Differential Equations
Fall 2021
Computational Practicum Assignment
Variant 19

Vladimir Makharev
v.makharev@innopolis.university
Innopolis University, B20-01

October 29, 2021

Contents

1 Report

1.1 Goal of the Computational Practicum

Given the initial value problem with the ODE of the first order and some interval. The task implement in favorite programming language following numerical methods:

- Euler's method
- Improved Euler's method
- Runge-Kutta method

in software application.
Using this application construct a corresponding approximation of the solution of a given initial value problem.
Implement the exact solution of an IVP in application.
Provide data visualization capability in the user interface of application.
Investigate the convergence of these numerical methods on different grid sizes.
Compare approximation errors of these methods plotting the corresponding chart for different grid sizes.
My implementation
Programming language: *Python 3.8*
Library for Graphical User Interface (GUI): *PySimpleGUI*
Additional useful libraries: *matplotlib, numpy, re, io*
Source code: GitHub Repository

1.2 Analytical solution of the IVP

IVP: $\begin{cases} y\prime(x) = 2x + y(x) - 3 \\ y(1) = 1 \\ x \in [1, 7] \end{cases}$

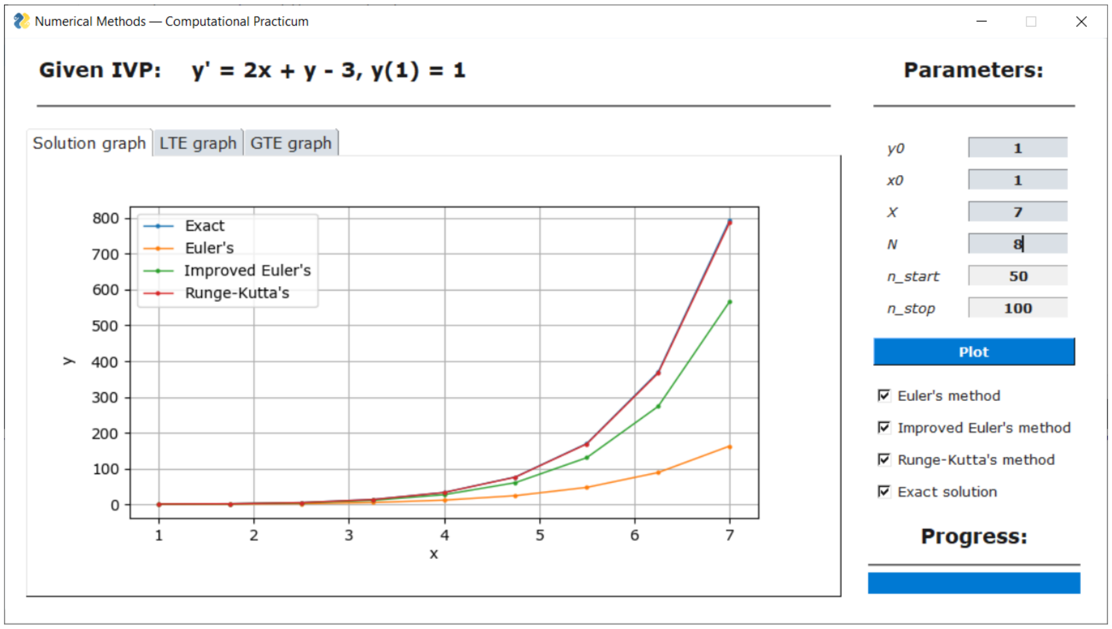
The First Order Linear Non-homogeneous Differential Equation $y\prime(x) - y(x) = 2x - 3$ is given.

