Study of Dynamic mode Decomposition on a Toy example

Karan Kabbur Hanumanthappa Manjunatha

December 30, 2022

Abstract

Dynamic Mode Decomposition(DMD) is an equation-free, data-driven method capable of providing an accurate decomposition of a complex system into spatiotemporal coherent structures that may be used for short-time future state prediction and control. In this work, we consider the toy model dataset which is combination of spatiotemporal signals and apply the DMD algorithm to obtain the DMD modes and spectra. We understand and interpretable results in terms of spatial structures and their associated temporal responses by comparing with the true solution to see the accuracy of the said method.

1 Introduction

In current research, data driven modeling and equation free dynamics of the complex systems has seen an unprecedented growth due to the availability of experimental time series data for variety of systems related to Finance, neuroscience, Plasma, epidemics and so on. The DMD algorithm was initially introduced by *Schmid*[3] and was applied in the field of fluid dynamics i.e. on the high dimensional fluid data. The DMD is powerful method and is successful because of its data driven and equation free method that decomposes the complex system accurately into spatiotemporal aspects which may be used for future state prediction and control.

Problem description: In this report, we focus on the DMD as a diagnostics tool to understand the high dimensional toy dataset spatio-temporal signals example. We try to extract low rank spatio-temporal features of the high dimensional dataset and compare it with the true solution to see its accuracy. A DMD reconstruction of the dataset was performed and visualized with the original dataset.

2 Theory

[2]Let us assume that the data is collected from a dynamical system given by the equation of the form:

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t; \mu) \tag{1}$$

where the meaning of the notations are as follows:
1. $\mathbf{x}(t) \in \mathbb{R}^n$ - represents the state of our dynamical system at time t and n is very large.
2. μ - represents parameters of the system
3. $\mathbf{f}()$ - represents the dynmaics of the system

We have the data but we do not know the dynamics $\mathbf{f}()$. Thus with the data measurements alone, we aim to approximate the dynamics of the systems and try to predict the future state with the equation free framework of DMD. The DMD procedure consists of modifying Eq. 1 into locally linear dynamical system with initial condition $\mathbf{x}(0)$ as follows:

$$\frac{d\mathbf{x}}{dt} = \mathcal{A}\mathbf{x} \tag{2}$$

The solution to Eq. 2 is given by[1]:

$$\mathbf{x}(t) = \sum_{k=1}^{n} \boldsymbol{\Phi}_k exp(\boldsymbol{\Omega}_k t) b_k = \boldsymbol{\Phi} exp(\boldsymbol{\Omega} t)\mathbf{b} \tag{3}$$

where the notations are:
1. $\boldsymbol{\Phi}_k, \boldsymbol{\Omega}_k$ - eigenvectors and eigenvalues of $\mathcal{A}$

2. b_k - coordinates of $\mathbf{x}(0)$ in eigenvector basis.
The discrete-time system analogous to continuos counterpart as in Eq. 2, sampled at time steps Δt in time:

$$\mathbf{x}_{k+1} = A\mathbf{x}_k \tag{4}$$

where we have matrix A:

$$A = exp(\mathcal{A}\Delta t) \tag{5}$$

The solution of discrete counterpart is given by:

$$\mathbf{x}_{k+1} = \sum_{j=1}^{r} \boldsymbol{\Phi}_j \boldsymbol{\Lambda}_j^k b_j = \boldsymbol{\Phi}\boldsymbol{\Lambda}^k \mathbf{b} \tag{6}$$

The notations are:
1. $\boldsymbol{\Phi}_k, \boldsymbol{\Lambda}_k$ - eigenvectors and eigenvalues of discrete matrix A
2. $\mathbf{b}$ - coefficients of the initial condition $\mathbf{x}_1$ in eigenvector basis given by $\mathbf{x}_1 = \boldsymbol{\Phi}\mathbf{b}$

