System Identification through Expression Optimization

Anonymous Authors[1]

Abstract

With the abundance of natural data from physical systems, much engineering and scientific value comes from an ability to discover the underlying, governing equations of a system, with little prior knowledge. Current approaches for data-driven system identification either find relationships in the data that aren't interpretable, or require significant prior knowledge from the user. This work describes a new approach to system identification that requires minimal user input and discovers governing equations that are parsimonious, generalizable and interpretable. This is enabled by recent advances in expression optimization, allowing for the automated discovery of mathematical expressions from a combinatorically large set of possibilities. Using simulated data, our approach correctly identifies both linear and nonlinear PDEs including the Navier-Stokes equations. It can also generate exact and approximate Koopman eigenfunctions for nonlinear ODEs. The ability to interpret large amounts of data will allow researchers to better understand and control important natural systems, such as the earths climate, for addressing global warming and fluid flow for more efficient energy generation and transportation.

1. Introduction

Over the past several decades machine learning and artificial intelligence have made great strides in learning patterns from large amounts of data. Although accurate, these systems are often uninterpretable by the human researchers who create them. They do not report an explanation for their predictions nor do they generalize well when the task is changed slightly.

Recently, however, strides have been made to create AI systems that, rather than just looking for trends, look for causal explanations of observed data (Bridewell, 2008, Brunton, 2016). If an AI system can correctly determine a simple underlying model for a system, then it can provide an explanatory account of the data and generalize in predicting behavior under a wider variety of conditions. This can help researchers more quickly discover models that explain experimental data.

[[Describe the current systems and there limitations. Specifically that system identification approaches fall into two categories 1) Those that require no user input but produce opaque results 2) Those that produce interpretable results but require lots of prior knowledge]]

The goal of this work was to create a system that could discover, from data, the governing equations or Koopman eigenfunction of a dynamical system. The system should require minimal prior knowledge from the user, be robust to noise and operate on small amounts of data. This paper is organized as follows. Section 2 describes how governing equations could be derived from data as well as the connection between equation discovery and the approximation of the Koopman operator. Section 3 details the approach known as system identification through expression optimization (SIEO) and section 4 reports the results of SIEO applied to four model systems. Finally, section 5 concludes and discusses directions for future investigation.

2. System Identification

The goal in system identification is to find a parsimonious description of a dynamical system using data. This section will outline some of the previous work on system identification, and then discuss the two forms of identification that are the focus of this work.

2.1. Past Approaches

[[

- **SINDy and HPM where you have to specify entire features**
- **DMD/POD where you get a low dimensional approximation without much user input but the resulting modes don't make contact with domain con-**

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

cepts.

- **extended DMD for Koopman approximation (need to specify features)**

]]

2.2. Governing Equations

One approach to understanding a dynamical system is to determine its underlying governing equations. If the system is a spatio-temporal process, then governing equation can be written as partial differential equation (PDE) of the form

$$\frac{\partial x(\mathbf{r},t)}{\partial t} = \theta_1 g_1(x(\mathbf{r},t)) + \ldots + \theta_n g_n(x(\mathbf{r},t)) \quad (1)$$

where $\mathbf{r}$ are the spatial dimension, x is a state variables, g_i are feature functions and θ_i are the corresponding coefficients. Given a mapping g_i, and a state x, the coefficients of the PDE can be found via a linear model of the form

$$y = \theta^T \tilde{x} \quad (2)$$

where $y = \partial u / \partial t$ and $\tilde{x} = \begin{bmatrix} g_1(u), & \ldots, & g_n(u) \end{bmatrix}^T$. A good approximation of θ^T can be found by sampling many y and $\tilde{x}$ points and then performing multiple linear regression. The goal, then, is to find the optimal set of features g_i (and corresponding θ_i) for producing the best fit between y and $\tilde{x}$.

Note that when discovering a governing equation, only the input data is transformed by a nonlinear mapping, while the output data remains unchanged. If a nonlinear mapping were found that could be applied to both the input and output, then it would be possible to convert a nonlinear system into a linear one. This is known as finding the Koopman operator of a system and it is discussed in the next section.

2.3. Koopman Operators

Nonlinear dynamical systems are far more challending to understand and control than their linear counterparts. One promising area of analysis is Koopman theory [CITATION] which states that any nonlinear dynamical system can be converted into a linear system under the state-space mapping

$$x \to \mathbf{g}(x) \quad (3)$$

where $\mathbf{g}(x) = \begin{bmatrix} g_1(x), g_2(x), \ldots \end{bmatrix}^T$ are the Koopman eigenfunctions. In general, $\mathbf{g}$ is infinite dimensional and therefore it must be approximated with a finite-dimensiona mapping.

