Computational practicum

# Name: Andrey Kuzmickiy Group: BS20-04

**Analytical solution (exact solution)**

⎧

*y*′ = 2*x*(*x*2 + *y*) *y*(0) = 0

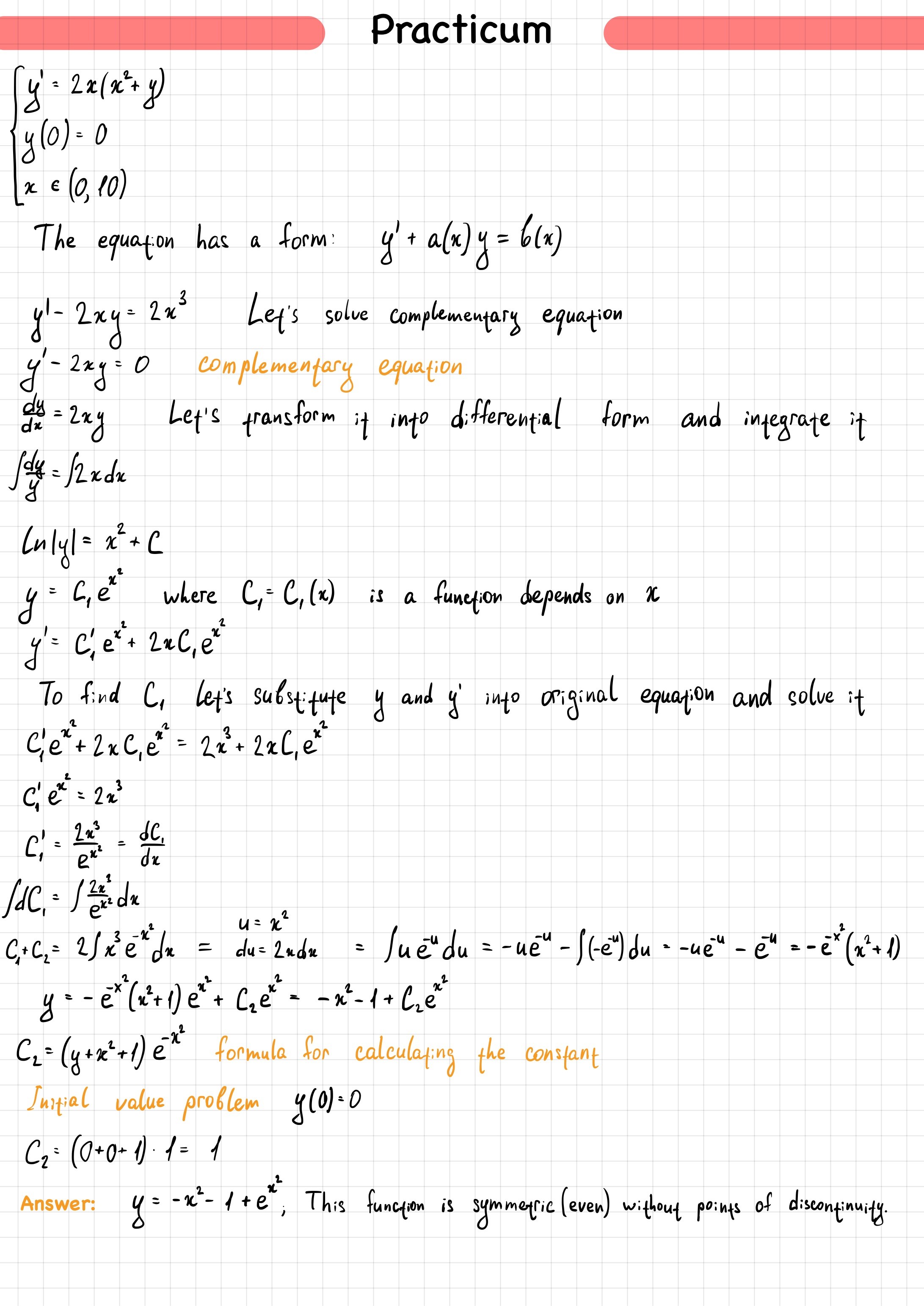
⎨

⎩

#### *x* ∈ [0, 10]

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

**Exact solution for given IVP (initial value problem)**: *y* = *ex*2 − *x*2 − 1



# Program's part

The program allows user to see the graph of the solution of the equation *y* = *C ex*2 − *x*2 − 1 with opportunity to change initial conditions, range and number of grid steps.