Let's solve the complementary equation: $y\prime(x) - y(x) = 0$.
$\frac{dy(x)}{y(x)} = dx$ (transforming the equation into the separable form)
$\int \frac{dy(x)}{y(x)} = \int dx$ (integrating both parts)
$\ln |y_c(x)| = x + C$
$y_c(x) = Ce^x$

$y(x) = C(x)e^x$ (applying the variation of a parameter method)
$y\prime(x) = C\prime(x)e^x + C(x)e^x$

$C\prime(x)e^x + C(x)e^x - C(x)e^x = 2x - 3$ (substituting $y(x), y\prime(x)$ into the initial equation)
$C\prime(x) = \frac{2x-3}{e^x}$
$\int C\prime(x) = \int \frac{2x-3}{e^x} dx$ (integrating by parts)
$C(x) = \int (2x-3)e^{-x}dx = -(2x-3)e^{-x} - \int -2e^x dx = (3-2x)e^{-x} - 2e^{-x} = (1-2x)e^{-x} + C_1$
$y(x) = ((1-2x)e^{-x} + C_1)e^x = 1 - 2x + C_1 e^x$ (**the general solution of the equation**)
Let's substitute the initial values: $1 = 1 - 2 \cdot 1 + C_1 e^1$
$C_1 = 2e^{-1}$
We can also express the integral **constant coefficient in terms of** x, y: $C_1 = (y - 1 + 2x)e^x$
So, we obtained the particular **solution of the equation**: $y_1(x) = 1 - 2x + 2e^{x-1}$

1.3 Graphical User Interface (GUI) screenshots

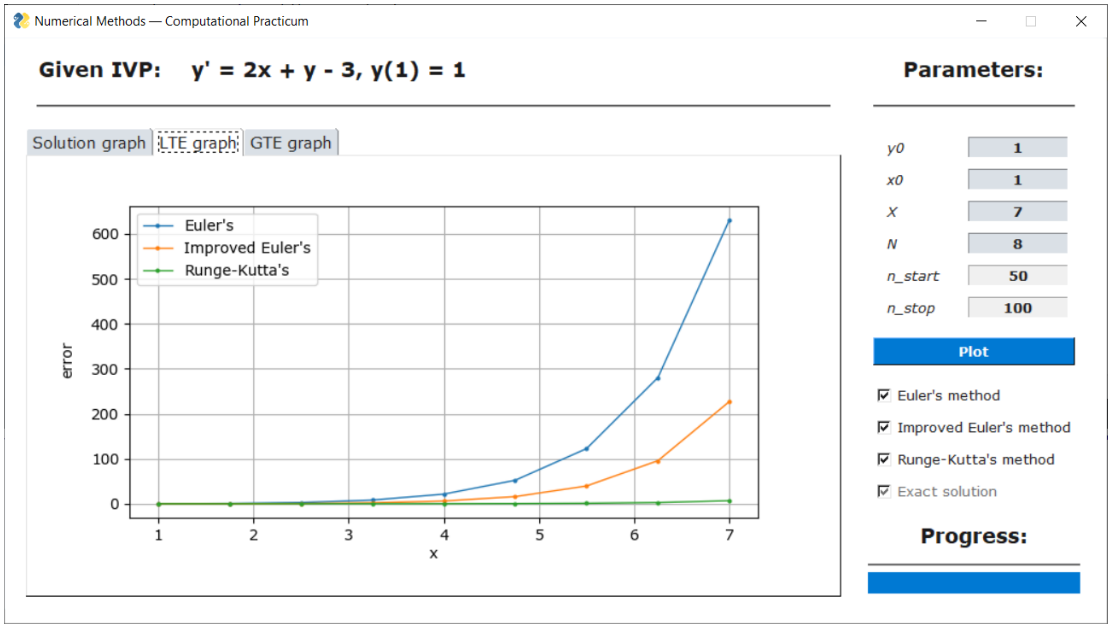


Given IVP: y' = 2x + y - 3, y(1) = 1

y0 1
x0 1
X 7
N 8
n_start 50
n_stop 100

☑ Euler's method
☑ Improved Euler's method
☑ Runge-Kutta's method
☑ Exact solution

