Programming assignment report

Author: Zhuchkov Alexey BS17-2

Variant: 4

Exaxt solution of initial value problem

$$y' = 1 + \frac{2x+1}{x^2} y \qquad x_0 = 1 \quad y_0 = 1 \quad X = 10$$

Find complementary:

$$y_1' = \frac{2x+1}{x^2} y_1$$

$$\int \frac{dy_1}{y_1} = \int \frac{2x+1}{x^2} dx = \int \frac{2\cancel{x}}{x^{\cancel{2}}} dx + \int \frac{dx}{x^2} = 2\ln|x| - \frac{1}{x}$$

$$\ln|y_1| = 2\ln|x| - \frac{1}{x}$$

$$y_1 = e^{\ln|x^2| - \frac{1}{x}} = \frac{x^2}{e^{\frac{1}{x}}}$$

Make substitution:

$$y = uy_1$$

$$y' = u'y_1 + uy_1'$$

$$u'y_1 + \overset{=0}{\left(uy_1' - \frac{uy_1(2x+1)}{x^2}\right)} = 1$$

$$u'y_1 = 1$$

$$\int du = \int \frac{e^{\frac{1}{x}}}{x^2} dx$$

$$u = \int \frac{e^{\frac{1}{x}}}{x^2} dx = -e^{\frac{1}{x}} + C$$

$$y = \frac{x^2}{e^{\frac{1}{x}}}\left(-e^{\frac{1}{x}} + C\right) = -x^2 + \frac{x^2}{e^{\frac{1}{x}}} C$$

$$C = \frac{e^{\frac{1}{x}}(y + x^2)}{x^2}$$

$$C = \frac{e^{\frac{1}{1}}(1 + 1^2)}{1^2} = 2e$$

$$\boxed{y = \frac{2x^2 e}{e^{\frac{1}{x}}} - x^2}$$

Structure of program

This is application on Python with use of *math*, *numpy*, *matplotlib*, *mpld3* and *ipywidgets* libraries

Srtucture is the following:
There's one python notebook file *Differential equations Assignment.ipynb* that contains all the functionality. There're several functions. Descriptions of all functions are below.

```
def funct(x, y):
```

- f(x) component of equation y'=f(x) from 4th variant

```
def exact(X, x0, y0):
```

- computes analytical solution for each x in list X with given initial values (x0, y0) using analytical solution

```
def euler(X, y0, step):

def improved_euler(X, y0, step):

def runge_kutta(X, y0, step):
```

- numerical procedures from finding approximated solutions. read more about it below.

```
def local_error(Y, YE):
```

- function that computes local error of numerical method by subtracting approximated result from analytical one

```
def global_error(x0, y0, x, P):
```

- function that computes the global error by choosing the maximum local error on each step with partition from n0 to N

```
def main(init_x, init_y, x, step, n0, N):
```

- computes solutions and plots the graphs

Description of methods

- Euler procedure

```
def euler(X, y0, step):
    '''
    Euler numerical procedure for solving ordinary
    differential equations with a given initial value
    :param X: list of x-values from x0 to x
    :param y0: initial y-value on y-axis
    :param step: a grid step
    :return: result of euler method
    '''
    yn = y0
    #list with solutions
    Y = [y0]
    # computes solutions for each x in X
    for x in X:
        yn = yn + step * funct(x, yn)
        Y.append(yn)
    # exclude the last element of the list
    # because with x(n) computes y(n+1) solutions
    return Y[:len(Y) - 1]
```

