MSAS – Assignment #1: Simulation

Matteo G. Juvara, matr. 249577

1 Implicit equations

Exercise 1

Consider the physical model of a mechanical system made of four rigid rods $\mathbf{a}_i$ with $i \in [1;4]$ in a kinematic chain, as shown in Fig. 1. The system has only two degrees of freedom. The objective is to determine, fixing the angle β, the corresponding angle of equilibrium α between the rods $\mathbf{a}_1$ and $\mathbf{a}_2$. Writing the kinematic closure, it is possible to obtain the following equation

$$\frac{a_1}{a_2}\cos\beta - \frac{a_1}{a_4}\cos\alpha - \cos\beta - \alpha = -\frac{a_1^2 + a_2^2 + a_3^2 + a_4^2}{2a_2a_4} \tag{1}$$

where a_i is the length of the corresponding rod. This equation, called the equation of Freudenstein, can be clearly re-written in the form $f(\alpha) = 0$ and solved, once β is fixed, for a particular value of α. Assume that $a_1 = 10$ cm, $a_2 = 13$ cm, $a_3 = 8$ cm, and $a_4 = 10$ cm and

1) Implement a general-purpose Newton solver (NS).

2) Solve Eq. (1) with NS for $\beta \in [0, \frac{2}{3}\pi]$ with the analytical derivative of $f(\alpha)$ and tolerance set to 10^{-5}. For every β value, use two initial guesses, -0.1 and $\frac{2}{3}\pi$. What do you observe? You can validate your results with the Matlab© built-in function `fsolve`.

3) Repeat point 2) using the derivative estimated through finite differences and compare the accuracy of the two methods.

4) Eq. (1) can be solved analytically only for specific values of β. Find at least one and compare the analytical solution with yours.

5) Repeat point 2) extending the β interval up to π. What do you observe?

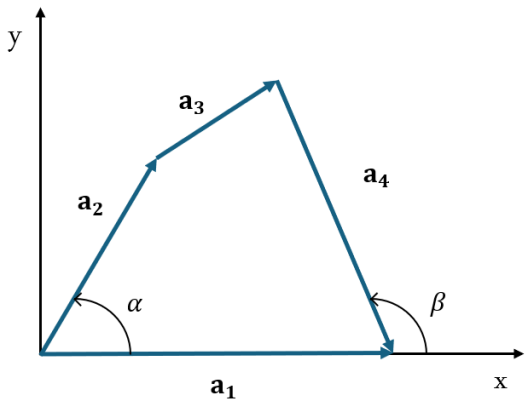

Figure 1: Kinematic chain.

(4 points)

Point 1.1

A Newton solver is an iterative method that approximates the solution of a $f(x) = 0$ equation. The algorithm starts with the function f, its derivative f', and an initial guess x_0, it then proceeds to compute a better approximation to the zeros until the requested tolerance obtained.

$$x_{n+1} = x_n + \frac{f(x_n)}{f'(x_n)} \qquad while \quad |x_{n+1} - x_n| < tol \tag{2}$$

The function `NS` is implemented to use the Newton method in Matlab. Two conditions must be met to obtain a numerical value from the function. Initially, the method reach convergence within the set maximum number of iterations and more importantly the function's derivative evaluated at x_n must differ from zero. The solution is set to `NaN` if the problem doesn't satisfy any of the previous conditions.

Point 1.2

To solve the problem through the use of the Newton method, the analytical derivative of Eq. (1) is computed:

$$\frac{df}{d\alpha} = sin(\alpha - \beta) + \frac{a_1}{a_4} \cdot sin(\alpha) \tag{3}$$

The values of β are obtained through the use of the `linspace` command, from 0 to $\frac{2}{3}\pi$ considering 500 intervals, and with a maximum number of iterations set to 100. The `NS` function is performed using the two distinct initial guesses.

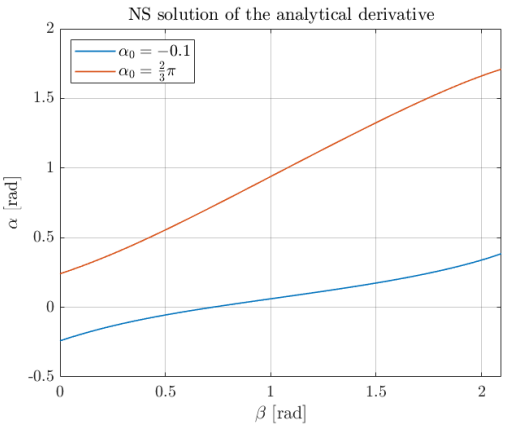

Figure 2: Solution of the analytical problem

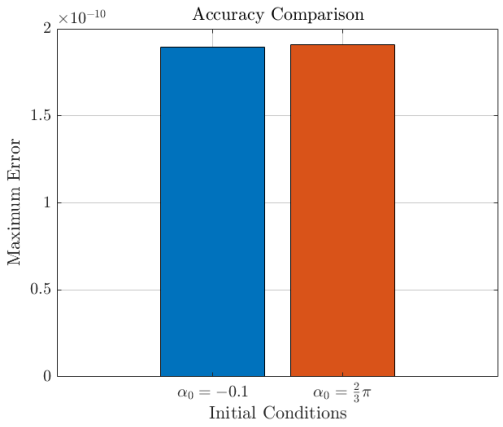

Figure 3: Error compared to `fzero`

As shown in Fig. 2, the initial guess choice strongly influences the solution obtained by the Newton's solver. On the other hand, Fig. 3 shows the mean error between this method and Matlab's `fzero` solver. It can be appreciated that the error is several orders of magnitude lower than the tolerances. The initial conditions seem to not affect the error significantly.

Point 1.3

The finite difference (FD) method is based on replacing the analytical derivatives of a function through finite differences, which can be computed directly using the function's values at discrete points. The function `NS_FD` is created to employ the central finite difference method, with the use of the following formula:

$$\frac{df}{dx} \simeq \frac{f(x+h) - f(x-h)}{2h} \tag{4}$$

The chosen step h is the square root of the machine epsilon $\sqrt{eps}$, as it is small enough to provide sufficient accuracy while avoiding excessive rounding errors.

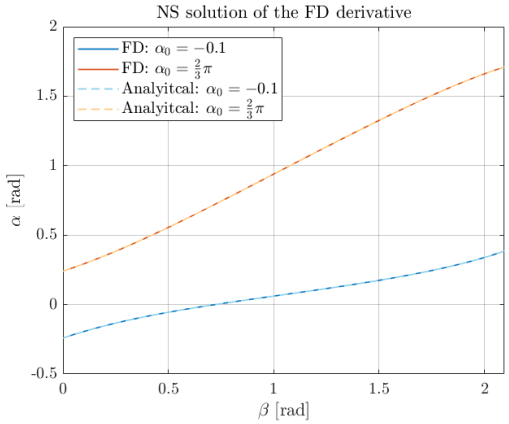

Figure 4: Solution of the FD problem

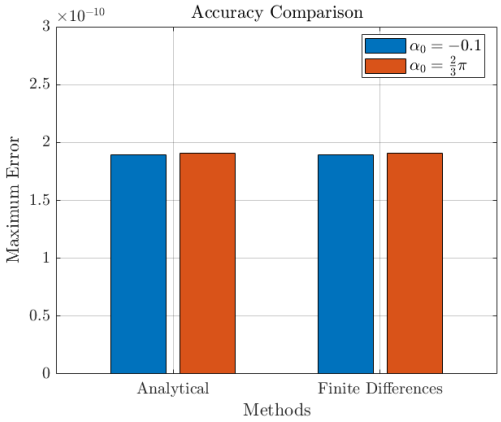

Figure 5: Accuracy of the method

As shown in Fig. 4, the behavior of the solution computed using the finite differences method is almost identical to that of the analytical problem. The mean difference with `fzero` has been computed also for the finite differences method. As seen in Fig. 5 the method yields very similar results to the problem solved with it's analytical derivative.

Point 1.4

For specific values of β the Freudenstain equation (1) can be solved analytically. One of this values is $\beta = 0$, and the analytical solutions for this specific case are reported below:

$$\alpha_1 = cos^{-1}\left(\frac{a_1^2 + 2a_1a_4 + a_2^2 - a_3^2 + a_4^2}{2a_1a_2 + 2a_2a_4}\right) = 0.2408$$
$$\alpha_2 = -cos^{-1}\left(\frac{a_1^2 + 2a_1a_4 + a_2^2 - a_3^2 + a_4^2}{2a_1a_2 + 2a_2a_4}\right) = -0.2408$$

Fig. 6 and Fig. 7 show the difference between the exact solution and the results of the previous points for $\beta = 0$, it can be concluded that all the methods obtain a comparable solution with the analytical one. It must be noted that the error for $\alpha_0 = -0.1$ is significantly lower than that for $\alpha_0 = \frac{2}{3}\pi$.

