Differential Equations Practicum Report

Ahmadsho Akdodshoev

October 28, 2021

1 Aim of computation

The aim of this practicum is to compare numerical methods for solving first order differential equations with each other and the exact solution. The numerical methods in question are

- Euler method
- Improved Euler method
- Runge-Kutta method

Four metrics for comparison need to be provided

- Approximate solution of each method
- LTE of each method
- GTE of each method
- Maximum GTE on interval of each method

Additionally, the behavior at points of discontinuity will be examined.

2 Exact Solution

Given a first order differential equation

$$y' = \frac{1}{x} + \frac{2y}{xlnx}$$

with initial values $x_0 = 2$, $y_0 = 0$. First, rewrite it in standard form

$$y' - \frac{2y}{xlnx} = \frac{1}{x}$$

This is a nonhomogeneous first order differential equation and it can be solved using the method of integrating factors.

A function $\mu(x)$ is said to be an integrating factor of a linear differential equation given as

$$y' + p(x)y = f(x)$$

if after a multiplication by that differential equation

$$\mu(x)y' + \mu(x)p(x)y = \mu(x)f(x)$$

the equality

$$\frac{d(\mu(x)y)}{dx} = \mu(x)p(x)$$

is true. If this equality holds, then

$$\mu'(x) = \mu(x)p(x)$$

Multiplying the equation by an integrating factor $\mu(x)$

$$\mu(x)y' - \mu(x)\frac{2y}{xlnx} = \mu(x)\frac{1}{x}$$

Following from the definition of an integrating factor, the following equality is true

$$\mu'(x) = -\mu(x)\frac{2}{xlnx}$$

$$\int \frac{d\mu}{\mu} = -\int \frac{2dx}{xlnx}$$

$$ln(\mu) = -\int \frac{2d(ln|x|)}{xlnx}$$

$$ln(\mu) = -2ln|ln|x||$$

$$\mu = e^{-2ln|ln|x||}$$

$$\mu = ln^{-2}|x|$$

Once again, following from the definition of integrating factors

$$\frac{d(\mu(x)y)}{dx} = \mu(x)\frac{1}{x}$$

Rewriting this with the found value of μ

$$\frac{d(\frac{y}{ln^2|x|})}{dx} = \frac{1}{xln^2|x|}$$

integrating both sides

$$\int d\left(\frac{y}{ln^2|x|}\right) = \int \frac{dx}{xln^2|x|}$$

$$y = ln^2|x| \int \frac{d(ln|x|)}{ln^2|x|}$$

$$y = ln^2|x|\left(C - \frac{1}{ln|x|}\right)$$

$$y = Cln^2|x| - ln|x|$$

The exact solution of this linear first order differential equation is $y = Cln^2|x| - ln|x|$.

2.1 Initial value problem

The initial values of this differential equation are given as $x_0 = 2$, $y_0 = 0$. Solving the exact solution in term of these initial values

$$C = \frac{y_0 + ln|x_0|}{ln^2|x_0|}$$

$$C = \frac{1}{ln|2|}$$

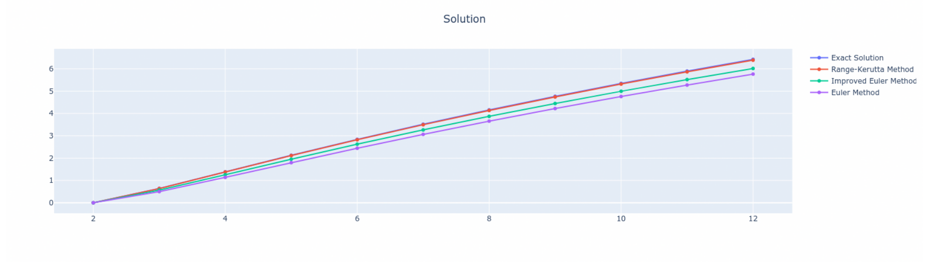
2.2 Points of discontinuity

The exact solution of the given differential equation has its points of discontinuity at $x = 0$. Whereas the differential equation itself has them at points $x \in \{0, 1\}$. Notably, the constant coefficient of the exact solution has discontinuity at point $x = 1$, since $ln^2|1| = 0$.

