Differential Equations
Computational Practicum

Yusuf Mesbah

Problem

Initial value problem:

$$y' = e^{2x} + e^{x} + y^{2} + 2ye^{x},\ y(0) = 0,\ x \in (0,\ 15)$$

Analytical Solution

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

Numerical Approximations

Three different numerical approximation methods were used to estimate the solution to the differential equation without solving it analytically: Euler, Improved Euler, and Runge-Kutta. Their errors were then measured to analyze their accuracy and compared to the exact solution. All of them are applied only to the given domain: $x \in (1,10)$ and the initial value problem:

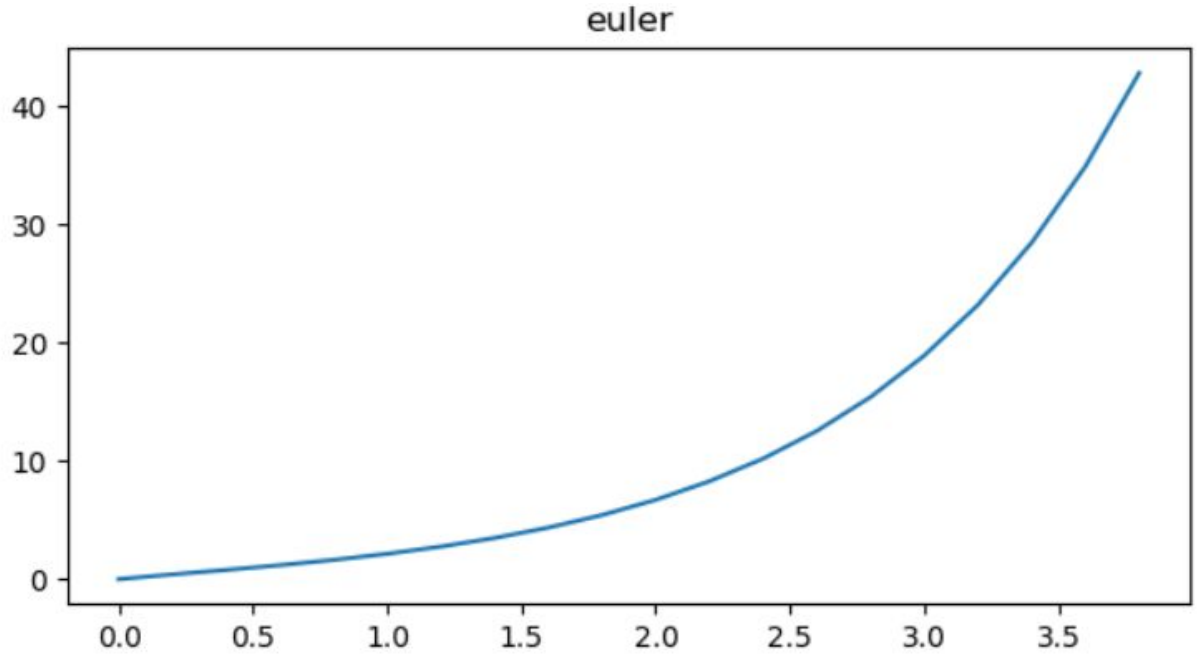
$$y'(x) = f\big(x, y(x)\big), \qquad y(x_0) = y_0$$

