Differential Equations Assignment Report

Student: Lukyanchikova Elena
Group: B17-02
Variant: #12
Link to the solution: github.com/elukyanchikova/de_numerical_methods
Used PL: Python 3.6

Exact Solution of the Initial Value Problem

$$\begin{cases} y' = xy^2 + 3xy \\ y(3) = 1 \\ x \in [0,\ 5.5] \end{cases}$$

This is a Bernoulli equation:

$y' - xy^2 = 3xy \mid : y^2$

Use substitution:

$\frac{y'}{y^2} - x = \frac{3x}{y} \ |z = \frac{1}{z},\ z' = -\frac{1}{y^2}$

$-z' - 3xz = x$

Use Method of variation of parameter:

$-z' - 3xz = 0$

$\int \frac{dz}{z} = -\int 3xdx$

$z = C_1 e^{(-3x^2)/2}$

$z' = C_1' e^{(-3x^2)/2} - 3xC_1 e^{(-3x^2)/2}$

Use Method of variation of parameter:

$-z' - 3xz = 0$

$\int \frac{dz}{z} = -\int 3xdx$

$z = C_1 e^{(-3x^2)/2}$

$z' = C_1' e^{(-3x^2)/2} - 3xC_1 e^{(-3x^2)/2}$

$-C_1' e^{(-3x^2)/2} + 3xC_1 e^{(-3x^2)/2} - 3xC_1 e^{(-3x^2)/2} = x$

$-C_1' e^{(-3x^2)/2} = x$

$C_1 = C_2 - \frac{e^{(-3x^2)/2}}{3}$

The solution:

$$z = \frac{1}{y} = C_2 e^{(-3x^2)/2} - \frac{1}{3}$$

$$y = \frac{1}{C_2 e^{(-3x^2)/2 - \frac{1}{3}}}$$

The Feasible region:

$$C_2 e^{(-3x^2)/2} - \frac{1}{3} \neq 0$$

$$x \neq \sqrt{\frac{2}{3} \ln |3C_2|}$$

According to IVP (x0,y0):

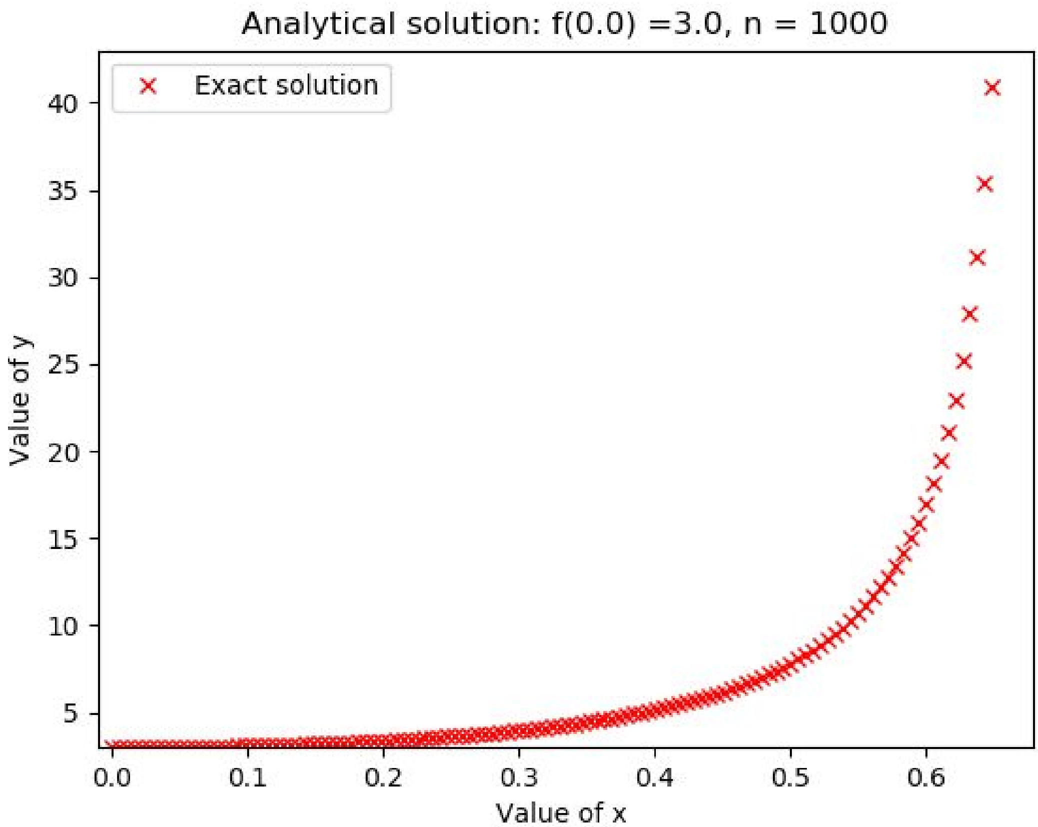
$$C_2 = const = \frac{3+y}{3y} e^{(3x^2)/2}$$

