Learning non-linear model dynamics from data using Dynamic Mode Decomposition

December 6, 2022

Ayush Gaggar, John Peiffer
Video Link

1 Introduction

Modeling nonlinear dynamic systems can be a quite difficult, even with the knowledge of a non-linear system. Further, nonlinear systems can be functions of unknown inputs, or impossible to numerically quantify. For example, attempting to model brain functionality as a set of inputs is impossible with current science; on the other hand, turbulent flow is an example of a nonlinear system that is imposible to numerically solve. However, this course has demonstrated that it is not necessary to devolop a full model of the system to predict its behaivor – *simply observing it gives us data to train different machine learning paradigms that approximate the unknown model's behavior.*

This project uses Dynamic Mode Decomposition (DMD) to exploit low-dimensionality in recorded data without having to learn a set of equations governing some example non-linear systems. We then experiment with different inputs to the DMD algorithm and their effect on how well the machine learning algorithm can predict future states of the system.

2 Problem Setup

Assume we sample N spatial points over M regurarly spaced sample points.
The goal is to learn a matrix A such that ***linearly*** multiplying the current state of the measurements by A gives a good approximation of the next state. This can be expressed as

$$\vec{x}_{j+1} = \vec{A}\vec{x}_j$$

A is also known as the Koopman operator [1], which is a *linear* operator from one timepoint to the next.
We now concatanate all the $M-1$ snapshots into a single vector X

$$\vec{X} = [\vec{x}_1, \vec{x}_2, ..., \vec{x}_{M-1}]$$

and also create another matrix X' with states from 1 to M

$$\vec{X}' = [\vec{x}_2, \vec{x}_3, ..., \vec{x}_M]$$

To move from one step in time to another, using linear algebra, giving

$$\vec{X}' = \vec{A}\vec{X}$$

We can then solve for

$$\vec{A} = \vec{X}'\vec{X}^{\dagger}$$

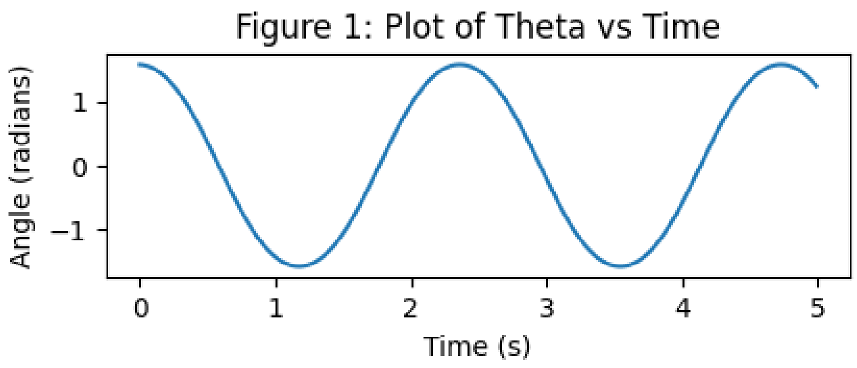
where $\vec{X}^{\dagger}$ is the psuedo-inverse of $\vec{X}$. While there may be $M-1$ matrices linearly transforming a state $\vec{x}$, the psuedo-inverse gives the least-squares fit that minimizes $\vec{A}$ across all timesteps. This method is remisicent of the analytical solution of linear regression with a least squares cost function, as done in class.

Finally, it is worth noting that the real DMD algorithm does not use the full $\vec{X}$. Instead, DMD uses Singular Value Decomposition to calculate $\vec{A}$ in a lower dimensional space of the data [2]. This project makes use of the python library PyDMD for this.

3 Single Pendulum Modeling

In the helper function we model the dynamics of a single pendulum system using Euler-Lagrange formulation. The important result of this is the variable *traj* which represents the state of the pendulum (θ) as a function of time. This trajectory will serve as the measurements from the "unknown" system that we will feed in to the DMD algorithm to learn the dynamics.

```
[16]: import sympy as sym
import numpy as np
import matplotlib.pyplot as plt
from helper import simulate, dyn, integrate,animate_single_pend

# initial conditions
x0 = np.array([np.pi/2, 0])
T = 5. ; dt = 0.01

traj = simulate(dyn, x0, [0, T], dt, integrate) # get trajectory of pendulum␣
 ↪over 5 seconds
plt.rcParams["figure.figsize"] = (5,1.5)
plt.plot(np.arange(int(T/dt))*dt, traj[0], label='theta')
plt.title(r'Figure 1: Plot of Theta vs Time');plt.xlabel("Time (s)")
plt.ylabel("Angle (radians)");plt.show()
```

