Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Chapter 10
DMD on Nonlinear Observables

An underlying assumption made throughout the book concerns the choice of observables and data used to execute the DMD algorithm. An implicit assumption is that the measured variables are, in fact, the correct variables to use. In many cases, this is often a reasonable assumption. For instance, one may measure the pressure and/or velocity in a fluid flow and be confident that executing the DMD algorithm on such variables can yield meaningful spatiotemporal patterns and predictions. Nonlinearity in the underlying system, however, can challenge our assumptions on simply using the measurements directly in DMD. Koopman theory suggests that a broader set of observables, namely functions of the measurements, may be more useful for characterizing the dynamics. The suggestion is that by potentially picking a broader set of observables, information may be gained on the underlying nonlinear manifold on which the nonlinear dynamical system evolves. Manifold learning is a current topic of intensive research interest in the machine learning community. Its goal is to more appropriately characterize the space on which data is embedded. In this chapter, we highlight how DMD can be modified through the choice of observables to potentially account for the nonlinear manifolds on which dynamics occur.

10.1 ▪ Koopman observables

Machine learning methods are of growing importance across the physical, engineering, and biological sciences. Although many of the techniques developed are primarily for clustering and classification purposes [204, 84, 29], some have been modified to handle complex spatiotemporal phenomena underlying dynamical systems. Not surprisingly, machine learning methods have also been integrated with the DMD infrastructure.

The integration of machine learning with DMD arises naturally within the framework of the Koopman operator and its observables, as defined in (3.18):

$$\mathscr{K}\mathbf{g}(\mathbf{x}) = \mathbf{g}(\mathbf{f}(\mathbf{x})) \,. \tag{10.1}$$

Recall that the Koopman operator is a *linear* operator that acts on scalar functions g_j that comprise a set of observables denoted by components of the vector $\mathbf{g}$. Let us once

Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

again consider these observables:

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} g_1(\mathbf{x}) \\ g_2(\mathbf{x}) \\ \vdots \\ g_n(\mathbf{x}) \end{bmatrix}. \tag{10.2}$$

The choice of appropriate observables g_j is often difficult to evaluate without expert knowledge. However, using ideas from the highly popular support vector machine (SVM) clustering algorithm, which is a subset of so-called kernel methods [53, 198, 271, 21], a principled technique for judiciously choosing the observables can be formulated. In particular, one can think of the $\mathbf{g}(\mathbf{x})$ as a mapping from the *physical space* to the *feature space*. In the feature space, an *intrinsic* representation of the underlying dynamical system is constructed by the triplet pair of Koopman eigenvalues λ_k, Koopman eigenfunctions $\varphi_k(\mathbf{x})$, and Koopman modes $\mathbf{v}_k$:

$$\mathbf{g}(\mathbf{x}) = \sum_{k=1}^{\infty} \mathbf{v}_k \varphi_k(\mathbf{x}), \tag{10.3a}$$

$$\mathbf{g}(\mathbf{f}(\mathbf{x})) = \sum_{k=1}^{\infty} \mathbf{v}_k \lambda_k \varphi_k(\mathbf{x}). \tag{10.3b}$$

Thus, the time evolution of solutions can be computed by simple multiplication with the Koopman eigenvalue in the space of Koopman eigenfunctions. Importantly, the Koopman operator thus captures the intrinsic properties of the nonlinear dynamical system (3.17), and its eigenfunctions define a nonlinear change of coordinates in which the system becomes linear. Although this was presented previously in the Koopman chapter, the interpretation here is in terms of a feature space. Or, more precisely, the observables $\mathbf{g}(\mathbf{x})$ define a linear evolution of a feature space.

The SVM literature and kernel methods [53, 198, 271, 21] suggest a number of techniques for constructing the feature space $\mathbf{g}(\mathbf{x})$. One common choice is the set of polynomials such that

$$g_j(x) = \left\{x, x^2, x^3, x^4, \ldots, x^n\right\}. \tag{10.4}$$

Alternatively, kernel methods have found a high degree of success using (i) radial basis functions, typically for problems defined on irregular domains; (ii) Hermite polynomials for problems defined on $\mathbb{R}^n$; and (iii) discontinuous spectral elements for large problems with block diagonal structures.

Regardless of the specific choice of feature space, the goal is to choose a sufficiently rich and diverse set of observables that allow an accurate approximation of the Koopman operator $\mathcal{K}$. Instead of choosing the correct observables, one then simply chooses a large set of candidate observables with the expectation that a sufficiently diverse set will include enough features for an accurate reconstruction of the triplet pair (10.3), which intrinsically characterizes the nonlinear dynamical system under consideration.

10.2 ▪ Nonlinear observables for partial differential equations

The examples in the Koopman chapter were easy to quantify and understand since they were chosen to elucidate the principles of the Koopman decomposition in relation to DMD. Moreover, the observables chosen were easily motivated from knowledge

Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

of the solution. A more realistic example comes from a partial differential equation example where analytic insight is more difficult to obtain. Thus, to further illustrate the Koopman decomposition in relation to DMD, consider the nonlinear Schrödinger (NLS) equation

$$i\frac{\partial q}{\partial t}+\frac{1}{2}\frac{\partial^2 q}{\partial \xi^2}+|q|^2 q=0, \tag{10.5}$$

where $q(\xi,t)$ is a function of space and time and a specific example of (1.36). The equation can be rewritten in the form

$$\frac{\partial q}{\partial t}=\frac{i}{2}\frac{\partial^2 q}{\partial \xi^2}+i|q|^2 q, \tag{10.6}$$

so that a spectral method solution can be applied. Fourier transforming in ξ, denoted by the hat symbol, gives the differential equation in the Fourier domain variables $\hat{q}$:

$$\frac{d\hat{q}}{dt}=-\frac{ik^2}{2}\hat{q}+i\widehat{|q|^2 q}. \tag{10.7}$$

By discretizing in the spatial variable, we can then evolve this equation with a standard time-stepping algorithm such as a fourth-order Runge–Kutta integrator. The following code generates a numerical approximation to the solution of (10.5).

***ALGORITHM* 10.1. NLS equation.**

```
%% Define spatial discretization
L=30; n=512;  %% Domain length and # points
xi2=linspace(-L/2,L/2,n+1);  %% domain discretization
xi=xi2(1:n);  %% periodic domain
k=(2*pi/L)*[0:n/2-1 -n/2:-1].';  %%  wavenumbers

%% Define time discretization
slices=20;
t=linspace(0,pi,slices+1); dt=t(2)-t(1);

%% Create initial conditions
q=2*(sech(xi)).';
qt=fft(q);
%% Combine signals

%% Solve with Runge-Kutta
[t,qtsol]=ode45('dmd_soliton_rhs',t,qt,[],k);

%% Bring back to time domain and store data
for j=1:length(t)
   qsol(j,:)=ifft(qtsol(j,:));
end
X = qsol.';
```

This code provides a complete numerical solution of the NLS equation (10.5), where the right-hand side of (10.7) is computed as follows.

Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

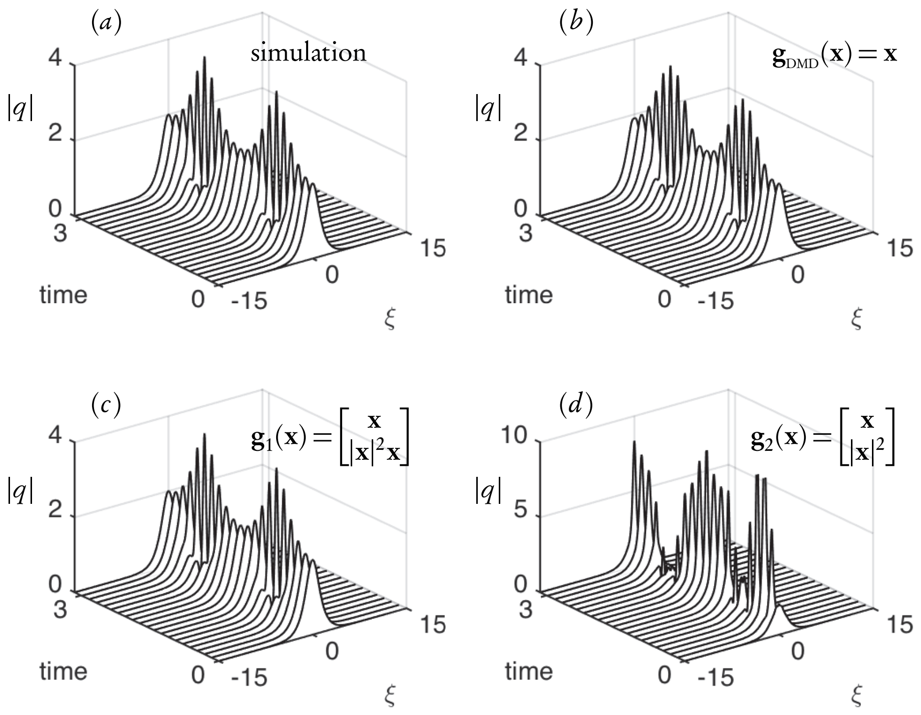


Figure 10.1. *Koopman reconstruction of the numerical simulation of the NLS equation* (10.5)*. The full simulation is shown in panel* (a) *along with three different reconstructions using different observables. The reconstruction in panel* (b) *uses the standard DMD algorithm, where the observable is given by* $\mathbf{g}_{DMD}(\mathbf{x}) = \mathbf{x}$*, where* $\mathbf{x} = q(\xi, t)$*. Two additional observables are considered in panels* (c) *and* (d)*, corresponding to* $\mathbf{g}_1(\mathbf{x}) = \begin{bmatrix}\mathbf{x} & |\mathbf{x}|^2\mathbf{x}\end{bmatrix}^T$ *and* $\mathbf{g}_2(\mathbf{x}) = \begin{bmatrix}\mathbf{x} & |\mathbf{x}|^2\end{bmatrix}^T$*, respectively. As shown in Figure* 10.3*, the observable vector* $\mathbf{g}_1(\mathbf{x})$*, which is based on the NLS nonlinearity, gives a superior reconstruction, highlighting the impact of a judicious choice of observables.*

Algorithm 10.2. Right-hand side of NLS equation.

```
function rhs=dmd_soliton_rhs(t,qt,dummy,k)
q=ifft(qt);
rhs=-(i/2)*(k.^2).*qt+i*fft( (abs(q).^2).*q );
```

Assumed here is that there is a total of 21 slices of time data over the interval $t \in [0, 2\pi)$. The state variables are an n-dimensional discretization of $q(\xi, t)$ so that $q(\xi, t_k) \rightarrow \xi_k$, where $n = 512$. These ξ_k are the columns of the generated data matrix $\mathbf{X}$; i.e., they are the state variable $\mathbf{x}$. In this specific example, we analyze the two-soliton solution that has the initial condition $q(\xi, 0) = 2\text{sech}(\xi)$. The output of this simulation is shown in Figure 10.1(a).

We now consider a dynamic mode decomposition and reconstruction of the dynamics as exemplified by the simulation. The DMD algorithm follows that outlined in the introduction. Here a low-rank approximation is given ($r = 10$).

ALGORITHM 10.3. **DMD on NLS equation.**

```
X1 = X(:,1:end-1); %% data snapshots
X2 = X(:,2:end); %% shifted data

[U2,Sigma2,V2] = svd(X1, 'econ');
r=10;  %% rank 10 truncation
U=U2(:,1:r); Sigma=Sigma2(1:r,1:r); V=V2(:,1:r);
Atilde = U'*X2*V/Sigma;
[W,D] = eig(Atilde);
Phi=X2*V/Sigma*W;

lambda=diag(D);
omega=log(lambda)/dt;

y0 = Phi\u;  %% project on initial conditions

q_modes = zeros(r,length(t));  %% DMD modes
for iter = 1:length(tf)
    q_modes(:,iter) =(y0.*exp(omega*(tf(iter))));
end

q_dmd = Phi*q_modes;   %% DMD approximation
```

Figure 10.1(b) shows the standard DMD approximation achieved. Explicitly assumed in the DMD reduction is the fact that the observables are simply the state variables $\mathbf{x}$, where $\mathbf{x} = q(\xi, t)$. More precisely, for DMD, we have

$$\mathbf{g}_{\text{DMD}}(\mathbf{x}) = \mathbf{x} \tag{10.8}$$

for the observables. Thus, the DMD approximation is a special case of Koopman. The DMD modes ($r = 10$) and DMD spectrum are shown in the left panel of Figure 10.2. An ideal approximation would have the eigenvalues aligned along the imaginary axis.

Koopman theory, however, allows for a much broader set of observables. In what follows, we consider two additional observables:

$$\mathbf{g}_1(\mathbf{x}) = \begin{bmatrix} \mathbf{x} \\ |\mathbf{x}|^2\mathbf{x} \end{bmatrix}, \tag{10.9a}$$

$$\mathbf{g}_2(\mathbf{x}) = \begin{bmatrix} \mathbf{x} \\ |\mathbf{x}|^2 \end{bmatrix}. \tag{10.9b}$$

The first observable, $\mathbf{g}_1(\mathbf{x})$, is inspired by the form of the nonlinearity in the NLS equation. The second, $\mathbf{g}_2(\mathbf{x})$, is chosen to have a simple quadratic nonlinearity. It has no special relationship to the governing equations. Note that the choice of the observable $|\mathbf{x}|^2$ in $\mathbf{g}_2(\mathbf{x})$ is relatively arbitrary. For instance, one can consider, instead of $|\mathbf{x}|^2$, a function such as $|\mathbf{x}|^5\mathbf{x}$, $\mathbf{x}^2$, $\mathbf{x}^3$, or $\mathbf{x}^5$. These all produce similar results to the $\mathbf{g}_2(\mathbf{x})$ selected in (10.9b). Specifically, the observable $\mathbf{g}_2(\mathbf{x})$ is inferior to either the DMD or judiciously selected $\mathbf{g}_1(\mathbf{x})$ for the Koopman reconstruction.

