Dynamic Mode Decomposition and Some Variants

by

Linkai Ma

New York University

September, 2022

Professor Jonathan Weare

Contents

1 Introduction

Since its birth from the fluid dynamics community[Schimid 2010; Rowley et al. 2009], dynamic mode decomposition (DMD) has gained an increasing popularity among computational science and engineering. DMD is an equation-free method for detecting linear trends from observed snapshot data of a dynamical system. It constructs low-dimensional modes from high-dimensional time series data. Compared to other model order reduction methods, such as Proper Orthogonal Decomposition (POD)[Berkooz et al. 1993], DMD couples the spatial dimension reduction modes with different time frequencies, enabling the dynamic modes to capture more complex behaviors of the system. Although the theory of DMD is far from complete, we have seen successful applications in many different fields, such as, atmospheric science, video analysis, neuroscience, finance, epidemiology, etc.

One interesting variant of DMD is the multiresolution DMD (mrDMD) [Kutz et al. 2016]. The mrDMD separates the slow and fast DMD modes and apply DMD recursively. It is capable of separating background and foreground data and was found to be extremely successful in video analysis. DMD is also related to the Koopman operator [Koopman 1931], which is a linear operator that acts on nonlinear observables in an infinitedimensional Hilbert space. The existence of this linear operator gives us hope to linearize complicated dynamics on the space of observables. Rowely et al were able to connect DMD with the Koopman operator in [Rowley et al. 2009]. Another example is the extended DMD method[Williams et al. 2016], which approximates the Koopman operator from a dictionary of function basis.

2 Background

2.1 Dynamical System and Data

In the DMD literature, we assume that our data is generated from a dynamical system

$$\frac{dx}{dt} = f(x; t; \theta)$$

where $x(t) \in \mathbb{R}^n$ is the state of our dynamical system at time t and θ is the parameter of our model.

The standard DMD aims to model our system with the simplest dynamical system:

$$\frac{dx}{dt} = \Lambda$$

where Λ is a constant matrix.

Under this assumption, the solution of our dynamical system is:

$$x(t) = e^{t\Lambda}x(0)$$

If we discretize our system with time step Δt, we should have

$$\forall k,\ \ x_{k+1} = e^{\Delta t\Lambda}x_k$$

If we denote $A = e^{\Delta t \Lambda}$, we have:

$$x_{k+1} = Ax_k$$

Suppose we collect $m + 1$ snapshot data, and label the data as:

$$X_L = \begin{bmatrix} | & | & & | \\ x_0 & x_1 & \dots & x_{m-1} \\ | & | & & | \end{bmatrix}$$

$$X_R = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \dots & x_m \\ | & | & & | \end{bmatrix}$$

Our model aims to find some constant matrix A such that $X_R \approx AX_L$.

2.2 Standard DMD

Dynamic Mode Decomposition (DMD) finds a best matrix A for the approximation $X_R \approx AX_L$ in the Frobenius sense.

$$A = \arg\min_{\Pi} ||X_R - \Pi X_L||_F^2$$

The solution to this optimization problem is not necessarily unique. One solution is $A = X_R X_L^\dagger$, where $\dagger$ is the Moore–Penrose inverse. We can compute this solution explicitly by using the SVD of X_L. Say $X_L = U\Sigma V^*$, then $A = X_R V \Sigma^\dagger U^*$.

In the DMD literature, the derivation of A is oftentimes omitted. We present below the proof of a slightly more generalized result.

Lemma 1

$$\text{For } C \in \mathbb{C}^{n\times m}, B \in \mathbb{C}^{k\times m}, \ \arg\min_{A\in\mathbb{C}^{n\times k}} ||C - AB||_F^2 = CB^\dagger$$

Proof. Let's look at this problem from a 'row view'.

$$\begin{pmatrix} — & c_1 & — \\ — & c_2 & — \\ & \vdots & \\ — & c_n & — \end{pmatrix} \approx A \begin{pmatrix} — & b_1 & — \\ — & b_2 & — \\ & \vdots & \\ — & b_k & — \end{pmatrix} = \begin{pmatrix} — & a_1 B & — \\ — & a_2 B & — \\ & \vdots & \\ — & a_n B & — \end{pmatrix}$$

where a_i, b_i, c_i's are rows of A, B and C.

To minimize the Frobenius norm of the difference, we need to approximate the rows of C by linear combinations of rows of B. Since we can choose the linear combination, i.e. the rows of A independently, we just need to minimize the error for each row independently.

The above can be done in two steps.

Step 1: project c_i on to the row space of B. Step 2: express the result as a linear combination of b_i's.

Denote $r = \text{rank}(B)$. Let's denote the reduced rank-r SVD of B as $B = U\Sigma V^*$.

For step 1, we can use the projector constructed from the SVD: $c_i V V^*$ gives the desired result.

For step 2, we make the following observation:

$$\begin{aligned} U\Sigma V^* &= B \\ V^* &= \Sigma^{-1} U^* B \\ c_i V V^* &= c_i V \Sigma^{-1} U^* B \end{aligned}$$

i.e.

$$c_i VV^* = c_i V\Sigma^{-1}U^* \begin{pmatrix} — & b_1 & — \\ — & b_2 & — \\ & \vdots & \\ — & b_k & — \end{pmatrix}$$

Therefore, the coordinates we are looking for is $a_i = c_i V\Sigma^{-1}U^*$

Putting this together as a matrix, we have: $A = CV\Sigma^{-1}U^* = CB^\dagger$ □

We then define $\widetilde{A}$ as the projection of A onto the dominant component of column space of X_L and find the eigenvectors W of $\widetilde{A}$. We define the projected dynamic modes as the eigenvectors W reconstructed back to the column space of X_L.

Algorithm 1 Standard DMD

input: $(n \times m)$ snapshot data matrices X_L, X_R; rank of DMD r

output: projected dynamic modes Φ of the data and the corresponding eigenvalues E

1. Compute reduced rank-r SVD of $X_L \approx U\Sigma V^*$

2. Define $\widetilde{A}$ as the projection of A onto the column space of U

$$\widetilde{A} = U^*AU = U^*X_R(X_L)^\dagger U = U^*X_R V\Sigma^{-1}U^*U = U^*X_R V\Sigma^{-1}$$

3. Compute eigenvectors and eigenvalues of $\widetilde{A}$

$$\widetilde{A}W = WE$$

4. Compute the projected dynamic modes

$$\Phi = UW$$

5. Return (Φ, E)

Note that Φ is the eigenvectors of $\widetilde{A}$ projected back to the column space of U. It is not the exact eigenvector of A. The exact eigenvectors are $X_R V \Sigma^{-1} W$. If we denote $\mathbb{P}_L$ as the orthogonal projection on the column space of X_L, then for nonzero eigenvalues, Φ is the eigenvectors of $\mathbb{P}_L A$ and $\Phi = \mathbb{P}_L E^{-1} X_R V \Sigma^{-1} W$. For details of the discussion between the projected eigenvectors and exact vectors, see theorem 2 and 3 in [Tu et al. 2014].

We can use the dynamic modes to reconstruct the data. Let $b = \Phi^{\dagger} x_0$, i.e. the projection of x_0 onto the projected dynamic modes in the least square sense. Then our data could be constructed by:

$$x_k = \sum_{i=1}^{r} \phi_i e_i^k b_i = \Phi E^k b$$

where ϕ_i is the i-th column of Φ, and e_i is the corresponding eigenvalue.

