I pledge that the work submitted for this coursework, both the report and the MATLAB code, is my own unassisted work unless stated otherwise.

CID........................ 01730921

Are you a Year 4 student?.. yes

Coursework 1

Fill in your CID and include the problem sheet in the coursework. Before you start working on the coursework, read the coursework guidelines. Any marks received for this coursework are only indicative and may be subject to moderation and scaling. *The mastery component is marked with a star.*

Exercise 1 (The Euler method for scalar ODEs) % of course mark: /3.0

a) Apply the Euler method to the initial value problem

$$x' = \frac{1}{8}(5 - x - 5025e^{-8t}), \quad x(0) = 100, \quad t \in [0, 10]. \tag{1}$$

b) Compute the numerical solution of (1) with the time steps $h = \{0.1, 0.05, 0.025\}$.

c) Compute the global error $|e_N|$ at $t = 10$ for $h = \{0.1, 0.05, 0.025\}$.

Exercise 2 (Taylor series methods for systems of ODEs) % of course mark: /3.0

a) Apply the TS(3) method to the initial value problem

$$x'' + 3x' + 2x = t^2, \; x(0) = 1, \; x'(0) = 0, \; t \in [0, 3]. \tag{2}$$

b) Compute the numerical solution of (2) with the time steps $h = \{0.1, 0.05, 0.025\}$.

c) Find an approximated number of time steps such that the global error $|e_N| < 10^{-3}$ at $t = 3$.

Exercise 3 (Linear multistep methods for systems of ODEs) % of course mark: /4.0
This exercise is for Year-3 students only!

a) Apply the 2-step Adams-Bashforth to the initial value problem (Chua's circuit)

$$\begin{cases} x' = \alpha(y - ax^3 - cx), \\ y' = x - y + z, \\ z' = -\beta y - \gamma z, \end{cases} \tag{3}$$

$$x(0) = 1, \; y(0) = z(0) = 0, \; t = [0, 100],$$
$$\alpha = 10, \; \beta = 15, \; \gamma = 0.01, \; a = 0.1, \; c = -0.2.$$

b) Compute the numerical solution of (3) with the time step $h = 0.01$.

c) Explain how you start the method.

Exercise 4 (Linear multistep methods for systems of ODEs) % of course mark: /6.5*

a) Derive the 3-step Adams-Bashforth method using Lagrange polynomials.

b) Calculate the order of the local truncation error of the method.

c) Apply the method to the Lorenz system

$$\begin{cases} x' = \sigma(y-x), \\ y' = x(\rho - z) - y, \\ z' = xy - \beta z, \\ \sigma = 10,\ \beta = 8/3,\ \rho = 28,\ x(t_0) = y(t_0) = z(t_0) = 1,\ t = [0, 100]. \end{cases} \tag{4}$$

d) Compute the numerical solution of (4) with the time step $h = 0.01$.

e) Explain how you start the method.

Coursework mark: **% of course mark**

Coursework Guidelines

Below is a set of guidelines to help you understand what coursework is and how to improve it.

Coursework

- The coursework requires more than just following what has been done in the lectures, some amount of individual work is expected.
- The coursework report should describe in a concise, clear, and coherent way of what you did, how you did it, and what results you have.
- The report should be understandable to the reader with the mathematical background, but unfamiliar with your current work.
- Do not bloat the report by paraphrasing or presenting the results in different forms.
- Use high-quality and carefully constructed figures with captions and annotated axis, put figures where they belong.
- All numerical solutions should be presented as graphs.
- Use tables only if they are more explanatory than figures. The maximum table length is a half page.
- All figures and tables should be embedded in the report. The report should contain all discussions and explanations of the methods and algorithms, and interpretations of your results and further conclusions.
- The report should be typeset in LaTeX or Word Editor and submitted as a single pdf-file.
- The maximum length of the report is ten A4-pages (additional 3 pages is allowed for Year 4 students); the problem sheet is not included in these ten pages.
- Do not include any codes in the report.
- Marks are not based solely on correctness. The results must be described and interpreted. The presentation and discussion is as important as the correctness of the results.

Codes

- You cannot use third party numerical software in the coursework.
- The code you developed should be well-structured and organised, as well as properly commented to allow the reader to understand what the code does and how it works.
- All codes should run out of the box and require no modification to generate the results presented in the report.

