Computational Practicum

Vladislav Lamzenkov, BS1905

November 1, 2020

1 Solve the differential equation

Given the equation:

$$y' = -\frac{y^2}{3} - \frac{2}{3x^2} \tag{1}$$

Before solving, we analyze the equation above: it has only first order derivative, thus, it is the differential equation with the first order. So, our equation is a nonlinear first order differential equation. Moreover, we have a Riccati's equation.

Step 1. Firstly, we know that $x \neq 0$(We obtain discontinuity at this point). To solve a Riccati's equation, we need to find any particular solution of the equation and then do the substitution $y = y_p(x) + z(x)$. Try to do the following substitution $y = \frac{a}{x}$ and $y' = \frac{-a}{x^2}$. Substitute into the original equation:

$$\frac{-a}{x^2} + \frac{a^2}{3x^2} = \frac{-2}{3x^2} \tag{2}$$

$$\frac{a^2 - 3a}{3x^2} = \frac{-2}{3x^2} \tag{3}$$

$$a^2 - 3a + 2 = 0 \tag{4}$$

So, we get $a_1 = 2$, $a_2 = 1$. We found the particular solution, so do the following substitution $y = \frac{2}{x} + z$, where $z = z(x)$. Then find the derivative of the substitution: $y' = \frac{-2}{x^2} + z'$. Then substitute it to (1):

$$\frac{-2}{x^2} + z' = \frac{-4}{3x^2} - \frac{4z}{3x} - \frac{z^2}{3} - \frac{2}{3x^2} \tag{5}$$

Assuming $z \neq 0$, divide both parts of the equation by z^2. By the way, $z = 0$ is a trivial solution of the equation (5).

$$\frac{z'}{z^2} + \frac{4}{3xz} = -\frac{1}{3} \tag{6}$$

Then do the following substitution $z = \frac{1}{u}$, where $u = u(x)$. Its derivative is $y' = \frac{-u'}{u^2}$. Substitute it into (6):

$$-u' + \frac{4u}{3x} = -\frac{1}{3} \tag{7}$$

Now we get Bernoulli's equation, solve its complementary:

$$-{u_c}' + \frac{4u_c}{3x} = 0 \tag{8}$$

Assuming $u_c \neq 0$, divide both parts of (8):

$$\int (\frac{d(u_c)}{u_c}) = \int (\frac{4dx}{3x}) \tag{9}$$

Thus, we get:

$$u_c = x^{\frac{4}{3}} \tag{10}$$

After this, do the following substitution $u = x^{\frac{4}{3}}p$, where $p = p(x)$ and $u' = \frac{4px^{\frac{1}{3}}}{3} + p'x^{\frac{4}{3}}$. Then substitute it to (7):

$$-p'x^{\frac{4}{3}} = -\frac{1}{3} \tag{11}$$

Then we have a separable equation, divide both parts of the equation by $x^{\frac{4}{3}}$(we assumed that $x \neq 0$) and integrate its both parts.

$$\int (dp) = \int (\frac{dx}{3x^{\frac{4}{3}}}) \tag{12}$$

After this, we obtain the following $p = \frac{-1}{x^{\frac{1}{3}}} + c$, where $c \in R$. Find u:

$$u = x^{\frac{4}{3}}(\frac{-1}{x^{\frac{1}{3}}} + c) = -x + cx^{\frac{4}{3}}, c \in R \tag{13}$$

Find z:

$$z = \frac{1}{cx^{\frac{4}{3}} - x}, c \in R \tag{14}$$

Finally, find y:

$$y = \frac{2}{x} + \frac{1}{cx^{\frac{4}{3}} - x}, c \in R \tag{15}$$

Step 2. Solve the initial value problem: $y(1) = 2$ From (15) we can get the expression to find c:

$$c = \frac{x^{\frac{4}{3}}(xy - 2)}{yx^2 - x} = \frac{1 * (1 * 2 - 2)}{2 - 1} = 0 \tag{16}$$

So, to sum up, we obtain the following solution for the given IVP: $y = \frac{2}{x} - \frac{1}{x} = \frac{1}{x}$

