Rabbits vs Sheep

Final Project

June 13, 2023

Jacopo Zacchigna, Devid Rosa

A project presented for the course
Dynamical System for Artificial Intelligence

Artificial Intelligence & Data Analytics
University of Trieste

Table of Contents

1 Find the fixed points and analyze their stability **2**
1.1 Linear analysis . . . 3

2 Finding the nullclines **3**
2.1 Finding the direction of the vector field along the nullclines . . . 4
2.1.1 x-nullclines . . . 4
2.1.2 y-nullclines . . . 4
2.2 Visualizing the nullclines . . . 4

3 Numerical Analyses of the vector field **5**
3.1 Import the necessary packages . . . 5
3.1.1 Define the differential equation . . . 5
3.2 Vector fields for a few different parameters . . . 7
3.3 Animated vector field . . . 9

4 Linearization of the flow near the fixed points **9**
4.1 Find the Jacobian . . . 9
4.2 Evaluate the Jacobian in the fixed points . . . 10
4.3 Relative Traces and Determinants . . . 10

5 Numerical simulation for orbits near fixed points **10**

6 Finding the manifold near the fixed points **13**
6.1 Eigenvalues and eigenvectors computations . . . 13
6.1.1 $J_{(0,0)}$ manifolds . . . 13
6.1.2 $J_{(1,1)}$ manifolds . . . 14
6.1.3 $J_{(0,2)}$ manifolds . . . 14
6.1.4 $J_{(3/2,0)}$ manifolds . . . 14

7 Analysing the non linear extension of the manifold **14**
7.1 Manifold visualization forward and backward in time . . . 15

8 Concluding our analyses **15**
8.1 Visualization of the entire differential equation . . . 15

1 Find the fixed points and analyze their stability

Let's start by looking at our differential equation:

The general equation has a and b as parameters

$$\begin{cases} \dot{x} = x(3 - ax - by) \\ \dot{y} = y(2 - x - y) \end{cases}$$

The entire of the code can be found in this GitHub Repository

The differential equations model the interaction between populations of rabbit and sheep. This is a classic Lotka-Volterra model of competition where we suppose that both species are competing for the same food supply but the amount available is limited.

In this model we include two parameters:

1. **a**: This parameter is included because each specie would grow to its carrying capacity in the absence of the other (logistic growth). Since rabbits have a massive ability to reproduce, we assign them a higher intrinsic growth rate.
2. **b**: This parameters is added to consider the conflicts between species, which we consider to be proportional to the size of each population. Furthermore, we assume that the conflicts reduce the growth rate for each species, but the effect is more severe for the rabbits.

Discussing our calculation To find the fixed points we study the following:

$$\begin{cases} \dot{x} = x(3 - ax - by) \\ \dot{y} = y(2 - x - y) \end{cases} \tag{1}$$

$$\begin{cases} 3 - ax - by = 0 \\ y = 0 \end{cases} \qquad \begin{cases} x = \dfrac{3}{a} \\ y = 0 \end{cases} \tag{2}$$

$$\begin{cases} x = 0 \\ 2 - x - y = 0 \end{cases} \qquad \begin{cases} x = 0 \\ y = 2 \end{cases} \tag{3}$$

Fixed points that are easy to find: $(0, 0) \quad \left(\dfrac{3}{a}, 0\right) \quad (0, 2)$

Specific case a=2, b=1 In this parameters configuration we can thus find:

1. $(0,0)$, $(0,2)$
2. $(1,1)$, $(\frac{3}{2},0)$

Those are the fixed points of our specific system.

1.1 Linear analysis

In this section we continue our analysis and look only at the linear model.

$$\begin{cases} \dot{x} = 3x \\ \dot{y} = 2y \end{cases} \implies \text{Fixed point:}(0,0)$$

$$A = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \implies \begin{cases} \tau = 5 \\ \Delta = 6 \\ \tau^2 - 4\Delta = 1 \end{cases}$$

Therefore, we say that (0,0) is an unstable node.

2 Finding the nullclines

The nullclines in our system with $a = 2$, $b = 1$ are:

$$\begin{cases} \dot{x} = x(3 - 2x - y) \\ \dot{y} = y(2 - x - y) \end{cases}$$

$$\left.\begin{array}{l} y = 3 - 2x \\ x = 0 \end{array}\right\} \implies \text{x-nullclines}$$

$$\left.\begin{array}{l} y = 2 - x \\ y = 0 \end{array}\right\} \implies \text{y-nullclines}$$

2.1 Finding the direction of the vector field along the nullclines

To find the direction of the vector field on the nullclines we study:

2.1.1 x-nullclines

1. To study the sign of the vector field we substitute $y = 3 - 2x$ in $y(2 - x - y)$:

$$(3 - 2x)(2 - x - (3 - 2x)) > 0 \implies -2x^2 + 5x - 3 > 0$$
$$\text{For } 1 < x < \frac{3}{2} \text{ we have positive values.}$$
$$\text{For the rest we have negative values.}$$

2. To study the sign of the vector field we substitute $x = 0$ in $y(2 - x - y)$:

$$y(2 - y) > 0 \implies (0, 2)$$
$$\text{For } 0 < y < 2 \text{ we have positive values.}$$
$$\text{For the rest we have negative values.}$$

