Student names: . . . (please update)

Instructions: Update this file (or recreate a similar one, e.g. in Word) to prepare your answers to the questions. Feel free to add text, equations and figures as needed. Hand-written notes, e.g. for the development of equations, can also be included e.g. as pictures (from your cell phone or from a scanner). ***This lab is not graded. However, the lab exercises are meant as a way to familiarise with dynamical systems and to study them using Python to prepare you for the final project.*** *This file does not need to be submitted and is provided for your own benefit. The graded exercises will have a similar format.*

In this exercise, you will familiarise with ODE integration methods, how to plot results and study integration error. The file `lab#.py` is provided to run all exercises in Python. Each `exercise#.py` can be run to run an exercise individually. The list of exercises and their dependencies are shown in Figure 1. When a file is run, message logs will be printed to indicate information such as what is currently being run and and what is left to be implemented. All warning messages are only present to guide you in the implementation, and can be deleted whenever the corresponding code has been implemented correctly.

$$\begin{tikzcd}
 & & \text{ex1\_functions.py} \\
 & \text{exercise1.py} \arrow[ur] \arrow[r] \arrow[dr] & \text{ex1\_integration.py} \\
\text{lab1.py} \arrow[ur] \arrow[r] & \text{exercise2.py} & \text{ex1\_errors.py}
\end{tikzcd}$$

Figure 1: Exercise files dependencies. In this lab, you will be modifying `exercise1.py`, `ex1_functions.py`, `ex1_integration.py`,`ex1_errors.py` and `exercise2.py`. It is recommended to check out `exercise1.py` before looking into the other `ex1_*.py` files.

Question 1: Numerical integration

1.a Compute the analytical solution x(t) for the following linear dynamical system. Provide here the calculation steps, then implement the solution in `ex1_functions.py::analytic_function()` and run `exercise1.py` to plot the result.

$$\dot{x} = 2 \cdot (5 - x), \quad x(t = 0) = 1 \tag{1}$$

$$\begin{aligned}
& & \frac{dx}{dt} &= 2 \cdot (5 - x) \\
&\Rightarrow & \frac{1}{5-x} dx &= 2dt \\
&\Rightarrow & \int_{x_0}^{x} \frac{1}{5-x} dx &= \int_{t_0}^{t} 2dt \\
&\Rightarrow & -\ln(5-x) + \ln(5-x_0) &= 2(t-t_0) + C \\
&\Rightarrow & \ln(\frac{5-x}{5-x_0}) &= -2(t-t_0) - C \\
&\Rightarrow & \frac{5-x}{5-x_0} &= e^{-2(t-t_0)-C} \\
&\Rightarrow & x(t) &= 5 - (5-x_0)e^{-2(t-t_0)-C}
\end{aligned} \tag{2}$$

Using $x_0 = 1$ at time $t_0 = 0$, we obtain $C = 0$, thus:

$$\boxed{x(t) = 5 - 4e^{-2t}} \tag{3}$$

Python code: `x_correct = 5-4*np.exp(-2*t)`

1.b In some cases, an ODE system may not have an analytical solution or it may be difficult to compute. Implement Euler integration in `ex1_integration.py::euler_integrate()`, then run `exercise1.py` again to compare the solution of `euler_integrate()` (with 0.2 timestep) to the analytical solution obtained previously and include a figure of the result here. Make sure to also implement `ex1_functions.py::function()` so that the code may be run correctly.
As a code template, check out `ex1_integration.py::euler_example()`.