3 Randomized DMD for High Dimensional Data

3.1 Motivation

The standard DMD algorithm is capable of capturing nonlinear trend from data without knowledge of the underlying equation. It has become a powerful data-driven tool for analyzing high-dimensional time series data and is gaining a growing popularity among the data science community. However, there is an unavoidable large computational cost when applying DMD to high dimensional data. For applications in fluids and climate data, the dimension is often the number of grid points of the discretization scheme.

If we assume the dimension of our data to be n and the number of time steps to be m, in the first step of the DMD algorithm, taking the SVD of X_L costs $O(n^2m)$, which becomes extremely expensive when n is large. For real world applications of the DMD, there's always a low-dimensional structure for the data. Hence we hope to find a computational efficient way to compute DMD based on the intrinsic low-rank structure of the data. Brunton, Erichson, Kutz and Mathelin[Erichson et al. 2019] proposed a randomized DMD method based on the randomized numerical linear algebra frame work by Halko, Martinsson, and Tropp[Halko et al. 2011]. We develop a similar randomized DMD method with faster convergence using the Randomized Block Krylov Method in[Musco and Musco 2015].

3.2 Randomized DMD

As stated in [Musco and Musco 2015], the Block Krylov Method is based on the fact that for finding low-rank approximations of a matrix A:

"There are better polynomials than A^q for denoising tail singular values."

The Randomized Block Krylov Method stores all the Krylov subspaces along the power iteration and find the low rank approximation from this larger space. We use this method to efficiently compute a low-rank basis for the time series data and perform DMD on the low-rank projection.

Algorithm 2 Randomized DMD

input: $(n \times m)$ data matrices X_L, X_R; target rank of DMD r, target rank $l = r + p$ for randomized low-rank approximation and target error ϵ.

output: projected dynamic modes Φ of the data and the corresponding eigenvalues E

1. Generate a random matrix $\Pi \in \mathbb{R}^{(m+1)\times l}$ with i.i.d. standard Gaussian entries.

2. Let $q = O(\frac{\log(m)}{\sqrt{\epsilon}})$, define

$$X = \begin{bmatrix} | & | & & | \\ x_0 & x_1 & \dots & x_m \\ | & | & & | \end{bmatrix}$$

Compute

$$K = \begin{bmatrix} X\Pi & (XX^*)X\Pi & (XX^*)^2 X\Pi & \dots & (XX^*)^q X\Pi \end{bmatrix}$$

(Note: For implementations, in order to have better numerical stability, perform QR every few steps along the power iteration.)

Let $[Q, R] = \text{qr}(K)$

3. Let $Y_L = Q^* X_L, Y_R = Q^* X_R$

4. Let $A_Y = Y_R Y_L^\dagger$. Compute reduced rank-r SVD of $Y_L \approx U \Sigma V^*$, then compute the projected $\tilde{A_Y} = U^* Y_R V \Sigma^{-1}$, compute the eigenvalue and eigenvectors of $\tilde{A_Y}$: $\tilde{A_Y} W_Y = W_Y E$

5. Finally, reconstruct the top r high-dimensional DMD modes by $\Phi = Q W_Y$, and return the eigenvalues E.

The difference between Algorithm 2 and Algorithm 1 is that the above algorithm uses Q from randomized low-rank approximation instead of the left singular vectors U to project the matrix A.

$$\begin{aligned} A_Y &= Y_R Y_L^\dagger \\ &= (Q^* X_R)(Q^* X_L)^\dagger \\ &= Q^* X_R X_L^\dagger (Q^*)^\dagger \\ &= Q^* X_R X_L^\dagger Q \\ &= Q^* A Q \end{aligned}$$

Reconstruction of the data from DMD modes follows the same procedure as the standard DMD.

3.3 Numerical Experiments

We compare our **Algorithm 2**, Block Krylov DMD (bkDMD), the algorithm proposed in [Erichson et al. 2019], Randomized DMD (rDMD) and deterministic DMD by testing on the data sets used in the original paper.

In the numerical experiments below, we compare the three methods by plotting the DMD eigenvalues. We also added some Gaussian noise to the data to see if our method is robust to noise. For snapshot data $\in \mathbb{R}^n$, we first computed the average 2-norm of the data and call it λ. Then if we want to add $\alpha\%$ noise, we draw a standard Gaussian in $\mathbb{R}^n$ and multiply it by $\frac{\alpha\%\lambda}{\sqrt{n}}$

3.3.1 Fluid Flow Behind a Cylinder

We used the simulation data provided by the above paper, which is the numerical simulation of vorticity of fluid flow behind a cylinder at Reynolds number $Re = 100$. The concatenated snapshot data has dimension $n = 89351$, and time steps $m = 151$.

We first test these methods on the original data and then add 20% Gaussian noise. We plot below 3.1 the singular values of the original data and the one with noise.

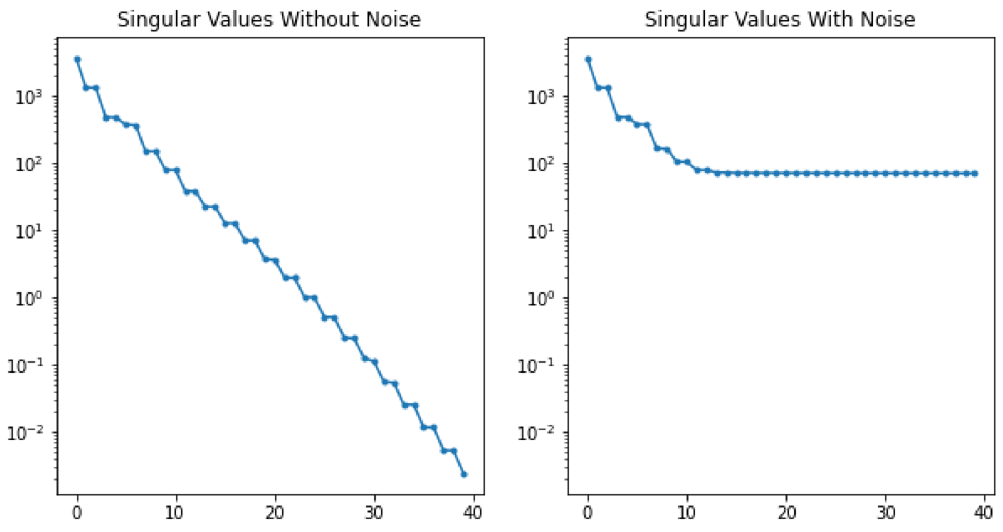

Figure 3.1: Singular values of the data

We observe that the original data has exponentially decaying singular values. Therefore, we should expect both bkDMD and rDMD to be performing extremely well. In fact, with only 1 power iteration and setting the damping parameter $p = 3$, both methods are able to capture the top 15 dynamic modes accurately. The rDMD had a 10 fold speed increase and the bkDMD had a 5 fold speed increase.

In order to see the benefit of bkDMD over rDMD, we added 20 percent noise to each snapshot of our data. In that case, the singular values are not exponentially decaying and keeping the entire Krylov subspace gives much better approximation. We tried to extract the top 15 modes with damping parameter $p = 3$. For rDMD, we used 10 power iterations and for bkDMD, we used 3. Both methods used about half of the time required by deterministic DMD. The result from the bkDMD is significantly better than rDMD.