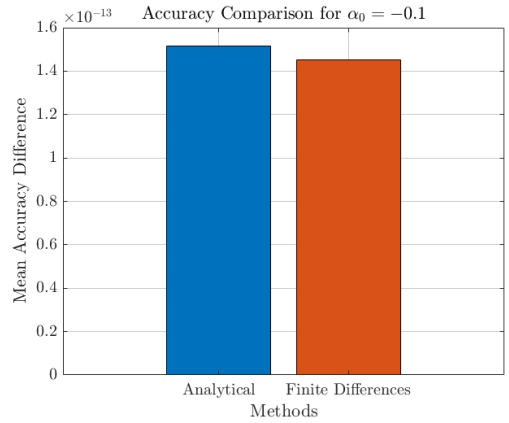

Figure 6: Error comparison for $\alpha_0 = -0.1$

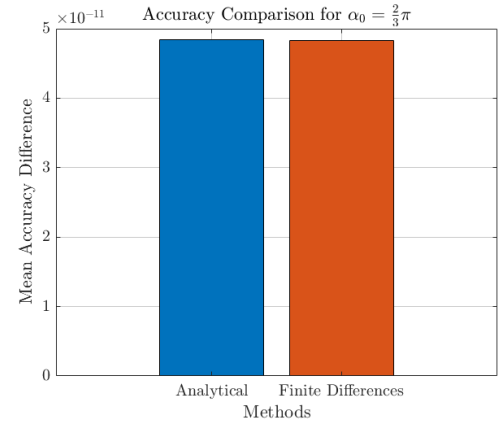

Figure 7: Error comparison for $\alpha_0 = \frac{2}{3}\pi$

Point 1.5

The same analysis performed for Point 2 is repeated, extending the β interval up to π.

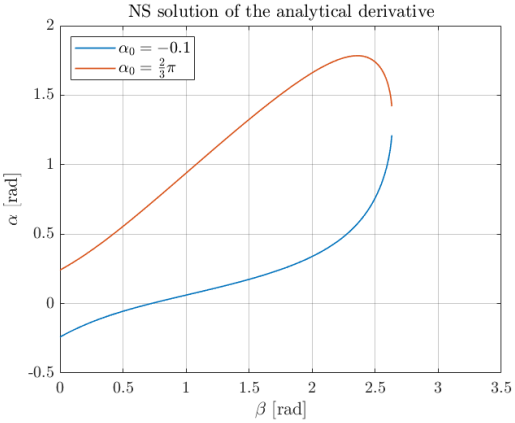

Figure 8: NS solution for $\beta \subset [0\ \pi]$

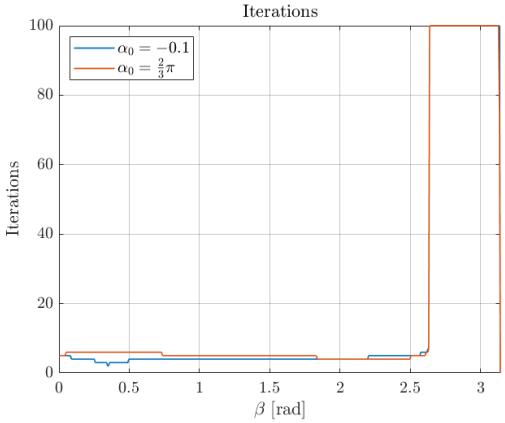

Figure 9: Iterations for $\beta \subset [0\ \pi]$

Fig. 8 shows that for $\alpha_0 = -0.1$, the value of α continues to increase, while, for $\alpha_0 = \frac{2}{3}\pi$, the trend reverses around $\beta \approx 2.36$ and α decreases as β increases. The curves stop at $\beta = 0.8383\pi$ as the Newton method no longer converges within the maximum allowed iterations. This phenomenon is easily visible in Fig. 9, where the iterations needed by the method for each beta are reported. For $\beta = \pi$, the number of iterations for both initial guesses lower to zero, as the derivative of Eq. (1) is zero, making the Newton method inapplicable.

2 Numerical solution of ODE

Exercise 2

The physical model of a moving object with decreasing mass is shown in Fig. 10. The mathematical model of the system is expressed by the equations:

$$m(t)\frac{dv}{dt} = \overline{F} + f(t) - \alpha \cdot v \tag{5}$$

$$m(t) = m_0 - c_m \cdot t \tag{6}$$

$$\overline{F} = -\ 0.5 \cdot \rho \cdot C_d \cdot A_m \cdot v^2 \tag{7}$$

Assuming that: $v(0) = 0$ [m/s], $m_0 = 20$ [kg], $c_m = 0.1$ [kg/s], $f(t) = 1$ [N], $\alpha = 0.01$ [Ns/m], $\rho = 0$ [kg/m^3], and Eq. (5) has the exact solution:

$$v(t) = \frac{f(t)}{\alpha} - \left[\frac{f(t)}{\alpha} - v(0)\right]\left[1 - \frac{c_m \cdot t}{m_0}\right]^{\frac{\alpha}{c_m}} \tag{8}$$

Answer to the following tasks:

1) Implement a general-purpose, fixed-step Heun's method (RK2);
2) Solve Eq. (5) using RK2 in $t \in [0, 160]$s for $h_1 = 50$, $h_2 = 20$, $h_3 = 10$, $h_4 = 1$ and compare the numerical vs the analytical solution;
3) Repeat points 1)–2) with RK4;
4) Trade off between CPU time & integration error.

Now, assuming that: the fluid density is $\rho = 900$ [kg/m^3], $C_d = 2.05$, and $A_m = 1$ [m^2], answer to the following tasks:

1) Solve Eq. (5) using RK2 in $t \in [0, 160]$s for $h_1 = 1$;
2) From the results of the previous point, use a proper ode of Matlab to solve Eq. (5) ;
3) Discuss the results.

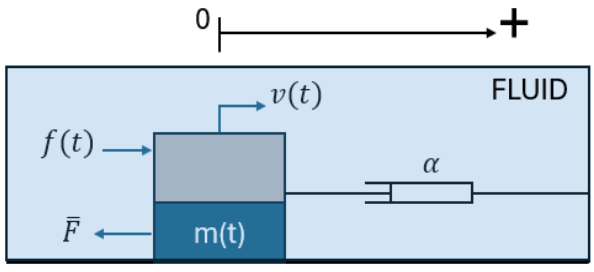

Figure 10: Moving object with decreasing mass.

(5 points)

Point 2.1

Heun's method is a second-order single-step integration method, which uses a forward Euler (FE) method as a predictor and then a union of an FE and a backward Euler (BE) method as a corrector, as shown below:

$$\begin{cases} x_{k+1}^P = x_k + h \cdot f(t_k, x_k) \\ x_{k+1}^C = x_k + \dfrac{h}{2} \cdot (f(t_k, x_k) + f(t_{k+1}, x_{k+1}^P) \end{cases}$$

The Matlab function `Heun` implements Heun's method. By substituting the predictor inside the corrector equation and then solving for x_{k+1} we obtain the solution of the problem. This is performed for every $i \subset [2 \;\; i_{max}]$, as the method has to receive as input the initial conditions of the problem x_1.

Point 2.2

The solution to Eq. (5) is obtained using different integration step sizes. The results are them compared with the analytical solution, computing the error at each step.

