Dynamic Mode Decomposition and Koopman Theory

Sourya Dey

Galois, Inc.

Abstract

Dynamic Mode Decomposition (DMD) is a technique to approximate generally non-linear dynamical systems using linear techniques, which are better understood and easier to analyze. Koopman theory extends DMD by transforming the original system into a new domain which facilitates linearization. This is a technical report on DMD and Koopman theory, with primary focus on explaining the underlying mathematics in clear and concise form. We include dimensions of vectors and marices, and step-by-step derivations of equations in order to assist the user in easily comprehending these concepts. This report will also enable users to implement DMD and Koopman theory in code.

Acknowledgements

The material in this report has been compiled from relevant portions of [1, 2].

Notation

Matrices are written in bold capital letters, e.g. $\boldsymbol{A}$, and vectors in bold small letters, e.g. $\boldsymbol{a}$. Dimensions of quantities are written under them, e.g. $\underset{m \times n}{\boldsymbol{A}}$ denotes a matrix $\boldsymbol{A}$ with m rows and n columns.

1 Dynamic Mode Decomposition

Consider a system whose state at index k is described by a vector $\boldsymbol{x}_k$. Essentially $\boldsymbol{x}_k$ is a vector of observed values sampled at index k. The index can be time, or any other independent variable. Similarly, the sampled state at index $k+1$ is $\boldsymbol{x}_{k+1}$. $\boldsymbol{x}$ has a finite number of dimensions equal to the number of observed values (assume n). This means that it can be expressed as a linear combination of n basis vectors.

The transition function to take $\boldsymbol{x}_k$ to $\boldsymbol{x}_{k+1}$ is F, i.e.:

$$\boldsymbol{x}_{k+1} = F(\boldsymbol{x}_k) \tag{1}$$

$F(\cdot)$ is, in general, a non-linear function, i.e. $F(a\boldsymbol{x}_1 + b\boldsymbol{x}_2) \neq aF(\boldsymbol{x}_1) + bF(\boldsymbol{x}_2)$.[1] **Our goal is to find a matrix $\boldsymbol{A}$ such that the transition dynamics become linear**, i.e.:

$$\underset{n\times 1}{\boldsymbol{x}_{k+1}} \approx \underset{n\times n}{\boldsymbol{A}}\,\underset{n\times 1}{\boldsymbol{x}_k} \tag{2}$$

This is extremely powerful. If this indeed holds, then we can use the eigendecomposition of $\boldsymbol{A}$ to get the state $\boldsymbol{x}_j$ at any index j. Assume the eigenvectors of $\boldsymbol{A}$ are $\boldsymbol{W}$, and the eigenvalues are $\boldsymbol{\Lambda}$, i.e. $\underset{n\times n}{\boldsymbol{A}}\,\underset{n\times n}{\boldsymbol{W}} = \underset{n\times n}{\boldsymbol{W}}\,\underset{n\times n}{\boldsymbol{\Lambda}}$. These eigenvectors are called the DMD modes and they form a basis in n-dimensional space, such that any vector x can be expressed as $\boldsymbol{x} = \sum_{i=1}^{n} b_i \boldsymbol{w}_i$, where the b values are the coefficients of the linear combination. Putting them together in a vector $\boldsymbol{b}$, we get $\boldsymbol{x} = \boldsymbol{W}\boldsymbol{b}$.

Assume the system has known initial state $\boldsymbol{x}_0 = \boldsymbol{W}\boldsymbol{b}$. We can solve for $\boldsymbol{b}$ as:

$$\underset{n\times 1}{\boldsymbol{b}} = \underset{n\times n}{\boldsymbol{W}^{-1}}\,\underset{n\times 1}{\boldsymbol{x}_0} \tag{3}$$

