Astrophysical Lab: Winter Semester 2024 / 2025

Title of experiment: Ordinary Differential Equations - Cosmological Models

Supervisor (1st afternoon): Varun Kushwaha

Supervisor (2nd afternoon): Varun Kushwaha

Group (number): 18

Names (of the students): Jennifer Fabà-Moreno (MA 1) and Pablo Vega del Castillo (MA 1)

(Please also indicate whether you are a Master's student in Astrophysics (MA) or in Physics (MP) and in which semester you are.)

1st afternoon (date, times of start and end): 13.30h - 18h

2nd afternoon (date, times of start and end): 13.30 - 17.30h

Time you needed to work on the report (at home) in person-hours, sum over all group members: 70 h

(This information does not enter into the evaluation, but it helps us to adjust the lab content.)

Here is space for your comments:

The professor helped us in the lab session. We discuss the results with him and he was very helpful. The manual has a lot of exercise compared to other lab sessions and thus the report is too extensive.

What did you like? How would you improve the labs? Would you actually like to implement these improvements yourself?

Signatures of the students, with date:

20/12/24

With your signatures you confirm that you have not used any references for your work other than those quoted in your lab report, and that you have not copied any content from other students' reports.

Evaluation of the lab report and the performance of the students during the lab (by the supervisor):

Signature of the lab supervisor, with date:

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Numerical Laboratory
Ordinary Differential Equations - Cosmological Models

Abstract. In this report, we study numerical methods for solving ordinary differential equations (ODEs). We use numerical integration techniques, such as Euler method, Runge-Kutta methods (RK4 and RK5) and stiff solvers. In particular, we analyze them in terms of accuracy, stability, and computational efficiency. We conduct a comparison of these methods through various cases, including single and coupled ODEs.
In the cosmological context, we apply these numerical methods to solve the equation that describes the evolution of the universe's scale factor $a(t)$. We study different cosmological models, by changing the values of their density parameters.
We experimentally prove the importance of method selection based on problem characteristics. We conclude that the RK5 and stiff integrators are the most adequate for handling systems with varying scales.

Jennifer Fabà-Moreno
Pablo Vega del Castillo
Group 18

Supervisor: Varun Kushwaha

December 20, 2024

1 Theoretical framework

Ordinary differential equations (ODEs) are a particular case of differential equations which describe how a function (or set of functions in the case of a system of ODEs) changes with respect to one independent variable. In the general case, an ODE takes on the following shape:

$$y^{(n)} = f(x, y, y', ..., y^{(n-1)}), \tag{1.1}$$

where $y^{(n)}$ denotes the n^{th} derivative of y with respect to x. For the purposes of this project we will focus on first order ODEs, that is:

$$y' = f(x, y). \tag{1.2}$$

In most cases, solving ODEs analytically can be challenging as exact solutions may not be found. In such cases, numerical methods come to aid. The main idea behind the numerical integrators that solve ODEs is discretizing the interval of study by either equidistant or adaptive-size steps. Then the solution is compute in these discretized points by means which depend on the solver used. In general:

$$y_{n+1} = y_n + h_n\varphi(x_n, y_n, y_{n+1}, h_n), \tag{1.3}$$

where $h_n = x_{n+1} - x_n$ is the step size, which for equidistant step size methods takes the form $(b-a)/N$ being a, b the lower and upper limits of the integration interval and N the number of divisions of the original interval. The function φ dictates how the solution for the next point is computed and is dependent of the algorithm being used.

For completeness, although we are not going to explicitly use it in the discussion of the results, we state the following theorem:

Theorem 1. ***Theorem (Picard–Lindelöf).***
Let $\alpha, \beta > 0$, $(x_0, y_0) \in \mathbb{R}^2$, *and*

$$R := \{(x, y) : |x - x_0| \leq \alpha, |y - y_0| \leq \beta\}.$$

Moreover, let $f \in C^0(R)$ *be a function with* $0 < \gamma := \max|f| < \infty$, *which satisfies a Lipschitz condition. Then, there exists exactly one function* $y \in C^1(I)$ *in* $I := [x_0 - \delta, x_0 + \delta]$, *where* $\delta := \min\{\alpha, \beta/\gamma\}$, *such that*

$$y'(x) = f(x, y(x)), \quad \forall x \in I, \quad \textit{and} \quad y(x_0) = y_0.$$

This theorem, states the existence and uniqueness of solutions for a given ODE under well-defined initial conditions. We will be implicitly using it when considering that the solution we have found for the different proposed problems is the one that we should be studying.

The different numerical integrators we are going to use and thoroughly study are:

- Single step methods.
  - **Euler method.** The simplest form φ can acquire comes from obtaining the the next solution point by evaluating the slope at the current point, that is:

    $$y_{n+1} = y_n + hf(x_n, y_n) \tag{1.4}$$

    It suffers from low accuracy and stability limitations as we will see in the tasks.
  - **Runge-Kutta methods.** The Euler method determines the solution for the next step based solely on the change in slope at the current step, whereas the RK4 method (or classic Runge–Kutta method) refines the slope by computing intermediate trial steps within consecutive points, providing a more accurate approximation of the solution.

- Adaptative step-size methods.
  We will see that step-size control is crucial for efficient integration. These methods dynamically adjust step sizes to maintain a balance between accuracy and computational cost, taking into account given tolerance levels. We use RK5 and Stiff integrators (implicit method) for such purposes, since they show robustness in handling these challenges.

We will employ these numerical integrators to solve the differential equation governing the time evolution of the scale factor $a(t)$, which represents the magnification of the spacetime metric independent of spatial coordinates. For a homogeneous and isotropic universe, the spacetime interval can be expressed as:

$$ds^2 = g_{\mu\nu}dx^\mu dx^\nu = dt^2 - a^2(t)\left[\frac{dr^2}{1-Kr^2} + r^2 d\Omega^2\right], \tag{1.5}$$

where K denotes the curvature of the universe, and $g_{\mu\nu}$ is the Friedmann-Lemaître-Robertson-Walker (FLRW) metric.

To connect spacetime geometry, encoded in the metric, with the universe's content, we use Einstein's field equations:

$$G_{\mu\nu} = \frac{8\pi G}{c^4} T_{\mu\nu}, \tag{1.6}$$

where $G_{\mu\nu}$ is the Einstein tensor derived from the metric, and $T_{\mu\nu}$ is the energy-momentum tensor. Assuming the universe's components behave as perfect fluids—characterized by their density ρ and isotropic pressure P, the energy-momentum tensor takes the form:

$$T_{\mu\nu} = \begin{pmatrix} \rho c^2 & 0 & 0 & 0 \\ 0 & -P & 0 & 0 \\ 0 & 0 & -P & 0 \\ 0 & 0 & 0 & -P \end{pmatrix}. \tag{1.7}$$

By combining Einstein's equations (1.6) with the FLRW metric (1.5), we derive the Friedmann equations, which govern the dynamics of $a(t)$:

$$\frac{\dot{a}^2}{a^2} = \frac{8\pi G}{3}\rho - \frac{Kc^2}{R^2} + \frac{\Lambda c^2}{3}, \qquad \frac{\ddot{a}}{a} = -\frac{4\pi G}{3c^2}(\rho c^2 + 3P) + \frac{\Lambda c^2}{3}, \tag{1.8}$$

where G Newtonian gravitational constant.

To model the evolution of different energy components, we combine both Friedmann equations Equation 1.8 and obtain the equation of state for both the total and each component of the energy density:

$$\dot{\rho}c^2 + 3H(t)(\rho c^2 + P) = 0, \tag{1.9}$$

where $H(t) = \frac{\dot{a}}{a}$. Every different energy form follows a different equation of state.

Now, we assume the relation $P = \omega\rho c^2$ for each component, to obtain how the density energy dilutes with the scale factor by integrating Equation 1.9. In our simplified study, we are going to assume that ω is not a function of time. Solving this for various ω gives:

- **Pressure-less matter** ($P = 0 \implies \omega = 0$):

$$\rho_m(a) = \rho_{m,0} a^{-3}, \tag{1.10}$$

  where the subscript 0 denotes the present value.

- **Radiation** ($P = \frac{1}{3}\rho c^2 \implies \omega = \frac{1}{3}$):

$$\rho_r(a) = \rho_{r,0} a^{-4}. \tag{1.11}$$

- **Cosmological constant** ($P = -\rho c^2 \implies \omega = -1$):

$$\rho_\Lambda(a) = \rho_\Lambda(0). \tag{1.12}$$

This parameter is associated with the vacuum (dark) energy of the universe.

Introducing the density parameters Ω_i, we can express the universe's evolution in terms of its energy components. These parameters are defined as:

$$\Omega_m = \frac{\rho_m}{\rho_c}, \quad \Omega_r = \frac{\rho_r}{\rho_c}, \quad \Omega_\Lambda = \frac{\rho_\Lambda}{\rho_c}, \tag{1.13}$$

where $\rho_c = \frac{3H^2}{8\pi G}$ is the critical density. We use the first Friedmann equation to derive its value. We have to assume $K = 0$, since it is defined as the energy density of a universe with $K = 0$:

$$\left(\frac{\dot{a}}{a}\right)^2 = \frac{8\pi G \rho_{cr}}{3} \iff \rho_{cr}(a) = \frac{3}{8\pi G}\left(\frac{\dot{a}}{a}\right)^2 \iff \rho_c(a) = \frac{3}{8\pi G}H^2 \tag{1.14}$$

With the value of the Hubble constant, that is, the Hubble parameter today, we can compute the value of the critical density today:

$$H_0 = 70\text{Km/s/Mpc} \implies \rho_{c,0} = 70^2 \frac{3}{8\pi \cdot G} = 2.69 \cdot 10^{41}\text{kg/Mpc} \tag{1.15}$$

where $G = 4.3 \cdot 10^{-9} \cdot$ Mpc/$M_\odot$ and we have taken into account that $M_\odot = 1.98 \cdot 10^{30}$ kg.

On the other hand, the curvature parameter is given by:

$$H_0^2 = (\Omega_{m,0} + \Omega_{r,0} + \Omega_{\Lambda,0})H_0^2 - \frac{Kc^2}{a_0^2} \implies \Omega_{K,0} = -\frac{Kc^2}{a_0^2 H_0^2} = 1 - \Omega_0, \quad \Omega_0 := \Omega_m + \Omega_r + \Omega_\Lambda. \tag{1.16}$$

Substituting these definitions into the Friedmann equation (1.8), we obtain the key differential equation for $a(t)$:

$$\left(\frac{\dot{a}}{H_0}\right)^2 = \Omega_K + \frac{\Omega_m}{a} + \frac{\Omega_r}{a^2} + a^2\Omega_\Lambda. \tag{1.17}$$

This equation will be solved numerically to explore different scenarios for the universe's evolution.

2 Tasks

2.1 Part 1. Integrator properties

2.1.1 Problem 1

Exercise 1.1.

- **Exercise 1: Euler integrator. You should complete the Euler integration routine. We will then test the function to see if it works correctly. We have already prepared the function, and defined the inputs and outputs that the function should have.**

  We complete the Euler integrator provided in the Jupyter notebook 1a. It is implemented in a way that allows to solve both a single ODE or a system of ODEs.

```
def eul(dydx,y0,a,b,n):
    """
    Integrate the system of ODEs given by y'(x,y)=dydx(x,y)
    with initial condition y(a)=y0 from a to b in n steps
    using the Euler method.

    Input:
        dydx : function to compute y'(x,y)
        y0   : 1-dim array of size m containing the initial values of y (at x0=a),
               where m is the number of differential equations in the system
        a    : start value of x
        b    : end value of x
        n    : number of steps to use for the interval [a,b]

    Output:
        x    : 1-dim array of size (n+1) containing the x values used in the
    integration
        y    : 2-dim array of size (n+1,m) containing the resulting y-values
    """
    print("Euler:")
    h = (b-a)/float(n)
    x = a + h*np.arange(n+1)
    y = np.zeros(((n+1),len(y0)))
    y[0] = y0 # assigns the entire row y[0,*]
    for i in range(n):
        y[i+1,:] = y[i,:] + h * dydx(x[i], y[i,:])
    print(": ",n,"calls of function")
    print(": ",n,"steps")
    return x,y

```

Listing 1: Euler integrator

We are now going to test this integrator working with the example 2.5 from the manual [1], that is, $y' = -sin(x)y^2$ with $y(0) = 2$.

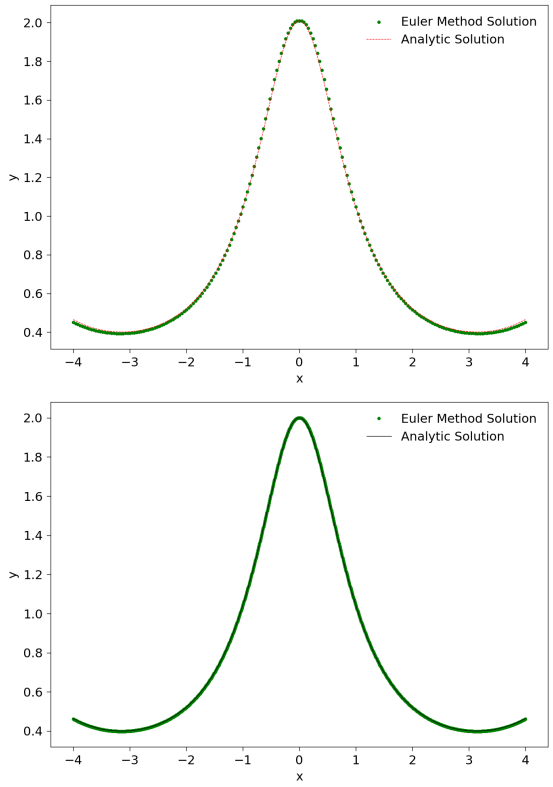

Figure 2.1: Tests of Euler integrator. Upper: $x_{lim} = 4$, n=100; lower: $x_{lim} = 4$, n=500.

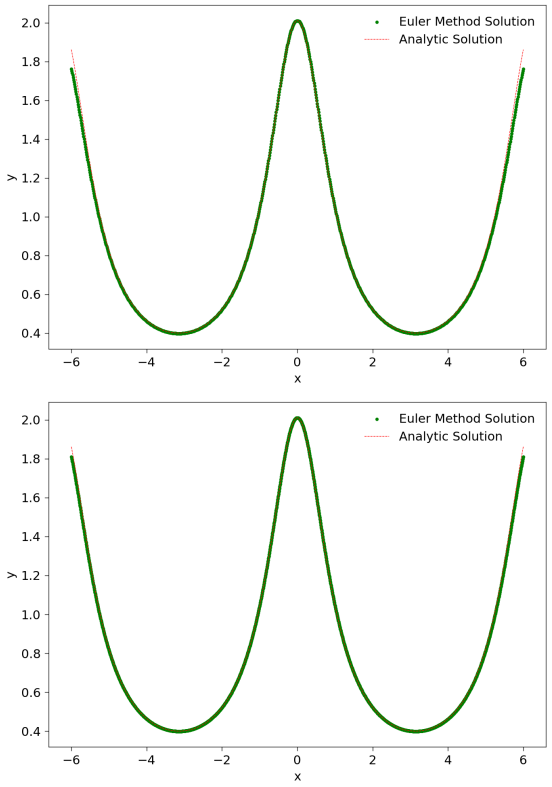

Figure 2.2: Tests of Euler integrator. Upper: $x_{lim} = 6$, n=500; lower: $x_{lim} = 5$, n=1000.

- **Exercise 2: Explore the step size, and number of steps. Modify the interesting parameters of the call to the Euler integrator. Document which are good choices to make.**

  In Figure 2.1 we show the numerical solution obtained with Euler method with different range in x and different number of steps. In the first case, we set $x_{lim} = 4$ and change the number of steps. We increase them from 100 to 500, since in the first case the endpoints are not properly approximated. We see that with the new choice of steps (n=500), the integrator is able to resolve smoothly the solution function in its different regions, including near critical points. Although now we are doing just a qualitative analysis, in the next sections we will properly use a numerical measure of error between the numerical approximations and the analytical result (when available).

  In Figure 2.2 we study the case $x_{lim} = 6$ and n=500 and n=10000. We increase the number of points following the same mindset as in the previous case, as the endpoints are again not properly covered by the Euler approximation.

  We conclude that the Euler integrator is accurate in this case but may need extra a higher number of steps than other numerical integration methods.

```
    def dydx_example_2_5(x, y):
    """
    Example 2.5 on page 12 of the ODE notes.
    This function accepts an x-value and y-value.
    """
    return -np.sin(x)*(y**2)

# The initial conditions.
x0 = 0.
```

```
y0 = [2.01]

# End of integration interval. Adjust this parameter.
x1 = 4.

# How many steps in both +x and -x directions. Adjust this parameter.
n_steps = 100

x_pos, y_pos = eul(dydx_example_2_5,y0,x0, x1,n_steps,)
x_neg, y_neg = eul(dydx_example_2_5,y0,x0,-x1,n_steps,)

# This neat trick combines two arrays:
x = np.append(np.flipud(x_neg), x_pos)
y = np.append(np.flipud(y_neg), y_pos)

# The y_analytic = results are on page 12 (2-6) of ODE pdf.
y_analytic = 1./(1.+1./y0[0]-np.cos(x))

# Let's compare the output of your Euler code with the analytic results:
plt.scatter(x, y, color = 'green', label='Euler Method Solution')
plt.plot(x, y_analytic, '--', color='red', label='Analytic Solution')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()

```

Listing 2: Test for Euler integrator. Code for Exercise 2.

- **Exercise 3: As a first exercise, we would now like to solve a first-order differential equation defined by:**

$$\partial_x y(x) = a y(x), \tag{2.1}$$

with $a = 2.5$ and $y(0) = 0.001$.

  - Write down your analytic solution. Format the solution nicely, and show your steps.
  - Then code the ODE $\partial_x y(x)$, and the exact solution function.
  - Read and understand the given code. Add additional comments to make the code easier to read. Then execute it, and explore the resulting plots. Compare the results when using 100 and 1000 steps.

In order to solve this ODE analytically, we propose an ansatz of the form e^{mx}. Substituting it into the ODE results in:

$$m e^{mx} = a e^{mx},$$

which means $m = a$ and so the solution has the following form: $y(x) = C e^{ax}$. To determine the value of the constant C, we use the condition given $y(0) = 0.001$, which gives the final solution:

$$y(x) = y(0) e^{ax} = 0.001 e^{2.5x}.$$

This ODE is also solvable by separation of variables and posterior integration.

Once we know the expression of the analytic solution, we proceed to solve it numerically. To do so, we code $\partial_x y(x)$, and the exact solution function.

```
a = 2.5
y0 = 0.001

def p1_dydx(x,y):
    return a*y
#Analytic solution:
```

```
def p1_analytic(x):
    return y0*np.exp(a*x)

```

Listing 3: $\partial_x y(x)$ and exact solution function.

Now we continue plotting the solutions obtained in the integration interval $[0, 10]$.

```
#Interval limits
p1_a = 0. #left limit
p1_b = 10. #right limit

#initial value for y
p1_y0 = [1e-3]

# Run the integration with 100 and 1000 steps.
p11_x_eul_100, p11_y_eul_100 = eul(p1_dydx,p1_y0,p1_a,p1_b,100)
p11_eul_100_label = 'Euler {0},{1}'.format(100,100)

p11_y_eul_exa_100 = p1_analytic(p11_x_eul_100)
p11_eul_exa_100_label = 'Analytic solution'

p11_x_eul_1000,p11_y_eul_1000 = eul(p1_dydx,p1_y0,p1_a,p1_b,1000)
p11_eul_1000_label = 'Euler {0},{1}'.format(1000,1000)

p11_y_eul_exa_1000 = p1_analytic(p11_x_eul_1000)
p11_eul_exa_1000_label = 'Analytic solution'

plt.figure()
plt.plot(p11_x_eul_1000, p11_y_eul_1000[:,0], 'co', label=p11_eul_1000_label)
plt.plot(p11_x_eul_100,  p11_y_eul_100 [:,0], 'mo', label=p11_eul_100_label)
plt.plot(p11_x_eul_1000, p11_y_eul_exa_1000, 'k-',  label=p11_eul_exa_1000_label)
plt.legend(loc='upper left')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
plt.close()

plt.figure()
plt.plot(p11_x_eul_1000, p11_y_eul_1000[:,0], 'co', label=p11_eul_1000_label)
plt.plot(p11_x_eul_100,  p11_y_eul_100 [:,0], 'mo', label=p11_eul_100_label)
plt.plot(p11_x_eul_1000, p11_y_eul_exa_1000, 'k-',  label=p11_eul_exa_1000_label)
plt.legend(loc='upper left')
plt.xlabel('x')
plt.ylabel('log(y)')
plt.yscale('log')
plt.show()
plt.close()
```

Listing 4: Plotting solutions of Equation 2.1 obtained with the Euler integrator.

Figure 2.3 shows the numerical solutions obtained with both 100 and 1000 steps as well as the analytic solution. We can see how the numerical solution with 1000 steps is a better approximation to the exact solution than the one with 100 steps. In particular we see how the solutions diverge when the slope of the exponential becomes steep. Although neither of the two choices is able to follow the exponential growth of the analytic solution, 1000 steps achieves a better result than 100 steps.

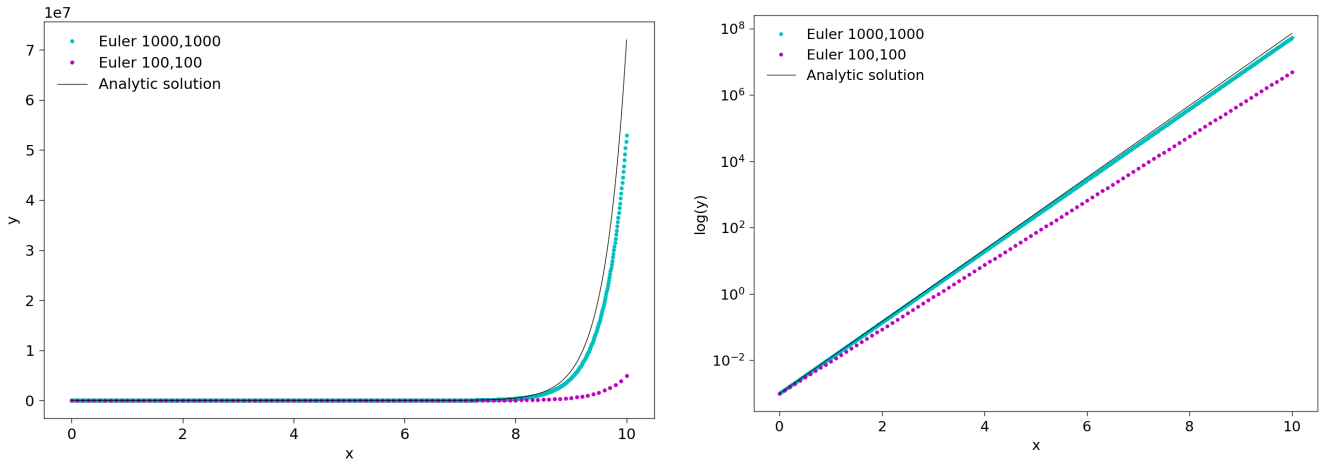

Figure 2.3: Numerical solution of Equation 2.1 with the Euler method using 100 and 1000 steps. Left: linear scaling in both axis. Right: y axis in log scale.

- **Exercise 4: Global discretization error. Let us now determine the "global discretization error" as given by equations 2.10 through 2.12 on page 2-3 of the PDF manual. Compute the maximum deviation and print this to the screen. Now turn the computation into a function that accepts two inputs, y_1 and y_2, and returns the maximum value of the global discretization error. Document the function and the code.**

  We first implement a method to obtain the global discretization errror which outputs the maximum error obtained within the integration interval.

```
def global_err(y_approx,y_exact):
    """
    Inputs:
    y_approx: numerical solution
    y_exact: analytical solution
    Output: maximum of the difference y_exact - y_approx
    """
    return np.max(y_exact - y_approx)

print("Global error from Euler with  100 steps:",global_err(p11_y_eul_100[:,0] ,
    p11_y_eul_exa_100 ))
print("Global error from Euler with 1000 steps:",global_err(p11_y_eul_1000[:,0],
    p11_y_eul_exa_1000))
```

Listing 5: Global discretization error function.

We apply this function to our previous solutions of Equation 2.1 with 100 and 1000 steps to get: The

Number of steps	Global error
100	$6.71 \cdot 10^7$
1000	$1.91 \cdot 10^7$

Table 2.1: Global error for Euler integrator and Equation 2.1

errors on Table 2.1 are both considerably large and this could be expected as seen from Figure 2.3. The solution function grows exponentially meaning that at some point its slope becomes really steep and the global error keeps accumulating. Also, as expected, the numerical solution with 1000 steps achieves less error compared to the one with 100 steps.

Problem 1.2

Runge-Kutta integrator

Exercise 2 (manual). Construct the tableau for the Collatz method.

The Collatz method is a 2nd order Runge-Kutta method which proceeds as follows:

$$k_1 = f(x_n, y_n), \tag{2.2}$$

$$k_2 = f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right), \tag{2.3}$$

$$y_{n+1} = y_n + hk_2, \tag{2.4}$$

Thus the tableau has the following shape:

0		
$\frac{1}{2}$	$\frac{1}{2}$	
	0	1

- **Exercise 5: Call the 4th-order Runge-Kutta code. How does the accuracy of RK4 compare with the Euler function that you wrote?**

  As the RK4 integrator was already implemented we have included its code in Appendix A.
  In Figure 2.6 we have plotted the numerical solutions of Equation 2.1 using the RK4 integrator with both 100 and 1000 steps as well as the exact solution. By simple inspection of this figure and Figure 2.3 we can see how the RK4 numerical solutions follow the exact solution more precisely than the Euler solutions. This is expected according to theory (): while Euler method had consistency $p = 1$, RK4 achieves a consistency of $p = 4$. Precisely, the global discretization error in Euler was $O(h^1)$ whereas for RK4 this error is reduced to $O(h^4)$. We have fitted to a linear function the relation of h with the global relative error of the predictions for each method. We expect a slope of 1 for the Euler method and 4 for the RK4, since we are considering a logarithmic scale. PONER LSTLISTING

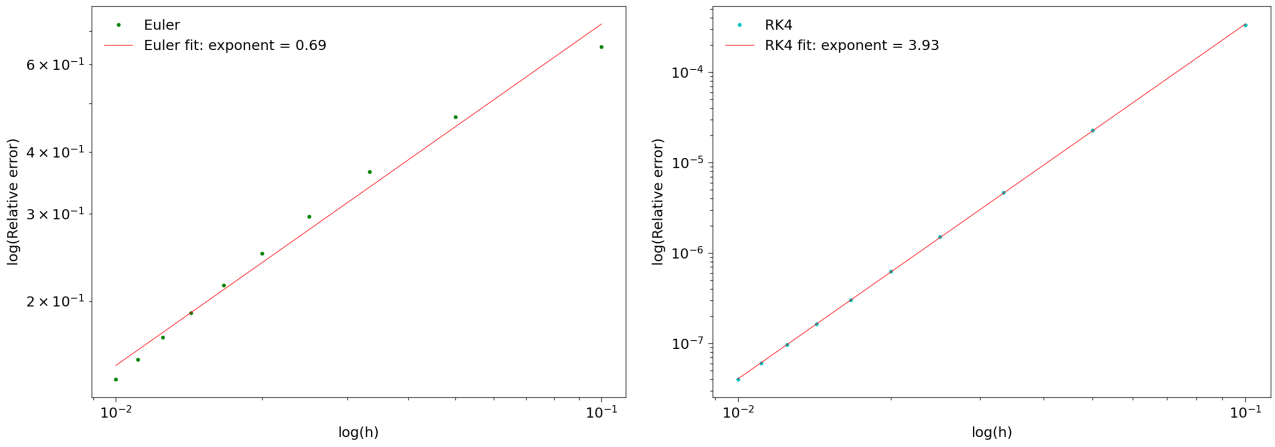

Figure 2.4: Left: Relative error dependency with h for Euler. Left: Relative error dependency with h for RK4.

In Figure 2.4, we can find the adjusted slopes. For the Euler case, we have a relative error of 0.31 and for the RK4 one of 0.02. In this case, we set the threshold for considering the slopes to adjust

Method	Slope of fit
Euler	0.69
RK4	3.93

Table 2.2: Fit slopes for Euler and RK4 step sizes to relative errors.

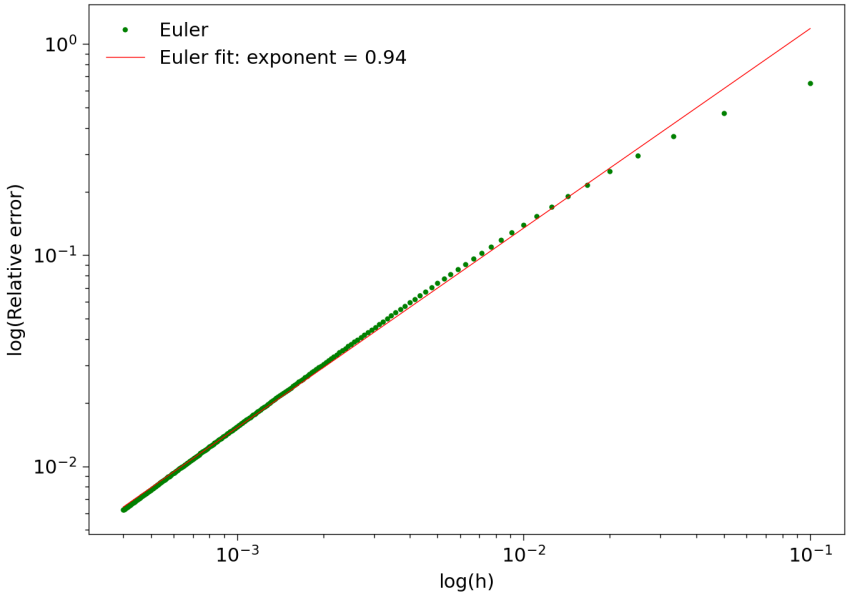

Figure 2.5: Relative error dependency with h for Euler ($n = 25000$).

the expect value to be 0.05. Thus, the RK4 presents the expected expression for the order of the global error with respect to the step size but the Euler does not. We increase the number of steps (i.e, reduce the step size) to see if the tendency of the slope approaches 1. In Figure 2.5, we can see that the slope tends to 1 when increasing the number of steps. It is still not below or threshold, but we suppose that its behavior with increasing n will lead to slope of 1. We do not test higher number of steps due to the computational time required.

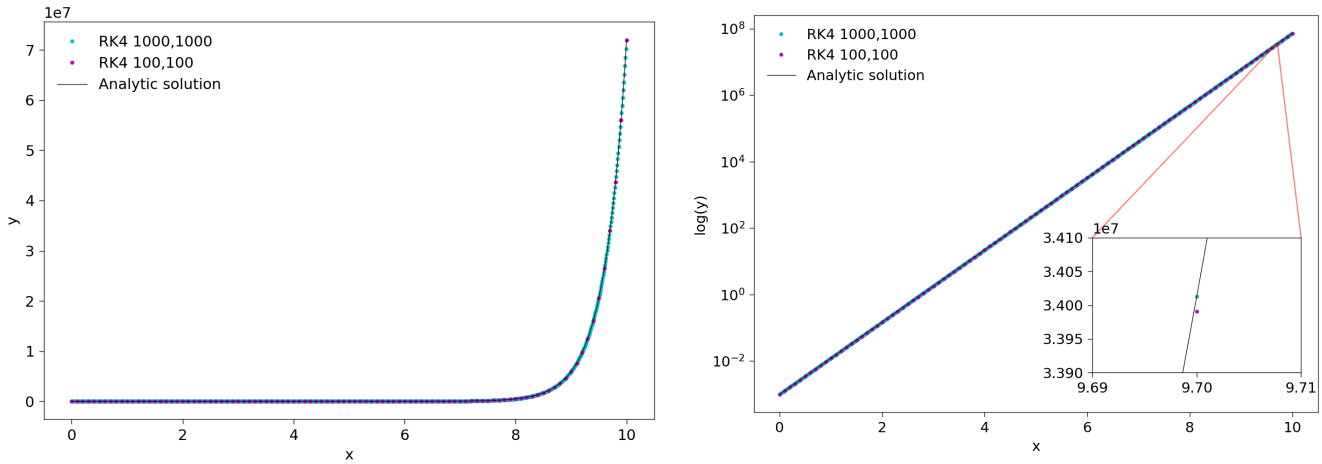

Figure 2.6: Numerical solution of Equation 2.1 with the RK4 method using 100 and 1000 steps. Left: linear scaling in both axis. Right: y axis in log scale. A zoomed in image has been added to compare the accuracy with respect to the choice in the number of steps.

Another way of ensure the better performance of the RK4 integrator is by looking at the global

discretization error as we did with Euler method. For RK4 we obtain: While Euler method with 1000

Number of steps	Global error
100	$4.76 \cdot 10^4$
1000	5.74

Table 2.3: Global error for RK4 integrator and Equation 2.1

steps resulted in a maximum global discretization error of $O(10^7)$, RK4 with 1000 steps has been able to reduce the maximum error to something of order below $O(10^1)$ (see Table 2.3). Furthermore, even with 100 steps, RK4 achieves a much more accurate result than Euler method with 1000 steps.

- **Exercise 6: Plot the relative discretization error of both methods. In this exercise you should calculate (and plot as a function of x) the relative difference between the approximate numeric solutions and the analytic solution. Why is it often more meaningful to look at the relative errors rather than the absolute errors?**

We first construct a new method to compute the relative error as shown below.

```
def rel_err(y_approx,y_exact):
    """
    Inputs:
    y_approx: numerical solution
    y_exact: analytical solution
    Output: relative error between numerical and analytical solutions
    """
    return (y_exact-y_approx) / y_exact
```

Listing 6: Relative error function.

Now we can plot the relative error of both methods, Euler and RK4, and for both 100 and 1000 steps.

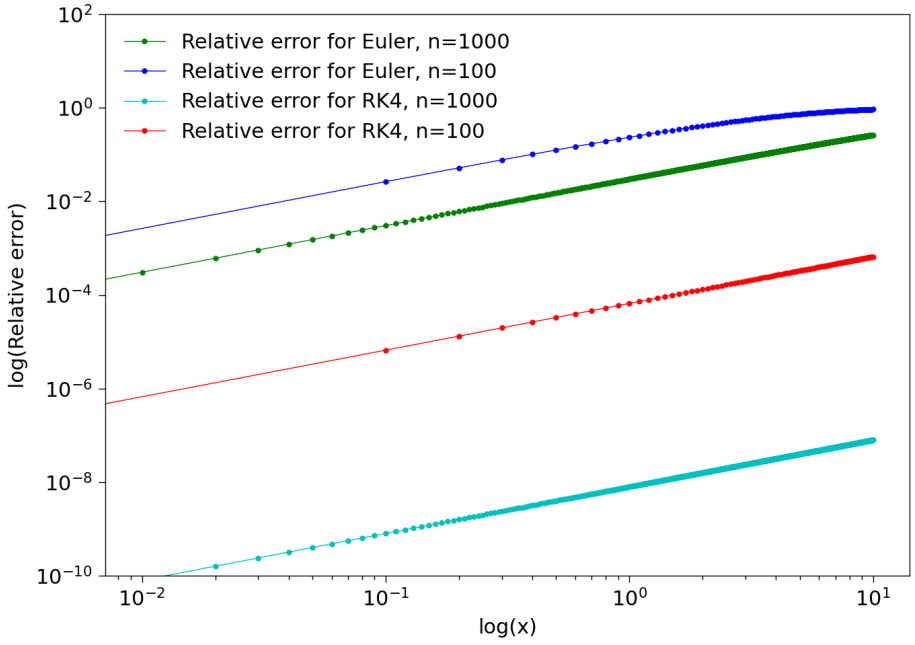

Figure 2.7: Relative errors of Euler and RK4 methods with 100 and 1000 steps.

From Figure 2.7 we can see how the relative errors increase along the integration interval as errors accumulate. This graph also shows how the RK4 integrator performs better than the Euler integrator with either 100 or 1000 steps. We can also observe how increasing from 100 to 1000 steps makes

a more significant improvement in accuracy for the RK4 integrator than it does for the Euler method.

For our study case, Equation 2.1, looking at the relative error is a more useful measure of the accuracy than the global error. This is because the relative error eliminates the dependence of the absolute error on the size of the function itself. In this case, as the exponential solution outputs values of great size, the relative error allows us to look at the differences between the numerical and the exact values ignoring the actual size of the values.

Problem 1.3

Other integrators (provided by the SciPy library, RKDP5 and STIFF). We have implemented these as classes in order to encapsulate some internally used variables.

- **Exercise 7: Call the adaptive-stepsize Runge-Kutta and the "stiff" integrators. Repeat the previous exercise with the adaptive-stepsize Runge-Kutta and the "stiff" integrators. How do they compare to the results from the RK4 integrator using 1000 steps? Note that you also need to define a function that returns the Jacobian of the system for use by the "stiff" integrator.**

We first construct the Jacobian of the right hand side of Equation 2.1 for the Stiff integrator.

```
def p1_jac(x,y):
    jac = np.array([a])
    return jac
```

Listing 7: Jacobian of right hand side of Equation 2.1.

We now use the RK5 and the Stiff integrators to obtain their numerical solutions of Equation 2.1. The default tolerance of 10^{-6} was used in both methods. The displayed information regarding the performance of both methods is showed in Table 2.4. While the Stiff method computes a greater number of steps than the RK5, the latter requires more function calls. We have to remember that RK5 computes the intermediate k values in each step, which involves calling the function.

Method	Calls of function	Total steps
RK5	655	110
Stiff	292	291

Table 2.4: Performance of the RK5 and Stiff integrators when solving Equation 2.1.

The numerical solutions from RK5 and Stiff are then plotted along with the analytical one in Figure 2.8.

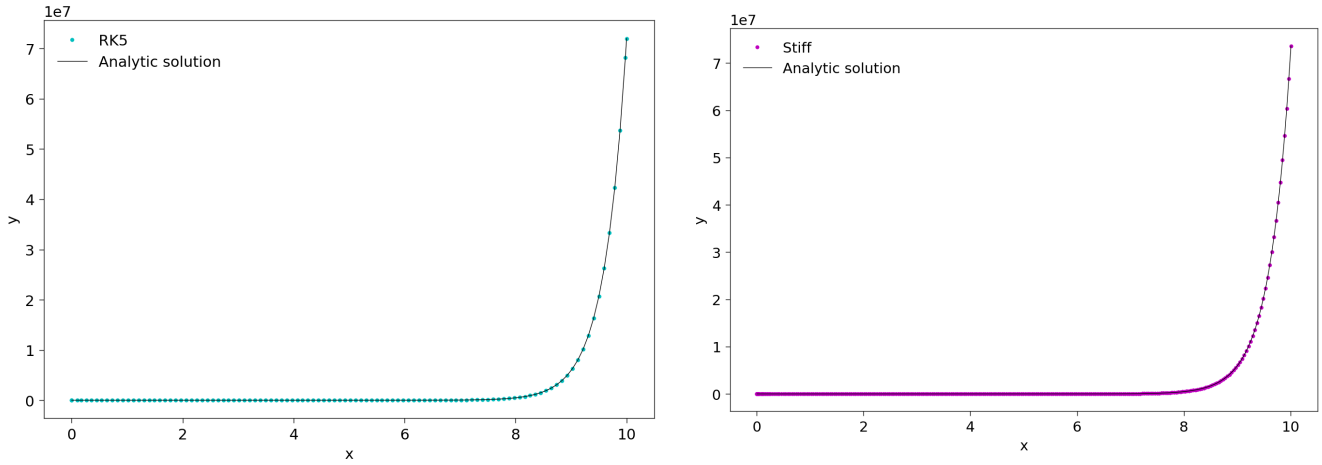

Figure 2.8: Numerical solution of Equation 2.1 with the RK5 (left) and Stiff (right) methods.

To compare with the previous integrators, Euler and RK4, we plot the relative errors of all of them in Figure 2.9.

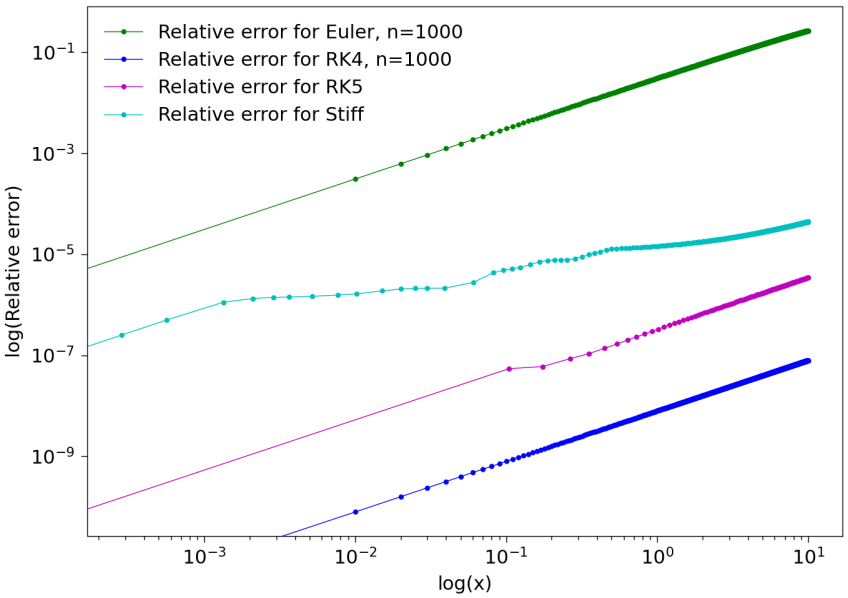

Figure 2.9: Relative errors of the four studied ODE solvers applied to Equation 2.1. Euler and RK4 use 1000 discretization steps while Stiff and RK5 are set at tol= 10^{-6}.

RK4 commits the least relative error among the four methods and the adaptive-stepsize methods, Stiff and RK5, perform better than the Euler method. Although one could think that RK4 is a better method than both adaptive-stepsize methods, we have to keep in mind that RK5 and Stiff are using 110 and 291 steps respectively, as shown in Table 2.4, while RK4 is set to 1000 steps.

Problem 1.4

- **Exercise 8: Accuracy of the other integrators. By varying the tolerance parameter, find the number of steps the adaptive-stepsize integrators need to obtain the same accuracy as the RK4 integrator with 1000 steps. How does the requested tolerance compare with the actually achieved accuracy? How does the accuracy of the adaptive-stepsize integrators compare to that of RK4 if the same number of steps is used?**

To define the accuracy of the integrators we are going to use the mean of the relative errors obtained in the integration interval. This way RK4 with 1000 steps results in a mean relative error of $\sim 4 \cdot 10^{-8}$. To achieve this level of accuracy, we find out that RK5 has to be set to a tolerance of 10^{-8}, resulting in a mean relative error of $\sim 2 \cdot 10^{-8}$, and Stiff requires a tolerance of 10^{-10}, resulting in a mean relative error of $\sim 8 \cdot 10^{-9}$. These results along with the number of steps required by each method are summarized in Table 2.5.

Method	Tolerance	Calls of function	Total steps	Relative error
RK5	10^{-8}	1669	279	$2 \cdot 10^{-8}$
Stiff	10^{-10}	1278	1277	$8 \cdot 10^{-9}$

Table 2.5: Performance of the RK5 and Stiff integrators.

We can see how the order of magnitude of the tolerance matches that of the relative error for the RK5 while for the Stiff method the relative error appears to be one around one order of magnitude higher than the tolerance. To get a clearer result we are going to look at the plot of the relative error vs. the tolerance for both methods when solving Equation 2.1. As we can see from Figure 2.10, the relative error of the RK5 method follows a power-law relationship with the tolerance in a way that the relative error has the same order of magnitude of the predefined tolerance. On the other hand, the Stiff method does not appear to follow such a clear power-law relationship. Furthermore, we see how the orders of magnitude of the relative error and the tolerance do not match as in the RK5 as, in fact, the relative error is typically between one and two orders of magnitude above the order of magnitude of the tolerance.

As RK4 with 1000 steps resulted in a mean relative error of $\sim 4 \cdot 10^{-8}$, we can state, from the data of Table 2.5, that, for the same number of steps, RK5 will achieve better accuracy than RK4. In fact, setting the tolerance of RK5 in $2 \cdot 10^{-11}$ leads to 975 total steps and a mean relative error of $\sim 4 \cdot 10^{-11}$. We conclude then that for the same number of steps RK5 performs better than RK4. In the case of Stiff, setting the tolerance to $2 \cdot 10^{-10}$ leads to 1006 total steps and a mean relative error of $\sim 3 \cdot 10^{-8}$ which is very similar to the value obtain from RK4.

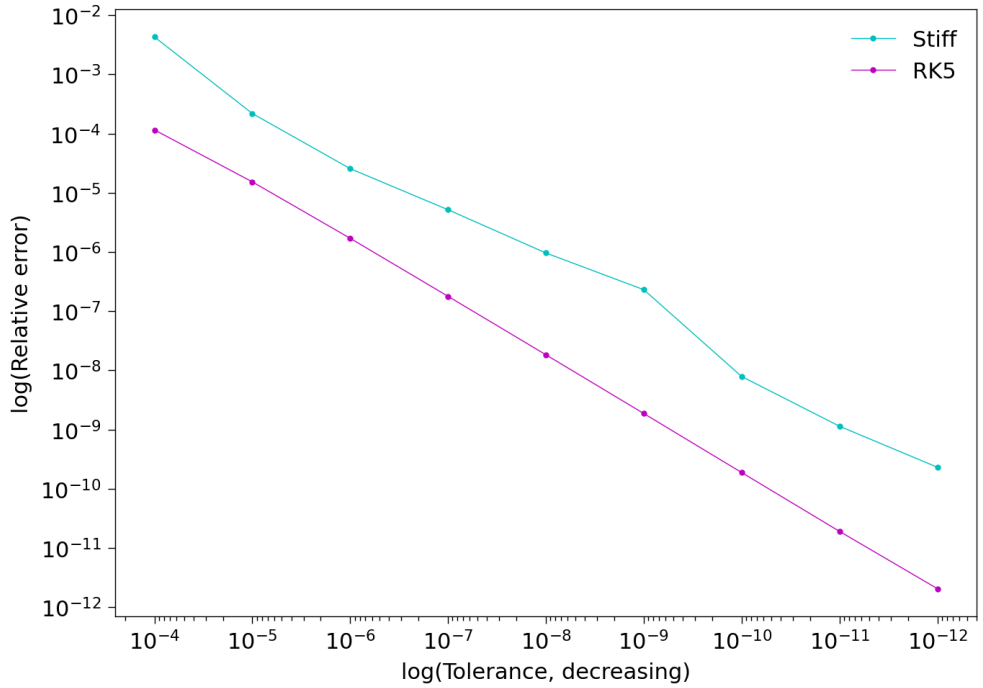

Figure 2.10: Mean relative error of RK5 and Stiff methods for decreasing tolerance when applied to Equation 2.1.

Problem 1.5

Summarize the results you obtained for Problem 1. Which integrator would you trust most, and why?

The relative errors obtained from the different integrators, are summarized in Table 2.6. We see how, for almost the same number of steps, RK5 is able to achieve the least relative error by a margin of around three orders of magnitude with respect to RK4 and Stiff methods. This makes RK5 the most reliable integrator for this particular problem. On the other hand, Euler method performs very poorly compared to the rest of the solvers. One could try to generalize this result, but, as we will see in the following sections, for certain ODEs known as *stiff* problems, the Stiff method might be able to outperform the RK5.

Although this has not been studied here, the computational cost of each method is also an important factor to take into consideration when choosing a method. From Table 2.6 we how due RK5 requires the most calls of function and this fact could lead to longer times of computation. Overall, the behavior of the equation to be solved, the required precision, the computational resources are all factors that play important roles in the selection of the solver.

Method	Tolerance	Calls of function	Total steps	Relative error
Euler	-	1000	1000	$1 \cdot 10^{-1}$
RK4	-	4000	1000	$4 \cdot 10{-}8$
RK5	$2 \cdot 10^{-11}$	5845	975	$4 \cdot 10^{-11}$
Stiff	$2 \cdot 10^{-10}$	1007	1006	$3 \cdot 10^{-8}$

Table 2.6: Summary of the relative errors from the different integrators when applied to Equation 2.1.

2.1.2 Problem 2. Coupled differential equations

We can also solve coupled differential equations, such as:

$$\begin{cases} \partial_x y_1 = 998 y_1 + 1998 y_2 \\ \partial_x y_2 = -999 y_1 - 1999 y_2 \end{cases}, \tag{2.5}$$

with initial conditions $y_1(0) = 1$ and $y_2(0) = 0$. To solve such a system with our above solvers, we must simply provide a 'dydx'-function that returns a vector of values, one element for each equation in the system. Similarly, we must provide a vector of initial values.

In this case, we are going to take advantage of the definition of the different integrators, where we have considered the possibility of having a system of differential equations.

Exercise 9 (Advanced)

Solve the above problem analytically. (Hint: as an intermediate result, you should find the eigenvalues of $\mathbf{A}$ as $\lambda_{1,2} = (-1, -1000)$).

We start by expressing Equation 2.5:

$$\dot{\mathbf{y}} = A\mathbf{y} \qquad \text{where } A = \begin{pmatrix} 998 & 1998 \\ -999 & -1999 \end{pmatrix} \text{ ,} \mathbf{y} = (y_1, y_2)^T \text{ and } \mathbf{y}_0 = (1, 0). \tag{2.6}$$

Thus, we can obtain the expression of $\mathbf{y}(x)$ using simple algebraic operations. We consider the expression of $\mathbf{y}$ in term of its modes:

$$\mathbf{y}(x) = e^{\lambda_1 x}\mathbf{v_1} + e^{\lambda_2 x}\mathbf{v_2}, \tag{2.7}$$

where $\lambda_{1,2}$ are the eigenvalues of A and $\mathbf{v}_{1,2}$ the corresponding eigenvectors. We can check that this ansatz satisfies the differential equation:

$$\dot{\mathbf{y}}(x) = \sum_i \lambda_i e^{\lambda_i x}\mathbf{v}_i \qquad \text{and} \qquad A\mathbf{y} = A\left(\sum_i e^{\lambda_i x}\mathbf{v}_i\right) = \sum_i e^{\lambda_i x} A\mathbf{v}_i \tag{2.8}$$

Using $A\mathbf{v}_i = \lambda_i \mathbf{v}_i$, we find:

$$A\mathbf{y}(X) = \sum_i e^{\lambda_i X}\lambda_i \mathbf{v}_i. \tag{2.9}$$

Both sides of $\dot{\mathbf{y}}(x) = A\mathbf{y}(x)$ match, confirming that the proposed solution satisfies the differential equation.

Thus, we continue with solving the system of differential equations:

$$\begin{pmatrix} 998-\lambda & 1998 \\ -999 & -1999-\lambda \end{pmatrix} \implies |A-\lambda Id| = (998-\lambda)(-1999-\lambda)+999\cdot 1998 = 0 \implies \lambda = \begin{cases} -1 \\ -1000 \end{cases} \tag{2.10}$$

We define $\lambda_1 = -1, \lambda_2 = -1000$ and we find the respective eigenvectors:

$$(A-\lambda_1 Id)\mathbf{v_1} = 0 \iff \begin{pmatrix} 999 & 1998 \\ -999 & -1998 \end{pmatrix}\begin{pmatrix} v_{1,1} \\ v_{1,2} \end{pmatrix} = \mathbf{0} \implies \mathbf{v_1} = (2,-1)^T \tag{2.11}$$

$$(A-\lambda_2 Id)\mathbf{v_2} = 0 \iff \begin{pmatrix} 1998 & 1998 \\ -999 & -999 \end{pmatrix}\begin{pmatrix} v_{2,1} \\ v_{2,2} \end{pmatrix} = \mathbf{0} \implies \mathbf{v_2} = (-1,1)^T \tag{2.12}$$

where we have taken into account the initial values for each component when choosing the values of $\mathbf{v_i}$. Finally, the analytic solution is:

$$\begin{cases} y_1(x) = 2e^{-x} - e^{-1000x} \\ y_2(x) = -e^{-x} + e^{-1000x} \end{cases} \tag{2.13}$$

In Equation 2.13, we see that both components of the solution are governed by the term e^{-x} (since we are studying positive x), so we expect an exponential decay for relative large x. However, although it is much smaller, the term e^{-1000x} introduces small fluctuations in the analytical function that might affect the accuracy of the numerical approximations. Let us study these numerical solutions.

Problem 2.1

Exercise 10. Solve the above system over the interval $[0,4]$ with all 4 integrators. For the fixed-step integrators use 10000 steps, for the others a tolerance of 10^{-5}. Plot and compare the relative errors as well. How many steps do RKDP5 and stiff need?

We reuse the already defined integrators for the previous exercise, but we need to define the function we have on the right hand side of the differential equation as well as the analytic solution and the jacobian of the new system, for the stiff integrator.

```
#parameters
p2_a = 0.
p2_b = 4.
p2_y0 = [1., 0.]



#functions
def p2_analytic(x):
    return np.array([2*np.exp(-x) - np.exp(-1000*x), -np.exp(-x) + np.exp(-1000*x)])

def p2_dydx(x,y):
    return np.array( [998*y[0] + 1998*y[1], -999*y[0] - 1999*y[1]])

def p2_jac(x,y):0
    return np.array([[998, 1998], [-999, -1999]])



#integrators
#euler
p21_x_eul_10000, p21_y_eul_10000 = eul(p2_dydx,p2_y0,p2_a,p2_b,10000)
p21_eul_10000_label = 'RK4 {0},{1}'.format(10000,10000)

p21_y_eul_exa_10000 = p2_analytic(p21_x_eul_10000)
p21_eul_exa_10000_label = 'Analytic solution'
```

```
27
28 rel_eul = rel_err(p21_y_eul_10000, p21_y_eul_exa_10000.T)
29
30
31 #rk4
32 p21_x_rk4_10000, p21_y_rk4_10000 = rk4(p2_dydx,p2_y0,p2_a,p2_b,10000)
33 p21_rk4_10000_label = 'RK4 {0},{1}'.format(10000,10000)
34
35 p21_y_rk4_exa_10000 = p2_analytic(p21_x_rk4_10000)
36
37 rel_rk4 = rel_err(p21_y_rk4_10000, p21_y_rk4_exa_10000.T)
38
39
40 #rk5
41 rk5_instance = RKDP5(p2_dydx, p2_y0, p2_a, p2_b, tol=1e-5)
42 rk5_instance.integrate()
43 x_rk5, result_rk5, _, _, _, _ = rk5_instance.result()
44
45 p21_y_rk5_exa = p2_analytic(x_rk5)
46
47 rel_rk5 = rel_err(result_rk5, p2_analytic(x_rk5).T)
48
49
50 #stiff
51 stiff_instance = STIFF(p2_dydx, p2_jac, p2_y0, p2_a, p2_b, tol=1e-5)
52 stiff_instance.integrate()
53 x_stiff, result_stiff, _, _, _, _ = stiff_instance.result()
54
55 p21_y_stiff_exa = p2_analytic(x_stiff)
56
57 rel_stiff = rel_err(result_stiff, p2_analytic(x_stiff).T)
58
59
60 #plots
61 #first component of y
62 plt.figure()
63 plt.plot(p21_x_eul_10000, p21_y_eul_10000[:,0], 'go', label=p21_eul_10000_label)
64 plt.plot(p21_x_rk4_10000, p21_y_rk4_10000[:,0], 'bo', label=p21_rk4_10000_label)
65 plt.plot(x_rk5, result_rk5[:,0], 'co', label='p12_rk5')
66 plt.plot(x_stiff, result_stiff[:,0], 'mo', label='p12_stiff')
67
68 plt.plot(p21_x_eul_10000, p21_y_eul_exa_10000.T[:,0], 'k-',  label=
       p21_eul_exa_10000_label)
69
70 plt.legend(loc='upper right')
71 plt.xlabel('x')
72 plt.ylabel('y_1')
73 plt.show()
74 plt.close()
75
76
77 plt.figure()
78 plt.plot(p21_x_eul_10000, np.abs(rel_eul[:,0]), 'g-o', label='Rel error euler')
79 plt.plot(p21_x_rk4_10000, np.abs(rel_rk4[:,0]), 'b-o', label='Rel error rk4')
80 plt.plot(x_stiff, np.abs(rel_stiff[:,0]), 'c-o', label='Rel error stiff')
81 plt.plot(x_rk5, np.abs(rel_rk5[:,0]), 'm-o', label='Rel error rk5')
82 plt.legend(loc='upper left')
83 plt.xlabel('log(x)')
84 plt.ylabel('log(Rel error)')
85 plt.xlim((1e-10, 4))
86 plt.xscale('log')
87 plt.yscale('log')
88 plt.show()
```

```
89 plt.close()
```

Listing 8: Different integrators for solving Equation 2.5.

Regarding the code for plotting the second component of $\mathbf{y}$, we use the same code but changing 0 to a 1 in the component of the vectors. Thus, we do not copy again the code here. We use the same number of steps for both the Euler and the RK4 method and the same tolerance for RK5 and Stiff integrators ($n = 10000$, $tol = 10^{-5}$).

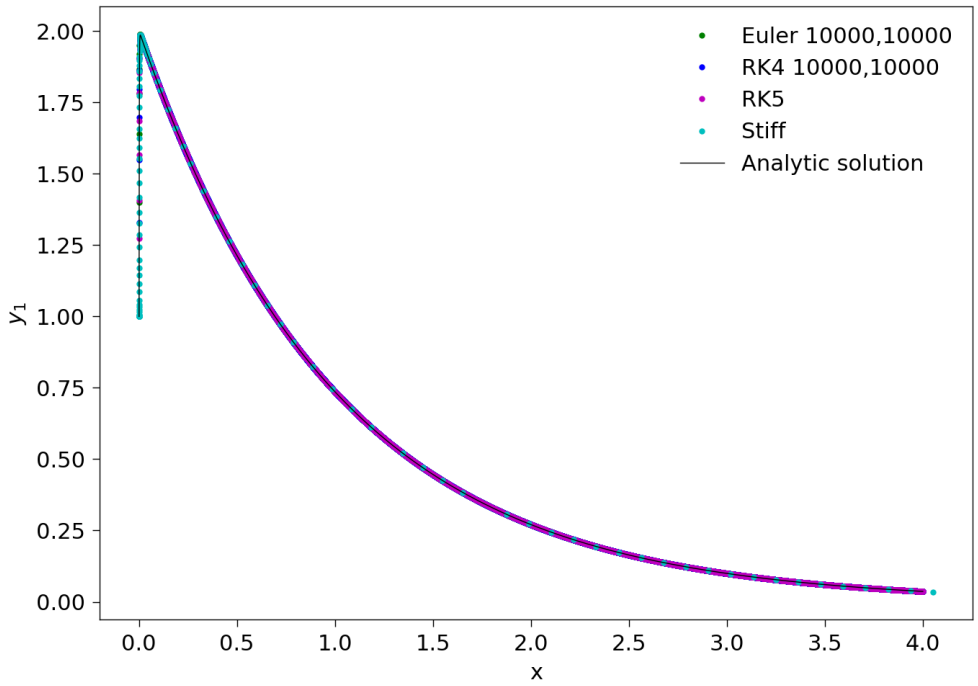

Figure 2.11: Numerical and analytical solution for Equation 2.5 $y_1(x)$.

In Figure 2.11 (Figure 2.14) [1], we see that, qualitatively, the numerical solutions of all the integrators are accurate approximations of the analytical expression of y_1 (y_2). They have been able to mimic the peak at low x as well as the exponential decay (increase) for larger ones; they all tend to 0 when the analytical solution does. With the given tolerance, RK5 method has needed 1233 steps and the Stiff one only 166. We see that both number are lower than 10000, the steps done by Euler and RK4. This implies a great reduction, which is related to the type of step used in integration. The first two methods, the Euler and the RK4, are fixed-step integrators, whereas the latter ones, RK5 and stiff, are adaptive-step ones. This is explicitly shown for the stiff integrator Figure B.2, where we can observe that when the approximation is in the smooth part of the function, the stiff integrator chooses to use less points than when it is on the steep part (lower x). Thus, it has adapted the number of steps to balance the accuracy of the results and the computational time/memory required, choosing more points where the behavior is volatile and less when it encounters a more stable region.

[1] Refer to Appendix B for seeing the individual approximations for each method, since here we show all them together.

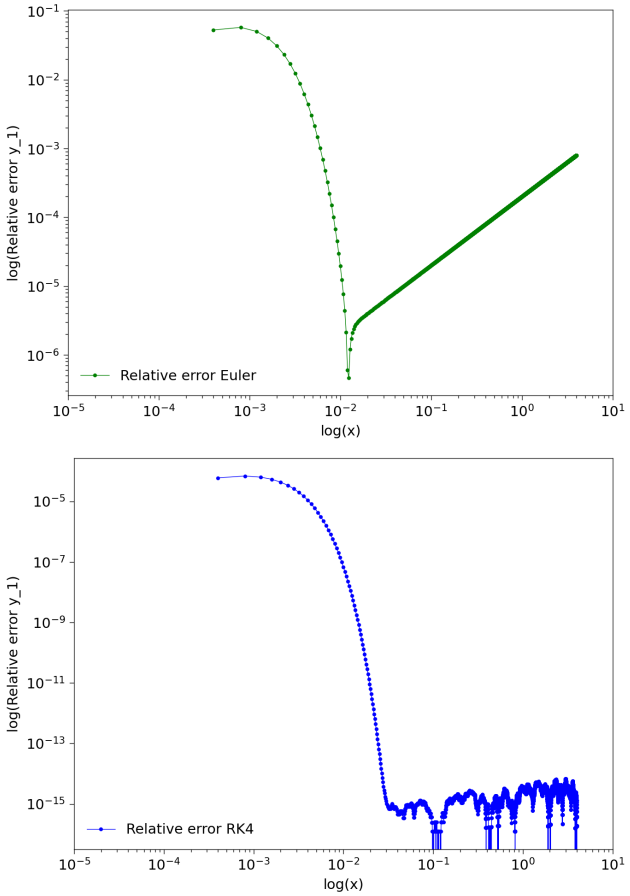

Figure 2.12: Euler's (upper) and RK4's (lower) relative error for Equation 2.5 $y_1(x)$. Logarithmic scale.

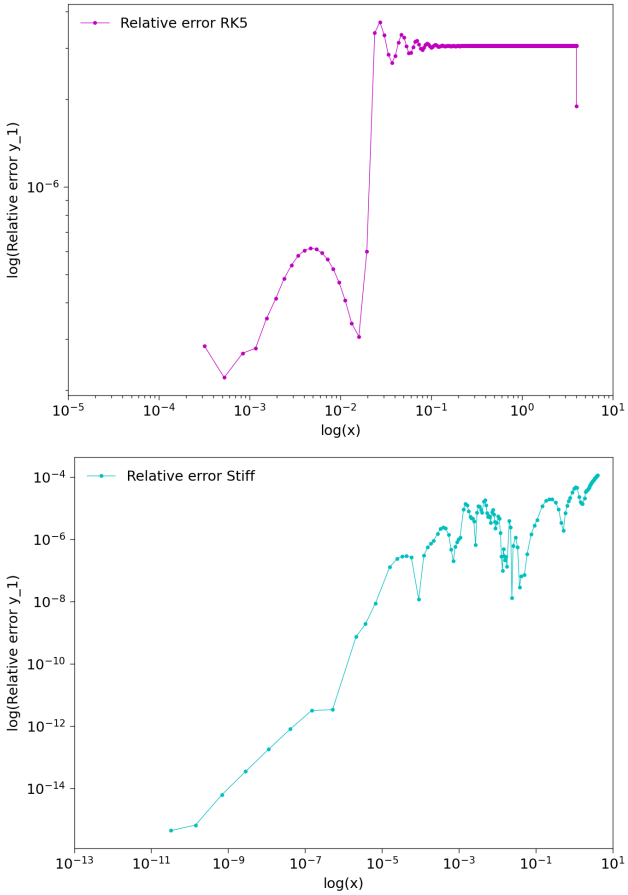

Figure 2.13: RK5's (upper) and Stiff's (lower) relative error for Equation 2.5 $y_1(x)$. Logarithmic scale.

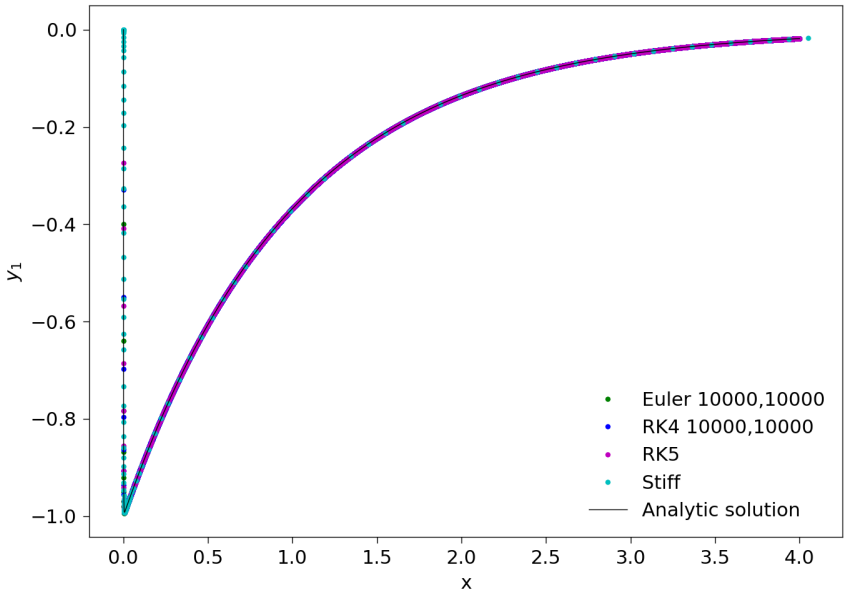

Figure 2.14: Numerical and analytical solution for Equation 2.5 $y_2(x)$.

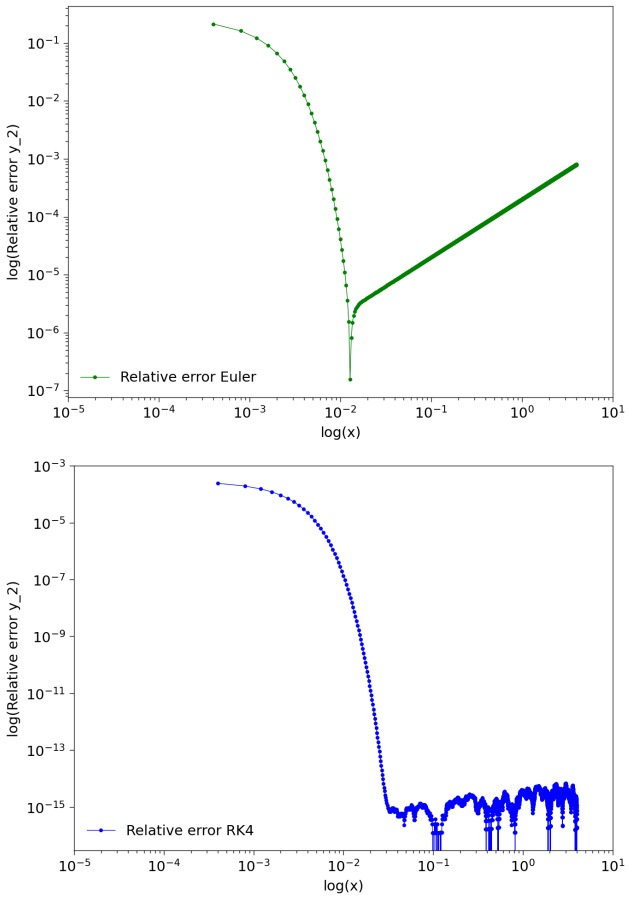

Figure 2.15: Euler's (upper) and RK4's (lower) relative error for Equation 2.5 $y_2(x)$. Logarithmic scale.

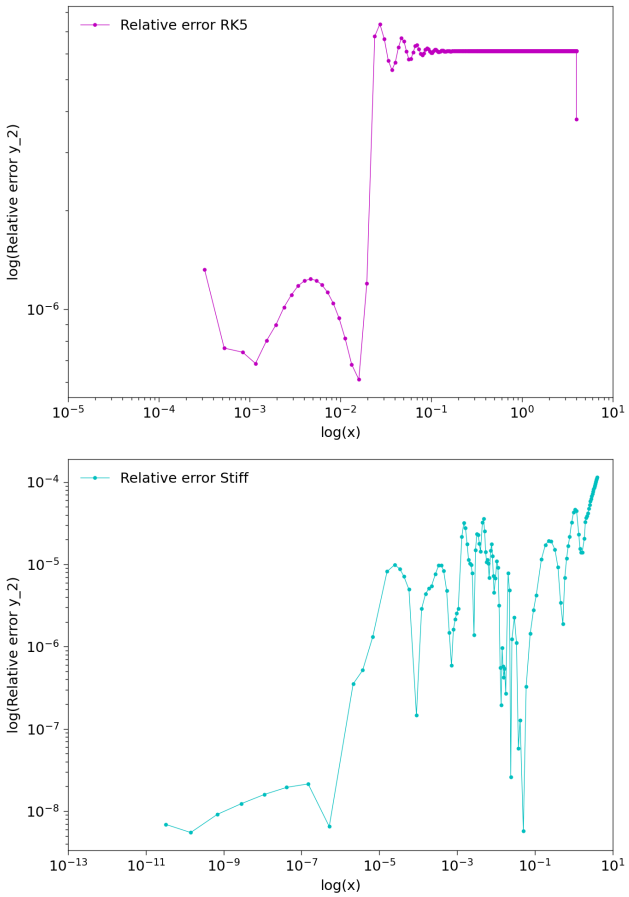

Figure 2.16: RK5's (upper) and Stiff's (lower) relative error for Equation 2.5 $y_2(x)$. Logarithmic scale.

On the other hand, in both Figure 2.13 and Figure 2.16, we can check on the errors of the 4 methods. We have used the logarithmic scale to be able to properly see the error evolution with x (refer to Figure B.5 to see the plot without log-scale). For both components we see that the relative errors are small and, unexpectedly, the RK4 method is the one with less relative error for x near 4. Furthermore, we see that it oscillates around the machine ϵ (the 0 of the float type). We might explain this behavior considering the rounding errors. As for large x the accuracy of the approximations is better, the rounding errors dominate since the discretization error is diminished. Not only we have to take into account the increase of precision but also the fact that the function we are approximating is nearly 0. Thus, the intermediate steps accumulate rounding errors due to the precision of the machine. After all, as we are working with small step sizes, the discretization error is negligible compared to the floating-point round-off error and this introduces random (since are tied to the precision limit of the computer) small oscillations in the result.

The question now should be: why do the other methods do not show the same behavior if they are also really accurate in the approximations? Let us give a brief comment on each of the other methods about why we consider that their behavior is different:

- Euler: as its discretization errors are larger than the ones of RK4 ($O(h^2)$ vs $O(h^4)$), they might overcome the rounding errors making the result less sensitive to them. When increasing x, the error accumulate given we are considering the global error.

- RK5 and stiff: as they both use adaptive-step integration, they might be more effective overcoming these rounding errors. However, we cannot provide a reason of why they have larger relative error than the RK4 method. In fact, this might not have a numerical explanation beyond how the

integrators are defined to solve the differential equations and the rounding errors and precision limit of the computer. On the other hand, we see that in the left part on the interval, small x, they outperform the other methods and we explain this because this region presents a steep change which this adaptive size methods better approximate. After this, the errors again increases due to the fact that we are considering global errors.

Problem 2.2 - Integrator stability

Exercise 11. Test the stability conditions derived in Section 2.5 of the PDF manual, both for the Euler integrator and for RK4. What maximum stepsize is allowed in each case? (Eigenvalues found in Exercise 9). How does this transform into a step number? (Allow for 3 additional steps accounting for rounding errors). Check the solution with the corresponding (minimum) step-numbers, particularly whether the numerical results decay to the analytic solution. Plot the corresponding numerical solutions in comparison to the exact one. Reduce the step number by ten and watch what happens.

We start by demonstrating the stability condition for the Euler method given in the manual:

$$h_{\max} < \frac{2}{\max_i c_i + \frac{\tau_i^2}{c_i}} \tag{2.14}$$

where h is the stepsize and $\lambda_i = -c_i + i\tau - i$ where $c_i > 0$ [1]. Recall λ_i are the eigenvalues of A.

Following the procedure of the manual, we have that:

$$\mathbf{y}_{n+1} = (Id + Ah)\mathbf{y}_n = C\mathbf{y}_n \tag{2.15}$$

As we are considering solution functions that exponentially decrease with x, in order to have an stable solution for a relative high number of steps n, we need $C^n \to 0$ and this happens only if the absolutely largest eigenvalue $|\lambda_{\max}| < 1$. This is because $C^n v = \lambda^n v$ and we want $C^n \to 0\ \forall v$, so $\lambda^n \to 0$ and this happens if and only if $|\lambda_i| < 1$ for all i. In particular, we can ask this condition for the maximum of them to have a sufficient condition.

The eigenvalues of C are of the form: $\lambda_{C,i} = 1 - c_i h + i\tau_i h$, given its relation with A: $C\mathbf{v} =_i (Id + Ah)\mathbf{v}_i = (1 + \lambda_i h)\mathbf{v}_i$. The condition for the absolute value reads:

$$(1 - c_i h)^2 + (\tau_i h)^2 = c_i^2 h^2 - 2c_i h + 1 + \tau_i^2 h^2 < 1 \implies c_i h \left[\left(c_i + \frac{\tau_i^2}{c_i} \right) h - 2 \right] < 0 \tag{2.16}$$

and as $c_i, h > 0$:

$$\max_i \left(c_i + \frac{\tau_i^2}{c_i} \right) h_{\max} - 2 < 0 \implies h_{\max} < \frac{2}{\max_i c_i + \frac{\tau_i^2}{c_i}} \tag{2.17}$$

Now, we can check which value of $h_{\max}$ saturates this condition in our case where $|\lambda_{\max}| = 1000$:

$$h_{\max} < 2 \cdot 10^{-3} \implies n_{min} > \frac{4}{2 \cdot 10^{-3}} = 2 \cdot 10^3 \tag{2.18}$$

where we have taken into account the definition of $n = \frac{(b-a)}{h}$. Thus, the maximum step size allowed is $2 \cdot 10^{-3}$ and the minimum number of steps required is $2 \cdot 10^3$. As suggested in the question, we set n to be the minimum required plus 3, so as to account for the rounding errors.

```
#integrators
x_eul, y_mat_eul = eul(p2_dydx,p2_y0,p2_a,p2_b,2003)


```

```
#plots
plt.figure()
plt.plot(x_eul,y_mat_eul[:,0], 'c-o', label='y1 euler')
plt.plot(x_eul, p2_analytic(x_eul)[0], 'k-',  label='y1')
plt.legend(loc='upper left')
plt.xlabel('x')
plt.ylabel('y_1')
plt.show()
plt.close()
```

Listing 9: Integrator stability code.

In Figure 2.17, we show the predictions and the original functions for both components of the solution of Equation 2.5. In both cases we observe the same behavior: the approximation begins oscillating around the analytical solution and converges to it at the end of the interval, hence it captures the exponential tendency to 0 of the analytical function. One has to take into account that we are using the minimum number of steps, this is the maximum space between steps, that make the solution stable. Indeed, have checked that they are stable, so now let us show that this is effectively the minimum n that provides this condition.

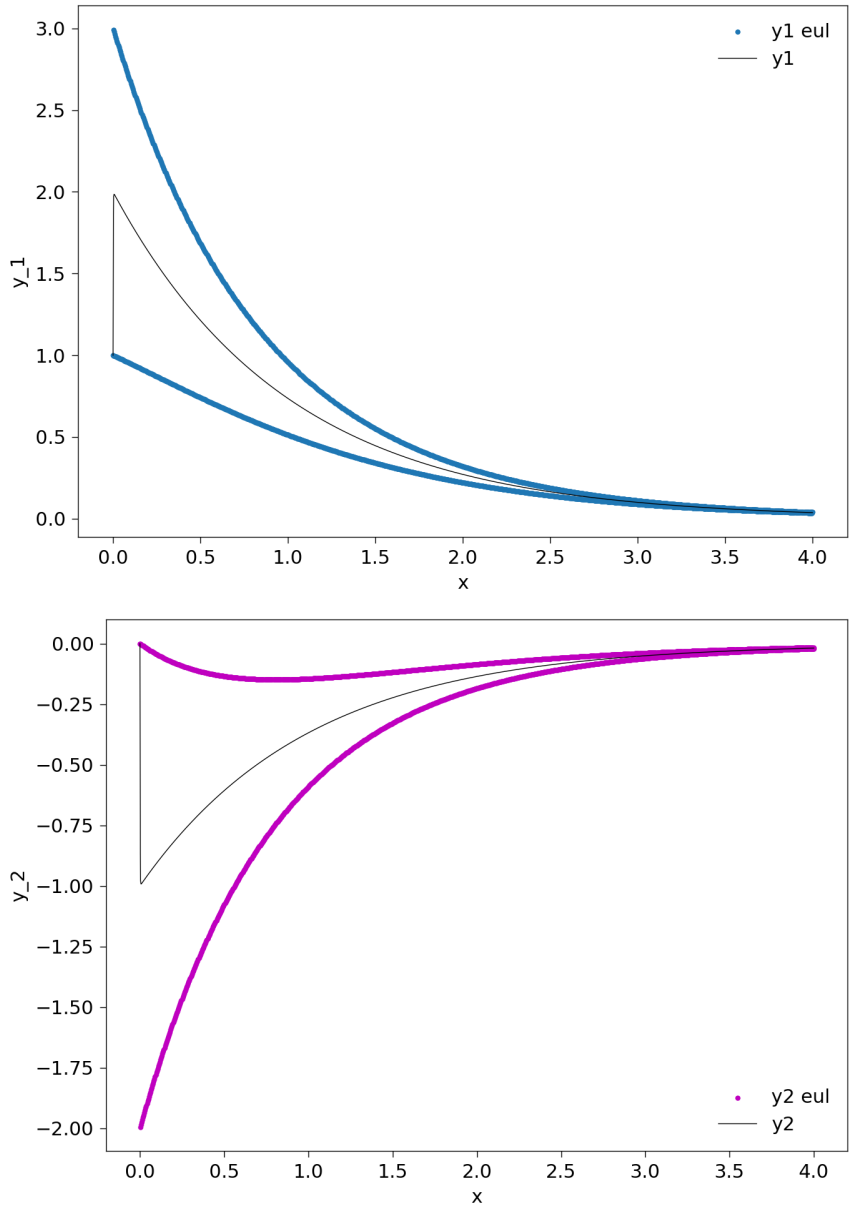

Figure 2.17: Euler approximation for Equation 2.5 $y_1(x)$ (upper) and $y_2(x)$ (lower) with $n = 2003$.

In Figure 2.18, we have built the Euler integrator with $n = 1999$ (1 step below the minimum n that makes the approximation stable [2]). We can see that the solution is not stable for it differs from the analytical solution with increasing x. Thus, we have numerically tested the stability condition mathematically proved. In this case, $h = 4/1999 = 0.002001$ which is greater than the value of h given at Equation 2.18. We note that the question asked to perform the approximation with 10 step less than the minimum, but in order to assess that this exact number is indeed the minimum, we have decided to test the integrator with just one step less. In fact, if the approximation does not converge to the analytical solution with $n = n_{\text{min}} - 1$, it will also fail with $n = n_{\text{min}} - 10$.

On the other hand, we assess the accuracy of the integrators by increasing the number of steps to $n = 4000$ (see Figure 2.19). In this case, we do not observe the previous initial variations on the approximations with respect to the analytical solution and see how the former adjust the latter perfectly.

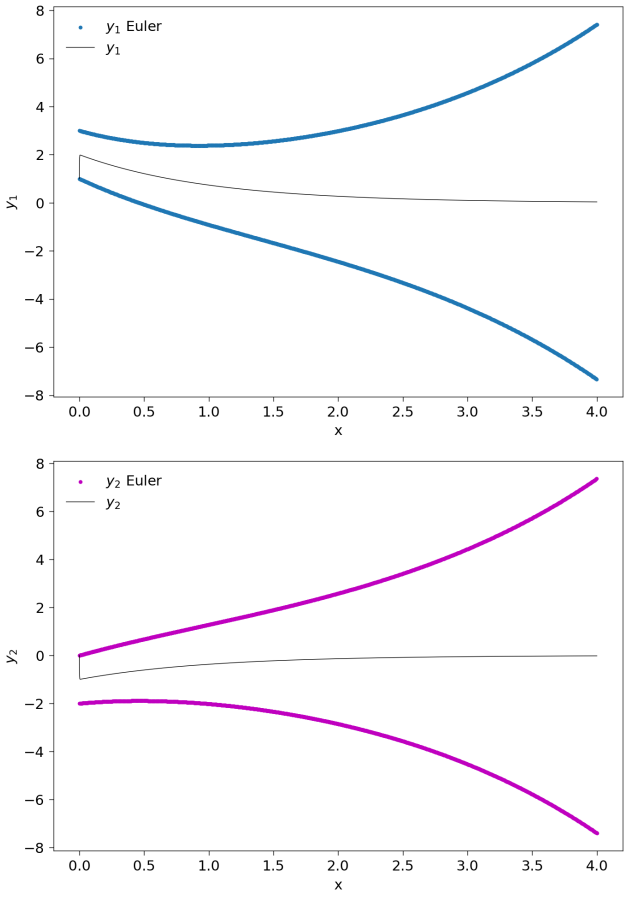

Figure 2.18: Euler approximation for Equation 2.5 $y_1(x)$ (upper) and $y_2(x)$ (lower) with $n = 1999$.

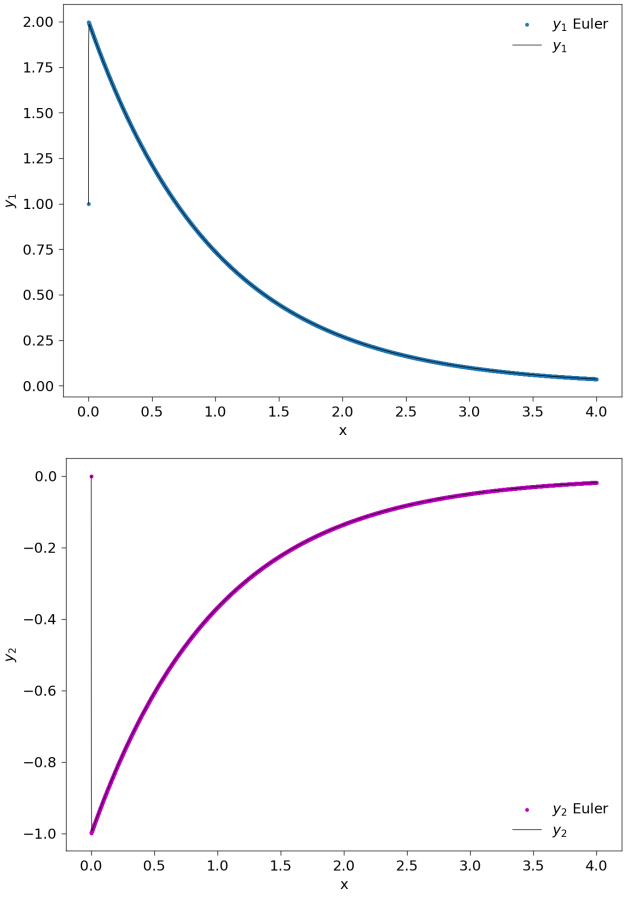

Figure 2.19: Euler approximation for Equation 2.5 $y_1(x)$ (upper) and $y_2(x)$ (lower) with $n = 4000$.

Now, we are going to check the stability condition for the RK4 method. In this case, the absolute stability region is given by:

$$|F(\mu)| < 1 \text{ where } \mu = \lambda h \tag{2.19}$$

where λ are the eigenvalues associated to the system of differential equations, $F(x) = \sum_{i=1}^{4} \frac{x^i}{i!}$ and h is the step size. Recall that in our case, $\lambda = -1, -1000$.

[2] Without taking into account the 3 additional steps for rounding errors.

Explicitly, the condition reads:

$$-1 < 1 + \lambda h + \frac{\lambda^2 h^2}{2} + \frac{\lambda^3 h^3}{6} + \frac{\lambda^4 h^4}{24} < 1 \tag{2.20}$$

Let us see which condition arise from each inequality:

$$1 + \lambda h + \frac{\lambda^2 h^2}{2} + \frac{\lambda^3 h^3}{6} + \frac{\lambda^4 h^4}{24} < 1 \implies \lambda h \left(1 + \frac{\lambda h}{2} + \frac{\lambda^2 h^2}{6} + \frac{\lambda^3 h^3}{6}\right) < 0 \qquad \text{Condition 1} \tag{2.21}$$

$$-1 < 1 + \lambda h + \frac{\lambda^2 h^2}{2} + \frac{\lambda^3 h^3}{6} + \frac{\lambda^4 h^4}{24} \implies 0 < 2 + \lambda h + \frac{\lambda^2 h^2}{2} + \frac{\lambda^3 h^3}{6} + \frac{\lambda^4 h^4}{24} \qquad \text{Condition 2} \tag{2.22}$$

We need to evaluate these conditions in each λ_i $i = 1, 2$ and consider the intersection of the allowed intervals for h (as we want to obtain an h_{max} for both solutions y_1, y_2)[3]. From this, we will obtain the maximum value of h that makes F lie in the region of absolute stability.

From Equation 2.21, we have need $\left(1 + \frac{\lambda h}{2} + \frac{\lambda^2 h^2}{6} + \frac{\lambda^3 h^3}{6}\right) > 0$, since $\lambda_i < 0$ for $i = 1, 2$. Thus, we have to find the zeros of this function as well as the ones of Equation 2.22:

$$1 + \frac{\lambda h}{2} + \frac{\lambda^2 h^2}{6} + \frac{\lambda^3 h^3}{6} = 0 \implies \begin{cases} h_1 = 2.78529 \\ h_{1000} = 0.0027853 \end{cases} \qquad 2 + \lambda h + \frac{\lambda^2 h^2}{2} + \frac{\lambda^3 h^3}{6} + \frac{\lambda^4 h^4}{24} = 0 \tag{2.23}$$

where the sub-indices for the h denotes in which λ_i we have evaluated the function before solving it. In the case of the second equation, we find that it has no real solution and it is always positive with regardless of the value of h, so it does not provide any constraint on its value. We support our discussion with Figure 2.20.

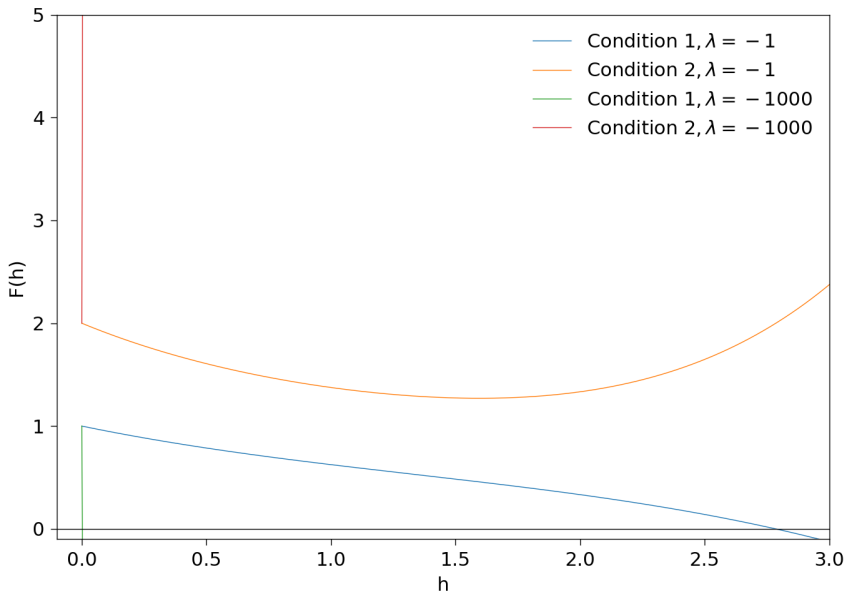

Figure 2.20: $F(\lambda h)$ for $\lambda = \lambda_{1,2}$.

We see how condition 2 is always satisfied for any value of h and condition 1 only for $h \in (0, 0.0027853)$. Thus, $h_{\text{max}} = 0.0027853$. This corresponds to $n_{\text{min}} = 1437$. Let us now, check this condition numerically, where again we allow for 3 additional steps for rounding errors.

In Figure 2.21, we see that although the approximation does not adjust the analytical function for small x, after some steps it has perfectly captured the analytical tendency to 0. As previously, we are going

[3] If we were interested in obtaining one for each component, we could independently consider the conditions for each λ_i.

to check whether it is the real minimum number of steps. By looking at Figure 2.22, where we have considered $n = n_{\text{min}} - 1$, we see that the approximation for both components does not fit the analytical solutions at all. It starts on the correct point (by definition), but after this it deviates from the expected tendency with x. Finally, we perform the RK4 approximation with $n = 1700$ and we see that it accurately reproduces the analytical solution.

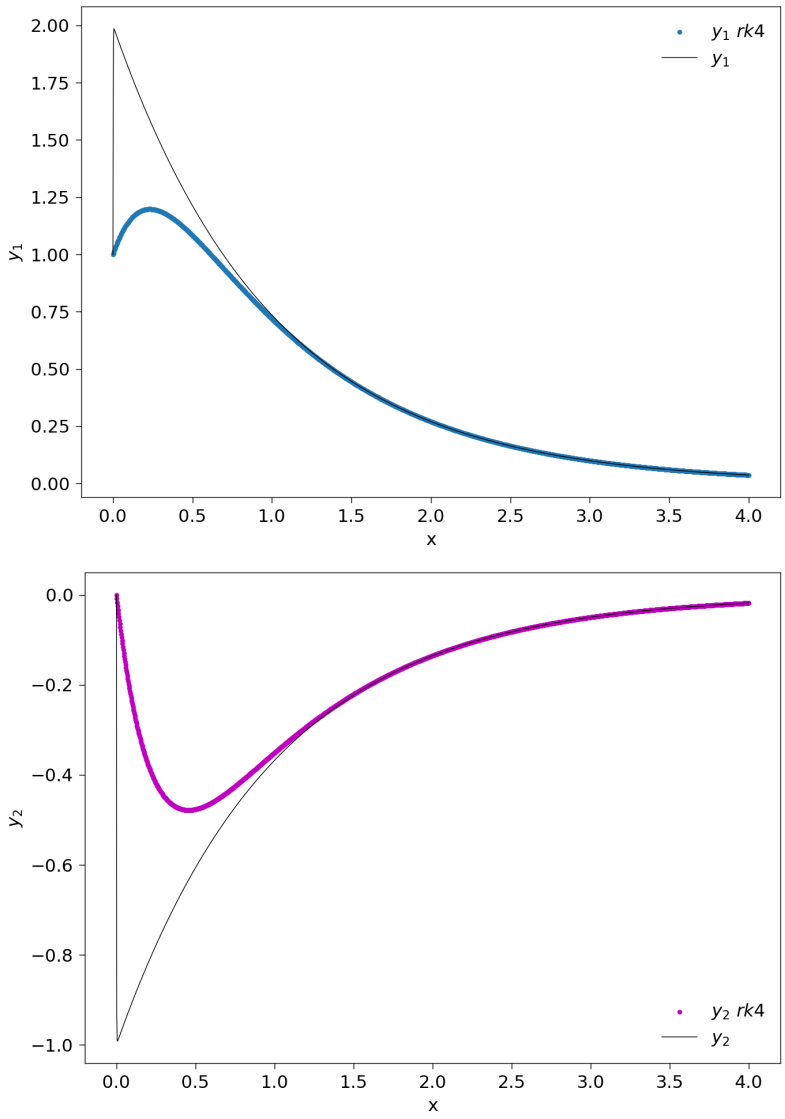

Figure 2.21: RK4 approximation for Equation 2.5 $y_1(x)$ (upper) and $y_2(x)$ (lower) with $n = 1440$.

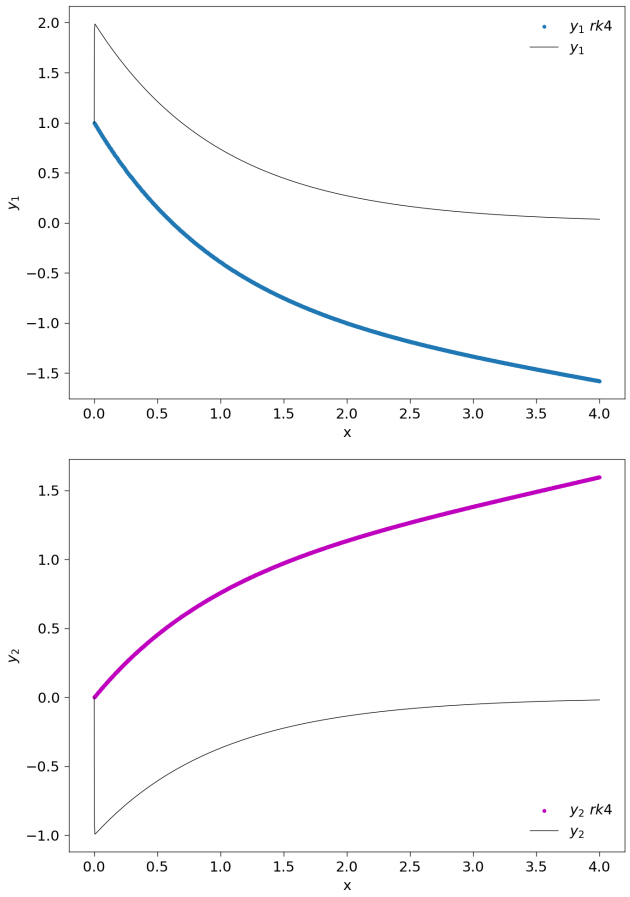

Figure 2.22: RK4 approximation for Equation 2.5 $y_1(x)$ (upper) and $y_2(x)$ (lower) with $n = 1436$.

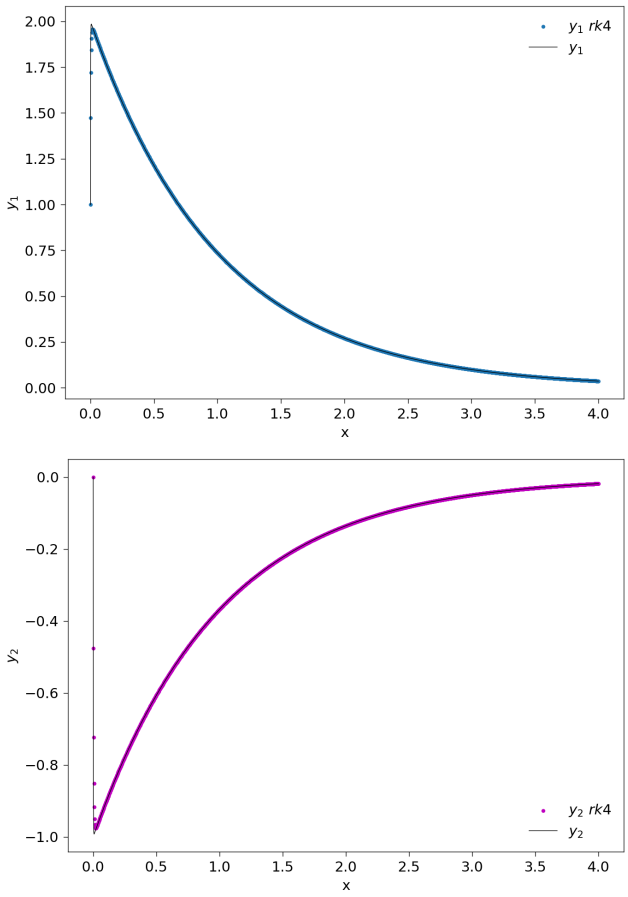

Figure 2.23: RK4 approximation for Equation 2.5 $y_1(x)$ (upper) and $y_2(x)$ (lower) with $n = 1700$.

2.1.3 Problem 3 - Three coupled ODEs

- **Exercise 12. Write down the corresponding set of ODEs. Construct the corresponding Jacobian as well. Discuss why this is a difficult problem to integrate.**

  The system of three coupled ODEs reads as follows:

$$\begin{cases} \partial_x y_1 = -0.013 y_1 - 1000 y_1 y_3 \\ \partial_x y_2 = -2500 y_2 y_3 \\ \partial_x y_3 = -0.013 y_1 - 1000 y_1 y_3 - 2500 y_2 y_3, \end{cases} \tag{2.24}$$

  and the corresponding Jacobian is:

$$J = \begin{pmatrix} -0.013 - 1000 y_3 & 0 & -1000 y_1 \\ 0 & -2500 y_3 & -2500 y_2 \\ -0.013 - 1000 y_3 & -2500 y_3 & -1000 y_1 - 2500 y_2 \end{pmatrix}. \tag{2.25}$$

  Unlike the previous coupled differential equations (Equation 2.5), this system contains non-linear terms of the form $y_1 y_3$ and $y_2 y_3$ and the Jacobian is dependent on the functions y_i themselves, meaning that we are not able to follow the procedure used for Equation 2.5.

- **Exercise 13. Write a corresponding Jacobian routine for use with the stiff integrator. Then run the integrators RKDP5 and stiff over the interval [0,50] and plot the results. Do the results agree? How many iterations does each of the integrators need?**

We implement a function to calculate the Jacobian for this problem as follows below.

```
def p3_jac(x,y):
    return np.array([[-0.013-1000.*y[2], 0, -1000.*y[0]],
                     [0, -2500.*y[2], -2500.*y[1]],
                     [-0.013-1000.*y[2], -2500.*y[2], -1000.*y[0]-2500.*y[1]]])
```

Listing 10: Jacobian corresponding to Equation 2.24.

Now we can call the RK5 and Stiff integrators to obtain the numerical solutions $y_i(x)$. These solutions are plotted in Figure 2.24. We can see how the solutions agree between both methods. However, it can be clearly seen how the RK5 solutions are being evaluated at much more points than the Stiff ones. To get a better idea of what is going on we can look now at the number of steps needed by each integrator in Table 2.7.

Method	Tolerance	Calls of function	Total steps
RK5	10^{-6}	346639	57774
Stiff	10^{-6}	176	176

Table 2.7: Performance of the RK5 and Stiff integrators.