2 Application GUI

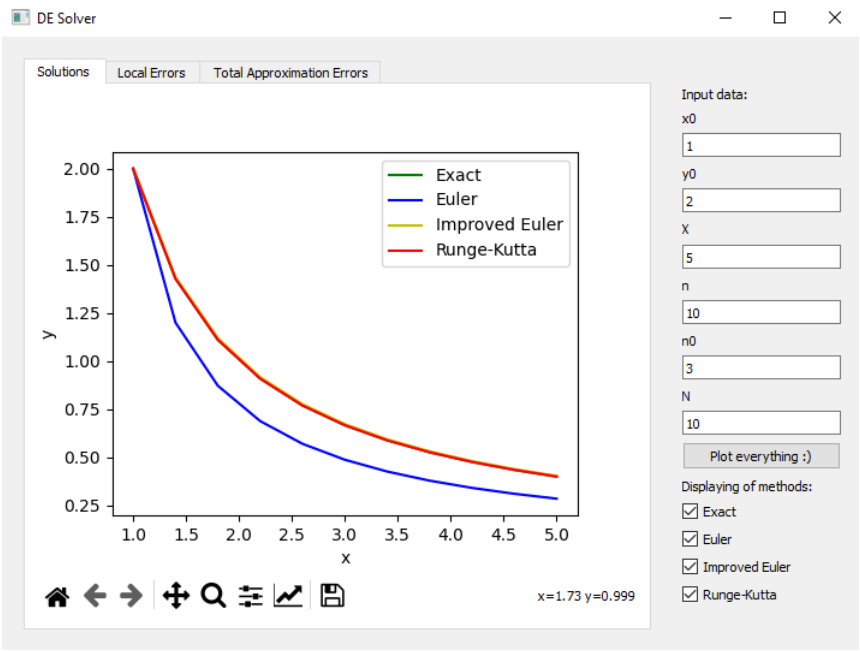

Figure 1. Tab with solutions.

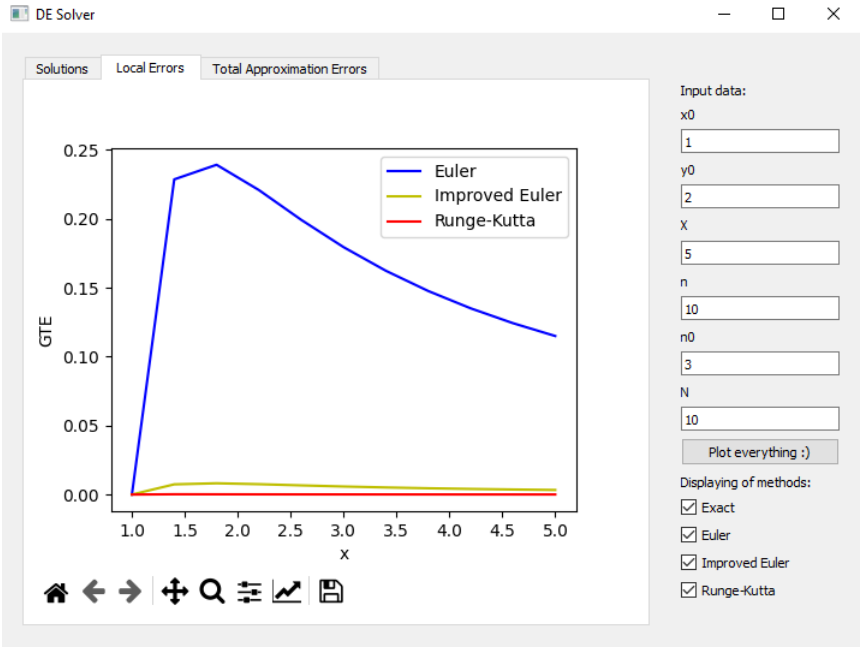

Figure 2. Tab with lolcal errors.

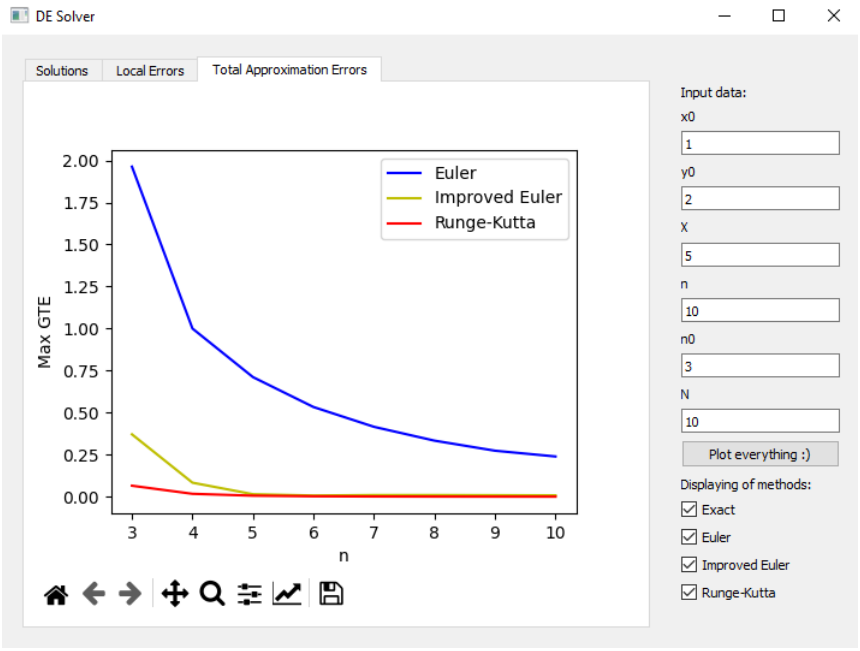

Figure 3. Tab with total approximation errors.