Then, we get $\boldsymbol{x}_1 = \boldsymbol{A}\boldsymbol{W}\boldsymbol{b} = \boldsymbol{W}\boldsymbol{\Lambda}\boldsymbol{b}$. Extending this to any index, we get the incredibly powerful:

$$\underset{n\times 1}{\boldsymbol{x}_k} = \underset{n\times n}{\boldsymbol{W}}\,\underset{n\times n}{\boldsymbol{\Lambda}^k}\,\underset{n\times 1}{\boldsymbol{b}} = \sum_{i=1}^{n} {\lambda_i}^k b_i \underset{n\times 1}{\boldsymbol{w}_i} \tag{4}$$

for any index k, including positive and negative values of high magnitude. When the index is time, this implies that we can compute states of the system way out into the future or back in the past. Note, however, that this means that **for a stable system, the eigenvalue with largest real part (i.e. dominant eigenvalue) should have real part equal to** 1. If it exceeds 1, the system will explode in the future and implode in the past. If it's less than 1, the system will explode in the past and implode in the future.

Note that ideally *all eigenvalues* (not just the dominant one) should lie on the unit circle, because they all affect the system (albeit to a lesser extent than the dominant eigenvalue). If the non-dominant eigenvalues are close to the origin (i.e. real part much less than 1), then the system still has a chance of exploding in the past. Such numerical instability should be kept in mind when predicting states of the system in the past.

2 Implementing DMD

More generally, suppose there exist $m+1$ measurements of $\boldsymbol{x}$ at different indexes, i.e. $\boldsymbol{x}_0$ to $\boldsymbol{x}_m$. These are collected as $n\times m$ matrices $\boldsymbol{X} = [\boldsymbol{x}_0, \cdots, \boldsymbol{x}_{m-1}]$ and $\boldsymbol{X'} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_m]$. Then, we get the matrix form of Eq. (2):

$$\underset{n\times m}{\boldsymbol{X'}} \approx \underset{n\times n}{\boldsymbol{A}}\,\underset{n\times m}{\boldsymbol{X}} \tag{5}$$

There can be 2 different cases:

[1] The function $F(\cdot)$ is written using a capital letter because typically the space is described as $\frac{d\boldsymbol{x}}{dt} = f(\boldsymbol{x})$. Integrating this from $t = k$ to $t = k+1$ gives $\boldsymbol{x}_{k+1} = F(\boldsymbol{x}_k)$.

Case 1: $m \geq n$

This is the standard regression case where we have more observations than features. We cannot solve for $\boldsymbol{A}$ exactly since we have mn equations and only n^2 variables. So, we try to minimize the least squares error $\|\boldsymbol{X'} - \boldsymbol{AX}\|^2$, which gives the solution:

$$\underset{n\times n}{\boldsymbol{A}} = \underset{n\times m}{\boldsymbol{X'}}\ \underset{m\times n}{\boldsymbol{X}^\dagger} \tag{6}$$

Case 2: $m < n$

This is an over-determined system where we are trying to solve for n^2 variables in only mn equations. There can be infinite solutions. Whatever solution for A we pick, it will be an $n \times n$ matrix with rank only m.

Singular Value Decomposition (SVD): Consider the SVD of $\boldsymbol{X}$:

$$\underset{n\times m}{\boldsymbol{X}} = \underset{n\times n}{\boldsymbol{U}}\ \underset{n\times m}{\boldsymbol{\Sigma}}\ \underset{m\times m}{\boldsymbol{V}^H} \tag{7}$$

where ^{H} is the conjugate transpose or Hermitian.

Truncated SVD: Since $m < n$, the rank of $\boldsymbol{X}$ is at most m. Let's assume the rank of $\boldsymbol{X}$ is actually r, which is less than m. (If it turns out that the rank of $\boldsymbol{X}$ is actually m, replace r with m in the following). Now, consider the truncated SVD. Since the rank is r, we need only keep the left r columns of both $\boldsymbol{U}$ and $\boldsymbol{V}$, and the top-left $r \times r$ portion of $\boldsymbol{\Sigma}$. Then we get:

$$\underset{n\times m}{\boldsymbol{X}} = \underset{n\times r}{\boldsymbol{U}}\ \underset{r\times r}{\boldsymbol{\Sigma}}\ \underset{r\times m}{\boldsymbol{V}^H} \tag{8}$$

The left singular vectors $\boldsymbol{U}$ are the Proper Orthogonal Decomposition (POD) modes.

Reduced dimensionality POD space: The original n-dimensional vectors $\boldsymbol{x}$ can be projected into the reduced r-dimensional space of POD modes using $\tilde{\boldsymbol{x}} = \boldsymbol{U}^H\boldsymbol{x}$. Doing this to the whole matrix at once gives:

$$\underset{r\times m}{\tilde{\boldsymbol{X}}} = \underset{r\times n}{\boldsymbol{U}^H}\ \underset{n\times m}{\boldsymbol{X}} \tag{9}$$

and to convert back, we use $\boldsymbol{x} = \boldsymbol{U}\tilde{\boldsymbol{x}}$ and $\boldsymbol{X} = \boldsymbol{U}\tilde{\boldsymbol{X}}$.

Since r has replaced n and $m \geq r$, we are back in Case 1. We will now proceed with this case.

2.1 The DMD algorithm

Finding $\boldsymbol{A}$: Using the SVD of $\boldsymbol{X}$, we can get $\boldsymbol{A} = \boldsymbol{X'X}^\dagger = \boldsymbol{X'V\Sigma}^{-1}\boldsymbol{U}^H$.

We can also get $\tilde{\boldsymbol{A}} = \tilde{\boldsymbol{X}}'\tilde{\boldsymbol{X}}^\dagger$. This is where it gets tricky. $\tilde{\boldsymbol{X}}'$ is the projection of $\boldsymbol{X}'$ on the same POD modes obtained from the SVD of $\boldsymbol{X}$, i.e. $\tilde{\boldsymbol{X}}' = \boldsymbol{U}^H\boldsymbol{X}'$. So we get:

$$\underset{r\times r}{\tilde{\boldsymbol{A}}} = \underset{r\times m}{\tilde{\boldsymbol{X}}'}\,\underset{m\times r}{\tilde{\boldsymbol{X}}^\dagger} \tag{10a}$$

$$= \underset{r\times n}{\boldsymbol{U}^H}\,\underset{n\times m}{\boldsymbol{X}'}\,\underset{m\times n}{\boldsymbol{X}^\dagger}\,\underset{n\times r}{\boldsymbol{U}} \tag{10b}$$

$$= \underset{r\times n}{\boldsymbol{U}^H}\,\underset{n\times n}{\boldsymbol{A}}\,\underset{n\times r}{\boldsymbol{U}} \tag{10c}$$

$$= \underset{r\times n}{\boldsymbol{U}^H}\,\underset{n\times m}{\boldsymbol{X}'}\,\underset{m\times r}{\boldsymbol{V}}\,\underset{r\times r}{\Sigma^{-1}} \tag{10d}$$

This means that we never have to compute $\boldsymbol{A}$, we can just work with the reduced dimensionality $\tilde{\boldsymbol{A}}$. The reduced dimension r can be chosen to be anything as long as it's upper-bounded by $\min(n, m)$.

Eigendecomposition We can now compute the eigendecomposition:

$$\underset{r\times r}{\tilde{\boldsymbol{A}}} = \underset{r\times r}{\tilde{\boldsymbol{W}}}\,\underset{r\times r}{\tilde{\boldsymbol{\Lambda}}}\,\underset{r\times r}{\tilde{\boldsymbol{W}}^{-1}} \tag{11}$$

The first r eigenvalues of $\boldsymbol{A}$ are the same as those of $\tilde{\boldsymbol{A}}$, i.e. $\boldsymbol{\Lambda} = \tilde{\boldsymbol{\Lambda}}$. Since the rank of $\boldsymbol{X}$ and $\boldsymbol{X}'$ is r, the rank of $\boldsymbol{A}$ is also r (since multiplying two rank r matrices makes the result rank r). So, the other $n - r$ eigenvalues of $\boldsymbol{A}$ are all 0.

This means that we only care about the first r eigenvectors of $\boldsymbol{A}$. The eigenvectors $\boldsymbol{W}$ of $\boldsymbol{A}$ can be found in 2 ways:

The *exact* eigenvectors are:

$$\underset{n\times r}{\boldsymbol{W}} = \underset{n\times m}{\boldsymbol{X}'}\,\underset{m\times r}{\boldsymbol{V}}\,\underset{r\times r}{\Sigma^{-1}}\,\underset{r\times r}{\tilde{\boldsymbol{W}}} \tag{12}$$

For a backward proof, we can compute $\boldsymbol{W}\boldsymbol{\Lambda}\boldsymbol{W}^\dagger$ to get back $\boldsymbol{A}$.

The *projected* eigenvectors are simply:

$$\underset{n\times r}{\boldsymbol{W}} = \underset{n\times r}{\boldsymbol{U}}\,\underset{r\times r}{\tilde{\boldsymbol{W}}} \tag{13}$$

which is what one would expect given the rules of the POD space are such that $\boldsymbol{X} = \boldsymbol{U}\tilde{\boldsymbol{X}}$. Note that this $\boldsymbol{W}$ is equal to $\boldsymbol{X}\boldsymbol{V}\boldsymbol{\Sigma}^{-1}\tilde{\boldsymbol{W}}$, which is equal to the exact formulation except that $\boldsymbol{X}'$ is replaced with $\boldsymbol{X}$. Thus, the projected formulation will be equal to the exact formulation if $\boldsymbol{X}$ and $\boldsymbol{X}'$ have the same column spaces.

Note that computing the projected eigenvectors is more numerically stable. However, it is recommended to use the exact formulation unless a particular eigenvalue and its corresponding exact eigenvector are both 0, in which case the projected eigenvector should be used.

Calculating other $\boldsymbol{x}$ values: The rest of the process is mostly the same. We find:

$$\underset{r\times 1}{\boldsymbol{b}} = \underset{r\times n}{\boldsymbol{W}^\dagger}\,\underset{n\times 1}{\boldsymbol{x}_0} \tag{14}$$

Then we can find states for any index k as:

$$\underset{n\times 1}{\boldsymbol{x}_k} = \underset{n\times r}{\boldsymbol{W}}\,\underset{r\times r}{\boldsymbol{\Lambda}^k}\,\underset{r\times 1}{\boldsymbol{b}} = \sum_{i=1}^{r} {\lambda_i}^k b_i\,\underset{n\times 1}{\boldsymbol{w}_i} \tag{15}$$

2.2 Converting between discrete and continuous indexes

Thus far we have dealt with discrete indexes, i.e. the vector $\boldsymbol{x}$ was sampled at indexes with integral k. What if we want to find a continuous representation $\boldsymbol{x}(k)$, where k can be non-integral? For example, we may wish to find the state $\boldsymbol{x}$ of the system at index 3.5. In this case, the dynamics of the system are described as:

$$\underset{n\times 1}{\frac{d\boldsymbol{x}}{dk}} = \underset{n\times nn\times 1}{\boldsymbol{\mathcal{A}}\,\boldsymbol{x}} \tag{16}$$

This has solution:

$$\boldsymbol{x}(k) = e^{\boldsymbol{\mathcal{A}}k}\boldsymbol{x}_0 \tag{17}$$

The eigendecomposition is $\boldsymbol{\mathcal{A}} = \boldsymbol{W\Omega W}^{-1}$. Recall that any function of $\boldsymbol{\mathcal{A}}$ gets applied to the eigenvalues, leaving the eigenvectors intact. As before, $\boldsymbol{x}_0 = \boldsymbol{Wb}$.

Then we can solve for any index as:

$$\underset{n\times 1}{\boldsymbol{x}(k)} = \boldsymbol{W}e^{\boldsymbol{\Omega}k}\boldsymbol{W}^{-1}\boldsymbol{x}_0 = \underset{n\times nn\times nn\times 1}{\boldsymbol{W}\,e^{\boldsymbol{\Omega}k}\,\boldsymbol{b}} \tag{18}$$

or, for the truncated case:

$$\underset{n\times 1}{\tilde{\boldsymbol{x}}(k)} = \underset{n\times rr\times rr\times 1}{\boldsymbol{W}\,e^{\boldsymbol{\Omega}k}\,\boldsymbol{b}} \tag{19}$$

This formulation is equivalent to the discrete index case if we write $\boldsymbol{A} = e^{\boldsymbol{\mathcal{A}}\Delta k}$, where Δk is the sampling interval. **This means that the eigenvectors of both matrices are the same $\boldsymbol{W}$, and the eigenvalues are related as:**

$$\lambda = e^{\omega\Delta k} \tag{20a}$$

$$\Rightarrow \omega = \frac{\ln\lambda}{\Delta k} \tag{20b}$$

As an example, suppose we want to find $\boldsymbol{x}$ at index 3.5. For the continuous index case, we would get $\boldsymbol{x}(3.5) = \boldsymbol{W}e^{3.5\boldsymbol{\Omega}}\boldsymbol{b}$. For the usual discrete sampling case with $\Delta k = 1$, we cannot find $\boldsymbol{x}_{3.5}$ since the index 3.5 isn't a valid sampling index. However, we can solve for $\boldsymbol{x}$ at index 3.5 if we take $\Delta k = 0.5$. This means that we need to retake the measurements so that $\boldsymbol{X}'$, which stands for one discrete sampling index ahead of $\boldsymbol{X}$, will now have an actual value that is 0.5 ahead. Then, $\boldsymbol{\Lambda} = e^{0.5\boldsymbol{\Omega}}$, so $e^{3.5\boldsymbol{\Omega}} = \boldsymbol{\Lambda}^7$. This makes sense since 3.5 is the 7th index in this new sampling scheme. Then we can solve $\boldsymbol{x}(3.5) = \boldsymbol{x}_7 = \boldsymbol{W\Lambda}^7\boldsymbol{b}$.

Tying it together: Discrete indexing is obviously the only way we can operate on a computer. So, assume we have state measurements at indexes $[0, 0.5, 1, 1.5, 2, 3, 4, 5]$, and we want to predict the states at indexes 2.5 and 8. We can do it in one out of two ways:

- Construct the discrete problem with $\Delta k = 0.5$. This is not possible for the complete data since the values at 2.5, 3.5 and 4.5 are unavailable. It becomes possible if we only calculate DMD using indexes $[0, 0.5, 1, 1.5, 2]$, which, in this new

scheme, become indexes $[0, 1, 2, 3, 4]$. Construct $\boldsymbol{X} = [\boldsymbol{x}_0 \cdots \boldsymbol{x}_3]$ (which is actually $[\boldsymbol{x}(0) \cdots \boldsymbol{x}(1.5)]$), and $\boldsymbol{X'} = [\boldsymbol{x}_1 \cdots \boldsymbol{x}_4]$ (which is actually $[\boldsymbol{x}(0.5) \cdots \boldsymbol{x}(2)]$), and proceed as before. Once the eigendecomposition of the system's $\boldsymbol{A}$ has been determined, calculate $\boldsymbol{x}(2.5)$ as $\boldsymbol{x}_5$ and $\boldsymbol{x}(8)$ as $\boldsymbol{x}_{16}$ using Eq. (15).

