Numerical Methods for solving Cauchy Problem

Vasilisa Sobolevskaya 17th variant

Differential Equations

14 November 2019

Analytical Solution of the given IVP

The given initial value problem: $y' = 3y^{\frac{2}{3}}$ $y(2)=1$

We need to find the general solution of $y' = 3y^{\frac{2}{3}}$

It is first-order nonlinear ordinary differential equation

Separable equation: $\frac{y'}{y^{\frac{2}{3}}} = 3$

$\boxed{① \frac{dy}{dx} \cdot y^{\frac{2}{3}} = 3}$ $\boxed{② \frac{dy}{y^{\frac{2}{3}}} = 3dx}$ $\boxed{③ \int \frac{dy}{y^{\frac{2}{3}}} = \int 3dx}$ $\boxed{④ 3\sqrt[3]{y} = 3x + C_1 \quad C_1 \text{- const}}$

$\boxed{⑤ \; y = \left(x + \frac{C_1}{3}\right)^3}$ - general solution of DE

the we solve IVP:

$\boxed{y(2)=1 \quad y = \left(x + \frac{C_1}{3}\right)^3}$ $\boxed{\frac{C_1}{3} = C_2 \quad C_2\text{-const} \quad y = (x + C_2)}$

$\boxed{1 = (2 + C_2)^3 \quad C_2 = -1}$ exact solution of IVP ✓

$\boxed{\text{Answer: } \quad y = (x-1)^3}$

Structure of the program

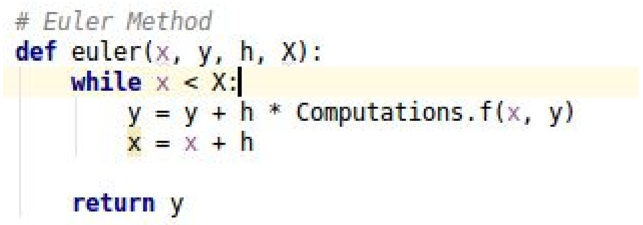
Here we have three parts: implementation of numerical methods, plotting the corresponding graphs and code of GUI.

Computational part

There are five methods that are used to construct solution of the given IVP.

They are:

- euler(x, y, h, X) - implementing Euler method

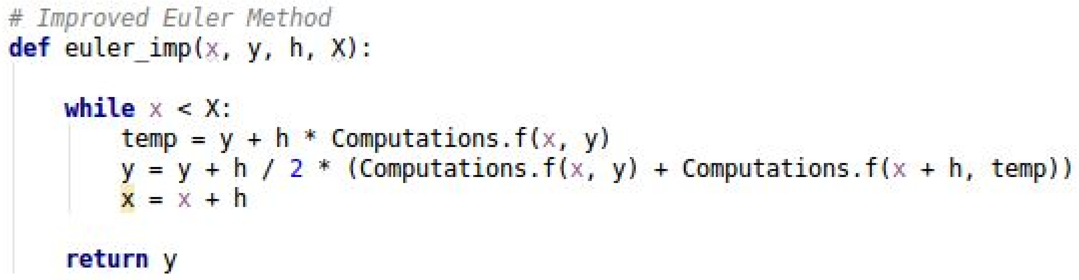


```
# Euler Method
def euler(x, y, h, X):
    while x < X:
        y = y + h * Computations.f(x, y)
        x = x + h

    return y
```

As arguments of method we have interval [x, X] we want to find solution on, the given value of function at x equal to y and step h.

According to Euler method $y_{j+1} = hf(x_j , y_j) + y_j$ is approximated value of function at x_j. We compute approximation of y at X step by step in cycle and return it.