Submission

- The coursework submission must be made via Turnitin drop box on your Blackboard page. You must complete and submit the coursework anonymously, **the deadline is 1pm on the date of submission** (unless stated otherwise). The coursework should be submitted as one zip-file containing the pdf-report and MATLAB (m-files only) or Python (py-files only) code, both the report and the code should be in the directory named `CID_Coursework#`. The zip-file should be named `CID_Coursework#.zip` . The executable MATLAB (or Python) scripts for the exercises should be named as follows: exercise1.m, exercise2.m, etc.

Numerical odes - Coursework 1

Francesco Piatti

October 2022

1 Question 1

1.1 Forward Euler method

The Euler method of a first-order ODE written in the form $x'(t) = f(t, x)$, $x(t_0) = x_0$, where $f : \mathbb{R}^2 \to \mathbb{R}$ is defined as:

$$x_{n+1} = x_n + h\, f_n, \quad x_0 = x(t_0),\ h \in \mathbb{R}^+ \tag{1}$$

where $x_n = x(t_n)$, $t_{n+1} = t_n + h$ and $f_n = f(t_n, x(t_n))$.

In the given problem we have $f(t, x) = (5 - x - 5025e^{-8t})/8$, $t_0 = 0$, $t_N = 10$, $x(t_0) = x_0 = 100$. Substituting we get:

$$x_{n+1} = x_n + h\,(5 - x_n - 5025e^{-8t_n})/8, \qquad x_0 = 100,\ n \in \{0, \dots, t_N/h - 1\} \tag{2}$$

1.2 Numerical results

The code, written in object-oriented Python, is designed as follows: default attributes are t_0, t_N, x_0, whereas the core methods (i.e. the functions) are:

1. *exact_solution*: has input t and outputs the numerical value of the exact solution below:

$$x(t) \;=\; 5 + 1675\exp(-8t)/21 + 320\exp(-t/8)/21 \tag{3}$$

   This solution is calculated using first-order linear non-homogeneous ODE formula [1].

2. f: takes as input t and x and returns a float according to the definition of f above

[1] $x(t) = \exp(-\int_0^t 1/8\, ds)\,[\int_0^t \exp(\int_0^u 1/8\, ds)\,(5 - 5025e^{-8u})/8\, du + 100]$

3. *euler_method*: takes as input h and returns an array of the values of $x(t)$, according to (2), for $t = t_0 + hi$, $i = 1, 2, \ldots, \mathrm{int}(t_N/h)$. This is done through a for loop.

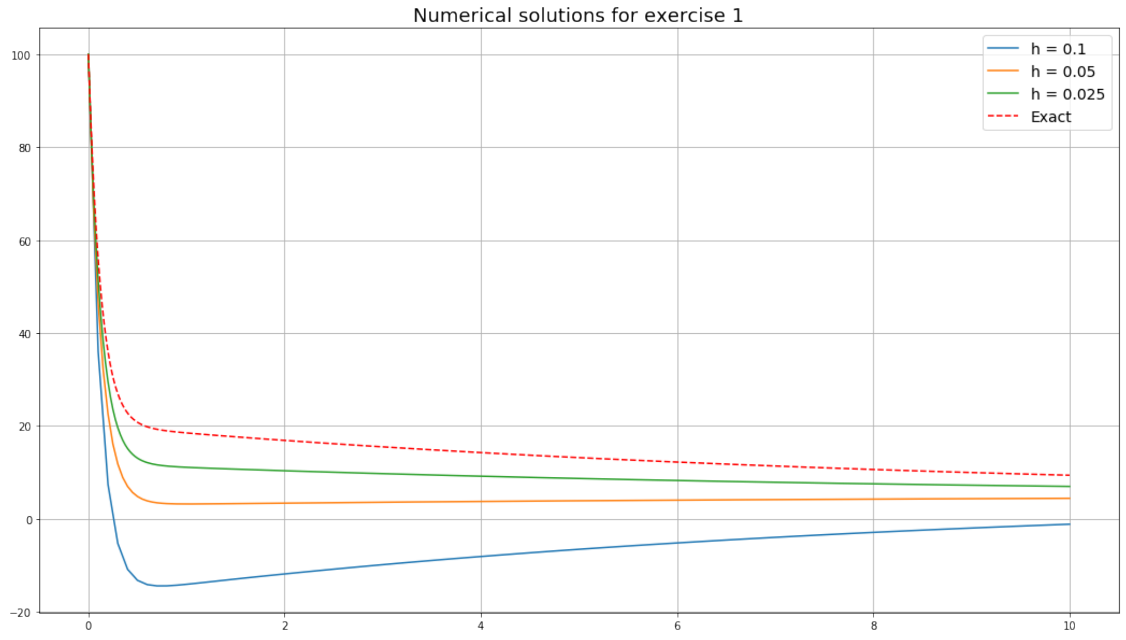
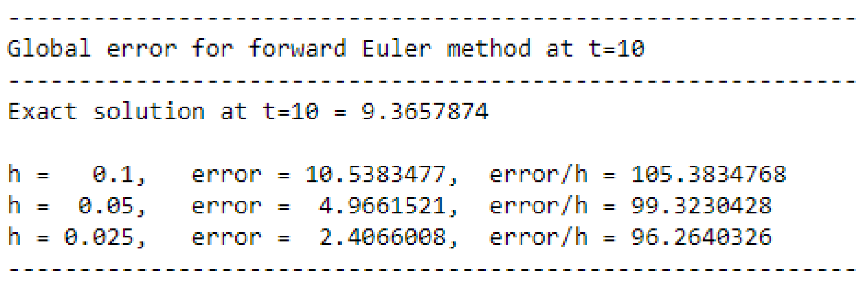


