A. Pseudocodes

Here we present pseudocodes for the dictionary learning method and batch learning methods described in the main body of the paper.

**Algorithm 1** Sparse ALD dictionary learning with Cholesky updates

The operation count for each step is included on the right.

**Inputs:** data matrix $X$, kernel $k$, sparsification tolerance $\nu$

**Output:** the sparse dictionary $\tilde{X}$

Optional: randomly permute the columns of $X$ for $t = 1 \rightarrow m$

for $t = 1 \rightarrow m$ do

Select new sample $x_t$

Compute $\tilde{k}_{t-1}$ with (3.11)

Compute $\pi_t$ with backsubstitution (3.10)

Compute $\delta_t$ using (3.9)

if $\delta_t \leq \nu$ (almost linearly dependent) then

Maintain the dictionary: $D_t = D_{t-1}$

else if $\delta_t > \nu$ (not almost linearly dependent) then

Update the dictionary: $D_t = D_{t-1} \cup \{x_t\}$

Update the Cholesky factor $C_t$ using (3.13)

end if

end for

**Algorithm 2** Learning the model and analysing the linear component

**Inputs:** data matrices $X$ and $Y$, kernel $k$, dictionary tolerance $\nu$

**Outputs:** model $f$, constant $c$, linear component $L$, nonlinear component $N$, eigenvectors $\psi$, and eigenvalues $\lambda$

Build the dictionary $\tilde{X}$ according to algorithm 1

Solve $\arg\min_{\tilde{W}} \|Y - \tilde{W} k(\tilde{X}, X)\|_F$ (for example, $\hat{W} = Y k(\tilde{X}, X)^\dagger$)

Define $S$ according to (4.7)

Form the model as $f(x) = \hat{W} k(\tilde{X}, x)$

Form $c$, $L$, and $N$ according to section 4(a) and a choice of base state

Compute the eigendecomposition of $L$ according to lemma 1

Form the eigenvectors $\psi$ and eigenvalues $\lambda$ according to (4.8)

B. Glossary of terms

Table 1 provides the nomenclature used throughout the paper. We have attempted to maintain consistency with DMD [15, 24], SINDy [4], and KRLS [7] where possible, although several changes were made to unify the notation. Importantly, the features are denoted by $\phi$ in this work, whereas
they are denoted by $\theta$ in SINDy. Similarly, the kernel weights are denoted by $W$ in this work, whereas they are denoted by $\Theta$ in KRLS.

Table 1: A summary of terms used in the paper.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>number</td>
<td>State dimension</td>
</tr>
<tr>
<td>$m$</td>
<td>number</td>
<td>Total number of samples</td>
</tr>
<tr>
<td>$N$</td>
<td>number</td>
<td>Dimension of nonlinear feature space</td>
</tr>
<tr>
<td>$x$</td>
<td>$n$-vector</td>
<td>State vector</td>
</tr>
<tr>
<td>$y$</td>
<td>$n$-vector</td>
<td>Model output (e.g. $y = \dot{x}$ or $y_k = x_{k+1}$)</td>
</tr>
<tr>
<td>$X$</td>
<td>$n \times m$ matrix</td>
<td>Right data matrix</td>
</tr>
<tr>
<td>$Y$</td>
<td>$n \times m$ matrix</td>
<td>Left data matrix</td>
</tr>
<tr>
<td>$F(x)$</td>
<td>function</td>
<td>Underlying system</td>
</tr>
<tr>
<td>$f(x)$</td>
<td>function</td>
<td>Approximate learned model</td>
</tr>
<tr>
<td>$c$</td>
<td>$n$-vector</td>
<td>Constant shift in model</td>
</tr>
<tr>
<td>$L$</td>
<td>$n \times n$ matrix</td>
<td>Linear component of true operator</td>
</tr>
<tr>
<td>$N$</td>
<td>function</td>
<td>Purely nonlinear component of operator</td>
</tr>
<tr>
<td>$A$</td>
<td>$n \times n$ matrix</td>
<td>DMD best-fit operator</td>
</tr>
<tr>
<td>$k(\cdot, \cdot)$</td>
<td>function</td>
<td>Kernel function</td>
</tr>
<tr>
<td>$\phi(\cdot)$</td>
<td>function</td>
<td>Nonlinear (implicit) feature space</td>
</tr>
<tr>
<td>$t$</td>
<td>continuous variable</td>
<td>time</td>
</tr>
<tr>
<td>$t$</td>
<td>discrete index</td>
<td>Snapshot number</td>
</tr>
<tr>
<td>$\nu$</td>
<td>number</td>
<td>Dictionary sparsification parameter</td>
</tr>
<tr>
<td>$\tilde{\cdot}$</td>
<td>tilda</td>
<td>Indicates quantity is connected to the dictionary</td>
</tr>
<tr>
<td>$\tilde{m}_t$</td>
<td>number</td>
<td>Number of samples in dictionary at time $t$</td>
</tr>
<tr>
<td>$\tilde{K}_t$</td>
<td>$\tilde{m}_t \times \tilde{m}_t$ matrix</td>
<td>Kernel matrix of dictionary elements at time $t$</td>
</tr>
<tr>
<td>$\tilde{C}_t$</td>
<td>$\tilde{m}_t \times \tilde{m}_t$ matrix</td>
<td>Cholesky decomposition of $\tilde{K}_t$</td>
</tr>
<tr>
<td>$\tilde{m}$</td>
<td>number</td>
<td>Final number of samples in the dictionary</td>
</tr>
<tr>
<td>$\Pi_t$</td>
<td>$\tilde{m}_t \times t$ matrix</td>
<td>Matrix that approximately maps samples before time $t$ onto the dictionary</td>
</tr>
<tr>
<td>$\hat{\cdot}$</td>
<td>hat</td>
<td>Indicates quantity is projected onto POD subspace</td>
</tr>
<tr>
<td>$\pi$</td>
<td>$n$-vector</td>
<td>Base state (e.g. statistical mean or equilibrium solution)</td>
</tr>
</tbody>
</table>

