Learning Dynamical Systems

The true ODE that governs the dynamical system is given as:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \mathbf{f}(x, y) = \begin{bmatrix} -0.1 & 2 \\ -2 & -0.1 \end{bmatrix} \begin{bmatrix} x^3 \\ y^3 \end{bmatrix} .$$

In this project, we will use both SINDy and Neural ODE to learn $\mathbf{f}(x, y)$. Note this system is autonomous, so $\mathbf{f}$ does not depend on t.

1. Using the initial condition: $x(0) = 0.1$, $y(0) = 0.1$, generate training data by numerically solving the ODE equation. (If you check the eigenvalues of the Jacobian matrix, the real parts are negative, so this is a stable system. However, when you generate the data using an explicit scheme, you would need to make sure the time step is small enough so that your numerical solution does not blow up.)
2. Use SINDy to learn the dynamical system:
   a) Construct the library of candidate functions of polynomial functions up to 5^{th}-order (function library can be constructed via custom library module – `https://pysindy.readthedocs.io/en/latest/api/pysindy.feature_library.html#module-pysindy.feature_library.custom_library`)
   b) Use (i) $\mathbf{x}$ and $\dot{\mathbf{x}}$ as data, and (ii) only use $\mathbf{x}$ as data (In SINDy code, if $\dot{\mathbf{x}}$ data is missing, they will be calculated using finite difference.) From this comparison, you can check if the method is robust to data noise.
   c) Report the fitted model.
   d) Construct the library of candidate functions of polynomial functions up to 5^{th}-order **without** x^3 and y^3 to fit a SINDy model. Report the fitted model. Solve the ODE with the fitted model numerically with the same initial condition and compare (i) the trajectories of system state variables with respect to time and (ii) the trajectory in phase space, against the training data. Solve the ODE with the trained model numerically with a different initial condition (e.g., $x(0) = 0.15$, $y(0) = 0.2$), and compare (i) the trajectories of system state variables with respect to time and (ii) the trajectory in phase space, against the solution solved from the exact model with the same initial condition.
3. Use Neural ODE to learn the dynamical system:
   a) Starting from the settings of the standard Neural ODE example, tune the network architecture and other settings to achieve small training loss (≤ 0.01 for this problem). Solve the ODE with the trained model numerically with the same initial condition and compare (i) the trajectories of system state variables with respect to time and (ii) the trajectory in phase space, against the training data. Solve the fitted model numerically with a different initial condition (e.g., $x(0) = 0.15$, $y(0) = 0.2$), and compare (i) the trajectories of system state variables with respect to time and (ii) the trajectory in phase space, against the solution solved from the exact model.
   b) Add certain level of noise (Gaussian with zero mean and certain magnitude of variance) in the training data and use the new noisy data to train the neural network until the training loss ≤ 0.01. Solve the ODE with the trained model numerically with a different initial condition (e.g., $x(0) = 0.15$, $y(0) = 0.2$), and compare (i) the trajectories of system state variables with respect to time and (ii) the trajectory in phase space, against the solution solved from the exact model with the same initial condition.