Dinmukhametov Salavat, BS17-7

# Differential Equations Computational Practicum Assignment Variant 5

https://github.com/slvt1/de_practicum

## Exact solution:

The differential equation is of the form

$y' = f(x, y)$ where $f(x, y) = cos(x) - y$

First of all let's solve $y' + y = 0$

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

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

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

$$y = C(x)e^{-x}$$

$$y' = C(x)'e^{-x} - C(x)e^{-x}$$

Now substitute it to initial d.e.

$$C(x)'e^{-x} - C(x)e^{-x} + C(x)e^{-x} = cos(x)$$

$$C(x)' = cos(x) * e^x$$

$$C(x) = \int cos(x) * e^x dx$$

Now let's use Integration by parts twice

$$C(x) = \int cos(x) * e^x dx = e^x(cos(x) + sin(x)) + \int cos(x) * e^x dx$$

$$\int cos(x) * e^x dx = 0.5e^x(cos(x) + sin(x)) + C$$

$$y = C(x) * e^{-x} = (0.5e^x(cos(x) + sin(x)) + C) * e^{-x} = Ce^{-x} + \frac{sin(x)}{2} + \frac{cos(x)}{2}$$

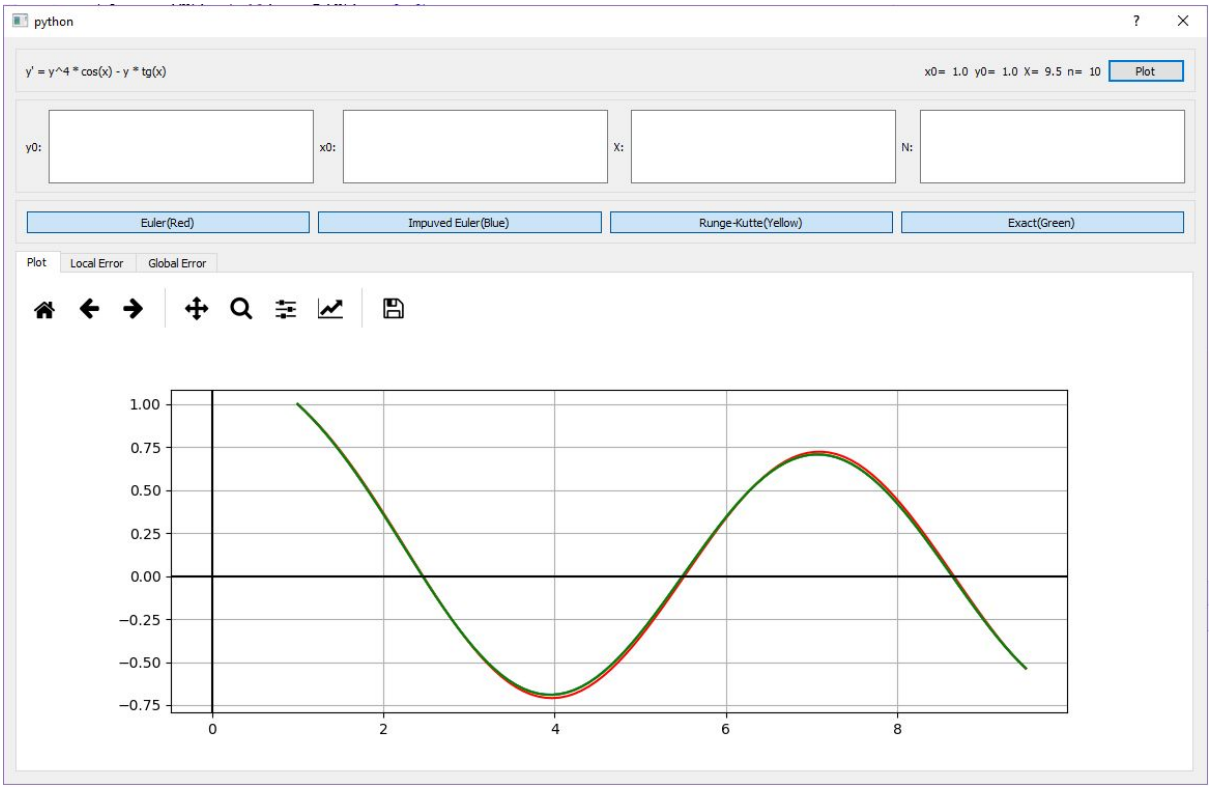
General solution - $y = Ce^{-x} + \frac{sin(x)}{2} + \frac{cos(x)}{2}$

