Computational practicum

Name: Andrey Kuzmickiy
Group: BS20-04

Analytical solution (exact solution)

$$\begin{cases} y' = 2x(x^2 + y) \\ y(0) = 0 \\ x \in [0, 10] \end{cases}$$

Points of discontinuity: There is no points of discontinuity in the equation.

Exact solution for given IVP (initial value problem): $y = e^{x^2} - x^2 - 1$

Practicum

$$\begin{cases} y' = 2x(x^2+y) \\ y(0) = 0 \\ x \in (0, 10) \end{cases}$$

The equation has a form: $y' + a(x)y = b(x)$

$y' - 2xy = 2x^3$ Let's solve complementary equation

$y' - 2xy = 0$ complementary equation

$\frac{dy}{dx} = 2xy$ Let's transform it into differential form and integrate it

$\int \frac{dy}{y} = \int 2x\,dx$

$\ln|y| = x^2 + C$

$y = C_1 e^{x^2}$ where $C_1 = C_1(x)$ is a function depends on x

$y' = C_1' e^{x^2} + 2xC_1 e^{x^2}$

To find C_1 let's substitute y and y' into original equation and solve it

$C_1' e^{x^2} + 2xC_1 e^{x^2} = 2x^3 + 2xC_1 e^{x^2}$

$C_1' e^{x^2} = 2x^3$

$C_1' = \frac{2x^3}{e^{x^2}} = \frac{dC_1}{dx}$

$\int dC_1 = \int \frac{2x^3}{e^{x^2}} dx$

$C_1 + C_2 = 2\int x^3 e^{-x^2} dx = \begin{matrix} u = x^2 \\ du = 2x\,dx \end{matrix} = \int u e^{-u} du = -ue^{-u} - \int (-e^{-u}) du = -ue^{-u} - e^{-u} = -e^{-x^2}(x^2+1)$

$y = -e^{-x^2}(x^2+1)e^{x^2} + C_2 e^{x^2} = -x^2 - 1 + C_2 e^{x^2}$

$C_2 = (y + x^2 + 1)e^{-x^2}$ formula for calculating the constant

Initial value problem $y(0) = 0$

$C_2 = (0 + 0 + 1) \cdot 1 = 1$

Answer: $y = -x^2 - 1 + e^{x^2}$; This function is symmetric (even) without points of discontinuity.

Program's part

The program allows user to see the graph of the solution of the equation $y = C_2 e^{x^2} - x^2 - 1$ with opportunity to change initial conditions, range and number of grid steps.

For calculating new exact solution the program use the following formula to calculate the constant C_2: $C_2 = (y + x^2 + 1)e^{-x^2}$