Euler's Method

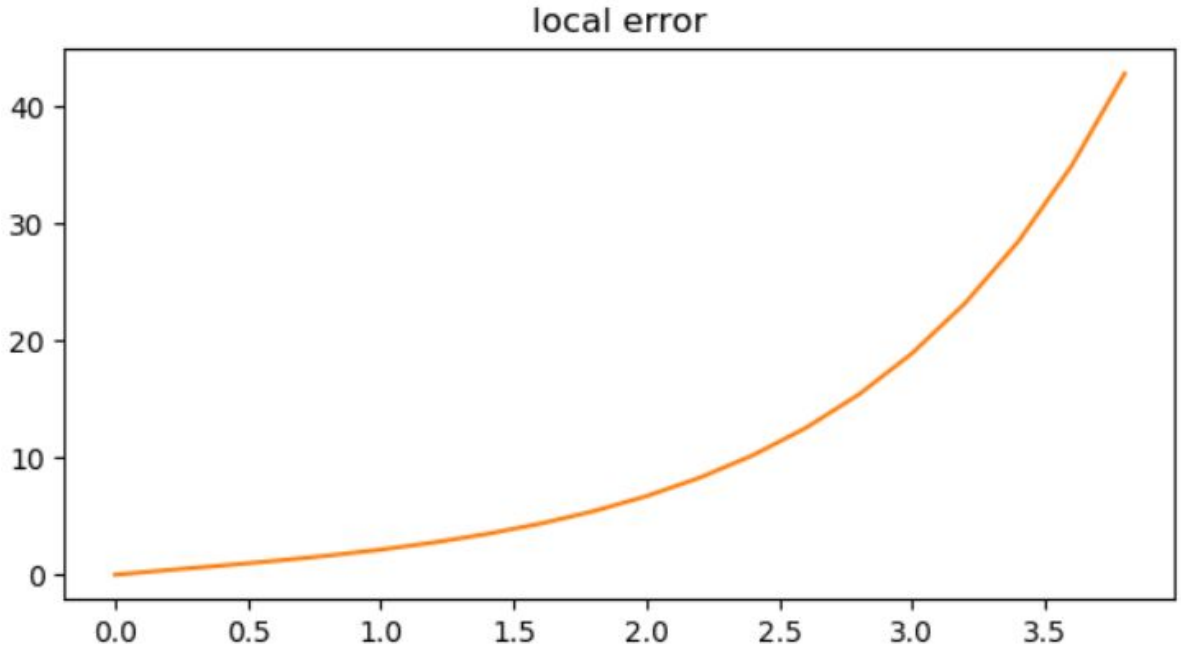
Euler's method is defined as follows:

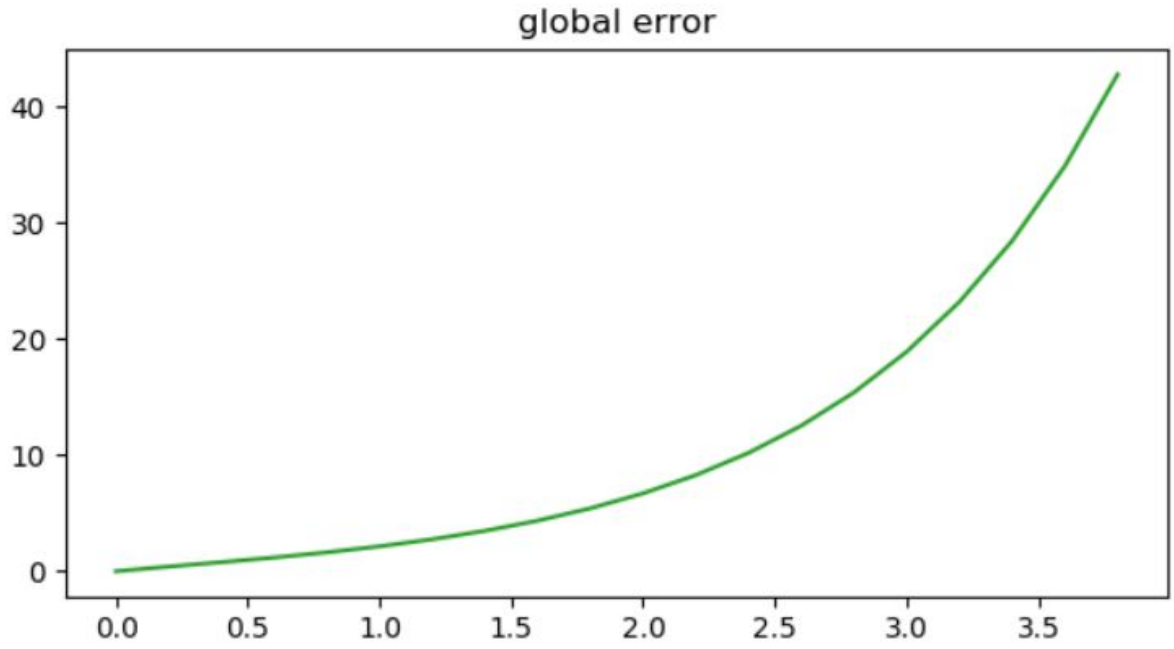
$$y_{n+1} = y_n + hf(x_n, y_n)$$
$$x_n = x_0 + nh$$

Where n is the step number, and h is the step size. The step size, h, affects the resolution accuracy of the approximation. The approximation approaches the exact solution as $\hbar \to 0$. For a particular subdomain of the function $(x0, X)$, we can define the number of steps, N, such that $\hbar = X - x0N$.



