Vadim Solovov
BS2-7 Group

Differential Equations
Computational Practicum Assignment

Variant 7

Variant 7

Given equation: $$y' = -x + \frac{y(2x+1)}{x}$$

This is First order Linear Nonhomogeneous differential equation, so we solve it accordingly by substituting: $y = uy_1$.

The solution: $$y = \frac{x}{2} + xe^{2x}c$$

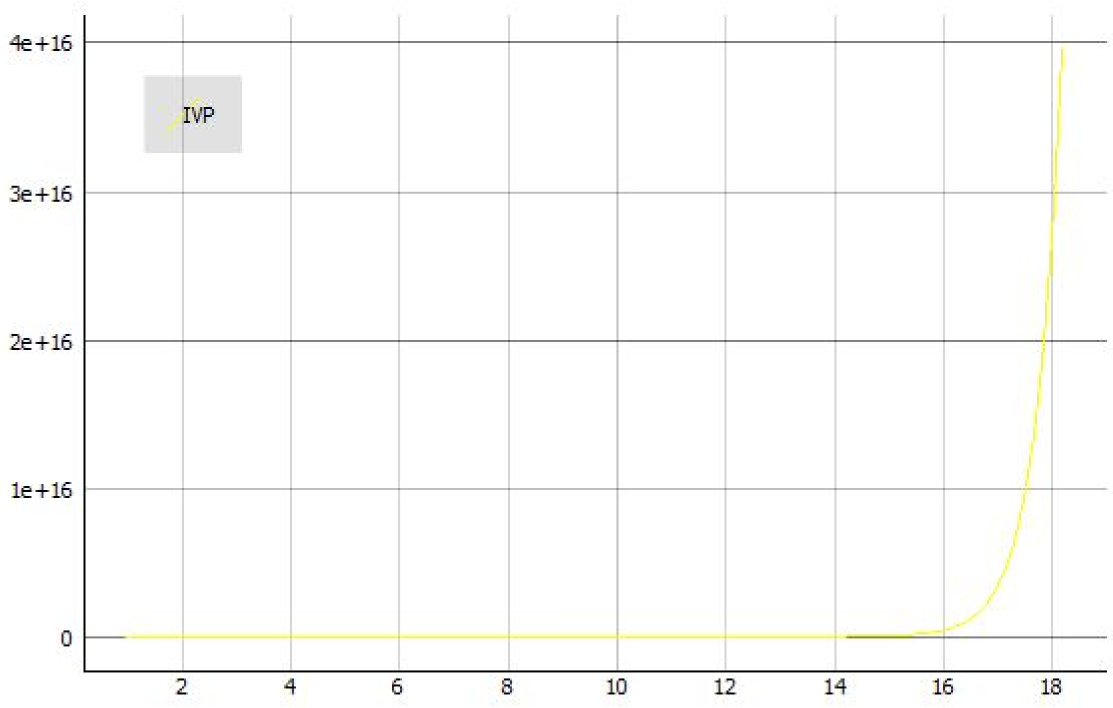
From this we can also get: $$c = \frac{2y - x}{2xe^{2x}}$$

The solution doesn't have points of discontinuity on given interval: $[1, 18.2]$

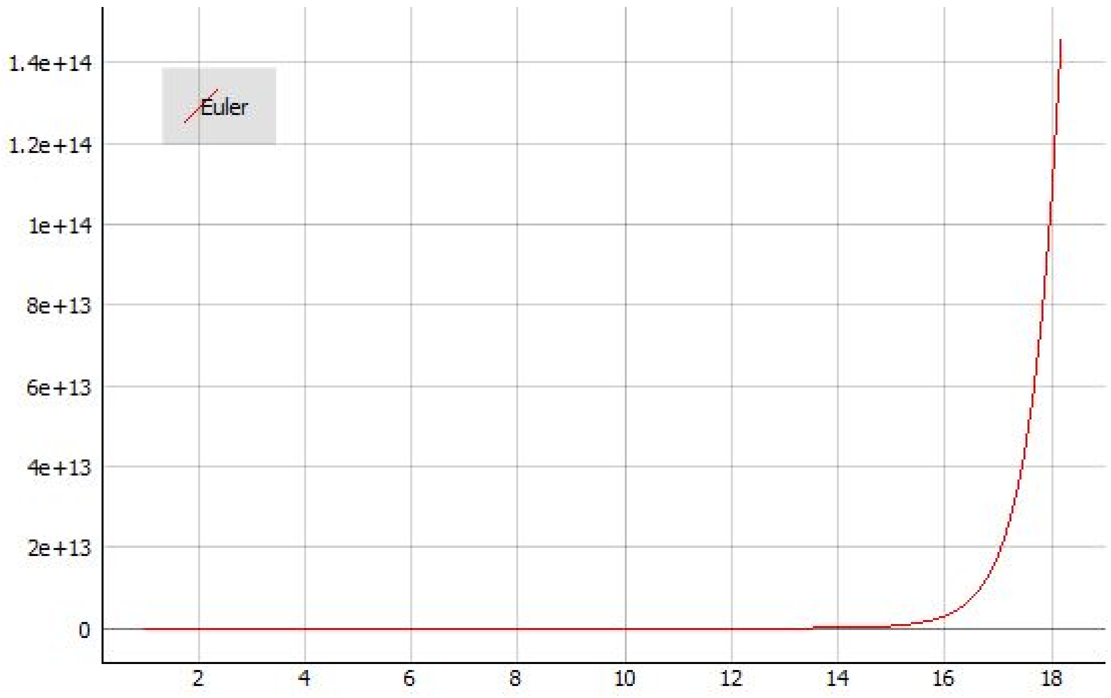
Plots of Numerical Solutions

Plots were made with x0 = 1; X= 18.2; y0 = 3; N = 100;