Figure 1: Exact solution is also plotted as a dashed line

We can draw the conclusion that the accuracy of the numerical solution is, as expected, inversely proportional to the magnitude of h. Yet, $h = 0.025$ is still not enough small to provide us with an accurate solution.

1.3 Global error at $t = 10$

Global error at $t = 10$ is the absolute difference of the exact solution at time 10 and the numerical one, i.e. $|e_N| = |x(10) - x_N|$. The results are presented in the table below.



```
------------------------------------------------------------
Global error for forward Euler method at t=10
------------------------------------------------------------
Exact solution at t=10 = 9.3657874

h =   0.1,   error = 10.5383477,  error/h = 105.3834768
h =  0.05,   error =  4.9661521,  error/h = 99.3230428
h = 0.025,   error =  2.4066008,  error/h = 96.2640326
------------------------------------------------------------
```

From the lecture notes, we know that the forward Euler method has convergence speed proportional to h, which is coherent with what we have found as the proportionality $|e_N|/h$ is almost constant.

2 Question 2

2.1 The Taylor-series(3) method

Given a system of first-order ODEs, written in the form $\mathbf{x}' = f(t,\mathbf{x}), \mathbf{x}(t_0) = \mathbf{x}_0$, where $\mathbf{x} \in \mathbb{R}^n$ and $f : \mathbb{R}^{n+1} \to \mathbb{R}^n$. Expanding $\mathbf{x}_{n+1} := \mathbf{x}(t_n + h)$ up to order 3 we get the TS(3) method. I.e.:

$$\mathbf{x}(t_n + h) = \mathbf{x}(t_n) + hf(t_n, \mathbf{x}(t_n)) + \frac{h^2}{2} f'(t_n, \mathbf{x}(t_n)) + \frac{h^3}{6} f''(t_n, \mathbf{x}(t_n)) \tag{4}$$

or more compactly:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + hf_n + \frac{h^2}{2} f'_n + \frac{h^3}{6} f''_n \tag{5}$$

The given IVP is:

$$x'' + 3x' + 2x = t^2, \quad x(0) = 1,\ x'(0) = 0,\ t \in [0,3] \tag{6}$$

We thus write our second-order ODE as a system of first-order ODEs by making the substitution $u = x$ and $v = x'$. Hence

$$\begin{pmatrix} u'(t) \\ v'(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ t^2 - 2u(t) - 3v(t) \end{pmatrix}, \qquad u(0) = 1,\ v(0) = 0 \tag{7}$$

Therefore, letting $u = u(t)$ and $v = v(t)$:

- $f(t,u,v) = (v,\ t^2 - 2u - 3v)$
- $f'(t,u,v) = (v',\ 2t - 2u' - 3v') = (t^2 - 2u - 3v,\ 2t - 3t^2 + 6u + 7v)$
- $f''(t,u,v) = (2t - 2u' - 3v',\ 2 - 6t + 6u' + 7v') = (2t - 3t^2 + 7v + 6u,\ 2 + 7t^2 - 6t - 14u - 15v)$

Using the notation $u_n = u(t_n)$ and similarly $v_n = v(t_n)$ and plugging the above formulas for the derivatives of $f(t,\mathbf{x})$ into (5) we get the numerical solution:

$$\begin{pmatrix} u_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} u_n \\ v_n \end{pmatrix} + h \begin{pmatrix} v_n \\ t_n^2 - 2u_n - 3v_n \end{pmatrix} + \frac{h^2}{2} \begin{pmatrix} t_n^2 - 2u_n - 3v_n \\ 2t_n - 3t_n^2 + 6u_n + 7v_n \end{pmatrix} + \frac{h^3}{6} \begin{pmatrix} 2t_n - 3t_n^2 + 6u_n + 7v_n \\ 2 + 7t_n^2 - 6t_n - 14u_n - 15v_n \end{pmatrix}$$

We then extract our solution $x(t_n)$ easily as in our substitution we had the identity $u = x$.