Please see the eigenvalue plots of our two tests belows:

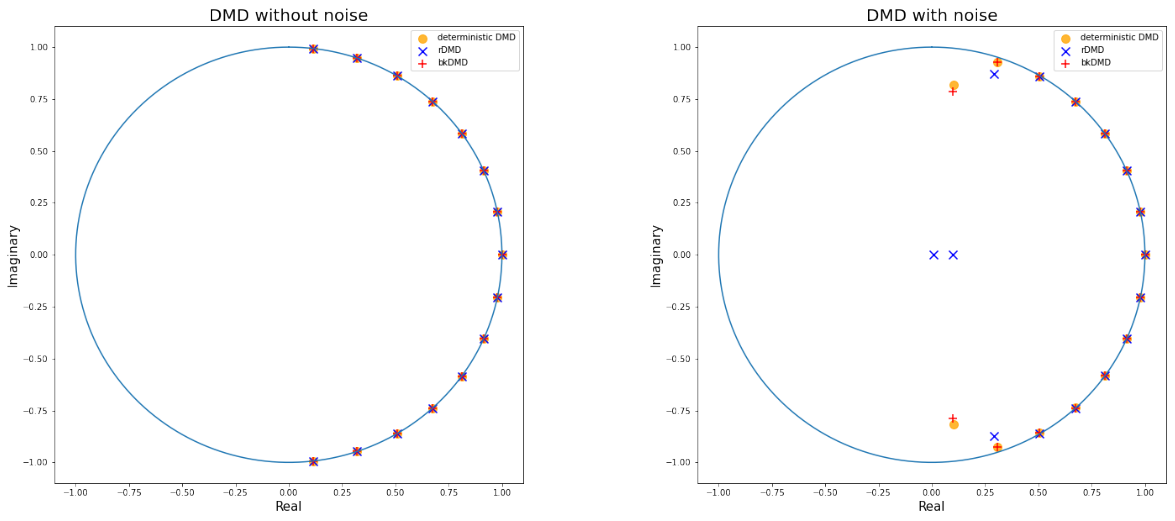

Figure 3.2: DMD eigenvalues of fluid data

3.3.2 Sea Surface Temperature Data

We also tested our method on a much larger data set, provided by National Oceanic and Atmospheric Administration (NOAA). The data set consists of daily sea surface temperature measured on a 1/4 ° global grid. We used the data from 1981 to 2022 and compared the three methods. The concatenated snapshot data has dimension $n = 691150$, and time steps $m = 15097$. Computing deterministic DMD on the original data was beyond our computational capacity, therefore we compute the weekly average first and test our methods on the shorter time series, with number of time steps $m = 2156$. Our data is downloaded from:

https://downloads.psl.noaa.gov/Datasets/noaa.oisst.v2.highres/

We measure the performance of our method by the same relative error defined by [Erichson et al. 2019]:

$$\rho(\widehat{X'}, \widehat{X_{DMD}}) = \frac{||\widehat{X'} - \widehat{X_{DMD}}||_F}{||\widehat{X_{DMD}}||_F}$$

where $\widehat{X_{DMD}}$ is the data reconstructed from deterministic DMD and $\widehat{X'}$ is the data reconstructed from the randomized method.

For the numerical experiments, we used a target rank $k = 20$, damping parameter $p = 5$ and $q = 3$ power iterations for both rDMD and bkDMD. Both methods had around 4 to 5 fold speed increase compared to deterministic DMD, with very small relative errors. The relative error for rDMD was about 8 times larger than the relative error for bkDMD.

We could also compare the two methods by looking at the eigenvalue plot 3.3:

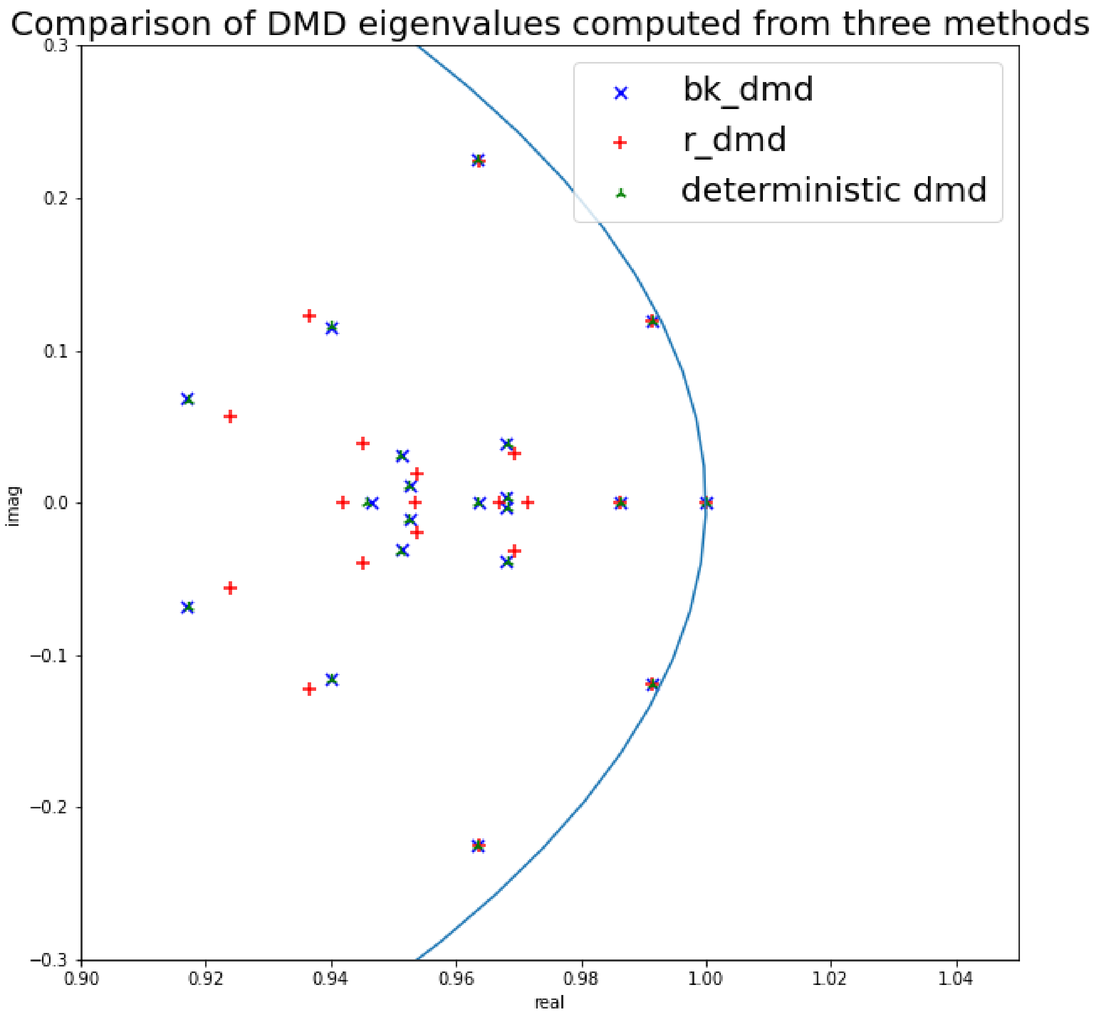

Figure 3.3: DMD eigenvalues of SST data

4 | DMD for Streaming Data

4.1 Motivation

In many applications of the DMD, the time series data to be analyzed come from streaming data. These time series could be, for example, user input from a website, ocean temperature over time, $\cdots$, etc. Updating the standard DMD algorithm is extremely expensive, since we need to compute the singular value decomposition of the entire dataset every time when we add an additional snapshot data. Zhang et al. proposed an efficient method to update the DMD for additional data based on the assumption that the data has full rank[Zhang et al. 2019]. They also developed an update approach in which weight for different time steps are incorporated so that DMD could be applied to systems that varies significantly in time.