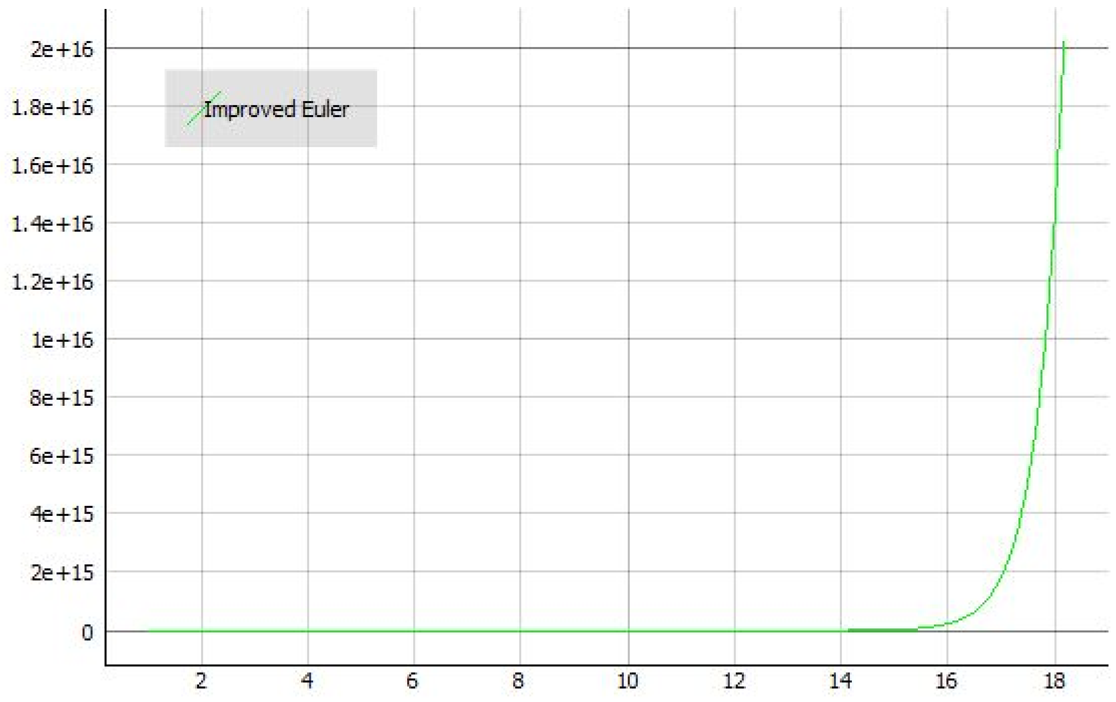
Plot for Initial Value Problem:



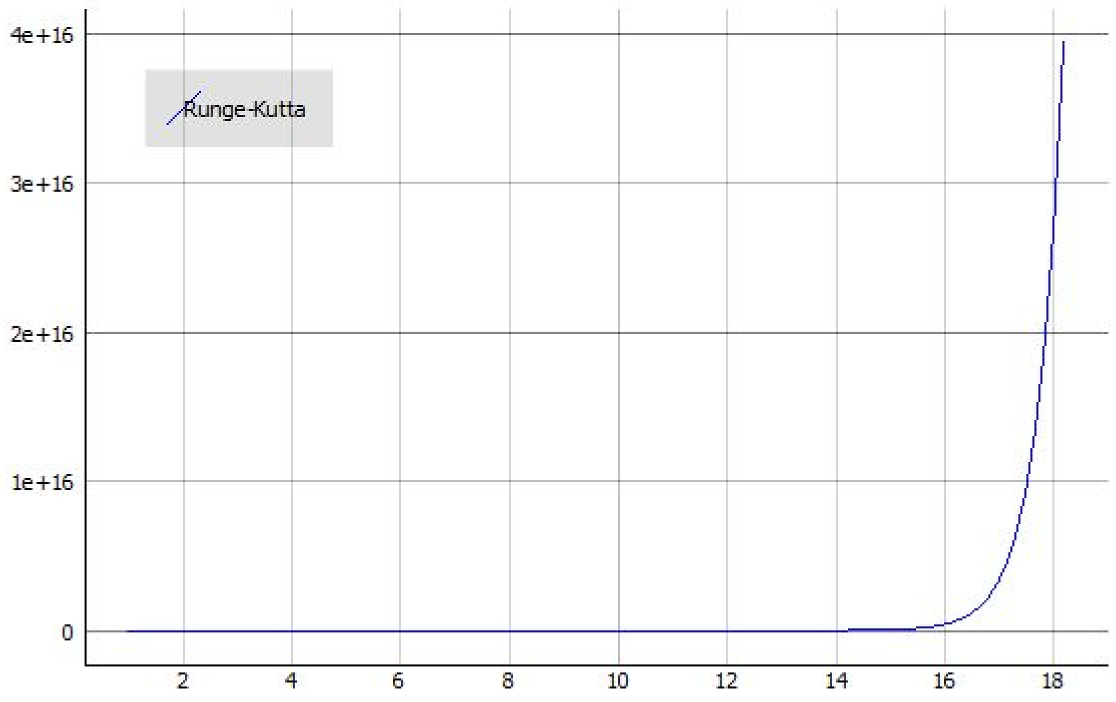
Plot of Euler's method:



Plot for Improved Euler’s method:



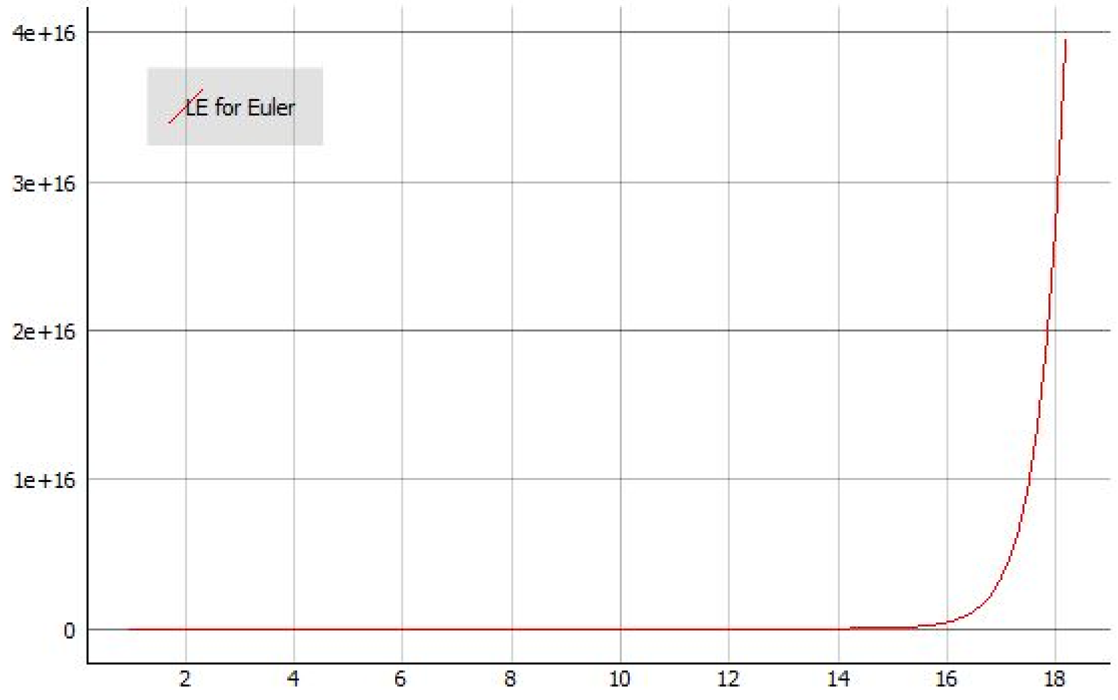
Plot for the Runge–Kutta method:



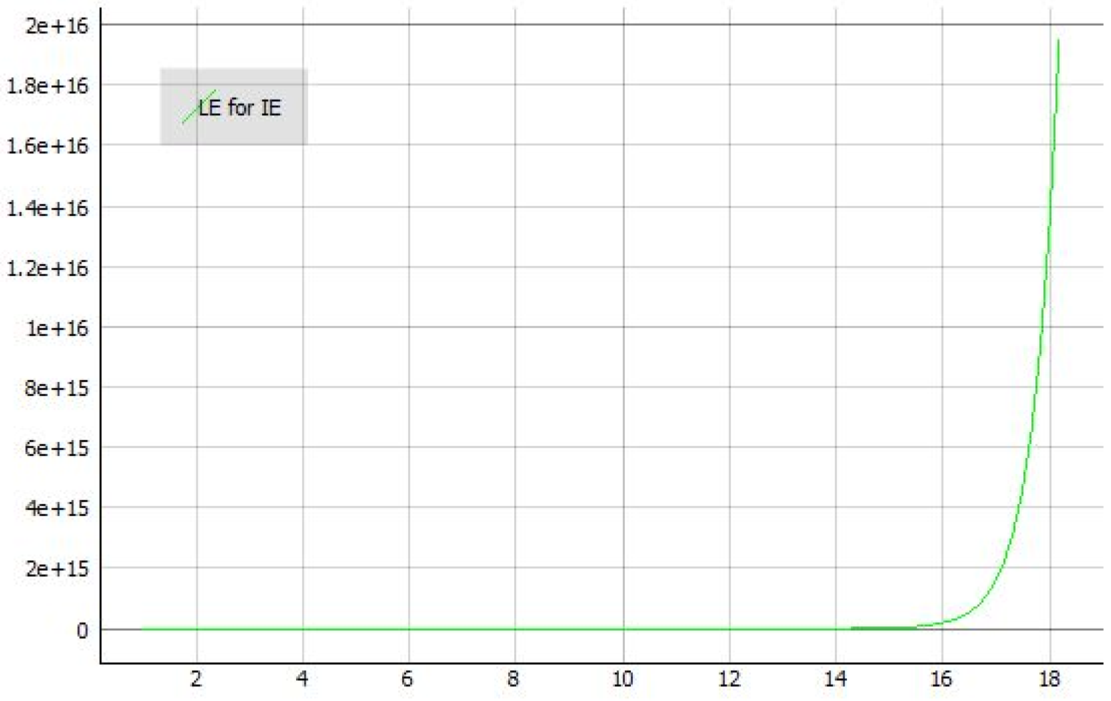
Plots for Local Errors of Numerical Methods:

Plots were made with x0 = 1; X= 18.2; y0 = 3; N = 100;

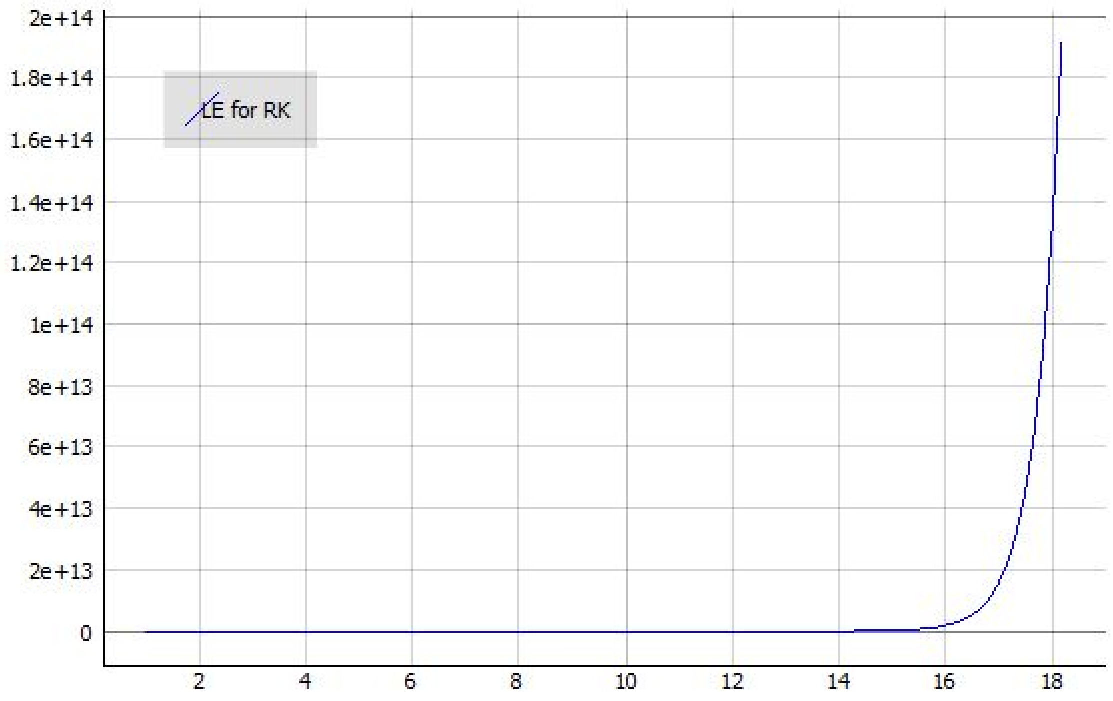
Plot of Local error for Euler’ method:



Plot of Local error for Improved Euler:

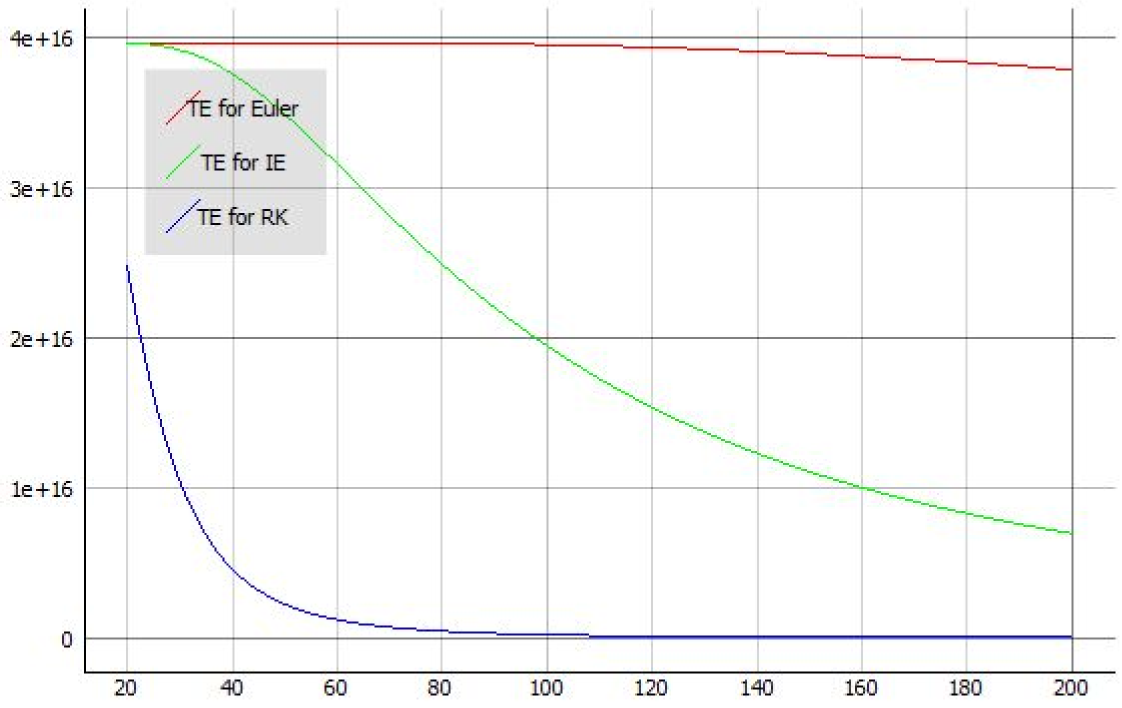


Plot of Local Error for Runge-Kutta:

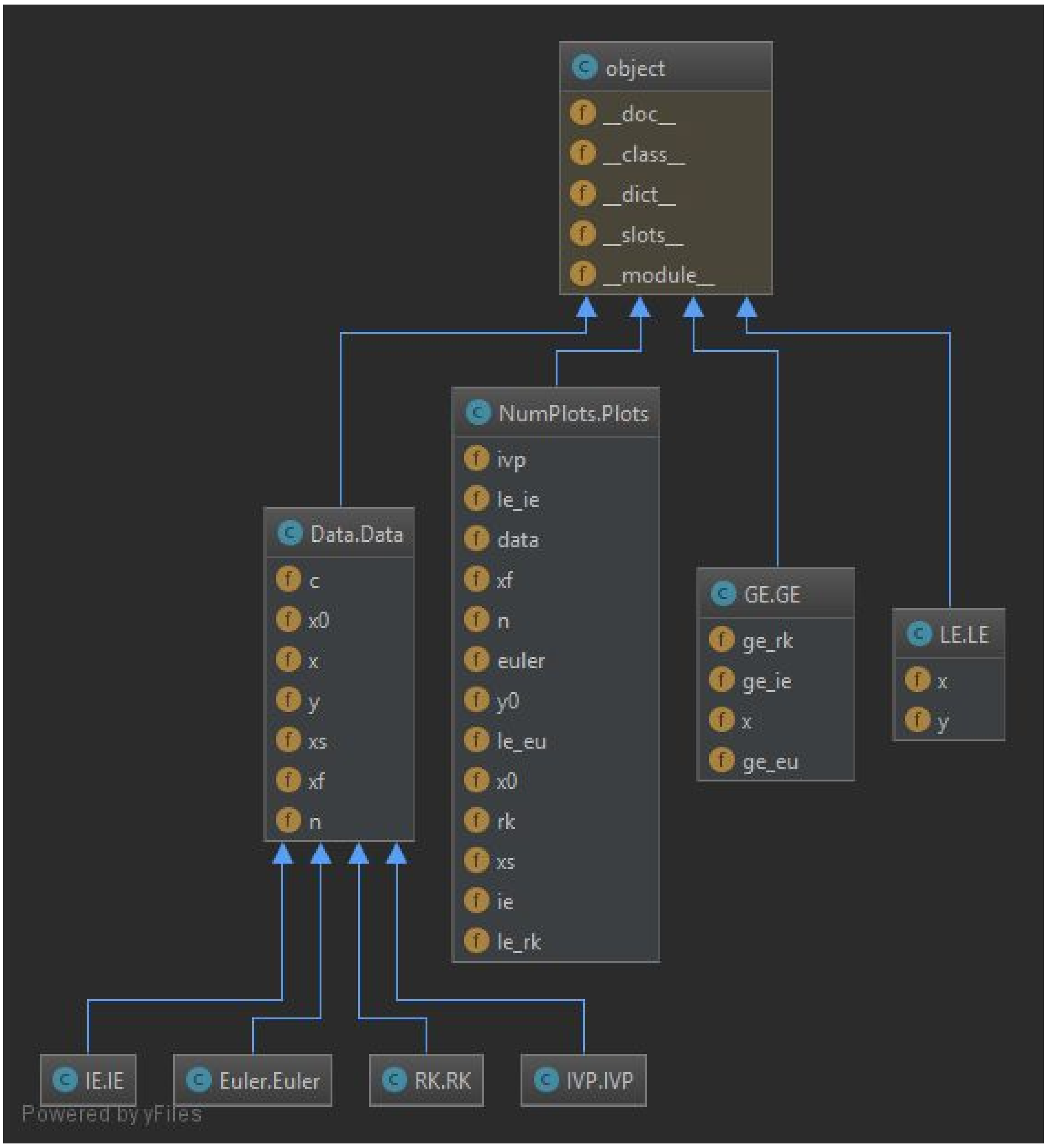


Plot of Total Approximation Error

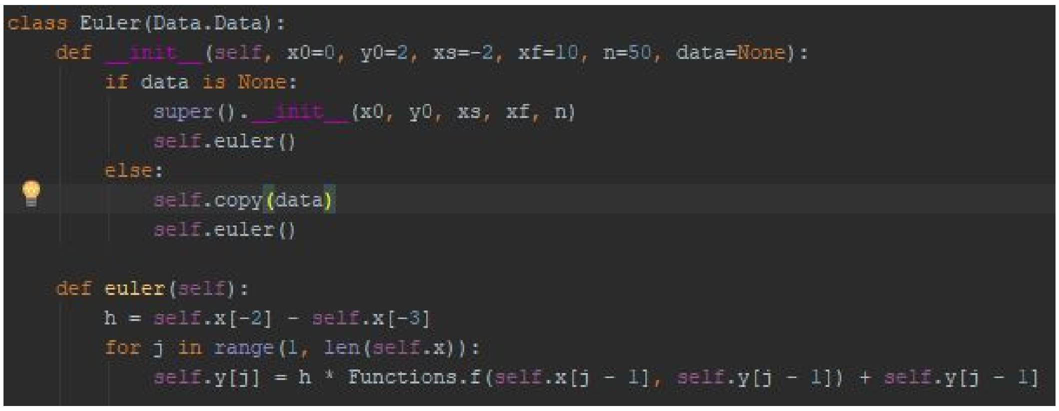
Plots was made for n0=20; N = 200;