The DMD algorithm produces a low-rank eigendecomposition Eq. 6of the matrix A that optimally fits the measured trajectory $\mathbf{x}_k$ for $k = 1, 2, ..., m$ in a least-square sense so that

$$||\mathbf{x}_{k+1} - A\mathbf{x}_k|| \tag{7}$$

is minimized across all the points for $k = 1, ..., m-1$. The minimization of error as given in Eq. 7 across all values of k is done by considering it into two large data matrices given by: $\mathbf{X} = \begin{pmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{m-1} \\ | & | & & | \end{pmatrix}$ and $\mathbf{X}' = \begin{pmatrix} | & | & & | \\ \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_m \\ | & | & & | \end{pmatrix}$ The locally linear approximation given by Eq. 2 written in terms of these matrices as:

$$\mathbf{X}' \approx A\mathbf{X} \tag{8}$$

The best fit A is given by:

$$A = \mathbf{X}'\mathbf{X}^{\dagger} \tag{9}$$

where $\dagger$ is the *Moore-Penrose pseudoinverse*. The DMD algorithm comes into picture here where instead of solving directly for A from the Eq. 9 which may lead to matrix being high dimensional, we project the data onto a low-rank subspace defined bu atmost $m-1$ *Proper Orthogonal Decomposition*(POD) modes and then solve for low dimensional evolution of $\tilde{A}$ that evolves on these POD coefficients. The DMD algorithm then uses this low-dimensional operator $\tilde{A}$ to reconstruct the leading nonzero eigenvalues and eigenvectors of the full-dimensional operator A.

3 Methods

The **DMD algorithm** implemented is as follows: Consider the spatio-temporal data matrix $\mathbf{X} \in \mathcal{R}^{n \times m}$, where n is the number of spatial points saved per time snapshot or in other words the dimensionality of the state space and m being the number of snapshots taken.

1. **Create DMD data matrices:** Create data matrix by considering the time snapshots $k = 1, ..., m-1$ i.e. $\mathbf{X} = \begin{pmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{m-1} \\ | & | & & | \end{pmatrix}$

   and its time shifted counterpart $k = 2, ..., m$ i.e. $\mathbf{X}' = \begin{pmatrix} | & | & & | \\ \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_m \\ | & | & & | \end{pmatrix}$
2. **Compute the *Singular Value Decomposition*(SVD) on $\mathbf{X}$**: $\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^*$ and perform the rank truncation by considering only r reduced SVD approximation of $\mathbf{X}$. The notations are:

   (a) $\mathbf{U} \in C^{n\times r}$ - The left singular vectors which are POD modes.
   (b) $\boldsymbol{\Sigma} \in C^{r\times r}$ - The diagonal matrix of singular values in decreasing order.
   (c) $\mathbf{V} \in C^{n\times r}$ - The right singular vectors
   (d) $*$ denotes the conjugate transpose of the matrix.
3. **Compute the best estimate** A by using the pseudo inverse of $\mathbf{X}$: $A = \mathbf{X'V\Sigma}^{-1}\mathbf{U}^* = \mathbf{XX}^\dagger$
4. **Compute the reduced ordered model:** We compute the $r \times r$ projection of the full matrix A onto the POD modes i.e. *Koopman matrix* $\tilde{A}$ of $dim = (r \times r)$: $\tilde{A} = \mathbf{U}^*A\mathbf{U} = \mathbf{U}^*\mathbf{X'V\Sigma}^{-1}$
5. **Compute eigendecomposition of $\tilde{A}$:** $\tilde{A}\mathbf{W} = \mathbf{W}\boldsymbol{\Lambda}$ where:
   (a) $\mathbf{W}$ - eigenvectors of $\tilde{A}$
   (b) $\boldsymbol{\Lambda}$ - diagonal matrix containing the eigen values $\boldsymbol{\Lambda}_k$ of the matrix $\tilde{A}$.
6. These eigenvalues matrix $\boldsymbol{\Lambda}$ is exactly the same for the matrix A. The **DMD spectra** is given by:
   (a) Discrete time eigenvalue matrix $\boldsymbol{\Lambda}$ whose diagonal entries are $\boldsymbol{\Lambda}_k$, $k = 1, ...r$
   (b) The continuous counterpart i.e. Continuous time-eigenvalue matrix $\boldsymbol{\Omega}$ whose diagonal entries are computed by: $\boldsymbol{\Omega}_k = \frac{ln(\boldsymbol{\Lambda}_k)}{\Delta t}$.
7. **Compute the DMD mode amplitudes:** The initial condition $\mathbf{x}_1 = \boldsymbol{\Phi}\mathbf{b}$ where $\mathbf{b}$ is the initial coefficient value that has to be computed. As $\boldsymbol{\Phi}$ is generally not a square matrix, we can obtain the DMD mode amplitudes by the equation: $\mathbf{b} = \boldsymbol{\Phi}^\dagger\mathbf{x}_1$
8. **DMD reconstruction:** Given the low rank approximation of both the eigenvalues $\boldsymbol{\Omega}$ and DMD eigenvectors $\boldsymbol{\Phi}$, we can reconstruct the future solutions by the equation:

$$\mathbf{x}(t) \approx \sum_{k=1}^{r} \boldsymbol{\Phi}_k exp(\boldsymbol{\Omega}_k t) b_k = \boldsymbol{\Phi} exp(\boldsymbol{\Omega} t)\mathbf{b} \tag{10}$$