3 Outcomes of computation

3.1 Outcomes at initial values

The computational practicum required the implementation of Euler method, improved Euler method and Runge-Kutta method of solving linear first order differential equations. The outcomes of computation align with the supposition that the Runge-Kutta method is the most optimal method among the three. In the following charts, the number of steps was set to $N = 10$ and the initial values are $x_0 = 2$, $X = 12$ and $y_0 = 0$.



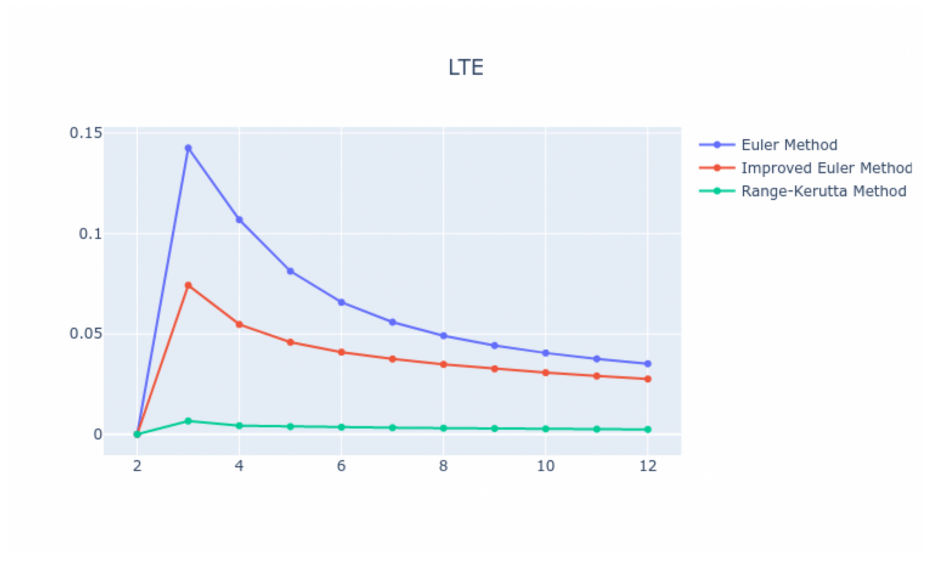
3.1.1 LTE

The LTE of Runge-Kutta method is so much lower than those of Euler method and improved Euler method because it is of order $O(h^4)$, whereas im. Euler method is of order $O(h^2)$ and Euler method is $O(h)$, where h is a length of a step. Here LTE is defined as

$$LTE_i = f(x_i) - f(x_{i-1}) - l(x_i)$$

$f(x_i)$ is the exact solution on x_i and $l(x_i)$ is the iterative value a numerical method that needs to be added to the outcome of computation f_{x_i} done on the previous step. Such that

$$f_{x_i} = f_{x_{i-1}} + l(x_i)$$



3.1.2 GTE

GTE is defined as

$$GTE_i = f(x_i) - f_{x_i}$$

Which can be rewritten following from the supposition that LTE of each of the given methods is bound by some value

$$GTE_i = f(x_i) - (f_{x_{i-1}} + l(x_i))$$

$$GTE_i = f(x_i) - (f_{x_{i-1}} + O(h^k))$$

Since the LTE of Runge-Kutta is order of magnitudes more precise than the other methods, its GTE results reflect that. GTE on step i can be said to be

$$GTE_i = \sum_{i=0}^{N} LTE_i$$

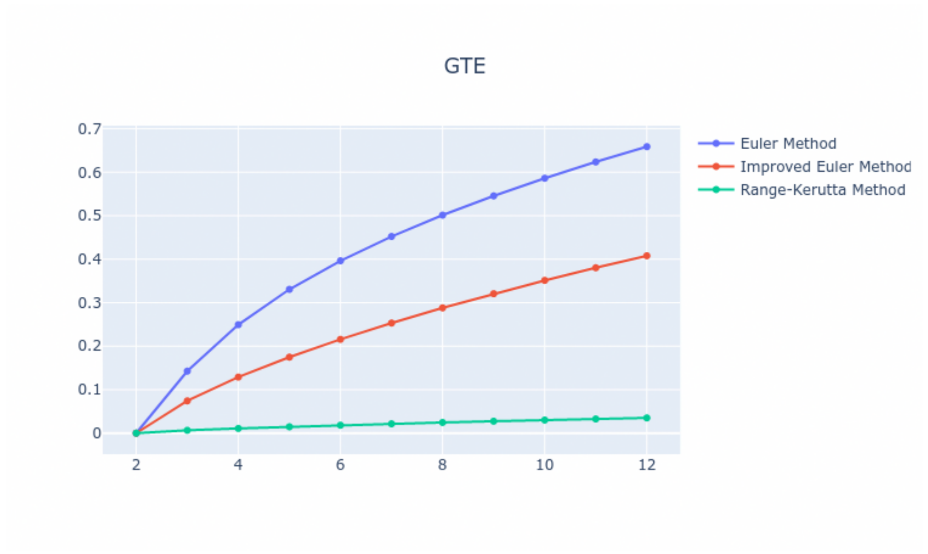
Supposing that such function $j(x)$ exists that LTE of all three methods on every step is h^k, where k is the order of approximation precision of a method and h is the step length. Then