C. Comparison to related data-driven methods

In this section we provide further details of related data-driven methods. The comparison is summarised in figure 10.
(i) Sparse identification of nonlinear dynamics

The SINDy algorithm \([4]\) was developed based on the observation that many complex dynamical systems may be expressed as systems of differential equations with only a few terms, so that they are \textit{sparse} in the feature space \(\phi(x)\). Thus, it is possible to solve for an expansion of the dynamics in (2.3) with only a few nonzero entries in \(\Xi\), corresponding to the active terms in the \(\phi(x)\) that are present in the dynamics. Solving for the sparse vector of coefficients, and therefore the dynamical system, is achieved through the following optimization

\[
\arg\min_{\Xi} \|Y - \Xi \phi(X)\|_F + \lambda \|\Xi\|_0. \tag{A 1}
\]

The \(\|\cdot\|_0\) term is not convex, although there are several relaxations that yield accurate sparse models. The SINDy algorithm has also been extended to include partially known physics \([17]\), such as conservation laws and symmetries, dramatically improving the ability to learn accurate models with less data. It is also possible with SINDy to disambiguate the linear and nonlinear model contributions, enabling linear stability analyses, even for strongly nonlinear systems. However, the feature library \((x)\) scales poorly with the state dimension \(n\), so SINDy is typically only applied to relatively low-dimensional systems. A recent tensor extension to SINDy \([9]\) provides the ability to handle much larger libraries, which is very promising. In the present work, we use kernel representations to obtain tractable implicit models that may be queried to extract structure, such as the disambiguated linear terms.

(ii) Extended DMD

The extended DMD \([26]\) was developed to improve the approximation of the Koopman operator by augmenting the DMD vector \(x\) with nonlinear functions of the state, similar to the feature vector \((x)\) above. However, instead of modeling \(x_{k+1}\) as a function of \(\phi(x_k)\), as in SINDy, eDMD models the evolution of \(\phi(x_{k+1})\), which results in a much larger regression problem.

\[
\arg\min_{\Xi} \|\phi(Y) - \Xi \phi(X)\|_F. \tag{A 2}
\]

This approach was then kernelized \([27]\) to make the algorithm computationally tractable.

(a) Connection to exact dynamic mode decomposition

We now elucidate the connection between the present work and exact dynamic mode decomposition of \([24]\) which was introduced in section 2(a). In particular, we demonstrate that exact DMD can be viewed as a special case of the present work when there is no sparsity promotion and the kernel is linear. The linear kernel is \(k(u, v) = u^* v\), so the implicit feature space is simply \(\phi(x) = x\). As such, the full model is the linear map \(f(x) = Lx\) where \(L\) is (4.7)

\[
L = W X^*. \tag{A 3}
\]

In exact DMD there is no sparsity promotion so the dictionary used in our algorithm is full: \(\Pi = I\) and \(X = X\). Moreover, \(W\) is given by (3.3) so

\[
L = Y k(X, X)^\dagger X^*. \tag{A 4}
\]

Expanding the kernel yields

\[
L = Y (X^* X)^\dagger X^* = Y X^*(X^*)^\dagger X^* = Y X^\dagger \tag{A 5}
\]

which is identical to the linear operator \(A\) from (2.11) defined by exact DMD. In (A 5) we used the identities for the Moore–Penrose pseudoinverse \((M^* M)^\dagger = M^\dagger (M^*)^\dagger\) and \(M^\dagger (M^*)^\dagger M^* = M^\dagger\) for any matrix \(M\).