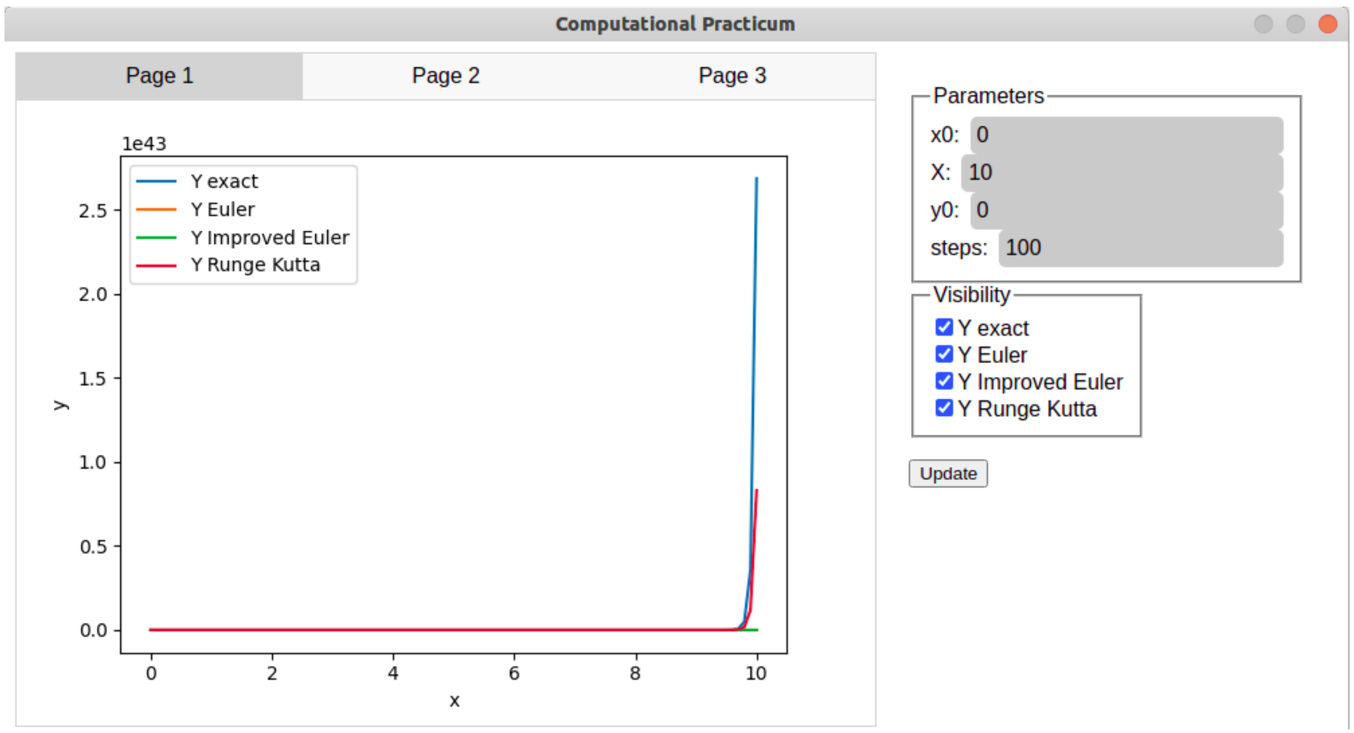
Graphs

Graph of solutions

There are 4 lines represented different types of the solution:

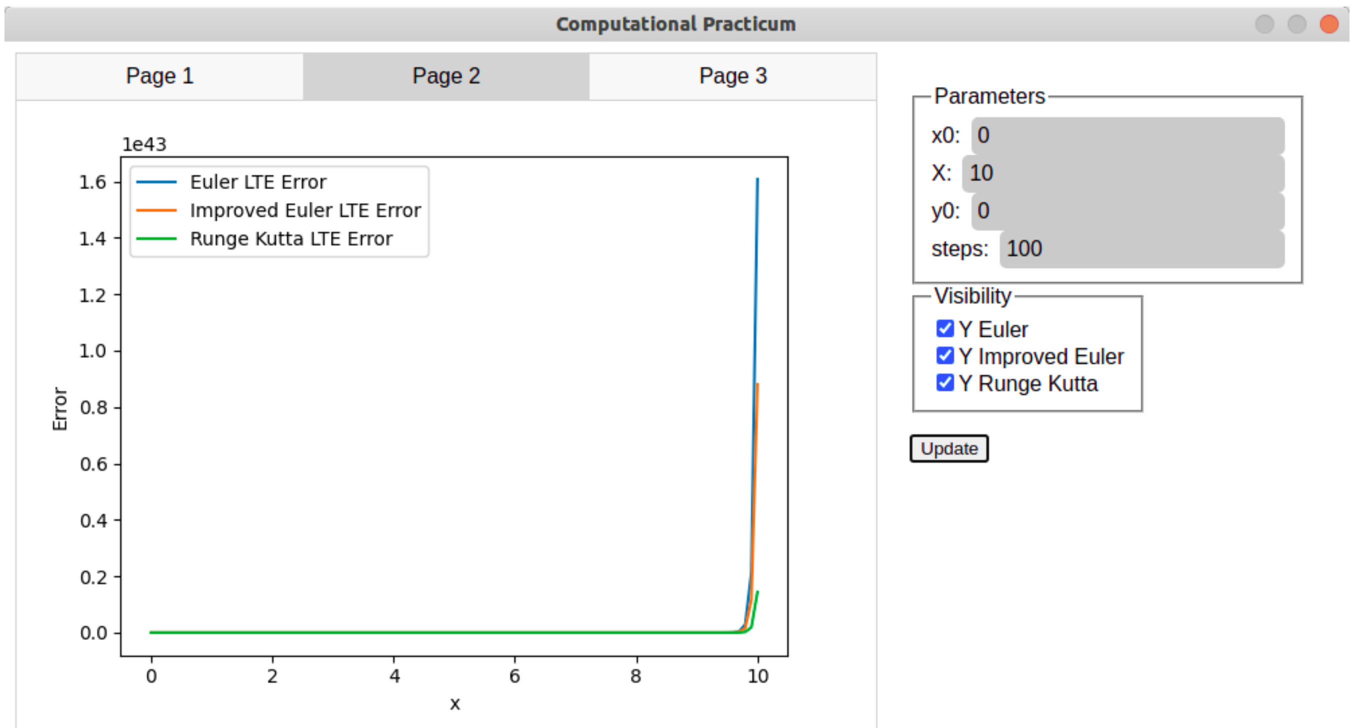
- Exact solution;
- Approximate solution using Euler's method;
- Approximate solution using Improved Euler's method;
- Approximate solution using Runge Kutta method.