- Construct the discrete problem with $\Delta k = 1$. Only consider indexes $[0, 1, 2, 3, 4, 5]$ for calculating DMD. Construct $\boldsymbol{X} = [\boldsymbol{x}_0 \cdots \boldsymbol{x}_4]$, $\boldsymbol{X'} = [\boldsymbol{x}_1 \cdots \boldsymbol{x}_5]$, and proceed as before. Once the eigendecomposition of the system's $\boldsymbol{A}$ has been determined, convert the discrete eigenvalues $\boldsymbol{\Lambda}$ to continuous eigenvalues $\boldsymbol{\Omega}$ using Eq. (20), and calculate $\boldsymbol{x}(2.5)$ and $\boldsymbol{x}(5)$ using Eq. (19).

2.3 Different starting index

Thus far we have assumed that the starting index is $\boldsymbol{x}_0$. What if it's not? Suppose we have measurements starting at $\boldsymbol{x}_i$, $i \neq 0$. There are 2 methods to deal with this:

Assume index 0 exists: Assume there is an un-measured $\boldsymbol{x}_0$. Then, $\boldsymbol{x}_i = \boldsymbol{A}^i \boldsymbol{x}_0$. Putting the earlier expression $\boldsymbol{x}_0 = \boldsymbol{W}\boldsymbol{b}$ into this yields:

$$\begin{aligned} \boldsymbol{x}_i &= \left(\boldsymbol{W}\boldsymbol{\Lambda}\boldsymbol{W}^{-1}\right)^i \boldsymbol{W}\boldsymbol{b} \\ &= \boldsymbol{W}\boldsymbol{\Lambda}^i\boldsymbol{b} \\ \Rightarrow \boldsymbol{b} &= \boldsymbol{\Lambda}^{-i}\boldsymbol{W}^{-1}\boldsymbol{x}_i \end{aligned} \tag{21}$$

In terms of the specific notation used so far, Eq. (14) now becomes:

$$\underset{r\times 1}{\boldsymbol{b}} = \underset{r\times r}{\boldsymbol{\Lambda}^{-i}}\,\underset{r\times n}{\boldsymbol{W}^{\dagger}}\,\underset{n\times 1}{\boldsymbol{x}_i} \tag{22}$$

Thus, we can get $\boldsymbol{b}$ in terms of whichever index $\boldsymbol{x}$ starts from by using the appropriate opposite power on the eigenvalue matrix. We don't need to know $\boldsymbol{x}_0$.

Then, any other value $\boldsymbol{x}_j$ can be obtained as:

$$\begin{aligned} \boldsymbol{x}_j &= \boldsymbol{W}\boldsymbol{\Lambda}^j\boldsymbol{b} \\ &= \boldsymbol{W}\boldsymbol{\Lambda}^j\boldsymbol{\Lambda}^{-i}\boldsymbol{W}^{\dagger}\boldsymbol{x}_i \\ &= \boldsymbol{A}^{j-i}\boldsymbol{x}_i \end{aligned} \tag{23}$$

which is as one would expect when computing index j from index i. Also note that j can be less than i, and everything will still hold. For example, $\boldsymbol{x}_0 = \boldsymbol{A}^{-i}\boldsymbol{x}_i$.

Shift the indexes so that i becomes 0: Thus just means subtracting i from any index. So, the given $\boldsymbol{x}_i$ in the original index space becomes $\boldsymbol{x}_{i-i} = \boldsymbol{x}_0$ in the shifted index space, and $\boldsymbol{x}_j$ becomes $\boldsymbol{x}_{j-i}$. That way, we retain the relation $\boldsymbol{b} = \boldsymbol{W}^{-1}\boldsymbol{x}_0$ in the shifted index space, i.e. we avoid powers on the eigenvalue matrix when computing $\boldsymbol{b}$ and can keep the math the same.