2.1.2 y-nullclines

1. To study the sign of the vector field we substitute $y = 2 - x$ in $x(3 - 2x - y)$:

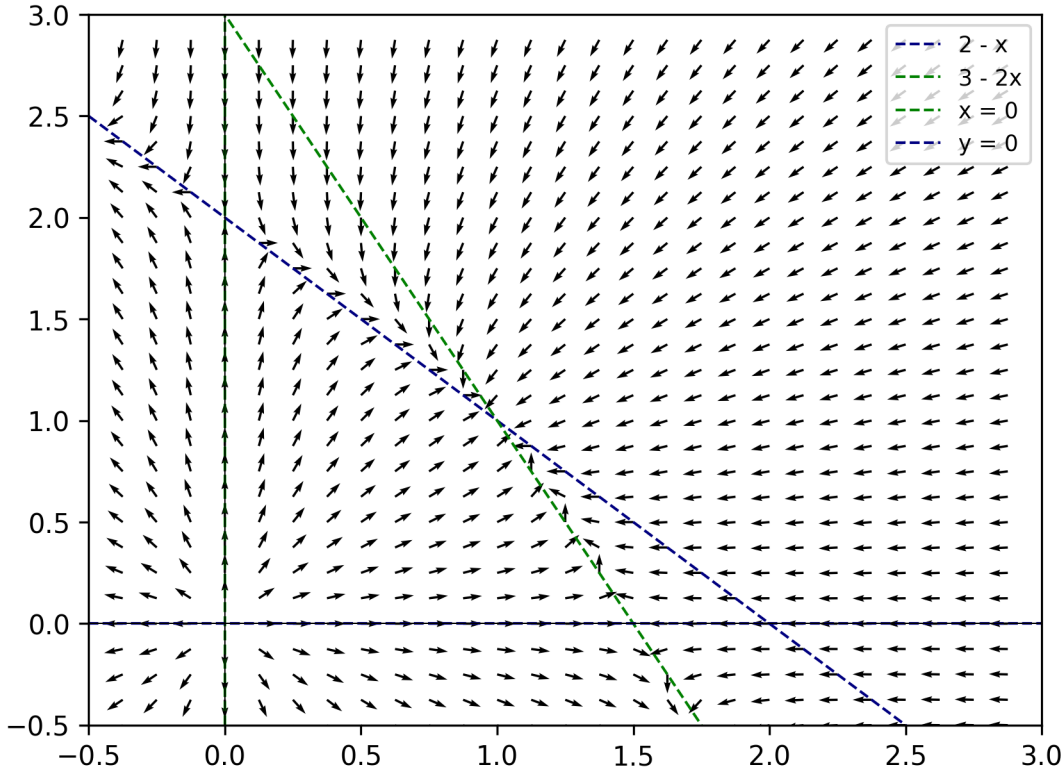
$$x(3 - 2x - (2 - x)) > 0 \implies x(1 - x) > 0$$
$$\text{For } 0 < x < 1 \text{ we have positive values.}$$
$$\text{For the rest we have negative values.}$$

2. To study the sign of the vector field we substitute $y = 0$ in $x(3 - 2x - y)$:

$$x(3 - 2x) > 0 \implies \left(0, \frac{3}{2}\right)$$
$$\text{For } 0 < x < \frac{3}{2} \text{ we have positive values.}$$
$$\text{For the rest we have negative values.}$$

2.2 Visualizing the nullclines

In this plot we visualize the direction of the vector field and plot the nullclines we found. The x-nullclines are shown in green whilst the y-nullclines are shown in blue.



In the figure we can indeed see that the vector field along the nullclines has only one component. Furthermore the vector are pointing in the direction we found in our analytic solution.

3 Numerical Analyses of the vector field

3.1 Import the necessary packages

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

We mainly need to import numpy and matplotlib.

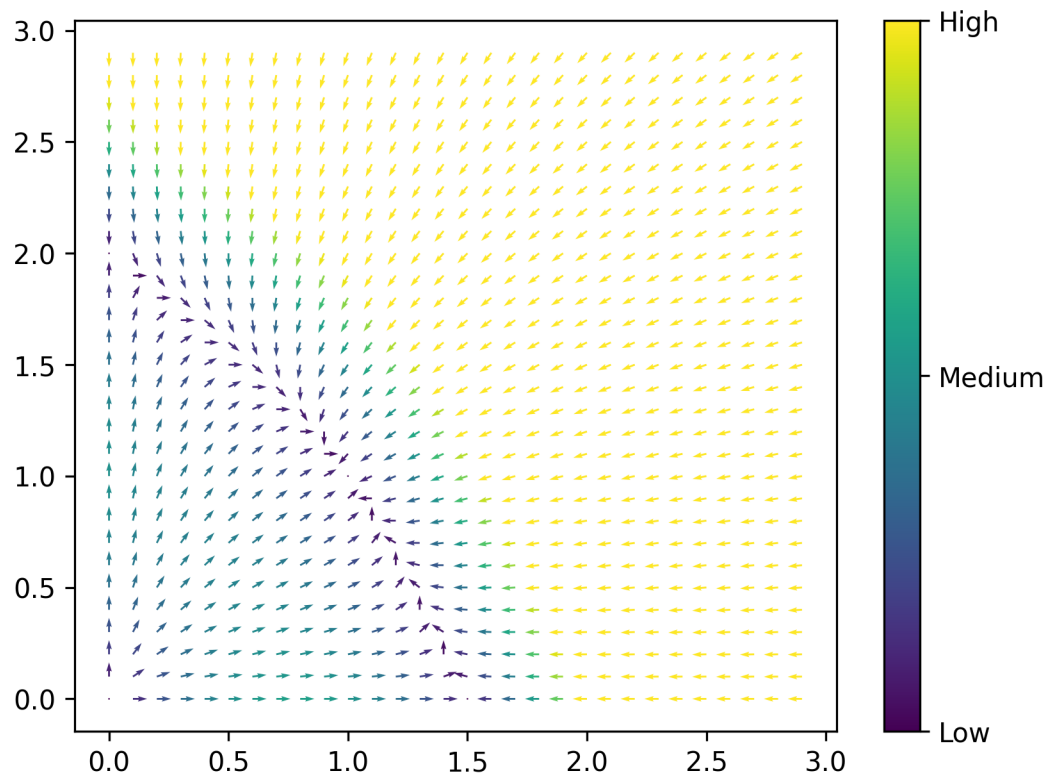
3.1.1 Define the differential equation

We define the differential equation as a function with a and b as parameters.