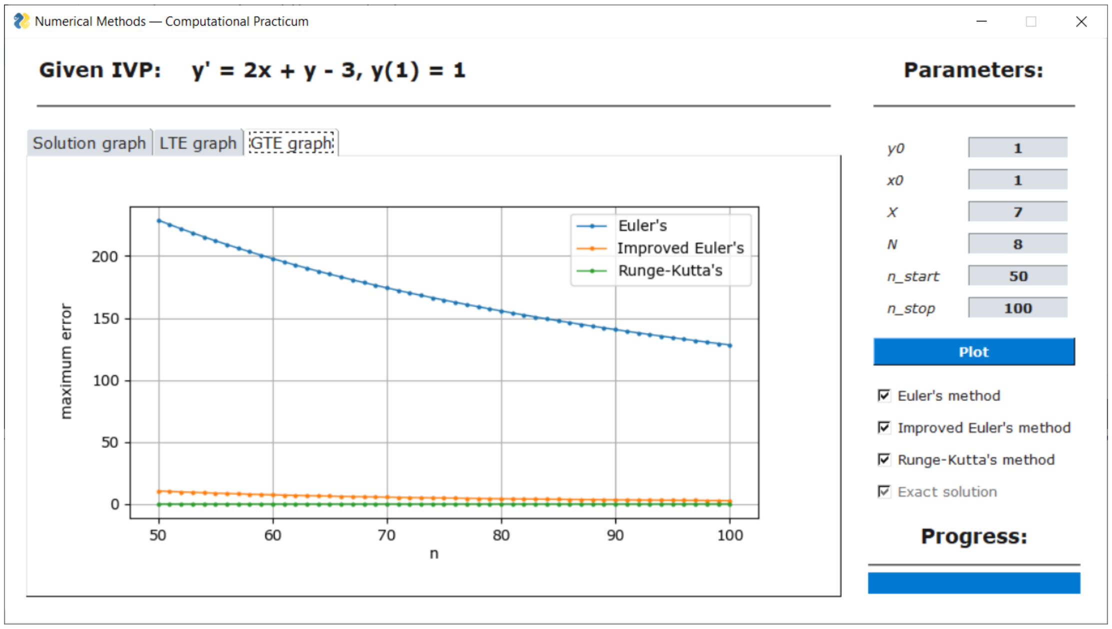
Figure 1: The Tab "Solution graph" shows exact and numerical solutions.



y0 1
x0 1
X 7
N 8
n_start 50
n_stop 100

☑ Euler's method
☑ Improved Euler's method
☑ Runge-Kutta's method
☑ Exact solution

Figure 2: The Tab "LTE graph" shows the local truncation errors (LTE).