Similarly, the eigenmodes computed by exact DMD are equivalent to those defined in lemma 1 in the special case of a linear kernel without sparsity promotion.
Figure 10: A comparison of methods for model discovery, including DMD \cite{21,23}, extended/kernel DMD \cite{26,27}, SINDy \cite{4}, and the proposed LANDO framework.

### D. Online learning variant

In this section we derive the rank-one update equations used in the online regression algorithm. The derivation is equivalent to that presented in \cite{7} except we consider vector-valued outputs and do not apply the inverted kernel matrix explicitly. A summary of the procedure may be found in the pseudocode in algorithm 3.

We define

$$Y_t = \begin{bmatrix} y_1 & y_2 & \cdots & y_t \end{bmatrix}$$

In the feature space, we may express all the samples up to time $t$ as

$$\Phi_t = \Phi_t \Pi_t + \Phi^\text{res}_t$$  \hspace{1cm} (A 1)

where

$$\Pi_t = \begin{bmatrix} \pi_1 & \pi_2 & \cdots & \pi_t \end{bmatrix} \in \mathbb{R}^{m_t \times t}$$  \hspace{1cm} (A 2)

maps the $m_t$ dictionary elements into the $t$ feature vectors with small residual error $\Phi^\text{res}_t$. By causality, the lower triangular elements of $\Pi_t$ are zero. Only the online version of the algorithm (see appendix D) uses $\Pi_t$ explicitly, and only requires $\pi_t$ at time $t$. Thus, $\pi_t$ can be overwritten at each iteration to save memory.

The minimisation problem, without regularisation, at time $t$ is

$$\arg\min_{\tilde{W}_t} \left\| Y_t - \tilde{W}_t \Phi_t \Phi^\top_t \right\|_F = \arg\min_{\tilde{W}_t} \left\| Y_t - \tilde{W}_t \Phi_t \Phi^\top_t \right\|_F.$$  \hspace{1cm} (A 3)
The representation (A 1) allows us to approximate the above as
\[
\arg\min_{\tilde{W}_t} \|Y_t - \tilde{W}_t \hat{\Phi}_t^* \Phi_t \Pi_t\|^2_F = \arg\min_{\tilde{W}_t} \|Y_t - \tilde{W}_t \tilde{K}_t \Pi_t\|^2_F. \tag{A 4}
\]
The minimiser of the above is
\[
\tilde{W}_t = Y_t \left( \tilde{K}_t \Pi_t^* \right)^\dagger = Y_t \Pi_t^* \tilde{K}_t^{-1} = Y_t \Pi_t^* \left( \Pi_t \Pi_t^* \right)^{-1} \tilde{K}_t^{-1}. \tag{A 5}
\]
In the above, we have used the fact that, by construction, \( \tilde{K}_t \) has full column rank and \( \Pi_t \) has full row rank.

When a new sample is considered, it falls into two cases as outlined in section 3(a). Either the sample is almost linearly dependent on the current dictionary elements, or it is not. The updating equations are different in each case and we derive them below. In what follows, it is convenient to define
\[
P_t = (\Pi_t \Pi_t^*)^{-1} \tag{A 6}
\]
and
\[
h_t = \frac{\pi_t^* P_{t-1}}{1 + \pi_t^* \Pi_{t-1} \pi_t}. \tag{A 7}
\]
Case I: Almost linearly dependent