Explicitly, from Table 2.7, RK5 is using many more steps than the Stiff integrator. Given that we do not have an analytical solution for this system of ODEs we can not obtain neither absolute nor relative errors. Furthermore, given that the number of steps of each method is different and that the solutions are evaluated in different x values we can not compute the difference between the $y_i(x)$ of each method to obtain an absolute error between them.

If we look at the coefficients that appear in Equation 2.24 we notice these vary several orders of magnitude and this could be a clear indicator that we are facing a *stiff* kind of problem. Indeed, looking at the lower plot of Figure 2.24 we see how the y_3 function is on the order of magnitude of (10^{-6}) which is significantly smaller than the orders of magnitudes of y_1 and y_2 (10^0). Thus, being a *stiff* problem, the Stiff solver is a better option as it needs many less steps than RK5 for the same tolerance.

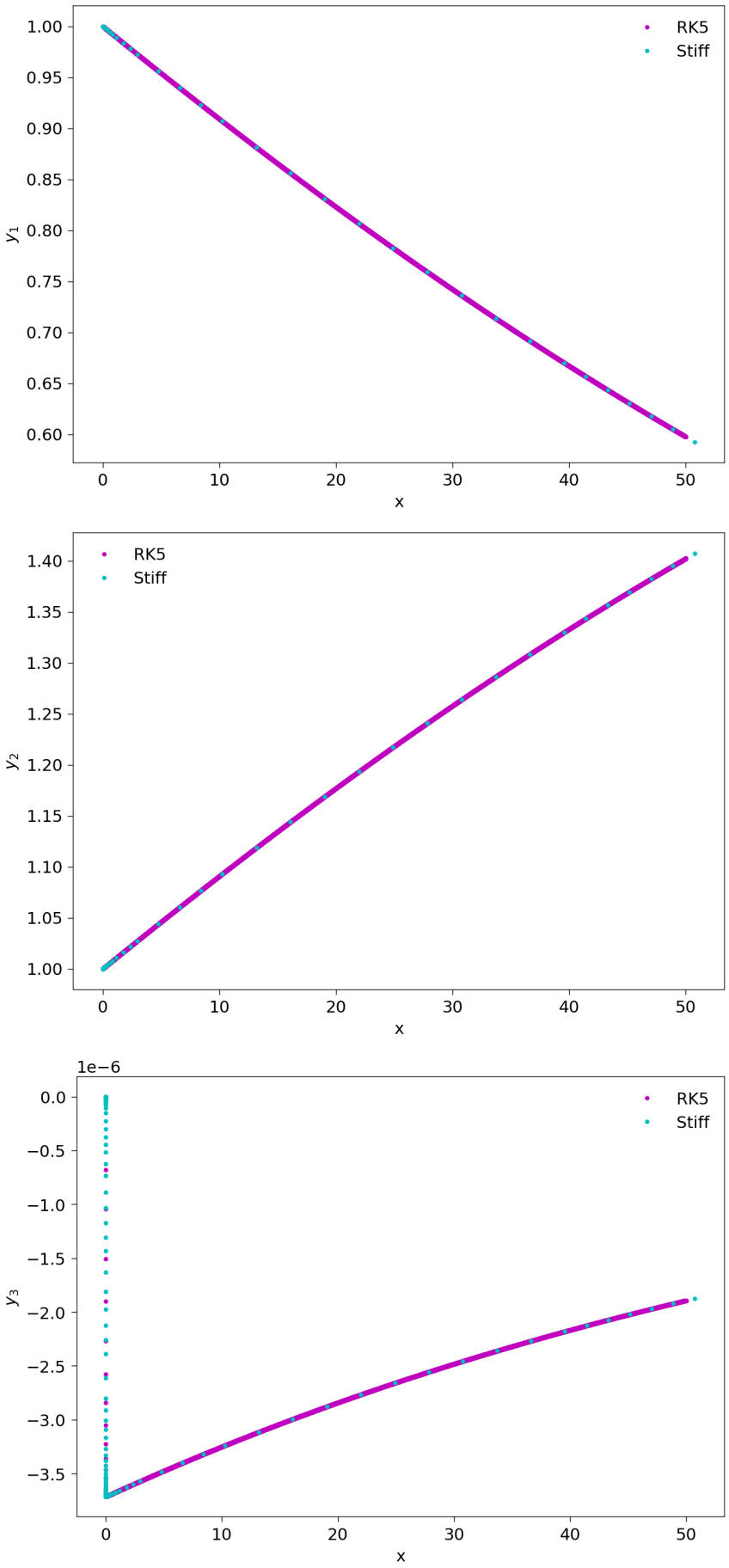

Figure 2.24: RK5 and Stiff integrator solutions for Equation 2.24 in the interval [0,50]: $y_1(x)$ (upper), $y_2(x)$ (middle) and $y_3(x)$ (lower).

2.1.4 Problem 4 (Advanced) - Accuracy and rounding errors

Exercise 14. As we will be potentially needing a huge number of steps, we may run into storage problems if we try to retain the information of every step. Create copies of the Euler and RK4 integrator routines that return only the final value at the end of the interval, and do not store the y (and x) values at each intermediate step.

```
def eul_mod(dydx,y0,a,b,n):
    h = (b-a)/float(n)
    x = a
    y = y0
    for i in range(n):
        y = y0 + h * dydx(x, y0)
        y0 = y
        x = x + h*i

    return y

def rk4_mod(dydx,y0,a,b,n):
    h = (b-a)/float(n)
    x = a
    y = y0
    for i in range(n):
        k1 = h*dydx(x      ,y0       )
        k2 = h*dydx(x+h*.5,y0+k1*.5)
        k3 = h*dydx(x+h*.5,y0+k2*.5)
        k4 = h*dydx(x+h    ,y0+k3    )
        y = y0 + (k1 + 2.*k2 + 2.*k3 + k4)/6.
        y0 = y
        x = x +i*h

    return y
```

Listing 11: Modified Euler and RK4 integrators.

In this case, as we are working with $1D$ vectors, we get rid of the vector nature of y in the definition of the integrators. As we do not need to store the intermediate steps, we overwrite the value of x and y and make the algorithm to only return the last step, that is, the approximation of $y(b)$.

Now call these routines with stepsizes varying over several orders of magnitude, record the final value and compare it to the analytic solution. Plot the resulting accuracy as a function of stepsize. Is there an optimum stepsize? If yes, how large is it? How do the different convergence orders manifest in the plot?

```
#parameters
p4_a = 0.
p4_b = 10.
p4_y0 = 1e-3

#funtions p1_dydx, p1_analytic from Exercise 1

#different steps and integrators
n = np.unique(np.round(np.logspace(1, 8, num=100)).astype(int))
error_eul = np.zeros(len(n))
error_rk4 = np.zeros(len(n))

for i in range(len(n)):
    y_eul = eul_mod(p1_dydx,p4_y0,p4_a,p4_b,n[i])
    error_eul[i] =  np.abs(y_eul - p1_analytic(p4_b))

    y_rk4 = rk4_mod(p1_dydx,p4_y0,p4_a,p4_b,n[i])
    error_rk4[i] = np.abs(y_rk4 - p1_analytic(p4_b))


```

```
#plots
plt.figure()
plt.plot(n1, error_rk4, 'g-o', label = 'RK4')
plt.plot(n1, error_eul, 'b-o', label = 'Euler')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Number steps')
plt.ylabel('Error')
plt.legend()
plt.show()
```

Listing 12: Integrators for different number of steps.

We decide to test the integrators with $n \in (10, 10^8)$. We set the maximum step number to be 10^8 due to the limited computational capacity we have. As mentioned, the error in this case is simply the comparison of the approximated value of the function in the last point of the interval and the analytical value at this point.

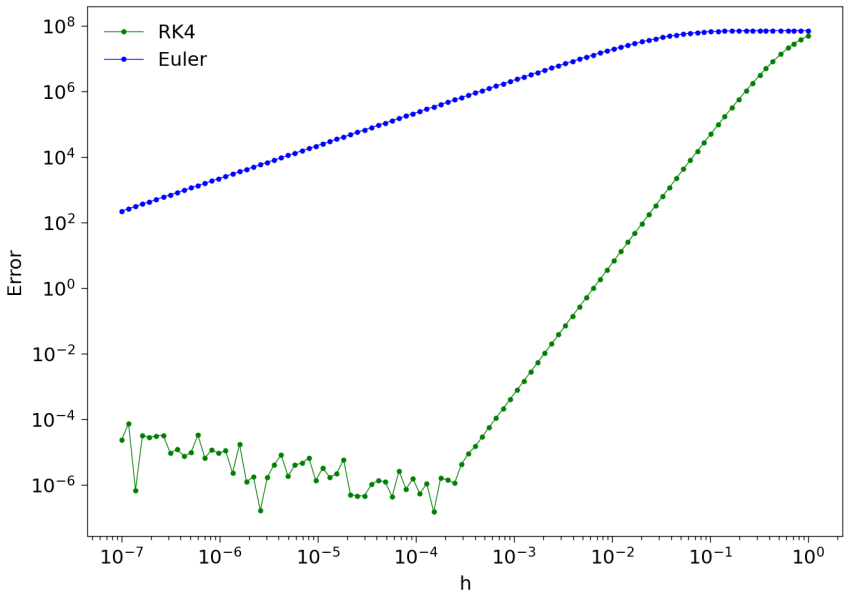

Figure 2.25: Error in approximation vs. h.

For obtaining Figure 2.25, we have converted n into h, to better see which is the optimum step-size. From this figure, we can see that for the RK4 we do have a minimum in the error, but this does not happen for the Euler method. However, we see a decreasing tendency for the error as we consider smaller step sizes. Although we are not able to explicitly see this optimum h, due to limitations in our computational resources, we know that it has to exist, since we have the functional form of it:

$$h_{opt} = \left(\frac{\delta}{cp}\right)^{\frac{1}{p+1}} \tag{2.26}$$

We have derived Equation 2.26 from the expression of the upper bound of the total of the estimation error(formula 2.27 of [1]):

$$\max_{n=0,1,\ldots,N-1} |t_{n+1}| \leq \frac{1}{L}\left(e^{L(b-a)} - 1\right)\left(ch^p + \frac{\delta}{h}\right) := Kf(h), \tag{2.27}$$

where $f(h) = ch^p + \frac{\delta}{h}$. As we want the optimum step-size, we derive $f(h)$ to see when it has a minimum:

$$f'(h) = cph^{p-1} - \frac{\delta}{h^2} = 0 \iff h_{opt} = \left(\frac{\delta}{cp}\right)^{\frac{1}{p+1}} \tag{2.28}$$

For the RK4, the numerical optimal step-size is $h_{opt} = 0.00015199$, where the minimum error is obtained. From Equation 2.26, we can obtain a value for the ratio δ/c, since we know the consistency order of RK4. Thus, we can presumably constraint the relation between the discretization and rounding errors:

$$\frac{\delta}{c} = h^{p+1}p \approx 3 \cdot 10^{-19} \tag{2.29}$$

On the other hand, we can also see what it is theoretically expected about the order of accuracy of both methods: the consistency order for the RK4 is higher than the Euler one. The error of the former method is smaller than the one for the latter for any given number of steps.

2.2 Part 2 - Cosmological application of solutions to ODEs

2.2.1 Problem 5

Exercise 15. Code the function that describes the evolution of the scale factor of the universe as an ODE. Neglect the radiation term, since this plays a role "only" in the very first epoch(s) of the Universe, together with inflation, which we will neglect as well, and which is justified as long as we are not interested in the details of these phases and have normalized all quantities to their present values. Remember that all times are in units of τ_H if we solve the equation for $\dot{a}/H_0$.

Neglecting the radiation component, we compute the time derivative of the scale factor, given by Equation 1.17, as follows:

```
def dydx_friedmann(t,a,omega_m,omega_l):
    """
    Inputs:
    t: time in units of Hubble time (independent variable)
    a: scale factor (dependent variable)
    omega_m: matter density parameter
    omega_l: cosmological constant density parameter
    """
    return np.sqrt(1. - (omega_m + omega_l) + omega_m / a + a**2 * omega_l)
```

Listing 13: Time derivative of the scale factor (a) as a function of Ω_m and Ω_Λ.

Problem 5.1
Exercise 16. Test your program now by means of your results from Exercise 5a/b from the PDF manual. Plot the results for both scenarios. As initial value, invert your result from Exercise 5a from the PDF manual (integrated from 0 to a) to obtain an approximate value $a(t_{start})$. In all of the following, we will adopt $t_{start} = 10^{-10}$ (in units of τ_H).

We recall Exercise 5 from the manual [1] where we are asked to calculate $t(z = 0)$, time elapsed after the big-bang, for two scenarios: (a) a flat universe filled with only matter and (b) an empty universe without vacuum energy.
For both cases we need to perform the following integral:

$$t'(a) = \int_0^a da \left(\Omega_K + \frac{\Omega_m}{a} + \frac{\Omega_r}{a^2} + a^2\Omega_\Lambda\right)^{-1/2}, \tag{2.30}$$

where $t'(a)$ denotes time in units of the Hubble time $\tau_H = 1/H_0$. For case (a) we only have $\Omega_m = 1$ which results in: $t'(z = 0) = t'(a = 1) = 2/3$. In case of (b) the only component is $\Omega_K = 1$ and so, $t'(z = 0) = t'(a = 1) = 1$.

For both cases (a) and (b), this integral also allows us to obtain the expression of a as function of t'. For case (a):

$$a(t') = \left(\frac{3}{2}t'\right)^{2/3}, \tag{2.31}$$

which is the well known Einstein-de Sitter universe. For case (b) the scale factor grows linearly with time:

$$a(t') = t'. \tag{2.32}$$

With these results in mind we can proceed to obtain the numerical solutions and represent them.

```
tstart = 1e-10
tend = 1.
astart_flatm = (3. / 2. * (tstart)) ** (2./3.)
astart_empty = tstart
```

```
5
6  # This describes an empty universe:
7  omega_m_empty = 0.
8  omega_l_empty = 0.
9
10 # This describes a flat universe with only matter:
11 omega_m_flatm = 1.
12 omega_l_flatm = 0.
13
14 t_empty,a_empty,nf,ng,nb,nt = rkdp(dydx_friedmann,astart_empty,tstart,tend,params=(
       omega_m_empty,omega_l_empty))
15 t_flatm,a_flatm,nf,ng,nb,nt = rkdp(dydx_friedmann,astart_flatm,tstart,tend,params=(
       omega_m_flatm,omega_l_flatm))
16
17 plt.figure()
18 plt.plot(t_empty, a_empty, label = r'Empty universe, $\Omega_K=1$')
19 plt.plot(t_flatm, a_flatm ,label = r'Flat + only matter universe, $\Omega_m=1$')
20 plt.xlabel(r'$t/\tau_{H}$')
21 plt.ylabel(r'$a$')
22 plt.legend()
23 plt.show()
```

Listing 14: Testing *dydx_friedmann* with cases from Exercise 5 from the manual.

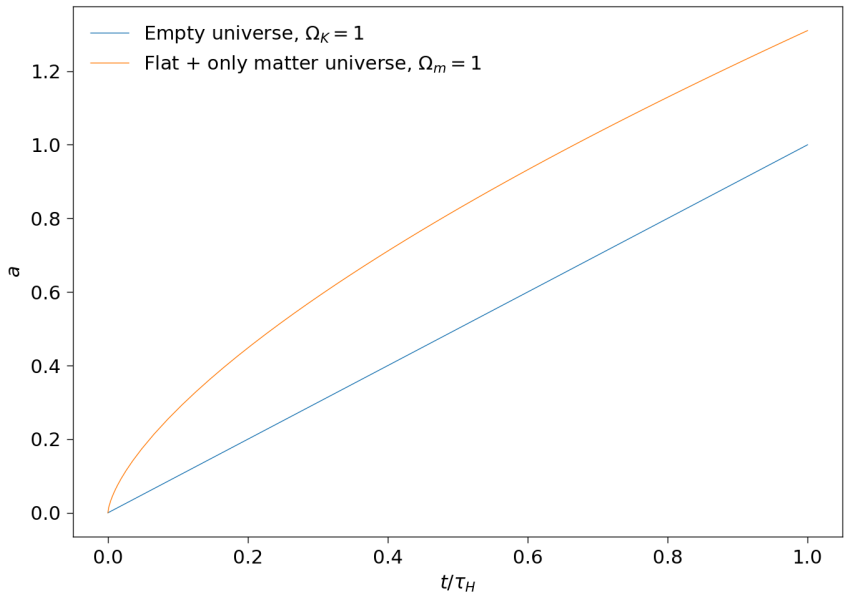

Figure 2.26: Evolution of the scale factor with respect to time (in units of the Hubble time, τ_H), for a flat universe filled with only matter ($\Omega_m = 1$) and for an empty universe ($\Omega_K = 1$).

From Figure 2.26 we see how the numerical solutions match what was expected theoretically. The empty Universe presents a linear growth of the scale factor with time. On the other hand, the flat universe with only matter, known also as the Einstein-de Sitter Universe, expands as expected.

Problem 5.2

Exercise 17. Now, try the solution for a matter-dominated, closed universe without cosmological constant, $\Omega_M = 3$. Calculate until $t_{\max} = 3.0$ and plot the result. Try to explain the observed behavior. Will the real universe also behave this way? Why or why not? If yes, what will the future evolution of the universe be like? If not, how could you modify the program to reproduce the "real" behavior?

In this case, we are going to study the evolution of the scale factor for a matter dominated, closed universe without cosmological constant. In this case: $\Omega_m = 3, \Omega_\Lambda = 0$. Thus $\Omega_K = -2$.

```
omega_m_p52 = 3.
omega_l_p52 = 0.

tend = 3.

t_p52,a_p52,nf,ng,nb,nt = rkdp(dydx_friedmann,astart,tstart,tend,params=(omega_m_p52,
    omega_l_p52))
t_p522,a_p522,nf2,ng2,nb2,nt2 = rkdp(dydx_friedmann2,a_p52[-1],t_p52[-1],tend,params=(
    omega_m_p52,omega_l_p52))
```

Listing 15: Matter-dominated, closed universe.

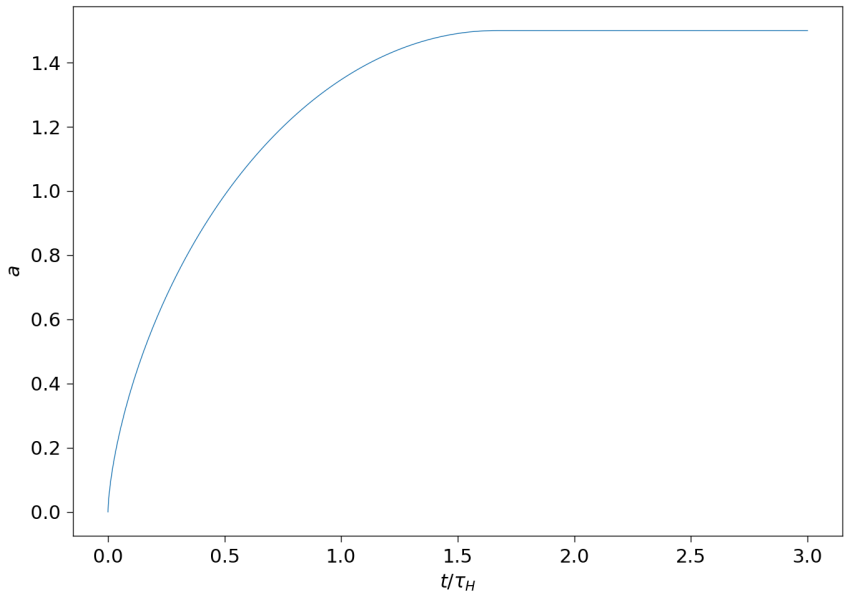

Figure 2.27: Evolution with time of the scale factor for a universe with: $\Omega_m = 3, \Omega_\Lambda = 0, \Omega_k = -2$.