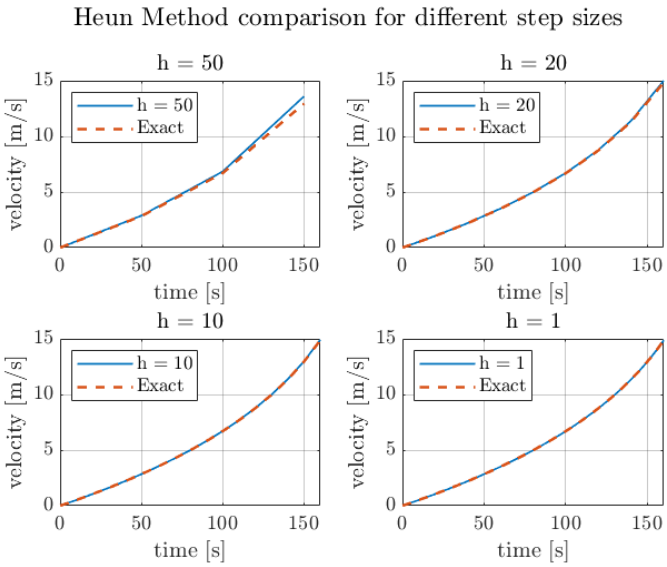

Figure 11: Heun's method solution

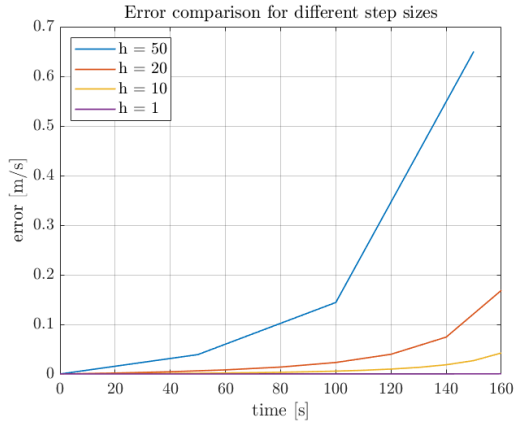

Figure 12: Error comparison

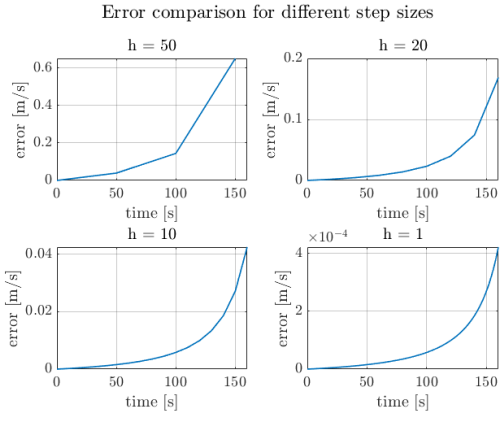

Figure 13: Error for each step size

As expected, increasing the step size reduces the precision of the solution, this is clearly visible in Fig. 11. A smaller step allows the method to follow more accurately the curve of the real function, reducing the local truncation error at each step. As shown in Fig. 12, it can be noted that the error increases proportionally in time and that the step size significantly decreases the error between the exact solution and the Heun's method.

Point 2.3

The fourth-order Runge-Kutta method (RK4) is a fourth-order method made up of three predictors and a corrector which contains them all. The RK4 method can bee seen below:

$$\begin{cases} x^{P1} = x_k + \dfrac{h}{2} \cdot f(t_k, x_k) \\ x^{P2} = x_k + \dfrac{h}{2} \cdot f(t_k + \dfrac{h}{2}, x^{P1}) \\ x^{P3} = x_k + h \cdot f(t_k + \dfrac{h}{2}, x^{P2}) \\ x_{k+1} = x_k + h \cdot \left[\dfrac{1}{6} f(t_k, x_k) + \dfrac{1}{3} f(t_k + \dfrac{h}{2}, x^{P1}) + \dfrac{1}{3} f(t_k + \dfrac{h}{2}, x^{P2}) + \dfrac{1}{6} \cdot f(t_k + h, x^{P3}) \right] \end{cases}$$

The Matlab function `RK4` is implemented to repeat the previous analysis with a fourth-order Runge-Kutta method. Similarly to the `Heun` function, this algorithm calculates each predictor and the corrector for every $i \subset [2 \quad i_{max}]$, as this method as well receives as input the initial conditions.

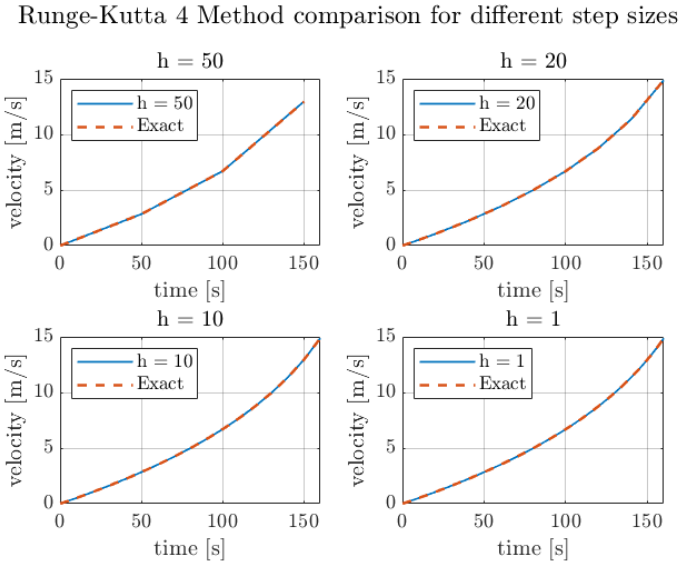

Figure 14: RK4 method solution

In Fig. 14, it is clearly visible how the behavior of the RK4 solution matches the exact solution more precisely than Heun's solution. This is confirmed by Fig. 15 and Fig. 16, where it's clearly visible the decrease of the errors of the RK4 method with respect to the errors computed for the Heun method. The lower is the step size h the more the error has decreased using the RK4 method, even by several order of magnitudes.

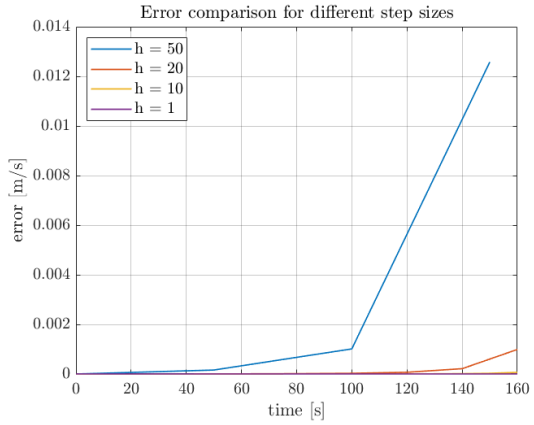

Figure 15: Error comparison

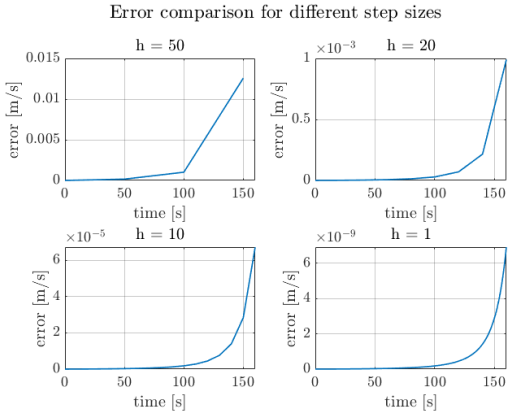

Figure 16: Error for each step size

Point 2.4

The CPU time is computed by calculating the mean elapsed time and the variance of the functions `Heun` and `RK4` 10000 times. The integration error, evaluated as seen in Eq. (9), mean and variance for the two methods are computed.

$$err(t_k) = |x_{RKi}(t_k) - x_{ex}(t_k)| \tag{9}$$

As seen earlier and reported clearly in Fig. 17 and Fig. 18, the error lowers significantly by lowering the step size or by increasing the order of the method, but the trend stays very similar between methods.

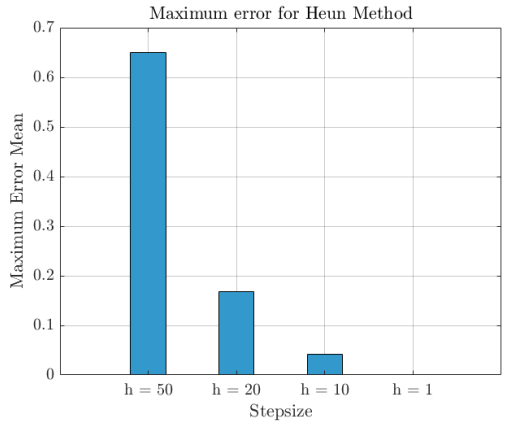

Figure 17: Errors of the Heun Method

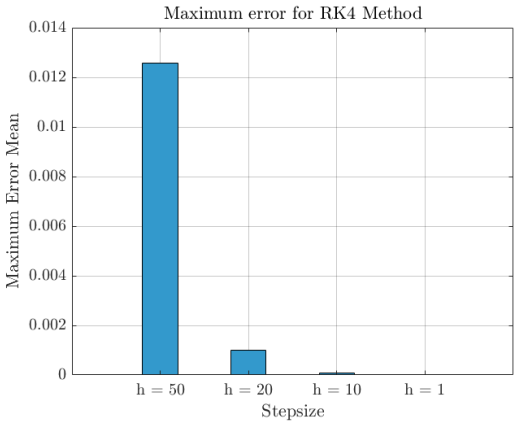

Figure 18: Errors of the RK4 Method

On the other hand the behavior of the CPU time is completely opposite to the error and the lowest values is obtained using the Heun method with h = 50 as a step size. In general the higher the step size and the lower the order of the method the lower will the CPU time be. It must be noted that, in contrast with the error, the CPU time for each method are comparable between each other and Heun's method is only slightly faster than the Runge-Kutta 4, as shwon in Fig. 19 and Fig. 20.