If the new sample is almost linearly dependent on the dictionary elements then the dictionary is not updated: \( D_t = D_{t-1} \). Since the dictionary doesn’t change, neither does the kernel matrix so \( \tilde{K}_t = \tilde{K}_{t-1} \). The update rule for \( \Pi_t \) is simply
\[
\Pi_t = \left[ \Pi_{t-1} \quad \pi_t \right]. \tag{A 8}
\]
Thus, \( \Pi_t \Pi_t^* = \Pi_{t-1} \Pi_{t-1}^* + \pi_t \pi_t^* \), which corresponds to a rank-1 update. Accordingly, the matrix inversion lemma says that the update rule for \( P_t \) is
\[
P_t = P_{t-1} - \frac{P_{t-1} \pi_t^* P_{t-1}}{1 + \pi_t^* P_{t-1} \pi_t} = P_{t-1} - P_{t-1} \pi_t h_t. \tag{A 9}
\]
We may now define the update rule for \( \tilde{W}_t \). Since
\[
Y_t \Pi_t^* = Y_{t-1} \Pi_{t-1}^* + y_t \pi_t^*, \tag{A 10}
\]
applying (A 9) to (A 5) produces
\[
\tilde{W}_t = Y_t \Pi_t^* P_t \tilde{K}_t^{-1} = (Y_{t-1} \Pi_{t-1}^* + y_t \pi_t^*) (P_{t-1} - P_{t-1} \pi_t h_t) \tilde{K}_t^{-1}. \tag{A 11}
\]
Expanding the brackets yields
\[
\tilde{W}_t = (Y_{t-1} \Pi_{t-1}^* P_{t-1} - Y_{t-1} \Pi_{t-1}^* P_{t-1} \pi_t h_t + y_t \pi_t^* P_t) \tilde{K}_t^{-1}. \tag{A 12}
\]
Since we are not adding an element to the dictionary, the kernel matrix and its inverse remain the same: \( \tilde{K}_t^{-1} = \tilde{K}_{t-1}^{-1} \). Additionally, from the regression in the previous iteration we have \( \tilde{W}_{t-1} = Y_{t-1} \Pi_{t-1}^* P_{t-1} \tilde{K}_{t-1}^{-1} \). Thus, (A 12) can be expressed as
\[
\tilde{W}_t = \tilde{W}_{t-1} + \left( y_t \pi_t^* P_t - \tilde{W}_{t-1} \tilde{K}_{t-1} \pi_t h_t \right) \tilde{K}_t^{-1}. \tag{A 13}
\]
Finally, on use of \( h_t = \pi_t^* P_t \) and (3.10), the update rule for \( \tilde{W}_t \) is
\[
\tilde{W}_t = \tilde{W}_{t-1} + \left( y_t - \tilde{W}_{t-1} \tilde{K}_{t-1} \right) h_t \tilde{K}_t^{-1}. \tag{A 14}
\]
As discussed in section 3(a), the product \( h_t \tilde{K}_t^{-1} \) should be computed with two backsubstitutions with the Cholesky factor \( C_t \).
Algorithm 3 Online learning algorithm
The operation count for each step is included on the right

**Inputs:** data matrices $X$ and $Y$, kernel $\kappa$, dictionary tolerance $\nu$

**Outputs:** model $f$, constant $c$, linear component $L$, nonlinear component $N$, eigenvectors $\psi$, and eigenvalues $\lambda$

Optional: randomly permute the columns of $X$ and $Y$ with the same permutation

for $t = 1 \rightarrow m$ do
  Select new sample pair $(x_t, y_t)$
  Compute $\delta_t$ according to algorithm 1
  Compute $\pi_t$ using (3.10) $O(\tilde{m}_t^2)$
  if $\delta_t \leq \nu$ (almost linearly dependent) then
    Maintain the dictionary: $D_t = D_{t-1}$
    Update $\tilde{W}_t$ using (A 14) $O(\tilde{m}_t^2)$
    Compute $h_t$ using (A 7) $O(\tilde{m}_t^2)$
    Update $P_t$ using (A 9) $O(\tilde{m}_t^2)$
  else if $\delta_t > \nu$ (not almost linearly dependent) then
    Update the dictionary: $D_t = D_{t-1} \cup \{x_t\}$
    Update the Cholesky factor $C_t$ using (3.13) $O(\tilde{m}_t^2)$
    Update $P_t$ using (A 15) $O(\tilde{m}_t^2)$
    Update $\tilde{W}_t$ using (A 17) $O(\tilde{m}_t^2)$
    Form the model $f$ according to (3.4)
  end if
end for

Define $S$ according to (4.7)
Form $L$, $c$ and $N$ according to section 4(a)
Compute the eigendecomposition of $\hat{L}$ according to lemma 1
Form the eigenvectors $\psi$ and eigenvalues $\lambda$ according to (4.8)

Case II: Not almost linearly dependent
In this case, the new vector is not almost linearly dependent. Accordingly, we must add $x_t$ to the dictionary: $D_t = D_{t-1} \cup \{x_t\}$.