2.2 Numerical solution

Similarly to question 1, we build a Python class that has as default attributes $x_0 = u_0,\ x_0' = v_0,\ t_0,\ t_N$ and the following core methods:

1. *exact_solution*: has input t and outputs the numerical value of the exact solution below:

$$x(t) \ = \ 5 + 1675 \exp(-8t)/21 + 320 \exp(-t/8)/21 \tag{8}$$

2. f: takes as input (t, u, v) and returns a float according to the definition of f above.

3. f': takes as input (t, u, v) and returns a float according to the definition of f' above.

4. f'': takes as input (t, u, v) and returns a float according to the definition of f'' above.

5. *ts3_method*: takes as input h and returns an array of the values of $x(t)$ for $t = t_0 + hi$, $i = 1, 2, \dots, \text{int}(t_N/h)$, according to (4). This is done through a for loop that computes u and v at each steps iterativly. The output is one dimensional array and corresponds to the values of u as $u = x$.

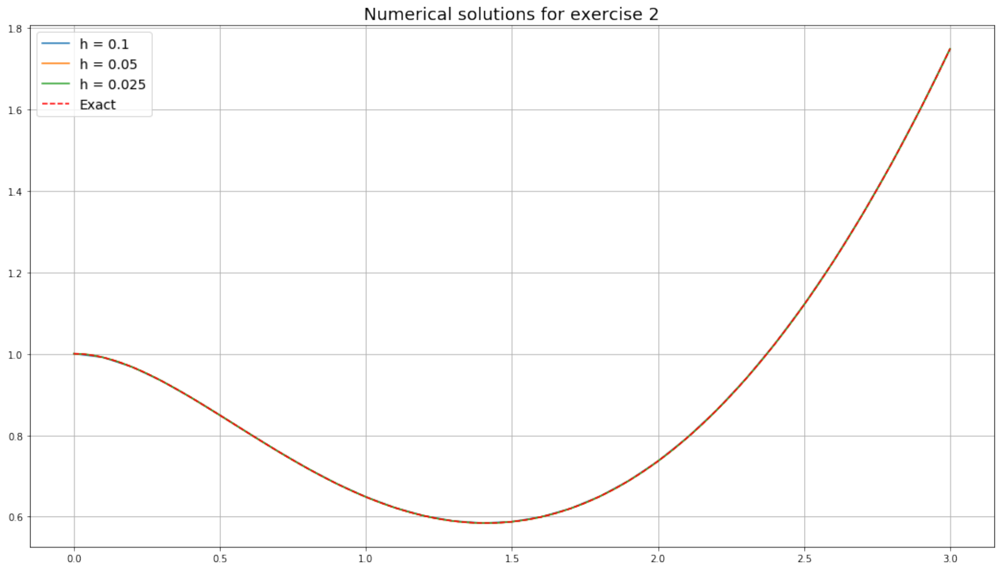


Figure 2: Exact solution is also plotted as a dashed line.

This method applied to the given IVP seems to be quite accurate also for the larger value of h here considered. Yet, if we zoom in, we realise that the 4 graphs are not perfectly aligned (as also shown by the global errors reported in part 3 below).

2.3 Number of steps analysis

In order to determine the minimum number of steps required such than the global error at time 3 is smaller that 10^{-3} we use the relationship between the number of steps and h (i.e. steps= t_N/h) to evaluate the global error for a range of *n_steps*. We use the aforementioned *ts3_method* to compute the global error. As reported in the table and in the graph, the first value of n below the threshold of 10^{-3} is steps=6.