4.2 DMD with single update

We propose another update approach without the full row rank assumption. The novelty of our method is that we take full advantage of the intrinsic low rank structure of the data. This method could be easily combined with the randomized DMD.

Given the data matrices X_L, X_R and p, q, r, we first compute randomzied DMD (rDMD or bkDMD) following algorithm 2 and keep the orthogonal matrix Q as output for further usage. In practice, to have better performance, we can also initialize with deterministic DMD, i.e. use the top $l = p + r$ left eigenvectors of X_L as our orthogonal matrix Q.

We introduce a new parameter δ, the update threshold and form the new data matrices as:

$$\widehat{X_L} = \begin{bmatrix} | & | & & | \\ x_0 & x_1 & \dots & x_m \\ | & | & & | \end{bmatrix}$$

$$\widehat{X_R} = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \dots & x_{m+1} \\ | & | & & | \end{bmatrix}$$

If the new snapshot data still lies in the column space of Q with relative error δ, i.e.:

$$\frac{||x_{m+1} - QQ^* x_{m+1}||_2}{||x_{m+1}||_2} \leq \delta$$

we keep Q unchanged.

Then, we redefine Y_L and Y_R as:

$$\begin{aligned} \widehat{Y_L} &= Q^* \widehat{X_L} \\ &= Q^* \left[X_L \,\middle|\, x_m \right] \\ &= \left[Q^* X_L \,\middle|\, Q^* x_m \right] \\ &= \left[Y_L \,\middle|\, Q^* x_m \right] \end{aligned}$$

$$\widehat{Y_R} = \left[Y_R \,\middle|\, Q^* x_{m+1} \right]$$

$$\begin{aligned}
\widehat{A}_Y &= \widehat{Y}_R \widehat{Y}_L^\dagger \\
&= \widehat{Y}_R \widehat{Y}_L^* (\widehat{Y}_L \widehat{Y}_L^*)^\dagger \\
&= (\sum_{k=1}^{m+1} y_k y_{k-1}^*)(\sum_{k=0}^{m} y_k y_k^*)^\dagger \\
&= (Y_R Y_L^* + y_{m+1} y_m^*)(Y_L Y_L^* + y_m y_m^*)^\dagger
\end{aligned}$$

where in the 2nd line we used the identity $A^\dagger = A^* A^{\dagger *} A^\dagger = A^* A^{*\dagger} A^\dagger = A^*(AA^*)^\dagger$

If $x_m \in \text{col}(X_L)$, we can further simplify our expression as:

$$\begin{aligned}
\widehat{A}_Y &= (Y_R Y_L^* + y_{m+1} y_m^*)[(Y_L Y_L^*)^\dagger + G] \\
&= A_Y + Y_R Y_L^* G + y_{m+1} y_m^* (Y_L Y_L^*)^\dagger + y_{m+1} y_m^* G
\end{aligned}$$

where $G = -(Y_L Y_L^*)^\dagger y_m y_m^* \frac{(Y_L Y_L^*)^\dagger}{[1 + y_m^* (Y_L Y_L^*)^\dagger y_m]}$. This result comes from a direct application of the seminal work of Cline[Cline 1965]:

Therorem 1. For any matrices, U and V, the generalized inverse of the sum $UU^* + VV^*$ can be written in the form:

$$(UU^* + VV^*)^\dagger = (I - C^{\dagger *} V^*) U^{\dagger *} [I - U^\dagger V (I - C^\dagger C) K V^* U^{\dagger *}] U^\dagger (I - V C^\dagger) + C^{\dagger *} C^\dagger$$

where

$$C = (I - UU^\dagger)V$$

and

$$K = [I + (I - C^\dagger C) V^* U^{\dagger *} U^\dagger V (I - C^\dagger C)]^{-1}$$

The assumption that $x_m \in \text{col}(X_L)$, or losely speaking: $||x_m - QQ^* x_m||_2$ is small is checked at the beginning of the previous iteration of the algorithm, when we decide whether an update of Q is necessary. If the assumption doesn't hold, we will need to compute $(Y_L Y_L^* + y_m y_m^*)^\dagger$ using SVD.

Since we are using this update method for the randomized DMD discussed in last section, we are assuming the spatial (n) and temporal (m) dimensions of the data to be much larger than the target rank (l). Computing the pseudoinverse of $Y_L Y_L^* + y_m y_m^*$ is very cheap as this matrix has dimension $l \times l$.

When the change of Q is necessary, we update Q by:

$$\widehat{Q} = \left[Q \;\middle|\; q_{l+1} \right]$$

where

$$q_{l+1} = \frac{x_{m+1} - QQ^* x_{m+1}}{||x_{m+1} - QQ^* x_{m+1}||_2}$$

Let's denote $\widetilde{Y_L} = q_{l+1}^* X_L$, $\widetilde{Y_R} = q_{l+1}^* X_R$, $\widetilde{y_m} = q_{l+1}^* x_m$, $\widetilde{y}_{m+1} = q_{l+1}^* x_{m+1}$,then

$$\begin{aligned} \widehat{Y_L} &= \begin{bmatrix} Q^* \\ q_{l+1}^* \end{bmatrix} \begin{bmatrix} X_L & x_m \end{bmatrix} \\ &= \begin{bmatrix} Y_L & y_m \\ \widetilde{Y_L} & \widetilde{y_m} \end{bmatrix} \end{aligned}$$

$$\begin{aligned} \widehat{Y_R} &= \begin{bmatrix} Q^* \\ q_{l+1}^* \end{bmatrix} \begin{bmatrix} X_R & x_{m+1} \end{bmatrix} \\ &= \begin{bmatrix} Y_R & y_{m+1} \\ \widetilde{Y_R} & \widetilde{y}_{m+1} \end{bmatrix} \end{aligned}$$

We also keep track of $\widehat{Y_R}\widehat{Y_L}^*$ and $\widehat{Y_L}\widehat{Y_L}^*$ for the use of future iteration:

$$
\begin{aligned}
\widehat{Y_R}\widehat{Y_L}^* &= \begin{bmatrix} Y_R & y_{m+1} \\ \widetilde{Y_R} & \widetilde{y}_{m+1} \end{bmatrix} \begin{bmatrix} Y_L^* & \widetilde{Y}_L^* \\ y_m^* & \widetilde{y}_m^* \end{bmatrix} \\
&= \begin{bmatrix} Y_R Y_L^* + y_{m+1}y_m^* & Y_R \widetilde{Y}_L^* + y_{m+1}\widetilde{y}_m^* \\ \widetilde{Y}_R Y_L^* + \widetilde{y}_{m+1}y_m^* & \widetilde{Y}_R \widetilde{Y_L}^* + \widetilde{y}_{m+1}\widetilde{y}_m^* \end{bmatrix}
\end{aligned}
$$

and

$$
\begin{aligned}
\widehat{Y_L}\widehat{Y_L}^* &= \begin{bmatrix} Y_L & y_m \\ \widetilde{Y_L} & \widetilde{y}_m \end{bmatrix} \begin{bmatrix} Y_L^* & \widetilde{Y}_L^* \\ y_m^* & \widetilde{y}_m^* \end{bmatrix} \\
&= \begin{bmatrix} Y_L Y_L^* + y_m y_m^* & Y_L \widetilde{Y}_L^* + y_m\widetilde{y}_m^* \\ \widetilde{Y}_L Y_L^* + \widetilde{y}_m y_m^* & \widetilde{Y}_L \widetilde{Y}_L^* + \widetilde{y}_m\widetilde{y}_m^* \end{bmatrix}
\end{aligned}
$$

