Analytical solution

1.1 First order differential equation:

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

1.2 General Solution

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

using variable separation:

$$\frac{dy}{dx} = \frac{y(3+2x)}{x^2}$$

$$\frac{dy}{y} = \frac{3+2x}{x^2}dx$$

$$\int \frac{1}{y}dy = \int \frac{3+2x}{x^2}dx$$

integrating both sides leads to

$$ln|y| + C_1 = 2ln|x| - \frac{3}{x} + C_2$$

$$ln|y| = 2ln|x| - \frac{3}{x} + C$$

general solution$\rightarrow y = x^2 e^{\frac{-3}{x}+C}$

1.3 Initial Value Problem (IVP):

$$y(1) = 1$$

substituting:

$$1 = 1^2 e^{\frac{-3}{1}+C}$$

$$C = 3$$

we can obtain the exact solution:

$$y = x^2 e^{\frac{-3}{x}+3}$$

and general solution for C:

$$C = ln\left|\frac{y_0}{x_0^2}\right| + \frac{3}{x_0}$$

C is not continuous at $x_0 = 0$

1.4 Discontiunuty points:

The general solution is not continuous at $x = 0$

Application

I implemented the application(Plot The Graph) using Pygame framework. In fact, Pygame is different from other frameworks. It provides a window. We can manipulate the color of each pixel in the window. Pygame doesn't provide so many features as other frameworks. For example in other frameworks with just a couple of lines of codes, you can easily display an interactive button or input field or some other stuff. But

here it doesn't work. Therefore, I implemented my own **Label, InputField, Button, ButtonWithLabel, CheckBox, Bar, MovalbeBar, Grid, MovableGrid** classes. I think most of these classes are clear by their names. Let's look at the class **Label**. I implemented this class to display some text on the screen. Or class **Grid** to accept an array of points and plot the lines between each point and display. I also implemented the class **NumericMethod**. This class has "**compute_points()**" and "**next_f()**" function. They don't do anything. You Should override these functions with some other functions to compute the points. I override "**compute_points()**" with the following function:

```
def compute_points(self):
    self._points = []
    if self.grid.final_x<self.iv.x or self.grid.xn <= self.grid.x0 or self.n <= 1:
        return
    h = (self.grid.xn - self.grid.x0) / (self.n - 1)
    point = Vector2(self.iv.x, self.iv.y)
    num_points = int((self.grid.final_x - self.iv.x) / h)+2
    for i in range(num_points):
        if point.x >= self.grid.start_x or point.x < self.grid.start_x <= point.x +h:
            self._points.append(point)
        point = self.next_f(point, h)
        if point is None:
            break
```

and "**next_f()**" with the following functions:

Function for Euler method. It accepts a point and an integer "h". It computes and returns the next point:

```
def euler_f(point, h):
    f_val = y_prime(point.x, point.y)
    if f_val is None:
        return None
    return Vector2(point.x + h, point.y + h * f_val)
```

The Function for Improved Euler method:

```
def imp_f(point, h):
    k_1_i = y_prime(point.x, point.y)
    if k_1_i is None:
```

```
        return None
    k_2_i = y_prime(point.x + h, point.y + h * k_1_i)
    if k_2_i is None:
        return None
    return Vector2(point.x + h,point.y + (h / 2) * (k_1_i + k_2_i))
```

The Function for Runge Kutta method:

```
def rk_f(point, h):
    k_1_i = y_prime(point.x, point.y)
    if k_1_i is None:
        return None
    k_2_i = y_prime(point.x + h / 2, point.y + h / 2 * k_1_i)
    if k_2_i is None:
        return None
    k_3_i = y_prime(point.x + h / 2, point.y + h / 2 * k_2_i)
    if k_3_i is None:
        return None
    k_4_i = y_prime(point.x + h, point.y + h * k_3_i)
    if k_4_i is None:
        return None
    return Vector2(point.x + h, point.y + (h / 6) * (k_1_i + 2 * k_2_i +
2 * k_3_i + k_4_i))
```