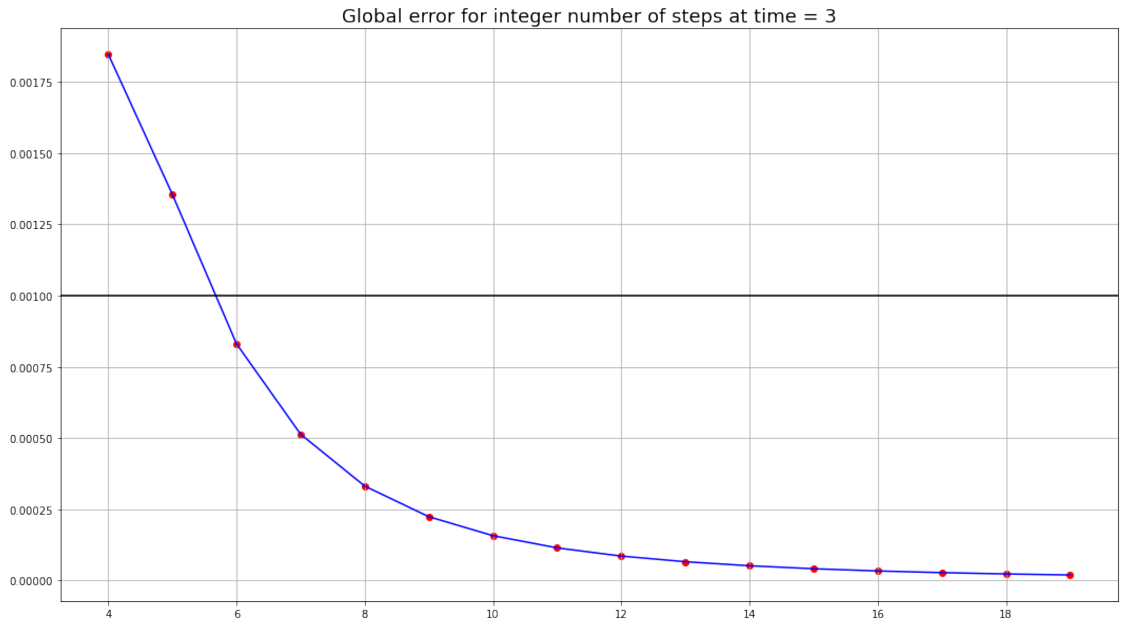
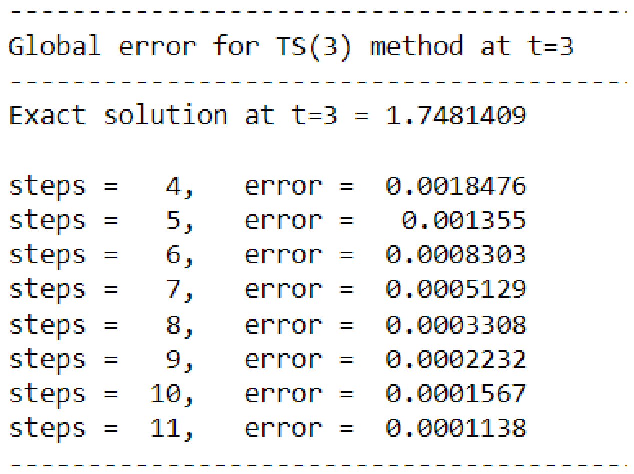


Figure 3: We notice from this graph that decreasing the length of the time step the numerical solution converges to the exact one.



```
---------------------------------------
Global error for TS(3) method at t=3
---------------------------------------
Exact solution at t=3 = 1.7481409

steps =   4,   error =  0.0018476
steps =   5,   error =   0.001355
steps =   6,   error =  0.0008303
steps =   7,   error =  0.0005129
steps =   8,   error =  0.0003308
steps =   9,   error =  0.0002232
steps =  10,   error =  0.0001567
steps =  11,   error =  0.0001138
---------------------------------------
```

3 Question 4

3.1 Derivation of the 3-step Adam-Bashforth method using Lagrange polynomials

Considering the following IVP

$$x' = f(t,x), \quad x(t_0) = x_0,\ t \in [t_0, t_N] \tag{9}$$

The integration of (9) over the interval $[t_n, t_{n+1}]$ gives:

$$\int_{t_n}^{t_{n+1}} x'\,dt \;=\; \int_{t_n}^{t_{n+1}} f(t,x)\,dt \quad \Longrightarrow \quad x(t_{n+1}) \;=\; x(t_n) + \int_{t_n}^{t_{n+1}} f(t,x)\,dt \tag{10}$$

To proceed with the numerical solution, we have to compute the integral of $f(t,x)$. In order to do so we can approximate $f(t,x)$ with Lagrange polynomials and then do the integration. Lagrange polynomials are defined as follows:

$$\mathcal{L}(t) := \sum_{j=0}^{k} f_j \ell_j(t), \qquad \ell_j(t) := \prod_{\substack{0\le m\le k\\ m\ne j}} \frac{t-t_m}{t_j-t_m} \tag{11}$$