As has been repeatedly stated, success of the Koopman decomposition relies almost exclusively on the choice of observables. Moreover, in practice the Koopman decomposition is performed on the set of observables, not the state-space variables. To demonstrate this, we compute the Koopman decomposition of the NLS equation (10.5) using the two observables (10.9). We illustrate the code on the observables $\mathbf{g}_1(\mathbf{x})$.

Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

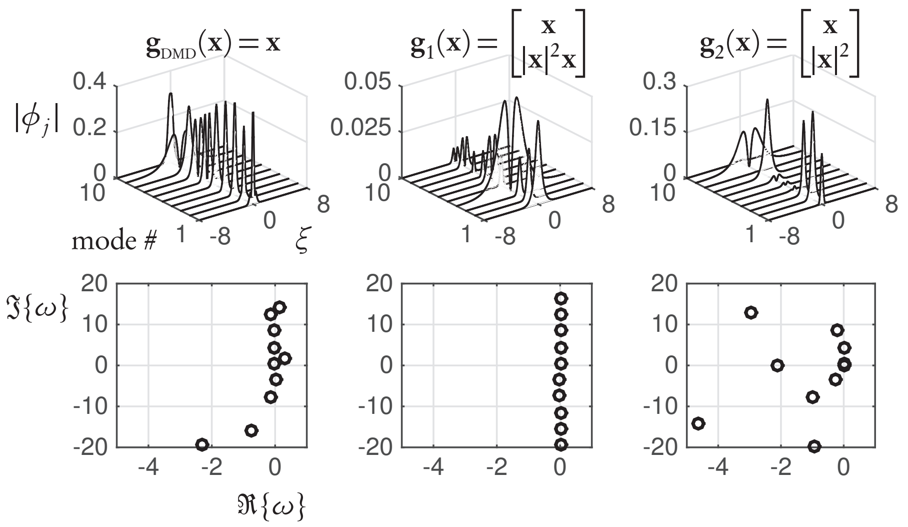


Figure 10.2. *Koopman eigenfunctions and eigenvalue distribution for the three observables considered in Figure* 10.1: *the standard DMD algorithm* $\mathbf{g}_{DMD}(\mathbf{x}) = \mathbf{x}$ *and* $\mathbf{g}_1(\mathbf{x}) = \left[\mathbf{x} \;\; |\mathbf{x}|^2\mathbf{x}\right]^T$, $\mathbf{g}_2(\mathbf{x}) = \left[\mathbf{x} \;\; |\mathbf{x}|^2\right]^T$. *Note that the observable* $\mathbf{g}_1(\mathbf{x})$ *gives the best approximation to the expected spectrum of purely imaginary eigenvalues.*

ALGORITHM **10.4. Koopman data of observables.**

```
Y1=[X1; (X1.*abs(X1).^2)];  %% Data observables g1
Y2=[X2; (X2.*abs(X2).^2)];  %% Shifted Data
```

Once the data matrices are created, which now have $2n$ rows of data, the DMD algorithm outlined previously can be run with the matrices Y1 and Y2 instead of X1 and X2. Near the reconstruction stage, only the state variables need to be recovered, which is done with the following code.

ALGORITHM **10.5. Koopman reconstruction.**

```
q2=[q; (q.*abs(q).^2)];
y0 = Phi2\q2;

q_modes = zeros(r,length(t));
for iter = 1:length(t)
q_modes(:,iter) =(y0.*exp(omega*(t(iter))));
end
q_dmd2 = Phi2*q_modes;

q_dmd = q_dmd2(1:n,:); %% Koopman approximation
Phi = Phi2(1:n,:); %% Koopman eigenfunctions
```

The algorithm produces both a state approximation, since the first n components are actually the state vector $\mathbf{x}$, and the associated Koopman eigenfunctions and eigenvalues. For the observables $\mathbf{g}_2(\mathbf{x})$, the only modification is in the data construction.

Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

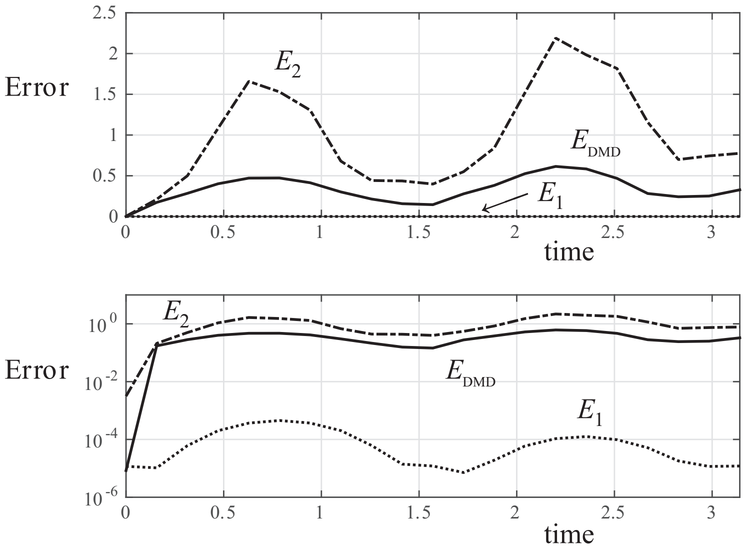


Figure 10.3. *Error analysis of the Koopman reconstruction of the simulated data shown in Figure* 10.1(a). *The three observables considered in panels* (b)–(d) *of Figure* 10.1 *are compared against the true solution. The observable corresponding to standard DMD theory is given by* $\mathbf{g}_{DMD}(\mathbf{x}) = \mathbf{x}$, *and it produces the error* E_{DMD}. *The observables* $\mathbf{g}_1(\mathbf{x})$ *and* $\mathbf{g}_2(\mathbf{x})$ *produce errors* E_1 *and* E_2, *respectively. Note that the judicious choice of observables* $\mathbf{g}_1(\mathbf{x})$ *produces a Koopman approximation to the dynamics that is four orders of magnitude better than DMD. The top panel depicts the error* (10.10), *while the bottom panel displays the error on a logarithmic scale.*

In particular, we would have the following.

ALGORITHM **10.6. Koopman data of observables.**

```
Y1=[X1; abs(X1).^2];   %% Data observables g1
Y2=[X2; abs(X2).^2];   %% Shifted Data
```

This can be used as the input data for the Koopman decomposition to produce a second Koopman approximation.