Exact Solution for (x0, y0) = (0, 3):

$$C_2 = 0.6666666666666666$$

$$y = \frac{1}{0.6666666666666666 e^{(-3x^2)/2 - \frac{1}{3}}}$$

Note: do not round the constant. Otherwise, it will decrease the presicion of the computation.



Structure of the program

Solution is presented in one python file.

It consist of implementations of 3 Numerical methods: Euler, Euler Improved and Runge-Kutta:
def euler_method(x, y, h)
def euler_improved_method(x, y, h)
def runge_kutta_method(x, y, h)

Python file also includes method for solving Initial Value problem
def exact_solution(x_current, const)

There are 2 auxiliary methods for calculating constant corressponding to the initial conditions and point x at which the solution does not exist (asymptote)
def calculate_const(x0, y0)
def calculate_asymptote(c)

The user should enter to the console:

- (x0,y0) - IVP conditions
- xf - upper-bound of [x0,xf]
- n - number of subintervals/steps (affects precision)
  At the main body of the program 4 arrays are creared and being filled with y values corresponding to each of the Solutions: numerical and exact.

The Global Truncation Error for each of Numerical method is being computed during execution.

Then program plots 5 graphs:

- Exact (Analtical) Solution
- Euler Method Approximation (in comparison with Exact)
- Euler Improved Method Approximation (in comparison with Exact)
- Runge-Kutta Approximation (in comparison with Exact)
- Analytical and Numerical Solutions at one graph
- Global Truncation Error for all Numerical Methods

Euler Method

Euler Method - first-order numerical procedure for solving ordinary differential equations with a given initial value.