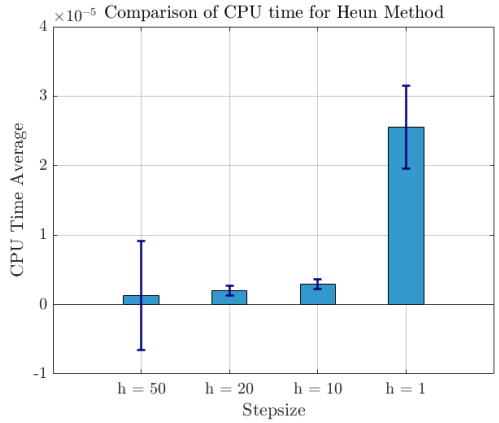

Figure 19: CPU time of the Heun Method

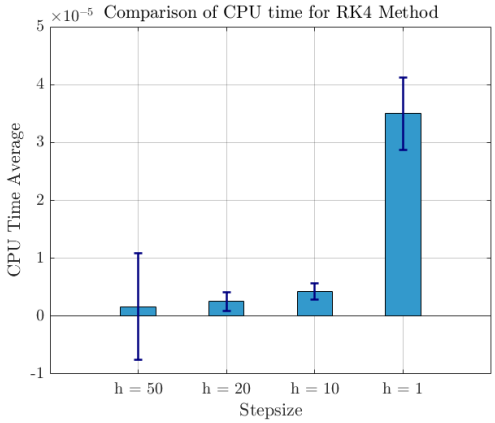

Figure 20: CPU time of the RK4 Method

Point 2.5

By changing the density to $\rho = 900 kg/m^3$, the solution obtained using Heun's method with h = 1 becomes unstable, as shown in Fig. 21. To understand the reason for the divergence, the eigenvalue of the problem is computed by linearising the equation and calculating the Jacobian, obtaining $\lambda = -3.0373$. The provided h is too large to guarantee the stability of the solution and to secure the convergence of Heun's method with the updated parameters, the step size should be lower than 0.6585, to ensure the eigenvalue will be inside the stability region of the method as seen in Fig. 22.

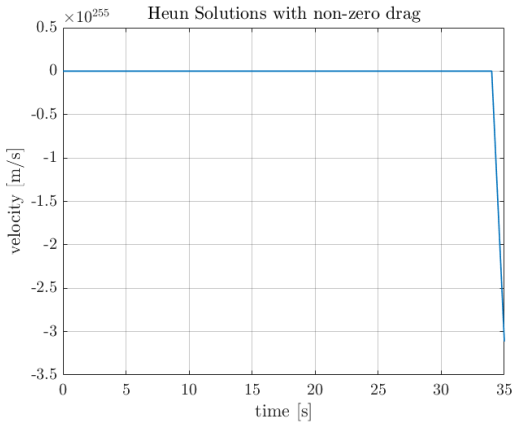

Figure 21: Heun's solution

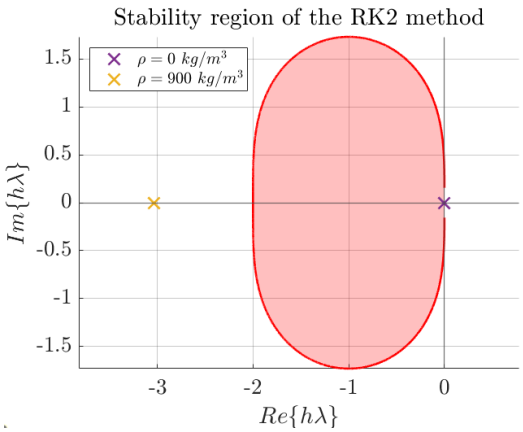

Figure 22: Heun's stability region

To properly choose an ode of Matlab, the problem's eigenvalues have to be part of the method's stability region. Considering the Runge-Kutta method, the linearized λ is inversely proportional to the mass (10), which decreases over time, so λ increases and the system becomes stiffer as time grows. For this reason Matlab's `ode15s` was chosen to solve this problem, the results are shown in Fig. 23.

$$\lambda = \frac{-\rho \cdot C_D \cdot A_m \cdot v^* - \alpha}{m(t)} \tag{10}$$

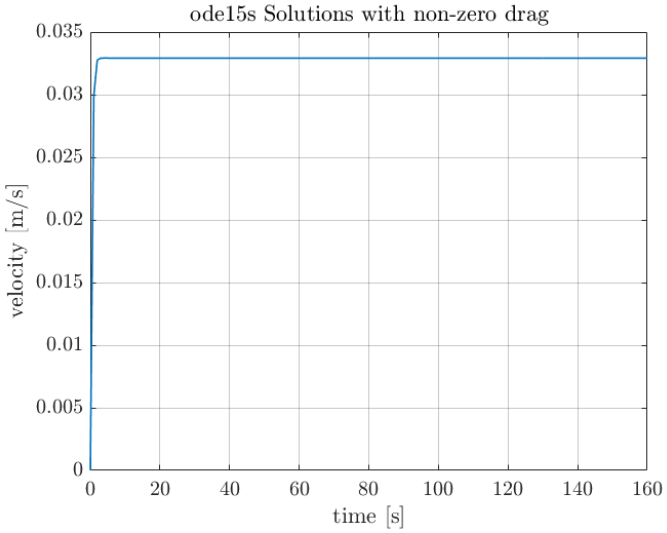

Figure 23: `ode15s` solution

Exercise 3

Let $\dot{\mathbf{x}} = A(\alpha)\mathbf{x}$ be a two-dimensional system with $A(\alpha) = [0, 1; -1, 2\cos\alpha]$. Notice that $A(\alpha)$ has a pair of complex conjugate eigenvalues on the unit circle; α denotes the angle from the Re$\{\lambda\}$-axis.

a) Runge-Kutta methods

1) Write the operator $F_{\text{RK2}}(h, \alpha)$ that maps $\mathbf{x}_k$ into $\mathbf{x}_{k+1}$, namely $\mathbf{x}_{k+1} = F_{\text{RK2}}(h, \alpha)\,\mathbf{x}_k$. 2) With $\alpha = \pi$, solve the problem "Find $h \geq 0$ s.t.max $(|\text{eig}(F(h, \alpha))|) = 1$". 3) Repeat point 2) for $\alpha \in [0, \pi]$ and draw the solutions in the $(h\lambda)$-plane. 4) Repeat points 1)–3) with RK4.

b) Backinterpolation methods

Consider the backinterpolation method $\text{BI2}_{0.3}$. 1) Derive the expression of the linear operator $B_{\text{BI2}_{0.3}}(h, \alpha)$ such that $\mathbf{x}_{k+1} = B_{\text{BI2}_{0.3}}(h, \alpha)\mathbf{x}_k$. 2) Using the same approach of a), draw the stability domain of $\text{BI2}_{0.3}$ in the $(h\lambda)$-plane. 3) Derive the domain of numerical stability of BI2_θ for the values of $\theta = [0.2, 0.4, 0.6, 0.8]$.

(8 points)

Point 3.1

The operator F_{RK2}, which maps x_k into x_{k+1}, is shown in Eq. (11).

$$F_{RK2} = I + h \cdot A + \frac{h^2}{2} \cdot A^2 \tag{11}$$

where I is the identity matrix, A is the system's matrix defined ad A = [0 1; 1 cos(α)], h is the step size, and α is the angle formed with the real axis.

To solve the problem "Find $h \geq 0$ s.t. $max(|eig(F(h, \alpha))|) = 1$", the function `stabilityPlot` is implemented, where the problem is solved using Matlab's `fzero` and the eigenvalues of A are computed for each α value. The α vector is defined using Matlab's `linspace` command, using n = 100, and this vectors is used as the initial conditions to compute h, unless specified otherwise. To plot the stability region of the method, the solved step is multiplied with the eigenvalues of matrix A for each value of α, the result is then plotted in the h-λ plane as seen in Fig. 24.

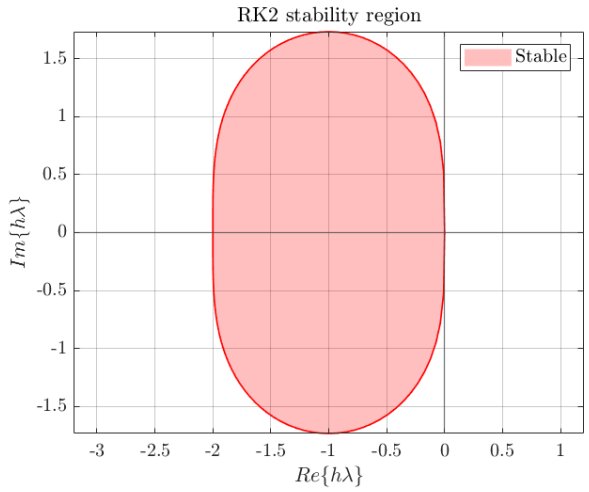

Figure 24: Stability region of the RK2 method

The same process is repeated for the fourth order Runge Kutta method, of which it's operator F_{RK4} it's reported in Eq. (12).

$$F_{RK4} = I + h \cdot A + \frac{h^2}{2} \cdot A^2 + \frac{h^3}{6} \cdot A^3 + \frac{h^4}{24} \cdot A^4 \tag{12}$$

For this method, the α vector is divided into two parts to ensure precision throughout the whole region boundary. The first vector ranges from 0 to $\pi/2$, split between 1000 intervals. In this region the stability region boundary varies in a steeper way, thus requiring a more refined α vector. On the other hand, the second vector ranges from $\pi/2$ to π, divided in 100 intervals, for the stability variation is softer. For this α vector a single guess vector is sufficient to correctly plot this region while, for the second α vector, two different guess vectors must to be employed to correctly recreate all of the curves of the boundary. Fig. 25 shows the stability region of the RK4 method.

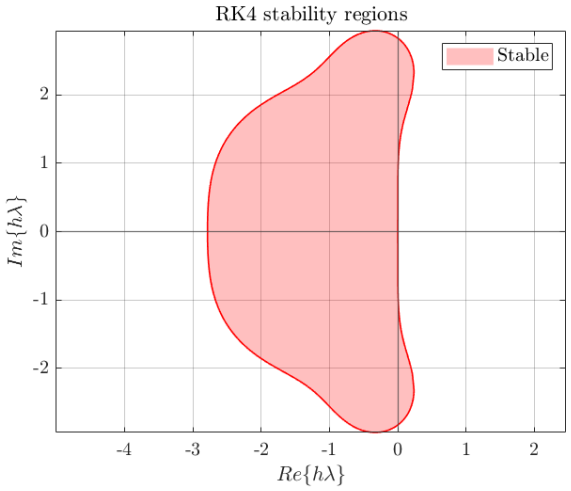

Figure 25: Stability region of the RK4 method

Point 3.2

The general operator, in function of theta, for back interpolation methods can be seen below.

$$B_{BI2} = \left[I - A \cdot (1-\theta)h + \frac{(A \cdot (1-\theta)h)^2}{2} \right]^{-1} \left[I + A \cdot \theta h + \frac{(A \cdot \theta h)^2}{2} \right] \tag{13}$$

By substituting $\theta = 0.3$ in Eq. (13), the operator for the specific method $B_{BI2_{0.3}}$ can be derived, as seen in Eq. (14).

$$B_{BI2_{0.3}} = \left[I - A \cdot 0.7h + \frac{(A \cdot 0.7h)^2}{2} \right]^{-1} \left[I + A \cdot 0.3h + \frac{(A \cdot 0.3h)^2}{2} \right] \tag{14}$$

The same is repeated for $\theta = [0.2, 0.4, 0.6, 0.8]$. Their stability regions have been plotted by using different discretization of the α vector as well as different initial guesses for each θ case. For $\theta = 0.2, 0.3$, and 0.8 a singe α vector is sufficient with a discretization of 100 intervals. On the other hand, for θ values of 0.4 and 0.6 two distinct α vectors have to be employed, one of which is more refined to ensure precision of the boundary near the imaginary axis.

This family of methods is implict for every $\theta < 0.5$. This means that the stability region is positioned outside the boundary, which grows for increasing θ. For $theta > 0.5$ the methods are

explicit and the stability region lies inside the boundary, which in contrary shrinks for increasing θ. The stability regions for all back interpolation methods are shown in the figures below.

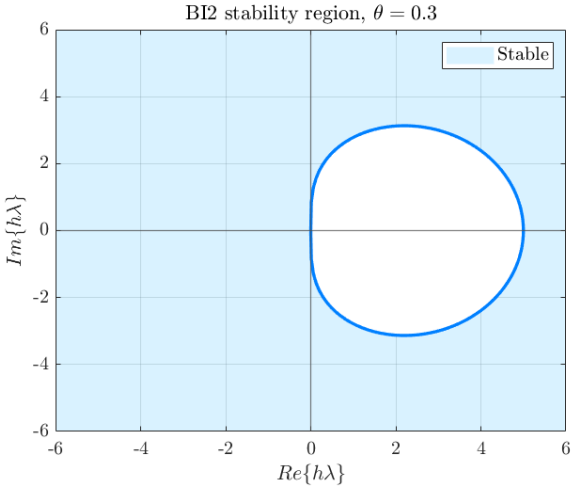

Figure 26: Stability region of the $BI2_{0.3}$ method

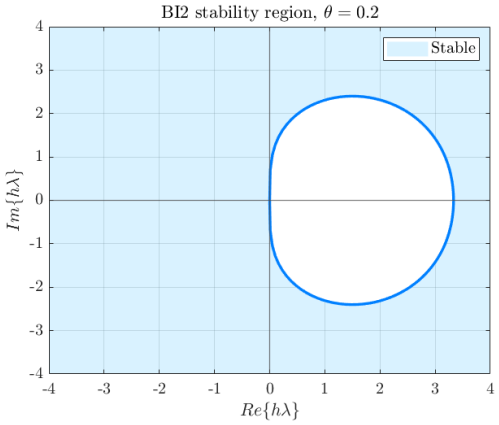

Figure 27: Stability region of the $BI2_{0.2}$ method

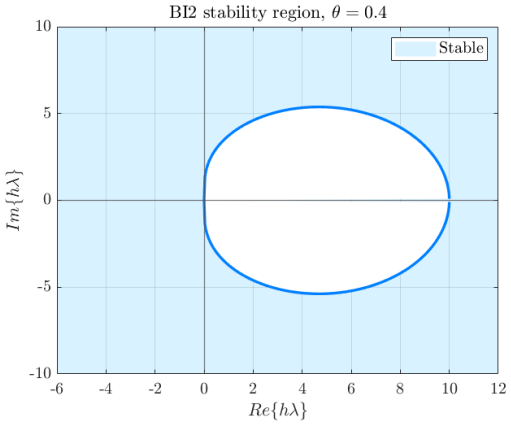

Figure 28: Stability region of the $BI2_{0.4}$ method

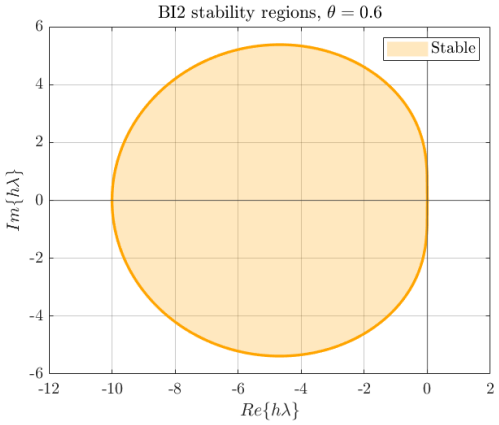

Figure 29: Stability region of the $BI2_{0.6}$ method

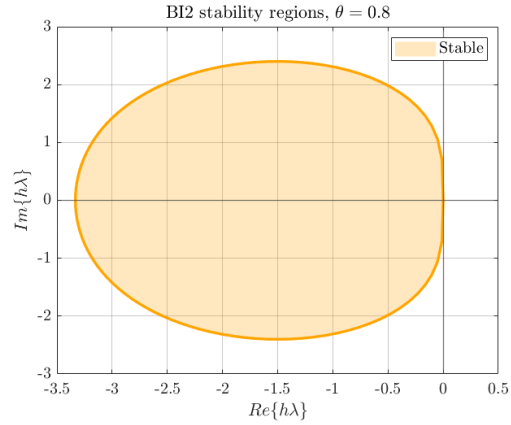

Figure 30: Stability region of the $BI2_{0.8}$ method

Exercise 4

The cooling problem of a high-temperature mass is shown in Fig. 31. Assuming that: K_c= 0.0042 [J/(s K)] and K_r = 6.15 x 10^{-11} [J/(s K^4)] are the convective and radiation heat loss coefficients respectively, $\widetilde{T}$ is the mass temperature in time, T_a = 277 [K] is the surrounding air temperature, C= 45 [J/K] is the mass thermal capacity, and $\widetilde{T}(0)$ = 555 [K] is the initial mass temperature.

1) Write the mathematical model of the system.
2) Solve the mathematical model using: RK2 with h= 720, and RK4 with h= 1440.
3) Compare the solution of point 2) with the "exact" one. Discuss the results. (Hint: use an ODE solver of Matlab).

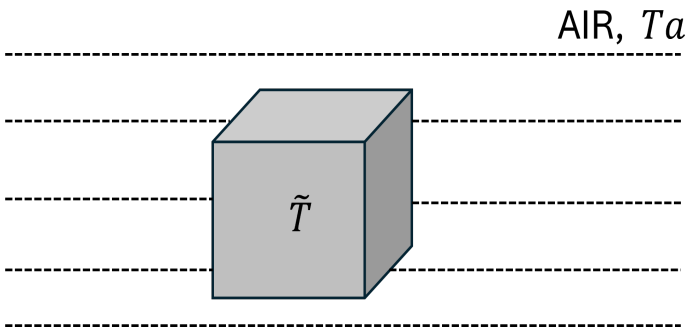

Figure 31: The cooling of high-temperature mass.

(4 points)

Point 4.1

The energy conservation equation is exploited to model the cooling problem of a high-temperature mass, assuming the internal body temperature to be lumped in $\widetilde{T}$.

$$\frac{dT}{dt} = \frac{(-Q_{conv} - Q_{rad})}{C} = -\frac{1}{C} \cdot [K_c \cdot (\widetilde{T} - T_a) - K_r \cdot (\widetilde{T}^4 - T_a^4)] \tag{15}$$

Point 4.2

The model is solved through the second order Runge-Kutta method, using the `Heun` function with h = 720, and through the fourth order Runge-Kutta method, using the `RK4` function with h = 1440. As a the maximum simulation time it was chosen to use 72000 seconds to ensure at least 100 time steps are performed for the RK2 method and 50 time steps are performed for the RK4 method. The solutions of both methods are reported in Fig. 32 and Fig. 33 and it must be noted that both methods show a steep temperature decrease initially, followed by a plateau at the fixed air temperature, indicating that an equilibrium state between the mass and the air has been reached.

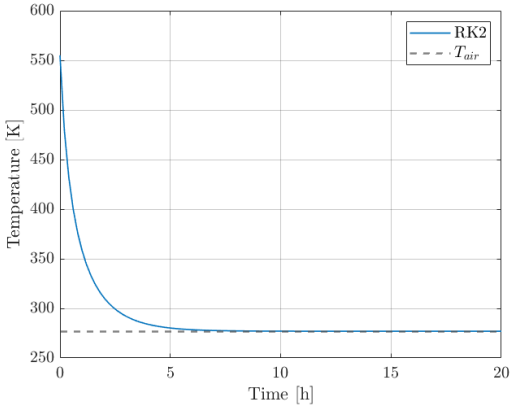

Figure 32: Solution using Heun

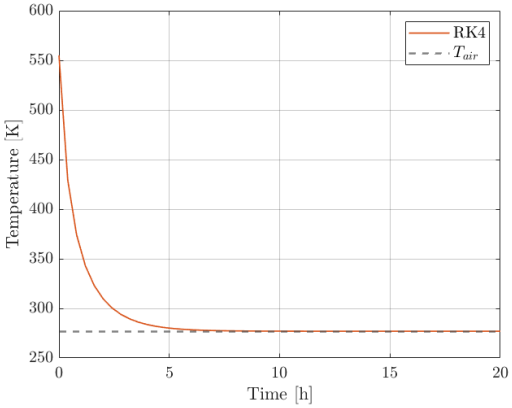

Figure 33: Solution using RK4

Point 4.3

The solutions of the two methods are compared with an 'exact' solution to validate the results. As an 'exact' solution it was chosen to use a higher-order method such as Matlab's `ode78` solver. This solver uses a variable step for the seventh and eight-order Runge-Kutta methods, which makes it particularly useful for problems with a smooth solution such as this one. Fig. 34 and Fig. 35 show the comparison between the methods and Matlab's solver and the error between them in time.

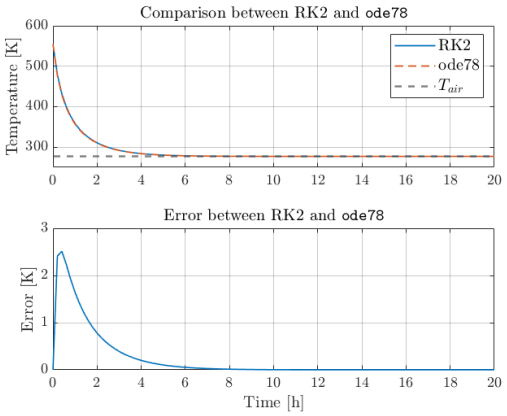

Figure 34: Comparison Heun vs `ode78`

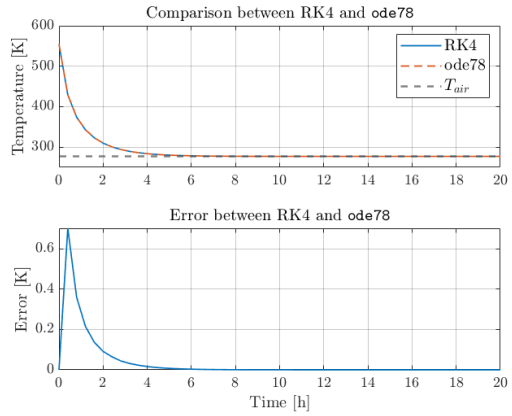

Figure 35: Comparison RK4 vs `ode78`

As expected, the fourth order Runge-Kutta method is more accurate compared to the second order Runge-Kutta method. It may be interesting to determine when the equilibrium between the temperatures of the mass and the air is achieved. The equilibrium can be considered achieved when the difference between the temperatures is lower than a specific tolerance, in this case set at `1e-3`. The equilibrium timestamps are reported in Table 1.

Method	RK2	RK4	`ode78`
Time Instant [s]	56880	57600	56880

Table 1: Equilibrium Timestamps

Exercise 5

Consider the electrical circuit in Fig. 36. At time t>=0 the switch disconnects the battery and the capacitor discharges its stored energy to the circuit. Answer to the following tasks:

1) Find the state variable form of the circuit with x_1=q and x_2=$\frac{dq}{dt}$, where q is the charge on capacitor.

2) Show that the system is stiff when the circuit parameters are R=25 [Ω], L = 20 [mH], C = 200 [mF], and v_0 = 12 [V]. Represent the eigenvalues on the ($h\lambda$)-plane both for RK2 and IEX4 stability domain.

3) Use IEX4 to simulate the transient response of the system with the parameters of point 2), and determine the largest step size such that RK2 yields a stable and accurate solution.

4) Discuss the results.

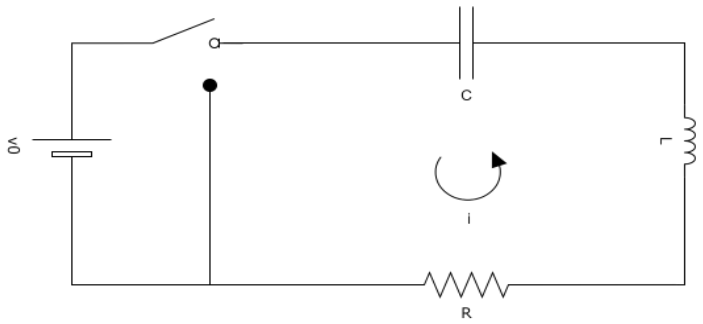

Figure 36: Electric circuit model.

(4 points)

Point 5.1

The equation of the RLC circuit are reported in Eq. (16) in function of the current. Knowing that the current can be written as the derivative of the charge, the RLC equation can be written in function of the charge as well, as shown in Eq. (17). Knowing that the capacitor equation is $C \cdot {}^{dV_c}/_{dt} = i$, the voltage of the capacitor can be written as $V_c = {}^{q}/_{C}$.

$$L \cdot \frac{di}{dt} + R \cdot i + V_c = 0 \qquad (16) \qquad\qquad L \cdot \frac{d^2q}{dt^2} + R \cdot \frac{dq}{dt} + \frac{q}{C} = 0 \qquad (17)$$

From this equation the state model of the system can be defined, using as variables $x_1 = q$ and $x_2 = {}^{dq}/_{dt}$. The system can be written in the form $\dot{x} = A \cdot x$ as:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ {}^{-1}/_{LC} & {}^{-R}/_{L} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad (18)$$

Point 5.2

The eigenvalues of the matrix A are λ_1 = -0.2 and λ_2 = -1248.8 . The second eigenvalue is very far from the first one and from the origin, making the system a stiff system. The second order Runge-Kutta method has difficulties solving this problem as the second eigenvalue is significantly outside the stability region, requiring a excessively low step size h, as seen in Fig. 37.

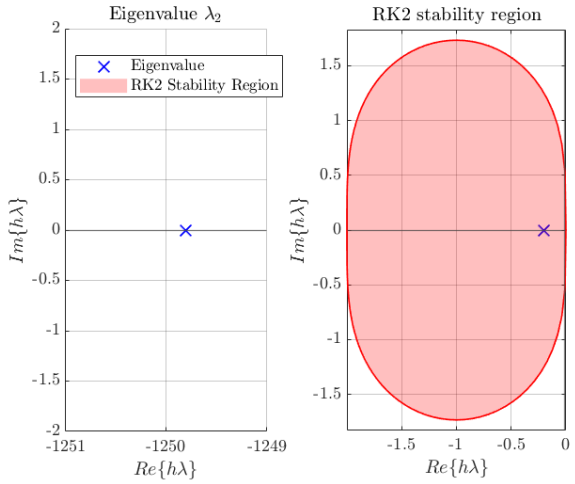

Figure 37: Stability region of the RK2 method

To correctly solve this problem an A-stable method is needed, such as the fourth-order interpolation extrapolation method. As seen in Fig. 38, the eigenvalues of the problem, although distant, fall in the stability region of the method.

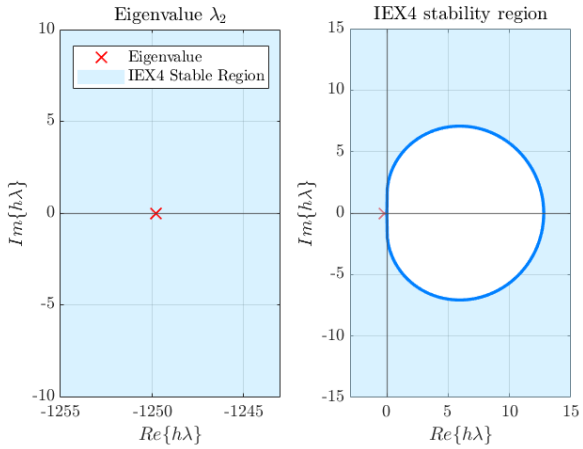

Figure 38: Stability region of the IEX4 method

Points 5.3 and 5.4

The function `IEX4` is implemented in Matlab to simulate the transient response of the system, considering a time step of h = 0.1 and maximum simulation time of thirty seconds.

As seen in Fig. 39 and Fig. 40, the capacitor's charge and current decrease exponentially over time, representing the discharge of the capacitor. Both the state variables tend to zero in the first 20 second of simulation without any oscillations, indicating that the system is over-damped ($\xi > 1$). This can be verified by computing the damping factor of the RLC circuit as $\xi = 0.5 \cdot R\sqrt{C/L} = 39.53$.

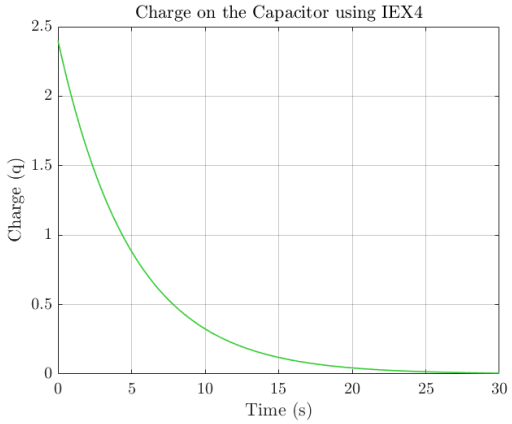

Figure 39: Charge using IEX4 method

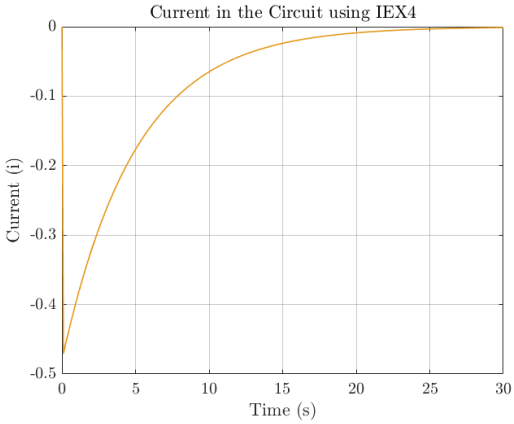

Figure 40: Current using IEX4 method

In order to solve the problem with the RK2 method, the step size must be lower than the following condition:

$$h < \frac{-2}{\lambda_2} = 0.0016 \tag{19}$$

The chosen step size, such that the RK2 method yields a stable and accurate solution, is h = 0.0015. Due to this very small step size, the simulation time and the computational effort increase significantly.

A comparison of the mean computational time between the two methods is performed by running both functions 100 times, balancing the statistical accuracy and the computational effort. The results are shown in Table 2 and it is clearly visible that the IEX4 method is statistically one order of magnitude faster than the RK2 method.

	Mean computational time [s]
Heun (RK2)	$9.74 \cdot 10^{-3}$
IEX4	$9.68 \cdot 10^{-4}$

Table 2: Computational time comparison

Exercise 6

Consider the bouncing ball physical model in Fig. 41. The mathematical model of the system is represented by the following equations:

$$\frac{dv}{dt} = -g + \frac{F_D}{m_b} + \frac{F_A}{m_b} \tag{20}$$

$$\frac{dx}{dt} = v \tag{21}$$

$$v^+ = -k \cdot v^- \tag{22}$$

$$F_D = -\ 0.5 \cdot \rho \cdot C_d \cdot A_b \cdot v \cdot |v| \tag{23}$$

$$F_A = \rho \cdot V_b \cdot g \tag{24}$$

where the initial conditions are: $v(0) = 0$ [m/s], and $x(0) = 10$ [m].

Assuming that: the ball velocity before (v^-) and after (v^+) the ground impact is governed by the attenuation factor $k = 0.9$, the air density is $\rho = 1.225$ [kg/m^3], the ball mass is $m_b = 1$ [kg], the ball area is $A_b = 0.07$ [m^2], the ball volume is $V_b = 0.014$ [m^3], and the drag coefficient is C_d=1.17, answer to the following tasks:

1) Solve the mathematical model in the time interval $t \in [0, 10)$s using an ode integrator of Matlab.

2) Repeat point 1) for ρ=15 [kg/m^3]. Which is the difference with respect to point 1)?

3) Now solve the problem when the fluid density is ρ=60 [kg/m^3] in $t \in [0, 10)$s. Discuss the result.

4) Repeat point 3) using RK4 for $h = 3$. Discuss the result.

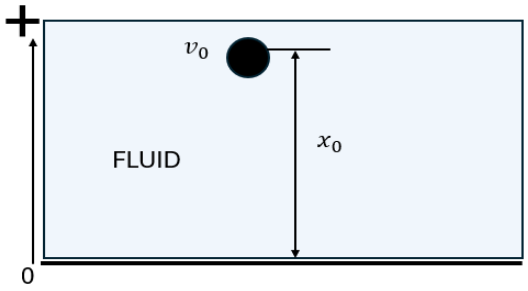

Figure 41: Bouncing ball system.

(5 points)

Point 6.1

The bouncing ball mathematical model is properly implemented in Matlab through the use of an event function. The event function defines the event that the solver has to detect while solving the equation as well as the actions it has to perform once detected. The event occurs when the specified event function crosses zero. The state variables chosen for the model are the vertical position of the ball x and the vertical velocity v.

The function `groundEvent` is implemented to detect if the position of the ball coincides with the ground before bouncing. When the event occurs the integration is stopped and the

initial parameters are changed before resuming integration. In particular the position is set to zero and the velocity is set to $v_k = -k \cdot v_{k-1}$, this is done to simulate the bouncing with an attenuation factor.

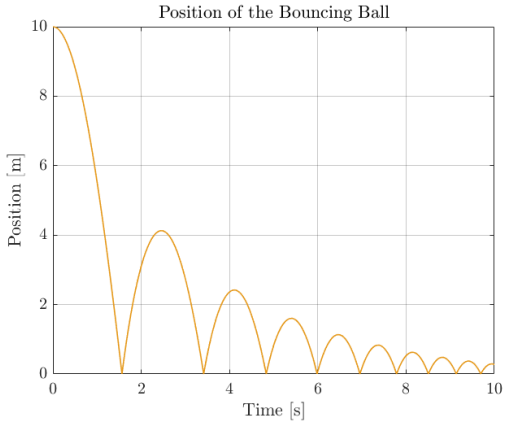

Figure 42: Position of the bouncing ball

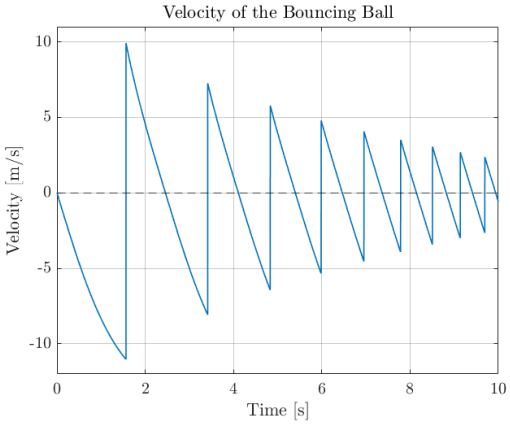

Figure 43: Velocity of the bouncing ball

The position x, reported in Fig. 42, shows several bounces with decreasing height due to the effect of drag. On the other hand, the velocity, shown in Fig. 43, exhibits discontinuities as the model instantly changes the velocity of the model after a bounce. Furthermore the module of the velocity decreases in time due to the drag effects.

Point 6.2

Increasing the density to $\rho = 15 kg/m^3$ has two main effects on the model. Firstly, an increase of the buoyancy force is observed, leading to larger bounces of the ball. Secondly, there is an increase of the drag force, leading to a higher decrease of position and velocity modules.

For this model a break condition is implemented is the solver to prevent from numerical discrepancies due to very small bounces at the end of the simulation. Once the velocity module decreases under a specific tolerance, set at `1e-6`, the simulation is stopped. The results are shown in the figures below and it can be noted that the ball stops bouncing before the simulation end time is reached.

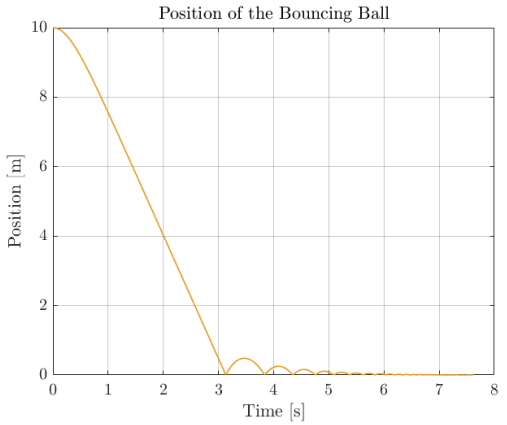

Figure 44: Position of the bouncing ball

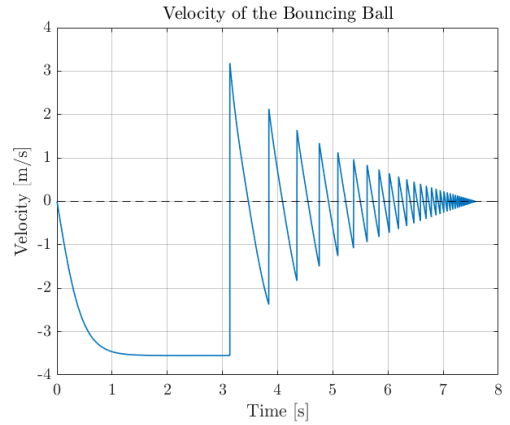

Figure 45: Velocity of the bouncing ball

Point 6.3

By further increasing the density to $\rho = 60 kg/m^3$, the buoyancy force is so high that the ball does not reach the ground within ten seconds, as shown in Fig. 46. The velocity decreases exponentially with no discontinuities present due to the absence of bouncing.

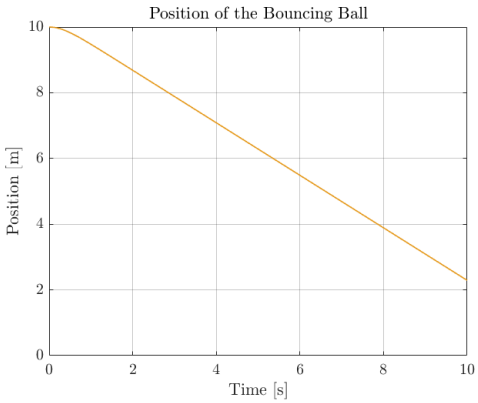

Figure 46: Position of the bouncing ball

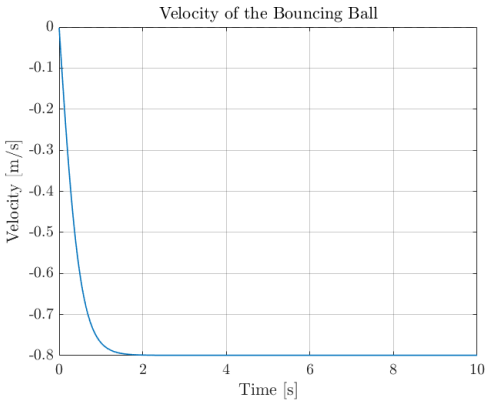

Figure 47: Velocity of the bouncing ball

Point 6.4

To verify if the RK4 method with a step size of h = 3 can successfully solve the problem, the system is linearized to compute its eigenvalues. Since both eigenvalues amount to $\lambda_1 = \lambda_2 = 0$, the linearized system can be considered marginally stable, as shown in Fig. 50. The stability fo the solution depends on the non-linearity effects and the numerical behavior of the solution. The choice of h plays a major role as larger step sizes may cause significant numerical errors which could lead to divergence in the solution. This happens for h = 3 while for lower step sizes, such as h = 0.5, the numerical solution is stable, as it can clearly be seen comparing Fig. 48 and Fig. 49.

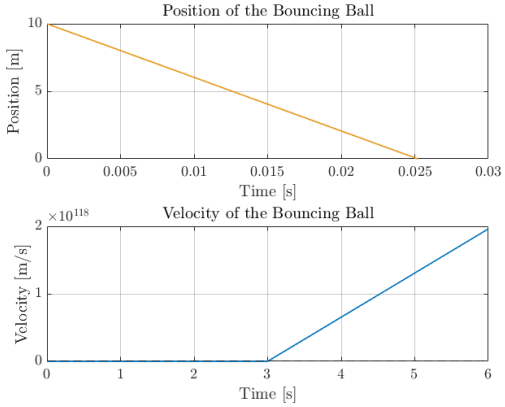

Figure 48: Solution using RK4 with h = 3

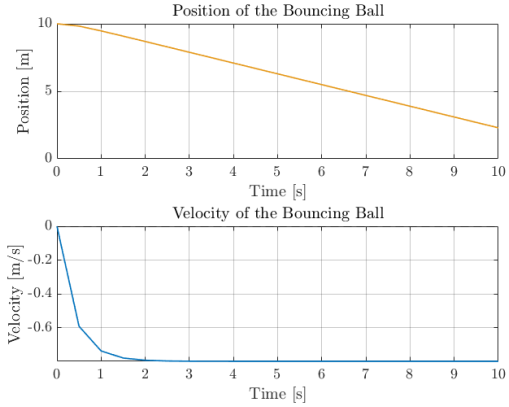

Figure 49: Solution using RK4 with h = 0.5

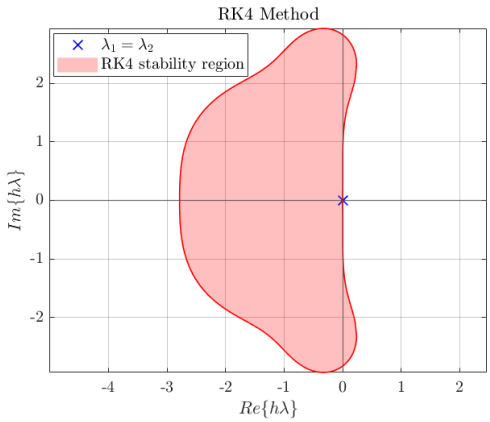

Figure 50: Stability region of the RK4 method and eigenvalues of the problem