Now for $y(1) = 1, C = 0.84$

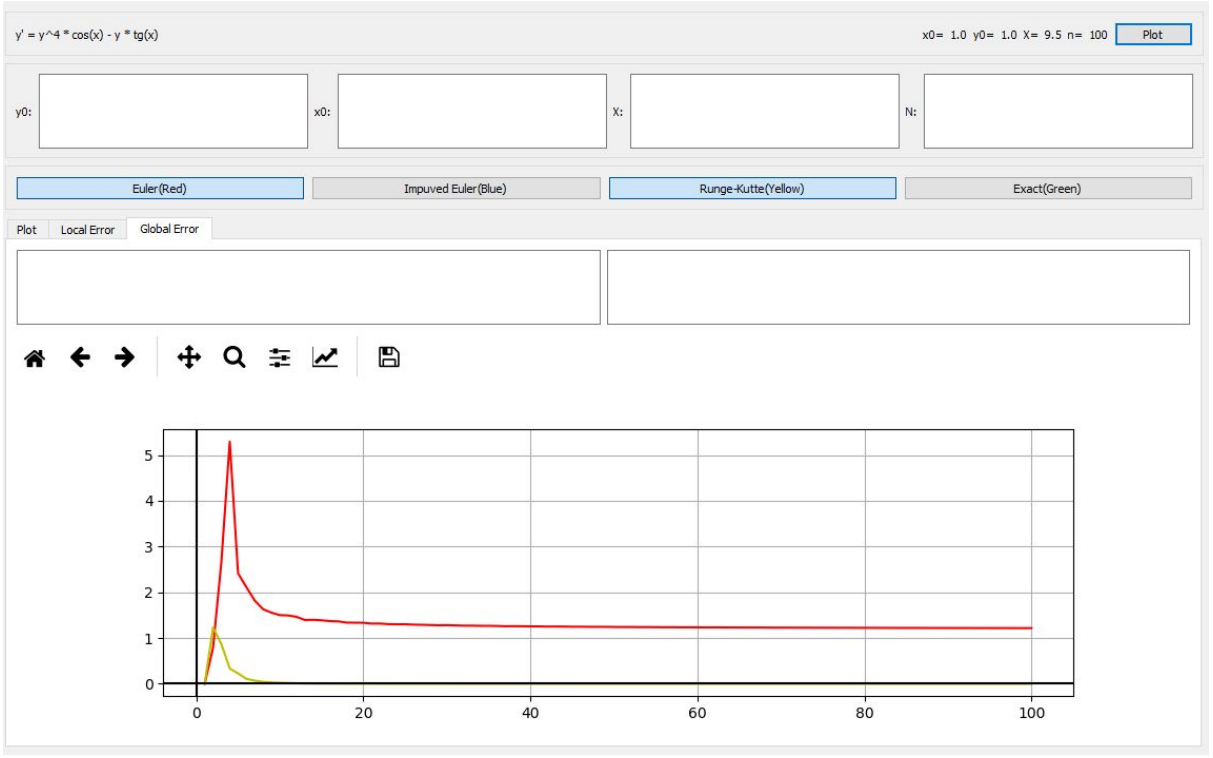
The exact solution when $y(1) = 1$

$$y = 0.84e^{-x} + \frac{sin(x)}{2} + \frac{cos(x)}{2}$$

My GUI has the following view

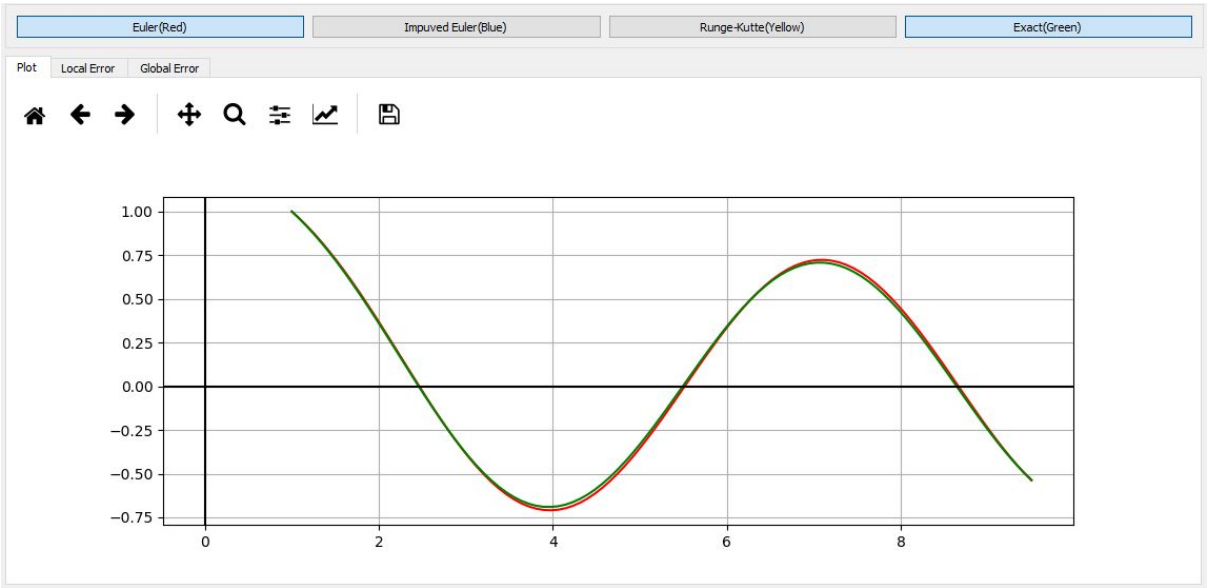



User may choose x0, y0, X,  n and for Global Error starting and finishing point, and methods that will be plotted.

