Differential equations assignment.

Ivan Rybin gr.5

November 12, 2019

Contents

1 Variant 14

Here is my variant:

$y' = (1 + y/x)ln((x + y)/x) + y/x$

$y_0 = 2, x_0 = 1$

2 Solution

Let $y = xv$, then $y' = xv' + v$. So:

$xv' + v = (1 + v)ln(1 + v) + v$

$xv' = (1 + v)ln(1 + v)$

$\frac{dv}{dx} = (1 + v)ln(1 + v)/x$

$\int \frac{dv}{(1+v)ln(1+v)} = \int \frac{dx}{x}$

$ln(ln(1 + v)) = ln(x) + c_1$

$ln(1 + v) = xe^{c_1}$

$1 + v = e^{e^{c_1 x}}$
$v = c_1^x - 1$
$y = (c_1^x - 1)x$
So we can count c_1:
$y_0 = 2, x_0 = 1$
$2 = (c_1^1 - 1)1$
$2 = c_1 - 1$
$c_1 = 3$
The final answer is:
$y = (3^x - 1)x$

3 UML diagram of classes and their fields

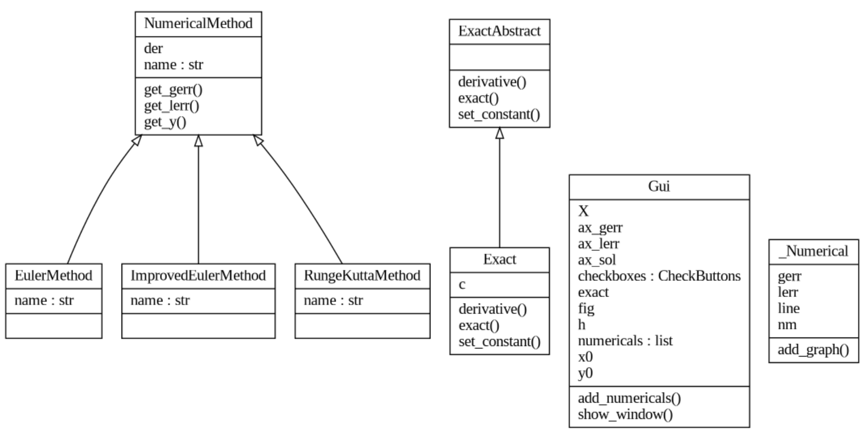

Figure 1: Classes, their methods, and fields, and their relations

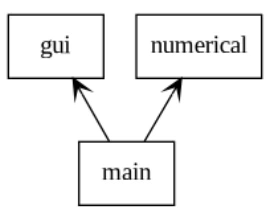

Figure 2: Relation of packages

4 Solution

My solution can be found at `https://github.com/i1i1/Innopolis_DE_assignment`.

I tried to make a program what won't depend on my variant, that is why `Exact` is derived of `ExactAbstract`, where user can implement only 3 functions in order to run another initial value problem. Because of that `main.py` file has only information which is needed to change variant, and it is only 24 lines of code.

```
class Exact(ExactAbstract):
    def derivative(x, y):
        return (1 + y/x) * m.log(1 + y/x) + y/x

    def exact(self, x):
        return (m.e ** (self.c * x) - 1) * x

    def set_constant(self, x0, y0):
        self.c = m.log(y0/x0 + 1) / x0
```

Figure 3: Implementation of my variant

Making program in python leaves even more space to work with. By using some commands like `eval` and `exec` which let you interprete code from string, there can be added functionality of specifying user-defined function from GUI.

Another great thing in design is that in order to create new numerical method user needs to write only one function `_next` which would calculate next point:

```
class EulerMethod(NumericalMethod):
    name = "Euler"

    def _next(self, h, x0, y0):
        return y0 + h * self.der(x0, y0)
```

