Differential equations computational practicum

Arina Drenyassova BS19-04
a.drenyasova@innopolis.university

1 Exact Solution

Github: 1.

$$\begin{cases} y' = \frac{1}{x} + \frac{2y}{xln(x)} \\ y(2) = 0 \\ x \in (2, 12) \end{cases}$$

2. Let's subtract $\frac{2y}{xln(x)}$ from both sides, when we have:

$$y' - \frac{2y}{xln(x)} = \frac{1}{x}$$

3. Let's $m(x) = e^{(\frac{2dx}{xln(x)})} = \frac{1}{ln^2(x)}$ and multiply both sides by m(x):

$$\frac{y'}{ln^2(x)} - \frac{2y}{xln^3(x)} = \frac{1}{xln^2(x)}$$

4. We know that $(\frac{1}{ln^2(x)})' = \frac{-2}{xln^3(x)}$, so let's substitute:

$$\frac{y'}{ln^2(x)} - (\frac{1}{ln^2(x)})' = \frac{1}{xln^2(x)}$$

5. By reserse production rule we obtaion:

$$\frac{d}{dx}(\frac{y'}{ln^2(x)}) = \frac{1}{xln^2(x)}$$

6. Let intergrate both sides with respect to x:

$$\int \frac{d}{dx}(\frac{y'}{ln^2(x)})dx = \int \frac{1}{xln^2(x)}dx$$

7. As result we have:

$$\frac{y}{ln^2(x)} = -\frac{1}{ln(x)} + c_1$$

8. Consequently y:

$$y = c_1 ln^2(x) - ln(x)$$

9. Let' s use $y(2) = 0$:

$$c_1 = \frac{1}{ln(2)}$$

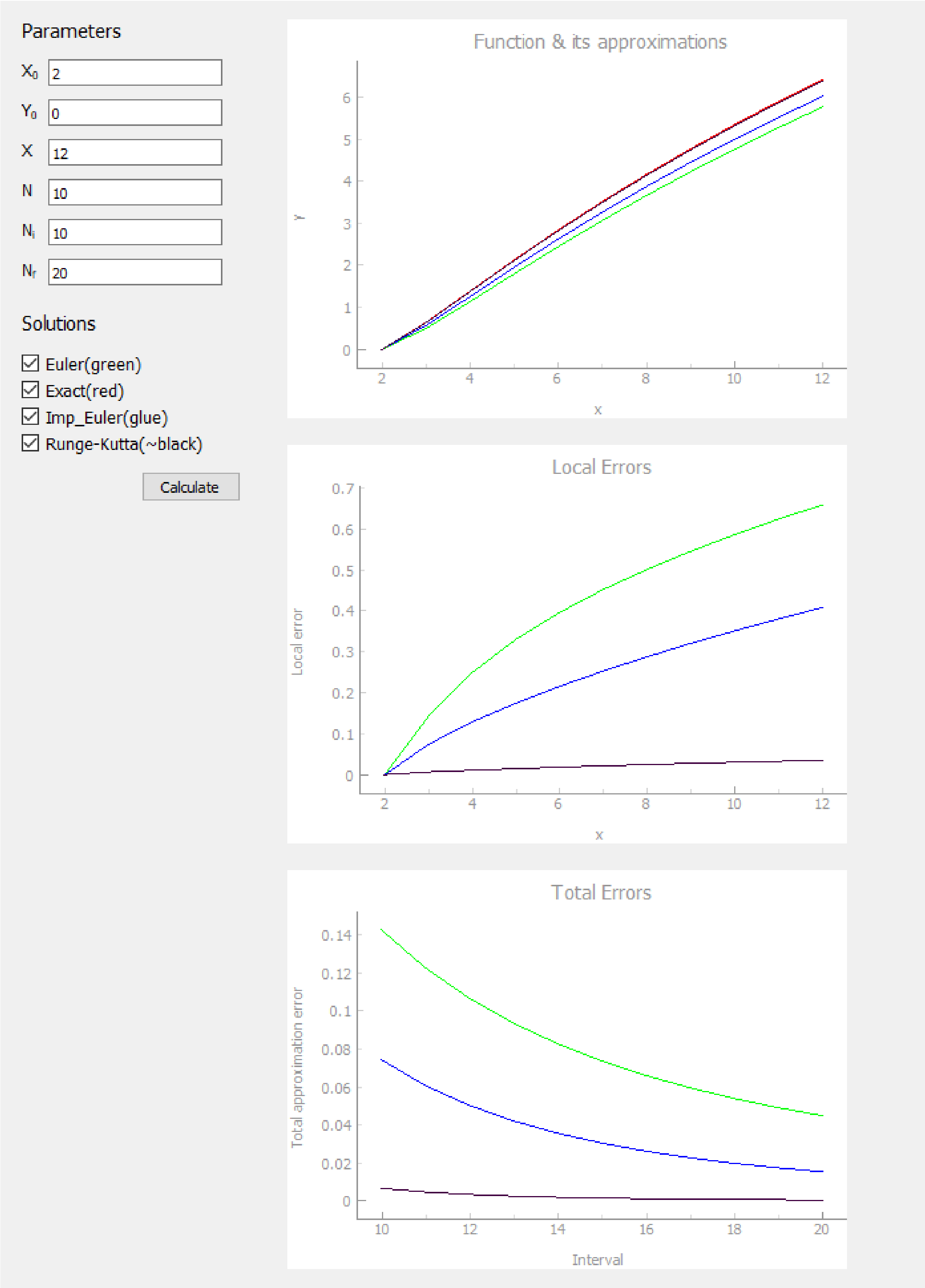
10. Exact solution equal to:

$$y = \frac{1}{ln(2)} ln^2(x) - ln(x)$$

Answer: $y = \frac{1}{ln(2)} ln^2(x) - ln(x)$ with no point of discontinuity on give interval $x \in (2, 12)$