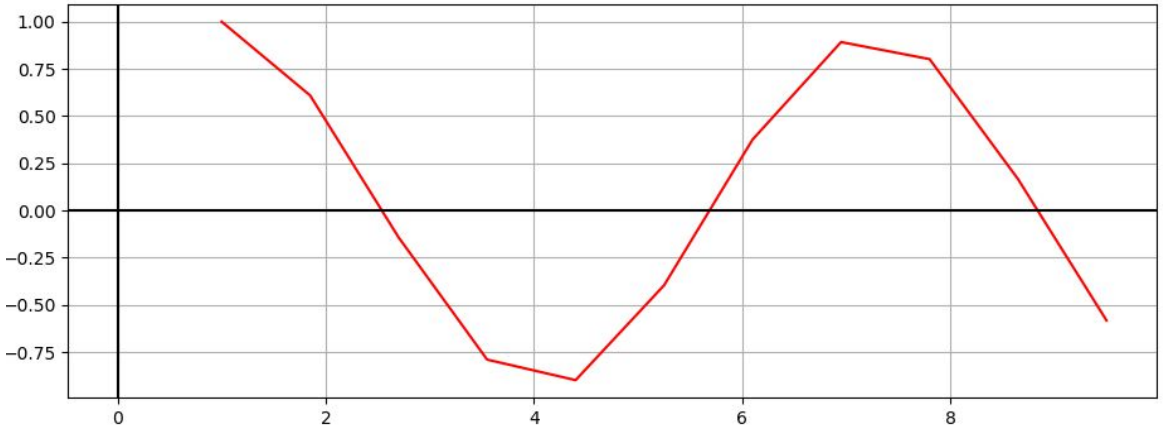


# Plots

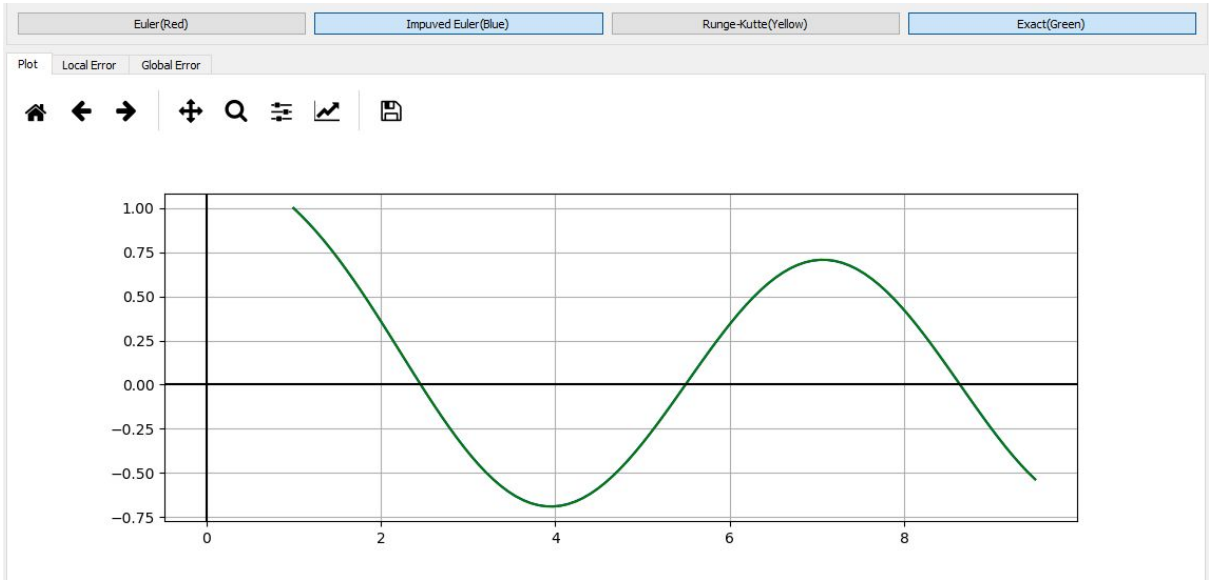
Euler(100 steps), Exact



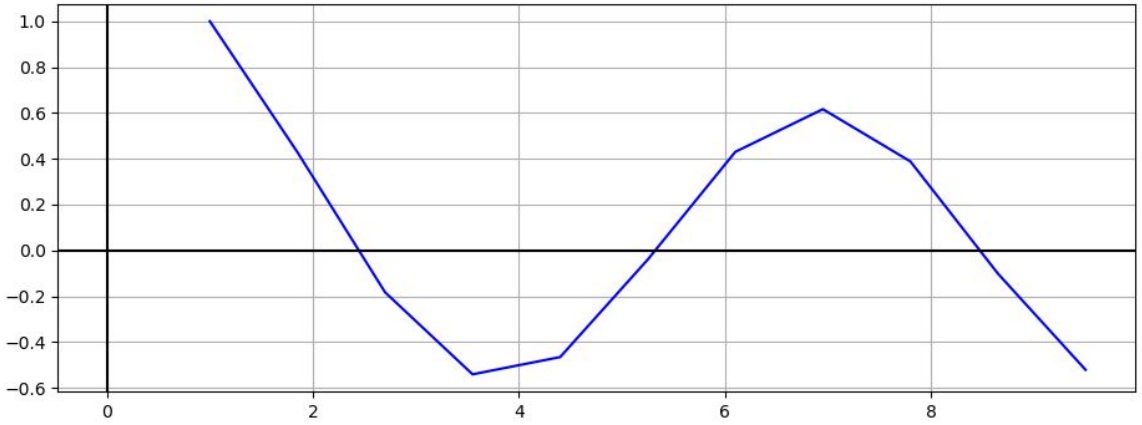