Now, suppose we want to compute $\boldsymbol{x}_j$ in the original index space. Then, in the shifted index space, we need to compute:

$$\boldsymbol{x}_{j-i} = \boldsymbol{W}\boldsymbol{\Lambda}^{j-i}\boldsymbol{b} \tag{24}$$

as one would expect. Also note that $\boldsymbol{x}_0$ in the original index space will be computed in the shifted index space as:

$$\begin{aligned}\boldsymbol{x}_{0-i} &= \boldsymbol{W}\boldsymbol{\Lambda}^{0-i}\boldsymbol{b} \\ &= \boldsymbol{W}\boldsymbol{\Lambda}^{0-i}\boldsymbol{W}^{-1}\boldsymbol{x}_0 \\ &= \boldsymbol{A}^{-i}\boldsymbol{x}_0 \\ &= \boldsymbol{A}^{-i}\boldsymbol{x}_{i-i}\end{aligned}$$

When expressed in the original index space, this becomes $\boldsymbol{x}_0 = \boldsymbol{A}^{-i}\boldsymbol{x}_i$, which is identical to the previous method. Thus, the two methods give identical results, as they should.

2.4 Multiple trajectories

A trajectory is defined as a run of a dynamical system from some initial state to some final state, i.e. $[\boldsymbol{x}_0, \cdots, \boldsymbol{x}_m]$. So far, we have been dealing with single trajectories. But it can so happen that a dynamical system is described by multiple trajectories instead of a single trajectory. For example, consider the system $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \psi$, where ψ denotes noise. We want a single $\boldsymbol{A}$ for the whole system since its dynamics don't change. But we also need a way to represent data from the multiple trajetories. This can be done by *concatenating the trajectories along the index dimension.*

Suppose we perform T runs of a system, which have $[m_1 + 1, m_2 + 1, \cdots, m_T + 1]$ indexed states, respectively. Using superscript for trajectory, the states can be described as $\left[\boldsymbol{x}_0^1, \cdots, \boldsymbol{x}_{m_1}^1\right], \left[\boldsymbol{x}_0^2, \cdots, \boldsymbol{x}_{m_2}^2\right], \cdots, \left[\boldsymbol{x}_0^T, \cdots, \boldsymbol{x}_{m_T}^T\right]$. Concatenating along the index dimension, we get:

$$\boldsymbol{X} = \left[\boldsymbol{x}_0^1, \cdots, \boldsymbol{x}_{m_1-1}^1, \quad \boldsymbol{x}_0^2, \cdots, \boldsymbol{x}_{m_2-1}^2, \quad \cdots, \quad \boldsymbol{x}_0^T, \cdots, \boldsymbol{x}_{m_T-1}^T\right] \tag{25a}$$

$$\boldsymbol{X}' = \left[\boldsymbol{x}_1^1, \cdots, \boldsymbol{x}_{m_1}^1, \quad \boldsymbol{x}_1^2, \cdots, \boldsymbol{x}_{m_2}^2, \quad \cdots, \quad \boldsymbol{x}_1^T, \cdots, \boldsymbol{x}_{m_T}^T\right] \tag{25b}$$

Both these matrices have dimensions $n \times M$, where $M = \sum_{t=1}^{T} m_t$. $\boldsymbol{X}$ leaves out the last sample of each trajectory, while $\boldsymbol{X}'$ leaves out the first sample of each trajectory.

The rest of the algorithm proceeds as given previously, except that m is replaced by M. This means that the reduced dimension (i.e. rank) r will be upper-bounded by $\min(n, M)$.

3 Koopman theory

The problem with DMD is that the original dynamics may not be linearizable. However, what if we transform the problem from the existing n-dimensional space to a different space? We can replace all vectors $\boldsymbol{x}$ with functions $g(\boldsymbol{x})$. Thus, $g(\boldsymbol{x})$ is an **encoding**. Essentially, now the encoded value(s) at index k is $g(\boldsymbol{x}_k)$ and at index $k+1$ is $g(\boldsymbol{x}_{k+1})$. Since $g(\boldsymbol{x})$ is a function, in general, it has an infinite number of dimensions, i.e. the number of basis functions whose linear combination equals $g(\boldsymbol{x})$ is infinite.