$y - axis$ represents solution for given x with values $\in [0, 2.7 * 10^{43}]$.



Graph of local errors

There are local truncation errors (LTE) of each method.

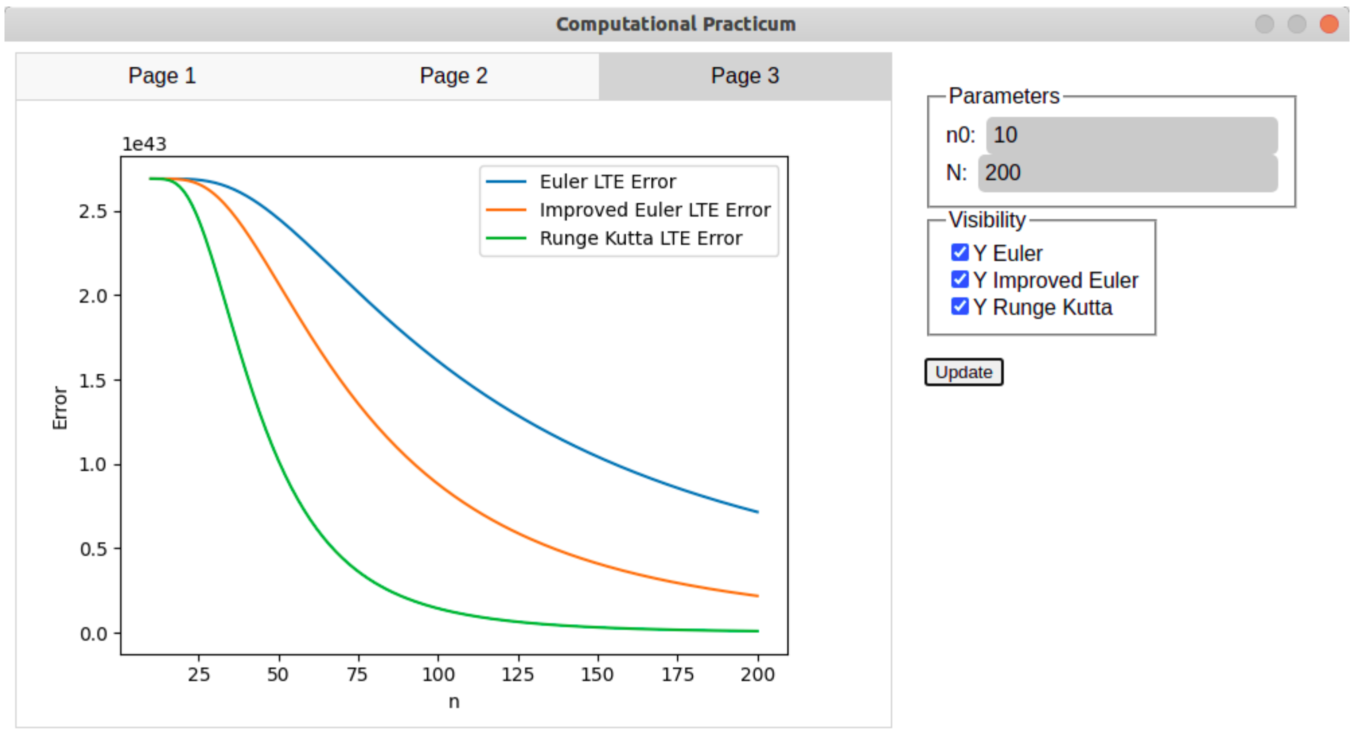


Graph of total approximation error