10 steps



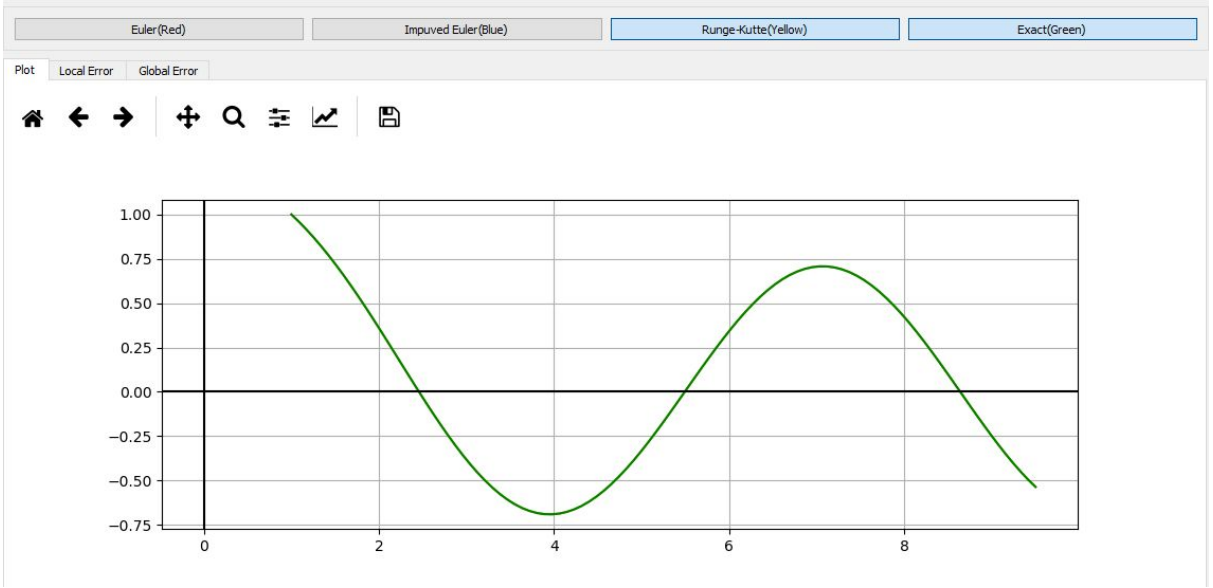

Improved Euler(100 steps), Exact



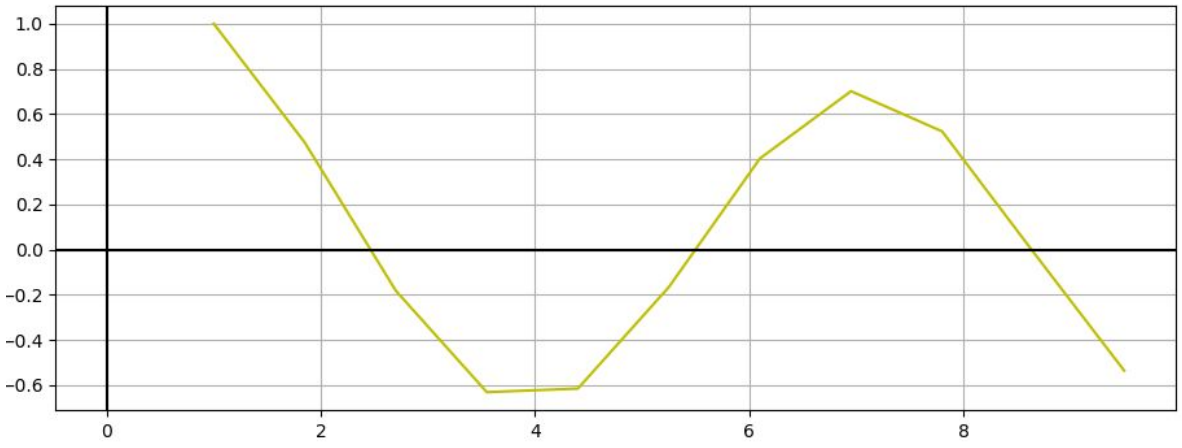

(10 steps)