The Koopman operator can take a nonlinear dynamical system of the form

$$\mathbf{x}^{t+1} = \mathbf{f}(\mathbf{x}^t) \quad (4)$$

and transform it into the linear system

$$\mathbf{g}(x)^{t+N} = K^N \mathbf{g}(x)^t \quad (5)$$

Thus, if the the Koopman operator is known, all of the mathematical technologies developed around linear systems can be brought to bear on a nonlinear system.

If the Koopman eigenfunctions contain the identity function $g(x) = x$ then the Koopman operator is said to be *state-inclusive* and the un-transformed state of the system can be found directly from the linear system (5), without finding an inverse transformation $\mathbf{g}^{-1}$.

Given a finite approximation to the Koopman eigenfunctions $\mathbf{g}(x) = \begin{bmatrix} g_1(x), & \ldots, & g_n \end{bmatrix}^T$, the matrix K can be found via a linear model of the form

$$y = K\tilde{x}$$

where $y = \mathbf{g}(x^{t+1})$ and $\tilde{x} = \mathbf{g}(x^t)$. If many points are sampled for y and $\tilde{x}$ then multiple linear regression can provide the least-squares estimate of K. The goal for approximating the Koopman operator is to find finite approximation $\mathbf{g}(x)$ that creates the best fit between y and $\tilde{x}$.

It should now be clear that the problems of equation discovery and approximating the Koopman operator are identical except for the transformation of the output data in the latter case. The next section describes a new approach to finding these nonlinear mappings from the vast number of possibilities.

3. System Identification through Expression Optimization (SIEO)

The process of discovering a nonlinear mapping for system identification from data has three steps.

1. Define a set of nonlinear features
2. Define a loss function to measure the success of a group of features (referred to as a *mapping*)
3. Search through the possible mappings to find the optimal

The first and third task each involve searching through a combinatorically large set of possibilities and therefore pose a substantial computational challenge. The next subsection discusses how the first task can be performed efficiently using a context-free grammar. The next subsection describes two useful loss functions for the two system identification tasks. The last section describes a search technique that combines expression optimization with a greedy heuristic search for finding the optimal mapping. The combination of these techniques comprise the approach known as System Identification through Expression Optimization (SIEO).

3.1. Context-Free Grammar

There are an infinite number of features that could make up a mapping, but, researchers don't generally know a priori which features will be good candidates. Thus, a design goal of SIEO is to require minimal prior knowledge from the user while still constraining the possible features to be searched.

To this end, SIEO relies on a domain-specific *grammar*, provided by the user, that gives the rules for generating feature functions. A grammar is a set of production rules that govern a language (or a set of expressions). Each rule can either be *non-terminal*, in which the rule relates generic expressions together (e.g. multiplication), or *terminal*, in which an expression is concretely defined (e.g. the observed data). Each expression in the language can be represented by a tree of operators, each of which is part of the grammar. Once a grammar is defined, expressions can be sampled from the grammar with varying levels of complexity (as defined by the depth of the expression tree).

For example, the grammar in figure 2 has rules for differentiation, multiplication, and division. Sampling from a grammar is a process of selecting production rules and then filling in any non-terminal expressions until only terminal expressions remain. For example, in the expression

$$u\frac{\partial u}{\partial x} \tag{6}$$

the first (non-terminal) production rule chosen is multiplication. Then, on the left side, the terminal expression u was chosen, and on the right side, the non-terminal derivative rule was chosen, followed by the terminal expression u. Grammars define a constrained, but still infinite, set of possible expressions which can be made finite be specifying a maximum expression depth for a grammar.

The benefit to this approach is two-fold

1. For the cost of defining a simple grammar, an infinite number possible expressions (ranging from simple to complex) can be produced
2. The researcher can encode into the grammar any information she may have regarding relationships between the input and output data.

As will be demonstrated in later sections, many familiar PDEs have features that come from a very simple grammar (2-3 rules) with a very short depth (2-3). The next subsection describes the loss functions used to compare different features.