The update rules for $\Pi_t$ and $P_t$ are simply

$$\Pi_t = \begin{bmatrix} \Pi_{t-1} & 0 \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad P_t = \begin{bmatrix} P_{t-1} & 0 \\ 0 & 1 \end{bmatrix}. \quad (A 15)$$

The update rule for $\tilde{W}_t$ is

$$\tilde{W}_t = Y_t \Pi_t \Pi_t \Pi_t = \begin{bmatrix} Y_{t-1} \Pi_{t-1} \Pi_{t-1} & y_t \end{bmatrix} \hat{K}_{t-1}^{-1}. \quad (A 16)$$

Finally, (3.12) allows us to express the update rule as

$$\tilde{W}_t = \tilde{W}_{t-1} + \left( \tilde{W}_{t-1} \hat{k}_{t-1} - y_t \right) \frac{\pi_t^2}{\delta_t} \quad \left( \tilde{W}_{t-1} \hat{k}_{t-1} - y_t \right)^{-1}. \quad (A 17)$$

This completes the derivation of the equations for the online regression algorithm. Alternatives to this proposed dictionary learning procedure include randomised methods [20, 25] and the recent method in [9].

E. Learning control laws

Our algorithm may also be used to simultaneously learn control laws and governing equations from data. An active control variable can significantly alter the behaviour of a dynamical system
and thus further disguise the underlying dynamics [14, 18]. In many practical scenarios – such as epidemiological modelling of disease spread where the control variables could be the distribution of vaccinations – it is infeasible to gather data on the unforced system so the effects of control must be disambiguated from the data [19]. This strategy can be used to uncover the dynamics of the unforced system, which can then inform design of effective control strategies. In such systems the underlying dynamics take the form

\[ y = F(x, u) \]  

(A 1)

where \( u \) is the control variable. Similarly to \( x \) and \( y \), the values of \( u \) are known at each snapshot time. As in dynamic mode decomposition with control (DMDc, [18]), we may write the supplemented state vector as \( \omega = [\begin{bmatrix} x \\ u \end{bmatrix}] \) so that (A 1) may be expressed as

\[ y = G(\omega). \]  

(A 2)

Our task is now to learn a model \( g \) that approximates the underlying system defined by \( G \).

We can use ideas explained in section 4(d) to design a suitable kernel. In particular, we can exploit the fact that kernels are closed under direct sums (4.17). Unless we believe that there are nonlinear pairings between the control variable and the state space, we can assume a kernel of the form

\[ k(\omega, \omega') = k_x(x, x') + k_u(u, u'). \]  

(A 3)

For example, we may have reason to believe that \( y \) is generated by quadratic interactions between the states \( x \) whereas the control variable has only a linear effect. Then, the kernel

\[ k(\omega, \omega') = (x^* x')^2 + (u^* u') \]  

(A 4)

induces the appropriate feature space. The algorithm of section 3 can then be applied with \( X \) replaced by the augmented data matrix \( \Omega = [\begin{bmatrix} \hat{X} \\ \hat{U} \end{bmatrix}] \) to learn a model of the form

\[ g(\omega) = \hat{W} k(\hat{\Omega}, \omega) \]  

(A 5)

where \( \hat{\Omega} \) is the augmented dictionary matrix. If the kernel takes the form (A 3) then the reconstruction/prediction model is

\[ g(\omega) = \hat{W}_x k_x(\hat{X}, x) + \hat{W}_u k_u(\hat{U}, u) \]  

(A 6)

where \( \hat{\Omega} = [\begin{bmatrix} \hat{X} \\ \hat{U} \end{bmatrix}] \) and \( \hat{W} = [\begin{bmatrix} \hat{W}_x \\ \hat{W}_u \end{bmatrix}] \). The unforced system can then be modelled by setting \( \hat{W}_u = 0 \). Furthermore, we can also compute local linear models (i.e., DMDc models) of the unforced system using the ideas of section 4(a). The analysis follows exactly except \( \hat{W} k(\hat{X}, x) \) in section 4(a) is replaced with \( \hat{W}_x k_x(\hat{X}, x) \). Note that if the kernel is taken to be

\[ k(\omega, \omega') = \omega^* \omega' = (x^* x') + (u^* u') \]  

(A 7)

then we recover the original DMDc formulation [18].

These ideas are also valid for kernels that don’t take the form (A 3) but the algebra is slightly more involved and we therefore do not report the results here.