Figure 10.1(c) and (d) shows the Koopman reconstruction of the simulated data for the observables (10.9). The observable $\mathbf{g}_1(\mathbf{x})$ provides an exceptional approximation to the evolution, while $\mathbf{g}_2(\mathbf{x})$ is quite poor. Indeed, the errors of the three approximations considered are shown in Figure 10.3, where the following error metric is used:

$$E_j(t_k) = \|\mathbf{x}(t_k) - \tilde{\mathbf{x}}(t_k)\|, \quad k = 1, 2, \ldots, m, \tag{10.10}$$

where $\mathbf{x}$ is the full simulation and $\tilde{\mathbf{x}}$ is the DMD or Koopman approximation. Here j corresponds to the DMD observable or one of the two observables given in (10.9). With the choice of observable $\mathbf{g}_1(\mathbf{x})$, which was judiciously chosen to match the nonlinearity of the NLS, the Koopman approximation of the dynamics is four orders of

Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

magnitude better than a DMD approximation. A poor choice of observables, given by $\mathbf{g}_2(\mathbf{x})$, gives the worst performance of all. Note also the difference in the Koopman eigenfunctions and eigenvalues, as shown in Figure 10.2. In particular, note that the great choice of observables $\mathbf{g}_1(\mathbf{x})$ aligns the eigenvalues along the imaginary axis, as is expected from the dynamics. It further suggests that much better long-time predictions can be achieved with the Koopman decomposition using $\mathbf{g}_1(\mathbf{x})$.

In summary, the example of NLS shows that the Koopman decomposition can greatly improve the reconstruction and future-state prediction of data. But a suitable observable is critical in making this happen. For NLS, a good observable can be guessed by the form of the equation. However, such a luxury is rarely afforded in systems where the dynamics are not known.

10.3 ▪ Extended and kernel DMD

Williams et al. [301, 302] have recently capitalized on the ideas of machine learning by implementing the kernel method and extended observables (10.4) within the DMD architecture. In their formulation, the matrices of observables are constructed as follows:

$$\mathbf{Y} = \begin{bmatrix} | & | & & | \\ \mathbf{g}(\mathbf{x}_1) & \mathbf{g}(\mathbf{x}_2) & \cdots & \mathbf{g}(\mathbf{x}_m) \\ | & | & & | \end{bmatrix}, \tag{10.11a}$$

$$\mathbf{Y}' = \begin{bmatrix} | & | & & | \\ \mathbf{g}(\mathbf{x}'_1) & \mathbf{g}(\mathbf{x}'_2) & \cdots & \mathbf{g}(\mathbf{x}'_m) \\ | & | & & | \end{bmatrix}, \tag{10.11b}$$

where the DMD algorithm produces the matrix decomposition

$$\mathbf{A}_\mathbf{Y} = \mathbf{Y}'\mathbf{Y}^\dagger \tag{10.12}$$

along with the low-rank counterpart $\tilde{\mathbf{A}}_\mathbf{Y}$. This is the standard treatment of the Koopman operator reconstruction except now the operator $\mathbf{A}_\mathbf{Y}$ can be computationally intractable to solve due to the large number of observables specified by the feature space of the machine learning methodology. Indeed, it is not clear that one can compute the low-rank $\tilde{\mathbf{A}}_\mathbf{Y}$ due to the size of matrix $\mathbf{A}_\mathbf{Y}$ and the computational complexity of the SVD.

The extended DMD method and the kernel DMD method are both mathematical techniques that find numerically efficient ways to approximate $\tilde{\mathbf{A}}_\mathbf{Y}$, which is a finite-dimensional approximation of the Koopman operator. In the former method, the number of snapshots m is assumed to be much larger than the number of observables n, so that $m \gg n$. In the latter technique, which is generally of greater interest, the number of snapshots is much less than the number of observables, so that $n \gg m$. In both cases, a significant reduction in computational expense can be achieved by generating the Koopman operator using the kernel trick from computer science. Thus, both the extended DMD and the kernel DMD are similar to the *method of snapshots* [262], which computes POD modes by performing an SVD on a matrix determined by the number of snapshots.

10.3.1 ▪ Extended DMD

The extended DMD method was developed to reduce the cost of evaluating the Koopman operator when $m \gg n$. Recall that the goal in constructing the Koopman op-

Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

erator is to generate an intrinsic characterization of the dynamical system measured by computing the triplet-pair of Koopman eigenvalues λ_k, Koopman eigenfunctions $\varphi_k(\mathbf{x})$, and Koopman modes $\mathbf{v}_k$.

Given that the data matrices $\mathbf{Y}, \mathbf{Y}' \in \mathbb{C}^{n\times m}$ are short and fat with $m \gg n$, one can consider the efficient computation for the Koopman operator

$$\begin{aligned} \mathbf{A}_Y &= \mathbf{Y}'\mathbf{Y}^\dagger \\ &= \mathbf{Y}'\mathbf{I}\mathbf{Y}^\dagger \\ &= \mathbf{Y}'\left(\mathbf{Y}^*\mathbf{Y}^{*\dagger}\right)\mathbf{Y}^\dagger \\ &= \left(\mathbf{Y}'\mathbf{Y}^*\right)\left(\mathbf{Y}\mathbf{Y}^*\right)^\dagger \\ &= \mathbf{A}_1\mathbf{A}_2^\dagger, \end{aligned} \tag{10.13}$$

where $\mathbf{I}$ is the identity in the second line and

$$\mathbf{A}_1 = \mathbf{Y}'\mathbf{Y}^*, \tag{10.14a}$$
$$\mathbf{A}_2 = \mathbf{Y}\mathbf{Y}^*. \tag{10.14b}$$

Note that in this formulation, both $\mathbf{A}_1$ and $\mathbf{A}_2 \in \mathbb{C}^{n\times n}$. Thus, these are much smaller matrices to compute than the original $n \times m$ formulation. This is highly advantageous not only for computational speed but also for memory storage when m is high dimensional. Indeed, the computational cost in the extended DMD framework is determined by the number n of features selected instead of the much larger number of snapshots taken [301]. There is another significant advantage. Unlike the formulation (10.12), which requires a computationally expensive pseudoinverse computation of $\mathbf{Y} \in \mathbb{C}^{n\times m}$, the extended DMD formulation (10.13) only requires a pseudoinverse of the matrix $\mathbf{A}_2 \in \mathbb{C}^{n\times n}$. The architecture of the extended DMD method is illustrated in Figure 10.4. This figure shows that one can trade an expensive matrix computation involving the pseudoinverse $\mathbf{Y}^\dagger$ for two matrix multiplications. These reduce the matrices to much smaller square matrices, only one of which needs to be inverted.

10.3.2 ▪ Kernel DMD

The kernel DMD method is ideally suited for the physically relevant case when $n \gg m$. Specifically, when considering a large and diverse set of feature space variables, then the n can grow quickly to be extremely high-dimensional. For instance, Williams et al. [302] estimate that using the space of multivariate polynomials up to degree 20 on a state-space of $\mathbb{R}^{256}$ yields $n \sim \mathcal{O}(10^{30})$. Thus, a 20th-degree polynomial for a feature space would yield a computationally intractable problem because the SVD scales like $\mathcal{O}(n^3)$, or $\mathcal{O}(10^{90})$. This is a classic example of the *curse of dimensionality*. In what follows, we will use a kernel trick to enact the same computational reduction as in extended DMD. In this case, the goal is to use the kernel method to reduce the computational complexity so as to be determined by the number m of snapshots taken, where $m \ll n$.