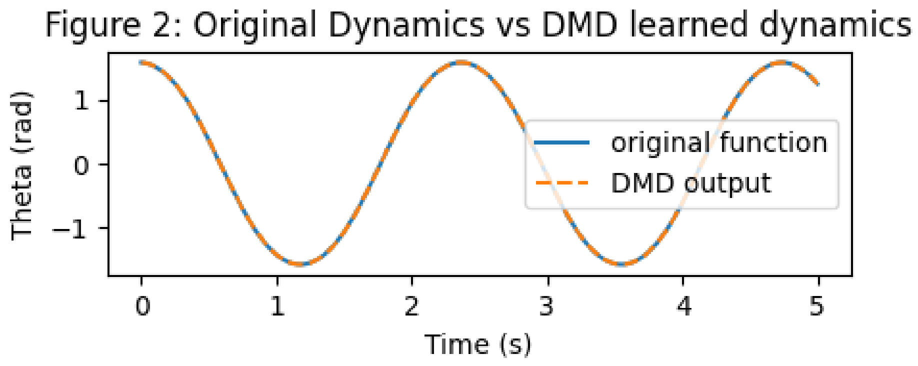


This plot represents a single pendulum (hanging downwards is its 0 location). It starts at a configuration of 90 degrees (horizontal), swings down past 0, and then to its other horizontal position (-pi/2) radians. The pendulum will never stop oscillating as there is no friction or damping. Figure 1 represents this data as we would measure from an unknown system.

4 Dynamic mode decomposition to learn and predict dynamics

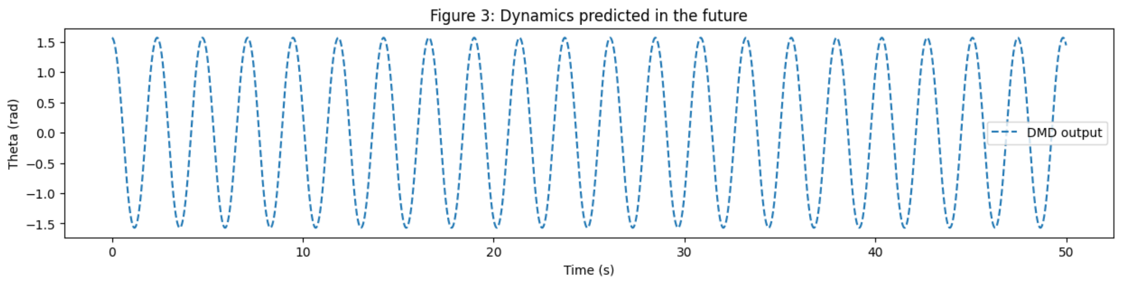
Here we input our "training" data which is the angle of the pendulum as it swung for 5 seconds (Figure 1). We use `HODMD` which is Higher Order DMD (more appropriate for periodic inputs) to fit the data and then plot the states predicted by the DMD vs the original function. Figure 2 shows that the DMD is able to learn the dynamics of the system!

[17]:
```python
from pydmd import HODMD

ydata = traj[0, :]  #training data (angle)
tdata = np.linspace(0, T, int(T/dt))

hodmd = HODMD(exact=True, opt=True, d=30).fit(ydata)
hodmd.original_time['dt'] = hodmd.dmd_time['dt'] = tdata[1] - tdata[0]
hodmd.original_time['t0'] = hodmd.dmd_time['t0'] = tdata[0]
hodmd.original_time['tend'] = hodmd.dmd_time['tend'] = tdata[-1]

plt.plot(tdata, traj[0], '-', label='original function')
plt.plot(tdata, hodmd.reconstructed_data[0].real, '--', label='DMD output')
plt.legend();plt.xlabel('Time (s)')
plt.ylabel('Theta (rad)')
plt.title('Figure 2: Original Dynamics vs DMD learned dynamics')
plt.show()
```



In the below block we extend the time to 50 seconds and plot the predicted dynamics from DMD. The algorithm predicts the pendulum will continue to swing back and forth indefinitely, which makes sense as our model has no friction and damping.