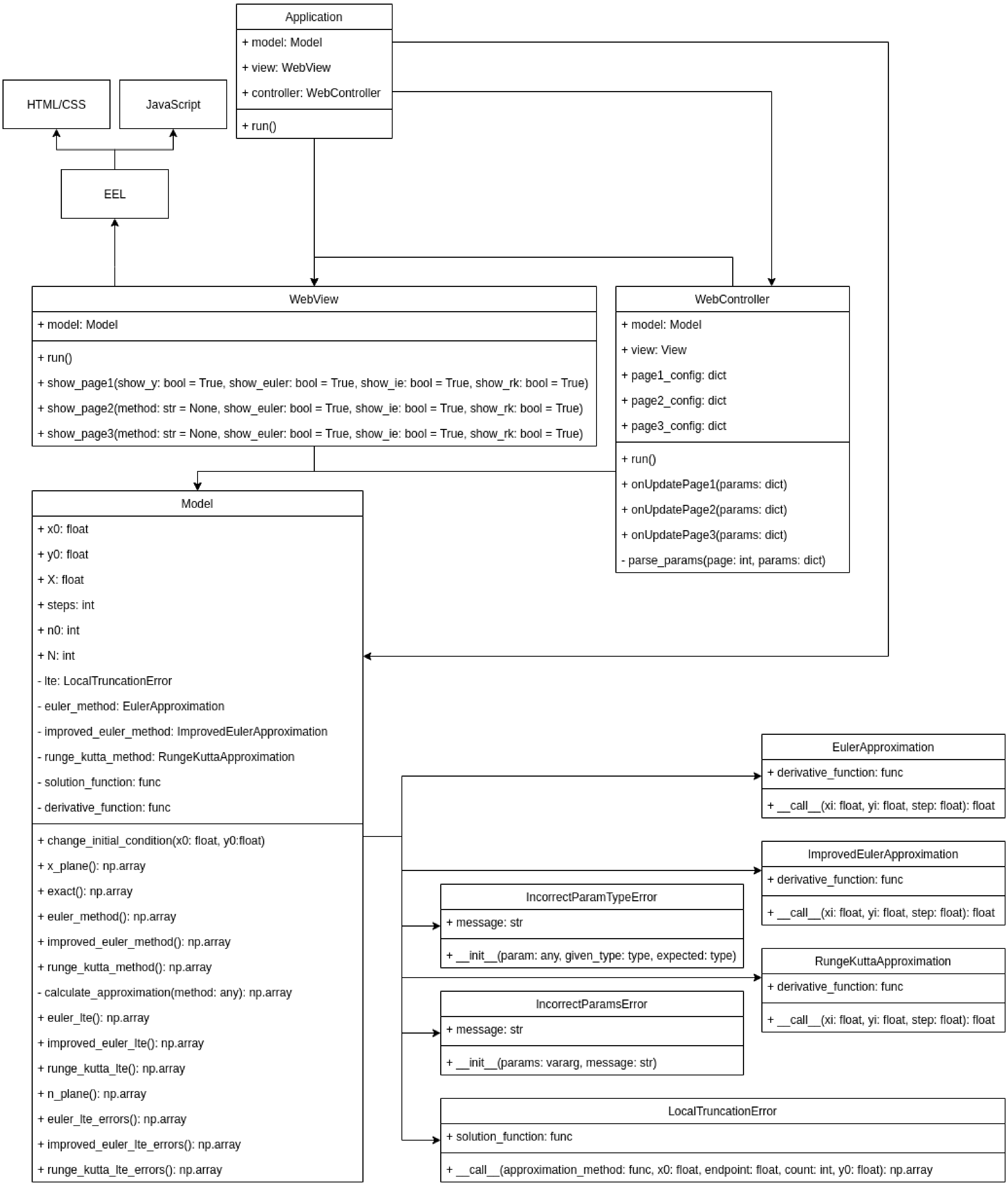
There are changing LTE of each approximation method depending on the given step.

It calculates the maximum local error on the range $[x0, X]$ for each number of steps on the range $[n0, N]$ with step 1.