- Improved Euler procedure

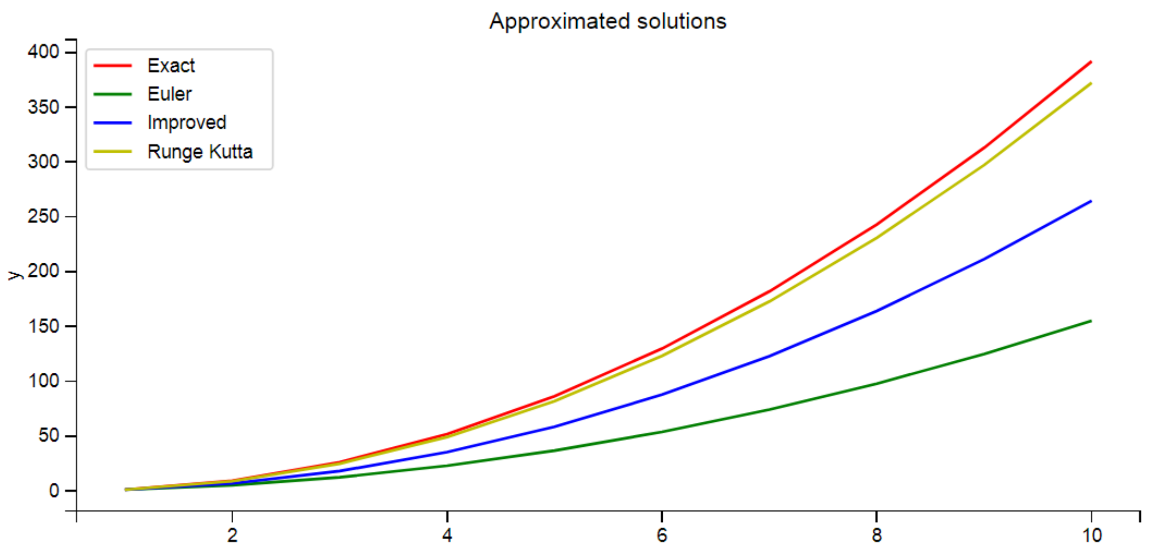
```
def improved_euler(X, y0, step):
    '''
    Improved Euler numerical procedure for solving
    ordinary differential equations with a given initial value
    :param X: list of x-values from x0 to x with given step
    :param y0: initial y-value on y-axis
    :param step: a grid step
    :return: result of improved euler method
    '''
    yni = y0
    #list with solutions
    Y = [yni]
    # computes solutions for each x in X
    for x in X:
        k1 = funct(x, yni)
        k2 = funct(x + step, yni + (step * k1))
        yni = yni + ((step / 2) * (k1 + k2))
        Y.append(yni)
    # exclude the last element of the list
    # because with x(n) computes y(n+1) solutions
    return Y[:len(Y) - 1]
```

- Runge-Kutta procedure

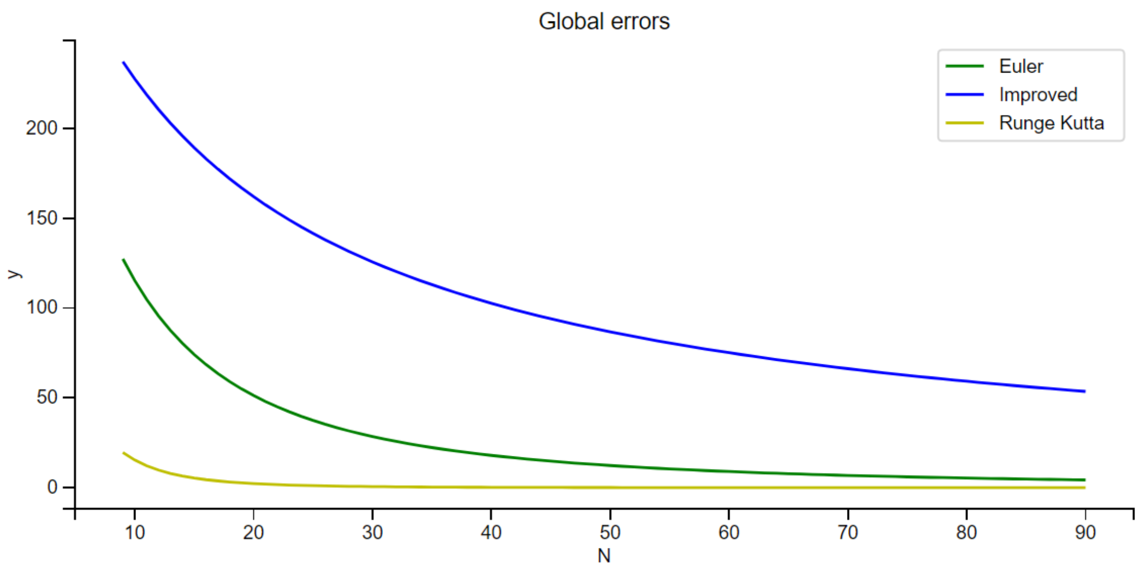
```
def runge_kutta(X, y0, step):
    '''
    Runge-Kutta's numerical procedure for solving
    ordinary differential equations with a given initial value
    :param X: list of x-values from x0 to x with given step
    :param y0: initial y-value on y-axis
    :param step: a grid step
    :return: result of Runge-Kutta method
    '''
    ynr = y0
    #list with solutions
    Y = [ynr]
    # computes solutions for each x in X
    for x in X:
        rk1 = funct(x, ynr)
        rk2 = funct(x + (step / 2), ynr + ((step / 2) * rk1))
        rk3 = funct(x + (step / 2), ynr + ((step / 2) * rk2))
        rk4 = funct(x + step, ynr + step * rk3)
        ynr = ynr + (step / 6) * (rk1 + 2 * rk2 + 2 * rk3 + rk4)
        Y.append(ynr)
    # exclude the last element of the list
    # because with x(n) computes y(n+1) solutions
    return Y[:len(Y) - 1]
```

Graphs

Approximations with step=1



Global errors with partition from 9 to 90



Github Link

https://github.com/Alexeyzhu/Differential-Equations