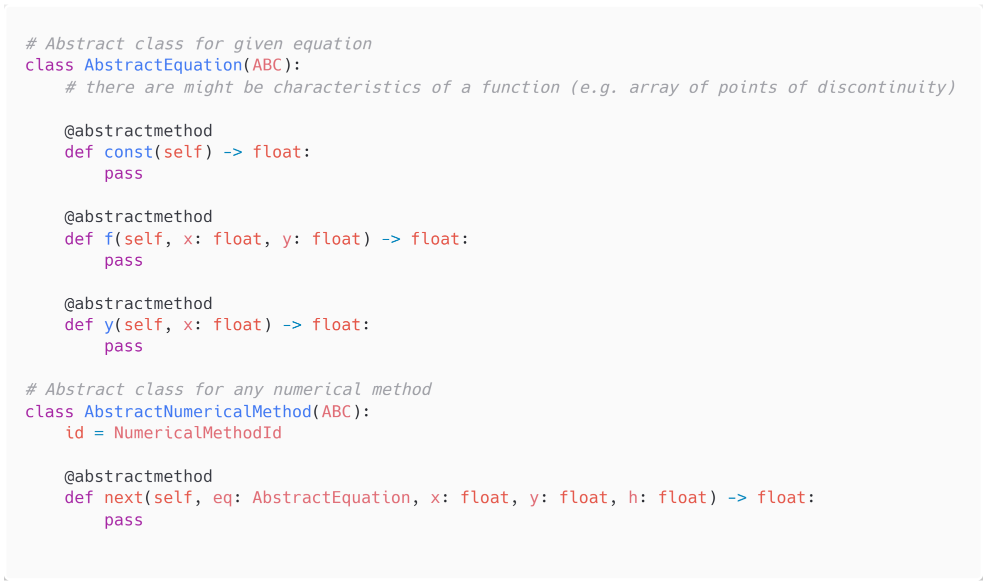


y0 1
x0 1
X 7
N 8
n_start 50
n_stop 100

☑ Euler's method
☑ Improved Euler's method
☑ Runge-Kutta's method
☑ Exact solution

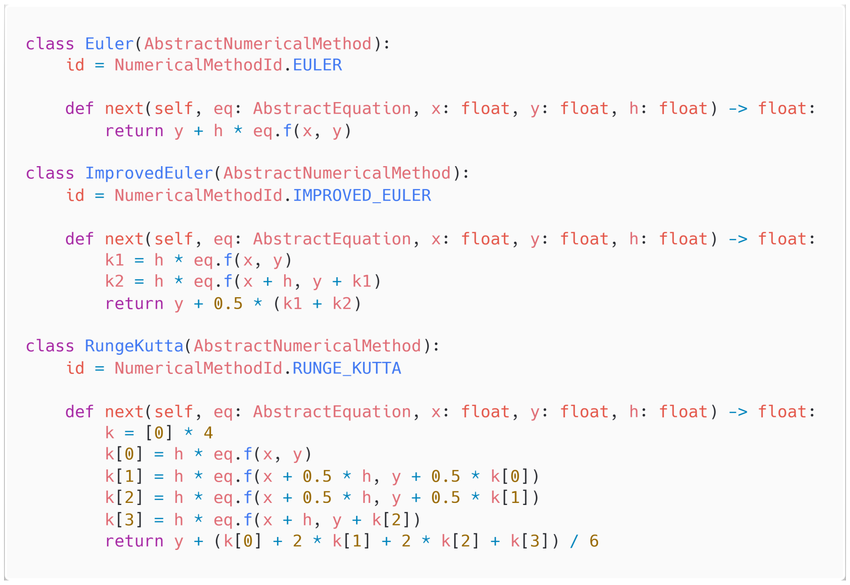
Figure 3: The Tab "GTE graph" shows the dependency of maximum LTE on the number of grid cells.

1.4 Code snippets



```python
# Abstract class for given equation
class AbstractEquation(ABC):
    # there are might be characteristics of a function (e.g. array of points of discontinuity)

    @abstractmethod
    def const(self) -> float:
        pass

    @abstractmethod
    def f(self, x: float, y: float) -> float:
        pass

    @abstractmethod
    def y(self, x: float) -> float:
        pass

# Abstract class for any numerical method
class AbstractNumericalMethod(ABC):
    id = NumericalMethodId

    @abstractmethod
    def next(self, eq: AbstractEquation, x: float, y: float, h: float) -> float:
        pass
```

Figure 4: The Abstract classes for Equation and Numerical Method.



```python
class Euler(AbstractNumericalMethod):
    id = NumericalMethodId.EULER

    def next(self, eq: AbstractEquation, x: float, y: float, h: float) -> float:
        return y + h * eq.f(x, y)

class ImprovedEuler(AbstractNumericalMethod):
    id = NumericalMethodId.IMPROVED_EULER

    def next(self, eq: AbstractEquation, x: float, y: float, h: float) -> float:
        k1 = h * eq.f(x, y)
        k2 = h * eq.f(x + h, y + k1)
        return y + 0.5 * (k1 + k2)

class RungeKutta(AbstractNumericalMethod):
    id = NumericalMethodId.RUNGE_KUTTA

    def next(self, eq: AbstractEquation, x: float, y: float, h: float) -> float:
        k = [0] * 4
        k[0] = h * eq.f(x, y)
        k[1] = h * eq.f(x + 0.5 * h, y + 0.5 * k[0])
        k[2] = h * eq.f(x + 0.5 * h, y + 0.5 * k[1])
        k[3] = h * eq.f(x + h, y + k[2])
        return y + (k[0] + 2 * k[1] + 2 * k[2] + k[3]) / 6
```