$$GTE_i < \sum_{i=0}^{N} h^k = \frac{N(N+1)}{2} h^k$$

This shows that GTE retain the order of approximation of LTE. To show an example from data, at x_{10} the GTE of Euler method is $g_{eu} = 0.66$, the GTE of improved Euler is $g_{ieu} = 0.41$, and the GTE of Runge-Kutta is $g_{rg} = 0.035$.

$$g_{eu}^2 = 0.66^2 = 0.4356 > 0.41 \rightarrow g_{eu}^2 > g_{ieu}^2 \; [O(h^2)]$$

$$g_{ieu}^2 = 0.41^2 = 0.1681 > 0.035 \rightarrow g_{ieu}^2 > g_{rg}^2 \; [O(h^4)]$$

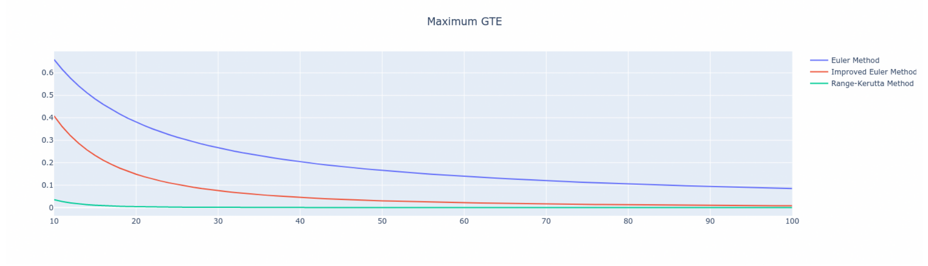


3.1.3 Maximum GTE

Unlike the previous section on GTE, this is about *maximum GTE* on an interval from $N_0 = 10$ to $N = 100$, where N is a number of steps. The max value of *maximum GTE* on step n is

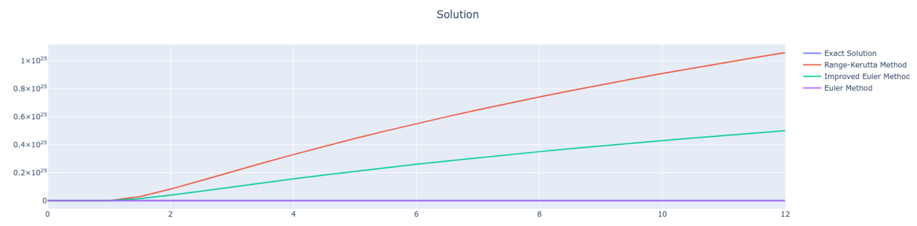
$$MAX\_GTE_n < \frac{n(n+1)}{2} h^k$$

To show an example from data, at $n = 15$ and $h = 2/3$ the GTE of Euler method is $g_{eu} = 0.48$, the GTE of improved Euler is $g_{ieu} = 0.23$, and the GTE of Runge-Kutta is $g_{rg} = 0.01$.