4 Results

I have considered dataset from a toy model by mixing two spatiotemporal signals. The aim of this assignment is to demonstrate the ability of DMD to efficiently decompose the signal into its constituent parts. The two signals considered are:

1.
$$f_1(x, t) = sech(x + 3)exp(i2.3t) \tag{11}$$
   with frequency $\omega_1 = 2.3$
2.
$$f_2(x, t) = 2sech(x)tanh(x)exp(i2.8t) \tag{12}$$
   with frequency $\omega_2 = 2.8$

The mixed signal is simply a sum of the two signals:

$$f(x, t) = f_1(x, t) + f_2(x, t) \tag{13}$$

The dataset $\mathbf{X}$ which is sum of mixed signals Eq.13, is obtained by taking 400 evenly spaced x values in the range -10 to 10 and 200 evenly spaced t values in the range 0 to 4π. The spatio-temporal signals $f_1(x, t)$, $f_2(x, t)$ and $f(x, t)$ are visualized in the Figures. 1a, 1b, 1c respectively.

From the Fig. 2 we observe that the decay of singular values of $\mathbf{X}$. This shows that the data may be appropriately represented as rank $r = 2$.

The DMD of $\mathbf{X}$ was computed, and a rank-2 approximate reconstruction given by Eq. 10 of the signal $\mathbf{X_{DMD}}(t)$ is shown in Fig. 1d. The reconstruction seems to be almost close to the original Fig. 1c.

From the Figs. 3, 4, we observe that spatial and temporal modes extracted from DMD almost closely match with the true solutions $f_1(x, t)$ and $f_2(x, t)$. DMD produces results that are exactly (to numerical precision) aligned with the true solution. The error computed as*l2-norm* difference between the two true spatial modes and modes extracted by DMD are 6.4×10^{-14} and 6.3×10^{-14} respectively and are really close to zero.