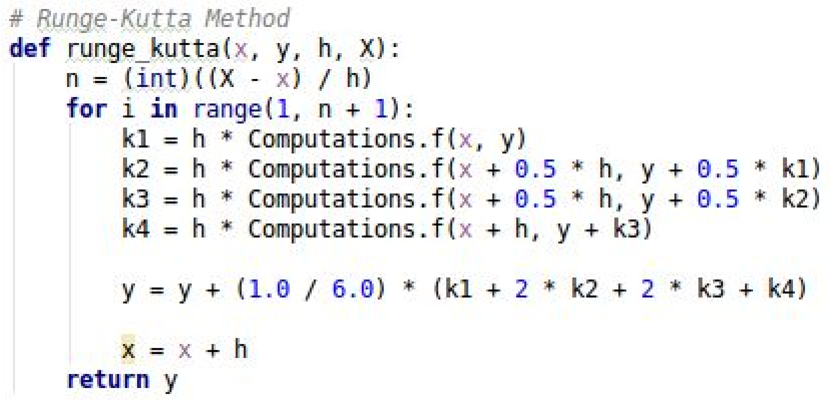
- euler_imp(x, y, h, X) - implementing Improved Euler method



```
# Improved Euler Method
def euler_imp(x, y, h, X):

    while x < X:
        temp = y + h * Computations.f(x, y)
        y = y + h / 2 * (Computations.f(x, y) + Computations.f(x + h, temp))
        x = x + h

    return y
```

As arguments of method we have interval [x, X] we want to find solution on, the given value of function at x equal to y and step h.
We compute an approximated value of function at X using improved Euler method and return it.

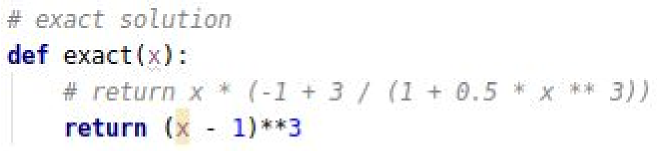
- runge_kutta(x, y, h, X) - implementing Runge-Kutta method



```
# Runge-Kutta Method
def runge_kutta(x, y, h, X):
    n = (int)((X - x) / h)
    for i in range(1, n + 1):
        k1 = h * Computations.f(x, y)
        k2 = h * Computations.f(x + 0.5 * h, y + 0.5 * k1)
        k3 = h * Computations.f(x + 0.5 * h, y + 0.5 * k2)
        k4 = h * Computations.f(x + h, y + k3)

        y = y + (1.0 / 6.0) * (k1 + 2 * k2 + 2 * k3 + k4)

        x = x + h
    return y
```

As arguments of method we have interval [x, X] we want to find solution on, the given value of function at x equal to y and step h. We compute approximated value of y at X in cycle and return it.

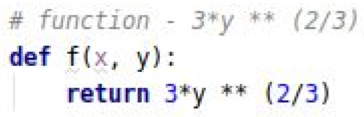
- exact(x) - exact solution of given IVP



```
# exact solution
def exact(x):
    # return x * (-1 + 3 / (1 + 0.5 * x ** 3))
    return (x - 1)**3
```

As argument of method we have value of x for which we compute corresponding value of function.

- f(x, y) - auxiliary method, used in computational ones



```
# function - 3*y ** (2/3)
def f(x, y):
    return 3*y ** (2/3)
```