2

For calculating new exact solution the program use the following formula to calculate the constant *C*2: *C*2

## Graphs

### Graph of solutions

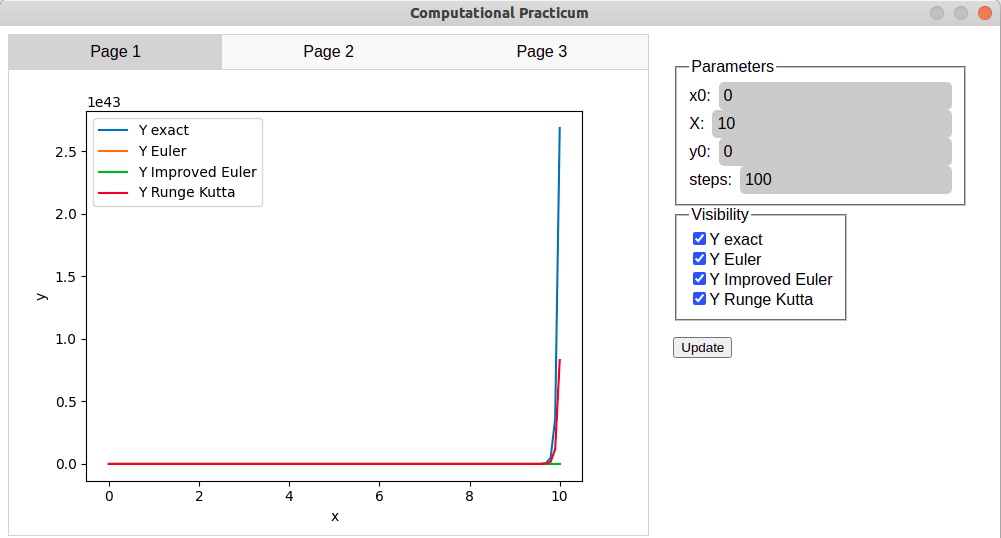
There are 4 lines represented different types of the solution: Exact solution;

Approximate solution using Euler's method;

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

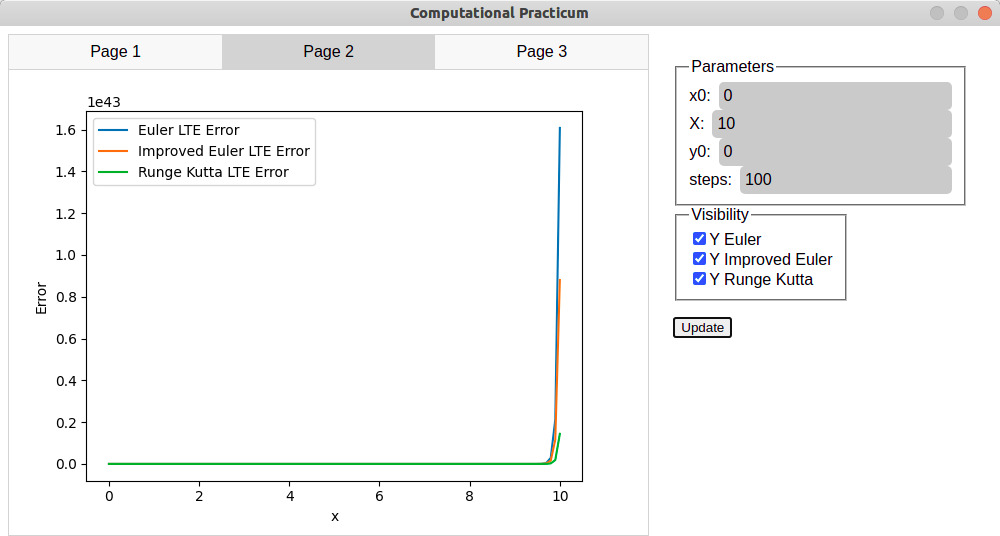
*y* − *axis* represents solution for given *x* with values ∈ [0, 2.7 ∗ 1043].

= (*y* + *x*2 + 1)*e*−*x*2



### Graph of local errors

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



### Graph of total approximation error

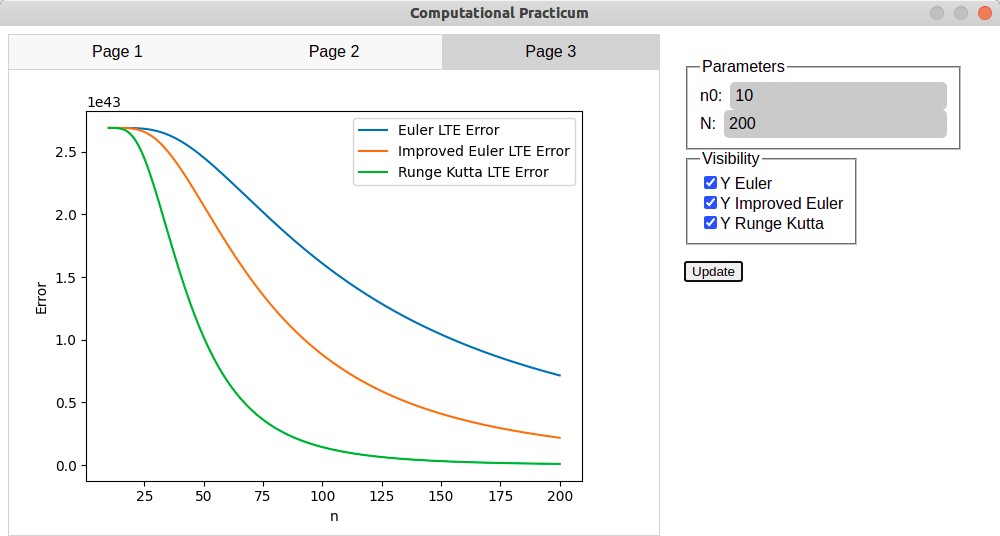
There are changing LTE of each approximation method depending on the given step.

It calculates the maximum local error on the range [*x*0, *X*] for each number of steps on the range [*n*0, *N*] with step 1.

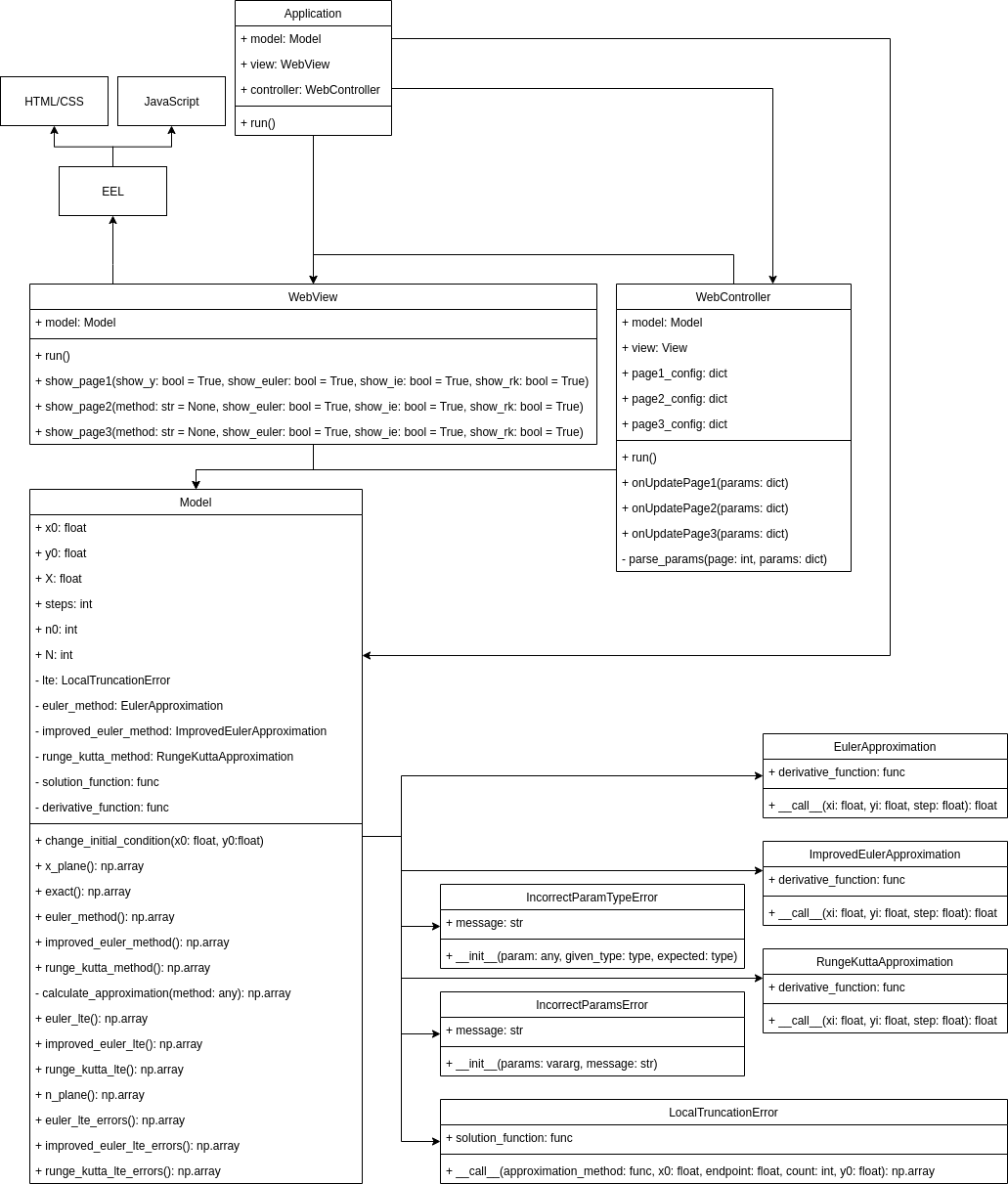
*n*0 - starting number of steps

*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)