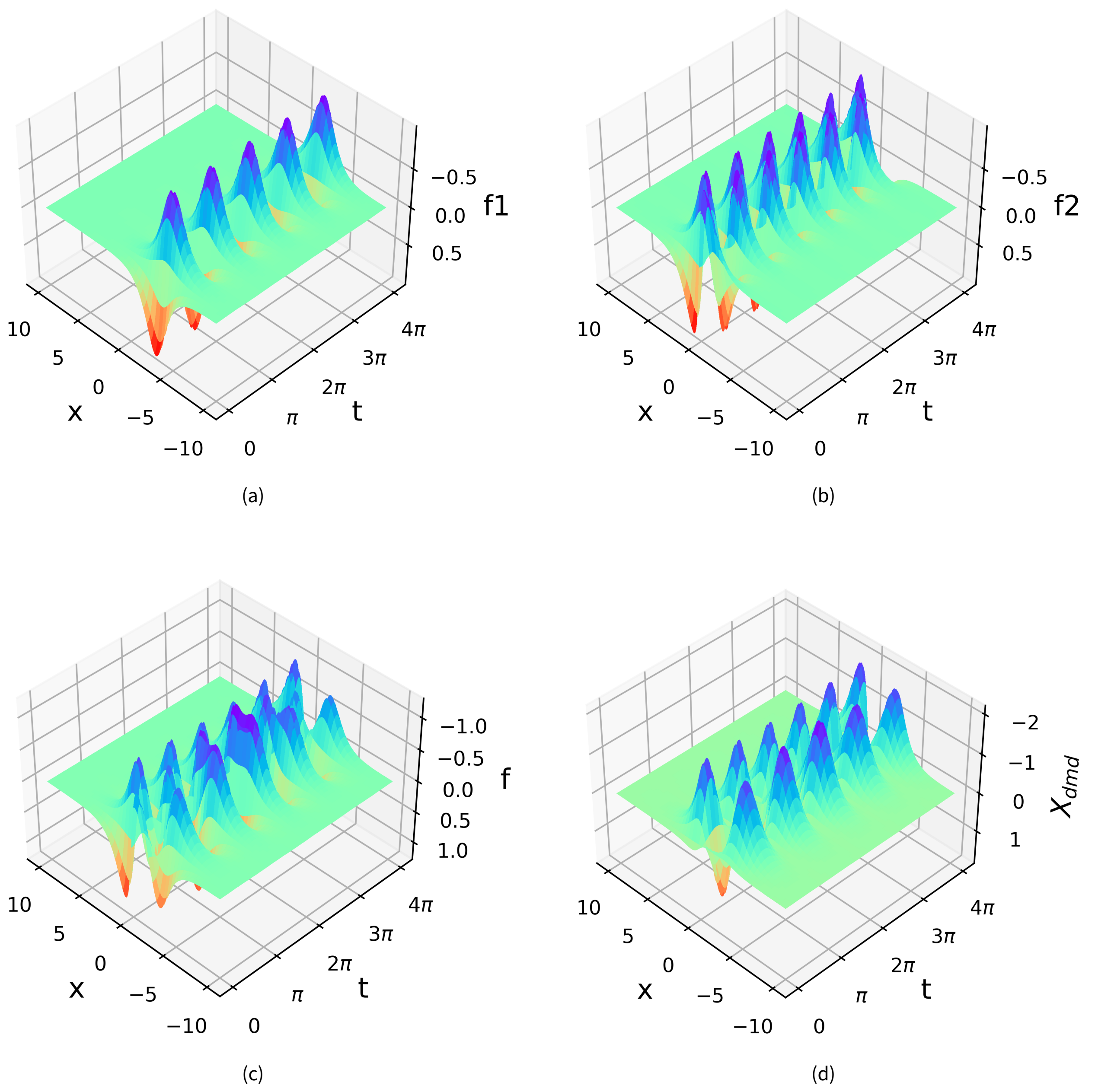

Figure 1. (a). Spatiotemporal signal given by $f_1(x,t)$, Eq.11, **(b).** Spatiotemporal signal given by $f_2(x,t)$, Eq.12, **(c).** Spatiotemporal signal given by $f(x,t) = f_1(x,t) + f_2(x,t)$, Eq.13, **(d).** The DMD rank-2 approximate reconstruction of the dataset **X**.

5 Conclusions

With toy model dataset considered in the report:

1. The rank-2 approximate reconstruction by DMD algorithm is almost perfect.
2. DMD produces results that are exactly (to numerical precision) aligned with the true solution when the spatial and temporal aspects of modes extracted from DMD and the original data were compared.

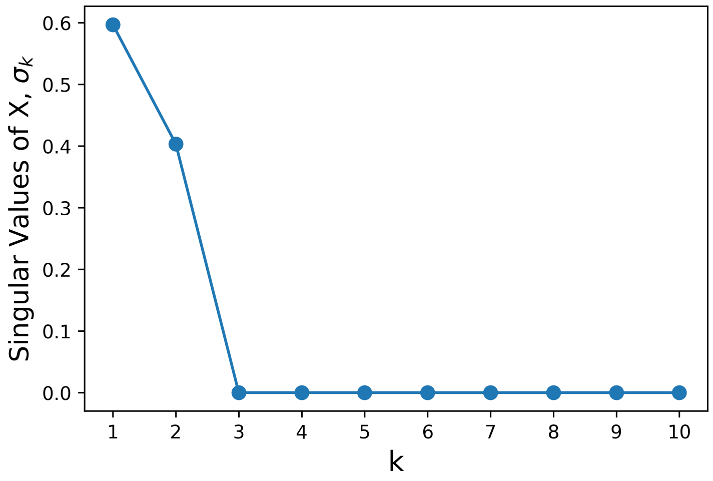

Figure 2. Singular values of **X** show that a rank-2 truncation is appropriate.

