Computational Practicum Report

Arseny Dremin BS20-06

Goals

Create an application that computes the numerical solution to the given ODE using various methods, such as:

- Euler Method

$$y_{i+1} = y_i + hf(x_i, y_i)$$

- Improved Euler Method

$$y_{i+1} = y_i + \frac{h}{2}\left(f(x_i, y_i) + f(x_{i+1}, y_i + hf(x_i, y_i))\right)$$

- Runge-Kutta Method

$$y_{i+1} = y_i + \frac{h}{6}(k_{1i} + 2k_{2i} + 2k_{3i} + k_{4i})$$

Computation range and initial value can be changed by the user.
Compare local and global truncation errors for each of the methods and find out the best one for the specified problem.

Exact Solution

$$y' = -y^2/3 - 2/3x^2$$

(1) $y' + y^2/3 = -\dfrac{2}{3x^2}$ (Ricatti Equation)

Let $y = \dfrac{c}{x}$ (Particular Solution)

$y' = -\dfrac{c}{x^2}$

$\xrightarrow{(1)}$ $-\dfrac{c}{x^2} + \dfrac{c^2}{3x^2} + \dfrac{2}{3x^2} = 0$

$\dfrac{c^2 - 3c + 2}{3x^2} = 0$ $\boxed{x \neq 0}$ (discontinuity)

$c = 1; 2$

We can choose any, e.g. $y = \dfrac{1}{x}$

$y = \dfrac{1}{x}$ (Particular Solution)

(2) $y = \dfrac{1}{x} + z(x)$ (General Solution)

$y' = -\dfrac{1}{x^2} + z'(x)$

$\xrightarrow{(1)}$ $-\dfrac{3}{3x^2} + z' + \dfrac{1}{3}\left(\dfrac{1}{x} + z\right)^2 = -\dfrac{2}{3x^2}$

(3) $z' + \dfrac{z^2}{3} + \dfrac{2}{3x} \cdot z = 0$

(Bernoulli Equation)

$m = 2$

Let $u = z^{1-m} = \dfrac{1}{z}$

$u' = -\dfrac{z'}{z^2}$

Divide (3) by z^2: $\boxed{z = 0}$

(3*) $\dfrac{z'}{z^2} + \dfrac{1}{3} + \dfrac{2}{3x} \cdot \dfrac{1}{z} = 0$

$-u' + \dfrac{2}{3x} \cdot u + \dfrac{1}{3} = 0$

$u' - \dfrac{2}{3x} u - \dfrac{1}{3} = 0$ (Linear ODE)

$u' - \dfrac{2}{3x} u = 0$ (Complementary)

$\dfrac{du}{u} = \dfrac{2}{3x} dx$

$u = Cx^{\frac{2}{3}}$

Variation of parameters:

$u = C(x) x^{\frac{2}{3}}$

$u' = c' x^{\frac{2}{3}} + \frac{2}{3} c x^{-\frac{1}{3}}$

substitute

$c' x^{\frac{2}{3}} + \frac{2}{3} c x^{-\frac{1}{3}} - \frac{2}{3x} \cdot c x^{\frac{2}{3}} - \frac{1}{3} = 0$

$3c' x^{\frac{2}{3}} + 2c x^{-\frac{1}{3}} + 2c x^{-\frac{1}{3}} - 1 = 0$

$c' x^{\frac{2}{3}} = \frac{1}{3}$

$dc = \frac{1}{3} x^{-\frac{2}{3}} dx$

$c = x^{\frac{1}{3}} + C_0$ ← constant

$\longrightarrow u = x + C_0 x^{\frac{2}{3}}$

$z = \dfrac{1}{u} = \dfrac{1}{x + C_0 x^{\frac{2}{3}}}$

$$y = \frac{1}{x} + \frac{1}{x + C_0 x^{\frac{2}{3}}}$$

(General solution)

Special case

$z = 0 \Rightarrow y = \frac{1}{x}$ - solution

Lets check it: $-\frac{1}{x^2} = -\frac{1}{3x^2} - \frac{2}{3x^2}$

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

So $y = \frac{1}{x}$ (particular solution)

Solutions: $\left[\begin{array}{l} y = \frac{1}{x} + \frac{1}{x + C_0 x^{\frac{2}{3}}} \\ y = \frac{1}{x} \end{array}\right.$, $x \neq 0$ (discontinuity)

IVP: $\begin{cases} x_0 = 1 \\ y_0 = 2 \end{cases} \Rightarrow 2 = 1 + \frac{1}{1 + C_0}$

$C_0 = 0$

IVP Answer: $\left[\begin{array}{l} y = \frac{1}{x} \\ y = \frac{2}{x} \end{array}\right.$ $(x \neq 0)$