Figure 4: Euler method implementation

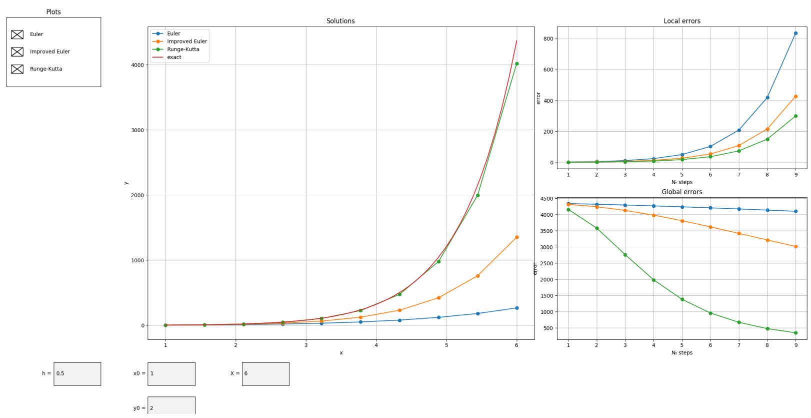

Figure 5: Errors with step 9 steps

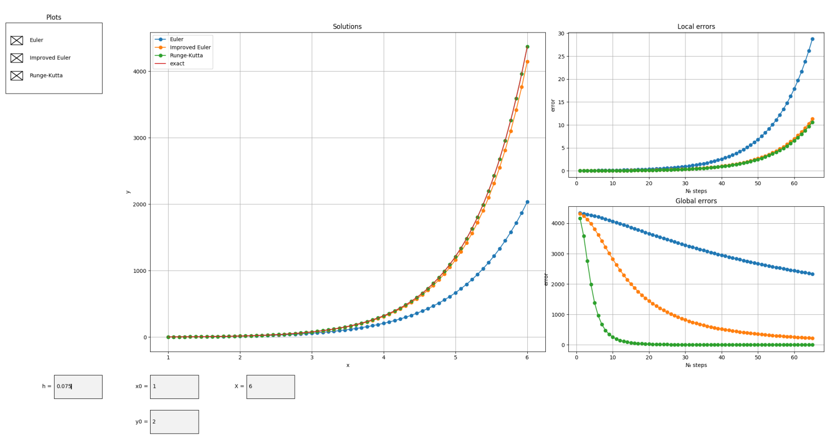

Figure 6: Errors with step 65 steps

5 Errors

As you can see on screenshots above you can see that local errors have some expanential form while global error from some point becomes linear.

Errors are calculated inside `NumericalMethod` class.

```
def get_lerr(self, x0, y0, X, h, exact):
    n = max(int((X-x0) // h), 2)
    x = np.linspace(x0, X, n)
    lerr = list()
    for i in range(1, n):
        y_num = self._next(h, x[i-1], exact(x[i-1]))
        y_exact = exact(x[i])
        lerr.append(abs(y_exact - y_num))
    return range(1, n), lerr
```

Figure 7: Local error function

```
def get_gerr(self, x0, y0, X, h, exact):
    n = max(int((X-x0) // h), 2)
    gerr = list()
    for i in range(2, n+1):
        x = np.linspace(x0, X, i)
        num = self.get_y(x, y0)
        ex = exact(x)
        gerr.append(abs(ex-num)[-1])
    return range(1, n), gerr
```