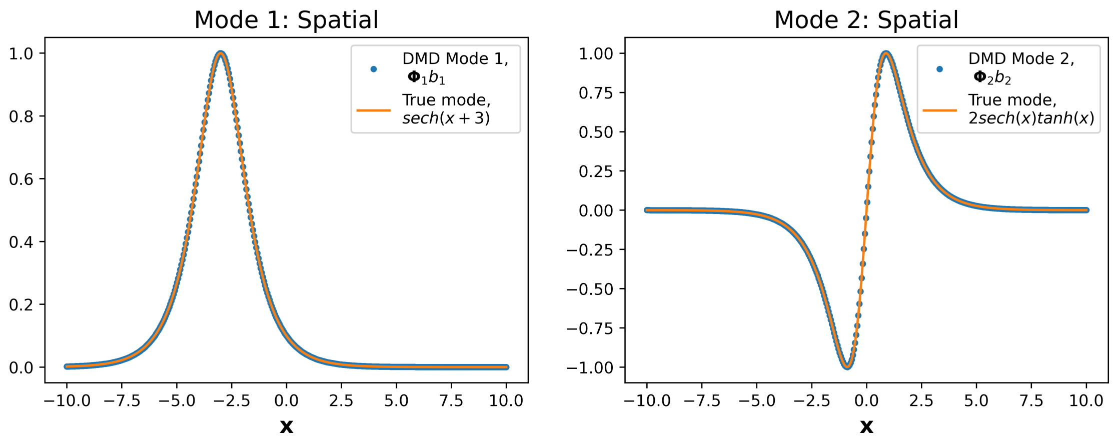

Figure 3. Spatial part of the first two modes along with the True solution obtained from Eqns. 11, 12. DMD produces results that are exactly the same as the true solution.

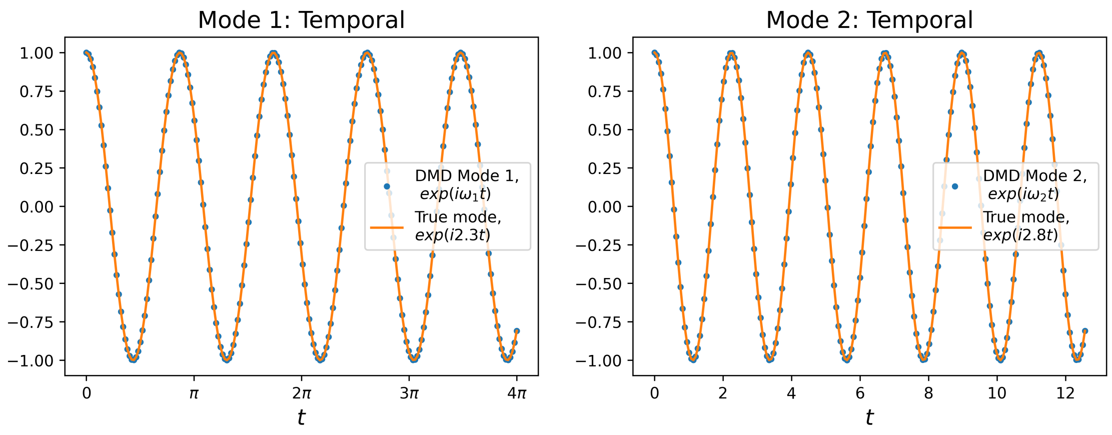

Figure 4. Temporal part of the first two modes along with the True solution obtained from Eqns. 11, 12. DMD produces results that are exactly the same as the true solution.

References

[1] W. E. Boyce and R. C. DiPrima. *Elementary Differential Equations*. 9th Ed. Wiley, 2008.

[2] J. Nathan Kutz et al. *Dynamic Mode Decomposition*. Society for Industrial and Applied Mathematics, Nov. 2016. DOI: 10.1137/1.9781611974508. URL: https://doi.org/10.1137/1.9781611974508.

[3] PETER J. SCHMID. “Dynamic mode decomposition of numerical and experimental data”. In: *Journal of Fluid Mechanics* 656 (July 2010), pp. 5–28. DOI: 10.1017/s0022112010001217. URL: https://doi.org/10.1017/s0022112010001217.

6 Appendix

```python

import numpy as np
import matplotlib.pyplot as plt

#### DATASET #####
# Define time and space discretizations
xi = np.linspace ( -10 ,10 ,400)
t = np.linspace (0 ,4*np.pi ,200)
dt = t[1] - t[0] ;
Xgrid ,T = np.meshgrid(xi ,t)

## Create two spatiotemporal patterns
f1 = np.reciprocal(np.cosh( Xgrid +3)) * (1* np.exp (1j*2.3* T))
f2 = np.reciprocal(np.cosh(Xgrid)) * np.tanh(Xgrid) * (2* np.exp (1j *2.8*T))

# combine signals and make data
f = f1+f2
X = f.T
```