The global error for a particular N is the difference between the approximation and the solution at the last point of the domain. It is proportional to the step size h, so it approaches 0 as h approaches 0





Improved Euler method

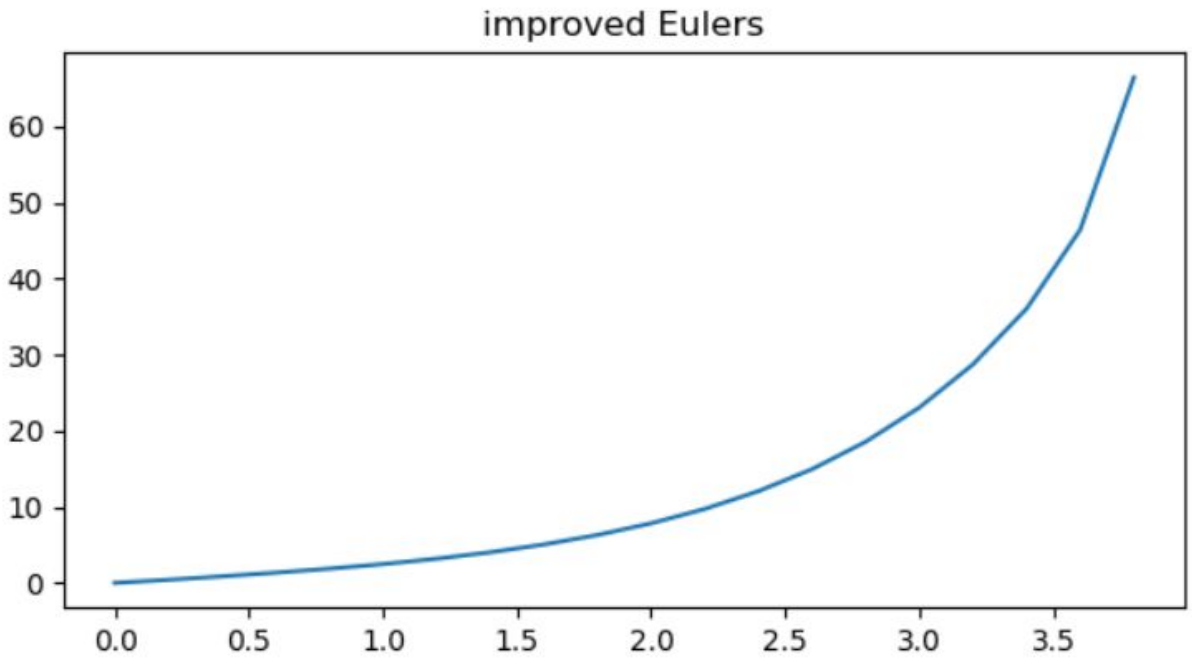
This method (also called modified Euler's method, Heun's method, or explicit trapezoidal method) is an improvement over Euler's method by

calculating an intermediate value:

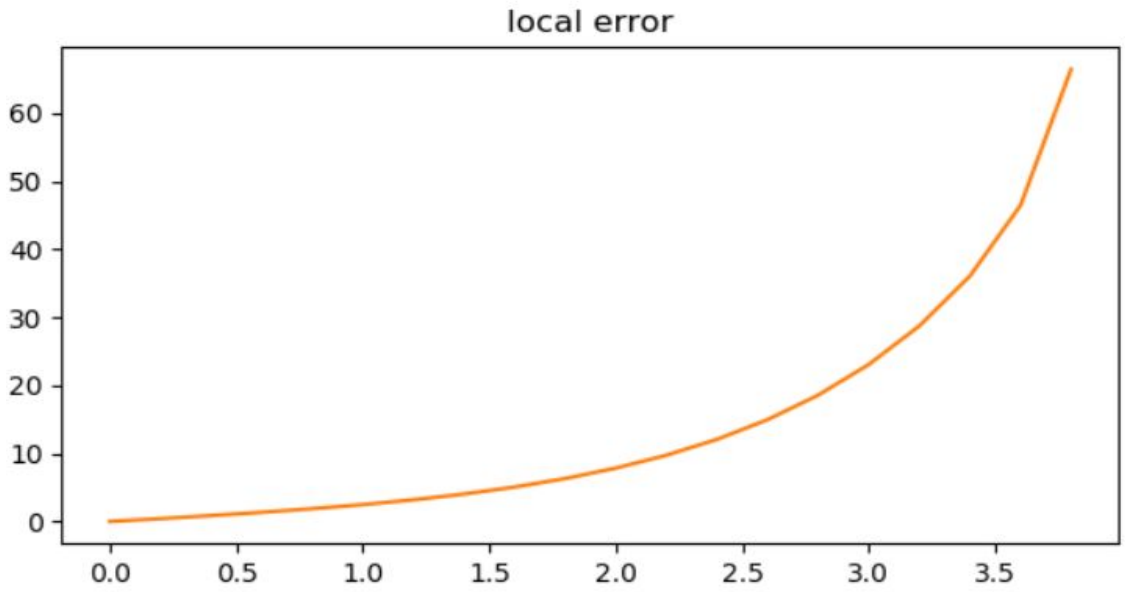
$$\tilde{y}_{n+1} = y_n + hf(x_n, y_n)$$

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, \tilde{y}_{n+1})]$$

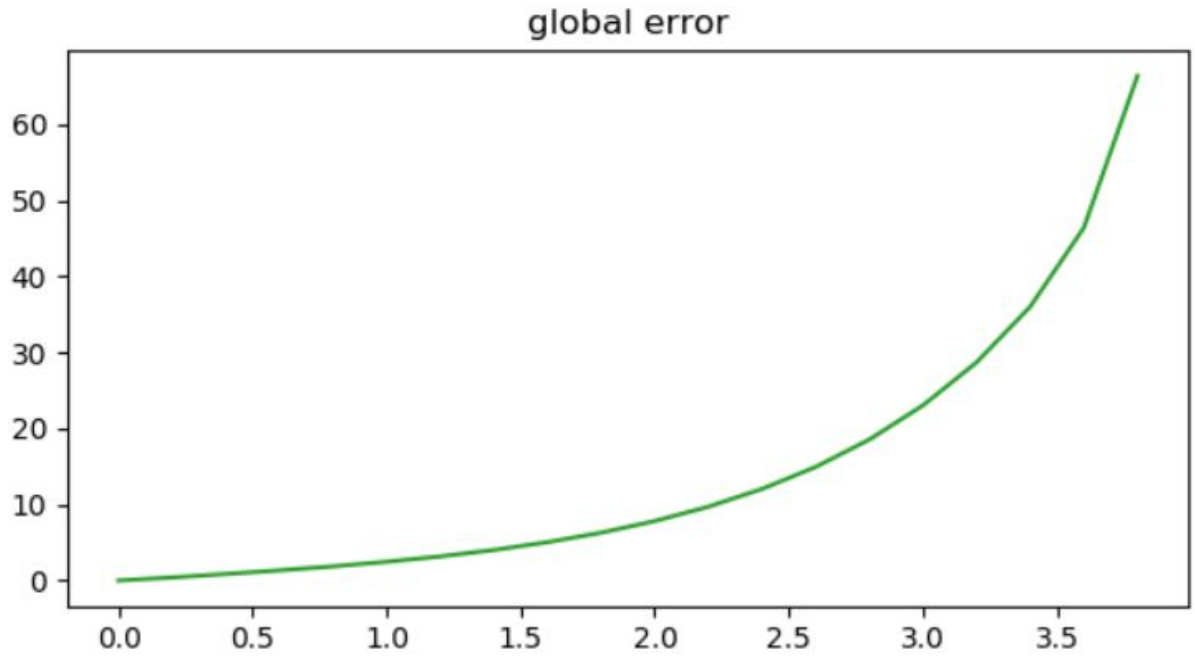
$$x_{n+1} = x_n + h$$



Local Error



Global



Runge-Kutta Method

The Runge-Kutta method is defined by adding more intermediate steps to the calculation:

$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$$
$$k_3 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right)$$
$$k_4 = hf(x_n + h, y_n + k_3)$$
$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$
$$x_{n+1} = x_n + h$$

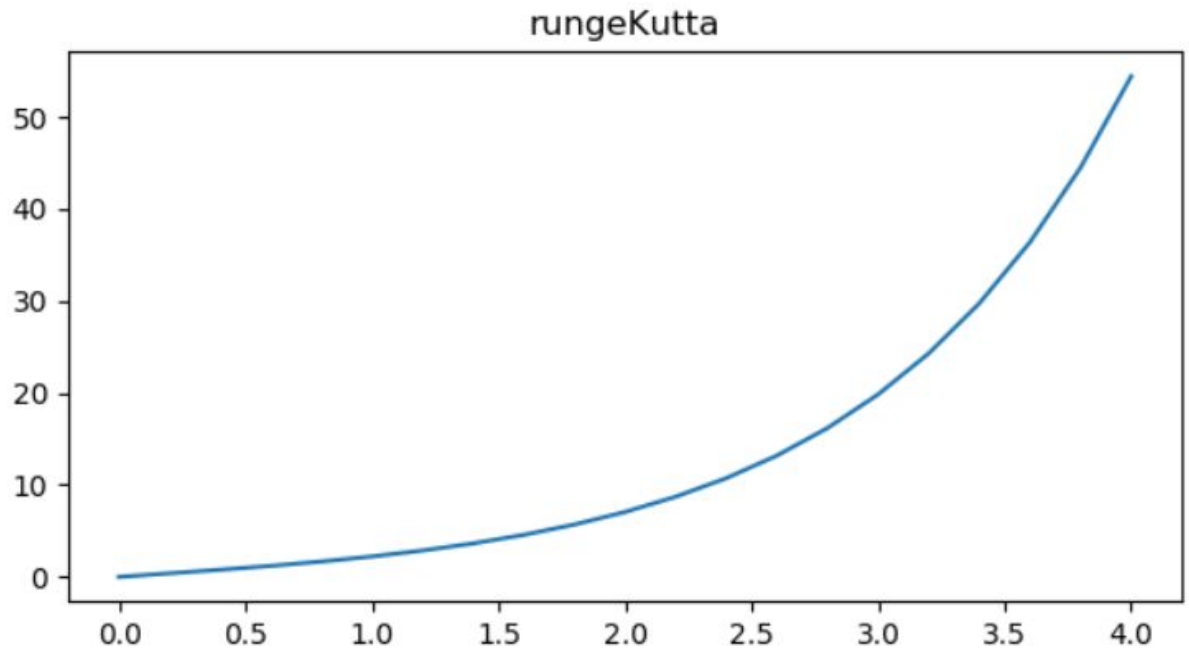
Where, according to Wikipedia,

• $k1$ is the increment based on the slope at the beginning of the interval, using y;

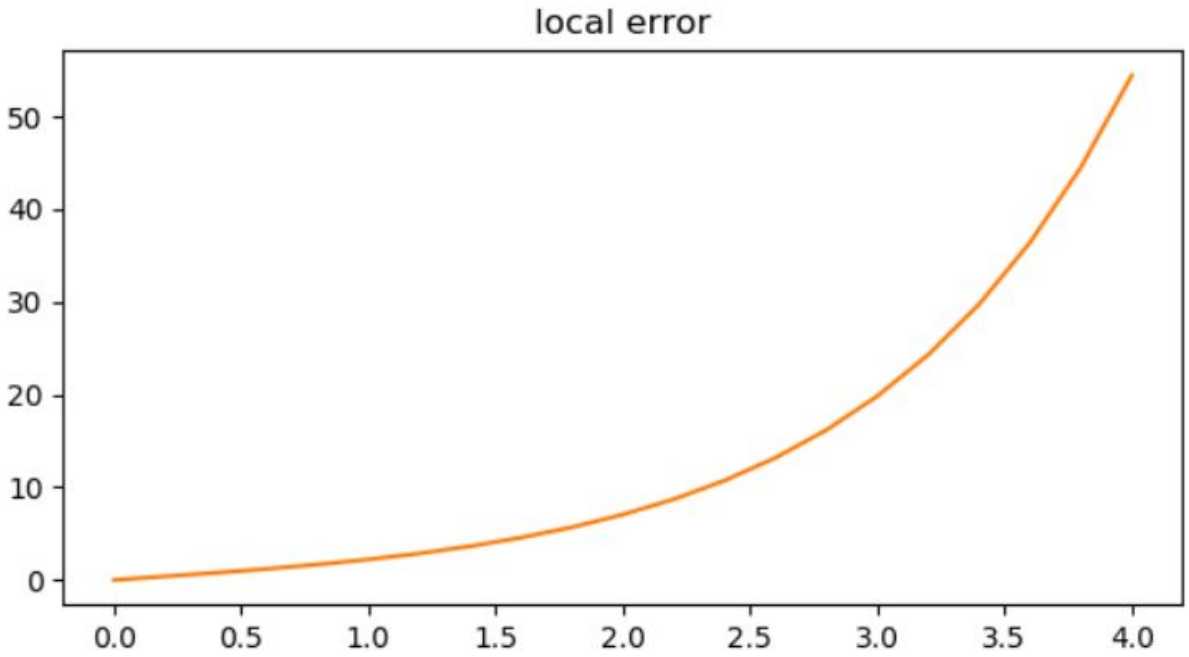
• $k2$ is the increment based on the slope at the midpoint of the interval, using y and $k1$;