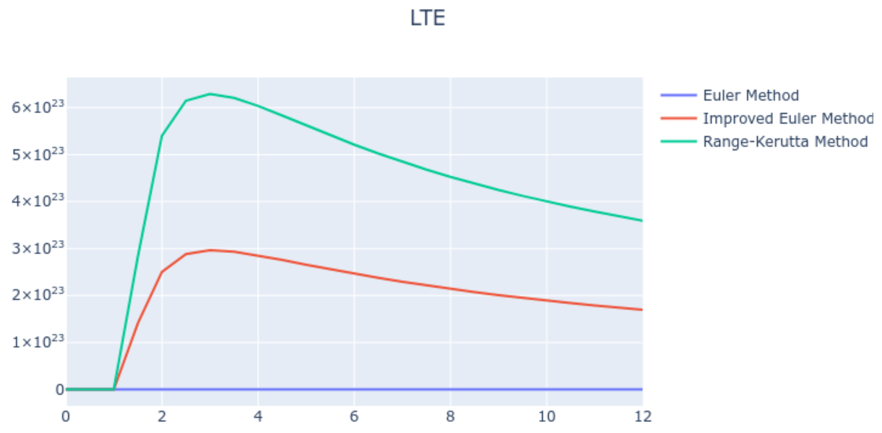


3.2 Outcomes at points of discontinuity

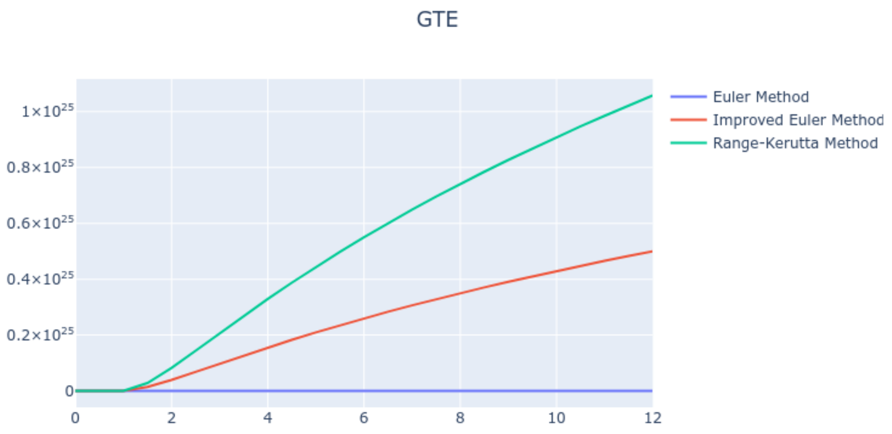
Given initial values of $x_0 = 0$, $X = 12$, $y_0 = 7$ and $N = 24$ the differential equation meets its critical points and due to that the results of numerical methods give unpredictable results. The approximate solution shows that the Runge-Kutta method returns results farthest from exact solution. Notably the split happens not in the neighborhood of point $x \to 0$, but at the neighborhood of point $x \to 1$. Despite the fact that Runge-Kutta method is so far away from other methods, Euler method, which is the closest to the exact solution, is still orders of magnitude away from the exact solution. This shows that these three methods are not applicable on inputs containing a point of discontinuity.