Interesting parts from code

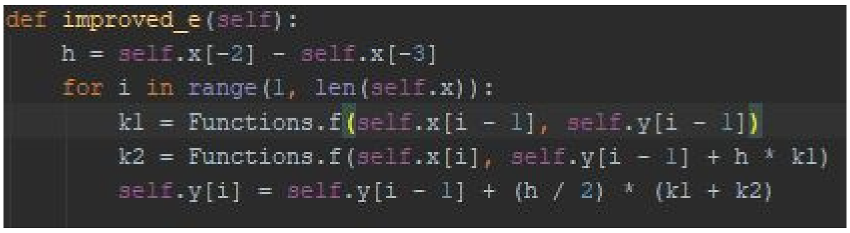


Euler's method:



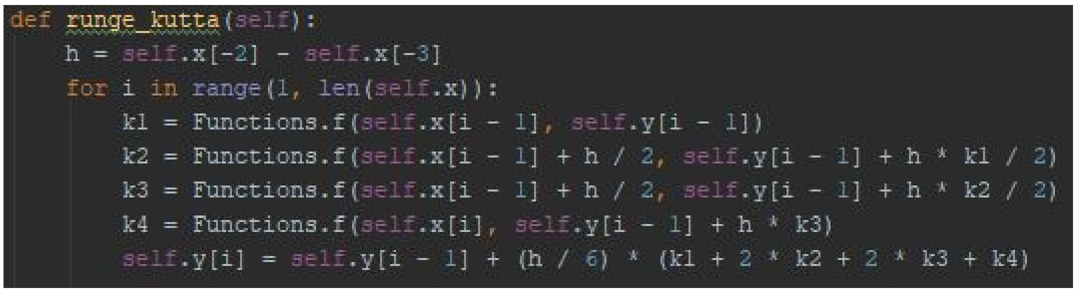
```
class Euler(Data.Data):
    def __init__(self, x0=0, y0=2, xs=-2, xf=10, n=50, data=None):
        if data is None:
            super().__init__(x0, y0, xs, xf, n)
            self.euler()
        else:
            self.copy(data)
            self.euler()

    def euler(self):
        h = self.x[-2] - self.x[-3]
        for j in range(1, len(self.x)):
            self.y[j] = h * Functions.f(self.x[j - 1], self.y[j - 1]) + self.y[j - 1]
```

Improved Euler (was implemented as in Euler, just now euler() is improved_e()):



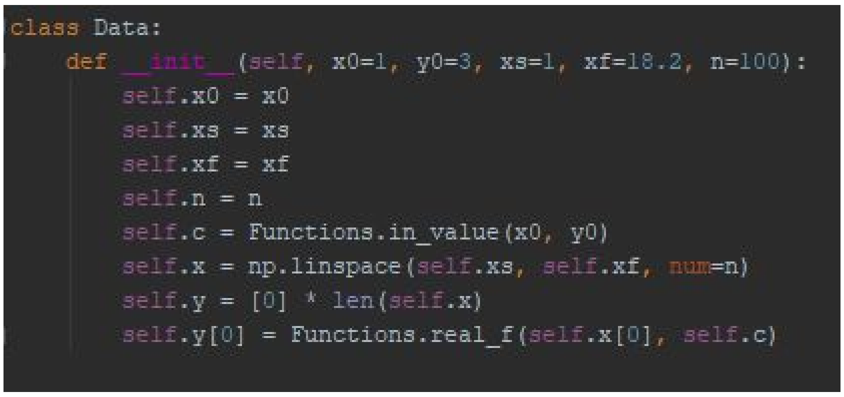
```
def improved_e(self):
    h = self.x[-2] - self.x[-3]
    for i in range(1, len(self.x)):
        k1 = Functions.f(self.x[i - 1], self.y[i - 1])
        k2 = Functions.f(self.x[i], self.y[i - 1] + h * k1)
        self.y[i] = self.y[i - 1] + (h / 2) * (k1 + k2)
```

Runge-Kutta method:



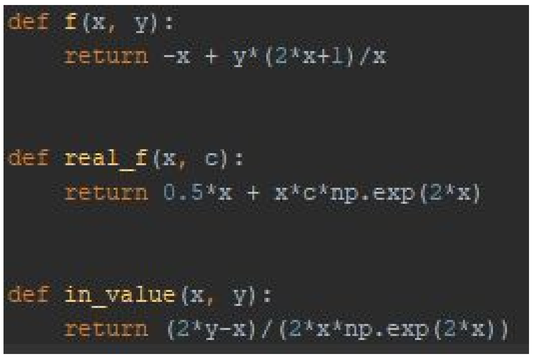
```
def runge_kutta(self):
    h = self.x[-2] - self.x[-3]
    for i in range(1, len(self.x)):
        k1 = Functions.f(self.x[i - 1], self.y[i - 1])
        k2 = Functions.f(self.x[i - 1] + h / 2, self.y[i - 1] + h * k1 / 2)
        k3 = Functions.f(self.x[i - 1] + h / 2, self.y[i - 1] + h * k2 / 2)
        k4 = Functions.f(self.x[i], self.y[i - 1] + h * k3)
        self.y[i] = self.y[i - 1] + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
```