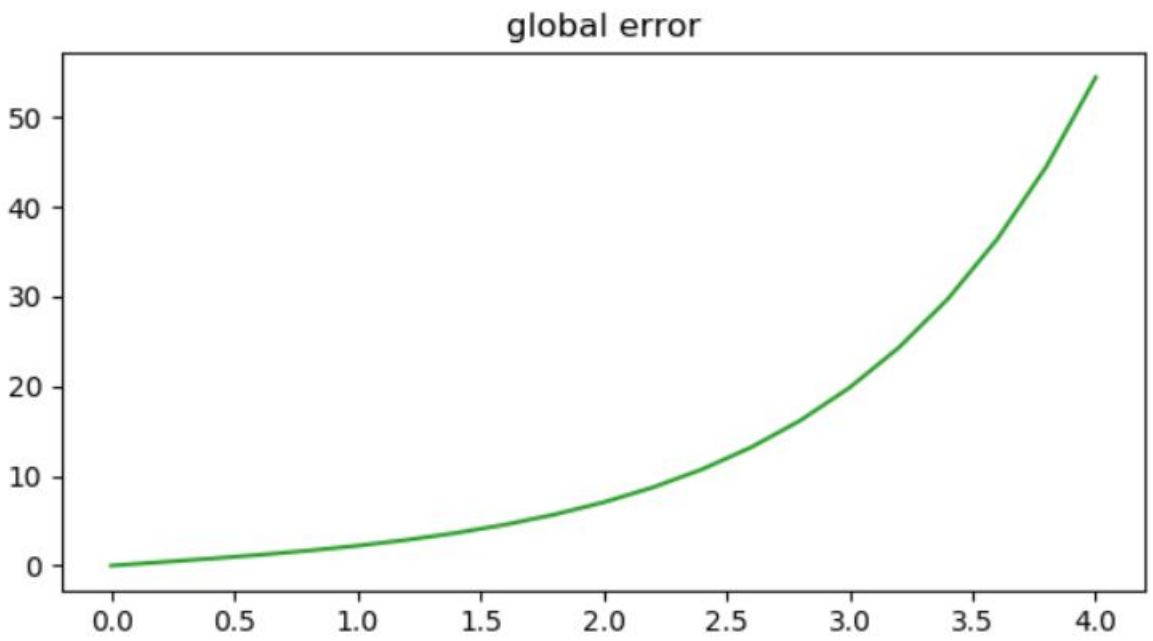
• $k3$ is again the increment based on the slope at the midpoint, but now using y and $k2$;

• $k4$ is the increment based on the slope at the end of the interval, using y and $k3$.