Coefficient equation

$$y = \frac{1}{x} + \frac{1}{x + C_0 x^{\frac{2}{3}}}$$

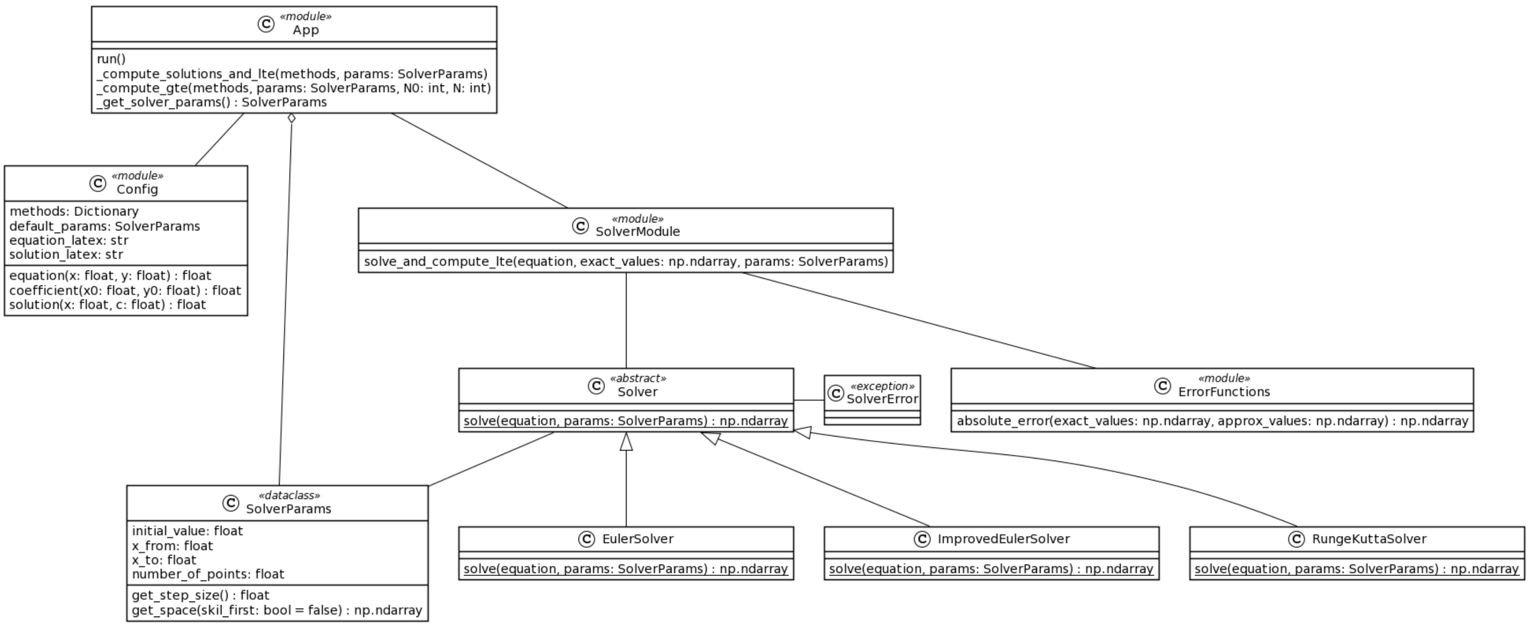
$$\Downarrow$$

$$xy(x + C_0 x^{\frac{2}{3}}) = x + C_0 x^{\frac{2}{3}} + x$$

$$x^2 y + C_0 x^{\frac{5}{3}} y = 2x + C_0 x^{\frac{2}{3}}$$

$$C_0 = \frac{x^{\frac{1}{3}}(2 - xy)}{xy - 1}$$

UML Diagram



[Full diagram link]

Code

The application is composed of two main components:

- App module (responsible for all user interface interactions)
- Solver module (responsible purely for computing errors and solutions).

Solver module contains multiple solver classes (inherited from the main Solver abstract class). New solver classes can be added to implement other methods.
The Solver abstract class has a single static method: solve, which takes an equation and SolverParams dataclass and returns an array of solved points. Inherited classes override this method.
The app also has a config file where the user can configure the equation to be solved as well as the methods to be used.