[2]:
```
def eq_diff(x,y,a,b):
    u = x * (3 - a * x - b * y)
    v = y * (2 - x - y)
    return (u,v)
```

The code below creates a graph that allows for interactive plotting of the parameters a and b using Python widgets, though in this example we are using it to create only one plot . The complete code can be found in this Jupyter notebook, inside the GitHub repository.

[3]:
```
# Import if we want to visualize the widgets
from ipywidgets import *

# Set the DPI of the figure
plt.rcParams['figure.dpi'] = 300

def update_plot(a=2, b=1, size=0.1):
    # Generate a vector of x and y
    x, y = np.meshgrid(np.arange(0, 3, size), np.arange(0, 3, size))
    u, v = eq_diff(x,y,a,b)

    # Compute the magnitude of each vector
    # Clip it so that we can get better coloring
    mag = np.clip(np.sqrt(u**2 + v**2), -10, 2)

    # Normalize the vectors
    norm = np.sqrt(u**2 + v**2)
    u = np.divide(u, norm, out=np.zeros_like(u), where=norm!=0)
    v = np.divide(v, norm, out=np.zeros_like(v), where=norm!=0)

    # Create the plot
    fig, ax = plt.subplots()

    # Set the colormap to 'cool'
    q = ax.quiver(x, y, u, v, mag, cmap='viridis')

    # Set the colorbar limits to the range of the magnitudes
    q.set_clim([mag.min(), mag.max()])

    # Set the colorbar ticks and labels
    cbar = plt.colorbar(q)
    cbar.set_ticks([mag.min(),(mag.min() + mag.max())/ 2, mag.max()])
    cbar.set_ticklabels(['Low', 'Medium', 'High'])

    plt.show()

update_plot()
```



The vector field plot showcases both the direction and relative magnitude of the vectors through a color coding method. This allows us to visually observe how the vector field has a smaller magnitude near the fixed points.

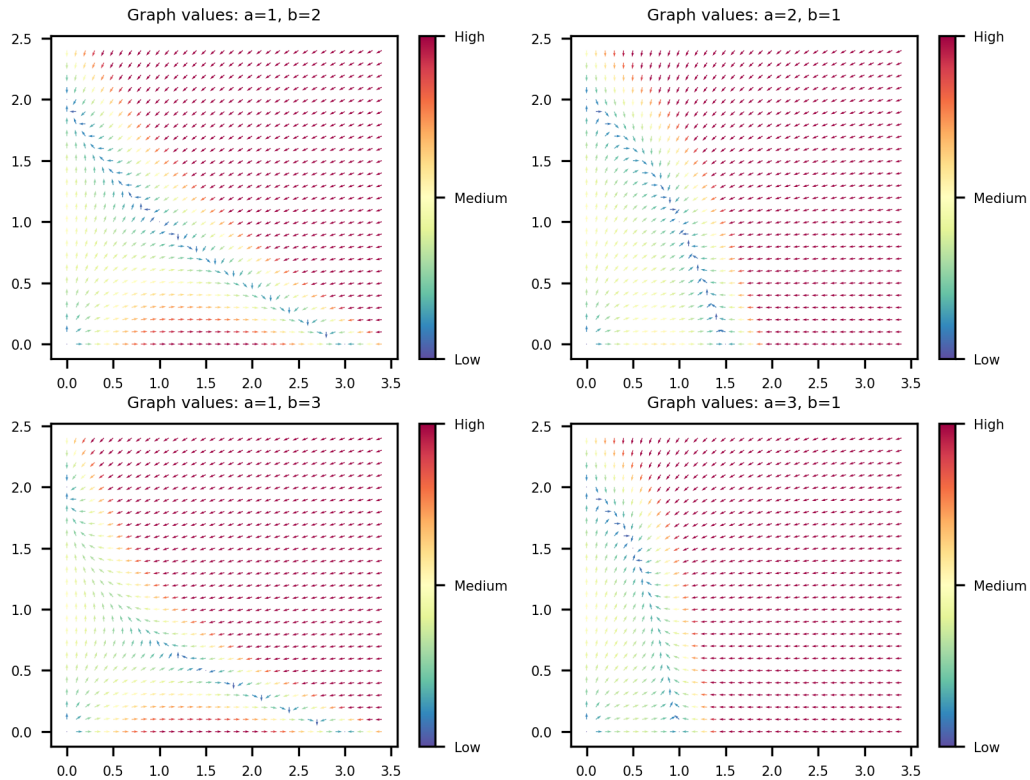
In the next section we will also start our numerical analysis of the vector field for different values of the parameters.

