Differential Equations Assignment

Daniil Fronts, Group 3

November 2019

1 Github repository:

https://github.com/Frodan/DE_assignment

2 Exact solution

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

$y_0 = 1, x_0 = \pi$

This is Linear First Order Differential equation.

Solve the complementary equation:

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

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

$ln(y) = ln(Cx)$

So, $y = C(x) * x$

$y' = C'(x) + C(x)$

$C'(x) + C(x) - C(x) = x * cos(x)$

$C'(x) = cosx \Rightarrow C(x) = sin(x) + C_1$

$y = (sin(x) + C_1) * x$

We have $y(\pi) = 1$, so

$1 = (sin(\pi) + C_1) * \pi$

$C_1 = \frac{1}{\pi}$

Answer: $y = (sin(x) + \frac{1}{\pi}) * x$

3 View explanation

For creating GUI was used Python 3 with matplotlib library. I used several checkboxes to let user choose what they want to be displayed on chart. They have default values and if user want to change them, he should select the checkbox with graph name. Also there is radiobuttons for error plot with two choises: "Global Error" or "Local Error" which defines what plot will be shown. There are also some textfields for defining initial values, by default they have values from our pdf task.

4 Model explanation

In project there are six classes. First of all, I have an abstract class "Solution" which has default methods for subclasses and abstract methods that all subclasses will redefine.
ExactMethod class uses answer from first part of report with possibility of changing y_0. Euler Method uses formula:

$y_i = y_{i-1} + step * (\frac{y_{i-1}}{x_{i-1}} + x_{i-1} * cos(x_{i-1}))$

Improved Euler Method uses formula:

$arg1 = x_{i-1} + \frac{step}{2}$

$arg2 = y_{i-1} + \frac{step}{2} * (\frac{y_{i-1}}{x_{i-1}} + x_{i-1} * cos(x_{i-1}))$

$y_i = y_{i-1} + step * (\frac{arg2}{arg1} + arg1 * cos(arg1))$

Runge-Kutta Method uses formula:

$k1 = (\frac{y_{i-1}}{x_{i-1}} + x_{i-1} * cos(x_{i-1}))$

$k2 = (\frac{y_{i-1} * \frac{step}{2} * k1}{x_{i-1} * \frac{step}{2}} + x_{i-1} * \frac{step}{2} * cos(x_{i-1} * \frac{step}{2}))$

$k3 = (\frac{y_{i-1} * \frac{step}{2} * k2}{x_{i-1} * \frac{step}{2}} + x_{i-1} * \frac{step}{2} * cos(x_{i-1} * \frac{step}{2}))$

$k4 = (\frac{y_{i-1} + step * k3}{x_{i-1} + step} + (x_{i-1} + step) * cos(x_{i-1} + step))$

$y_i = y_{i-1} * \frac{step}{6} * (k1 + 2 * k2 + 2 * k3 + k4)$

All methods classes has methods, that responds for counting global and local errors. Local error is calculated as an absolute value of difference between a global error on $step_i$ and a global error on $step_{i-1}$. Global error methods initializes everything and find the max Error for every step. After calculations maximum global error on domain for concrete step is added to graph.

5 Aprovement of solidity principles

SINGLE RESPONSIBILITY PRINCIPLE

If we consider class "Solution" we will see that all the methods are called in handle method. "__init__" is responsible for setting initial values. "update" is responsible for updating values, given by user. "get_xlist" returns list of x value for every point, "get_ylist" returns list of y value for every point. Same for local and global error lists. "plot" method adds graph to the screen and returns it as object.

LISKOV SUBSTITUTION PRINCIPLE

If we will use in declaration of any solution class not their exact class, but the parent class "Solution", nothing will change since we initialize them as exactly one concrete solution each.

INTERFACE SEGREGATION PRINCIPLE

App was created the way, that it will not break if user will not give some values: in this case default values will be used or program will wait for values it needs. The only thing is needed to emphasize is that if in X-axes domain will contain zero, then graph will be printed not correct. As a hint for user, in each textfield it is written what should be placed there.
So, as was proven above app satisfies solidity principles.