In our problem we have $k=2$, i.e. we take the Lagrange interpolation through three points (t_{n-2}, t_{n-1}, t_n)

$$\mathcal{L}(t) := \frac{(t-t_{n-1})(t-t_{n-2})}{(t_n-t_{n-1})(t_n-t_{n-2})} f_n + \frac{(t-t_n)(t-t_{n-2})}{(t_{n-1}-t_n)(t_{n-1}-t_{n-2})} f_{n-1} + \frac{(t-t_n)(t-t_{n-1})}{(t_{n-2}-t_n)(t_{n-2}-t_{n-1})} f_{n-2} \tag{12}$$

As $t_n - t_{n-2} = 2h$ and $t_n - t_{n-1} = h$, we can rewrite the above equation as:

$$\mathcal{L}(t) = \frac{(t-t_{n-1})(t-t_{n-2})}{2h^2} f_n - \frac{(t-t_n)(t-t_{n-2})}{h^2} f_{n-1} + \frac{(t-t_n)(t-t_{n-1})}{2h^2} f_{n-2} \tag{13}$$

$$
\begin{aligned}
\int_{t_n}^{t_{n+1}} f(t,x)\,dt \;&\approx\; \int_{t_n}^{t_{n+1}} \mathcal{L}(t)\,dt \\
&= \int_{t_n}^{t_{n+1}} \left(\frac{(t-t_{n-1})(t-t_{n-2})}{2h^2}\, f_n \;-\; \frac{(t-t_n)(t-t_{n-2})}{h^2}\, f_{n-1} \;+\; \frac{(t-t_n)(t-t_{n-1})}{2h^2}\, f_{n-2} \right) dt \\
&= \Bigg[\frac{2t^3 - 3t^2(t_{n-1}+t_{n-2}) + 6t(t_{n-1}t_{n-2})}{12h^2}\, f_n \;-\; \frac{2t^3 - 3t^2(t_n+t_{n-2}) + 6t(t_n t_{n-2})}{6h^2}\, f_{n-1} \\
&+\; \frac{2t^3 - 3t^2(t_n+t_{n-1}) + 6t(t_n t_{n-1})}{12h^2}\, f_{n-2} \Bigg]_{t_n}^{t_{n+1}} \\
&= \frac{2t_{n+1}^3 - 3t_{n+1}^2(t_{n-1}+t_{n-2}) + 6t_{n+1}(t_{n-1}t_{n-2}) - 2t_n^3 + 3t_n^2(t_{n-1}+t_{n-2}) - 6t_n(t_{n-1}t_{n-2})}{12h^2}\, f_n \\
&- \frac{2t_{n+1}^3 - 3t_{n+1}^2(t_n+t_{n-2}) + 6t_{n+1}(t_n t_{n-2}) - 2t_n^3 + 3t_n^2(t_n+t_{n-2}) - 6t_n(t_n t_{n-2})}{6h^2}\, f_{n-1} \\
&+ \frac{2t_{n+1}^3 - 3t_{n+1}^2(t_n+t_{n-1}) + 6t_{n+1}(t_n t_{n-1}) - 2t_n^3 + 3t_n^2(t_n+t_{n-1}) - 6t_n(t_n t_{n-1})}{12h^2}\, f_{n-2}
\end{aligned}
$$

Rewriting everything in terms of t_n using the fact that $t_{n-2} = t_n - 2h$, $t_{n-1} = t_n - h$ and $t_{n+1} = t_n + h$ we get cancellations that allow us to conclude that the equation above equals:

$$\frac{23h^2}{12h^2} f_n - \frac{8h^2}{6h^2} f_{n-1} + \frac{5h^2}{12h^2} f_{n-2}$$

$$\Longrightarrow \quad \int_{t_n}^{t_{n+1}} f(t,x)\,dt \;\approx\; \frac{h}{12}(23f_n - 16f_{n-1} + 5f_{n-2}) \tag{14}$$

Therefore the three-step Adam-Bashforth numerical scheme is obtained by combining (10) and (14) :

$$x_{n+1} = x_n + \frac{h}{12}(23f_n - 16f_{n-1} + 5f_{n-2}) \tag{15}$$

$$\text{i.e. } x(t_{n+1}) = x(t_n) + \frac{h}{12}(23f(t_n,x_n) - 16f(t_{n-1},x_{n-1}) + 5f(t_{n-2},x_{n-2})) \tag{16}$$

or equivalently

$$x_{n+3} = x_{n+2} + \frac{h}{12}(23f_{n+2} - 16f_{n+1} + 5f_n) \tag{17}$$

$$\text{i.e. } x(t_{n+3}) = x(t_{n+2}) + \frac{h}{12}(23f(t_{n+2},x_{n+2}) - 16f(t_{n+1},x_{n+1}) + 5f(t_n,x_n)) \tag{18}$$