Runge-Kutta(100 steps), Exact

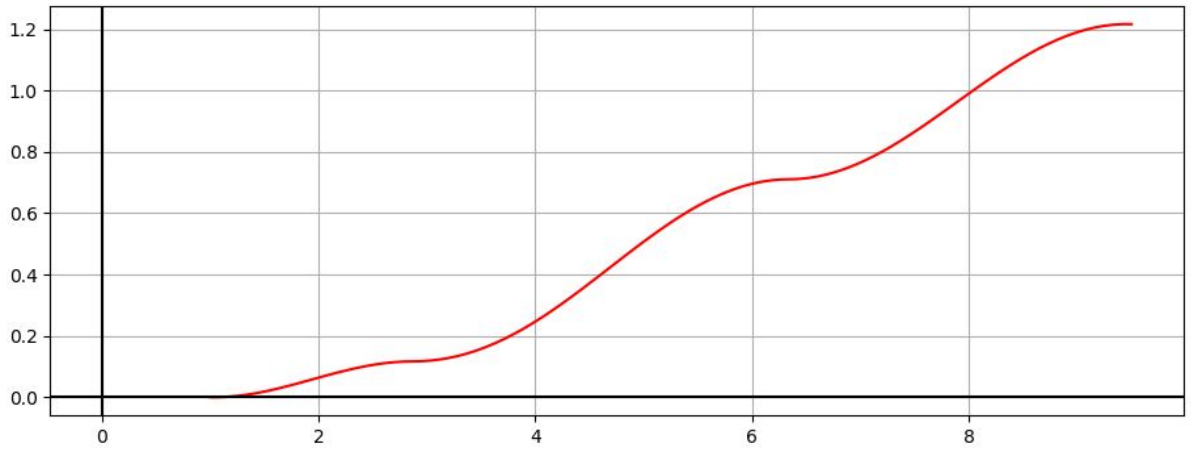



10 steps

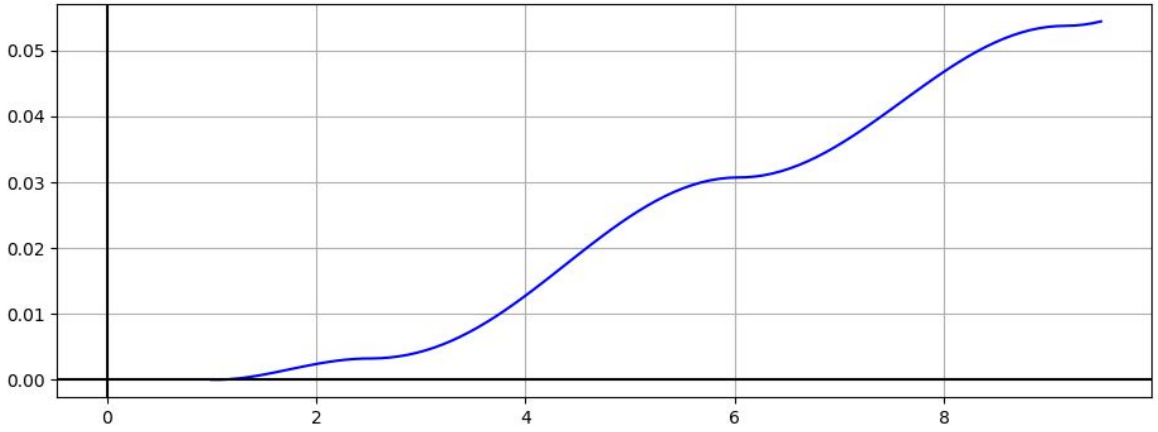

1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
2
4
6
8

# Errors

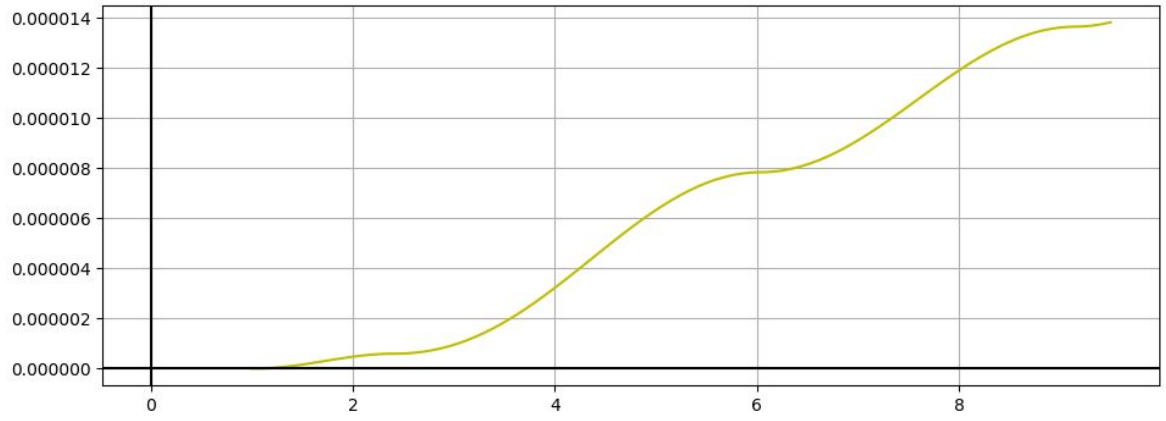
Euler



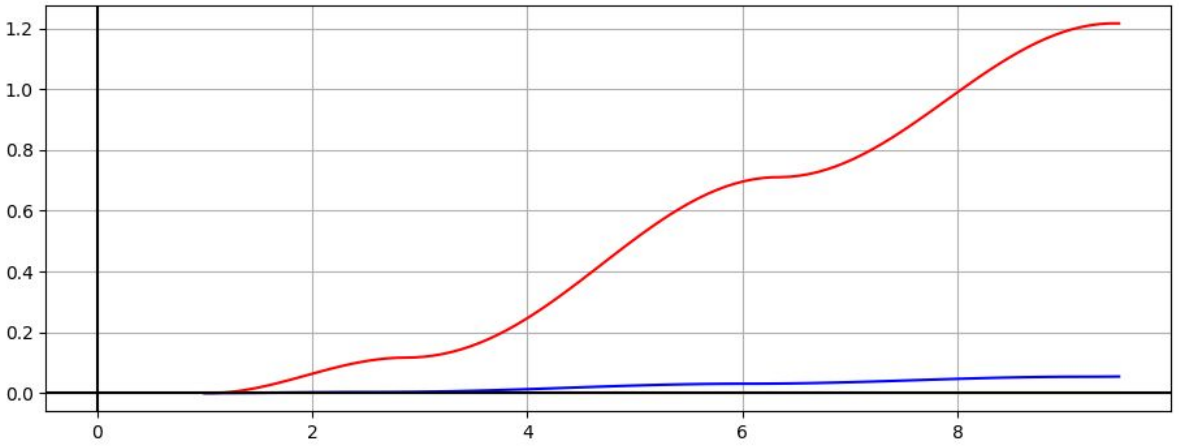