In Figure 2.27, we can see how the evolution of the scale factor seems to become constant when it reaches the value $t'_c \approx 1.5$ [4]. Moreover, when we run the code we get this error: `RuntimeWarning: invalid value encountered in sqrt return np.sqrt(1. - (omega_m + omega_l) + omega_m / y + y**2 * omega_l)`.
This shows us that the term inside the square root of the `dydx` function has become 0 at some point. By substituting our values of Ω_i in `dydx_friedmann` function, we found:

$$-2 + \frac{3}{a} = 0 \iff a = \frac{3}{2} \tag{2.33}$$

Indeed, this is the value at which $a(t')$ gets stuck.

In fact, the machine interprets the negative values of the function as the 0, so the derivative of a is set to this value after reaching $a = 1.5$. As the algorithm of `rkdp` considers the previous steps when computing the next ones, the derivative of a remains 0 and thus a is constant. However, this would not be the real behavior of a, since the fact that it becomes constant is a numerical error of the computer. In order to solve it, we define the function `dydx` with a minus sign, that is, we consider the other solution of $\dot{a}$. Thus, we define its expression for $t' > t'_c$ which enables us to see the evolution of a after this time.

[4] The subindex c stands for critical

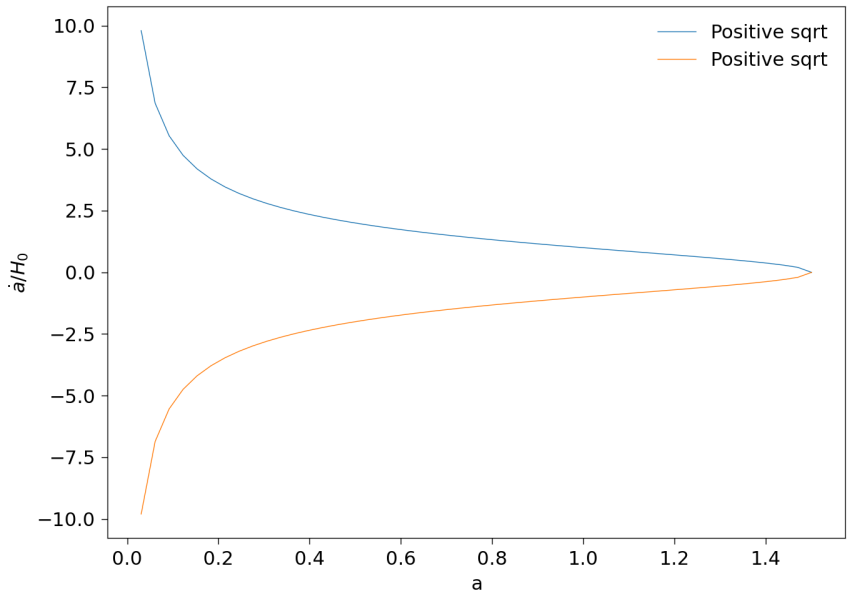

Figure 2.28: $\frac{\dot{a}}{H_0}$ as a function of a.

Physically, we can consider t_c' to be turnover point for the expansion of this universe. Thus, we want the derivative of a to represent this change in the evolution of a, hence we assign a negative value to its derivative (we are now in a contracting phase). On the other hand, in Figure 2.28 we can see that building $\dot{a}/H_0$ this way, we ensure that its expression remains continuous and evolves in a physically meaningful way, without introducing any discontinuities. Thus, the negative square root allows the model to reflect the transition in a mathematically consistent way.

```
def dydx_friedmann2(x,y,omega_m,omega_l):
    return - np.sqrt(1. - (omega_m + omega_l) + omega_m / y + y**2 * omega_l)

plt.figure()
plt.plot(t_p52, a_p52, 'b-', label = 'dydx_friedmann')
plt.plot(t_p522, a_p522, 'b-', label = 'dydx_friedmann2')
plt.xlabel(r'$t/\tau_H$')
plt.ylabel('a(t)')
plt.show()


#\dot{a}/H_0 as a function of a
x = np.linspace(0,1.4999999999) #(x=a)
plt.plot(x, (-2+3/x)**(1/2), label = 'Positive sqrt')
plt.plot(x, -(-2+3/x)**(1/2), label = 'Positive sqrt')
plt.legend()
plt.xlabel('a')
plt.ylabel(r'$\dot{a}/H_0$')
plt.show()
```

Listing 16: Matter-dominated, closed universe. Part 2

In Figure 2.29 we can see the real expected behavior of a universe with the stated conditions of this exercise (refer to Appendix B to see where we have changed the expression of $\dot{a}$). We can see how the effect of being matter dominated, closed and without dark energy contribution, make it recollapse at a symmetric time with respect to when the expansion reaches its maximum ($a(t)$ maximum).

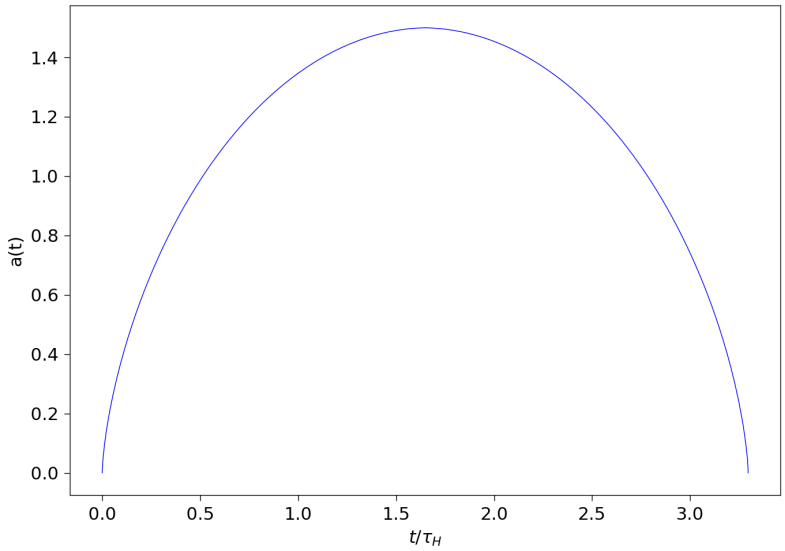

Figure 2.29: Evolution with time of the scale factor for a universe with: $\Omega_m = 3, \Omega_\Lambda = 0, \Omega_k = -2$, real behavior.

Problem 5.3
Exercise 18: Solutions for various parameter combinations

In this exercise we are going to study the evolution of different proposed universes. We give a brief explanation of the behavior with time for each of them basing our discussion in [2] (Figure 23.1). In the 2 first and last cases we are going to discuss about the universes by comparing them. We will study the big crunch scenario, where the universe contracts due to gravity overcoming the expansion (eventually collapsing into a singularity) and the big rip case, where the universe continues to expand at an accelerating rate, eventually tearing apart all matter.

We generalize the expression of the deceleration parameter q_0 when the universe is filled also with dark energy component (we consider no radiation):

$$\frac{\ddot{a}}{a}\Big|_{t_0} = -\frac{4\pi G}{3}\sum_i(\rho_i + 3P_i) = -\frac{4\pi G}{3}(\rho_m - 2\rho_\Lambda) = -\frac{H_0^2}{2}(\Omega_M - 2\Omega_\Lambda) \tag{2.34}$$

where we have considered $P_M = 0$ and $P_\Lambda = -\rho_\Lambda$. Thus:

$$q_0 := -\frac{\ddot{a}(t_0)a(t_0)}{\dot{a}^2(t_0)} = -\frac{\ddot{a}(t_0)}{\dot{a}(t_0)H_0^2} = \frac{\Omega_M}{2} - \Omega_\Lambda \tag{2.35}$$

For q_0 to be negative (accelerating cosmic scale), we need: $\Omega_M < 2\Omega_\Lambda$. We will compute the value of q_0 for each universe.

- $\mathbf{\Omega_M = 3.0, \Omega_\Lambda = 0.1}$.
  We have a similar universe than in Exercise 18, but in this case, it also has a small component of dark energy (introduced by the cosmological constant). In Figure 2.30, we see the effect of the introduction of this parameter into the equation of the evolution of $\dot{a}$ Equation 1.17. However, as the matter component tries to make the universe collapse and it is much bigger than the value of the cosmological constant density parameter and Ω_Λ is not big enough to counteract this effect. We have $q_0 = 1.4 > 0$, so the universe is decelerating.

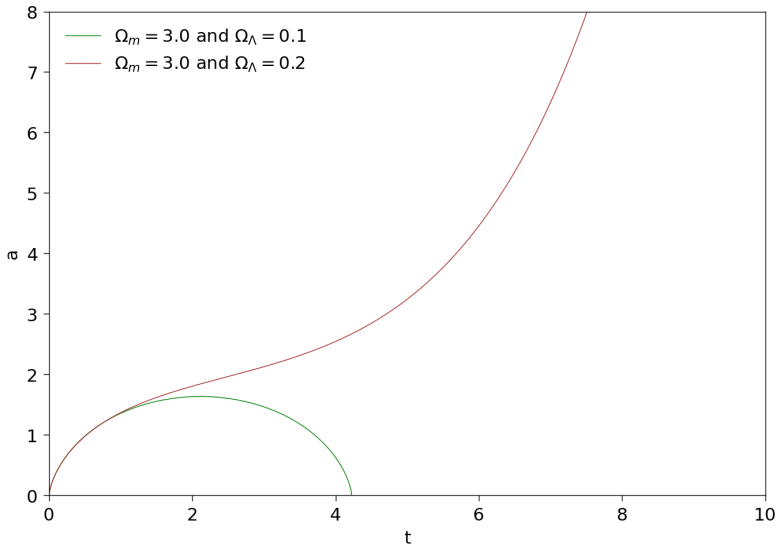

Figure 2.30: Matter dominated universes with low dark energy contribution.

- $\boldsymbol{\Omega_M = 3.0, \Omega_\Lambda = 0.2}$.
  On the other hand, with $\Omega_\Lambda = 0.2$, we see that the universe is exponentially expanding, so the dark energy effect is greater than the curvature and the matter. One must take into account that both matter and positive curvature 'try' to make the universe collapse and the cosmological constant contribution is the only that makes it expand. We see how a little variation of the dark energy component is crucial in the faith of the universe. Here $q_0 = 1.8$, so the universe is again decelerating.

- $\boldsymbol{\Omega_M = 0.0, \Omega_\Lambda = -0.1}$ (Anti de Sitter universe).
  In this case, we are studying a universe with negative dark energy component empty space has a negative energy density but a positive pressure. Thus, the intrinsic curvature of spacetime is determined by the cosmological constant $\Omega_K = 1 - \Omega_\Lambda$ and thus, the curvature is positive (recall Equation 1.16. We conclude that this universe will collapse, as seen in Figure 2.31. This universe has $q_0 = 0.1$.

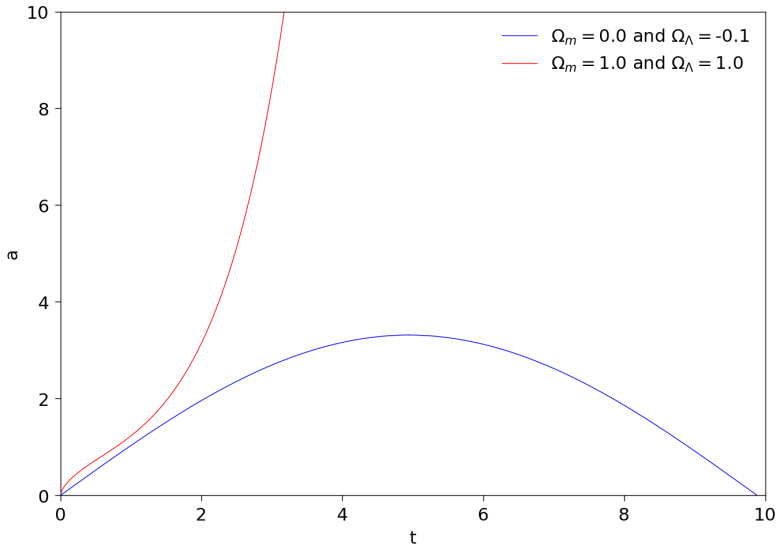

Figure 2.31: Anti de Sitter universe and dark energy-matter universe.

- $\boldsymbol{\Omega_M = 1.0, \Omega_\Lambda = 1.0}$

In this case, we have a universe with equal contribution of cosmological constant and matter. The curvature density parameter is $\Omega_K = -1$, so the curvature is positive: we have a closed universe. However, the contribution of the dark energy overcomes the other 2 factors and the universe expands indefinitely. Here, $q_0 = -0.5$, so the universe has accelerated expansion today.

- $\boldsymbol{\Omega_M = 1.0, \Omega_\Lambda = 2.55}$ (Loitering universe).
  We have a dark energy dominated universe, with positive curvature. In Figure 2.32, we see that it expands forever but loiters before it. Thus, the cosmological constant dominates. In this case, $q_0 = -2.05$, so the universe has accelerated expansion today.

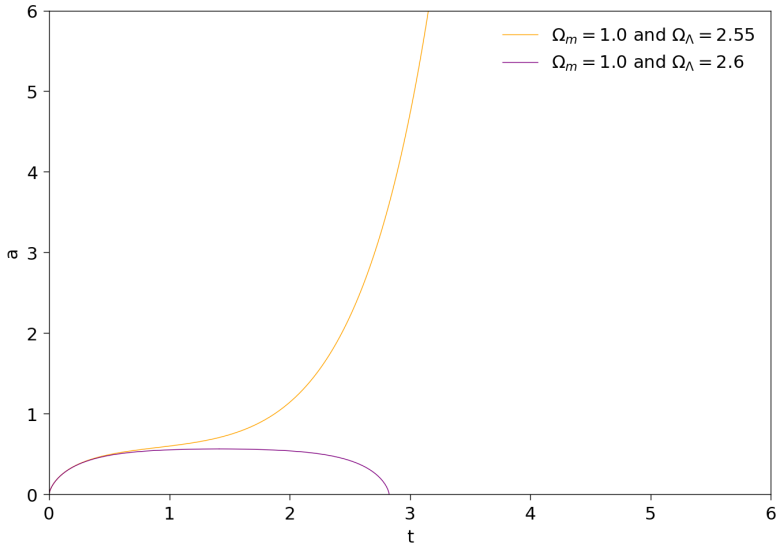

Figure 2.32: Dark energy dominated universes.

- $\boldsymbol{\Omega_M = 1.0, \Omega_\Lambda = 2.6}$
  In this final case, the universe collapses although the cosmological constant parameter is bigger than the matter one. We could argue this because the curvature of the universe is positive and the contribution of this term and the matter one overcomes the dark energy effect. Finally, $q_0 = -2.1$ so again the universe is expanding with acceleration.

The difference between this and the previous universe lies on the value of the cosmological constant and the fact that it might be close to the critical value Λ_c, defined as ([2] Eq. 23.25):

$$\Lambda_c = \frac{4}{9C^2} \tag{2.36}$$

where C is a constant of integration for the integration of the second Friedmann equation Equation 1.8, when introducing the first one into it and isolating all the a terms in the left-hand side of the equation:

$$\frac{2a\ddot{a} + \dot{a}^2 + k}{a^2} - \Lambda = -8\pi P \tag{2.37}$$

We find ourselves in a situation where $\Lambda > \Lambda_c$ (Loitering universe) and $0 < \Lambda < \Lambda_c$. Just for completeness, we mention that $\Lambda = \Lambda_c$ corresponds to the Eddinton-Lemaitre or Einstein universe, depending on the value of the matter components.

Problem 5.4
Exercise 19: Constraints on Ω_m and Ω_Λ. Using the observationally well-proven fact that our present Universe is very close to flatness, use your simulations to obtain the (Ω_m,Ω_Λ)

pair that is consistent with the present Hubble-parameter and $t_0 = 13.7 \pm 0.2$ Gyr (from the WMAP team). Analyze roughly the corresponding errors.

Given the age of the Universe, t_0, and the present Hubble parameter, $H_0 = 73 \pm 5$ km/s/Mpc, we get the measured time in units of the Hubble time as $\tau_0 \equiv t_0' = t_0/\tau_H = H_0 t_0$ and its uncertainty can be obtain via propagation of errors resulting in:

$$\Delta\tau_0 = \tau_0 \sqrt{\left(\frac{\Delta t_0}{t_0}\right)^2 + \left(\frac{\Delta H_0}{H_0}\right)^2}. \tag{2.38}$$

Now we can set the uncertainty in the value of τ_0 as $\tau_0 \pm \Delta\tau_0$. With this upper and lower limits of τ_0 we now fit the value of Ω_m so that when solving for the scale factor a the conditions $a(t' = \tau_0) = a(t' = \tau_0 + \Delta\tau_0) = a(t' = \tau_0 - \Delta\tau_0) = 1$ are satisfied.

```
H0 = 73 * 3.15 / 3.086 / 1000 #in Gyrs -1
delta_H0 = 5 * 3.15 / 3.086 / 1000
t0 = 13.7
delta_t0 = 0.2

tau00 = t0 * H0
delta_tau = np.sqrt((t0*delta_H0)**2 + (H0*delta_t0)**2)
tau0p = tau00 + delta_tau
tau0m = tau00 - delta_tau

omega_m = 0.244
tau0,a0,_, _, _, _ = rkdp(dydx_friedmann,astart,tstart,tau00,
params=(omega_m,1-omega_m))
# a = 1.00034497

omega_m = 0.189
taup,ap,_, _, _, _ = rkdp(dydx_friedmann,astart,tstart,tau0p,
params=(omega_m,1-omega_m))
# a  = 1.00030205

omega_m = 0.317
taum,am,_, _, _, _ = rkdp(dydx_friedmann,astart,tstart,tau0m,
params=(omega_m,1-omega_m))
# a = 1.00005954
```

Listing 17: Computation of the constraints in the Ω_m parameter.

Thus, the constraints for Ω_m are represented in the following Table 2.8.

	$\tau_0 - \Delta\tau_0$	τ_0	$\tau_0 + \Delta\tau_0$
Ω_m	0.317	0.244	0.189

Table 2.8: Range of uncertainty of the Ω_m parameter as a result of the uncertainty in the value of τ_0.

In Figure 2.33 we have plotted the evolution of the scale factor for the three values obtained for Ω_m. We can notice how the Universe has been expanding in the three cases. In particular, for $\Omega_m = 0.317$, that is for $t_0' = \tau_0 - \Delta\tau_0$, the expansion has been greater than for the other two cases. This makes us think that the Universe at these stages is still in a matter dominated era and is the matter component the one that drives the expansion. In Figure 2.34 we have extended the time axis and we are able to see how for the three cases at some point the cosmological constant component takes over and the expansion becomes more accelerated. Now the universe that has expanded the most is the one with $\Omega_m = 0.189$, that is, the one with the largest component of cosmological constant.

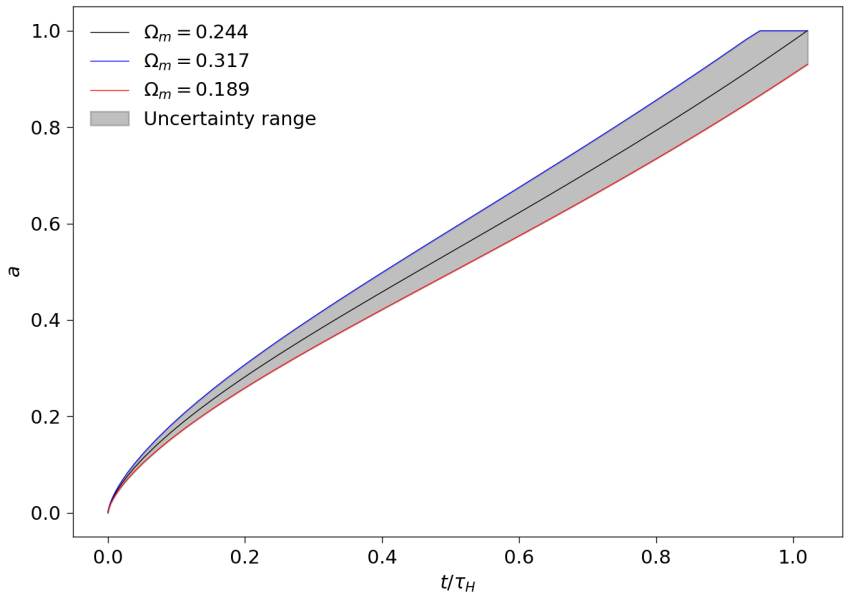

Figure 2.33: Evolution of the scale factor with time. A shaded region has been added, representing the uncertainty in the value of the scale factor due to the range of Ω_m values.