Listing 1. Create Spatiotemporal signals data

```python
r=2 # rank 2 truncation

# Create DMD data matrices
# X1 = X, data matrix
# X2 =  X  , shifted data matrix
X1 = X[:, :-1]# 1 to m-1
X2 = X[:, 1:] # 2 to m

## Perform singular value decomposition on X1
U, S, Vh = np.linalg.svd(X1, full_matrices = False)
# rank truncation
Ur = U[:, : r]
Sr = S[: r]
V = Vh.conj().T # Vh is conj transpose, we need V
Vr = V[:, :r]


## Compute the Koopman matrix Atilde and DMD modes
# @ represents matrix multiplication
A_tilde = Ur.conj().T @ X2 @ (Vr*np.reciprocal(Sr))
## Perform eigenvalue decomposition on A_tilde
# returns eigvals and eigvectors
D, W = np.linalg.eig(A_tilde)
srt = np.argsort(D)[::-1] # indices of eigenvalues from largest to smallest
D = D[srt] # sorted eigenvalues
W = W[:, srt] # sorted eigenvectors

# DMD modes
Phi = X2 @ Vr @ np.diag(np.reciprocal(Sr)) @ W

```

```python
## DMD Spectra
lamda = D#discrete -time eigenvalues
omega = np.round( np.diag(np.log(lamda))/dt, 4)# Continuos time eigenvalues

## Compute DMD mode amplitudes b
x1 = X1[:, 0]
b = np.linalg.pinv(Phi)@x1#(Phi.T/ x1).T


## DMD reconstruction
time_dynamics = np.zeros((r, len(t)))
for i in range(len(t)):
    time_dynamics[:,i] = np.exp( omega*t[i])@b

Xdmd = Phi @ time_dynamics
```

Listing 2. DMD algorithm applied to dataset

```python
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(xi, np.real(Phi[:,0]*b[0]),".", label="DMD Mode 1, \n $\mathbf{\Phi}_1 b_1$")
true_spatial_mode1 = np.reciprocal(np.cosh( xi +3))

plt.plot(xi, np.real(true_spatial_mode1), label="True mode, \n$sech(x+3)$")
plt.legend(loc="best")
plt.xlabel("$\mathbf{x}$", size=14)
plt.title("Mode 1: Spatial", size=15)


plt.subplot(1,2,2)

plt.plot(xi, np.real(Phi[:,1]*b[1]),".", label="DMD Mode 2, \n $\mathbf{\Phi}_2 b_2$")
true_spatial_mode2 = 2*np.reciprocal(np.cosh(xi)) * np.tanh(xi)

plt.plot(xi, np.real(true_spatial_mode2), label="True mode, \n$2 sech(x) tanh(x)$")
plt.legend(loc="best")
plt.xlabel("$\mathbf{x}$", size=14)
plt.title("Mode 2: Spatial", size=15)
plt.savefig("spatial_modes.png", dpi=600, facecolor="white", bbox_inches="tight")
```

Listing 3. Visualize and compare true and DMD spatial modes

```python
fig,ax = plt.subplots(1,2,figsize=(12,4))
ax[0].plot(t,np.exp( omega[0,0]*t) ,".", label="DMD Mode 1, \n $exp(i\omega_1t)$")
true_temp_mode1 = np.exp (1j*2.3* t)

ax[0].plot(t, true_temp_mode1, label="True mode, \n$exp(i2.3t)$")
ax[0].legend(loc="best")
ax[0].set_xlabel("$t$", size=14)
ax[0].set_title("Mode 1: Temporal", size=15)
ax[0].xaxis.set_major_locator(plt.MultipleLocator(np.pi))
ax[0].xaxis.set_major_formatter(plt.FuncFormatter(multiple_formatter()))

ax[1].plot(t,np.exp( omega[1,1]*t) ,".", label="DMD Mode 2, \n $exp(i\omega_2t)$")
true_temp_mode2 = np.exp (1j*2.8* t)

ax[1].plot(t, true_temp_mode2, label="True mode, \n$exp(i2.8t)$")
ax[1].legend(loc="best")
ax[1].set_xlabel("$t$", size=14)
ax[1].set_title("Mode 2: Temporal", size=15)
plt.savefig("temporal_modes.png", dpi=600, facecolor="white", bbox_inches="tight")
```