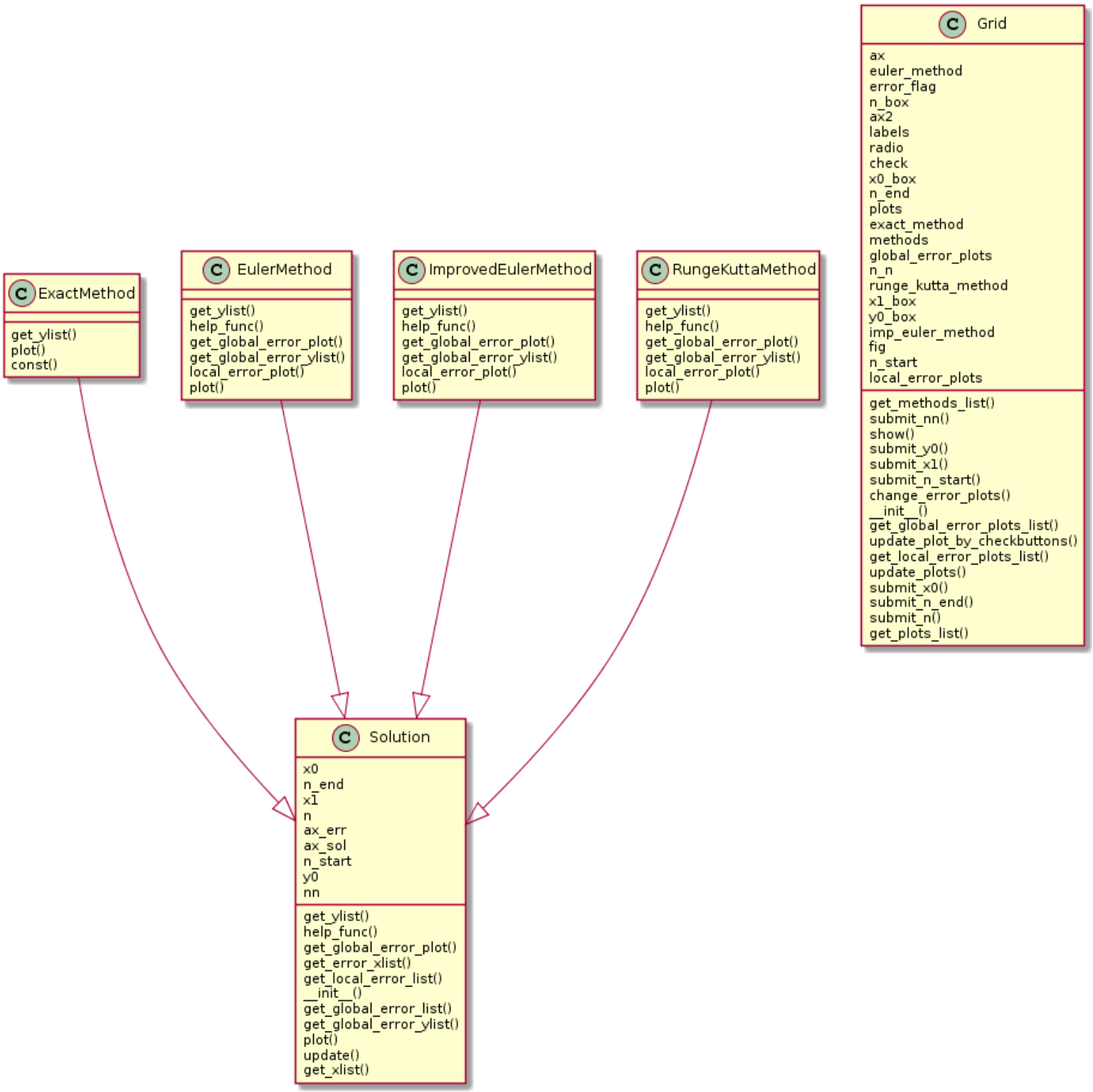
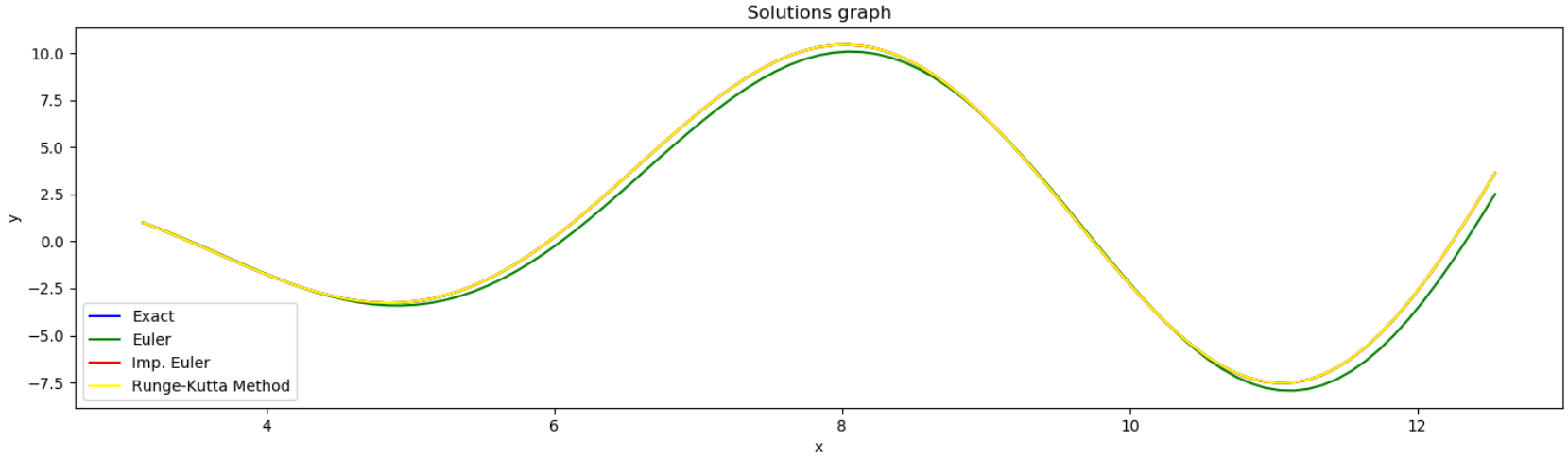