Figure 8: Global error function

6 Screenshots

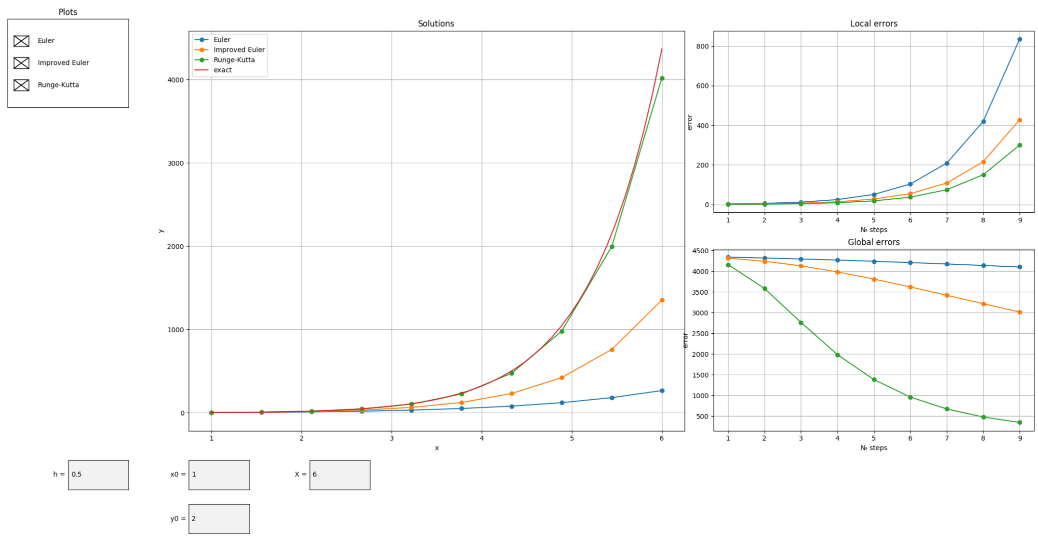

Figure 9: Original view

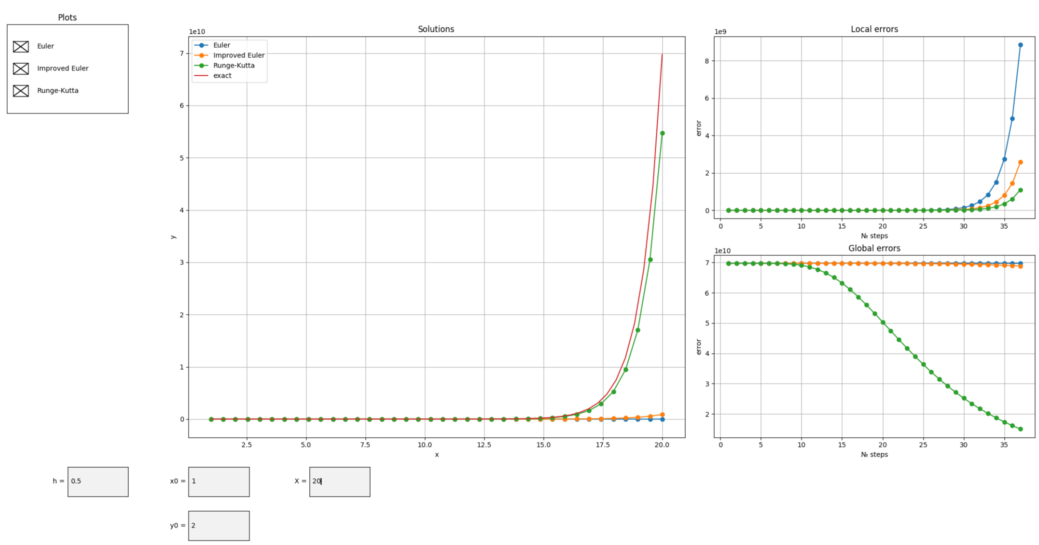

Figure 10: Increasing X

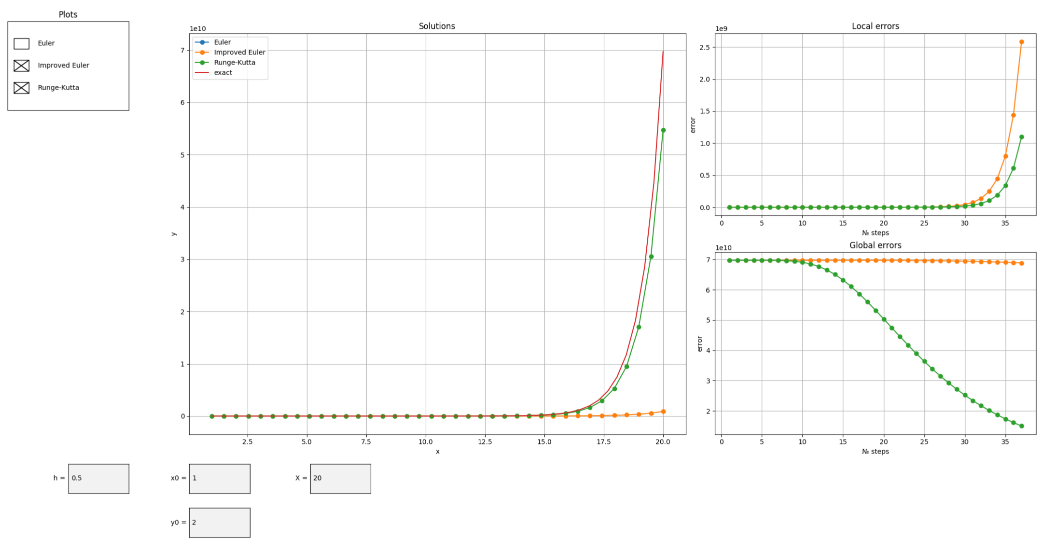

Figure 11: Removing Euler method