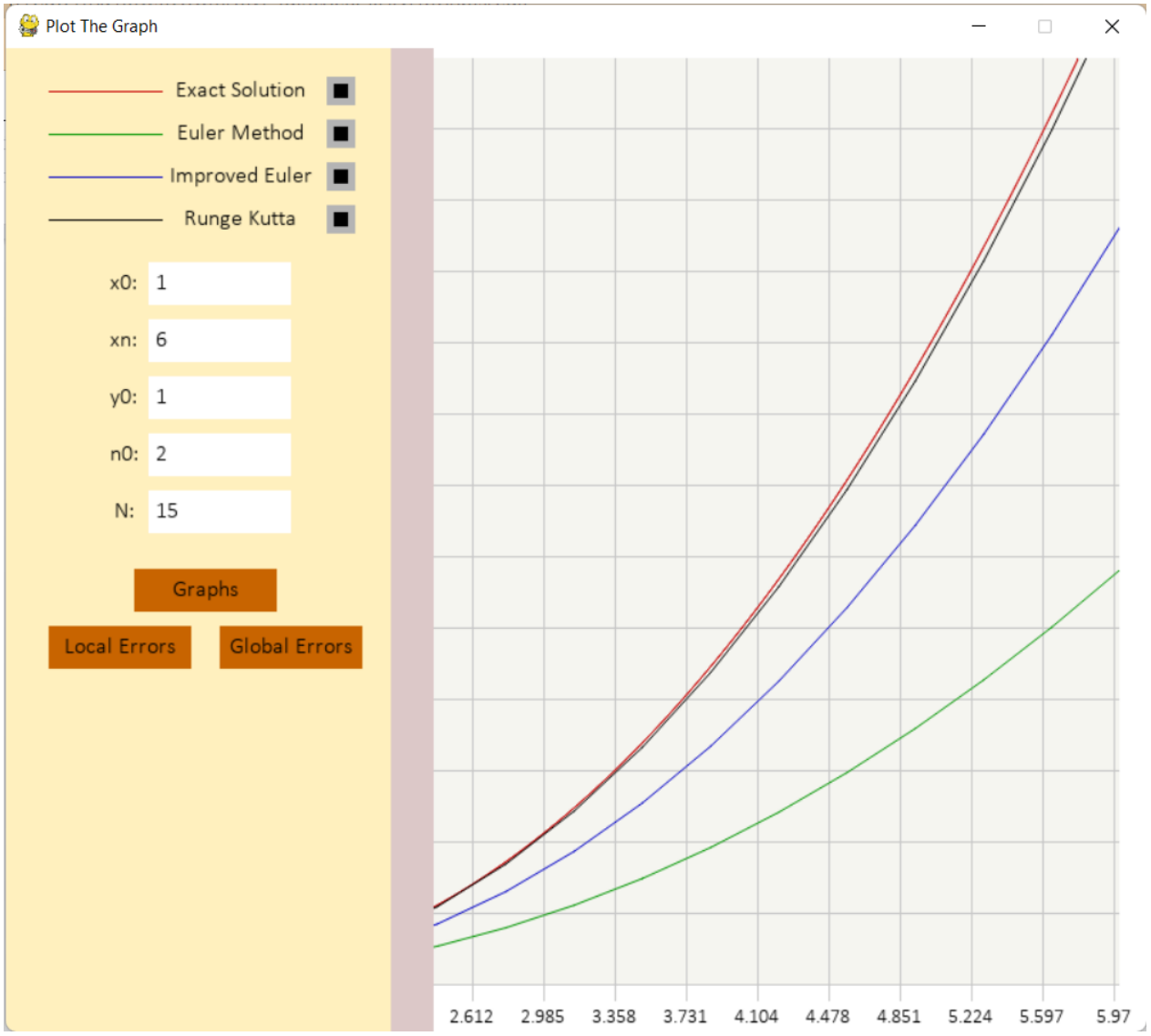
In order to start work with the application, you should have installed Python and Pygame on your machine.