Grid
ax
euler_method
error_flag
n_box
ax2
labels
radio
check
x0_box
n_end
plots
exact_method
methods
global_error_plots
n_n
runge_kutta_method
x1_box
y0_box
imp_euler_method
fig
n_start
local_error_plots
get_methods_list()
submit_nn()
show()
submit_y0()
submit_x1()
submit_n_start()
change_error_plots()
__init__()
get_global_error_plots_list()
update_plot_by_checkbuttons()
get_local_error_plots_list()
update_plots()
submit_x0()
submit_n_end()
submit_n()
get_plots_list()
ExactMethod
get_ylist()
plot()
const()
EulerMethod
get_ylist()
help_func()
get_global_error_plot()
get_global_error_ylist()
local_error_plot()
plot()
ImprovedEulerMethod
get_ylist()
help_func()
get_global_error_plot()
get_global_error_ylist()
local_error_plot()
plot()
RungeKuttaMethod
get_ylist()
help_func()
get_global_error_plot()
get_global_error_ylist()
local_error_plot()
plot()
Solution
x0
n_end
x1
n
ax_err
ax_sol
n_start
y0
nn
get_ylist()
help_func()
get_global_error_plot()
get_error_xlist()
get_local_error_list()
__init__()
get_global_error_list()
get_global_error_ylist()
plot()
update()
get_xlist()