Improved Euler




Runge-Kutta

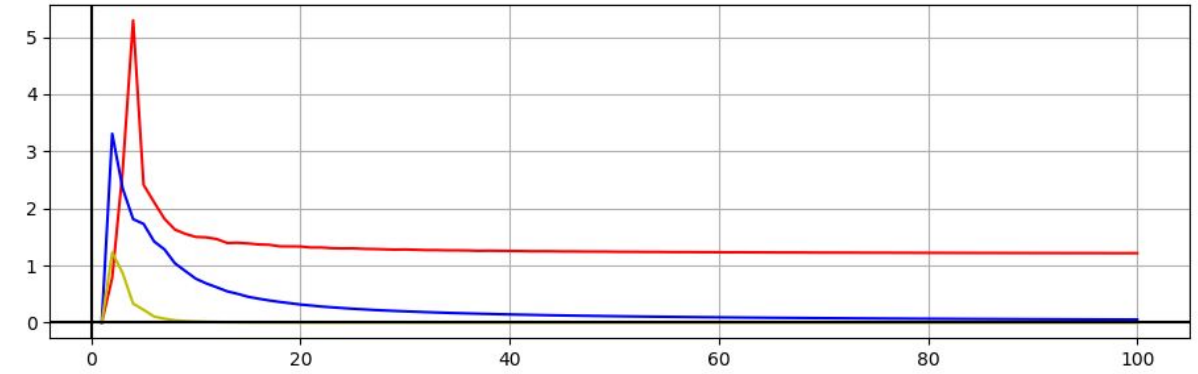



Together

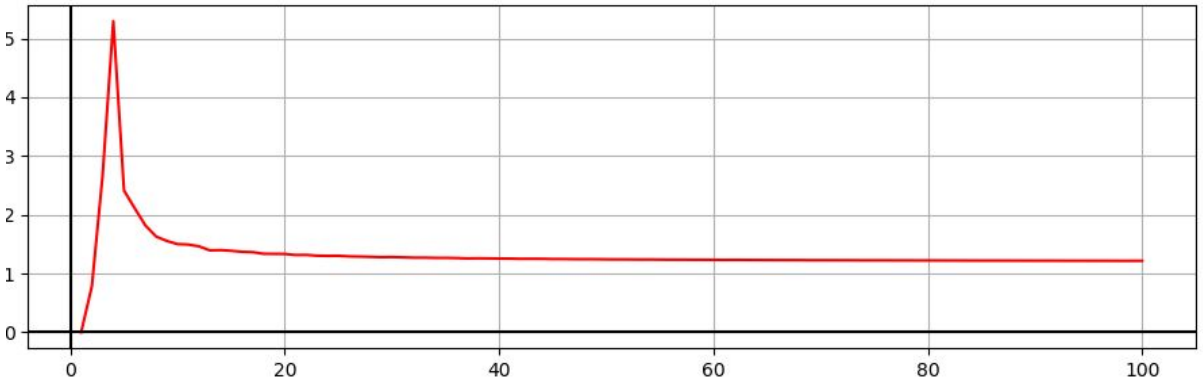

1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8

# Global errors: [1:100]

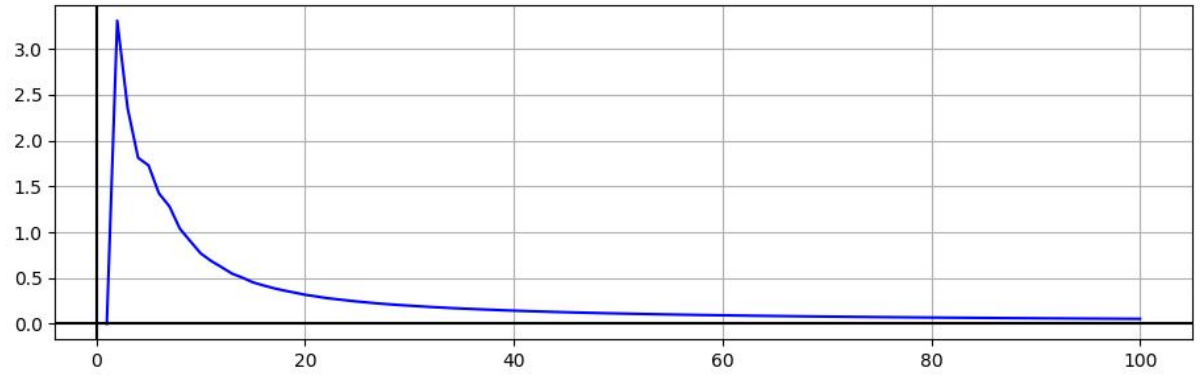
Euler(Red), Improved Euler(Blue), Runge-Kutta(Yellow)