3.2 Vector fields for a few different parameters

Leveraging the function: *create_phase_diagrams*, that can also be found in the notebook, we create the following plots with this code:

[4]:
```
# Set the DPI of the figure
plt.rcParams['figure.dpi'] = 300
# Set the font size of the labels and title
plt.rcParams['font.size'] = 5

# Create a 2x2 grid of subplots
fig, axs = plt.subplots(2, 2)

a_vals = [1, 2, 1, 3]
b_vals = [2, 1, 3, 1]
```

```
create_phase_diagrams(a_vals,b_vals,axs)

# Display the plot
plt.show()
```



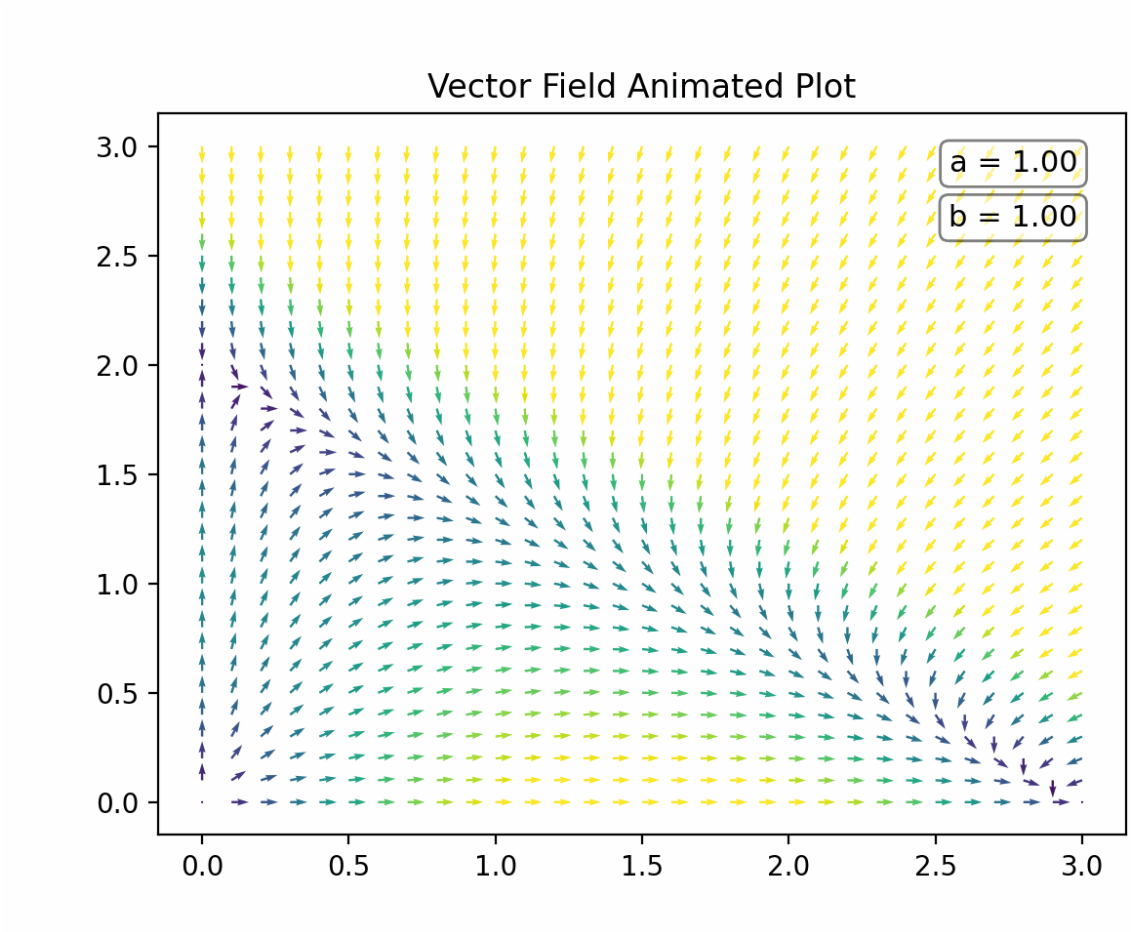
Indeed, we can observe that as long as the parameter b is greater than a, the vector field showcase what is expected from the *'principle of competitive exclusion'*. This principle states that one species drives the other to extinction.

Therefore, we expect trajectories starting below the stable manifold to result in the eventual extinction of the sheep (which is what we see), while those starting above lead to the eventual extinction of the rabbits.

On the other hand when a is greater than b, we observe a peculiar situation: there appear to be a stable fixed point where both species survive. This can be explained by the fact that the parameter representing the interference between the species is equal, preventing either species from completely dominating the other.

3.3 Animated vector field

Below we will also showcase an animated version of the vector field [1], where the parameters a and b vary. This gif has been created with: this python script



This animation can a be quite useful tool to visually demonstrates the impact of changing the vector field across a range of parameters.

4 Linearization of the flow near the fixed points

4.1 Find the Jacobian

To linearize the flow near fixed points, one can study the Jacobian of the system.

$$J = \begin{bmatrix} 3 - 2ax - by & -bx \\ -y & 2 - x - 2y \end{bmatrix} \tag{4}$$

[1] Not all pdf viewer support animated gif (Adobe Acrobat Reader does)

In particular lets study it for: a = 2, b = 1

$$\begin{cases} \dot{x} = x(3 - 2x - y) \\ \dot{y} = y(2 - x - y) \end{cases} \tag{5}$$

Therefore the Jacobian we get is:

$$J = \begin{bmatrix} 3 - 4x - y & -x \\ -y & 2 - x - 2y \end{bmatrix} \tag{6}$$

4.2 Evaluate the Jacobian in the fixed points

After deriving the general expression for the Jacobian, we evaluate it in the fixed points.