Figure 1

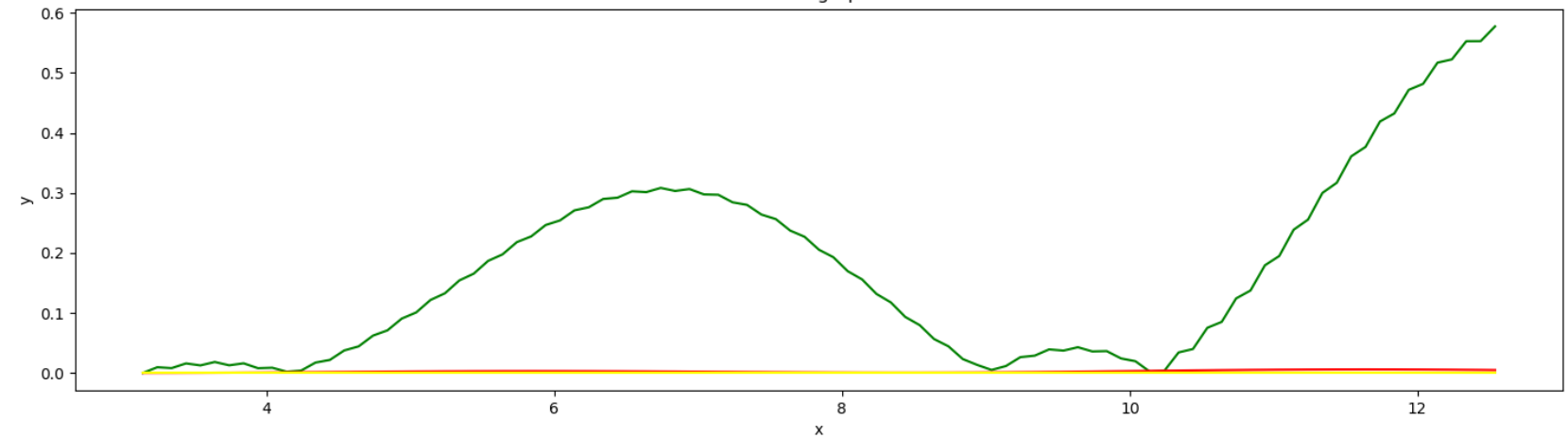
Exact
Euler
Imp. Euler
Runge-Kutta Method