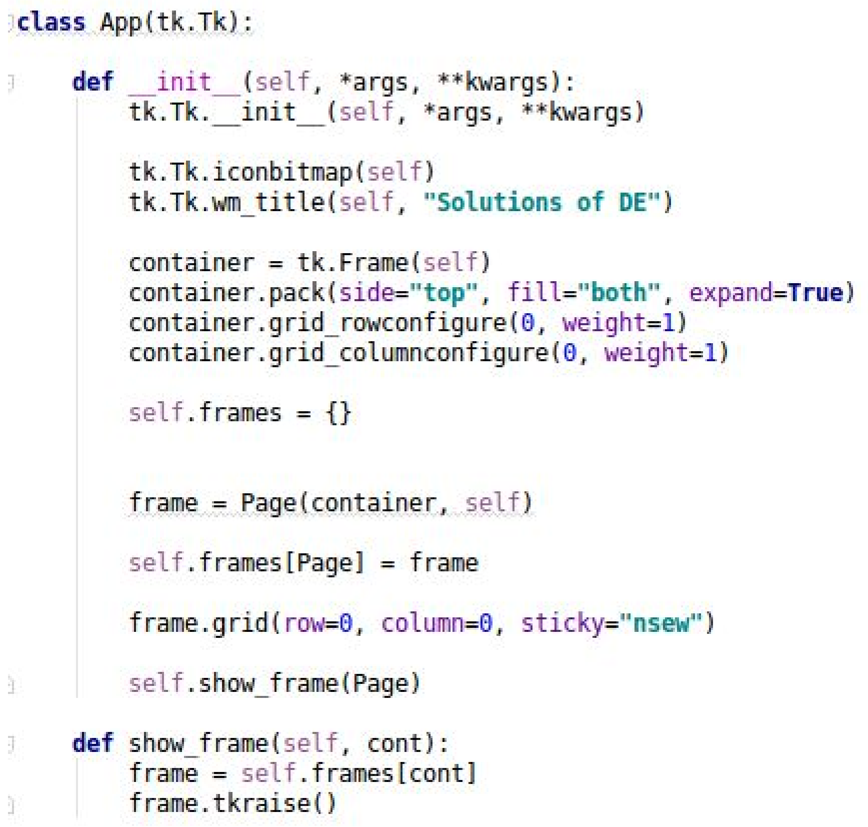
As arguments of method we have values of x and y at x and the returned value is derivative of the function at (x, y)

All of them belong to the class Computations.

GUI

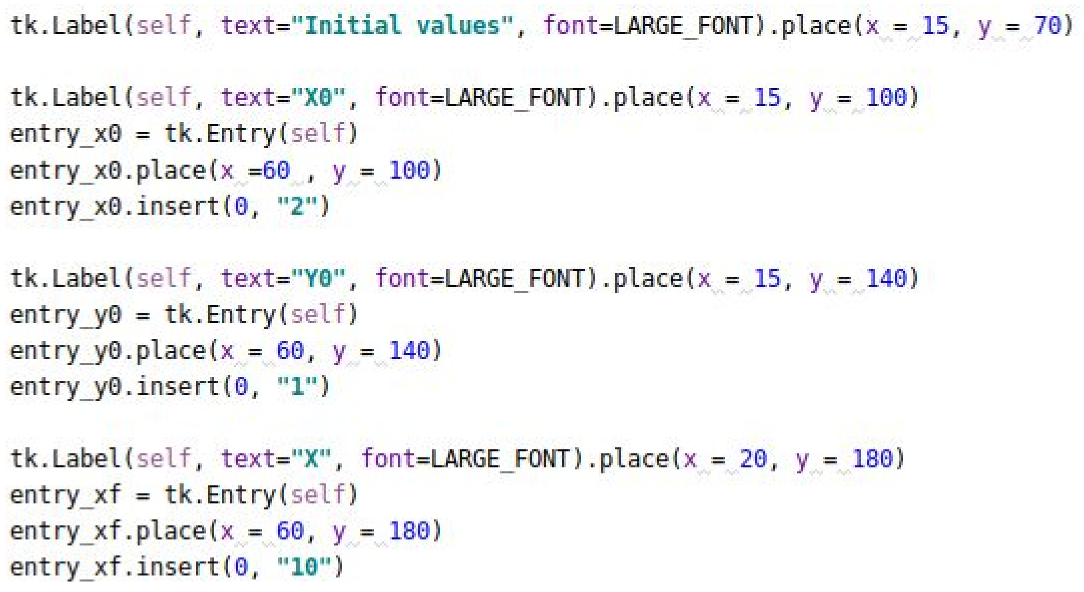
For User Interface creation the Tkinter library was chosen.

The application allows user to observe the charts of given function on the interested interval. The App class initializes application itself and describes the process of showing the window: sets window title and geometry.



```
class App(tk.Tk):

    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)

        tk.Tk.iconbitmap(self)
        tk.Tk.wm_title(self, "Solutions of DE")

        container = tk.Frame(self)
        container.pack(side="top", fill="both", expand=True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.frames = {}


        frame = Page(container, self)

        self.frames[Page] = frame

        frame.grid(row=0, column=0, sticky="nsew")

        self.show_frame(Page)

    def show_frame(self, cont):
        frame = self.frames[cont]
        frame.tkraise()
```

The __Init_ function in the Page class sets off the main window of application. Here all the elements are set on the canvas with 'place' geometry (with given coordinates and size).



```
tk.Label(self, text="Initial values", font=LARGE_FONT).place(x = 15, y = 70)

tk.Label(self, text="X0", font=LARGE_FONT).place(x = 15, y = 100)
entry_x0 = tk.Entry(self)
entry_x0.place(x =60 , y = 100)
entry_x0.insert(0, "2")

tk.Label(self, text="Y0", font=LARGE_FONT).place(x = 15, y = 140)
entry_y0 = tk.Entry(self)
entry_y0.place(x = 60, y = 140)
entry_y0.insert(0, "1")

tk.Label(self, text="X", font=LARGE_FONT).place(x = 20, y = 180)
entry_xf = tk.Entry(self)
entry_xf.place(x = 60, y = 180)
entry_xf.insert(0, "10")
```

Drawing Plots

For graphs construction the matplotlib library and numpy package were used. There also was imported the TkAgg backend to synchronise Matplotlib with Tkinter.
The construction of corresponding plots is performed in command draw_plots in initializing method of application page.
Firstly app receives initial values from input entries and computes the length of step:

```
x0 = float(entry_x0.get())
y0 = float(entry_y0.get())
xf = float(entry_xf.get())
n = int(entry_n.get())

h = (xf - x0) / (n - 1)
```

than it computes corresponding values of function with different methods:

```
y1[0] = y0
y2[0] = y0
y3[0] = y0
y4[0] = y0
for i in range(1, n):
    y1[i] = Computations.euler(x0, y0, h, x[i])
    y2[i] = Computations.euler_imp(x0, y0, h, x[i])
    y3[i] = Computations.runge_kutta(x0, y0, h, x[i])
    y4[i] = Computations.exact(x[i])
```

and finally draws the plot:

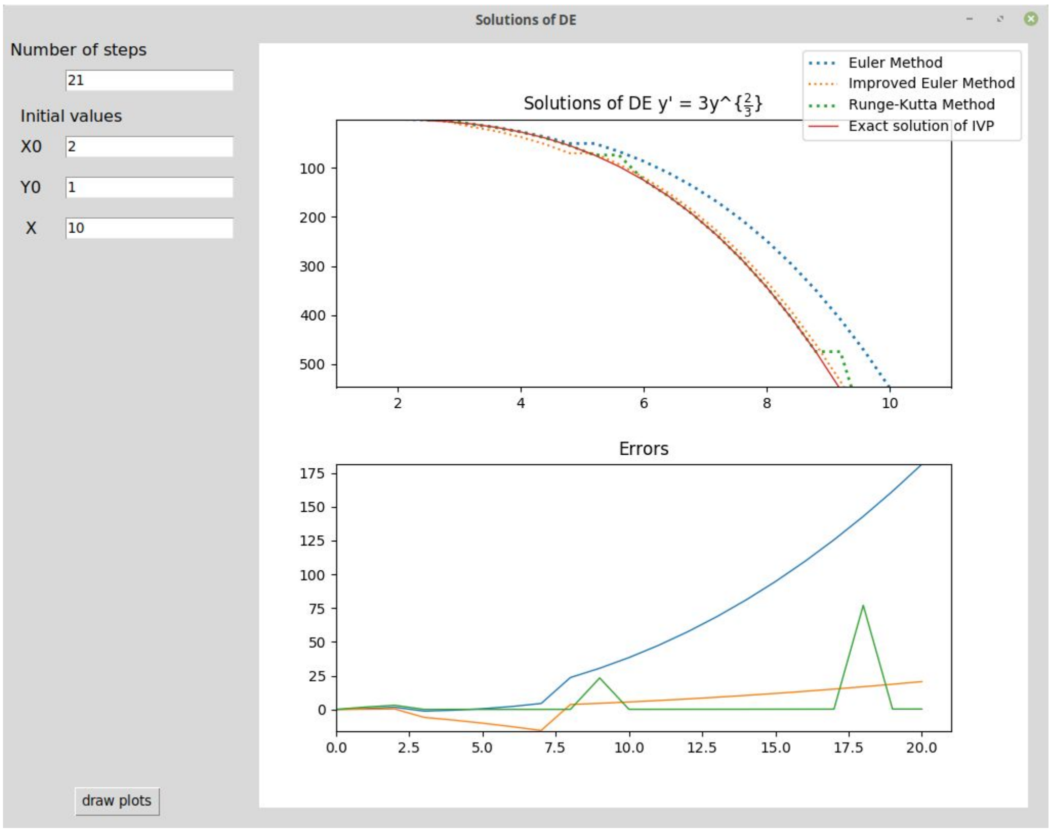
```
ax1 = f.add_axes([0.1, 0.55, 0.8, 0.35], xlim=(x_limit[0], x_limit[1]), ylim=(y_limit[0], y_limit[1]),
                 title="Solutions of DE y' = " r"3y^{$\frac{2}{3}$}")
ax1.plot(x, y1, ':', linewidth=2.0)
ax1.plot(x, y2, ':')
ax1.plot(x, y3, ':', linewidth=2.0)
ax1.plot(x, y4, linewidth=1.0)
```

In the same way the construction of the global error graphs is organised.

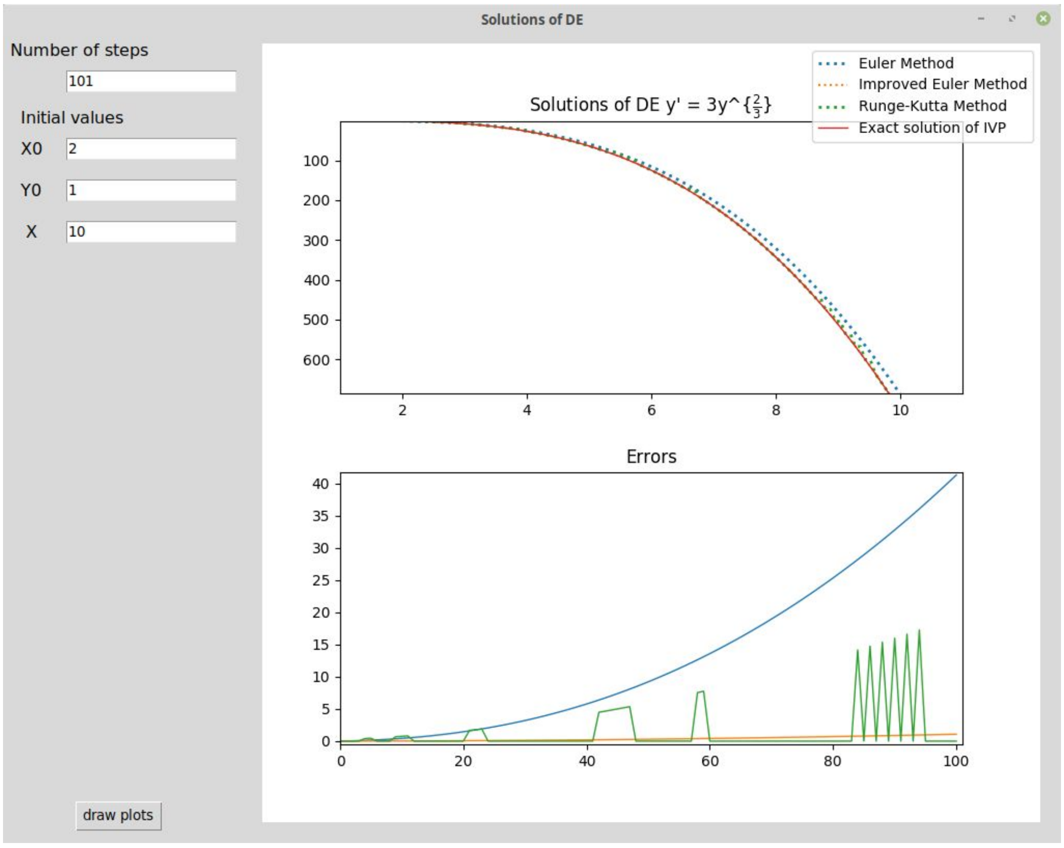
Result

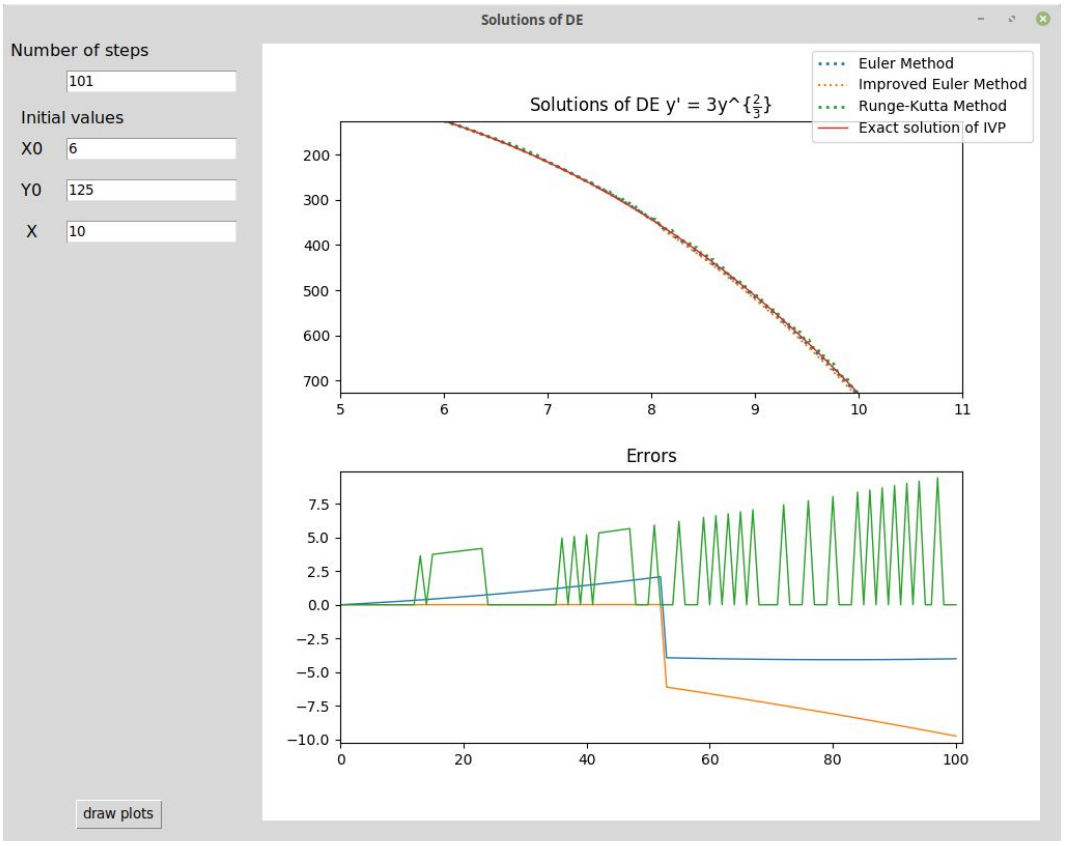
The resulting application constructs approximation of the solution of given IVP and provides the data visualisation capability in the user interface.
Initially there is only clear canvas with entries with default values.

As the result of pushing the ‘draw plots’ button the charts of corresponding approximations and errors will appear.



There is possibility to change the initial conditions, e. g. number of steps





Number of steps: 101

Initial values

X0: 6

Y0: 125

X: 10

Link to GitHub repository:https://github.com/IsumaNagasaki/DECompPract