Then the new $\widehat{A}_Y = \widehat{Y_R}\widehat{Y_L}^\dagger = \widehat{Y_R}\widehat{Y_L}^*(\widehat{Y_L}\widehat{Y_L}^*)^\dagger$. Notice that we should use the second expression, i.e., compute the pseudo-inverse of $\widehat{Y_L}\widehat{Y_L}^*$ since this matrix has dimension $l \times l$, while $\widehat{Y_L}$ has dimension $l \times m$.

4.3 DMD with block update

This method could also be easily adapted when we want to update the DMD every few steps, instead of updating it for every new snapshot data.

Suppose we want to update the DMD with additional s snapshot data $x_{m+1}, x_{m+2}, \cdots, x_{m+s}$.

We need to check if we need to update Q first: compute $||x_i - QQ^*x_i||_2$ for all the new data one by one and update Q accordingly.

If we don't need to update Q:

$$\widehat{Y_L} = \left[\begin{array}{c|ccc} Y_L & y_m & y_{m+1} \cdots & y_{m+s-1} \end{array} \right]$$

$$\widehat{Y_R} = \left[\begin{array}{c|ccc} Y_R & y_{m+1} & y_{m+2} \cdots & y_{m+s} \end{array} \right]$$

$$\begin{aligned} \widehat{A_Y} &= \widehat{Y_R}\widehat{Y_L}^{\dagger} \\ &= \widehat{Y_R}\widehat{Y_L}^{*}(\widehat{Y_L}\widehat{Y_L}^{*})^{\dagger} \\ &= (Y_R Y_L^* + \sum_{i=1}^{s} y_{m+i} y_{m+i-1}^{*})(Y_L Y_L^* + \sum_{i=1}^{s} y_{m+i-1} y_{m+i-1}^{*})^{\dagger} \end{aligned}$$

If we need to add t new dimensions, $q_{l+1}, \cdots, q_{l+t}$:

Following the same notation, we define

$$\widetilde{Y_L} = \begin{bmatrix} q_{l+1}^* X_L \\ q_{l+2}^* X_L \\ \cdots \\ q_{l+t}^* X_L \end{bmatrix}, \ \widetilde{Y_R} = \begin{bmatrix} q_{l+1}^* X_R \\ q_{l+2}^* X_R \\ \cdots \\ q_{l+t}^* X_R \end{bmatrix}, \ \widetilde{y_m} = \begin{bmatrix} q_{l+1}^* x_m \\ q_{l+2}^* x_m \\ \cdots \\ q_{l+t}^* x_m \end{bmatrix}, \ \widetilde{y}_{m+1} = \begin{bmatrix} q_{l+1}^* x_{m+1} \\ q_{l+2}^* x_{m+1} \\ \cdots \\ q_{l+t}^* x_{m+1} \end{bmatrix}$$

then the computation is the same:

$$\widehat{Y_R}\widehat{Y_L}^{*} = \begin{bmatrix} Y_R Y_L^* + \sum_{i=1}^{s} y_{m+i}\, y_{m+i-1}^{*} & Y_R \widetilde{Y_L}^* + \sum_{i=1}^{s} y_{m+i}\, \widetilde{y}_{m+i-1}^{*} \\ \widetilde{Y_R} Y_L^* + \sum_{i=1}^{s} \widetilde{y}_{m+i}\, y_{m+i-1}^{*} & \widetilde{Y_R}\widetilde{Y_L}^{*} + \sum_{i=1}^{s} \widetilde{y}_{m+i}\, \widetilde{y}_{m+i-1}^{*} \end{bmatrix}$$

$$\widehat{Y_L}\widehat{Y_L}^{*} = \begin{bmatrix} Y_L Y_L^* + \sum_{i=1}^{s} y_{m+i-1}\, y_{m+i-1}^{*} & Y_L \widetilde{Y_L}^* + \sum_{i=1}^{s} y_{m+i-1}\, \widetilde{y}_{m+i-1}^{*} \\ \widetilde{Y_L} Y_L^* + \sum_{i=1}^{s} \widetilde{y}_{m+i-1}\, y_{m+i-1}^{*} & \widetilde{Y_L}\widetilde{Y_L}^{*} + \sum_{i=1}^{s} \widetilde{y}_{m+i-1}\, \widetilde{y}_{m+i-1}^{*} \end{bmatrix}$$

4.4 Numerical Experiments

4.4.1 DMD with single update

We test our update method on the fluid data in section 3.3.1. We try to capture the top 15 dynamic modes with damping parameter $p = 5$. We separate out the last 30 snapshot data and add them as new data one by one. We then plot the DMD eigenvalues at time 0, 10, 20, 30 for the following four methods: deterministic DMD each step, update each step by the method discussed above, rDMD each step, bkDMD each step. For the update method, we initialize with deterministic DMD. For rDMD, we used $q = 5$ power iterations and for bkDMD $q = 2$ power iterations. As we can see from 4.1 , all of the methods are able to capture the DMD mode accurately. The update method had an over 30-fold speed increase compared to computing deterministic DMD for each step. Both rDMD and bkDMD consumed around 60 percent of the time used by deterministic DMD and the update method had over 20-fold speed increase compared to rDMD and bkDMD.

When 20% Gaussian noise was added to each snapshot data, our update method had around 10-fold speed increase compared to deterministic DMD. We observe in the last time step, our update method was able to capture the DMD modes correctly while bkDMD lost 2 modes and rDMD lost 4. Please see 4.2 for details.

4.4.2 DMD with block update

We perform the same experiment with the same parameters as in the single update case. In order to test our block update method, we separate out the last 60 snapshot data and add them as new block data in 3 steps, each step adding 20 timesteps. Again, all methods were able to capture the DMD modes correctly. Our update method had an over 4-fold speed increase compared to the deterministic DMD. When adding 20% Gaussian noise, we observe similar results. Our update method was able to capture the DMD modes more accurately. See 4.3 and 4.4.

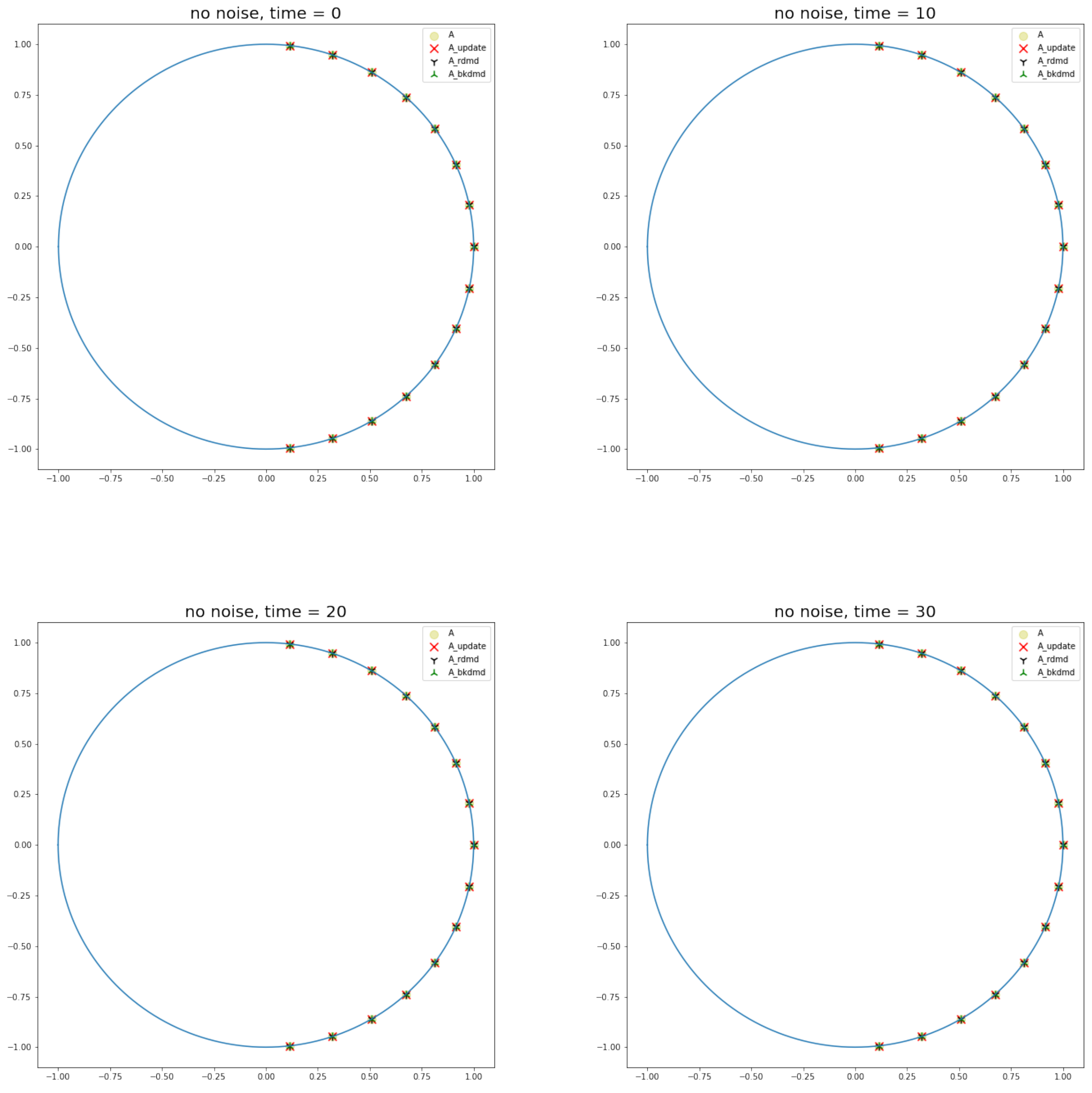

Figure 4.1: DMD eigenvalues of single update without noise

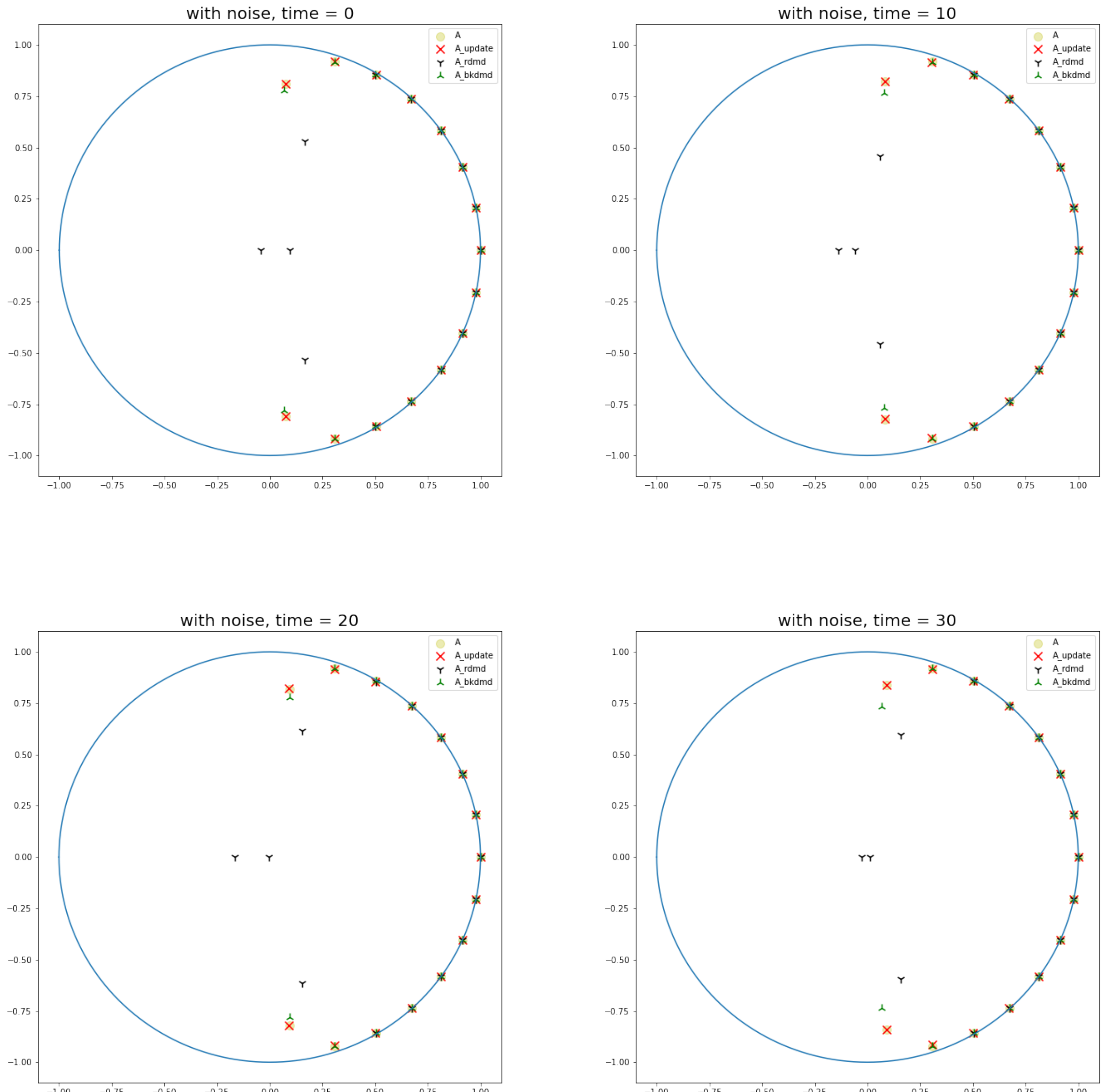

Figure 4.2: DMD eigenvalues of single update with noise

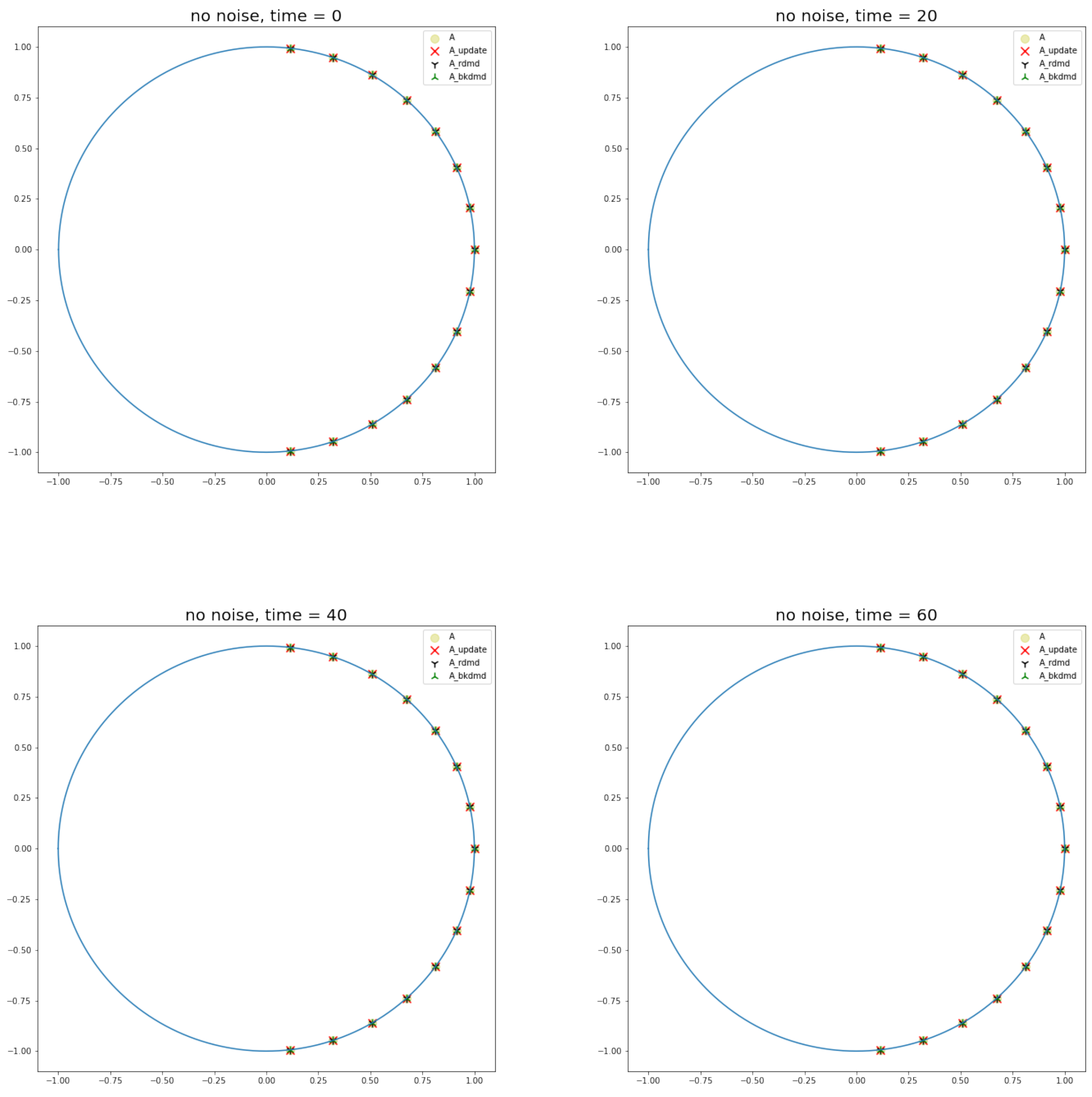

Figure 4.3: DMD eigenvalues of blcok update without noise

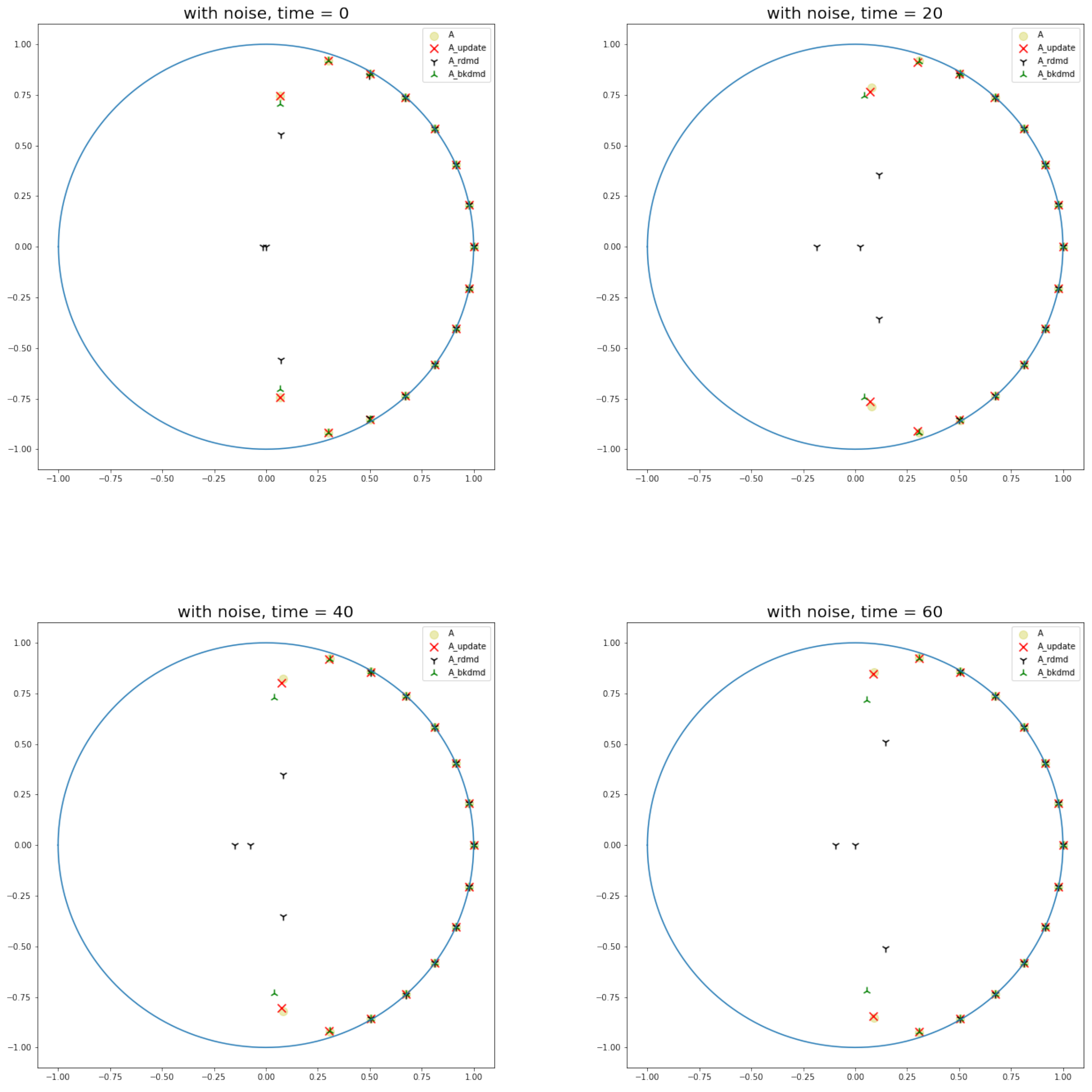

Figure 4.4: DMD eigenvalues of block update with noise

A Appendix

A.1 Python Code

```
import numpy as np
ct = lambda x: x.T.conjugate()
qr = lambda x: np.linalg.qr(x)[0]

def rDMD (X,p,q,k):
    XL = X[:,:-1]
    XR = X[:,1:]
    l = k + p
    n,m = X.shape
    omega = np.random.normal(size=(m-1,l))
    Y = XL@omega
    for i in range(q):
        Q = qr(Y)
        Q = qr(ct(XL)@Q)
        Y = XL@Q
    Q = qr(Y)
    Y = ct(Q)@X
    YL = Y[:,:-1]; YR = Y[:,1:]
    u,s,vh = np.linalg.svd(YL,full_matrices=False)
    u = u[:,:k]; vh = vh[:k,:]; s = s[:k]
    AY = ct(u)@YR@ct(vh)@np.diag(1/s)
    YRYL = YR@ct(YL); YLYL = YL@ct(YL)
    ym = YR[:,-1:]
    return AY, YRYL, YLYL, Q, X, Y

def bkDMD (X,p,q,k):
    XL = X[:,:-1]
    XR = X[:,1:]
    l = k + p
    n,m = X.shape
    omega = np.random.normal(size=(m-1,l))
    K = np.zeros((n,l*(q+1)))
    K[:,:l] = qr(XL@omega)
    for i in range(q):
        last_K = K[:,i*l:(i+1)*l]
        new_K = qr(ct(XL)@last_K)
        new_K = qr(XL@new_K)
        K[:,(i+1)*l:(i+2)*l] = new_K
    Q = qr(K)
    Y = ct(Q)@X
    YL = Y[:,:-1]; YR = Y[:,1:]
    u,s,vh = np.linalg.svd(YL,full_matrices=False)
    u = u[:,:k]; vh = vh[:k,:]; s = s[:k]
    AY = ct(u)@YR@ct(vh)@np.diag(1/s)
    YRYL = YR@ct(YL); YLYL = YL@ct(YL)
    ym = YR[:,-1:]
    return AY, YRYL, YLYL, Q, X, Y

def DMD_update(YRYL, YLYL, Q, X,Y, k, new_x, delta = 0.01):
    # this function updates the DMD modes
    # when additional snapshot data becomes available.
# input: YRYL,YLYL,Q,X,Y from previous iteration
# k: the target rank of the DMD
# new_x: the new snapshot data of shape n x s
# (we are allowing the number of new snapshots to be more than one)
# delta: the update threshold
# output: AY, YRYL, YLYL, Q, X, Y
l = YLYL.shape[0]; n = X.shape[0]; s = new_x.shape[1]
assert(new_x.shape[0] == n)
ym = Y[:,-1:]
for i in range(s): # check the new snapshot data one by one
    new_data = new_x[:,i:i+1]
    q = new_data - Q@(ct(Q)@new_data); q_norm = np.linalg.norm(q)
    rel_err = q_norm/np.linalg.norm(new_data)
    if rel_err <= delta: pass
    else: Q = np.block([Q,q/q_norm])
Q_old,Q_new = Q[:,:l],Q[:,l:]
new_y = ct(Q_old)@new_x
old_y = np.block([Y[:,-1:],new_y[:,:-1]])
YRYL_new = YRYL + new_y@ct(old_y);YLYL_new = YLYL + old_y@ct(old_y)
Y_new = np.block([Y,new_y])
if Q.shape[1] > l: # Q is updated
    print('Q is updated, we added '+str(Q.shape[1]-l)+' new dimensions')
    Y_tilde = ct(Q_new)@X
    YL_tilde,YR_tilde = Y_tilde[:,:-1], Y_tilde[:,1:]
    new_y_tilde = ct(Q_new)@new_x
    old_y_tilde = np.block([Y_tilde[:,-1:],new_y_tilde[:,:-1]])
    YL = Y[:,:-1]; YR = Y[:,1:]
    YRYL_new =
    np.block([ [YRYL_new, YR@ct(YL_tilde)+new_y@ct(old_y_tilde)],
    [YR_tilde@ct(YL)+new_y_tilde@ct(old_y),
    YR_tilde@ct(YL_tilde)+new_y_tilde@ct(old_y_tilde)]] )
    YLYL_new =
    np.block([ [YLYL_new, YL@ct(YL_tilde)+old_y@ct(old_y_tilde)],
    [YL_tilde@ct(YL)+old_y_tilde@ct(old_y) ,
    YL_tilde@ct(YL_tilde)+old_y_tilde@ct(old_y_tilde)]] )
    Y_new = np.block([[Y,new_y],[Y_tilde ,new_y_tilde]])
u,s,vh = np.linalg.svd(YLYL_new,full_matrices=False ,hermitian=True)
u = u[:,:k]; vh = vh[:k,:]; s = s[:k]
AY = ct(u)@YRYL_new@ct(vh)@np.diag(1/s)
return AY, YRYL_new, YLYL_new, Q, np.block([X,new_x]), Y_new
```

Bibliography