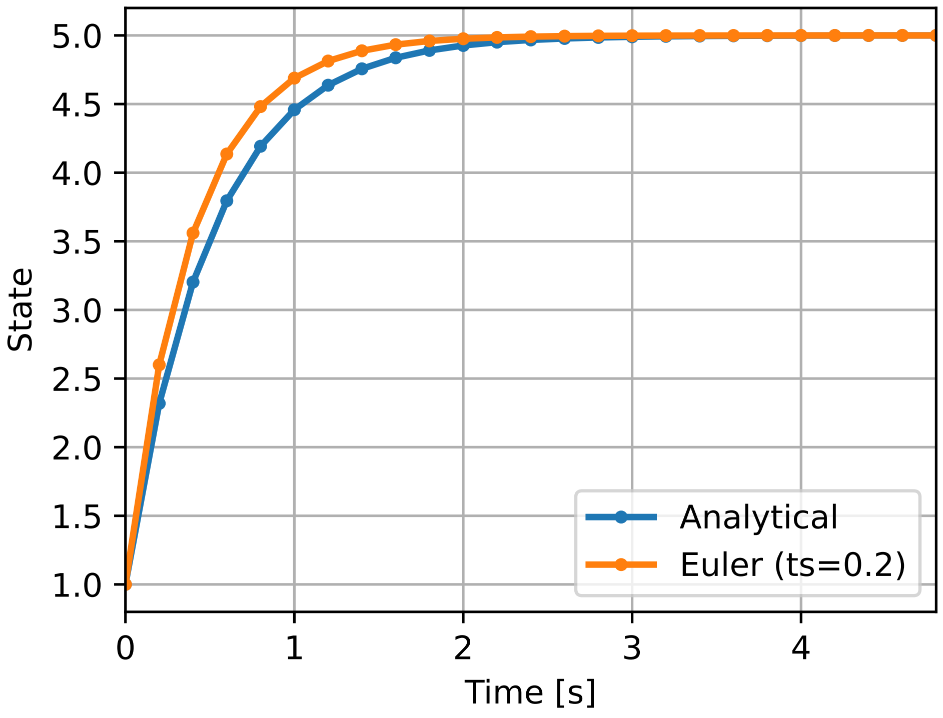


Figure 2: We can see that Euler integration with a 0.2 time steps approximately corresponds to the analytical solution.

By computing the average abosulute error using:

$$err = \frac{1}{N}\sum_{i=1}^{N}|f_{method}(i) - f_{analytical}(i)| \tag{4}$$

with N being the number of state samples, the method being Euler in this case and $f_{analytical}()$ being the equation given by 3. According to this equation, we obtain an average error of ~ 0.085 for the Euler method in this case. We also obtain a maximum absolute error of ~ 0.357.

1.c Various efficient libraries are available to facilitate ODE integration, such as Scipy. In this exercise first implement your own Runge-Kutta 4th order method (in `ex1_integration.py::ode_intgrate_rk()`). Then use scipy.odeint Lsoda method (in `ex1_integration.py::ode_intgrate()`) and scipy.ode adaptive Runge-Kutta Dopri method of order 4(5) (in `ex1_integration.py::ode_intgrate_dopri()`) and solve the ode (overimpose plots of the solution using different methods and markers). Compare all these methods (euler, runge-kutta, lsoda and dopri) by defining an error function (there are multiple ways you could do this, choose an appropriate method and explain why) and number of time steps.

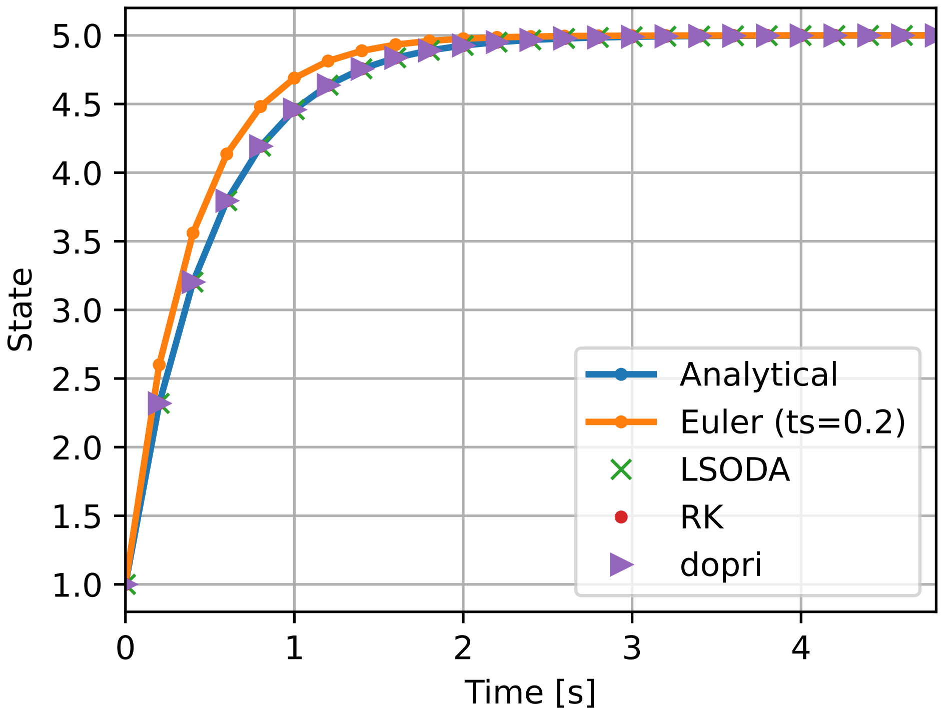


Figure 3: The Euler, Runge-Kutta and LSODA integration methods plotted and compared to the analytical solution

Method	Average error	Max error	# Timesteps
Euler	~ 0.085	~ 0.357	25
Runge-Kutta	~ 0.0001	~ 0.0004	25
dopri	$\sim$ 6.41e−8	$\sim$ 1.43e−7	Adaptive
LSODA	$\sim$ 1.17e−8	$\sim$ 5.79e−8	Adaptive

Table 1: Error analysis for different integration methods

Maintaining the 0.2 time step from the previous exercise, we observe that the error is much smaller using the Runge-Kutta method for a same number of integration times steps (25). The LSODA and dopri also obtains small error values similarly to the Runge-Kutta method, although this algorithm uses an adaptive time step which allows to control the tolerance on the error, at the cost of additional computation.

1.d The error comparison between Euler and Runge-Kutta of question 1.c is not fair for the Euler method. Briefly say why. Choose another number of time steps for the Euler method that is fairer when compared to Runge-Kutta. Provide the error values, number of time steps, and include a figure comparing the two integration methods. Briefly discuss which integration method is best.

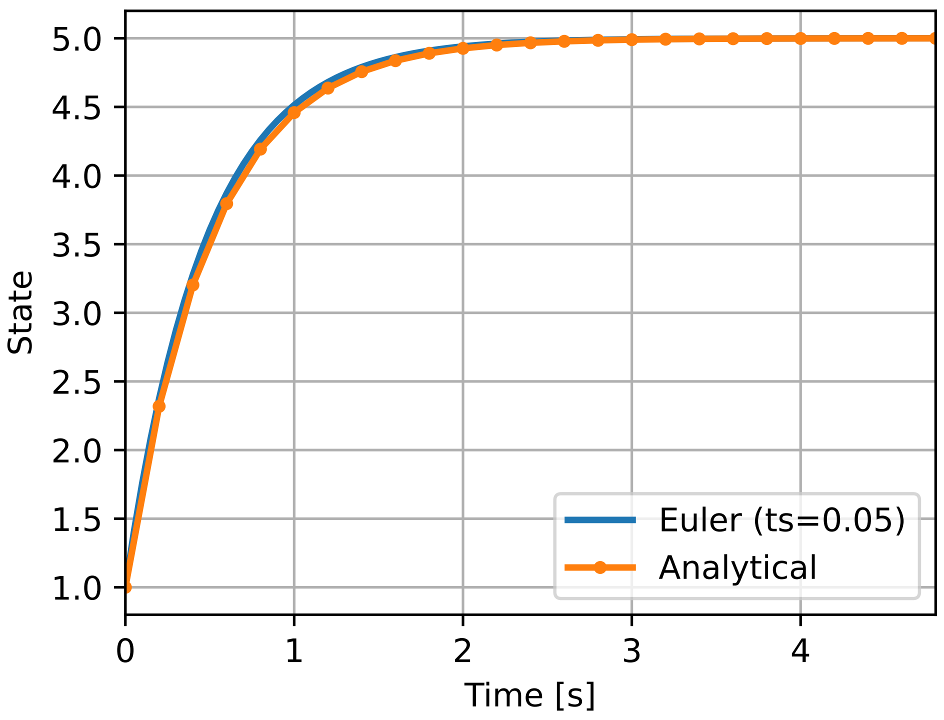


Figure 4: Euler integration using smaller time step (ts = 0.05)