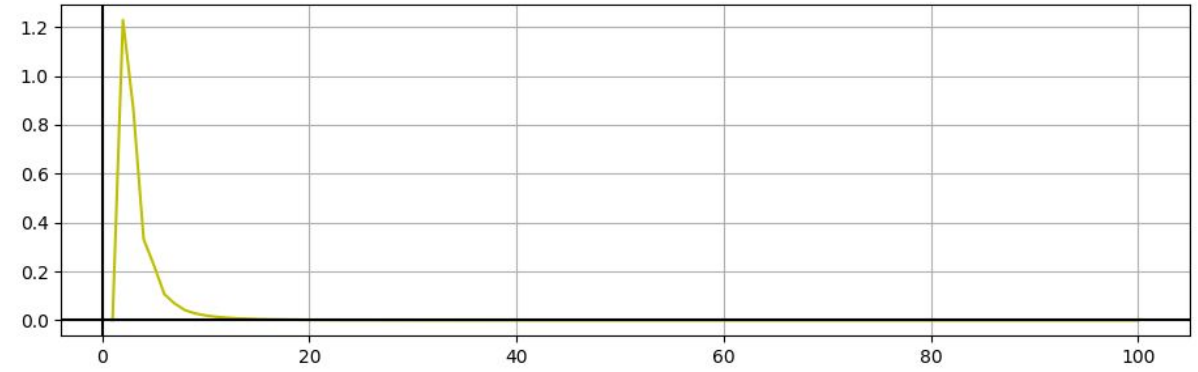

Euler



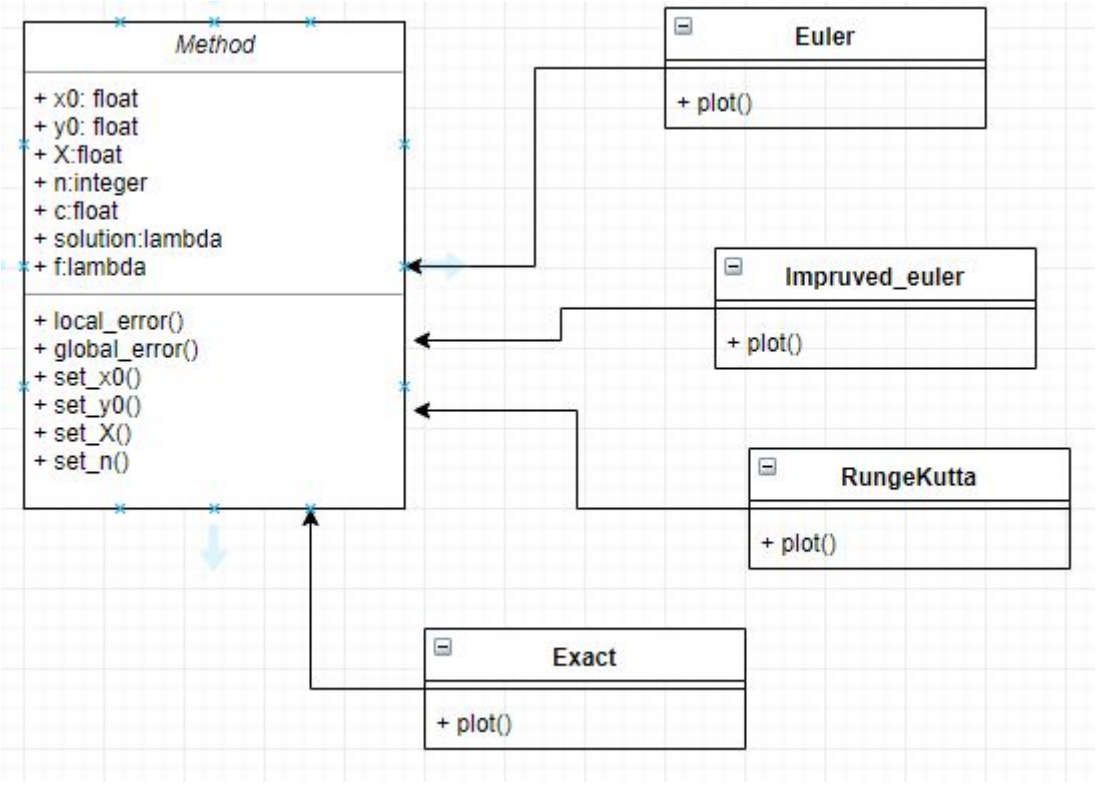

Improved Euler



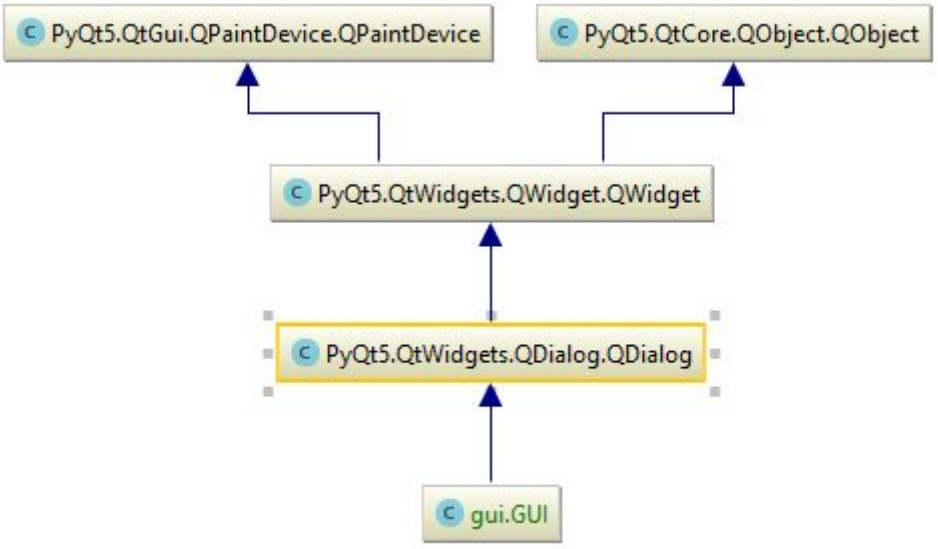

Runge-Kutta



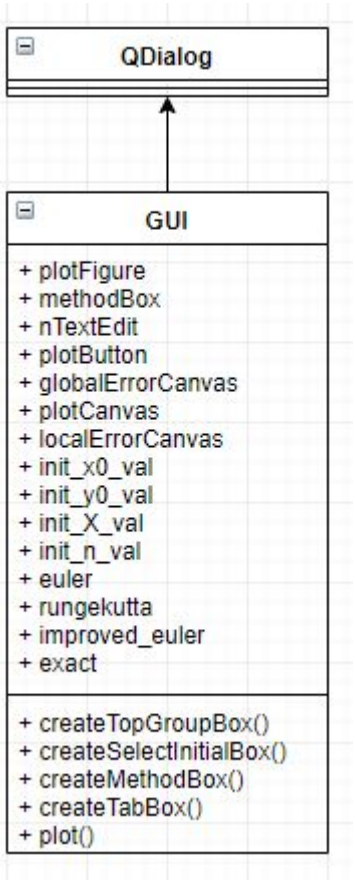
## Structure of the program




Class Method gets local error points, global error points, because calculating all of them the same, but each method has unique plot function realization.
I used Python 3.6 and libraries Mathplotlib 3.0, PyQt 5.11.3




gui.GUI is created GUI.




All create functions just creating GUI elements, plot function is draw all plots.

# Interesting parts of code

Given function and solution.

```
self.solution = lambda x: self.c*math.pow(math.e, -x) + math.sin(x)/2.0 + math.cos(x)/2.0
self.f = lambda x, y: math.cos(x) - y
```

Local  error function

```
def local_error(self):
    x, y = self.plot()
    error = [abs(self.solution(x[0]) - y[0])]
    for i in range(len(x[1:])):
        error.append(error[-1]+abs(self.solution(x[i]) - y[i]))
    return error
```

Global error function

```
def global_error(self, startn=1, finishn=100):
    errory, errorx = [], []
    last_h = self.h
    for i in range(startn, finishn+1):
        self.h = i
        self.step = (self.X - self.x0) / float(i)
        lerror = self.local_error()
        errory.append(lerror[-1])
        errorx.append(i)
    self.set_n(last_h)
    return  errorx, errory
```

Euler plot function