Class Data I used for storing lists X and Y and other information about plot:



```
class Data:
    def __init__(self, x0=1, y0=3, xs=1, xf=18.2, n=100):
        self.x0 = x0
        self.xs = xs
        self.xf = xf
        self.n = n
        self.c = Functions.in_value(x0, y0)
        self.x = np.linspace(self.xs, self.xf, num=n)
        self.y = [0] * len(self.x)
        self.y[0] = Functions.real_f(self.x[0], self.c)
```

In Functions I have equations that may be needed in IVP, Data, Euler etc.:

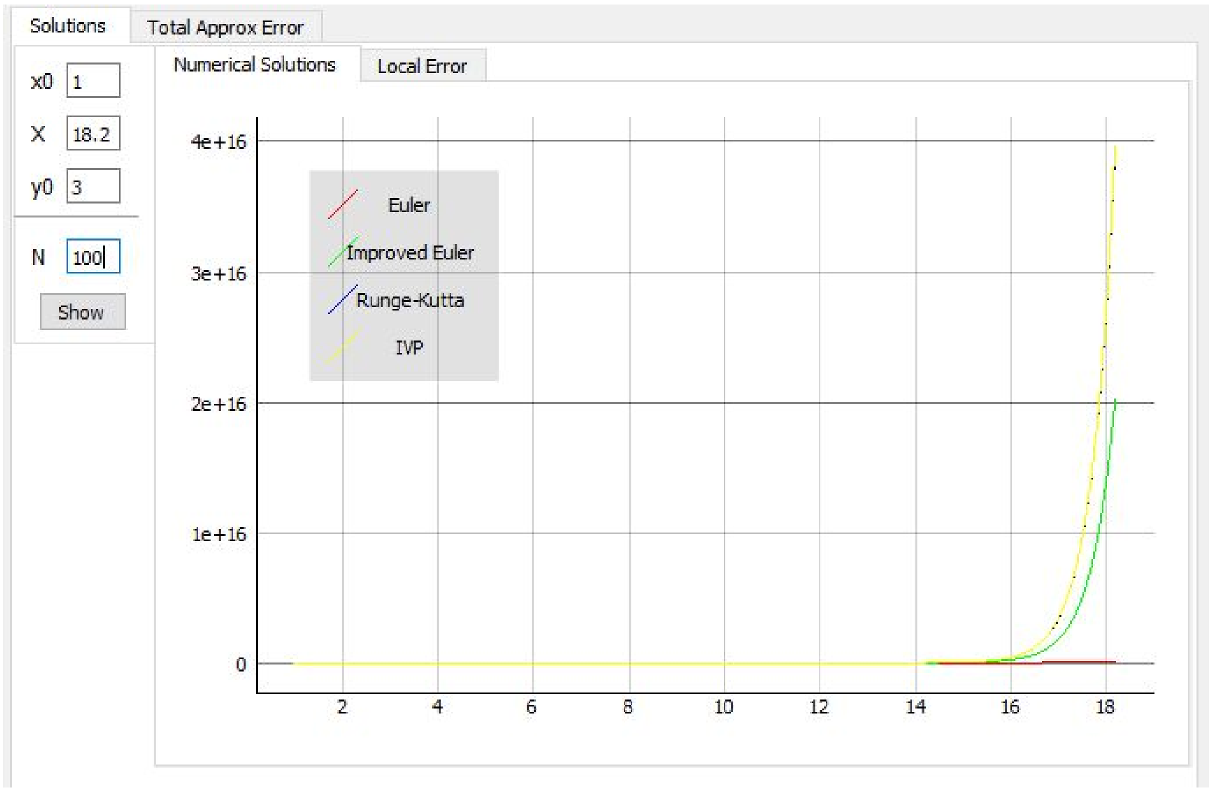


```
def f(x, y):
    return -x + y*(2*x+1)/x


def real_f(x, c):
    return 0.5*x + x*c*np.exp(2*x)


def in_value(x, y):
    return (2*y-x)/(2*x*np.exp(2*x))
```

NumPlots.Plots and TE are classes I use for outputting plots in GUI.

GUI

Tab for Plots of Numerical Solutions:



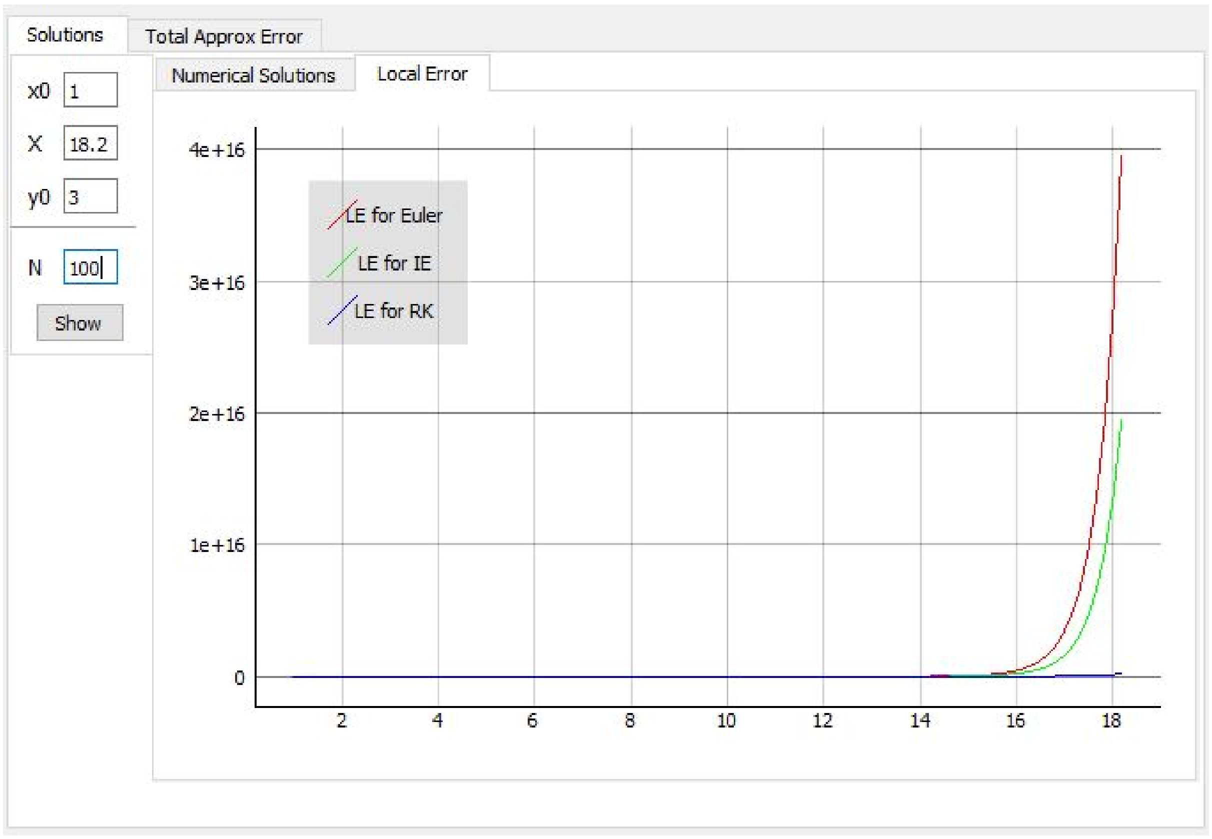
x0 1

X 18.2

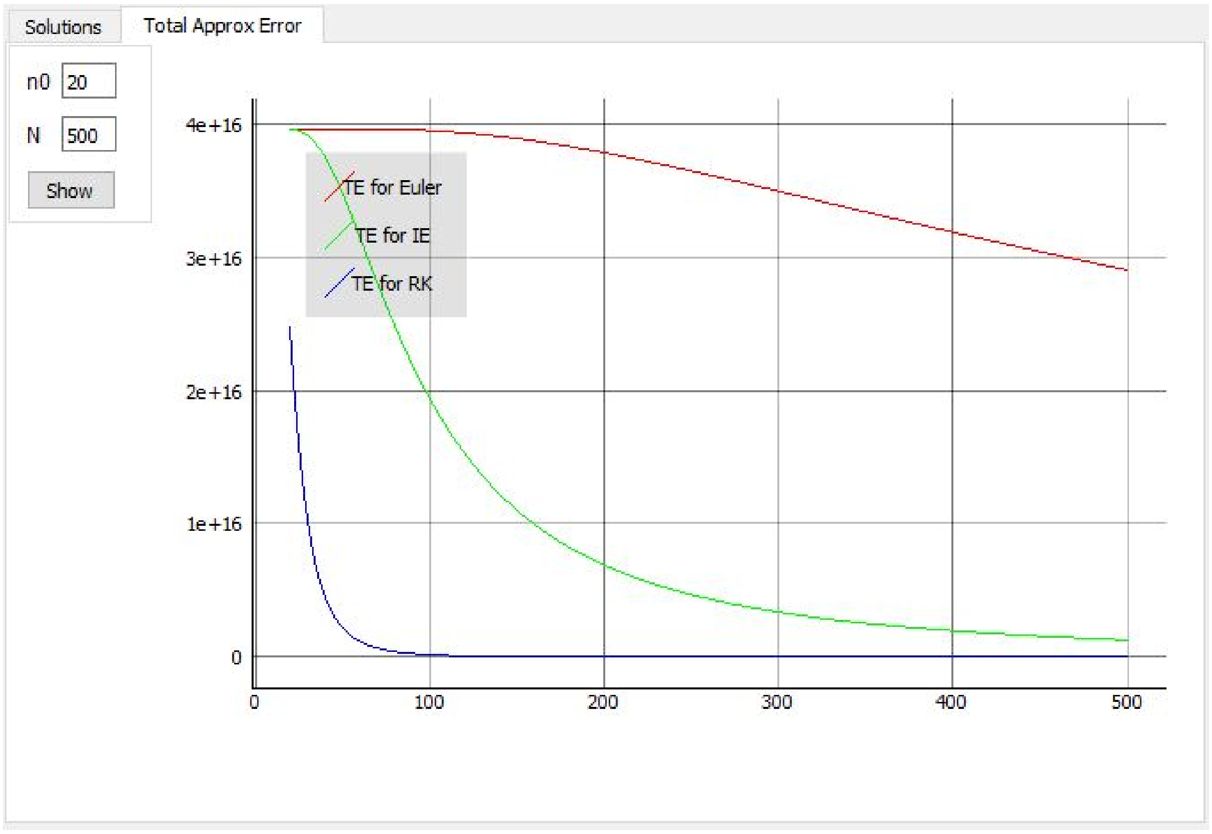
y0 3

N 100

Tab for Local Error Plots:



Tab for Total Approximation Error:


Solutions
Total Approx Error
n0 20
N 500
Show
TE for Euler
TE for IE
TE for RK
4e+16
3e+16
2e+16
1e+16
0
0
100
200
300
400
500