3.2.1 LTE

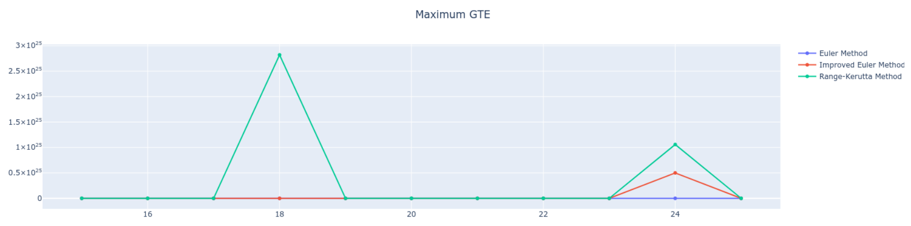


3.2.2 GTE

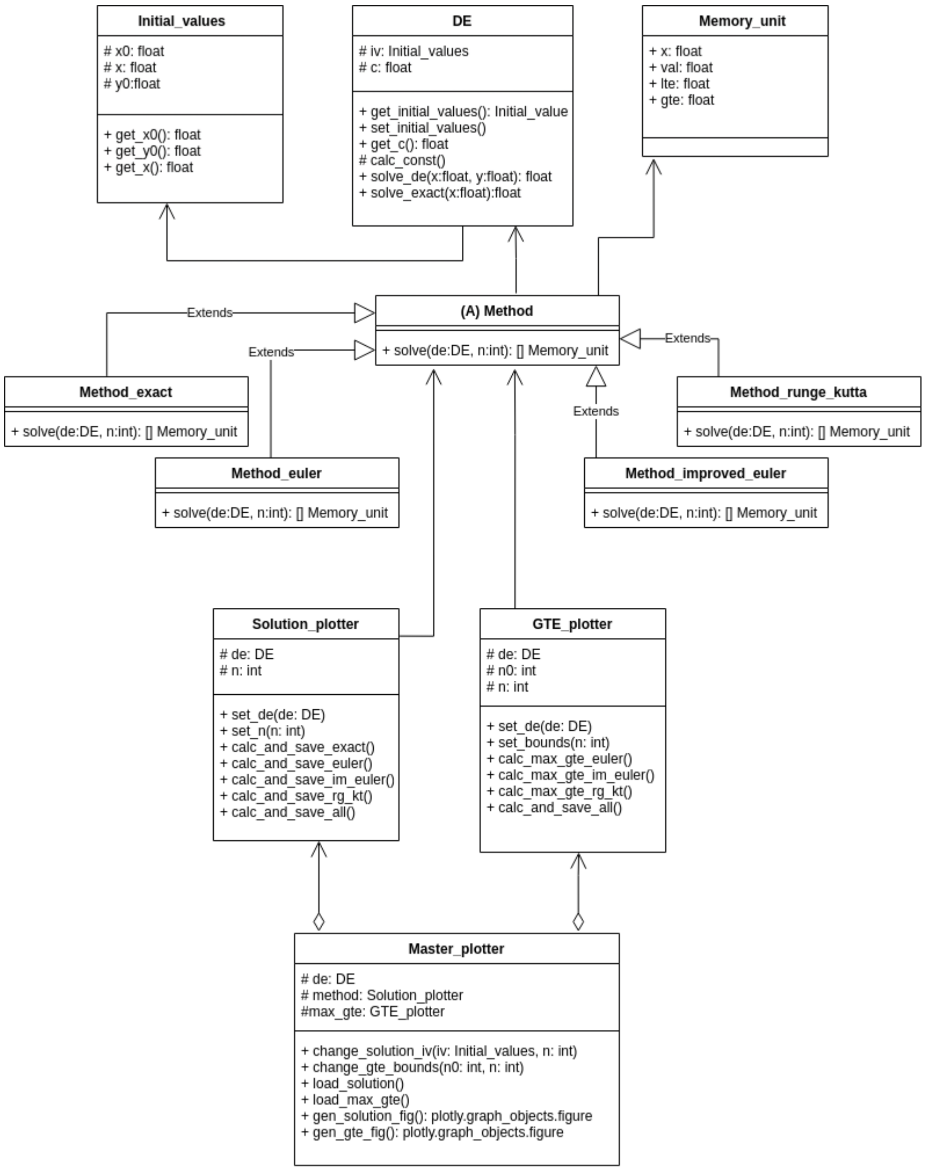


3.2.3 Maximum GTE

Maximum GTE with number of steps from $N_0 = 15$ to $N = 25$ shows that the methods give unpredictable results even based on the number of steps. For some reason the maximum global truncation error of Runge-Kutta method saw a spike on $n = 16$. The maximum GTEs range in the order of 10^{26} for Runge-Kutta, 10^{25} for improved Euler and 10^{17} for Euler methods.



4 UML


Initial_values
# x0: float
# x: float
# y0:float
+ get_x0(): float
+ get_y0(): float
+ get_x(): float
DE
# iv: Initial_values
# c: float
+ get_initial_values(): Initial_value
+ set_initial_values()
+ get_c(): float
# calc_const()
+ solve_de(x:float, y:float): float
+ solve_exact(x:float):float
Memory_unit
+ x: float
+ val: float
+ lte: float
+ gte: float
(A) Method
+ solve(de:DE, n:int): [] Memory_unit
Extends
Extends
Extends
Extends
Method_exact
+ solve(de:DE, n:int): [] Memory_unit
Method_euler
+ solve(de:DE, n:int): [] Memory_unit
Method_runge_kutta
+ solve(de:DE, n:int): [] Memory_unit
Method_improved_euler
+ solve(de:DE, n:int): [] Memory_unit
Solution_plotter
# de: DE
# n: int
+ set_de(de: DE)
+ set_n(n: int)
+ calc_and_save_exact()
+ calc_and_save_euler()
+ calc_and_save_im_euler()
+ calc_and_save_rg_kt()
+ calc_and_save_all()
GTE_plotter
# de: DE
# n0: int
# n: int
+ set_de(de: DE)
+ set_bounds(n: int)
+ calc_max_gte_euler()
+ calc_max_gte_im_euler()
+ calc_max_gte_rg_kt()
+ calc_and_save_all()
Master_plotter
# de: DE
# method: Solution_plotter
#max_gte: GTE_plotter
+ change_solution_iv(iv: Initial_values, n: int)
+ change_gte_bounds(n0: int, n: int)
+ load_solution()
+ load_max_gte()
+ gen_solution_fig(): plotly.graph_objects.figure
+ gen_gte_fig(): plotly.graph_objects.figure