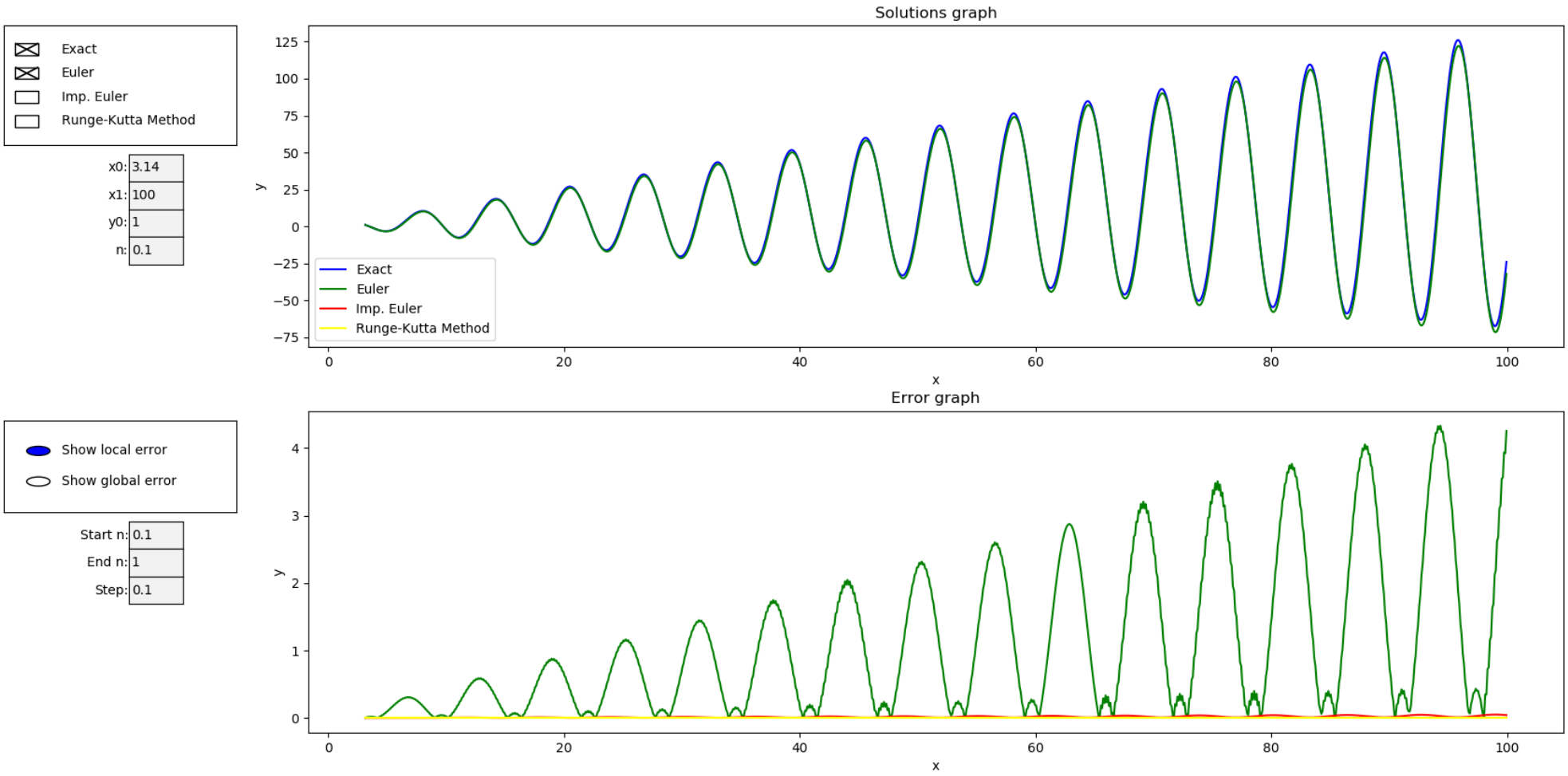
x0: 3.14
x1: 12.56
y0: 1
n: 0.1



Show local error
Show global error

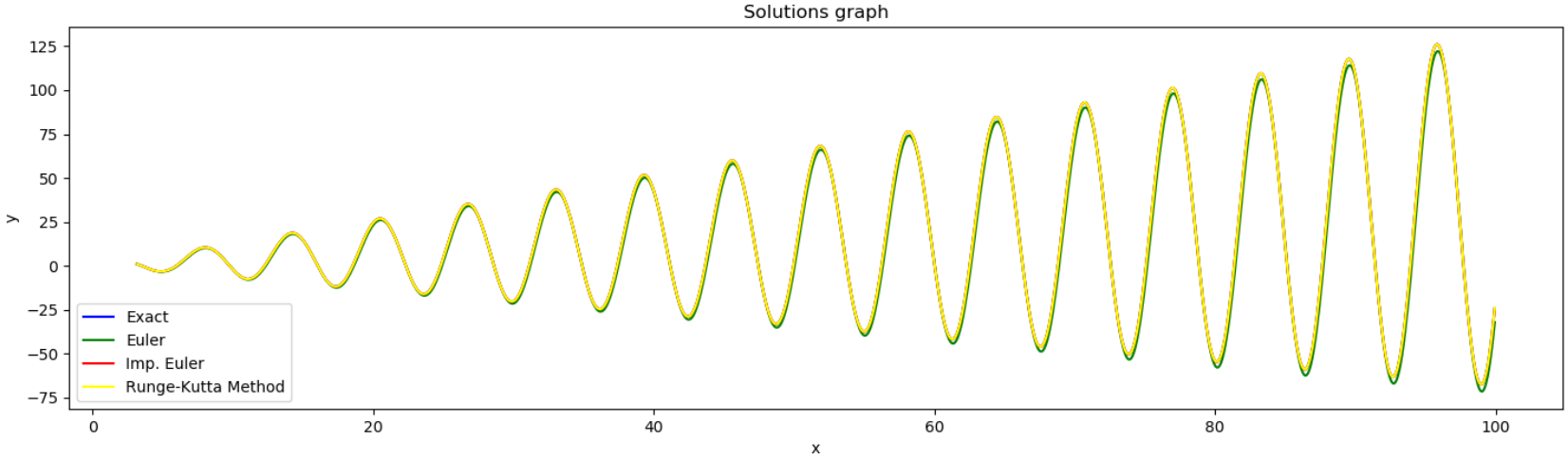
Start n: 0.1
End n: 1
Step: 0.1




Exact
Euler
Imp. Euler
Runge-Kutta Method
x0: 3.14
x1: 100
y0: 1
n: 0.1
Solutions graph
y
x
Exact
Euler
Imp. Euler
Runge-Kutta Method
Error graph
Show local error
Show global error
Start n: 0.1
End n: 1
Step: 0.1

Exact
Euler
Imp. Euler
Runge-Kutta Method

x0:	3.14
x1:	100
y0:	1
n:	0.1



Show local error
Show global error

Start n:	0.1
End n:	5
Step:	0.1