$$\begin{cases} y'(t) = f(y(t), t) \\ y(t_0) = y_0 \end{cases}$$

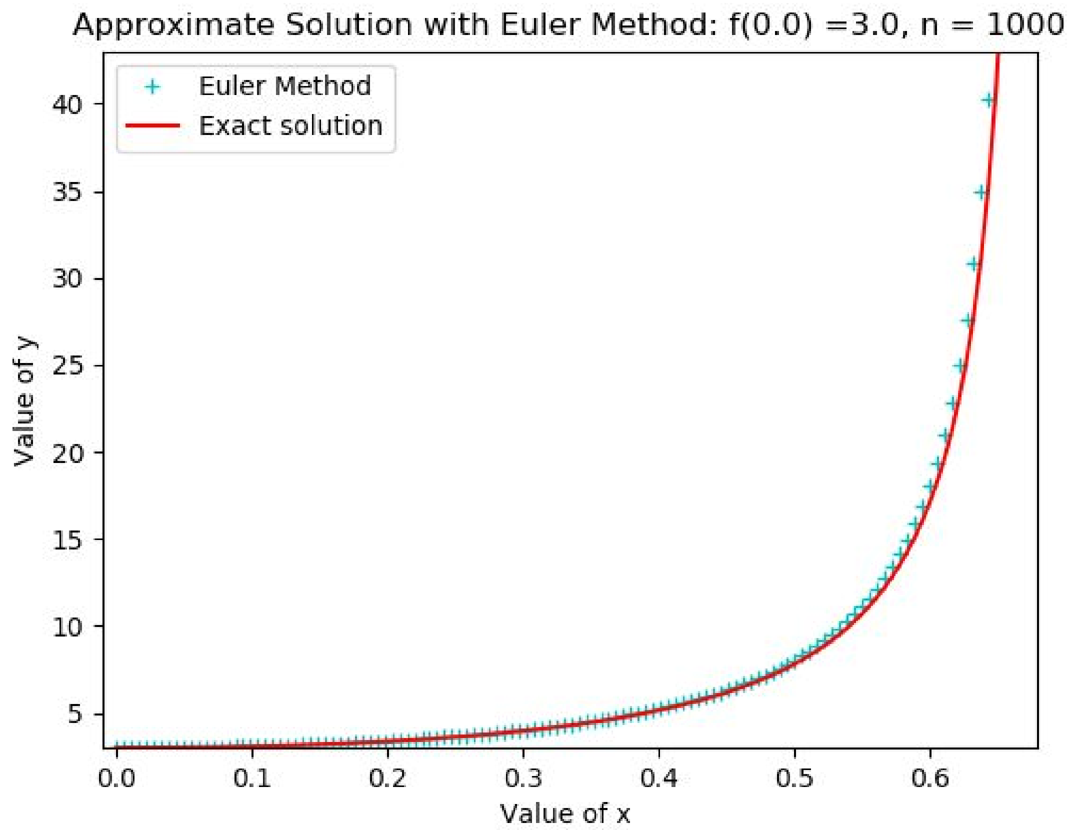
Choose a value h for the size of every step and set $t_n = t_0 + nh$

At the step t_n to $t_{n+1} = t_n + h$:
$y_{n+1} = y_n + hf(t_n, y_n)$

Implemented method:

```
'''method for computing Approximate values using Euler method'''


def euler_method(x, y, h):
    y_n = h * f(x, y) + y
    return y_n
```



Euler Improved Method

Euler Improved Method - second-order numerical procedure for solving ordinary differential equations with a given initial value.

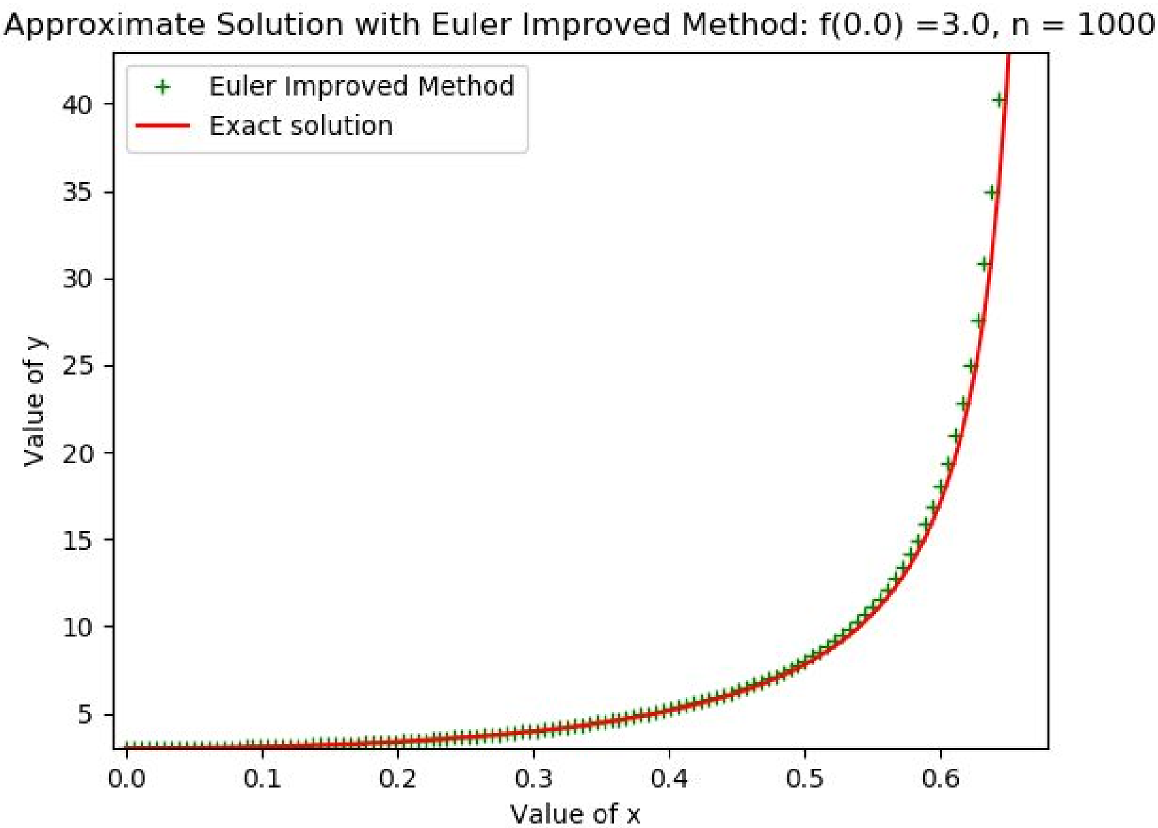
Idea: modify Euler Method such that the *segments*(connect y_i and y_{i+1} at the graph) should be *parallel* to the *tangent* which are drawn to the graph of the function $y' = f(x, y)$ in the middle (point $\frac{x_{n+1} - x_n}{2}$) of the split interval. This should improve the quality of the approximation.