The transition operator to take g($\boldsymbol{x}_k$) to g($\boldsymbol{x}_{k+1}$) is $\boldsymbol{K}$, which is the Koopman operator. So:

$$g(\boldsymbol{x}_{k+1}) = \boldsymbol{K}g(\boldsymbol{x}_k) \tag{26}$$

We assume that $\boldsymbol{K}$ is a linear operator, i.e. it is a (infinite-dimensional) matrix. **Essentially, the Koopman operator transforms the problem of a non-linear transition function acting in a finite-dimensional vector space to the problem of a linear operator acting on an infinite-dimensional function space.**

The basis functions for this encoded space can be the eigenfunctions of $\boldsymbol{K}$, i.e. the functions $\{\phi(\cdot)\}$ which satisfy $\boldsymbol{K}\phi(\boldsymbol{x}) = \lambda\phi(\boldsymbol{x})$. So, any function $g(\cdot)$ on which $\boldsymbol{K}$ operates can be written as a linear, infinite combination of the Koopman eigenfunctions $\{\phi(\cdot)\}$.

3.1 Koopman operator as function composition

Note that $\boldsymbol{K}g(\boldsymbol{x}_k) = g(\boldsymbol{x}_{k+1})$. But as per Eq. (1), $\boldsymbol{x}_{k+1} = F(\boldsymbol{x}_k)$. This implies that $\boldsymbol{K}g(\boldsymbol{x}_k) = g(F(\boldsymbol{x}_k))$. Dropping the index:

$$\boldsymbol{K}g(\boldsymbol{x}) = g(F(\boldsymbol{x})) \tag{27}$$

So, the Koopman operator can be regarded as function composition.

4 Implementing Koopman theory as generalized DMD

The Koopman operator cannot be implemented in practice since it is infinite dimensional. However, we can implement a finite approximation by assuming the encoded space to be p-dimensional, where p is finite. Then, we can write:

$$\underset{p\times 1}{\boldsymbol{y}} = g\left(\underset{n\times 1}{\boldsymbol{x}}\right) \tag{28}$$

and use a finite-dimensional $p \times p$ Koopman matrix $\boldsymbol{K}$ as the equivalent of the $n \times n$ DMD matrix $\boldsymbol{A}$ from Eq. (2):

$$\underset{p\times 1}{\boldsymbol{y}_{k+1}} \approx \underset{p\times p}{\boldsymbol{K}}\,\underset{p\times 1}{\boldsymbol{y}_k} \tag{29}$$

Thus, once we have the $\boldsymbol{y}$ values, **one can perform DMD exactly as described thus far by replacing $\boldsymbol{x}$ with $\boldsymbol{y}$, $\boldsymbol{X}$ with $\boldsymbol{Y}$, n with p, $\boldsymbol{A}$ with $\boldsymbol{K}$, and solving for $\boldsymbol{y}_j$ at unknown indexes j.**

Finally, one needs to convert back from the encoded space to the original space using:

$$\underset{n\times 1}{\boldsymbol{x}} = h\left(\underset{p\times 1}{\boldsymbol{y}}\right) \tag{30}$$

where $h(\cdot)$ should approximate $g^{-1}(\cdot)$.

Note that the term *extended dynamic mode decomposition (EDMD)* refers to this same technique, except it uses orthonormal polynomial basis functions as $g(\cdot)$. Koopman theory in more general and can work with any $g(\cdot)$. For example, one can use a neural network encoder-decoder architecture to represent $g(\cdot)$ and $h(\cdot)$.

References

[1] J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor. *Dynamic Mode Decomposition: Data Driven Modeling of Complex Systems*. Society for Industrial and Applied Mathematics, 2016.

[2] Jonathan H. Tu, Clarence W. Rowley, Dirk M. Luchtenburg, Steven L. Brunton, and J. Nathan Kutz. On dynamic mode decomposition: Theory and applications. *Journal of Computational Dynamics*, 1(2):391–421, 2014.

Appendix: Variable reference

m – Number of data measurements excluding the 0th measurement. Thus, total number of data measurements is $m + 1$.

n – Number of original states.

p – Number of transformed or encoded states.

r – Rank.