3.2 Local truncation error

To begin with, the Taylor expansion of the exact solution reads:

$$x(t+h) = x(t) + hx'(t) + \frac{h^2}{2}x''(t) + \frac{h^3}{6}x'''(t) + \frac{h^4}{24}x^{(4)}(t) + \mathcal{O}(h^5) \tag{19}$$

Now, considering the numerical scheme (15), recall:

- $f_{n-1} = f(t_{n-1}, x_{n-1}) = x'(t_n - h)$
- $f_{n-2} = f(t_{n-2}, x_{n-2}) = x'(t_n - 2h)$

Their respective Taylor expansions are:

- $x'(t_n - h) = x'(t_n) - hx''(t_n) + \frac{h^2}{2}x'''(t_n) - \frac{h^3}{6}x^{(4)}(t_n) + \mathcal{O}(h^5)$
- $x'(t_n - 2h) = x'(t_n) - 2hx''(t_n) + 2h^2x'''(t_n) - \frac{8h^3}{6}x^{(4)}(t_n) + \mathcal{O}(h^5)$

Plugging this into the numerical scheme we get:

$$\begin{aligned} x(t_n+h) = x(t_n) + \frac{h}{12}\Bigg(23x'(t_n) - 16\left(x'(t_n) - hx''(t_n) + \frac{h^2}{2}x'''(t_n) - \frac{h^3}{6}x^{(4)}(t_n)\right) \\ + 5\left(x'(t_n) - 2hx''(t_n) + 2h^2x'''(t_n) - \frac{8h^3}{6}x^{(4)}(t_n)\right)\Bigg) + \mathcal{O}(h^5) \end{aligned}$$

Rearranging

$$x(t_n+h) \;=\; x(t_n) \;+\; hx'(t_n) \;+\; \frac{h^2}{2}x''(t_n) \;+\; \frac{h^3}{6}x'''(t_n) \;-\; \frac{h^4}{3}x^{(4)}(t_n) + \mathcal{O}(h^5) \tag{20}$$

Local truncation error is defined as $x(t+h) - x(t_n+h)$. Notice that performing (19)-(20), the terms cancel out up to the term with h^3 (included). Therefore

$$LTI \;:=\; x(t+h) - x(t_n+h) \;=\; \mathcal{O}(h^4) \tag{21}$$

3.3 Lorentz system application

Let's re write the Lorentz system in vector form. Let $\mathbf{x} = (x, y, z) = (x(t), y(t), z(t))$, thus $\mathbf{x}' = f(t, \mathbf{x})$ where

$$f(t, \mathbf{x}) = \begin{pmatrix} \sigma(y - x) \\ x(\rho - z) - y \\ xy - \beta z \end{pmatrix} \tag{22}$$

and $\sigma = 10$, $\beta = 8/3$, $\rho = 28$, $x(t_0) = y(t_0) = z(t_0) = 1$, $t = [0, 100]$.

Using the usual notation $f_n = f(t_n, \mathbf{x}_n) = f(t_n, x(t_n), y(t_n), z(t_n))$, we first compute $\mathbf{x}_1$ and $\mathbf{x}_2$ using forward Euler method (see section 3.4) and then we are ready to use the numerical scheme of equation (15):