Solver Params dataclass.

```
class SolverParams:
    """Class for storing integration parameters

    Args:
        initial_value (float): value of y(x_from)
        x_from (float): integration range start
        x_to (float): integration range end
        number_of_points (int): number of points to integrate at

    """
    initial_value: float
    x_from: float
    x_to: float
    number_of_points: int

    def get_step_size(self) -> float:
        """Computes step size for integration"""
        return (self.x_to - self.x_from) / (self.number_of_points - 1)

    def get_space(self, skip_last=False) -> np.ndarray:
        """Get an array of all points inside the params range"""
        if skip_last:
            return np.linspace(self.x_from, self.x_to, self.number_of_points)[:-1]
        return np.linspace(self.x_from, self.x_to, self.number_of_points)
```

Below is the implementation of EulerSolver class, other inherited classes follow similar implementation.

```
class EulerSolver(Solver):
    @staticmethod
    def solve(equation, params: SolverParams) -> np.ndarray:
        """Solve the differential equation using Euler Method

        Args:
            equation (Callable): equation of type f(x, y)
            params (SolverParams): steps, initial value and range

        Returns:
            np.ndarray: approximated y values of the function
        """

        step_size = params.get_step_size()
        if step_size >= 1:
            raise SolverError(f"Solver step size {step_size} is >= 1")

        y = params.initial_value
        values = [y]
        for x in params.get_space(skip_last=True):
            y += step_size * equation(x, y)
            values.append(y)
        return np.array(values)
```

User interactions are processed by the [Streamlit]() library which provides a web interface and graphing tools for python programs. Graphing is also done by [Streamlit]() with the use of [Altair]() graphs.
App module uses [Streamlit]() to get user parameters and calls methods from solver classes to get a solution.

Below is the example of getting user parameters and using them to compute solutions.

```
# Get integration parameters
with params_col:
    params = _get_solver_params()

# Solve using given methods and compute errors
try:
    solution_data, lte_data = _compute_solutions_and_lte(config.methods, params)
except SolverError as e:
    solutions_col.error(e)
    return
```

Below is the actual code for computing solutions and LTE.

```
def _compute_solutions_and_lte(methods, params: SolverParams):
    solve_space = params.get_space()

    # Solve using different methods
    solutions, errors = {}, {}
    exact_solution = ExactSolver.solve(config.solution, config.coefficient, params)
    for name, solver in methods.items():
        solutions[name], errors[name] = \
            solve_and_compute_lte(solver, config.equation, exact_solution, params)

    solutions['_exact'] = exact_solution
    solution_data = pd.DataFrame(solutions, index=solve_space)
    error_data = pd.DataFrame(errors, index=solve_space)

    return solution_data, error_data
```

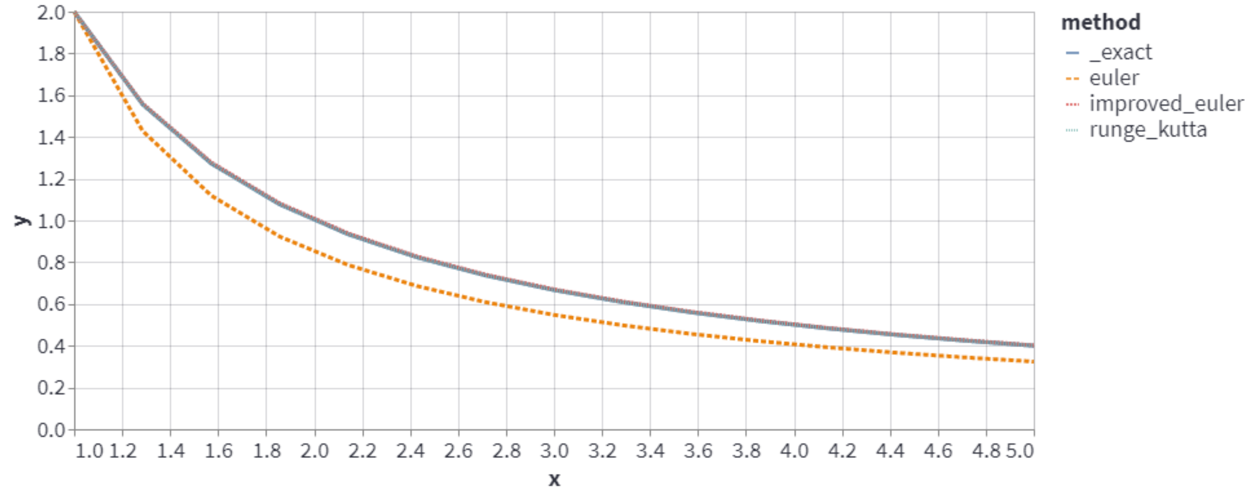
Code for computing GTE.

```
def _compute_gte(methods, params: SolverParams, N0: int, N: int):
    errors = {name: [] for name in methods}

    for n in range(N0, N):
        n_params = SolverParams(params.initial_value, params.x_from, params.x_to, n)
        exact_solution = ExactSolver.solve(config.solution, config.coefficient, n_params)
        for name, solver in methods.items():
            errors[name].append(np.max(solve_and_compute_lte(solver, config.equation, exact_solution, n_params)[1]))

    # GTE Plot Dataframe
    gte_data = pd.DataFrame(errors, index=range(N0, N))

    return gte_data
```