5 Code

5.1 DE class

The differential equation class holds the initial values and the constant factor of the exact solution. Every time the initial values are set, a method *calc_const()* is called to re-evaluate the constant factor.

```
class DE:
...
    def set_initial_value(self, iv):
        self._iv = iv
        self._calc_constant()

    def _calc_constant(self):
        x = self._iv.get_x0()
        y = self._iv.get_y0()
        if x==0:
            x = APR_ZERO
        elif x==1:
            x = 1+APR_ZERO
        self._c = (y+math.log(x))/(math.log(x)**2)
```

5.2 Method classes

Method is an abstract class that has only one method, which is *solve()*. Theoretically numerous other methods can be added to this program with little change in code structure. Classes inheriting from the abstract class *Method* have the following structure

```
class Method_runge_kutta(Method):
    def solve(self, de: DE, n):
        mem = []

        iv = de.get_initial_value()
        x = iv.get_x0()
        y0 = iv.get_y0()
        xf = iv.get_X()

        prev_val = y0
        step_len = (xf-x)/n
        mem.append(Memory_unit(x, y0, 0, 0))
        for i in range (1, n+1):
            temp1 = de.solve_de(x, prev_val)
            temp2 = de.solve_de(x+step_len/2, prev_val+step_len/2*temp1)
            temp3 = de.solve_de(x+step_len/2, prev_val+step_len/2*temp2)
            temp4 = de.solve_de(x+step_len, prev_val+step_len*temp3)
            cur_step = step_len/6*(temp1+2*temp2+2*temp3+temp4)

            val = prev_val + cur_step

            lte = de.solve_exact(x+step_len) - de.solve_exact(x) - cur_step
            lte = abs(lte)

            gte = de.solve_exact(x+step_len) - val
            gte = abs(gte)

            x = x+step_len
            mem.append(Memory_unit(x, val, lte, gte))
            prev_val = val

        return mem
```

5.2.1 Memory unit class

Method classes receive a differential equation and a number of steps, and return a list of *Memory_units*. *Memory_units* is a structure for holding together x_i, f_{x_i}, lte_i and gte_i.

5.3 Solution_plotter class

The *Solution plotter* class is a class that start all methods on separate threads and saves the outputs each method in separate .csv files.

```
def calc_and_save_all(self):
        t_ex = threading.Thread(target=self.calc_and_save_exact)
        t_eu = threading.Thread(target=self.calc_and_save_euler)
        t_ieu = threading.Thread(target=self.calc_and_save_im_euler)
        t_rk = threading.Thread(target=self.calc_and_save_rg_kt)

        t_ex.start()
        t_eu.start()
        t_ieu.start()
        t_rk.start()

        t_ex.join()
        t_eu.join()
        t_ieu.join()
        t_rk.join()
```

5.4 GTE_plotter class

GTE plotter is similar in structure to *Solution plotter* class, but it is much more computationally intense. For each numerical method it starts a thread that calculates the approximate solution of differential equation in n steps for every n on the interval from N_0 to N. On each iteration of n, it saves the value of maximum GTE for every method to separate .csv files. The initiation of thread is similar to aforementioned class, but the computation inside them is different

```
def calc_max_gte_im_euler(self):
        meth = Method_improved_euler()
        f = open("data/gte_im_euler.csv", "w")
        f.write("x,max GTE\n")
        for i in range(self._n0, self._n+1):
            mem = meth.solve(self._de, i)
            max_gte=0
            for j in mem:
                max_gte = max(max_gte, j.gte)
            f.write("%f,%f\n"%(i, max_gte))
```

5.5 Master_plotter class

This class aggregates the two previously mentioned classes. It manages reading the .csv data files, updating them and returning their values in a format acceptable by the GUI API.

6 Conclusion of the computational practicum

This computational practicum concludes that Runge-Kutta methods is orders of magnitude more precise than Euler and improved Euler methods for solving first order differential equations. Additionally, numerical methods should be used cautiously on domains that include a point of discontinuity. At those points none of these three methods could be used for solving the given differential equation.