DOING PHYSICS WITH PYTHON

[2D] NON-LINEAR DYNAMICAL SYSTEMS THEORETICAL CONSIDERATIONS

Ian Cooper
matlabvisualphysics@gmail.com

DOWNLOAD DIRECTORIES FOR PYTHON CODE

Google drive

GitHub

Jason Bramburger
Hyperbolic Fixed Points - Dynamical Systems | Lecture 16

https://www.youtube.com/watch?v=P5zQ1uUysGA&t=942s

Stephen Lynch

Dynamical Systems with Applications using Python

This paper considers the theory behind [2D] non-linear dynamical systems. The Python Code solves a pair of non-linear ODEs in x and y. The solution gives the time evolution of the two variables x and y, the phase portrait (quiver plot and streamplot), the nullclines, vector fields, eigenvectors, and find and classify critical points in the phase plane.

INTRODUCTION

This article considers how Python can be used to solve [2D] non-linear dynamical systems. The [2D] systems are described by a pair of ordinary differential equations (ODEs) in *x* and *y*. The ODEs are solved numerically using the Python function **odeint**. The solutions for *x* and *y* are displayed graphically as time evolution plots and phase portrait plots. For a dynamical system in [2D] dimensions the ODEs can be expressed as

$$\dot{x} = f(x, y) \qquad \dot{y} = g(x, y)$$

The phase portrait is a [2D] figure showing how the qualitative behaviour of system is determined as *x* and *y* vary with *t*. With the appropriate number of trajectories plotted, it should be possible to determine where any trajectory will end up from any given initial condition.

The **direction field** or **vector field** gives the gradients *dy*
dx. The slope of the trajectories at each point in the vector field is given by

$$\frac{dy}{dx} = \frac{\dot{y}}{\dot{x}}$$

The contour lines for which *dy/dx* is a constant are called **isoclines**.

The contour lines for which *dy/dt* = 0 and *d/xdt* = 0 are called ***nullclines***. Isoclines may be used to help with the construction of the phase portrait. For example, the nullclines for which $\dot{x} = 0$ and $\dot{y} = 0$ are used to determine where the trajectories have vertical and horizontal tangent lines, respectively. If $\dot{x} = 0$, then there is no motion horizontally, and trajectories are either stationary or move vertically. When $\dot{y} = 0$, then there is no motion vertically, and trajectories are either stationary or move horizontally.

An **equilibrium** occurs at **critical points** or **fixed points** (x_e, y_e) of a dynamical system generated by system of ordinary differential equations (ODEs) where a solution that does not change with time.

$$\dot{x} = f(x_e, y_e) = 0 \qquad \dot{y} = g(x_e, y_e) = 0$$

FIX POINT STABILITY

The stability of typical equilibria for smooth ODEs is determined by the sign of real part of the eigenvalues of the Jacobian matrix. These eigenvalues are often referred to as the eigenvalues of the equilibrium. In [2D] systems the Jacobian matrix is

$$\mathbf{J}(x_e, y_e) = \begin{pmatrix} \partial f / \partial x & \partial f / \partial y \\ \partial g / \partial x & \partial g / \partial y \end{pmatrix}_{|x=x_e, y=y_e}$$

and has two eigenvalues, which are either both real or complex-conjugates. The eigenvalues and eigenfunctions can be found using the Python function **eig**. A critical point is called hyperbolic if the real part of the eigenvalues of the Jacobian matrix $\mathbf{J}(x_e, y_e)$ is nonzero. If the real part of either of the eigenvalues of the Jacobian is equal to zero, then the critical point is called non-hyperbolic. The Jacobian matrix is derived using a Taylor's expansion near a fixed point where 2nd order and above are ignored. We say that the eigenvalues of a system are linearized around a fixed point

A **hyperbolic fixed** point is a type of equilibrium point in a dynamical system where the system's behaviour is strongly expanding and contracting in different directions, with no centre directions where it neither expands nor contracts significantly. This characteristic is determined by the eigenvalues of the system's Jacobian matrix where the real parts of all eigenvalues must be non-zero (Re(λ) $\neq$ 0). Hyperbolic fixed points are considered robust, meaning they are stable under small perturbations of the system. Near a hyperbolic fixed point, the phase plane of the nonlinear system is topologically equivalent to the phase plane of its linearization. The Hartman-Grobman theorem guarantees that the behaviour near a hyperbolic fixed point in a non-linear system resembles the behaviour of its linearized counterpart, making it easier to study.

For a hyperbolic fixed point is robust as a small change to the system will likely result in a new, hyperbolic fixed point close to the original one. However, for a non-hyperbolic fixed point, which has an eigenvalue with its real part of zero, it lacks this robustness and can disappear or change behaviour with small perturbations. The robustness of hyperbolic fixed points makes them reliable indicators of the local behaviour of a system, simplifying the analysis of complex non-linear systems.