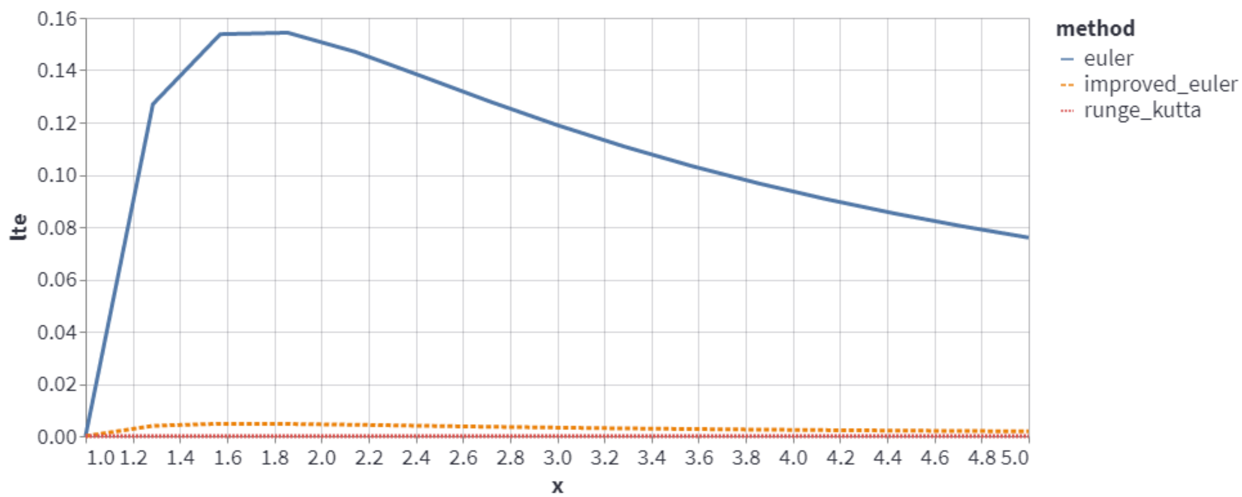
[Full code repo on github]()

Charts

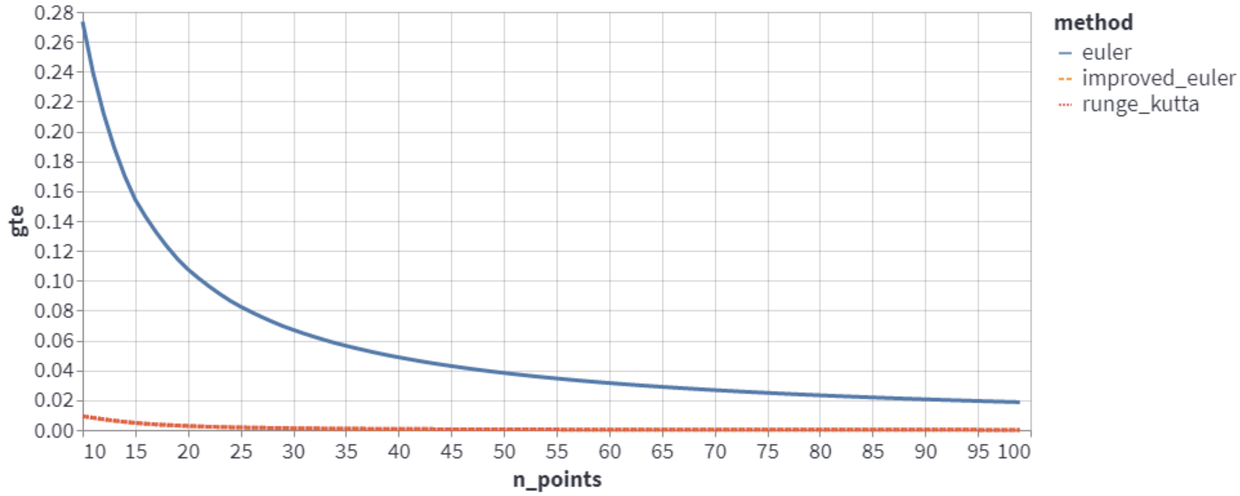
Even for a small number of steps (15 in the picture below) all methods seem to be converging towards the exact solution.
Improved Euler and RK4 almost match the exact solution curve.



Local errors for the chart above (RK4 error is negligible compared to Euler)



Global errors are also decreasing as the number of steps increases.



Results

As evident by the charts, Improved Euler Method and Runge-Kutta (k=4) methods give the closest results to the analytic solution, even for big step sizes. The Euler method has a much higher truncation error compared to the other two methods.