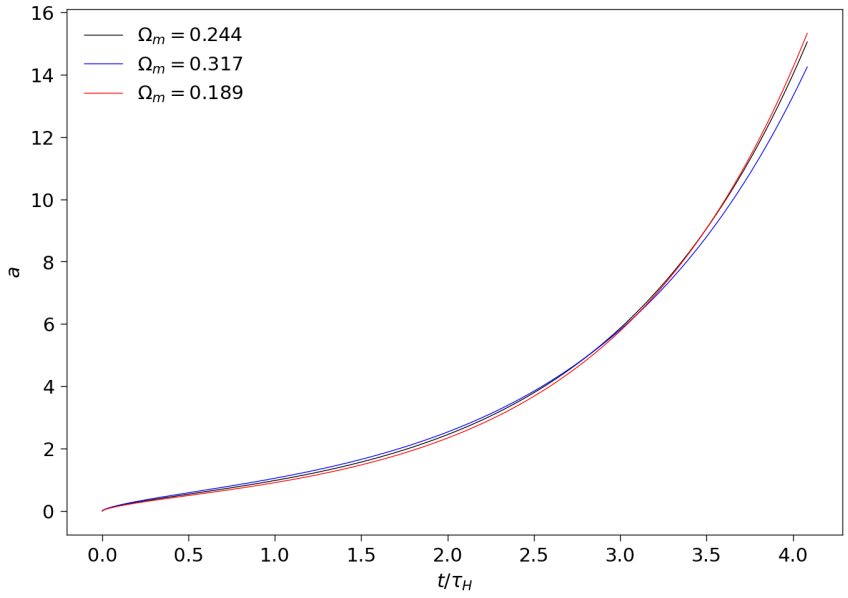

Figure 2.34: Evolution of the scale factor for the three values of Ω_m for a longer time interval.

Problem 5.5
Exercise 20: The (Ω_m,Ω_Λ) diagram.

As a final step in our study of cosmological models, we reproduce the well-known (Ω_m,Ω_Λ) diagram. This diagram allows us to visualize how depending on the choice of the pair of values (Ω_m,Ω_Λ), the curvature and the outcome of the universe are defined.

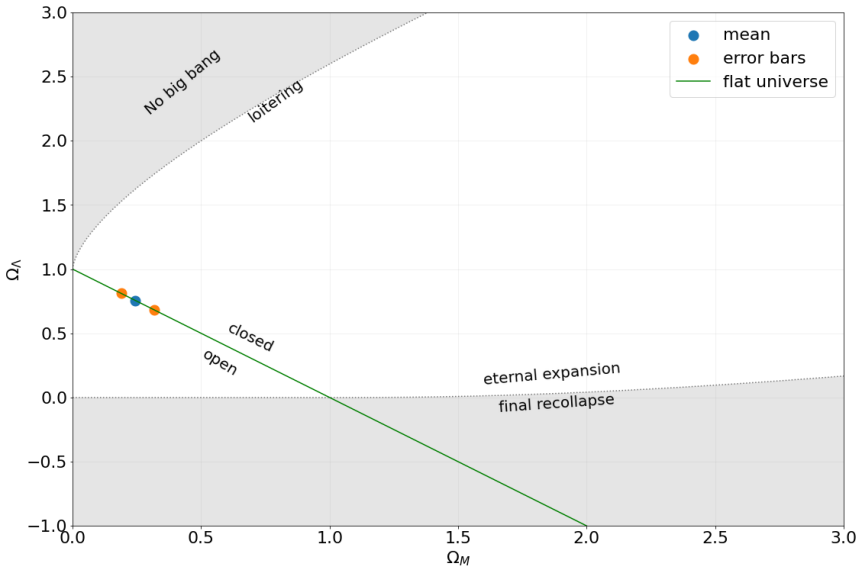

Figure 2.35: (Ω_m,Ω_Λ) diagram. The diagram shows the resulting characteristics of the universe for each pair of values (Ω_m,Ω_Λ). The previously constraints obtained for these parameters are also showed in the diagram.

3 Conclusions

The main goals of this project were to implement and test different numerical solvers of ODEs, compare their performance in terms of the error they produce and analyze how these solvers behave in different types of problems. In particular, we study their accuracy, efficiency and stability, under varying conditions, step sizes, and tolerances. Also, we studied a cosmological application in which ODEs integrators are needed. In particular, we focused on studying the evolution of the cosmic scale factor in different scenarios.

Firstly, we studied 2 single step methods: Euler and RK4. We arrived to the conclusion that the RK4 method performs better (produces less relative error) than the Euler method for the same number of steps (Table 2.1 and Table 2.3). On the other hand, we found the expected convergence rates for RK4 and Euler methods, studied the existence of an optimal step size (given by Equation 2.26) and analysed the stability of the integrators, comparing each with their stability condition Equation 2.14, Equation 2.19, respectively.

Next, we introduced two adaptive-step size methods, the Stiff and the RK5 integrators, which are specialized solvers for stiff problems. We proceed to test the four solvers with Equation 2.1. Setting them with a similar number of steps, we find out that RK5 gives the best result (or the least error). We also applied the 4 integrators to a coupled system of ODEs Equation 2.5 and to three coupled ODEs Equation 2.24. In the first case, we discuss their errors and their origin. We might conclude that for this case the RK4 method is the one that achieves less relative error. On the other hand, as on the second case we cannot provide an analytical result, we compare the results of RK5 and Stiff integrators and conclude that the latter is the most effective one, since it only needs 176 steps while the RK5 needs order of 10^3 steps. As the solutions of this system of equations take values in different scales of magnitude, we are working with an stiff problem, so we expected that this method outperformed the RK5 one.

Finally, the numerical integrators were applied to solving the evolution of the universe's scale factor under different scenarios (matter-dominated, empty, closed universes or dark energy dominated). The results were compared with known theoretical behaviors in order to explain the evolution found: collapsing and expanding universes. This experiment highlights the practical utility of numerical integrators in real-world applications.

Overall, adaptive-step integrators (RK5) are preferred for problems requiring high precision, while considering computational efficiency. For stiff problems, specialized solvers (Stiff integrator) significantly outperform general-purpose methods in terms of accuracy. Thus, this study highlights the importance of method selection based on problem characteristics. For future work, integrating more complex Cosmological models or exploring other Astrophysical fields could yield further insights of ODEs in scientific applications.

4 Author's note

We are aware of the maximum extension of the report (12 text pages), but we have adapted the extension of the report to the quantity of Exercises proposed in the Manual [1].

All the formulae are from the manual provided for the laboratory session, except otherwise stated.

Bibliography

[1] Joachim Puls/Fabian Heitsch. *Manual*. https://www.dropbox.com/scl/fo/9ofk6160mw7q4m08yh7wg/AI7_lzwxePoXc8EjM5zlzcw?rlkey=jbppyetylwavajemtb991f51g&st=pmwavs6c&dl=0. Accessed: Dec, 2024 (cit. on pp. 5, 22, 31, 33, 44).

[2] Ray d'Invemo. *Introducing Einstein's relativity*. Oxford University Press, 1992 (cit. on pp. 37, 39).

A Additional codes

Here we add the codes that were already implemented in the Jupyter Notebook provided.

```
def rk4(dydx,y0,a,b,n):
    """
    Integrate the system of ODEs given by y'(x,y)=dydx(x,y)
    with initial condition y(a)=y0 from a to b in n steps
    using the classical 4th-order Runge-Kutta method.

    Input:
        dydx : function to compute y'(x,y)
        y0   : 1-dim array of size m containing the initial values of y (at x0=a),
               where m is the number of differential equations in the system
        a    : start value of x
        b    : end value of x
        n    : number of steps to use for the interval [a,b]

    Output:
        x    : 1-dim array of size (n+1) containing the x values used in the
    integration
        y    : 2-dim array of size (n+1,m) containing the resulting y-values
    """
    print("rk4:")
    h = (b-a)/float(n)
    x = a + h*np.arange(n+1)
    y = np.zeros(((n+1),len(y0)))
    y[0] = y0
    for i in range(n):
        xi = x[i]; yi = y[i]
        k1 = h*dydx(xi      ,yi      )
        k2 = h*dydx(xi+h*.5,yi+k1*.5)
        k3 = h*dydx(xi+h*.5,yi+k2*.5)
        k4 = h*dydx(xi+h    ,yi+k3   )
        y[i+1] = yi + (k1 + 2.*k2 + 2.*k3 + k4)/6.
    print(": ",4*n,"calls of function")
    print(": ",n,"steps")
    return x,y
```

Listing 18: RK4 integrator

B Additional plots

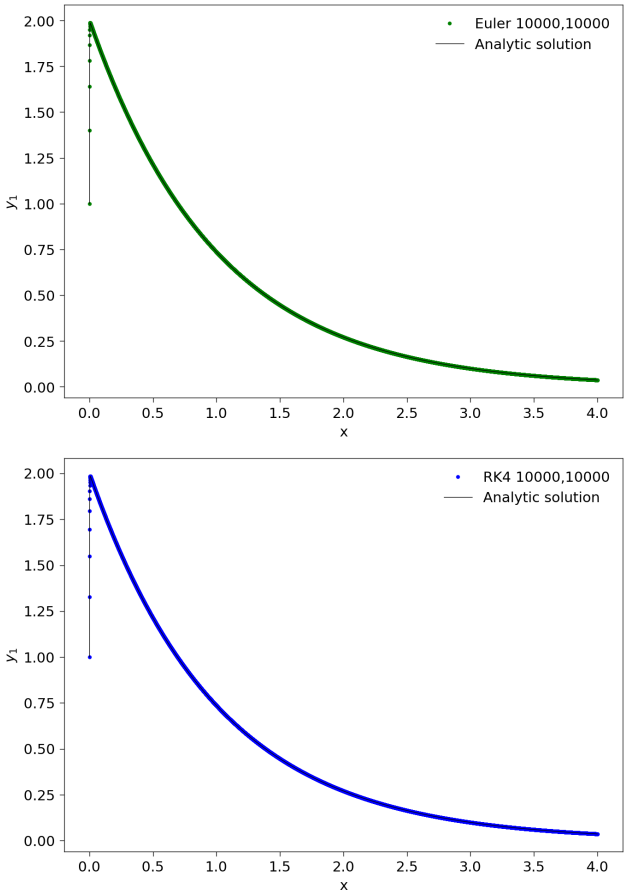

Figure B.1: Euler (upper) and RK4 (lower) approximations for Equation 2.5 $y_1(x)$ $n = 10000$.

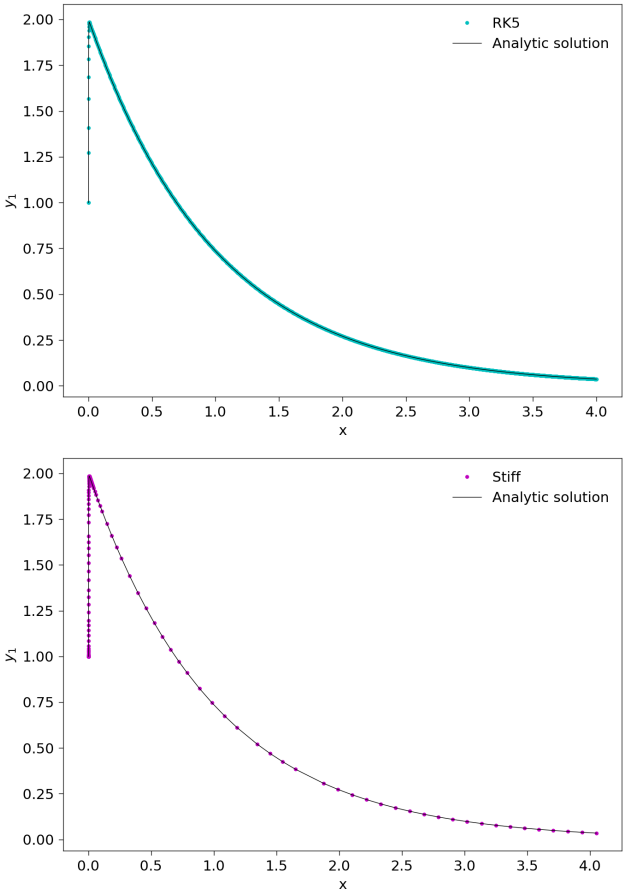

Figure B.2: RK5 (upper) and stiff (lower) approximations for Equation 2.5 $y_1(x)$ $tol = 10^{-5}$.

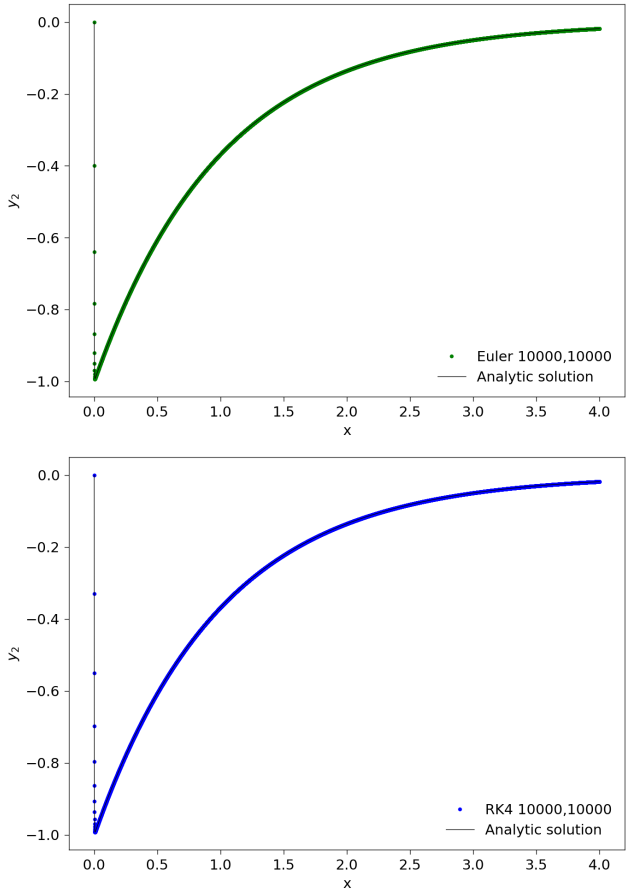

Figure B.3: Euler (upper) and RK4 (lower) approximations for Equation 2.5 $y_2(x)$ $n = 10000$.

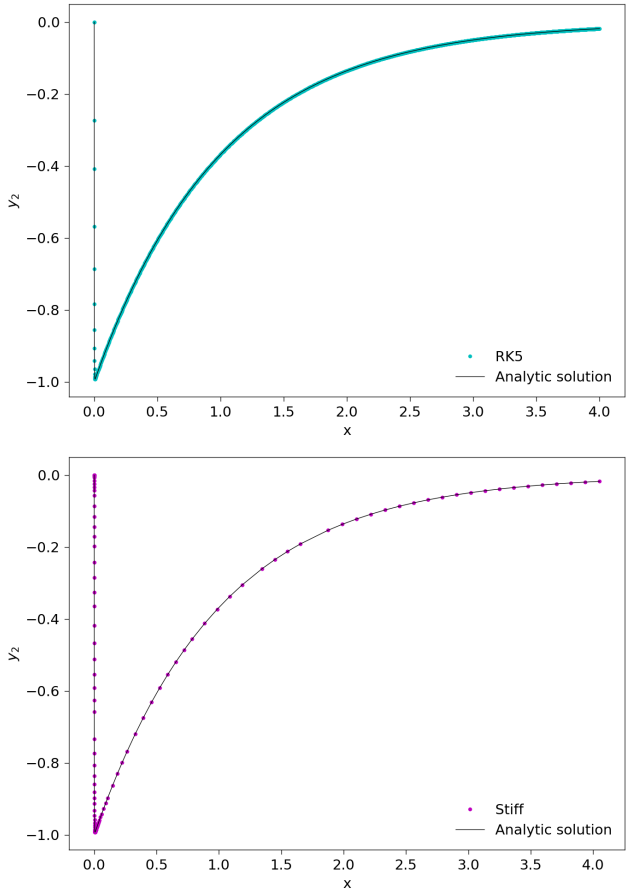

Figure B.4: RK5 (upper) and stiff (lower) approximations for Equation 2.5 $y_2(x)$ $tol = 10^{-5}$.

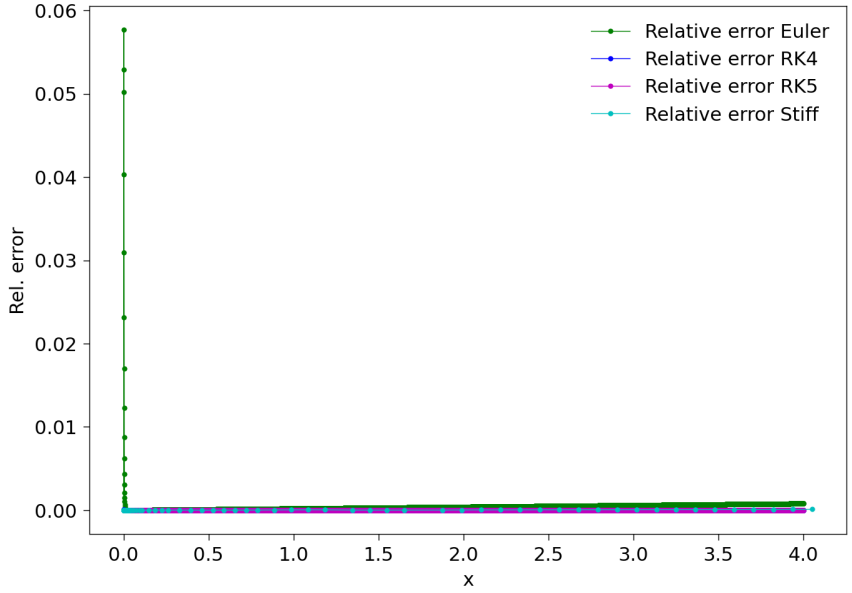

Figure B.5: Relative error for all methods for $y_1(x)$.

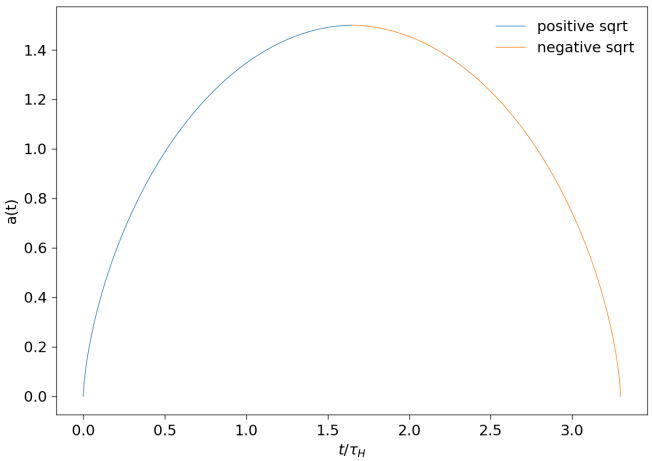

Figure B.6: Evolution with time of the scale factor for a universe with: $\Omega_m = 3, \Omega_\Lambda = 0, \Omega_k = -2$, 2 components.