- n_0 - starting number of steps
- N - end number of steps
- N - end number of steps



UML class diagram



Parts of the code

Project structure

```
project
├── app  // Main directory for the application
│   ├── controller
│   │   ├── console_controller.py
│   │   └── web_controller.py
│   ├── __init__.py
│   ├── __main__.py  // Starting point for the application
│   ├── model
│   │   ├── approximations  // Folder for approximation functions
│   │   │   ├── euler_method.py
│   │   │   ├── improved_euler_method.py
│   │   │   └── runge_kutta_method.py
│   │   ├── errors  // Folder for truncation error classes
│   │   │   ├── gte.py
│   │   │   └── lte.py
│   │   ├── exceptions  // Custom exceptions
│   │   │   ├── incorrect_params_error.py
│   │   │   └── incorrect_param_type_error.py
│   │   └── model.py  // Business logic
│   └── view
│       ├── console_view.py
│       ├── static
│       │   ├── css
│       │   │   └── style.css
│       │   ├── img
│       │   │   └── graph.png
│       │   ├── index.html
│       │   └── js
│       │       ├── controller.js
│       │       └── tabs.js
│       └── web_view.py  // Visualization of the application
├── README.md
├── report
│   └── Report.pdf  // File with report
├── requirements.txt
└── tests  // Folder for tests
    └── model
        ├── test_euler_method.py
        ├── test_gte.py
        ├── test_improved_method.py
        ├── test_lte.py
        └── test_runge_kutta_method.py
```

Run application (__main__.py)

```
if __name__ == '__main__':
    app = Application(model, view, controller)
    app.run()
```

Run graphical user interface (web_view.py)

```
def run(self) -> None:
    self._change_image({}, 1, callback_needed=False)
    eel.init('view/static')
    eel.start('index.html', size=(1000, 600))
```

Calculation of LTE (lte.py)

```
arr = np.zeros(shape=steps, dtype=np.float64)
xi = x0
y_real = y0
for i, x in enumerate(np.linspace(x0, endpoint, steps)):
    if i == 0:
        continue
    y_approximate = approximation_method(xi, y_real, step)
    y_real = self.solution_function(x)
    arr[i] = abs(y_real - y_approximate)
    xi = x
return arr
```

Plotting and saving a graph (web_view.py)

```
def _change_image(table: dict, page_number: int, callback_needed=True) -> None:
    for key in table.keys():
        if key == 'X':
            continue
        plt.plot(table['X'], table[key], label=key)
    if page_number == 1:
        plt.xlabel('x')
        plt.ylabel('y')
    elif page_number == 2:
        plt.xlabel('x')
        plt.ylabel('Error')
    elif page_number == 3:
        plt.xlabel('n')
        plt.ylabel('Error')
    if len(table) > 1:
        plt.legend()
    plt.savefig('view/static/img/graph.png', bbox_inches='tight', transparent=True)
    if callback_needed:
        eel.updateImage()()
    plt.close()
```

Tests of the application

Code for testing **local truncation error** using 3 methods of approximation:

```
def setUp(self):
    test_func = lambda x: (x * (1 + x ** 2 / 3)) / (1 - x ** 2 / 3)
    self.derivative_func = lambda x, y: (y ** 2 + x * y - x ** 2) / x ** 2

    euler_method = EulerApproximation(self.derivative_func)
    improved_euler_method = ImprovedEulerApproximation(self.derivative_func)
    runge_kutta_method = RungeKuttaApproximation(self.derivative_func)

    self.lte = LocalTruncationError(test_func)

def test_euler(self):
    expected = np.array([0., 0.087150835, 0.13986887, 0.2441393, 0.48296002, 1.1715976], dtype=np.float32)
    val = self.lte(EulerApproximation(self.derivative_func), 1, 1.5, count=6)

    self.assertIs(type(val), np.ndarray)
    self.assertEqual(len(val), 6)
    self.assertEqual(len(val), len(expected))
    np.testing.assert_array_almost_equal(val, expected)

def test_improved_euler(self):
    expected = np.array([0., 0.01368145, 0.023602538, 0.04599498, 0.106638946, 0.32514724], dtype=np.float32)
    val = self.lte(ImprovedEulerApproximation(self.derivative_func), x0=1.0, endpoint=1.5, count=6, y0=2.0)

    self.assertIs(type(val), np.ndarray)
    self.assertEqual(len(val), 6)
    self.assertEqual(len(val), len(expected))
    np.testing.assert_array_almost_equal(val, expected)

def test_rkm(self):
    expected = np.array([0., 0.000145517, 0.00025022254, 0.0005286229, 0.0014980546, 0.0067819976],
                        dtype=np.float32)
    val = self.lte(RungeKuttaApproximation(self.derivative_func), x0=1., endpoint=1.5, step=.1)

    self.assertIs(type(val), np.ndarray)
    self.assertEqual(len(val), 6)
    self.assertEqual(len(val), len(expected))
    np.testing.assert_array_almost_equal(val, expected)
```