3.2. Choice of Loss Function

When attempting to discover a governing equation to explain data, the object is to find a mapping that describes the output data well, but also to find a mapping that is simple (i.e. has fewer features). To balance this multi-objective, the negative adjusted R^2 value (R^2_{adj}) of the fit was used for the loss function. The traditional R^2 value always increases when new features are added so it makes our algorithm susceptible to overfitting. R^2_{adj}, however, penalizes the addition of more features via [CITATION]

$$R^2_{\text{adj}} = R^2 - \alpha(1 - R^2)(n-1)$$

where n is the number of features in the model. Note that when $n = 1$, this expression reduces to the traditional R^2 value. But when $n > 1$, then the difference between R^2 and 1 is scaled by the number of parameters. Thus, as a rule of thumb, $\alpha \in [0, \infty)$ should be large when you expect there to be excellent agreement between the input and output (i.e. for synthetic, noise-free data) and should be closer to 0 when near-perfect agreement is not expected (i.e. real world data). The loss function is explicitly defined as

$$l_R(\theta, \mathbf{g}) = -R^2_{\text{adj}}(\theta, \mathbf{g}) \tag{7}$$

For the goal of discovering Koopman eigenfunctions, however, parsimony is no longer a consideration so a new loss function needs to be used. Given a linear system with n_g number of Koopman eigenfunctions, the loss function associated with the Koopman operator $K\mathbf{g}$ is given by

$$l_k(K\mathbf{g}) = \frac{1}{n_g}||\mathbf{g}(\mathbf{x})^{t+1} - K\mathbf{g}(\mathbf{x})^t||_2^2$$

The scaling by $1/n_g$ is so that that algorithm does not get penalized for adding more Koopman eigenfunctions to approximate the nonlinear system, and instead focuses on closely matching each new state variable on each iteration. This loss function is referred to as the *average sum of squares*. The following subsection ties together the use of a grammar and the loss functions to explain how expression optimization and heuristic search are used to select features.

3.3. Feature Selection Algorithms

The problem of selecting from a group of possible features is a challenging one. If there are N possible features then there are 2^N possible subsets to pick from. It is infeasible to search the entire space of combinations for even moderate values of N, so heuristic searches need to be employed.

Previous approaches have used sparsity-promoting regression techniques such as LASSO [Citation] but these approaches are brittle when the input data suffers from mulicollinearity [PDE citation]. In this work, the heuristic that performed the best was a modified form of a greedy search algorithm that combined both forward and backward search. The high-level structure of the algorithm is given in Algorithm 1.

Algorithm 1 Feature Selection

```
Input: Loss function $l_f$,
       New feature generator $n_f$,
       Threshold $t$
Initialize $mapping = [\,]$
repeat
  $old\_mapping \leftarrow mapping$
  $mapping = \text{forward\_search}(mapping, lf, n_f, t)$
  $mapping = \text{backward\_search}(mapping, lf, n_f, t)$
until $old\_mapping = mapping$
```

Algorithm 2 Forward Search

```
Input: Loss function $l_f$,
       Current mapping $mapping$,
       New feature generator $n_f$,
       Threshold $t$
Initialize $lowest = l_f(mapping)$
repeat
  $mapping = \text{concat}(mapping,\ n_f(mapping))$
  if $l_f(mapping) < lowest$ then
    $lowest = l_f(mapping)$
  else
    return $mapping$
  end if
until $lowest < t$
```

The algorithm starts by performing a greedy forward search (Algorithm 2), where, on each iteration, a feature is greedily added to the mapping until there is either no further improvement or a specified threshold is hit. Then, the feature selection algorithm performs a greedy backward search on the mapping (Algorithm 3), where features are removed if their absence reduces the loss function. This entire process iterates until there is no further improvement of the mapping, at which point it is returned.

The substantial missing piece of this process is how to find the next feature to add to the list. As described previously, a comprehensive search is infeasible, so an expression optimization technique must be used.

3.3.1. Expression Optimization

There are several excellent algorithms for expression optimization using a grammar, but the one used in SIEO is Grammatical Evolution (GE), an expression optimization approach introduced by [CITATION]. Each feature that is sampled from the grammar is expressed as an unbounded integer array that can be parsed from left to right into an expression tree. Each integer array is transformed in a manner similar to genetic algorithms. The basic operations are

Algorithm 3 Backward Search

```
Input: Loss function $l_f$,
       Current mapping $mapping$,
       Threshold $t$
Initialize $lowest = l_f(mapping)$
repeat
  Initialize $losses = [\,]$
  for $feat$ in $mapping$ do
    $new\_mapping = \text{remove}(mapping, feat)$
    $\text{push}(losses, l_f(new\_mapping))$
  end for
  $i = \arg\max(losses)$
  if $losses[i] < lowest$ then
    $lowest = losses[i]$
    $mapping = \text{remove}(mapping, feat)$
  else
    return $mapping$
  end if
until $lowest < t$
```

1. **Selection** - The integer arrays with the best fitness (lowest loss function) are selected from the population
2. **Crossover** - The fittest individuals produce offspring that carry traits from the parents
3. **Mutation** - The integer arrays are randomly altered (while preserving the rules of the grammar)
4. **Gene Duplication** - Where useful genes are copied in the integer array so that they are less likely to be lost
5. **Pruning** - Some integer arrays will contain genetic information that is unused (put there through crossover). So with some probability unused genes are pruned in an individual.