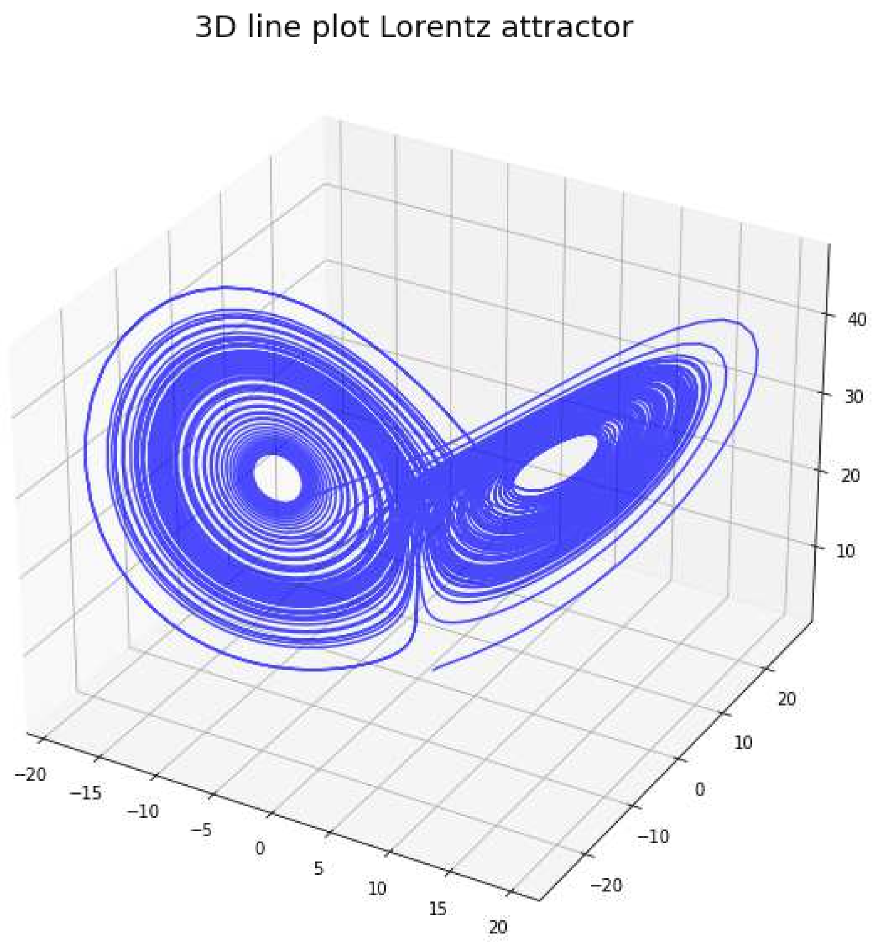
$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \\ z_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} + \frac{23h}{12} \begin{pmatrix} \sigma(y_n - x_n) \\ x_n(\rho - z_n) - y_n \\ x_n y_n - \beta z_n \end{pmatrix} - \frac{16h}{12} \begin{pmatrix} \sigma(y_{n-1} - x_{n-1}) \\ x_{n-1}(\rho - z_{n-1}) - y_{n-1} \\ x_{n-1} y_{n-1} - \beta z_{n-1} \end{pmatrix} + \frac{5h}{12} \begin{pmatrix} \sigma(y_{n-2} - x_{n-2}) \\ x_{n-2}(\rho - z_{n-2}) - y_{n-2} \\ x_{n-2} y_{n-2} - \beta z_{n-2} \end{pmatrix}$$

Given $x_n = x(t_n)$, $y_n = y(t_n)$, $z_n = z(t_n)$, $x(t_0) = y(t_0) = z(t_0) = 1$ and $t_{n+1} = t_n + h$.

Once again the computational algorithm is a Python class that takes as input the time parameters, initial values and the constants of the IVP. Methods are: f that takes as input (t_n, x, y, z) and outputs a three-dimensional vector (as a 1-dimensional array) of the values of $x'(t_n), y'(t_n), z'(t_n)$. Finally, the numerical scheme is computed in the function *numerical_solution* that performs twice the Euler method to get $\mathbf{x}_1$ and $\mathbf{x}_2$ and then proceeds with a for loop to implement the AB(3) numerical scheme.

3.4 How to start the method

Adam-Bashforth(3) is a three-step method. Therefore $\mathbf{x}_{n+1}$ depends on $\mathbf{x}_n, \mathbf{x}_{n-1}, \mathbf{x}_{n-2}$ and in order to be able to compute the first point of this numerical scheme, we need to have $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$. As $\mathbf{x}_0$ is given in the IVP, we use a forward Euler method to compute the next two points. Recall that Euler method for an ODE



of the from $\mathbf{x}'(t) = f(t, \mathbf{x})$ is defined as: $\mathbf{x}_{n+1} = \mathbf{x}_n + h\, f_n$. Therefore:

$$\begin{aligned}\mathbf{x}_1 &= \mathbf{x}_0 + h\, f(t_0, \mathbf{x}_0)\\ \mathbf{x}_2 &= \mathbf{x}_1 + h\, f(t_1, \mathbf{x}_1)\end{aligned}$$

where f is described in (22), $\mathbf{x}_1 = \mathbf{x}(t_0 + h)$ and $\mathbf{x}_2 = \mathbf{x}(t_1 + h)$. Once we have $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$ we can start the AB(3) as in equation (15).