Given that the data matrices $\mathbf{Y}, \mathbf{Y}' \in \mathbb{C}^{n\times m}$ are now tall and skinny with $n \gg m$, one can consider the efficient computation for the Koopman operator by projecting to the principal component space captured by the SVD of the data matrix $\mathbf{Y}$:

$$\mathbf{Y} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^*. \tag{10.15}$$

Thus, when considering the Koopman eigenvalue problem

$$\mathbf{K}\mathbf{y}_k = \lambda_k\mathbf{y}_k, \tag{10.16}$$

Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

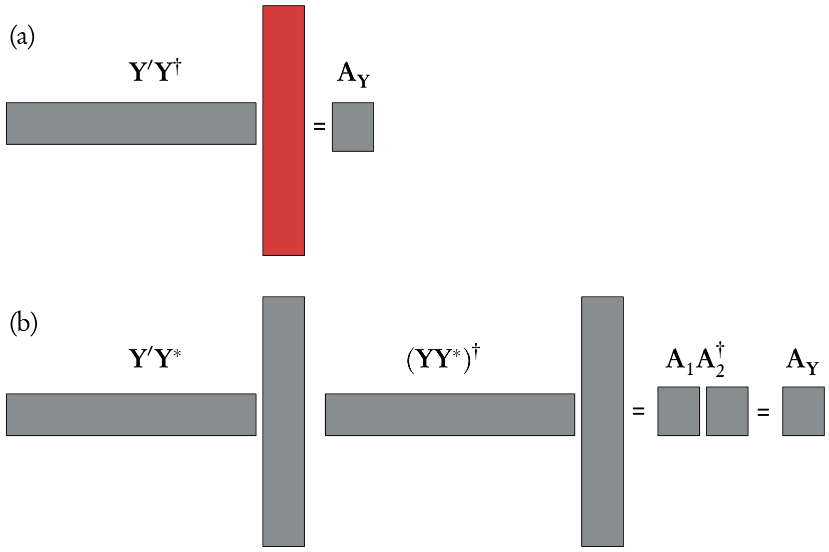

(a)
$\mathbf{Y}'\mathbf{Y}^\dagger$
$\mathbf{A_Y}$
=
(b)
$\mathbf{Y}'\mathbf{Y}^*$
$(\mathbf{YY}^*)^\dagger$
$\mathbf{A}_1\mathbf{A}_2^\dagger$
$\mathbf{A_Y}$
=
=

Figure 10.4. *Extended DMD architecture where the number m of snapshots collected is much greater than the number of observables n, and expensive matrix computations are denoted in red.* (a) *The Koopman decomposition would require computing the inner product $\mathbf{Y}'\mathbf{Y}^\dagger$. Although this gives the desired matrix $\mathbf{A_Y} \in \mathbb{C}^{n\times n}$, it requires the expensive computation of the pseudoinverse $\mathbf{Y}^\dagger \in \mathbb{C}^{m\times n}$.* (b) *Extended DMD circumvents the expensive pseudoinverse computation by instead computing two matrices $\mathbf{A}_1, \mathbf{A}_2 \in \mathbb{C}^{n\times n}$. Although the pseudoinverse of $\mathbf{A}_2$ must be computed, it is a much smaller matrix. Thus, the computational cost of the extended DMD is determined by the number n of features selected instead of the much larger number m of snapshots taken.*

one can consider that the eigenvector itself can be constructed by the expansion

$$\mathbf{y}_k = \mathbf{U}\hat{\mathbf{y}}_k\,. \tag{10.17}$$

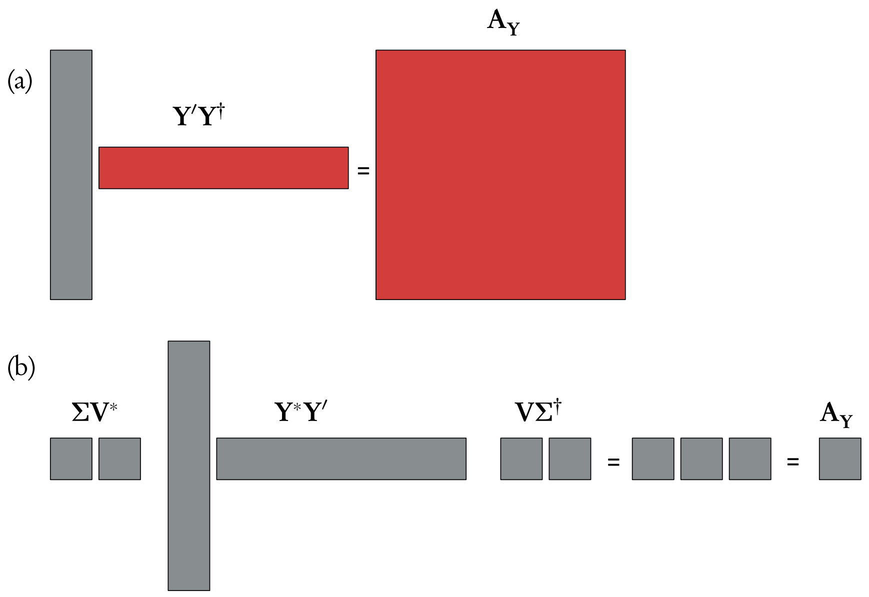
Inserting this projection into the eigenvalue problem results in

$$\begin{aligned}
\lambda_k \mathbf{y}_k &= \mathbf{K}\mathbf{y}_k,\\
\lambda_k \mathbf{U}\hat{\mathbf{y}}_k &= \mathbf{KU}\hat{\mathbf{y}}_k\\
&= \mathbf{Y}'\mathbf{Y}^\dagger\mathbf{U}\hat{\mathbf{y}}_k\\
&= \mathbf{Y}'\mathbf{Y}^\dagger\left(\mathbf{YV}\mathbf{\Sigma}^\dagger\right)\hat{\mathbf{y}}_k\\
&= \mathbf{Y}'\left(\mathbf{V}\mathbf{\Sigma}^\dagger\right)\hat{\mathbf{y}}_k\\
&= \mathbf{IY}'\left(\mathbf{V}\mathbf{\Sigma}^\dagger\right)\hat{\mathbf{y}}_k\\
&= (\mathbf{Y}^*)^\dagger\,\mathbf{Y}^*\mathbf{Y}'\left(\mathbf{V}\mathbf{\Sigma}^\dagger\right)\hat{\mathbf{y}}_k\\
&= (\mathbf{Y}^*)^\dagger\left(\mathbf{Y}^*\mathbf{Y}'\right)\left(\mathbf{V}\mathbf{\Sigma}^\dagger\right)\hat{\mathbf{y}}_k\\
&= \mathbf{U}(\mathbf{\Sigma V}^*)\left(\mathbf{Y}^*\mathbf{Y}'\right)\left(\mathbf{V}\mathbf{\Sigma}^\dagger\right)\hat{\mathbf{y}}_k\\
&= \mathbf{UK}\hat{\mathbf{y}}_k,
\end{aligned} \tag{10.18}$$

where $\mathbf{I}$ is the identity in the sixth line and the Koopman operator is now evaluated by the expression

$$\mathbf{A_Y} = (\mathbf{\Sigma V}^*)\left(\mathbf{Y}^*\mathbf{Y}'\right)\left(\mathbf{V}\mathbf{\Sigma}^\dagger\right). \tag{10.19}$$

Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy



(a)
$\mathbf{Y}'\mathbf{Y}^\dagger$
=
$\mathbf{A_Y}$
(b)
$\Sigma\mathbf{V}^*$
$\mathbf{Y}^*\mathbf{Y}'$
$\mathbf{V}\Sigma^\dagger$
=
=
$\mathbf{A_Y}$

Figure 10.5. *Kernel DMD architecture where the number m of snapshots collected is much smaller than the number of observables n, and expensive matrix computations are denoted in red.* (a) *The Koopman decomposition would require computing the inner product* $\mathbf{Y}'\mathbf{Y}^\dagger$*, thus requiring the expensive computation of the pseudoinverse* $\mathbf{Y}^\dagger \in \mathbb{C}^{m\times n}$ *and producing the unwieldy matrix* $\mathbf{Y}^\dagger \in \mathbb{C}^{n\times n}$. (b) *Extended DMD circumvents these expensive computations by instead computing three matrices, which all produce* $\mathbb{C}^{m\times m}$ *matrices. Only a single, trivial pseudoinverse must now be computed, but on the diagonal matrix* Σ*. Thus, the computational cost of the kernel DMD is determined by the number m of snapshots rather than the much larger number n of features selected.*

Like the extended DMD algorithm, the advantage of this formulation is that $(\Sigma\mathbf{V}^*) \in \mathbb{C}^{m\times m}$, $(\mathbf{Y}^*\mathbf{Y}') \in \mathbb{C}^{m\times m}$, and $\left(\mathbf{V}\Sigma^\dagger\right) \in \mathbb{C}^{m\times m}$. Thus, as suggested, the computation of the Koopman operator is determined by the number of snapshots taken rather than the number of features. Of course, an SVD operation is required to evaluate (10.19). But the required Σ and $\mathbf{V}$ matrices can be computed by considering the eigenvalue decomposition of the $m \times m$ matrix $\mathbf{Y}^*\mathbf{Y}\mathbf{V} = \Sigma^2\mathbf{V}$. Thus, once again, all computations are constrained by the number of snapshots m versus the number of features n, where $n \gg m$. Figure 10.5 illustrates the kernel DMD computational infrastructure. The kernel DMD circumvents not only the expensive computation of the pseudoinverse $\mathbf{Y}^\dagger$ but the construction of the large matrix $\mathbf{A_Y}$ that results from the outer product. Instead, the computation is broken down into three matrix multiplications that are projected to the feature space embedding $\mathbf{U}$. Thus, low-dimensional matrices are produced, making the evaluation of $\mathbf{A_Y}$ tractable.

10.3.3 ▪ The kernel trick for observables

Both the extended DMD and the kernel DMD aim to reduce the computational cost of evaluating the approximation of the Koopman operator, denoted here by $\mathbf{A_Y}$. In both methods, it is desirable to take a large number of observables and snapshots and produce the Koopman approximation $\mathbf{A_Y} \in \mathbb{C}^{p\times p}$, where $p = \min(m,n)$. Ultimately, the kernel DMD method is the most relevant in practice, as the number of observables

Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

(features) can rapidly grow to make n extremely high dimensional. It is conjectured that taking a large enough, and rich enough, set of observables allows one to produce an accurate representation of the Koopman operator.

Kernel methods are a class of algorithms used for generic pattern analysis tasks such as the clustering and classification of high-dimensional data. Indeed, they are now commonly used in machine learning tasks, with SVM being one of the most common and successful implementations. For many data-intensive algorithms that perform pattern analysis tasks, the data is typically represented in its raw form before being explicitly transformed into feature vector representations via a user-specified feature map. For instance, a PCA determines the feature space via SVD. However, computing the SVD is extremely expensive, perhaps prohibitively so, when the data under consideration is high dimensional. In contrast, kernel methods require only a user-specified kernel, i.e., a similarity function over pairs of data points in raw representation. So just like the extended and kernel DMD architecture, the goal of the kernel technique is to reduce the computational complexity of a given problem by intelligent choices of the features or observables. In what follows, features and Koopman observables will be used interchangeably.

In the context of the Koopman operator, the kernel trick [53, 198, 271, 21] will define a function $f(\mathbf{x},\mathbf{x}')$ that can be related to the observables $g_j(\mathbf{x})$ used to construct $\mathbf{Y}$ and $\mathbf{Y}'$. Consider the simple example of a polynomial kernel

$$f(\mathbf{x},\mathbf{x}') = \left(1+\mathbf{x}^*\mathbf{x}'\right)^2, \tag{10.20}$$

where $\mathbf{x}$ and $\mathbf{x}'$ are data points in $\mathbb{R}^2$. When expanded out, the kernel function

$$\begin{aligned} f(\mathbf{x},\mathbf{x}') &= \left(1+x_1x_1'+x_2x_2'\right)^2 \\ &= \left(1+2x_1x_1'+2x_2x_2'+2x_1x_2x_1'x_2'+x_1^2x_1'^2+x_2^2x_2'^2\right) \\ &= \mathbf{Y}^*(\mathbf{x}')\mathbf{Y}(\mathbf{x}), \end{aligned} \tag{10.21}$$

provided

$$\mathbf{Y}(\mathbf{x}) = \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \sqrt{2}x_1x_2 \\ x_1{}^2 \\ x_2{}^2 \end{bmatrix}. \tag{10.22}$$

Note that for this case, both the Koopman observables in (10.22) and the kernel function (10.20) are equivalent representations that are paired through the expansion (10.21). However, the kernel trick is basically centered around the realization that (10.20) is a significantly more efficient representation of the polynomial variables that emerge from expanding as in (10.21). More precisely, instead of defining our typical Koopman observables $g_i(\mathbf{x})$ used to construct (10.22), we instead define the kernel function (10.20), as it provides an ideal representation of the feature space for the various kernels selected and provides an implicit computation of the inner products required for the Koopman operator. Indeed, the computation of the observables reduces to a simple inner product between vector pairs, providing an efficient algorithm for producing the feature space (Koopman observables) representation.

To illustrate the computational savings of the kernel trick more clearly, consider a polynomial kernel of degree p acting on data vectors $\mathbf{x}$ and $\mathbf{x}'$ in $\mathbb{R}^n$. In this case, the

Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

kernel method simply defines

$$f(\mathbf{x},\mathbf{x}') = \left(1+\mathbf{x}^*\mathbf{x}'\right)^p, \tag{10.23}$$

which requires the computation of the inner product $\alpha = \mathbf{x}^*\mathbf{x}'$. This requires $\mathcal{O}(n)$ operations and produces the constant α. It is then trivial to finish computing $f(\mathbf{x},\mathbf{x}') = (1+\alpha)^p$. Thus, overall, the computational cost for this pth-degree polynomial kernel is $\mathcal{O}(n)$. In contrast, to construct the equivalent set of observables using $\mathbf{Y}$ would require a vector of observables of length $\mathcal{O}(n^2)$, which takes the form

$$\mathbf{Y}(\mathbf{x}) = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \\ x_1^2 \\ \vdots \\ x_n^2 \\ {x_1}^p \\ \vdots \\ {x_n}^p \\ x_1x_2 \\ x_1x_3 \\ \vdots \\ x_1x_p \\ x_2x_3 \\ \vdots \\ x_px_p \end{bmatrix}. \tag{10.24}$$

Once it was formed, one would still be required to compute $\mathbf{Y}^*(\mathbf{x}')\mathbf{Y}(\mathbf{x})$. This is a significantly larger computation than the kernel form (10.23). Indeed, for a large set of observables, the kernel trick enables the actual computation of the inner products, while the form (10.24) can remain intractable.