These operations are applied to a large number of individuals over many generations in order to produce optimal features from the large space of possibilities. In the following section, SIEO is applied to several model systems to demonstrate several possible applications.

4. Results

SIEO was tested on four separate model problems to demonstrate its capabilities and probe its robustness. The first two problems are equation discovery for PDEs. The first is used to demonstrate that SIEO can operate with limited and noisy data. The second is to demonstrate that the algorithm scales to problems with multiple spatial dimensions and multiple state variables. The next two model systems show the utility of SIEO for approximating Koopman functions, first, by finding them exactly and then by finding a good finite approximation.

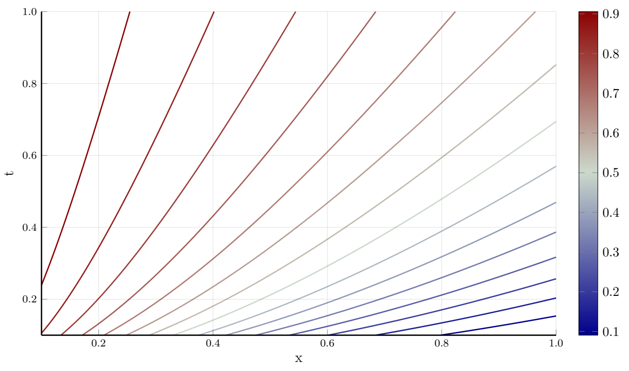


Figure 1. Unsteady data from the 1D advection-diffusion equation

4.1. Advection-Diffusion Equation

The first model problem is the unsteady 1D advection-diffusion equation which describes the general linear transport of material u in a fluid. The governing equation has the form

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} - v\frac{\partial u}{\partial x} \tag{8}$$

where D is the diffusion coefficient, v is the fluid velocity. Synthetic data is generated using the exact solution derived by [Van Genuchten citation 1982] which for boundary conditions

$$u(x,0) = u_0 \tag{9}$$

$$u(0,t) = u_L \tag{10}$$

$$\frac{\partial u}{\partial x}(\infty,t) = 0 \tag{11}$$

is found to be

$$\begin{aligned} u(x,t) = u_0 + \frac{1}{2}(u_L - u_0)\Bigg(&\operatorname{erfc}\left[\frac{x-vt}{2\sqrt{Dt}}\right] \\ &+ \exp(vx/D)\operatorname{erfc}\left[\frac{x+vt}{2\sqrt{Dt}}\right]\Bigg) \end{aligned} \tag{12}$$

This solution is plotted in figure 1 for $x > 0, t > 0, D = 1$, $v = 1$, and $U_L = 1$.

To apply SIEO the exact solution was sampled at 200 temporal points in the range $t = (0, 0.1)$ and between 10 and 200 spatial points. Then, additive white Gaussian noise at a level $\eta \in (0, 0.2)$ was added to the data such that the standard deviation of the noise is given by $\sigma = \eta\,\mathrm{std}(u)$. The grammar shown in figure 2 was used for the expression optimization routine and was limited to a tree depth of 3. The adjusted R^2 loss function was used with the parameter $\alpha = 0.5$ and the solution was denoised using total variation filtering [tv citation] both before and after the numerical differentiation. The results of six test cases are shown in table 1. After the trial number, the first two columns show the number of spatial sample points used and the noise level, respectively. The last two columns show the percent error in the model parameters, v and D when compared to their exact values. In all six test cases, SIEO determined the correct form of the governing equation. When the noise is 1-5 % and the number of sample points is 35 or larger, the error in the parameters remains small. In the two most challenging test cases (trials 3 and 6), the fit parameters v and D show high levels of error but the correct form the equation was still discovered accurately which demonstrates the robustness of SIEO to find an accurate and generalizable model with large amounts of noise or limited data.

$$\mathbb{R} \mapsto u$$
$$\mathbb{R} \mapsto \frac{d\mathbb{R}}{dx}$$
$$\mathbb{R} \mapsto \mathbb{R} \times \mathbb{R}$$
$$\mathbb{R} \mapsto \mathbb{R}/\mathbb{R}$$