Hyperbolic fixed points are important since they define directions where the system strongly expands or contracts, and these directions are crucial for understanding the overall dynamics. :

The eigenvalues of a system linearized around a fixed point can determine the stability behaviour of a system around the fixed point. The particular stability behaviour depends upon the existence of real and imaginary components of the eigenvalues, along with the signs of the real components and the distinctness of their values. That is, the eigenvalues give us the **local stability** around the fixed point.

To analyse the stability of a fixed point in a [2D] dynamical system, you must first find the fixed point and then linearize the system by calculating the Jacobian matrix at that fixed point.

After finding this stability, you can show whether the system will be stable, unstable and undamped fluctuations, or unstable system in which the amplitude of the fluctuation is always increasing (such system will not be able to return to steady state). For the undamped situation, the constant fluctuation will be hard on the system and can lead to equipment failure or with the ever-increasing amplitude of the fluctuations catastrophic failure will be the result.

The eigenvalues of the Jacobian matrix may be real or complex and this determines the local nature of a fixed point. in the local region.

Real eigenvalues $\lambda_0 \;\; \lambda_1$

Complex eigenvalues $\lambda_0 = a - b\,j \quad \lambda_1 = a + b\,j \qquad b \neq 0$
where a and b are real scalars

Real eigenvalues $\lambda_0 \;\; \lambda_1$

For real eigenvalues, in the locality of a fixed point the flow is either towards (stable) or away (unstable) from the fixed point. There is no oscillatory motion.

- Both eigenvalues are distinct and negative:
  $\lambda_0 \neq \lambda_1 \;\; \lambda_0 < 0 \quad \lambda_1 < 0$

Fixed point is **stable** - small perturbations will decay, and the system will return to the fixed point. A solutions starting near a stable fixed point will approach it over time (source node with vector field pointing inward).

- Both eigenvalues are distinct and positive:

  $\lambda_0 \neq \lambda_1 \quad \lambda_0 > 0 \quad \lambda_1 > 0$

Fixed point is **unstable**, and small perturbations will grow, causing the system to move away from the fixed point. A solution starting near an unstable fixed point will move away from it (source node with vector field pointing outward).

One eigenvalue negative and one positive: $\lambda_0 \neq \lambda_1 \quad \lambda_0 < 0 \quad \lambda_1 > 0$

Unstable saddle node: a point where a series of minimum and maximum points converge at one area in the vector field, without hitting the point. [3D] surface plot the function looks like a saddle.

- Repeated positive eigenvalues $\lambda_0 = \lambda_1 > 0$

ONE linearly independent eigenvector. **Unstable** fixed point is a **degenerate node**.

- Repeated positive eigenvalues $\lambda_0 = \lambda_1 > 0$

TWO linearly independent eigenvector. **Unstable** fixed point is a singular **degenerate node** **(source)**.

- Repeated negative eigenvalues $\lambda_0 = \lambda_1 < 0$

Fixed point is a **stable sink**.

Complex eigenvalues

$\lambda_0 = a - b\,j \quad \lambda_1 = a + b\,j \qquad b \neq 0$

where a (real part) and b (imaginary part) are real scalars

The system will be oscillatory and the stability of oscillating systems is determined entirely by examination of the real part.

- Positive real part $a > 0$ Focus: **unstable oscillator** with a spiral whose amplitude increases with time. This can be visualized as a vector tracing a spiral away from the fixed point in the phase portrait. The plot of response with time of this situation would look sinusoidal with ever-increasing amplitude. This situation is usually undesirable as any external disturbance will result in the system itself not returning to the steady-state.

- Negative real part $a < 0$ Focus: **stable oscillator** with a spiral whose amplitude decreases with time. This can be visualized as a vector tracing a spiral toward the fixed point in the phase portrait. The plot of response with time of this situation would look sinusoidal with ever-decreasing amplitude. This system is stable since steady state will be reached even after a disturbance to the system. The oscillation will bring the system back to the fixed point. It is important to know that having all negative real parts of eigenvalues is a necessary and sufficient condition of a stable system.

- Zero real part $a = 0$ (eigenvalue purely imaginary) Centre: **stable** undamped oscillator. This can be visualized in two dimensions as a vector tracing an ellipse around a point (centre) in the phase portrait. The plot of response with time would look sinusoidal.

Python Code for [2D] non-linear dynamical systems

- Define the system and system dimensions
- Solve the system equations to find the fixed points.
- Solve the system equations to find the x and y nullclines.
- Calculate the Jacobian matrix and find its eigenvalues and eigenvectors. Determine the stability of the fixed points.
- Plot the vector field in the phase portrait (x vs y plot) using a streamplot or quiver plot. Add to the plot the fixed points and the nullclines.
- Specify the initial condition, $x(0)$ and $y(0)$. Solve the ODEs to give the trajectory using the Python function **odeint**. Plot the time evolution of the solutions, t vs x(t) and t vs $y(t)$. The trajectory can also be added to the phase portrait.