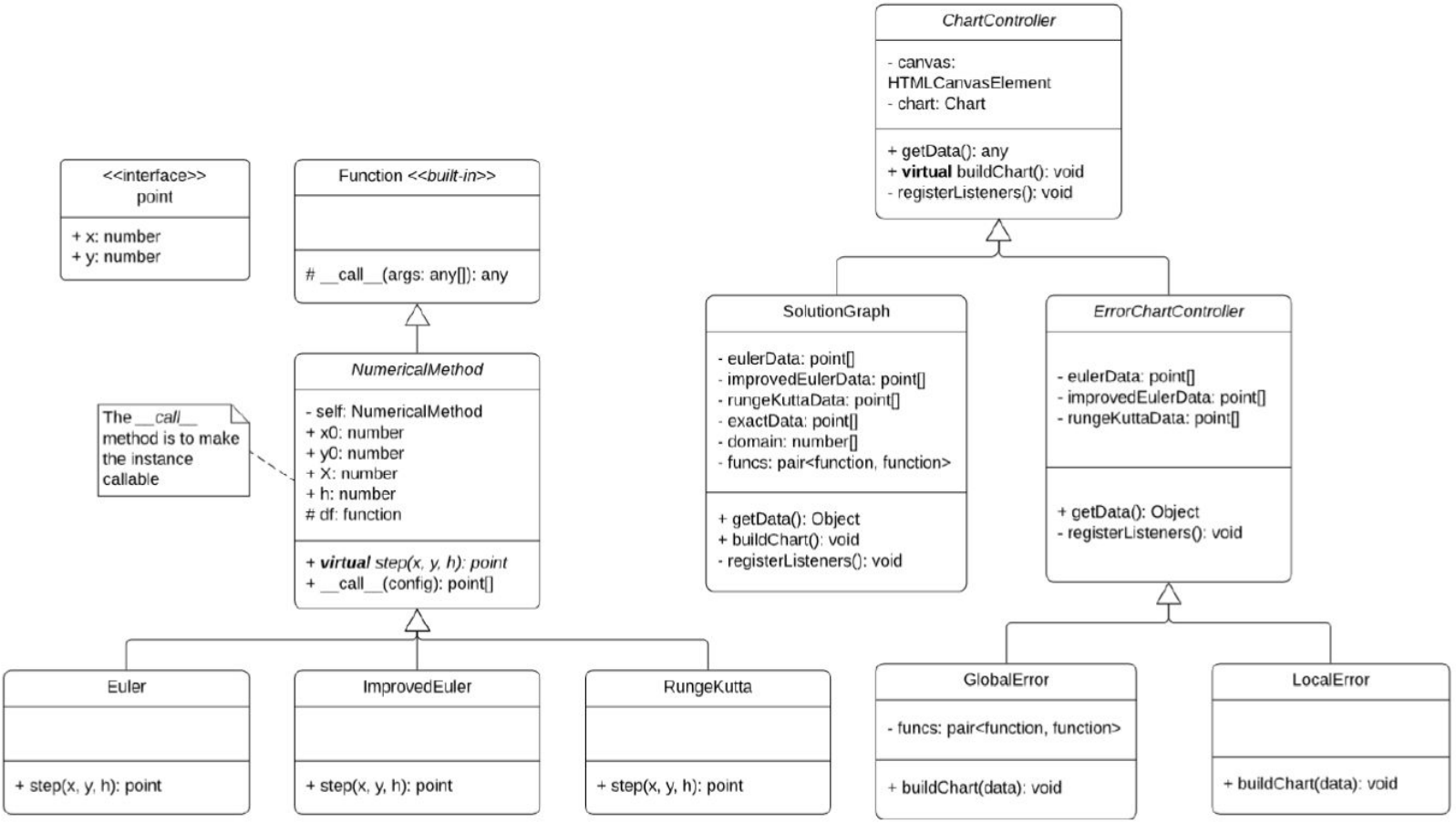


It can be observed that this method approximates the solution much better than the previous two, even for a small number of steps (big step size – in this graph $\hbar$=1).





UML Class Diagram



Code Structure

The code follows the OOP design principles, appropriately applied to Python.

A very interesting part of the code is the implementation of the exact abstract class. It inherits from the built-in class Function, giving its instances the ability to be called as functions. This simplifies extending the code, since one only needs to extend this class, implementing the step method, and can then access the internal structure of the instance like a normal class or call it like a function.

Example:

```
x = np.linspace(x0, x, N)
c = 1.0/(y0 x: tuple ))+x0
y = 1.0/(c-x) + np.exp(x)
```