$$y_{i+1} = y_i + \frac{h}{2}(f(x_i, y_i) + f(x_{i+1}, y_i + hf(x_i, y_i)))$$

Implemented method:

```python
'''method for computing Approximate values using Euler improved method'''


def euler_improved_method(x, y, h):
    temp = y + h * f(x, y)
    y_n = y + (h / 2) * (f(x, y) + f(x + h, temp))
    return y_n
```



Runge-Kutta Method

Runge-Kutta Method of the forth order - forth-order numerical procedure for solving ordinary differential equations with a given initial value.

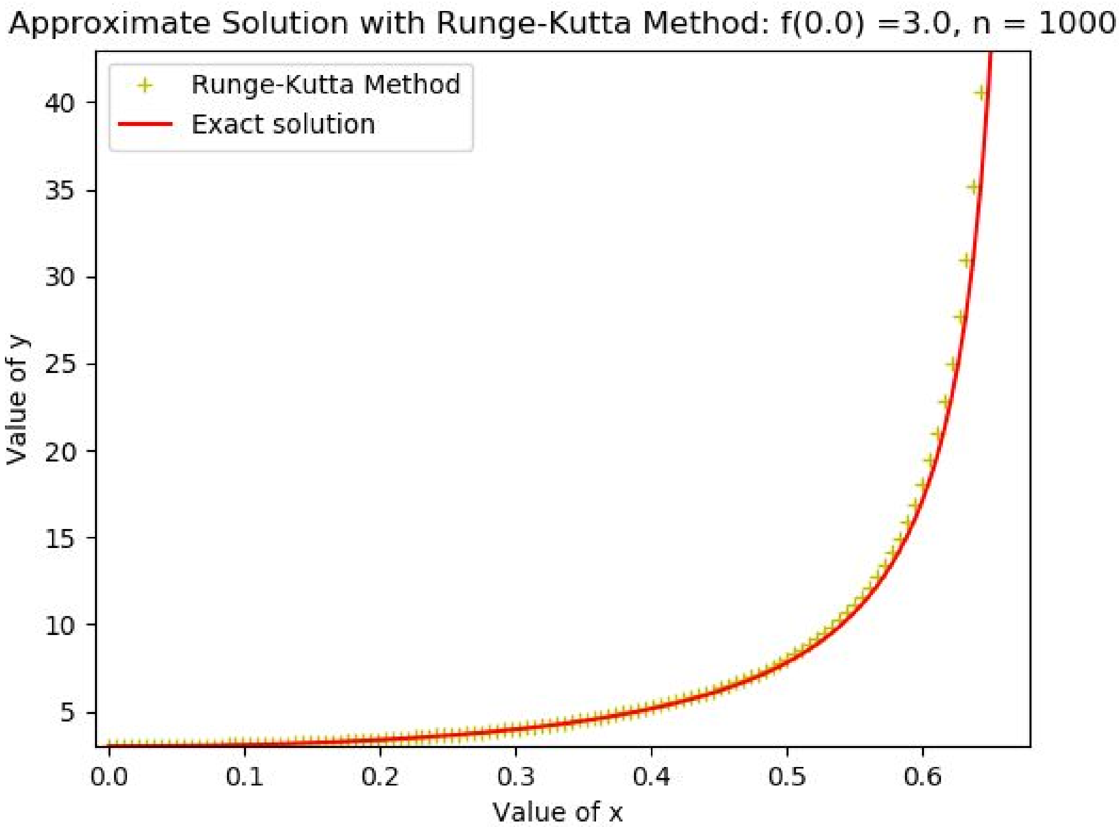
$$\begin{cases} y' = f(y, x) \\ y(x_0) = y_0 \mid iterative\ formula \\ y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k4) \\ k_1 = f(x, y) \\ k_2 = f(x + \frac{h}{2},\ y + \frac{h}{2}k1) \\ k_3 = f(x + \frac{h}{2},\ y + \frac{h}{2}k_2) \\ k_4 = f(x + h,\ y + k_3 h) \end{cases}$$

Implemented method:

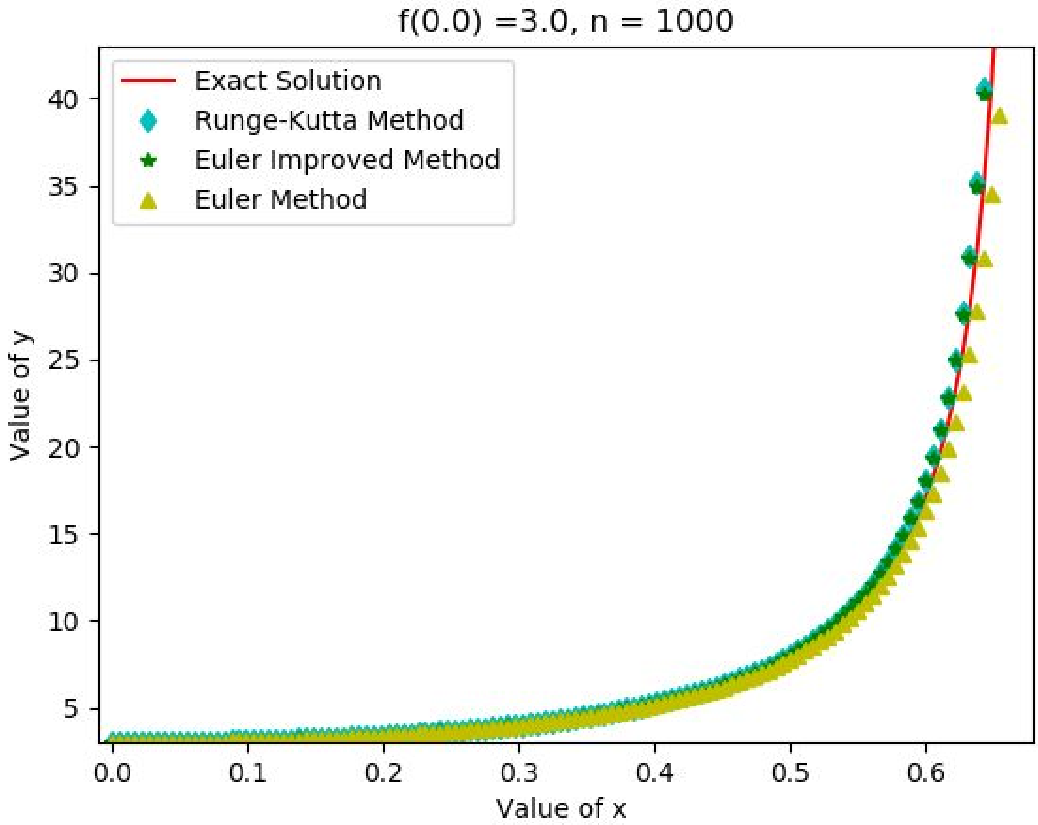
```
'''method for computing Approximate values using Runge-Kutta method of 4rd order'''


def runge_kutta_method(x, y, h):
    k1 = f(x, y)
    k2 = f(x + 0.5 * h, y + 0.5 * h * k1)
    k3 = f(x + 0.5 * h, y + 0.5 * h * k2)
    k4 = f(x + h, y + k3 * h)

    y_n = y + (h / 6.0) * (k1 + 2 * k2 + 2 * k3 + k4)
    return y_n
```



Numerical and Analytical Solutions at one graph



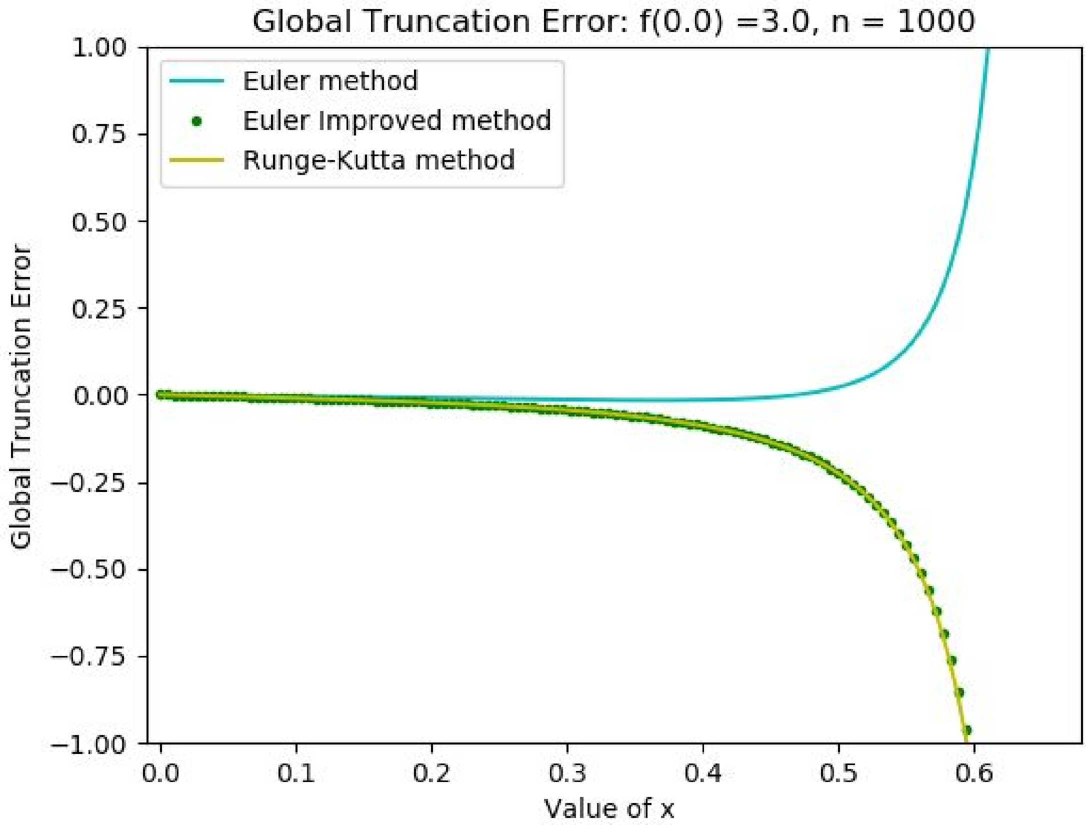
Global Truncation Error

Global Truncation Error – the cumulative error caused by many iterations

$e_n = y(x_n) - y_n$

$y(x_n)\ -\ exact\ solution,$

$y_n\ -\ approximate\ numerical\ method\ solution$



Summary:

Runge-Kutta Approximation Solution and Euler Improved Method give almost equal GTE and fit the Analytical Solution better than Euler Method.

However, there is an interval $x \in [0, 1.025)$ where all approximations are similarly good.