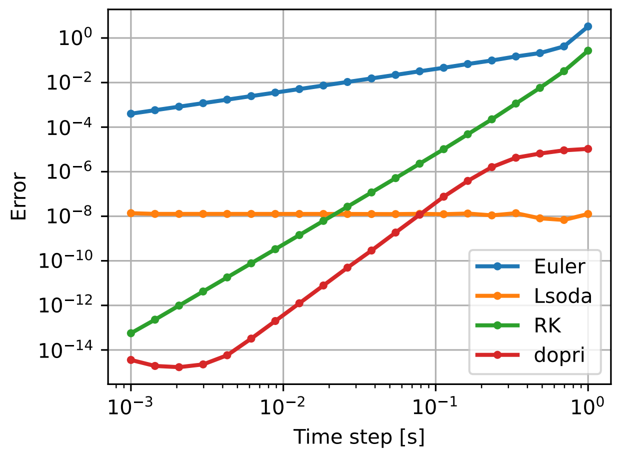
The Runge-Kutta integration method computes the derivatives 4 times at each iteration. A fair comparison is therefore to make 4 Euler steps for each Runge-Kutta step.

Method	Average error	Max error	# Timesteps
Euler	~ 0.085	~ 0.357	25
Runge-Kutta	~ 0.0001	~ 0.0004	100
dopri	$\sim 6.41e{-8}$	$\sim 1.43e{-7}$	Adaptive
LSODA	$\sim 1.17e{-8}$	$\sim 5.79e{-8}$	Adaptive
Euler (smaller ts)	~ 0.020	~ 0.077	100

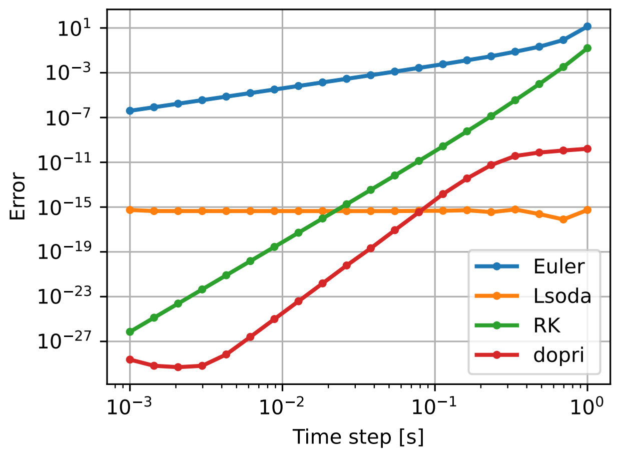
Table 2: Error analysis for different integration methods including smaller Euler integration with smaller time step

The error is still much smaller using Runge-Kutta time steps (ode45), even if approximately the same number of derivatives are computed with both methods (100 times for Euler, $25 \cdot 4 = 100$ times for Runge-Kutta).

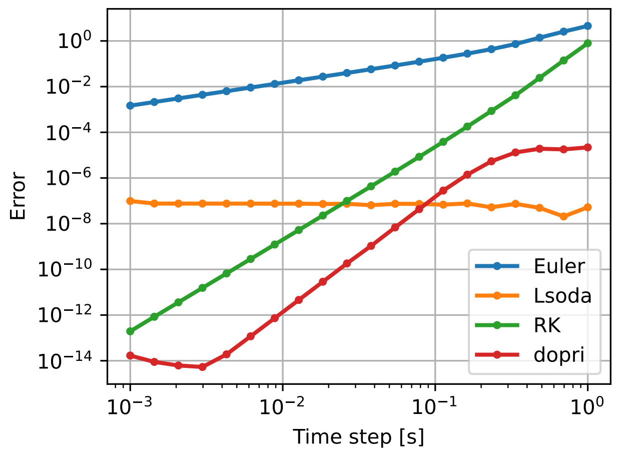
1.e Test the role of the step size by plotting the integration error as a function of step size. You can use `ex1_errors.py::compute_error()` to do this by completing the code in `ex1_errors.py::error()`. How accurate is the solution compared to the analytical solution for different step sizes? Include here a graph showing the error against the step size. Explain which error measure you used (there are several options). The error of the adaptive solvers can be reduced by setting a smaller value of the relative tolerance of the solver (rtol). Reduce rtol and check the solver's accuracy.