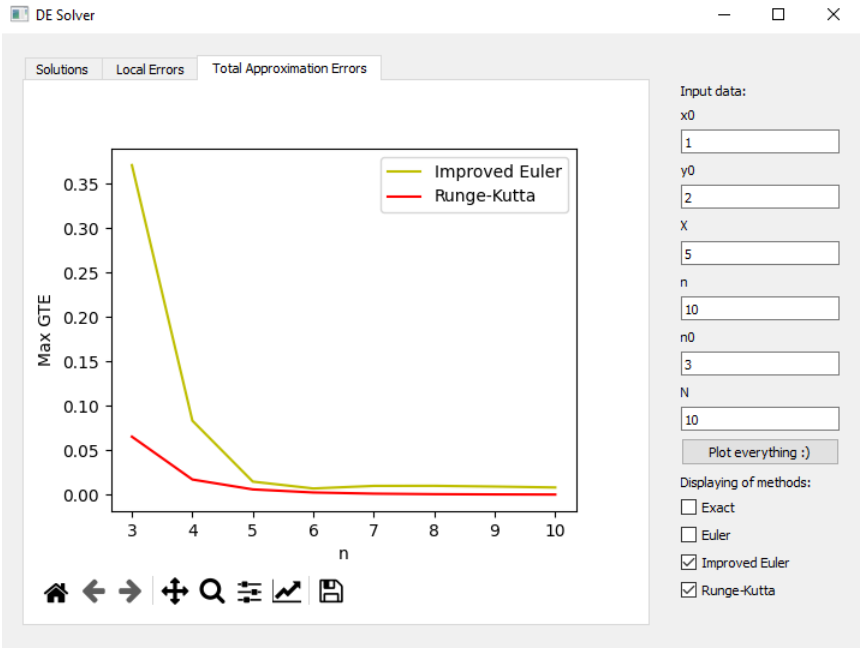

Figure 4. Tab with total approximation errors with disabled 'Exact' and 'Euler' methods.

3 UML Class Diagram

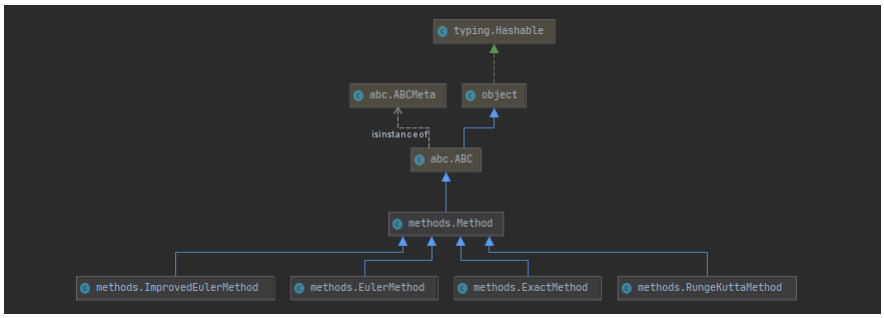

Figure 5. UML of methods.

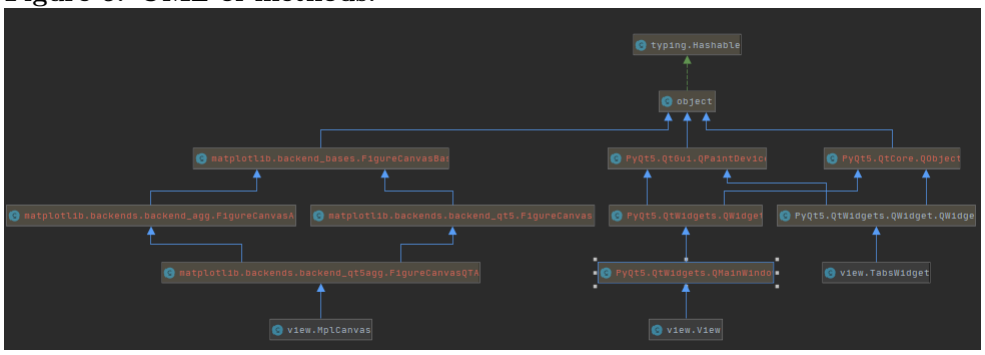

Figure 6. UML of View.