```
def plot(self):
    x = [self.x0]
    y = [self.y0]
    for i in range(int(self.h)):
        yn = y[-1] + self.step * self.f(x[-1], y[-1])
        x.append(x[-1]+self.step)
        y.append(yn)
    return x, y
```

Improved euler plot function

```
def plot(self):
    x = [self.x0]
    y = [self.y0]
    for i in range(int(self.h)):
        yn = y[-1] + self.step * (self.f(x[-1], y[-1]) + self.f(x[-1] + self.step,
                                  y[-1] + self.step * self.f(x[-1], y[-1]))) / 2.0
        x.append(x[-1]+self.step)
        y.append(yn)
    return x, y
```

Runge-Kutta plot function

```
def plot(self):
    x = [self.x0]
    y = [self.y0]
    k2 = lambda x, y, h: self.f(x + h / 2.0, y + (h / 2.0) * self.f(x, y))
    k3 = lambda x, y, h: self.f(x + h / 2.0, y + (h / 2.0) * k2(x, y, h))
    k4 = lambda x, y, h: self.f(x + h, y + h * k3(x, y, h))
    for i in range(int(self.h)):
        yn = y[-1] + (self.step / 6.0) * (self.f(x[-1], y[-1]) + k2(x[-1], y[-1], self.step) * 2.0 +
                                          k3(x[-1], y[-1], self.step) * 2.0 + k4(x[-1], y[-1], self.step))
        x.append(x[-1]+self.step)
        y.append(yn)
    return x, y
```

Exact plot function

```
def plot(self):
    x = [self.x0]
    y = [self.y0]
    for i in range(int(self.h)):
        y.append(self.solution(x[-1]+self.step))
        x.append(x[-1]+self.step)
    return x, y
```