Figure 5: The Classes for numerical methods: Euler's, Improved Euler's, Runge-Kutta.

```python
class Equation19(AbstractEquation):
    # Constructor for the 19th equation
    def __init__(self, x0: float, y0: float):
        # Initial values y(1) = 1
        self.x0 = x0
        self.y0 = y0

    # Integral constant coefficient = (y0 - 1 + 2 * x0) / exp(x0)
    def const(self) -> float:
        return (self.y0 - 1 + 2 * self.x0) / exp(self.x0)

    # Given function y' = f(x, y) = 2x + y - 3
    def f(self, x: float, y: float) -> float:
        return 2 * x + y - 3

    # Solution y = 1 - 2x + const * exp(x)
    def y(self, x: float) -> float:
        return 1 - 2 * x + self.const() * exp(x)
```

Figure 6: The Class for given equation of variant 19.

```python
# View
class Analyzer:
    # ...

    def plot_solution(self, plot, cbs):
        #...

    def plot_lte(self, plot, cbs):
        # ...

    def plot_gte(self, start: int, stop: int, plot, cbs):
        for i, solver in enumerate(self.solvers):
            if cbs[i + 1]:
                axis_size = stop + 1 - start
                if axis_size < 0:
                    start, stop = stop, start
                    axis_size = stop + 1 - start
                x_axis_gte = np.arange(start, stop + 1)  # Returns array [n0, n0 + 1, ..., nf]
                y_axis_gte = np.empty(axis_size)
                for index, ni in enumerate(range(start, stop + 1)):
                    temp_solver = solver
                    temp_x0 = self.eq.x0
                    temp_xf = solver.x_axis[len(solver.x_axis) - 1]
                    temp_step = (temp_xf - temp_x0) / ni
                    temp_solver.solve(ni, temp_step)  # TODO: Try except
                    y_axis_gte[index] = temp_solver.gte()
                plot.plot(x_axis_gte, y_axis_gte, label=solver.method.id,
                          marker='o', linewidth=1, markersize=2)
        if True in cbs[1:]:
            plot.grid()
            plot.legend()
        else:
            plot.axis('off')
```

Figure 7: The View class of the MVC programming design pattern with one of the plotting function.