The choice of kernel is important. Although there is an infinite set of possibilities, a few common choices are typically used in machine learning. The following three kernels are the workhorses of SVM-based data methods:

$$\begin{aligned} &\text{polynomial kernel (degree } p) & f(\mathbf{x},\mathbf{x}') &= \left(a+\mathbf{x}^*\mathbf{x}'\right)^p, & (10.25a)\\ &\text{radial basis functions} & f(\mathbf{x},\mathbf{x}') &= \exp\left(-a|\mathbf{x}-\mathbf{x}'|^2\right), & (10.25b)\\ &\text{sigmoid kernel} & f(\mathbf{x},\mathbf{x}') &= \tanh\left(\mathbf{x}^*\mathbf{x}'+a\right). & (10.25c)\end{aligned}$$

The advantages of the kernel trick are quite clear. For the polynomial kernel, for instance, a 20th-degree polynomial ($p = 20$) using (10.25a) is trivial. In fact, the kernel function does not compute all the inner products directly. In contrast, using our standard Koopman observables $g(\mathbf{x}_j)$ would require one to explicitly write out all the terms generated from a 20th-degree polynomial on an n-dimensional data set, which is a monumental computational task in comparison.

In practice, then, instead of defining the observables $\mathbf{Y}^*\mathbf{Y}'$ in (10.19) with (10.11), we will define the observables $\mathbf{Y}$ using the kernel $f(\mathbf{x},\mathbf{x}')$ for producing inner products

Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

associated with the feature space. Specifically, we will consider the observable matrix elements as defined by

$$\mathbf{Y}^*\mathbf{Y}'(j,k)=f(\mathbf{x}_j,\mathbf{x}'_k), \tag{10.26}$$

where (j,k) denotes the jth row and kth column of the correlation matrix, and $\mathbf{x}_j$ and $\mathbf{x}'_k$ are the jth and kth columns of data. The kernel DMD formulation (10.19) also requires that we compute the matrices $\mathbf{V}$ and $\mathbf{\Sigma}$. Recall the definition (10.15), so that

$$\mathbf{Y}^*\mathbf{Y}\mathbf{V}=\Sigma^2\mathbf{V}. \tag{10.27}$$

As before, we can compute the matrix elements of $\mathbf{Y}^*\mathbf{Y}$ using the kernel method, so that

$$\mathbf{Y}^*\mathbf{Y}(j,k)=f(\mathbf{x}_j,\mathbf{x}_k) \tag{10.28}$$

where (j,k) denotes the jth row and kth column of the correlation matrix, and $\mathbf{x}_j$ and $\mathbf{x}_k$ are the jth and kth columns of data. Thus, all of the key inner products are produced by using the efficient kernel method and projecting directly to the feature space. All that remains is to choose a kernel. Once determined, the inner products (10.26) and (10.28) on the kernel function allow us to evaluate the matrix $\mathbf{A_Y}$ via (10.19). As a final note, if the linear kernel function $f(\mathbf{x},\mathbf{x}) = \mathbf{x}^*\mathbf{y}$ is chosen, the kernel DMD reduces to the standard DMD algorithm.

10.4 ▪ Implementing extended and kernel DMD

To demonstrate how the extended and kernel DMD methods are implemented in practice, we once again return to the example code for the NLS Algorithm **10.3**. This toy model will exemplify the merits of and potential challenges with the methods developed in the preceding section.

For the extended DMD technique, it is assumed that the number of snapshots taken is typically far larger than the number of state variables, or observables, recorded in the data matrix $\mathbf{Y}$. In the example Algorithm **10.3**, only 21 snapshots of data were collected relative to the 512 Fourier modes (state variables) used to represent the solution. Instead, we could take a large number of snapshots by simply redefining our time variable with the following code.

***Algorithm* 10.7. Extended DMD sampling.**

```
slices=2000;
t=linspace(0,2*pi,slices+1); dt=t(2)-t(1);
```

This gives 2000 snapshots relative to the 512 state measurements, thus suggesting the use of the extended DMD algorithm outlined in Figure 10.4. The following code produces the matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ that are used in efficiently computing $\mathbf{A_Y}$ in (10.13).

***Algorithm* 10.8. Extended DMD Koopman approximation.**

```
X=usol.'; X1=(X(:,1:end-1)); X2=(X(:,2:end));

tic
Ay1=X2*pinv(X1);
toc
```

Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

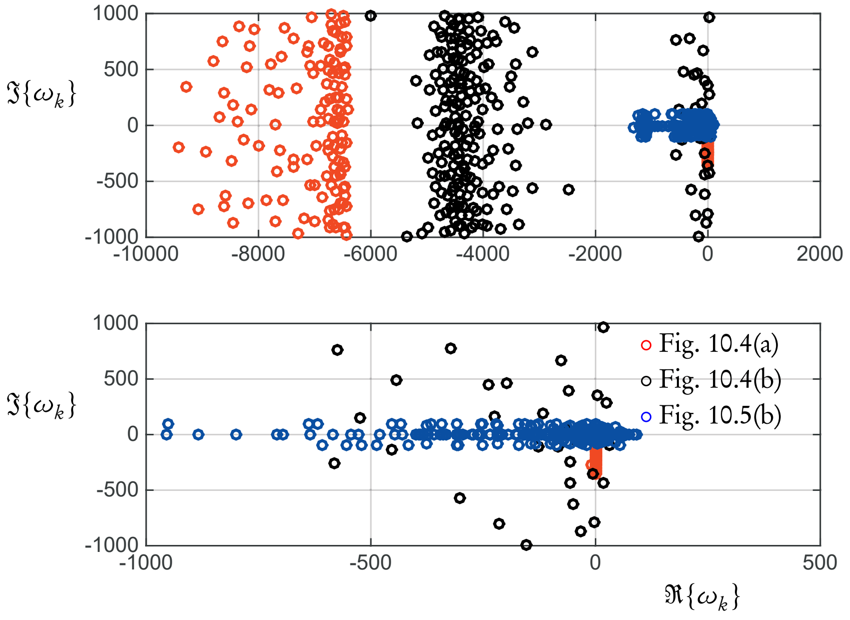


Figure 10.6. *Comparison of eigenvalue spectrum for the extended (black circles) and kernel (blue circles) DMD methods. The full computation of the spectrum is also shown (red circles). The logarithms of the eigenvalues of* $\mathbf{A_Y}$*, divided by* Δt*, are plotted in the panels.*

```
tic
A1=X2*X1.';
A2=X1*X1.';
Ay2=A1*pinv(A2);
toc
```

Note that the tic-toc commands in MATLAB are used to estimate the computational time for the two procedures. Extended DMD (Figure 10.4(b)) computes the spectra approximately an order of magnitude faster than the standard method (Figure 10.4(a)) for the matrices used. The computational savings scales with the size of the matrices and the disparity between the number of snapshots and the number of state variables recorded, i.e., $m \gg n$.