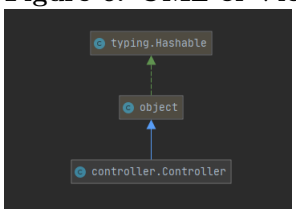

Figure 7. UML of Controller.

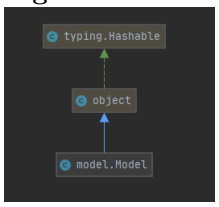

Figure 8. UML of Model.

4 Source code

For this assignment I decided to use language Python 3.8 with the following libraries:
1. *numpy* - working with arrays.
2. *matplotlib* - build charts.
3. *PyQt5* - GUI interface.

The code is written following OOP design and SOLID principle. The entire project is organized using ModelViewController architectural pattern:

- **Model** holds the data structure which the program is working with.
- **View** is the representation of graphical information as shown to the user.
- **Controller** accepts input from the user and handle them.

The source code is available on **Github**:
https://github.com/mcflydesigner/Differential-Equation-Computation

Most interesting parts of the source code:

```
class Method(ABC):
    """ Abstract class for each method of solving DE """
    def __init__(self, de, X, n):
        self._de = de
        self._X = X
        self._n = n
        #Calculate the step
        self._h = (X - de.ivp.x0) / n
        self._xs = list(np.linspace(de.ivp.x0, X, n+1))
        self._ys = np.array([de.ivp.y0])
    @abstractmethod
    def solve(self):
        """ Function to solve the DE """
        pass
    @abstractmethod
    def getColor(self):
        """ The color will be used on the chart. """
        pass
    def getXs(self):
        """ Getter of x-coordinates of the method """
        return self._xs
    def getYs(self):
        """ Getter of y-coordinates of the method """
        return self._ys
```

Figure 9. Abstract class for methods.

```
class ExactMethod(Method):
    """
        Implementation of Exact Method for solving DE.
        This method is not numerical.
    """
    def __init__(self, de, X, n):
        super().__init__(de, X, n)


    def solve(self):
        for x in self._xs[1:]:
            y = self._de.getExactSolution(x)
            self._ys = np.append(self._ys, y)

    def getColor(self):
        return 'g'

    def __str__(self):
        return 'Exact'
```

Figure 10. The class for Exact method.

```python
class EulerMethod(Method):
    """
        Implementation of Euler Method for solving DE.
        This method is numerical.
    """

    def __init__(self, de, X, n):
        super().__init__(de, X, n)

    def solve(self):
        for x in self._xs[:-1]:
            y = self._ys[-1] + self._h * self._de.f(x, self._ys[-1])
            self._ys = np.append(self._ys, y)

    def getColor(self):
        return 'b'

    def __str__(self):
        return 'Euler'
```

Figure 11. The class for Euler method.

```python
class ImprovedEulerMethod(Method):
    """
        Implementation of Improved Euler Method for solving DE.
        This method is numerical.
    """

    def __init__(self, de, X, n):
        super().__init__(de, X, n)

    def solve(self):
        for x in self._xs[:-1]:
            y = self._ys[-1] + (self._h / 2) * ((self._de.f(x, self._ys[-1]) +
                                                 self._de.f(x + self._h, self._ys[-1] +
                                                            self._h * self._de.f(x, self._ys[-1]))))

            self._ys = np.append(self._ys, y)

    def getColor(self):
        return 'y'

    def __str__(self):
        return 'Improved Euler'
```

Figure 12. The class for Improved Euler method.

```python
class RungeKuttaMethod(Method):
    """
            Implementation of Runge-Kutta Method for solving DE.
            This method is numerical.
        """

    def __init__(self, de, X, n):
        super().__init__(de, X, n)

    def solve(self):
        for x in self._xs[:-1]:
            k1 = self._de.f(x, self._ys[-1])
            k2 = self._de.f(x + self._h / 2, self._ys[-1] + (self._h / 2) * k1)
            k3 = self._de.f(x + self._h / 2, self._ys[-1] + (self._h / 2) * k2)
            k4 = self._de.f(x + self._h, self._ys[-1] + self._h * k3)

            y = self._ys[-1] + (self._h / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
            self._ys = np.append(self._ys, y)

    def getColor(self):
        return 'r'

    def __str__(self):
        return 'Runge-Kutta'
```

Figure 13. The class for Runge-Kutta method.