To run the application you just need to run ***main.py***

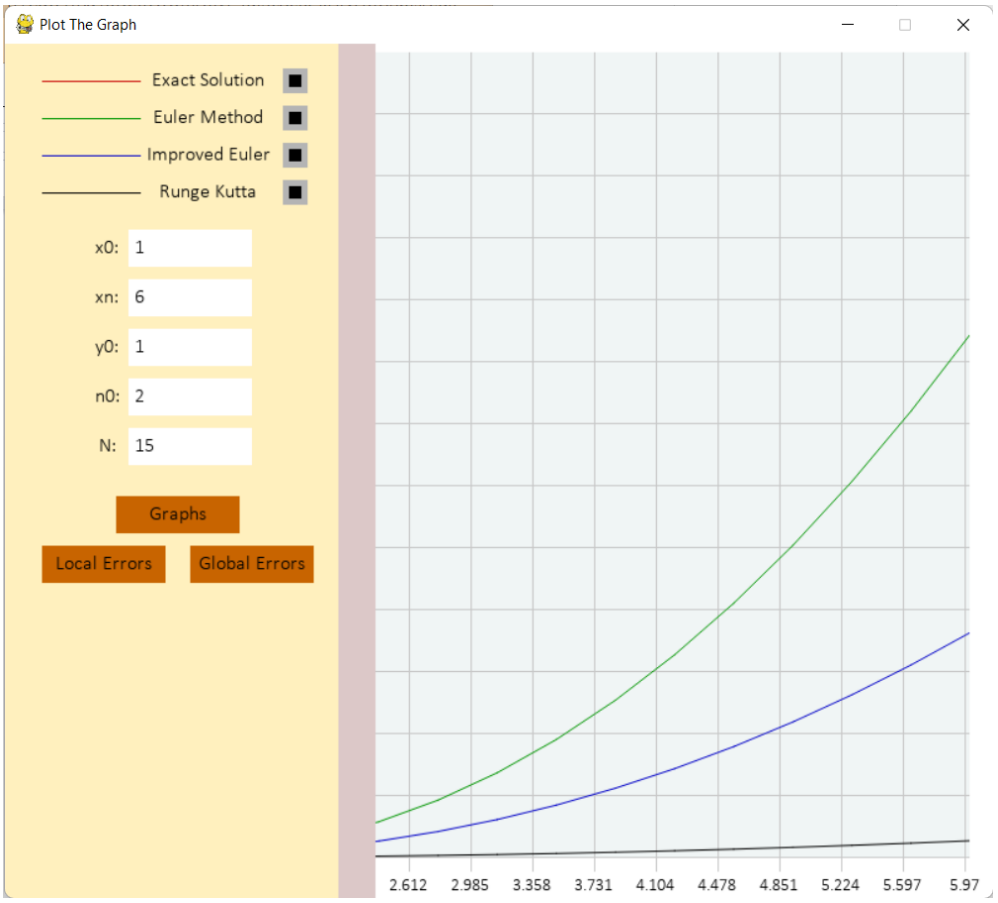
When you have run the application successfully, you will see a bar on the left-hand side. There is a long **pink vertical button** on the right-hand side of the bar. The button is responsible for opening and closing the bar. You can **click** on a particular place of a grid and **drag** to change the intervals or you can enter intervals manually. For global truncation error dragging the grid is disabled. Also, you can zoom the grid. By pressing the button "**x**" or "**y**" and rotating the **mouse wheel** you zoom the **x**-axis or **y**-axis respectively.

Some pictures from the applications:

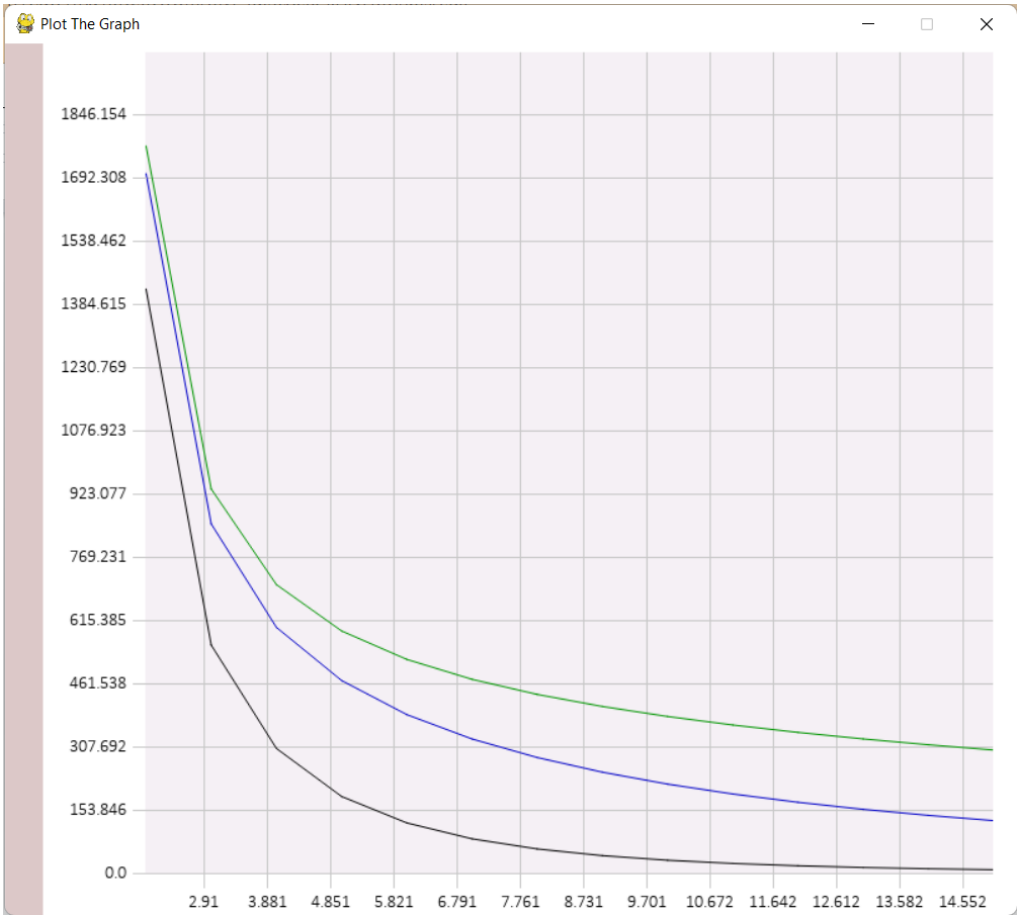
The Graph of methods:



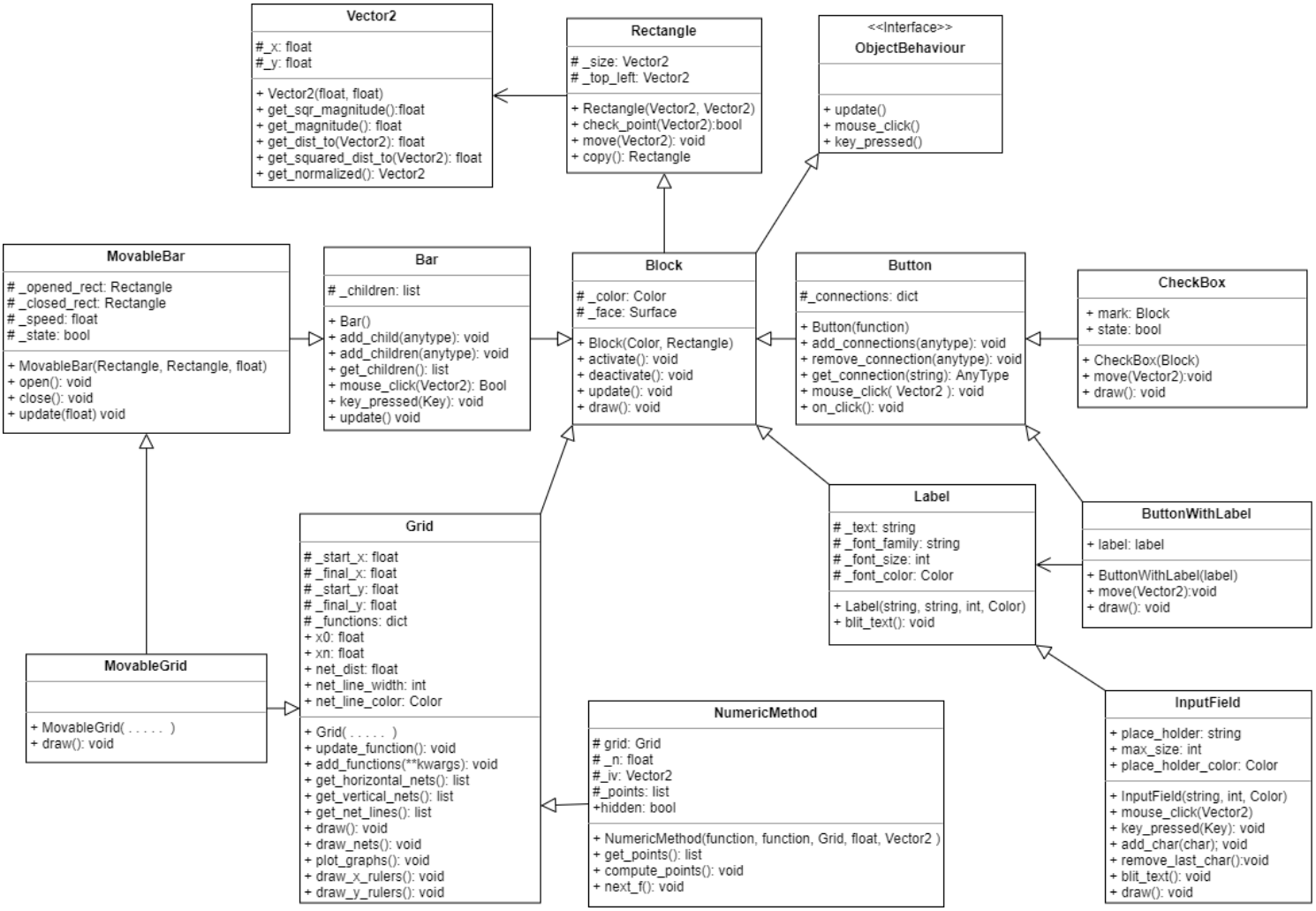
The Graph of Local Errors:



The Graph of Global Errors:



UML diagram


Vector2
#_x: float
#_y: float
+ Vector2(float, float)
+ get_sqr_magnitude():float
+ get_magnitude(): float
+ get_dist_to(Vector2): float
+ get_squared_dist_to(Vector2): float
+ get_normalized(): Vector2
Rectangle
_size: Vector2
_top_left: Vector2
+ Rectangle(Vector2, Vector2)
+ check_point(Vector2):bool
+ move(Vector2): void
+ copy(): Rectangle
<<Interface>>
ObjectBehaviour
+ update()
+ mouse_click()
+ key_pressed()
MovableBar
_opened_rect: Rectangle
_closed_rect: Rectangle
_speed: float
_state: bool
+ MovableBar(Rectangle, Rectangle, float)
+ open(): void
+ close(): void
+ update(float) void
Bar
_children: list
+ Bar()
+ add_child(anytype): void
+ add_children(anytype): void
+ get_children(): list
+ mouse_click(Vector2): Bool
+ key_pressed(Key): void
+ update() void
Block
_color: Color
_face: Surface
+ Block(Color, Rectangle)
+ activate(): void
+ deactivate(): void
+ update(): void
+ draw(): void
Button
#_connections: dict
+ Button(function)
+ add_connections(anytype): void
+ remove_connection(anytype): void
+ get_connection(string): AnyType
+ mouse_click(Vector2): void
+ on_click(): void
CheckBox
+ mark: Block
+ state: bool
+ CheckBox(Block)
+ move(Vector2):void
+ draw(): void
Label
_text: string
_font_family: string
_font_size: int
_font_color: Color
+ Label(string, string, int, Color)
+ blit_text(): void
ButtonWithLabel
+ label: label
+ ButtonWithLabel(label)
+ move(Vector2):void
+ draw(): void
Grid
_start_x: float
_final_x: float
_start_y: float
_final_y: float
_functions: dict
+ x0: float
+ xn: float
+ net_dist: float
+ net_line_width: int
+ net_line_color: Color
+ Grid(. . . . .)
+ update_function(): void
+ add_functions(**kwargs): void
+ get_horizontal_nets(): list
+ get_vertical_nets(): list
+ get_net_lines(): list
+ draw(): void
+ draw_nets(): void
+ plot_graphs(): void
+ draw_x_rulers(): void
+ draw_y_rulers(): void
MovableGrid
+ MovableGrid(. . . . .)
+ draw(): void
NumericMethod
grid: Grid
_n: float
#_iv: Vector2
#_points: list
+hidden: bool
+ NumericMethod(function, function, Grid, float, Vector2)
+ get_points(): list
+ compute_points(): void
+ next_f(): void
InputField
+ place_holder: string
+ max_size: int
+ place_holder_color: Color
+ InputField(string, int, Color)
+ mouse_click(Vector2)
+ key_pressed(Key): void
+ add_char(char); void
+ remove_last_char():void
+ blit_text(): void
+ draw(): void