Listing 4. Visualize and compare true and DMD temporal modes

```python
# plot of singular values
plt.plot(range(1,11),S[:10]/sum(S),"-o", markersize=7)
plt.xlabel("k", size=15)
plt.ylabel("Singular Values of X, $\sigma_k$", size=14)
plt.xticks(range(1,11), range(1,11))
plt.savefig("sing_vals.png", dpi=600, facecolor="white", bbox_inches="tight")

plt.show()
```

Listing 5. Plot the singular values of X

```python


def multiple_formatter(denominator=2, number=np.pi, latex='\pi'):
    def gcd(a, b):
        while b:
            a, b = b, a%b
        return a
    def _multiple_formatter(x, pos):
        den = denominator
        num = np.int(np.rint(den*x/number))
        com = gcd(num,den)
        (num,den) = (int(num/com),int(den/com))
        if den==1:
            if num==0:
                return r'$0$'
            if num==1:
                return r'$%s$'%latex
            elif num==-1:
                return r'$-%s$'%latex
            else:
                return r'$%s%s$'%(num,latex)
        else:
            if num==1:
                return r'$\frac{%s}{%s}$'%(latex,den)
            elif num==-1:
                return r'$\frac{-%s}{%s}$'%(latex,den)
            else:
                return r'$\frac{%s%s}{%s}$'%(num,latex,den)
    return _multiple_formatter

class Multiple:
    def __init__(self, denominator=2, number=np.pi, latex='\pi'):
        self.denominator = denominator
        self.number = number
        self.latex = latex

    def locator(self):
        return plt.MultipleLocator(self.number / self.denominator)

    def formatter(self):
        return plt.FuncFormatter(multiple_formatter(self.denominator, self.number, self.latex))


######  f1 ##########
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
surf = ax.plot_surface(Xgrid, T, np.real(f1), linewidth=0, antialiased=False,cmap="rainbow" )
# Set an equal aspect ratio
#ax.set_aspect("auto")

ax.yaxis.set_major_locator(plt.MultipleLocator(np.pi))
ax.yaxis.set_major_formatter(plt.FuncFormatter(multiple_formatter()))
ax.set_xlabel("x", size=14)
ax.set_ylabel("t", size=14)
ax.set_zlabel("f1", size=14)
ax.view_init(-140, 45)
plt.savefig("f1.png", dpi=600, facecolor="white", bbox_inches="tight")


####### f2 ##########
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
surf = ax.plot_surface(Xgrid, T, np.real(f2), linewidth=0, antialiased=False,cmap="rainbow" )
# Set an equal aspect ratio
#ax.set_aspect("auto")

ax.yaxis.set_major_locator(plt.MultipleLocator(np.pi))
ax.yaxis.set_major_formatter(plt.FuncFormatter(multiple_formatter()))
ax.set_xlabel("x", size=14)
ax.set_ylabel("t", size=14)
ax.set_zlabel("f2", size=14)
ax.view_init(-140, 45)
plt.savefig("f2.png", dpi=600, facecolor="white", bbox_inches="tight")


########  f  ###############
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
surf = ax.plot_surface(Xgrid, T, np.real(f), linewidth=0, antialiased=False,cmap="rainbow" )
# Set an equal aspect ratio
#ax.set_aspect("auto")

ax.yaxis.set_major_locator(plt.MultipleLocator(np.pi))
ax.yaxis.set_major_formatter(plt.FuncFormatter(multiple_formatter()))
ax.set_xlabel("x", size=14)
ax.set_ylabel("t", size=14)
ax.set_zlabel("f", size=14)
ax.view_init(-140, 45)
plt.savefig("f.png", dpi=600, facecolor="white", bbox_inches="tight")


##############  DMD reconstruction Xdmd  ############
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
surf = ax.plot_surface(Xgrid, T, np.real(Xdmd.T), linewidth=0, antialiased=False,cmap="rainbow" )
# Set an equal aspect ratio
#ax.set_aspect("auto")

ax.yaxis.set_major_locator(plt.MultipleLocator(np.pi))
ax.yaxis.set_major_formatter(plt.FuncFormatter(multiple_formatter()))
ax.set_xlabel("x", size=14)
ax.set_ylabel("t", size=14)
ax.set_zlabel("$X_{dmd}$", size=14)
ax.view_init(-140, 45)
plt.savefig("dmd_recons.png", dpi=600, facecolor="white", bbox_inches="tight")
```

Listing 6. Visualize the spatiotemporal signals $f1(x,t)$, $f2(x,t)$, $f(x,t)$ and DMD reconstruction $\mathbf{X}_{DMD}$