F. Sensitivity to noise

All machine learning algorithms must be understood in the context of their sensitivity to noise. To explore the effects of noise, we applied our learning framework to noise-contaminated data generated by the Lorenz system from section 6(a). The data are contaminated with Gaussian noise of magnitude 5% of the variance of the original data; the noisy training data is visualised in the left panel of figure 11. We use the same parameters as section 6(a), but the system is now integrated to \( t = 50 \). Unlike section 6(a), we assume that we only have access to snapshot measurements of \( x \) and velocity measurements \( \dot{x} \) are unavailable. Therefore, we approximate the derivative \( \dot{x} \) from the noisy snapshot data with a total-variation regularisation scheme [5]. Then, we use
the algorithm of section 3 to learn the Lorenz system with a quadratic kernel. The results of the learned model are illustrated in the middle panel of figure 11, along with the true local linear model, both evaluated at the equilibrium point $\mathbf{x} = \begin{bmatrix} -\sqrt{3}(\rho - 1) & -\sqrt{3}(\rho - 1) & \rho - 1 \end{bmatrix}^T$. The reconstructed trajectory shows good qualitative agreement with the true model, and the local linear model is a good approximation to the true linearisation. The accuracy of these approximations usually improves as more samples are added.

We also demonstrate the effect of noise on identifying the spectrum of the viscous Burgers’ equation from section 5(c). Here, the kinematic viscosity is $\nu = 0.1$ and the equations are integrated to $t = 4$. The data are snapshots of the solution which are then corrupted by Gaussian noise of varying magnitude. No velocity measurements $\dot{\mathbf{x}}$ are used and we do not de-noise the data. Figure 12 plots the first three (repeated) eigenvalues learned by the algorithm for 20 trials with a quadratic kernel. The first two eigenvalues are recovered accurately for small values of
noise but the sensitivity to noise increases for eigenvalues of larger magnitude. Again, these approximations can be improved by adding more samples, or by a suitable de-noising.

Further experiments indicate that, in the absence of de-noising, the algorithm is robust to noise in $Y$ but relatively sensitive to noise in $X$. These observations can be explained through the linear algebra and statistics underlying our kernel learning framework. The impact of noise in $X$ is felt in two forms. Firstly, noise may cause elements to be included in the dictionary which may otherwise have been excluded. The dictionary is independent of $Y$ and is therefore unaffected by noise in $Y$. If the sparsification parameter is not chosen carefully then noise in $X$ may cause the dictionary to become dense. Secondly, noise causes errors in the final regression, whether performed online or in batch. As is typical of least-squares regressions, our algorithm is an unbiased estimator when the noise is restricted to $Y$. This is because least-squares methods implicitly assume that there are no “errors in variables” [10]. This assumption becomes invalid when $X$ is contaminated by noise. As such, when noise is present, the naïve pseudoinverse (3.14) solution may prove insufficient. For example, a similar issue arise in DMD when there is noise in $X$, and several approaches have been proposed to mitigate the effects of noise [1–3, 6, 12, 22], for example solving the regression problem with total least squares (TLS) [12]. Experiments with TLS in our present setting were found to be unsuccessful, in part because the TLS problem is unstable [11] but also because our problem is nonlinear. In particular, TLS only guarantees the best solution to $\arg\min_{W} \| W^T k(X, X) - Y \|_F$ only when the errors in $X$ and $Y$ are column-wise independent and identically distributed with zero mean and covariance matrix $\sigma^2 I$ [13, chapter 8]. Therefore, there are no guarantees of the effectiveness of TLS for our problem, which is $\arg\min_{W} \| W^T k(X, X) - Y \|_F$. In the case of nonlinear dynamics, whether with SINDy or our kernel approach, the noise in $X$ is stretched and transformed through the nonlinearity, adding nonlinear correlations.

In summary, we have demonstrated that our algorithm can remain effective in the presence of noise. The Lorenz example in figure 11 indicates that the algorithm is effective when applied to derivatives computed from noisy data by total-variation regularisation. Additionally, experiments and theory suggest that the present framework is insensitive to noise in $Y$ but moderately sensitive to noise in $X$. In addition to total-variation regularisation, there are several other methods that could be deployed to limit the effects of noise. For example, one technique could combine the KRLS algorithm with a Kalman filter, as explored in [16]. Another filtering approach would be to use the optimal hard threshold criterion for singular values of [8]. Fully addressing the challenge of noise is an area of active ongoing research and will be the focus of future work.

References