2 Computational part

For the computation part of the assignment Python programming language was used. For user interface Qt designer. The main window is shown below.



2.1 Requirements

Requirements for building the code from source code are:
- Python 3.6
- PyQt5
- pyqtgraph
- numpy
- scipy.integrate

2.2 Input. For solving IVP and local error

x_0 and y_0 - Initial values for a diff. equation.
X - end point for numerical solutions.
N - number of intervals for numerical solutions.

2.3 Input. For solving total approximation error

N_i - starting point of intervals.
N_f - end point of intervals.

2.4 Input Validation

x_0, y_0 and X must be dot-separable rational numbers.
N_i, N_f must be positive integers where $N_i < N_f$.

2.5 GUI components

- ***QCheckBox*** is an option button that can be switched on (checked) or off (unchecked). QCheckBox has isChecked() that return true or false.
- ***QLabel*** is used for displaying text or an image. No user interaction functionality is provided.
- ***QLineEdit*** widget is a one-line text editor. A line edit allows the user to enter and edit a single line of text.
- ***Qwidget*** is a subclass of QPaintDevice, subclasses can be used to display custom content that is composed using a series of painting operations with an instance of the QPainter class.

2.6 Computational code

Exact method:

```
class Exact_Solution:
    def __init__(self):
        return

    def solve(self, de: DE):
        try:
            ysa = []
            xs =[de.x0]
            for i in range(1, de.N + 1):
                xs.append(de.x0 + (i * de.h))
            dydx = eval("lambda y, x :" + de.string)
            ys = odeint(dydx, de.y0, xs)
            for i in range(0, len(ys)):
                ysa.append(ys[i][0])
            return xs, ysa
        except:
            print("Can not find exact solution")
            return [], []
```

Euler method:

```
class Euler_Solution:
    def __init__(self):
        return

    def solve(self, de: DE):
        try:
            xs = [de.x0]
            ys = [de.y0]
            h = (de.x - de.x0) / de.N
            for i in range(1, de.N+1):
                xs.append(de.x0 + (i * h))
                ys.append(ys[i-1] + (h * de.f(xs[i-1], ys[i-1])))
            return xs, ys
        except:
            print("Can not find euler solution")
            return [], []
```

Improved Euler method:

```
class Improved_Euler_Solution:
    def __init__(self):
        return

    def solve(self, de: DE):
        try:
            xs = [de.x0]
            ys = [de.y0]
            y_temp = de.y0
            h = (de.x - de.x0) / de.N
            for i in range(1, de.N+1):
                k1 = de.f(xs[i-1], y_temp)
                k2 = de.f(xs[i-1] + h, y_temp + h*k1)
                k = (k1 + k2) / 2
                y_temp += h * k
                xs.append(de.x0 + (i * h))
                ys.append(y_temp)
            return xs, ys
        except:
            print("Can not find improved euler solution")
            return [], []
```

Runge–Kutta method:

```python
class Runge_Kutta_Solution:
    def __init__(self):
        return

    def solve(self, de: DE):
        try:
            xs = [de.x0]
            ys = [de.y0]
            h = (de.x - de.x0) / de.N
            for i in range(1, de.N+1):
                xs.append(de.x0 + (i * h))
                k1 = de.fi(ys[i-1], xs[i-1])
                k2 = de.fi(ys[i-1] + k1 * h/2, xs[i-1] + h/2)
                k3 = de.fi(ys[i-1] + k2 * h/2, xs[i-1] + h/2)
                k4 = de.fi(ys[i - 1] + h * k3, xs[i - 1] + h)
                k = h/6 * (k1 + k2*2 + 2*k3 +k4)
                ys.append(ys[i-1] + k)
            return xs, ys
        except:
            print("Can not find runge kutta solution")
            return [], []
```

Local Error:

```python
    def local_calc(self, y: list, appx: list, appy: list):
        try:
            x = []
            err = []
            for i in range(len(appx)):
                x.append(appx[i])
                err.append(math.fabs(appy[i] - y[i]))
            return x, err
        except ValueError:
            print("Can not find local error")
            return [], []
```

Total Error:

```python
    def global_calc(self, ey, appx, appy):
        try:
            g_x, g_y = self.local_calc(ey, appx, appy)
            x = []
            err = []
            for i in range(1, min(len(g_x), len(g_y))):
                x.append(g_x[i])
                err.append(math.fabs(g_y[i] - g_y[i - 1]))
            return x, err
        except ValueError:
            print("Can not find total error")
            return [], []
```

3 UML diagram