1.5 Unified Modeling Language (UML) class diagram

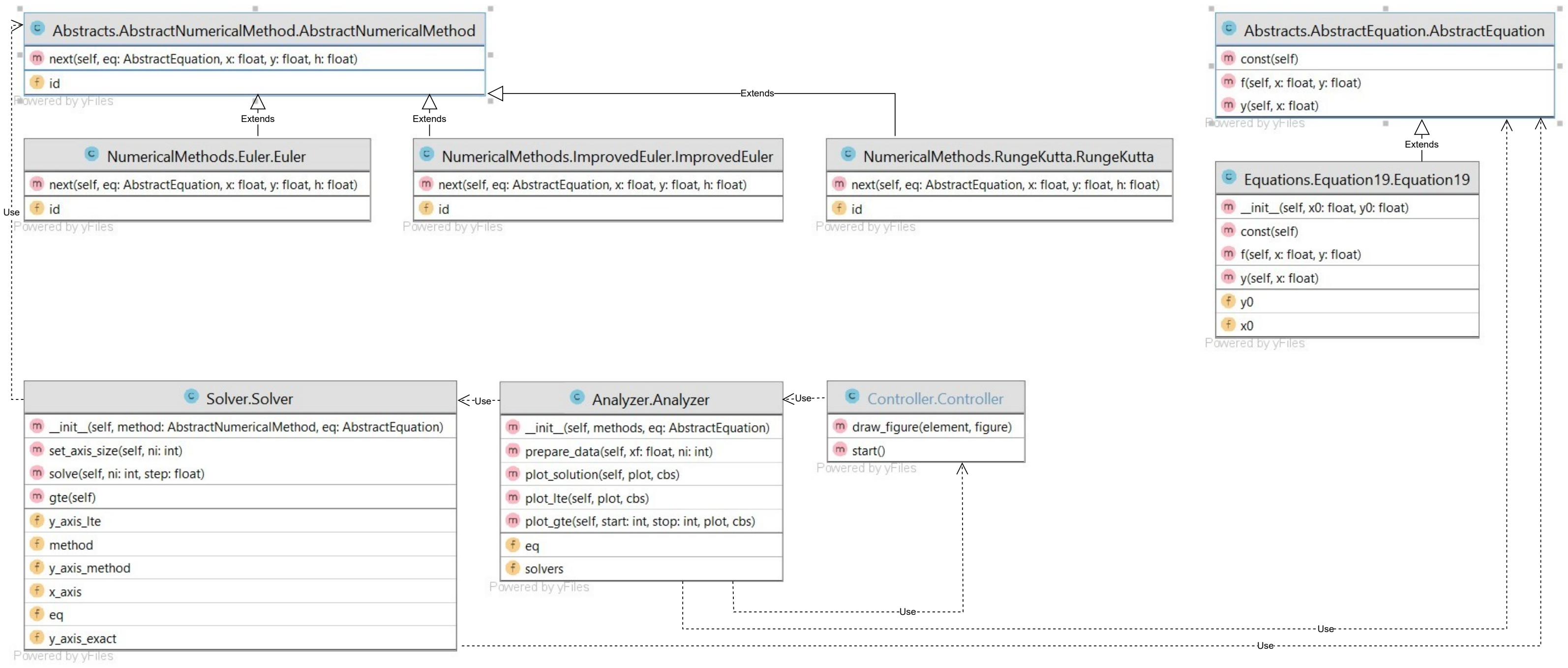

Abstracts.AbstractNumericalMethod.AbstractNumericalMethod
next(self, eq: AbstractEquation, x: float, y: float, h: float)
id
Extends
Extends
Extends
NumericalMethods.Euler.Euler
next(self, eq: AbstractEquation, x: float, y: float, h: float)
id
NumericalMethods.ImprovedEuler.ImprovedEuler
next(self, eq: AbstractEquation, x: float, y: float, h: float)
id
NumericalMethods.RungeKutta.RungeKutta
next(self, eq: AbstractEquation, x: float, y: float, h: float)
id
Abstracts.AbstractEquation.AbstractEquation
const(self)
f(self, x: float, y: float)
y(self, x: float)
Extends
Equations.Equation19.Equation19
__init__(self, x0: float, y0: float)
const(self)
f(self, x: float, y: float)
y(self, x: float)
y0
x0
Use
Solver.Solver
__init__(self, method: AbstractNumericalMethod, eq: AbstractEquation)
set_axis_size(self, ni: int)
solve(self, ni: int, step: float)
gte(self)
y_axis_lte
method
y_axis_method
x_axis
eq
y_axis_exact
Use
Analyzer.Analyzer
__init__(self, methods, eq: AbstractEquation)
prepare_data(self, xf: float, ni: int)
plot_solution(self, plot, cbs)
plot_lte(self, plot, cbs)
plot_gte(self, start: int, stop: int, plot, cbs)
eq
solvers
Use
Controller.Controller
draw_figure(element, figure)
start()
Use
Use
Use
Powered by yFiles

Figure 8: The UML class diagram.