$$J_{(0,0)} = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \qquad J_{(1,1)} = \begin{bmatrix} -2 & -1 \\ -1 & -1 \end{bmatrix}$$

$$J_{(0,2)} = \begin{bmatrix} 1 & 0 \\ -2 & -2 \end{bmatrix} \qquad J_{(3/2,0)} = \begin{bmatrix} -3 & -\frac{3}{2} \\ 0 & \frac{1}{2} \end{bmatrix}$$

4.3 Relative Traces and Determinants

$$\begin{array}{lll} J_{(0,0)},\ \tau = \ \ 5 \quad \Delta = 6 \ \longrightarrow (\tau^2 - 4\Delta) = 1 & & \textbf{\textit{Unstable Node}} \\ J_{(1,1)},\ \tau = -3 \quad \Delta = 1 \ \longrightarrow (\tau^2 - 4\Delta) = 5 & & \textbf{\textit{Stable Node}} \\ J_{(0,2)},\ \tau = -1 \quad \Delta = -2 & & \textbf{\textit{Saddle node}} \\ J_{(3/2,0)},\ \tau = -\dfrac{5}{2} \quad \Delta = -\dfrac{3}{2} & & \textbf{\textit{Saddle node}} \end{array}$$

5 Numerical simulation for orbits near fixed points

Install numba and import it for just in time compilation

[5]:
```
%pip -q install numba
```

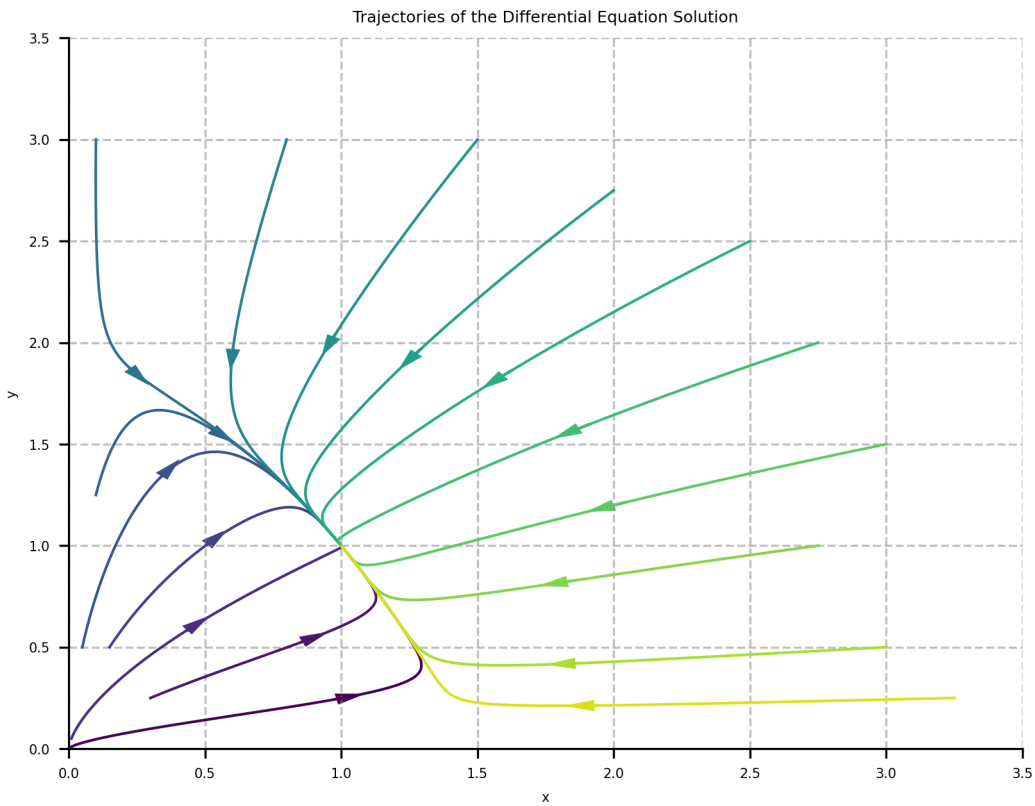
[6]:
```
from numba import *
import numpy as np

# import it from the library directory
from library.eqdiff_solver import *
# Define a function that takes 2D vector and a, b as parameters
```

```python
@njit(cache=True)
def f(x, a=2.0, b=1.0) -> np.ndarray[float64]:
    # Define the equations
    u = x[0] * (3 - a * x[0] - b * x[1])
    v = x[1] * (2 - x[0] - x[1])
    return np.array([u, v])
```

[7]:
```python
def draw_trajectories(f, dt=0.1, final_time=1, points=[[0, 0], [1, 1],␣
↪[2, 2]]):
    # Convert points to a NumPy array
    x0_array = np.array(points).reshape(-1, 2)

    # Compute the trajectories
    trajectories = vectorized_runge_kutta(f, x0_array, dt, final_time)

    # Create a plot
    fig, ax = plt.subplots()
    cmap = plt.get_cmap("viridis")

    # Iterate through the trajectories
    for i in range(len(x0_array)):
        x_traj = trajectories[i][:, 0]
        y_traj = trajectories[i][:, 1]
        color = cmap(i / len(x0_array))

        # Plot the trajectory
        ax.plot(x_traj, y_traj, color=color, linewidth=1)

        # Compute the cumulative distance along the trajectory
        distances = np.sqrt(np.diff(x_traj)**2 + np.diff(y_traj)**2)
        cumulative_distances = np.cumsum(distances)

        # Find the index where the distance is half of the total distance
        middle_index = np.argmax(cumulative_distances >=␣
↪cumulative_distances[-1] / 2)

        # Find the arrow direction using the middle two points
        dx = x_traj[middle_index] - x_traj[middle_index - 1]
        dy = y_traj[middle_index] - y_traj[middle_index - 1]

        # Add a small arrow at the middle of the trajectory
        ax.arrow(x_traj[middle_index - 1], y_traj[middle_index - 1], dx,␣
↪dy, head_width=0.04, head_length=0.08, fc=color, ec=color)
```

```
    # Set labels, title, and axis limits
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_xlim(0, 3.5)
    ax.set_ylim(0, 3.5)
    ax.set_title("Trajectories of the Differential Equation Solution")

    # Set the background color and remove unnecessary spines
    fig.patch.set_facecolor("white")
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)

    # Add grid lines
    ax.grid(True, linestyle="--", color="gray", alpha=0.5)
```

[8]:
```
# A few interesting starting points
starting_points = [[0.01, 0.01], [0.3, 0.25], [0.01, 0.05],
                   [0.15, 0.5], [0.05, 0.5], [0.1, 1.25], [0.1, 3],
                   [0.8,3], [1.5, 3], [2,2.75], [2.5,2.5],
                   [2.75, 2], [3,1.5], [2.75, 1], [3,0.5], [3.25, 0.25]]

draw_trajectories(f=f, dt = 0.01, final_time = 15,
                                          points = starting_points)
```



Indeed, from this plot, we can see that we have found stable points that seem to agree with the computed trajectory. Furthermore the plots provide us with an idea of the manifold's position. Additionally, the trajectory can help us speculate that some of the manifold we have found by linearizing the differential equation merge. Although we will take a deeper look at the manifold position in the following paragraph, this is generally what we would expect.

6 Finding the manifold near the fixed points

To find the manifolds near the fixed points we will look at the eigenvalues and eigenvectors of the Jacobian amtrixes evaluated in the respective fixed points:

6.1 Eigenvalues and eigenvectors computations

6.1.1 $J_{(0,0)}$ manifolds

$$\ker(A - 3I) = \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} \Longrightarrow \langle (1,0) \rangle$$
$$\ker(A - 2I) = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \Longrightarrow \langle (0,1) \rangle$$

6.1.2 $J_{(1,1)}$ manifolds

$$\ker\left(A+\frac{3-\sqrt{5}}{2}I\right)=\begin{bmatrix}-\frac{1+\sqrt{5}}{2} & -1\\ -1 & \frac{1-\sqrt{5}}{2}\end{bmatrix}\longrightarrow\begin{bmatrix}1 & \frac{-1+\sqrt{5}}{2}\\ -\frac{1+\sqrt{5}}{2} & -1\end{bmatrix}\longrightarrow\begin{bmatrix}1 & \frac{-1+\sqrt{5}}{2}\\ 0 & 0\end{bmatrix}$$

$$\Longrightarrow\langle(1-\sqrt{5},2)\rangle$$

$$\ker\left(A+\frac{3+\sqrt{5}}{2}I\right)=\begin{bmatrix}\frac{-1+\sqrt{5}}{2} & -1\\ -1 & \frac{1+\sqrt{5}}{2}\end{bmatrix}\longrightarrow\begin{bmatrix}1 & -\frac{1+\sqrt{5}}{2}\\ \frac{-1+\sqrt{5}}{2} & -1\end{bmatrix}\longrightarrow\begin{bmatrix}1 & -\frac{1+\sqrt{5}}{2}\\ 0 & 0\end{bmatrix}$$

$$\Longrightarrow\langle(1+\sqrt{5},2)\rangle$$

6.1.3 $J_{(0,2)}$ manifolds

$$\ker(A-I)=\ker\begin{bmatrix}0 & 0\\ -2 & -3\end{bmatrix}\Longrightarrow\langle(-3,2)\rangle$$

$$\ker(A+2I)=\ker\begin{bmatrix}3 & 0\\ -2 & 0\end{bmatrix}\Longrightarrow\langle(0,1)\rangle$$

6.1.4 $J_{(3/2,0)}$ manifolds

$$\ker(A-\frac{1}{2}I)=\begin{bmatrix}-\frac{7}{2} & -\frac{3}{2}\\ 0 & 0\end{bmatrix}\Longrightarrow\langle(-3,7)\rangle$$

$$\ker(A+3I)=\begin{bmatrix}0 & -\frac{3}{2}\\ 0 & \frac{7}{2}\end{bmatrix}\Longrightarrow\langle(1,0)\rangle$$

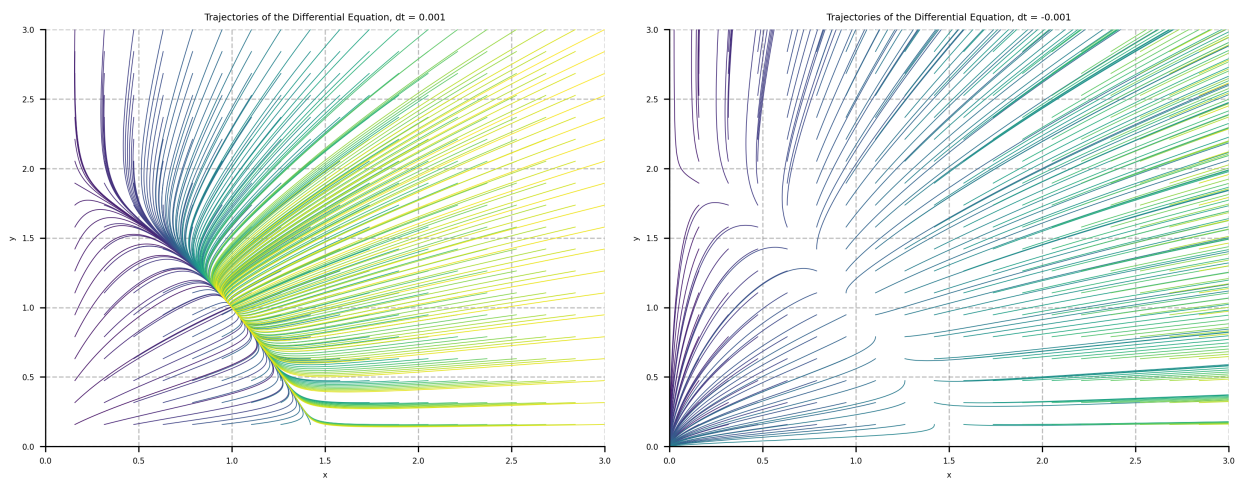
---

7 Analysing the non linear extension of the manifold

In this section we well better visualize the non linear extension of the manifold, by leveraging numerical integration . We used a custom implementation of Runge Kutta at the forth order (jit compiled with numba) that can be found in this python file.

7.1 Manifold visualization forward and backward in time

[9]:
```
# using phase_diagram_trajectories defined inside library.eqdiff_solver␣
↪f_solver
phase_diagram_trajectories(f, dt=0.001, final_time=10, num=20, start=-0,␣
↪end=3)

phase_diagram_trajectories(f, dt=-0.001, final_time=-10, num=20,␣
↪start=-0, end=3)

# Close all open figures
plt.close('all')
```



As we can see in the first image the some of the manifolds are indeed merging, this is expected since this is the simulation of the non-linear differential equation.

In this second image we showcase what happens if we go backward in time. Indeed as expected we get a reversal of the manifold's stability. We believe this could also be an useful way to visualize our differential equation from another prospective.

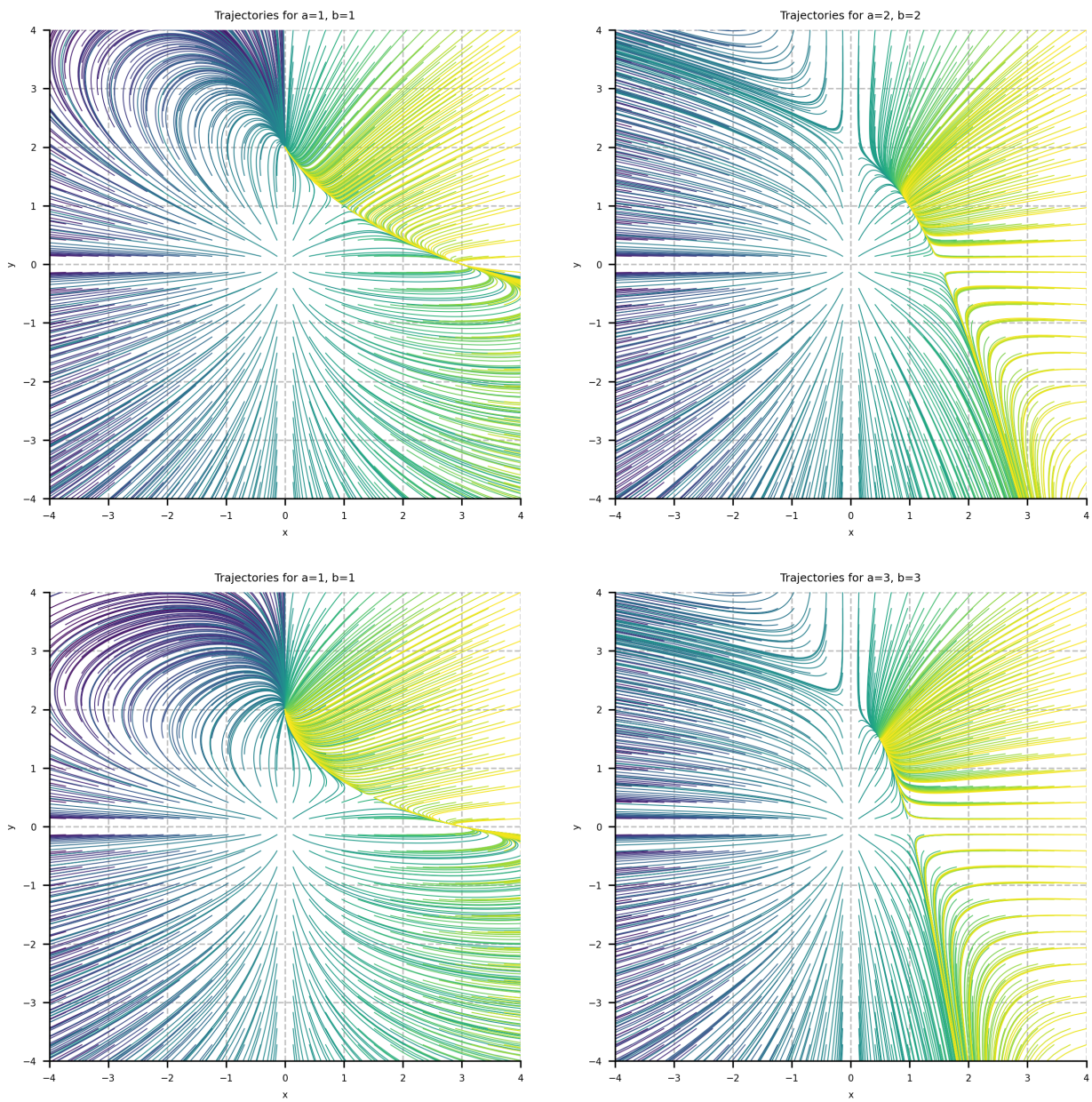
8 Concluding our analyses

To conclude our analyses we create a few more plots for different values of a and b. These plots allow us to visualize the trajectories and gain a better understanding of how the manifolds bend outside of our region of interest (the positive part).

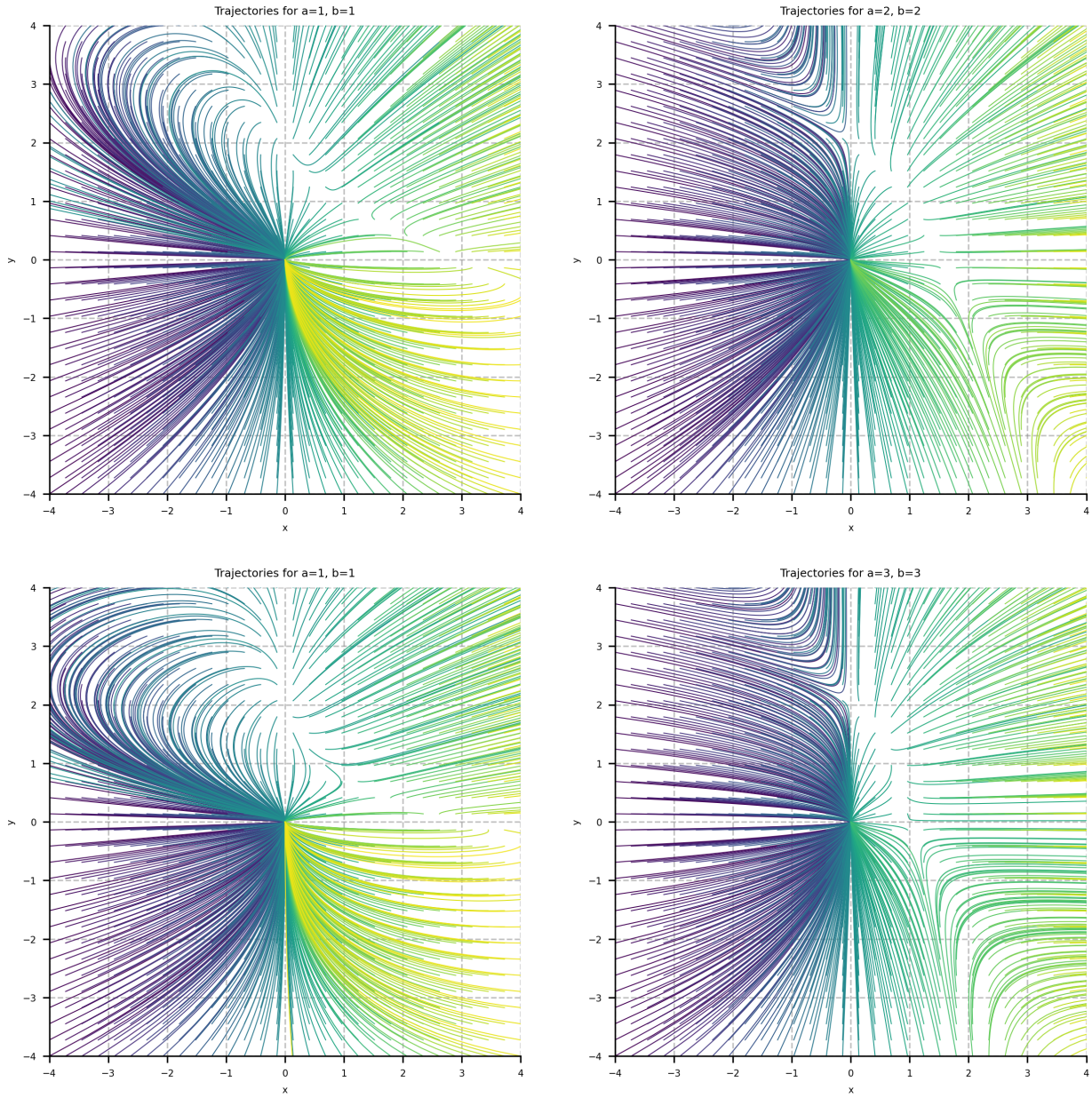
8.1 Visualization of the entire differential equation

Finally, we present these plots to provide a more comprehensive view of the behavior of the system. Furthermore, we have also added 4 plots where we instead go backwards in time to see the difference.

[10]:
```
# a and b values
list_ab = [(1, 2), (2, 1), (1, 3), (3, 1)]
# Phase diagram forward in time
multiple_phase_diagram_trajectories(f, list_ab, dt=0.01, final_time=10,␣
↪num=30, start=-4, end=4)
# Phase diagram reversing time
multiple_phase_diagram_trajectories(f, list_ab, dt=-0.01, final_time=-10,␣
↪num=30, start=-4, end=4)
# Close all open figures
plt.close('all')
```



In these phase portraits, we can observe that for various values of a and b, the manifolds bend in response to the interaction between the different fixed points. Additionally, we notice that the trajectories are fast when they are far from the fixed point, but they slow down as they approach it.[2]



Finally, by running Runge-Kutta backward in time instead of forward we can indeed see that the manifold's stability changes.

[2]Other visualization of system of differential equations for more values of a and b are inside the manim animation