(a) L1



(b) L2



(c) Linf

Figure 5: Error in function of time step

$$err_{L1} = \frac{1}{N}\sum_{i=1}^{N} |f_{method}(i) - f_{analytical}(i)| \tag{5}$$

$$err_{L2} = \frac{1}{N}\sum_{i=1}^{N} (f_{method}(i) - f_{analytical}(i))^2 \tag{6}$$

$$err_{Linf} = max\left(|f_{method}(i) - f_{analytical}(i)|\right) \tag{7}$$

The Lsoda method uses a fixed L1 error tolerance of $rtol \sim 1e-8$. The dopri method uses a higher rtol=1e-4. In both cases the tolerance is reached by the solver and can be reduced by changing rtol.

Question 2: Stability analysis

2.a Find the fixed points of the following linear dynamical system, and analyze their stability (briefly describe the calculation steps).

$$\dot{x} = Ax, \qquad A = \begin{pmatrix} 1 & 4 \\ -4 & -2 \end{pmatrix} \tag{8}$$

Fixed point $\vec{x_o}$:

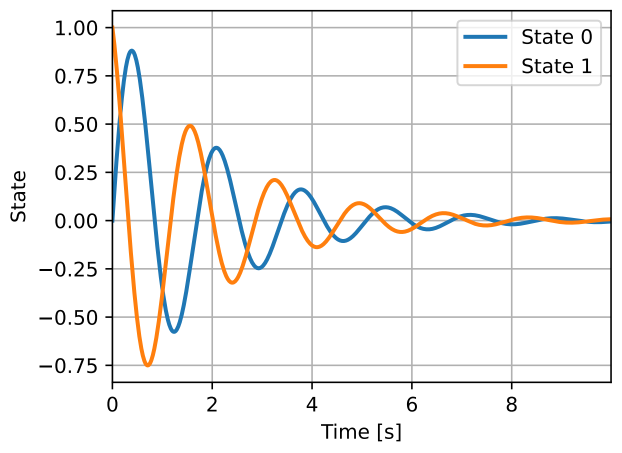
$$\begin{aligned} \dot{\vec{x_0}} = \vec{0} = \begin{pmatrix} 1 & 4 \\ -4 & -2 \end{pmatrix} \vec{x_o} \\ \Rightarrow \quad \boxed{\vec{x_o} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}} \end{aligned} \tag{9}$$

Eigenvalues λ:

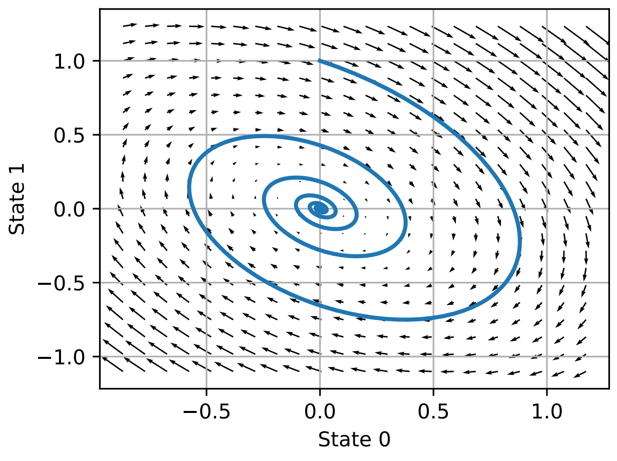
$$\begin{aligned} \det(A - \lambda I) = 0 \\ \Rightarrow \quad \det \begin{pmatrix} 1-\lambda & 4 \\ -4 & -2-\lambda \end{pmatrix} = 0 \\ \Rightarrow \quad (1-\lambda)(-2-\lambda) + 16 = 0 \\ \Rightarrow \quad \lambda^2 + \lambda + 14 = 0 \\ \boxed{\lambda^{\pm} = \frac{-1 \pm \sqrt{1 - 4 \cdot 14}}{2} = -\frac{1}{2} \pm \frac{\sqrt{-55}}{2}} \end{aligned} \tag{10}$$

The fixed point is stable because the real part of both eigenvalues are smaller than 0.
Non-zero imaginary part: the system exhibits **(damped) oscillations**.

2.b Perform numerical integration from different initial conditions to verify the stability properties. See `exercise2.py::exercise2()` for implementation. Include some figures with these different time evolutions and their corresponding phase portrait and explain their roles.