Berkooz, G., Holmes, P., and Lumley, J. L. (1993). The proper orthogonal decomposition in the analysis of turbulent flows. *Annual Review of Fluid Mechanics*, 25(1):539–575.

Cline, R. E. (1965). Representations for the generalized inverse of sums of matrices. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, 2(1):99–114.

Erichson, N. B., Mathelin, L., Kutz, J. N., and Brunton, S. L. (2019). Randomized dynamic mode decomposition. *SIAM Journal on Applied Dynamical Systems*, 18(4):1867–1891.

Halko, N., Martinsson, P.-G., and Tropp, J. A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288.

Koopman, B. O. (1931). Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318.

Kutz, J. N., Fu, X., and Brunton, S. L. (2016). Multiresolution dynamic mode decomposition. *SIAM Journal on Applied Dynamical Systems*, 15(2):713–735.

Musco, C. and Musco, C. (2015). Randomized block krylov methods for stronger and faster approximate singular value decomposition. *Advances in neural information processing systems*, 28.

Rowley, C. W., Mezić, I., Bagheri, S., Schlatter, P., and Henningson, D. S. (2009). Spectral analysis of nonlinear flows. *Journal of fluid mechanics*, 641:115–127.

Schimid, P. J. (2010). Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28.

Tu, J. H., , Rowley, C. W., Luchtenburg, D. M., Brunton, S. L., and and, J. N. K. (2014). On dynamic mode decomposition: Theory and applications. *Journal of Computational Dynamics*, 1(2):391–421.

Williams, M. O., Hemati, M. S., Dawson, S. T., Kevrekidis, I. G., and Rowley, C. W. (2016). Extending data-driven koopman analysis to actuated systems. *IFAC-PapersOnLine*, 49(18):704–709.

Zhang, H., Rowley, C. W., Deem, E. A., and Cattafesta, L. N. (2019). Online dynamic mode decomposition for time-varying systems. *SIAM Journal on Applied Dynamical Systems*, 18(3):1586–1609.