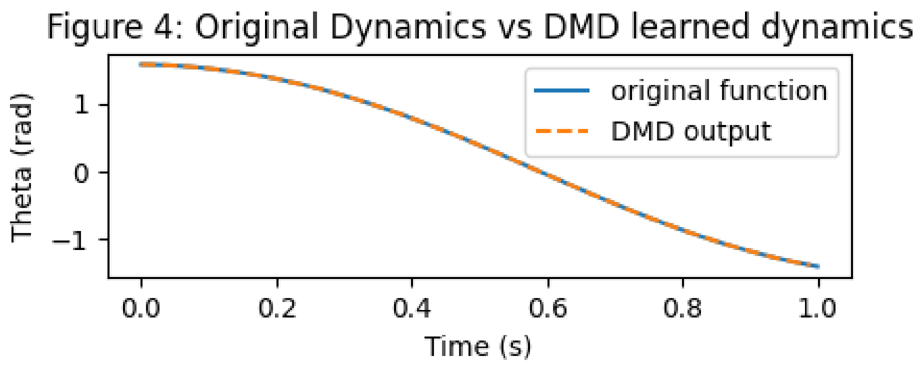
[18]:
```
hodmd.dmd_time['tend'] = 50     # extend DMD

plt.rcParams["figure.figsize"] = (15,3)
plt.plot(hodmd.dmd_timesteps, hodmd.reconstructed_data[0].real, '--',␣
 ↪label='DMD output')
plt.xlabel('Time (s)');plt.ylabel('Theta (rad)')
plt.title('Figure 3: Dynamics predicted in the future')
plt.legend();plt.show()
```



5 Predicting dynamics from incomplete data

Here we will only train DMD with a quarter of a full swing's worth of data (from initial to zero configuration). In Figure 4 we see that it can replicate the configuration of the data it was trained on – this is expected. Just as in other ML models, the error is low for training data if the model is sufficiently complex (which DMD is).

[19]:
```
ydata = traj[0, :100]  #training data (angle)
T=1
tdata = np.linspace(0, T, int(T/dt))

hodmd = HODMD(exact=True, opt=True, d=30).fit(ydata)
hodmd.original_time['dt'] = hodmd.dmd_time['dt'] = tdata[1] - tdata[0]
hodmd.original_time['t0'] = hodmd.dmd_time['t0'] = tdata[0]
hodmd.original_time['tend'] = hodmd.dmd_time['tend'] = tdata[-1]

plt.rcParams["figure.figsize"] = (5,1.5)
plt.plot(tdata, ydata, '-', label='original function')
plt.plot(tdata[:-1], hodmd.reconstructed_data[0].real, '--', label='DMD output')
plt.legend()
plt.xlabel('Time (s)');plt.ylabel('Theta (rad)')
plt.title('Figure 4: Original Dynamics vs DMD learned dynamics')
plt.show()
```



The following cell uses the partially trained data to do prediction. The upswing immediately following the training data does a very nice job of predicting what the model will do. In fact, it works very nicely on predicting several more oscillations. However, around 15 seconds, the model becomes unstable and quickly heads towards infinity by 20 seconds. In contrast, the model trained on multiple periods was able to continue prediction for 50 seconds (and likely indefinitely). This illustrates the importance of having a diverse and representative training set in any machine learning problem!

```
[20]: hodmd.dmd_time['tend'] = 18 # extend DMD

      plt.rcParams["figure.figsize"] = (15,3)
      plt.plot(hodmd.dmd_timesteps[:-1], hodmd.reconstructed_data[0].real, '--',␣
        ↪label='DMD output')
      plt.xlabel('Time (s)');plt.ylabel('Theta (rad)')
      plt.title('Figure 5: Dynamics predicted from incomplete data')
      plt.legend();plt.show()
```

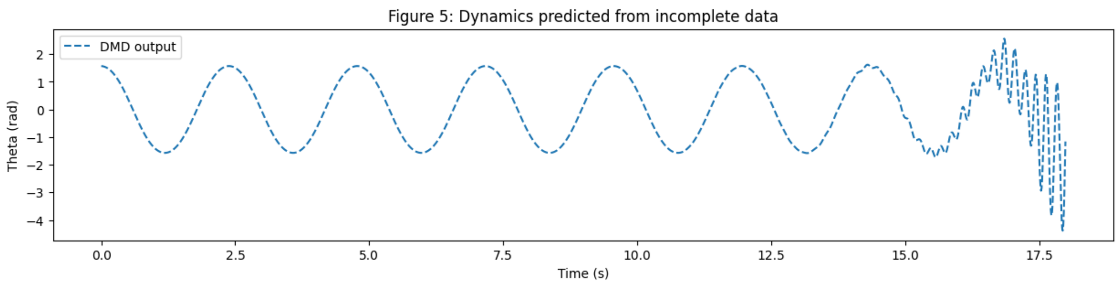


Figure 5: Dynamics predicted from incomplete data
DMD output
Theta (rad)
Time (s)