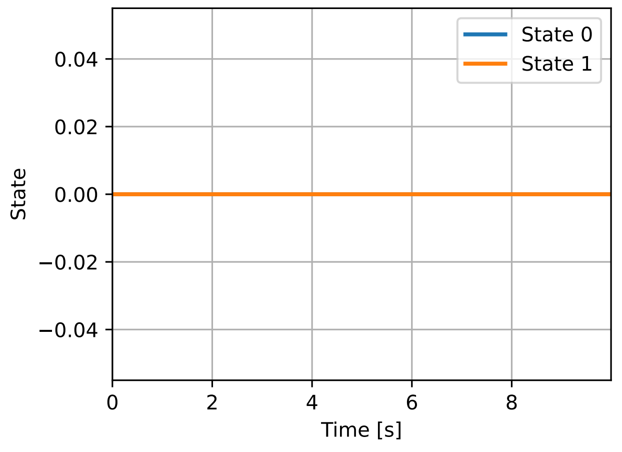
(a) State



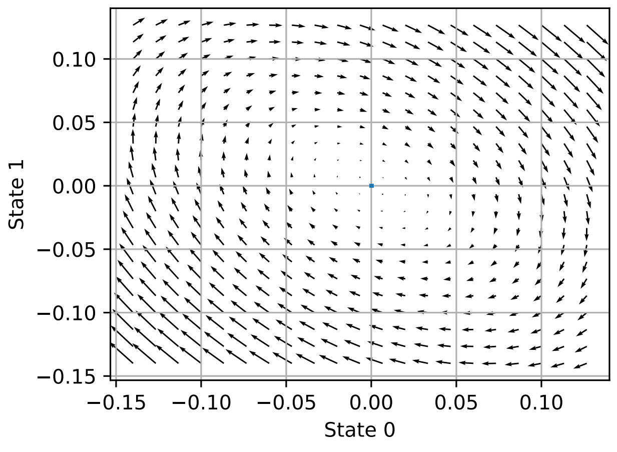
(b) Phase

Figure 6: The left graph shows typical oscillations with decreasing amplitude (damped oscillations) in function of time. The right graph shows the evolution of both states, but the time information is lost.

The damped oscillations correspond to an inward spiral in the phase plane. The quiver plot is a vector plot that shows the general behavior of the system in the phase plane.



(a) State



(b) Phase

Figure 7: These graphs show the evolution of the system from (0,0) to demonstrate the fixed point behavior.

2.c Change one value in matrix A such that the time evolution becomes periodic for some initial conditions. Say which value and include a time evolution figure.

In order for the system to become periodic, we need the eigenvalues λ to be purely imaginary, say by modifying the first element of the matrix:

$$\begin{aligned}
&\det(A-\lambda I)=0\\
\Rightarrow\quad &\det\begin{pmatrix} a-\lambda & 4\\ -4 & -2-\lambda\end{pmatrix}=0\\
\Rightarrow\quad &(a-\lambda)(-2-\lambda)+16=0\\
\Rightarrow\quad &\lambda^2+(2-a)\lambda+(16-2a)=0\\
\lambda^{\pm} = &\frac{-(2-a)\pm\sqrt{(2-a)^2-4\cdot(16-2a)}}{2}
\end{aligned} \tag{11}$$

By selecting $a=2$, we obtain:

$$\lambda^{\pm}=\frac{-(2-2)\pm\sqrt{(2-2)^2-4\cdot(16-4)}}{2}=\pm\sqrt{12}j \tag{12}$$

Thus the system changed to:

$$\dot{x}=Ax,\qquad A=\begin{pmatrix}2 & 4\\ -4 & -2\end{pmatrix} \tag{13}$$

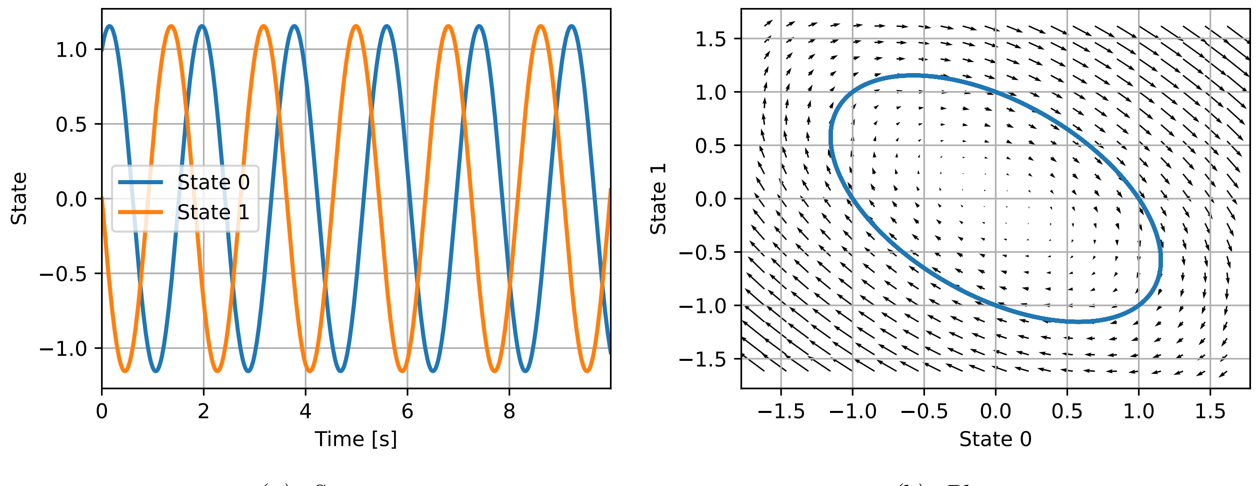


Figure 8: A is changed to: A = [[2,4],[-4,-2]] Eigen values are now complex numbers with real part = 0. This leads to oscillatory behavior without damping.

2.d Compute the eigenvalues and eigenvectors of the following system. What can you say about the stability of its fixed point? What is the meaning of its eigenvectors? Try to plot the flow of the system to get an intuition of its behavior.

$$\dot{x} = Ax, \qquad A = \begin{pmatrix} 1 & 1 \\ 4 & -2 \end{pmatrix} \tag{14}$$

The first part of the exercise is solved as in 2.c.

Eigenvalues λ:

$$\begin{aligned} & \det(A - \lambda I) = 0 \\ \Rightarrow \quad & \det\begin{pmatrix} 1-\lambda & 1 \\ 4 & -2-\lambda \end{pmatrix} = 0 \\ \Rightarrow \quad & (1-\lambda)(-2-\lambda) - 4 = 0 \\ \Rightarrow \quad & \lambda^2 + \lambda - 6 = 0 \\ & \boxed{\lambda^{\pm} = \frac{-1 \pm \sqrt{1+4\cdot 6}}{2} = -\frac{1}{2} \pm \frac{5}{2}} \end{aligned} \tag{15}$$

The eigenvalues of the system are equal to λ_1 = 2 and λ_2 = -3
The fixed point is a saddle node because the real part of one eigenvalue is greater than 0.

Eigenvectors $\vec{v}$:
Note that the matrix is singular, therefore we just consider one of the two equations.
There are infinite solutions to the system, we will consider vectors with unitary norm.

$$\begin{aligned} & (A - \lambda_1 I)\vec{v_1} = 0 \\ \Rightarrow \quad & (A - 2I)\vec{v_1} = 0 \\ \Rightarrow \quad & \begin{pmatrix} -1 & 1 \\ 4 & -4 \end{pmatrix}\vec{v_1} = 0 \\ \Rightarrow \quad & v_{1x} = v_{1y} \\ & \boxed{\vec{v_1} = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}} \end{aligned} \tag{16}$$

$$\begin{aligned} & (A - \lambda_2 I)\vec{v_2} = 0 \\ \Rightarrow \quad & (A + 3I)\vec{v_2} = 0 \\ \Rightarrow \quad & \begin{pmatrix} 4 & 1 \\ 4 & 1 \end{pmatrix}\vec{v_2} = 0 \\ \Rightarrow \quad & 4v_{2x} = -v_{2y} \\ & \boxed{\vec{v_2} = \begin{pmatrix} 1/\sqrt{17} \\ -4/\sqrt{17} \end{pmatrix}} \end{aligned} \tag{17}$$

You can observe in Figure 9 how the eigenvector associated to the negative value corresponds to a line of stable solutions (the stable manifold). Due to the uniqueness theorem, trajectories cannot cross, therefore the stable manifold is effectively dividing the phase space in two regions. Points starting on the left of the manifold will diverge towards $(-\inf, -\inf)$, points starting on the right of the manifold will diverge towards $(+\inf, +\inf)$.

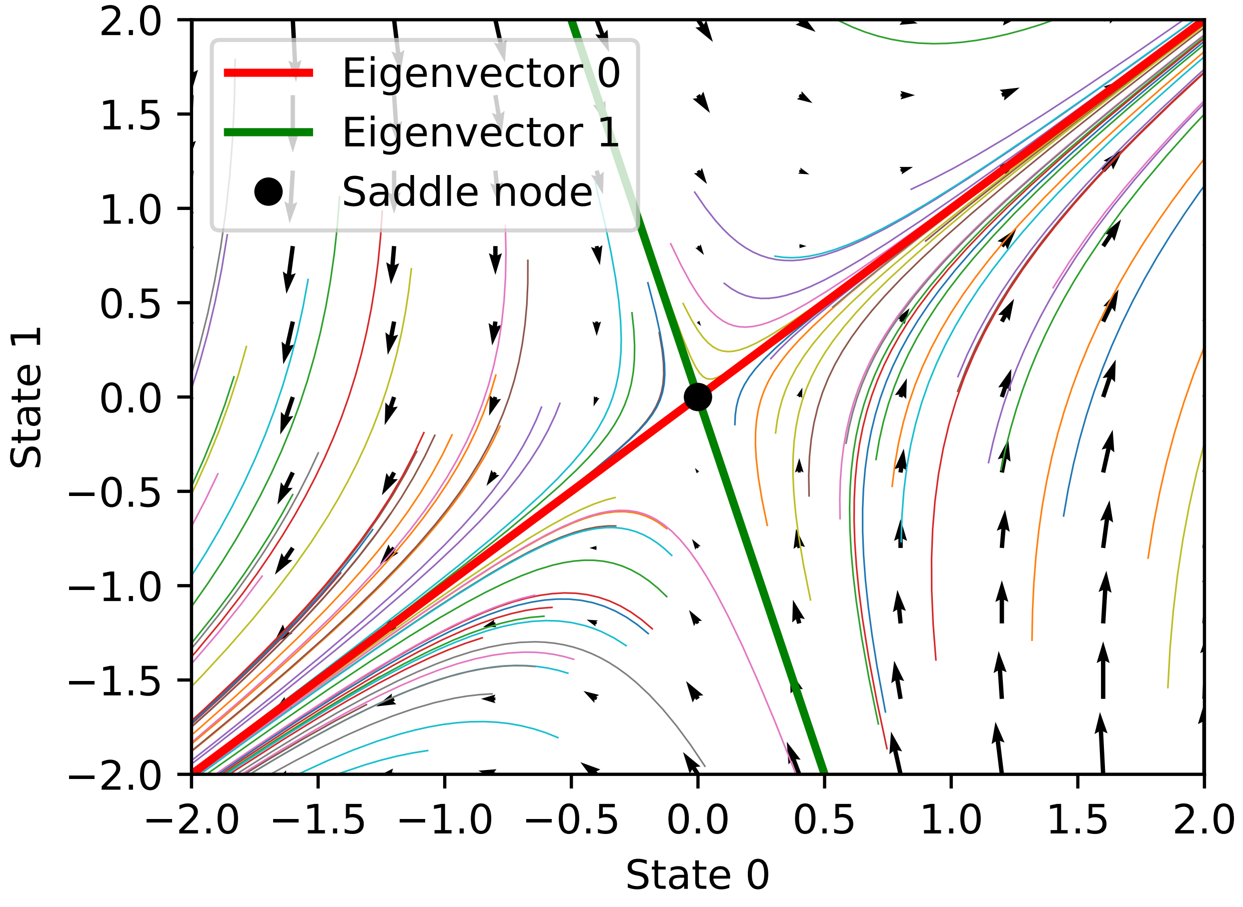


Figure 9: Phase diagram of the system with a saddle point. The eigenvalues of the saddle point correspond to the stable and unstable manifold of the system