The extended DMD method can be compared to the full spectral computation. Figure 10.6 shows both the full computation of the eigenvalues $\mathbf{A_Y}$ (red circles) and the extended DMD approximation to the eigenvalues (black circles).

The kernel DMD is also considered using the kernel method for evaluating the observables. For this case, we once again consider a more limited snapshot matrix given by Algorithm 10.9.

***Algorithm* 10.9. Kernel DMD sampling.**

```
slices=200;
t=linspace(0,2*pi,slices+1); dt=t(2)-t(1);
```

In addition, a radial basis function kernel is assumed for the kernel function

$$f(\mathbf{x},\mathbf{x}') = \exp\left(-|\mathbf{x}-\mathbf{x}'|^2\right). \tag{10.29}$$

The absolute value is important for the case of the NLS equation considered due to the nonlinear evolution of the phase. The following code computes the observables for the DMD algorithm using the radial basis function kernel.

***Algorithm* 10.10. Kernel DMD approximation.**

```
X=usol.'; X1=(X(:,1:end-1)); X2=(X(:,2:end));
[n,m]=size(X);

for j=1:m-1
 for jj=1:m-1
  YtYp(j,jj)=exp(-abs((X1(:,jj)-X2(:,j)).'*(X1(:,jj)-X2(:,j))));
   YsY(j,jj)=exp(-abs((X1(:,jj)-X2(:,j)).'*(X1(:,jj)-X2(:,j))));
 end
end

[V,Sig]=eig(YsY);
Ay=(Sig*V')*(YtYp)*(V/Sig);

[W,D]=eig(Ay);
```

The eigenvalues of the matrix approximating $\mathbf{A_Y}$ (see Figure 10.5) are shown in Figure 10.6 (blue dots). Comparison can be made to the extended DMD approximation as well.

One of the most challenging aspects of Koopman theory, extended DMD, and kernel DMD is the selection of observables, or functions of the state-space variables. This remains an open question in the Koopman architecture: what are the most accurate $g_j(\mathbf{x})$ to be used for mapping a finite-dimensional nonlinear dynamical system to the Koopman representation of a high-dimensional linear operator? To highlight the significance of this question, consider that a number of kernels could be used to approximate the dynamics of the NLS. So, instead of simply considering the radial basis functions, we can also consider three additional observables:

$$f(\mathbf{x},\mathbf{x}') = \left(1+\mathbf{x}^*\mathbf{x}'\right)^{20}, \tag{10.30a}$$

$$f(\mathbf{x},\mathbf{x}') = \left(a+|\mathbf{x}^*||\mathbf{x}'|\right)^{20}, \tag{10.30b}$$

$$f(\mathbf{x},\mathbf{x}') = \exp(-\mathbf{x}^*\mathbf{x}'). \tag{10.30c}$$

The first is the standard polynomial kernel of 20th degree. The second instead takes the absolute value of the variable in a polynomial variable to remove the phase, and the third is a Gaussian kernel that uses the same inner product as the polynomial kernel.

Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

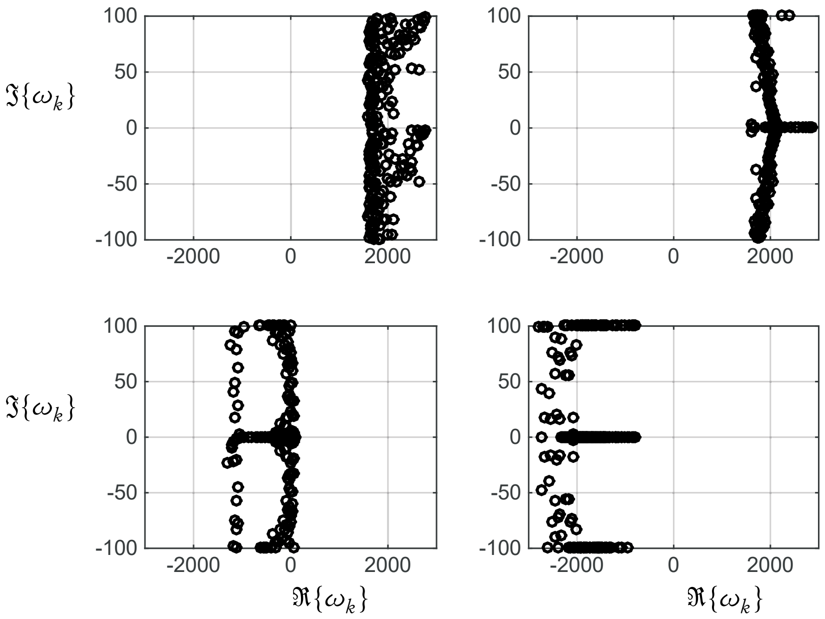


Figure 10.7. *Comparison of different spectra generated from the kernel method. Clockwise from top right are the kernels* (10.30(a)), (10.30(b)), (10.29), *and* (10.30(c)). *Note the tremendous variability in the spectrum from the different kernels. The logarithms of the eigenvalues of* $\mathbf{A}_Y$, *divided by* Δt, *are plotted in the panels.*

The code to produce these three kernels, as well as the original radial basis function, is given in Algorithm **10.11**.

***ALGORITHM* 10.11. Different kernel approximations.**

```
YtYp1(j,jj)=(1+((X1(:,jj)).')*(X2(:,j)))^p;
 YsY1(j,jj)=(1+((X1(:,jj)).')*(X1(:,j)))^p;

YtYp2(j,jj)=(1+(abs(X1(:,jj)).')*abs(X2(:,j)))^p;
 YsY2(j,jj)=(1+(abs(X1(:,jj)).')*abs(X1(:,j)))^p;

YtYp3(j,jj)=exp(-abs((X1(:,jj)-X2(:,j)).'*(X1(:,jj)-X2(:,j))))
    ;
 YsY3(j,jj)=exp(-abs((X1(:,jj)-X2(:,j)).'*(X1(:,jj)-X2(:,j))))
    ;

YtYp4(j,jj)=exp(-abs((X1(:,jj).')*(X2(:,j))));
 YsY4(j,jj)=exp(-abs((X1(:,jj).')*(X2(:,j))));
```

These three new kernels are compared to each other and the radial basis function already used in Figure 10.6. Figure 10.7 shows the spectra generated by these four kernels. Note the tremendous variability of the results based on the choice of kernel. Ultimately, much like the NLS example considered previously, the choice of kernel must be carefully selected for either the extended or kernel DMD to give anything reasonable. Cross-validation techniques could potentially be used to select a suitable kernel for applications of interest. It could also ensure that overfitting of the data does not occur.

Downloaded 11/11/23 to 137.189.241.44 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy