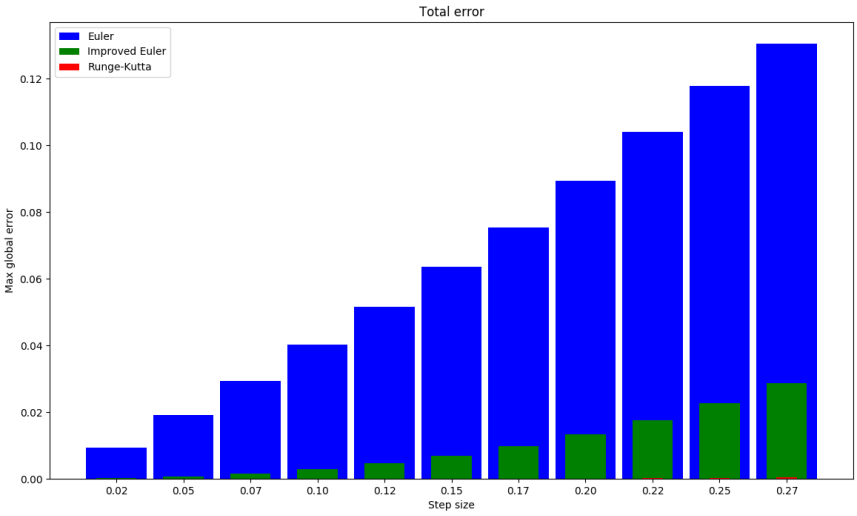

Figure 3: Total error

Total error

The third file generated by the program is `total_error.png`. The total error is the maximum value of all calculated global errors. The file contains some (by default 11) bars where the height of the bar is the total error for a specified step. The middle bar (it's recommended to have odd number of bars) shows the total error for the step size specified in the main function. The step sizes of bars from the left side are less than the original step size and from the right side are greater than the original one.

Source code

The most important part of the source code is a function (actually lambda function) which solves an equation using a specified method.

```
solve = lambda xs, ys, method, h, yp : list(map(lambda
    x : ys.append(method(x, ys[-1], h, yp)), xs[:-1]))
```

Where:

1. `xs` is a list of points on the x axis.
2. `ys` is a list which contains y coordinate of initial point and all new y coordinates will be added.
3. `method` is another lambda which computes the next y coordinate of the approximated plot.
4. `h` is a step size.
5. `yp` is $y\prime$ from the problem's statement.

This function doesn't return anything but append the result into `ys` list.

The simplest method to approximate an equation is Euler's method. It looks like this.

```
euler = lambda x, y, h, yp : y + h * yp(x, y)
```

Where:

1. `x` is a current x coordinate
2. `x` is a current x coordinate
3. `h` is a step size
4. `yp` is $y\prime$ from the problem's statement.

This function returns the y coordinate of the next point.

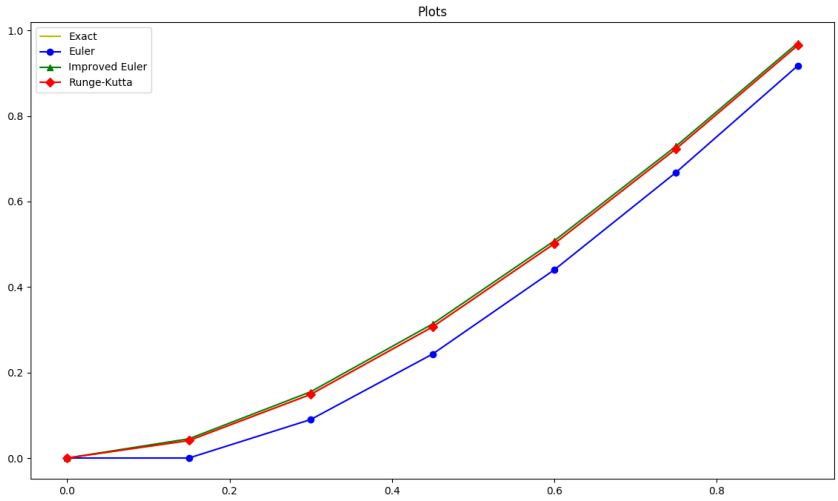

Figure 4: Approximated plots on the interval from 0 to 1

Part III

The total approximation error is analyzed in section "Total error" of the previous part. It's clear that the total error increases with increasing the step size and decreases otherwise. It's a common behavior for all approximation methods. Also It's easy to see that Euler's method is the worst one and Runge-Kutta is the best one. It's also an expected behavior.

Analyzing the graph of errors we can see that the global error increases when x increases until it reaches a point 0.5 and then starts to decrease. This is true for all methods of approximation. This behavior can be explained by analyzing the exact solution. The solution is a sum of linear function and exponential function in power of $-2x$. When x is small the the exponential function play greater role in growing of y. But later the value of the exponential function becomes very small (because of negative power) and the plot looks like a line. It's easy to approximate a line knowing it's derivative. That is why the error decreases with growing x after a point 0.5. And the value increases from 0 to 0.5 because the value of exponential function is relatively high that the approximation methods can't easily follow the function. It can be more noticeable on a plot form 0 to 1 drawn on the upper part of this

page.

To make this plot type this in python interpretator and open file named `plots.png` to see how bad some approximation are on this interval.

```
>>> import diffeq
>>> diffeq.main(X=1)
```