Figure 2. Grammar used for the advection-diffusion equation

Table 1. Error in solution parameters for different noise levels and spatial resolutions

Trial	Sample Points	Noise	Error in v	Error in D
1	200	1%	0.78 %	1.54 %
2	200	5%	5.64 %	2.84 %
3	200	20%	30.12 %	6.60 %
4	100	0%	0.84 %	0.29 %
5	35	0%	6.38 %	2.21 %
6	10	0%	1.19 %	19.02 %

4.2. Navier-Stokes Equations

The second test system is the incompressible 2D Navier-Stokes equations which describe the motion of a viscous fluid. The Navier-Stokes equations are given by

$$\frac{\partial u}{\partial x} + \partial v \partial y = 0 \tag{13}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \frac{\mu}{\rho}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{14}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial v} + \frac{\mu}{\rho}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{15}$$

where ρ is the fluid density, (u, v) is the fluid velocity, p is the pressure, and μ is the viscosity. The first equation represents the conservation of mass and the next two equations represent conservation of momentum in the x and y directions, respectively. These equations are nonlinear, have multiple dimensions (x and y) and multiple state variables (u, v, and p), which makes them a challenging model system to identify.

Synthetic data was collected from the simulation of vortex

Table 2. Error in Navier-Stokes Coefficients

PARAMETER	ERROR
θ_{x2}/θ_{x1}	2.87 %
θ_{x3}/θ_{x1}	1.00 %
θ_{x4}/θ_{x1}	2.53 %
θ_{y2}/θ_{x1}	8.47 %
θ_{y3}/θ_{x1}	0.92 %

shedding behind a circular cylinder at a Reynold's number of 50. The simulation was performed using *PyFR* [citation] and 10 snapshots were taken with a timestep of $dt = 0.001$, $M = 0.2$ and $\rho \approx 1$ (with a very small amount of compressibility in the solution). A single frame of the simulation is shown in figure 3, with u, v, and p on separate plots. The grammar in figure 4 was used to produce features up to a depth of 3, yielding 363 features and 2^{363} mappings to be searched.

SIEO was performed on the the vortex shedding data with the adjusted R^2 loss function and $\alpha = 0.2$. The algorithm found equations 16 and 17 to explain the unsteady Navier-Stokes data

$$\frac{\partial u}{\partial t} = \theta_{x1}\frac{\partial}{\partial y}(uv) + \theta_{x2}v\frac{\partial u}{\partial y} + \theta_{x3}\frac{\partial p}{\partial y} + \theta_{x4}\frac{\partial^2 u}{\partial y^2} \quad (16)$$

$$\frac{\partial v}{\partial t} = \theta_{y1}u\frac{\partial v}{\partial x}(uv) + \theta_{y2}v\frac{\partial v}{\partial y} + \theta_{y3}\frac{\partial p}{\partial y} \quad (17)$$

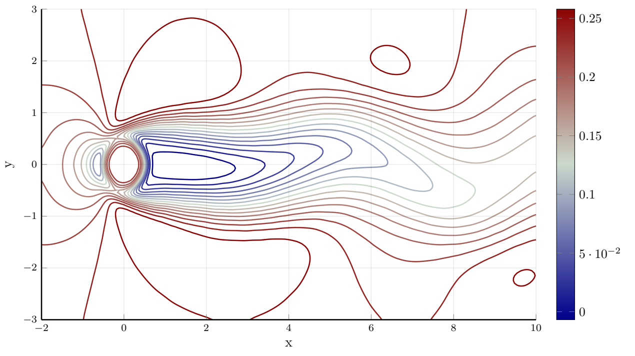
Note that the first equation requires the substitution $\partial v/\partial y = -\partial u/\partial x$ (from the continuity equation) to transform it to the more well-known form of the x-momentum equation. Also note the absence of the viscous terms $\partial^2 u/\partial x^2$, $\partial^2 v/\partial x^2$, and $\partial^2 v/\partial y^2$. These terms are not included because in this particular fluid flow they are much smaller in magnitude than the other terms in the governing equation. This shows that for SIEO to identify a process, that process must be sufficiently active in the data collected.

The ratios of the model coefficients were compared to determine the error in the model parameters. The results are shown in table 2. In general, the model parameter error is very small and the discovered equations could be used for prediction or control of other similar types of flow field (i.e. those that don't have significant contributions from the three omitted viscous terms).

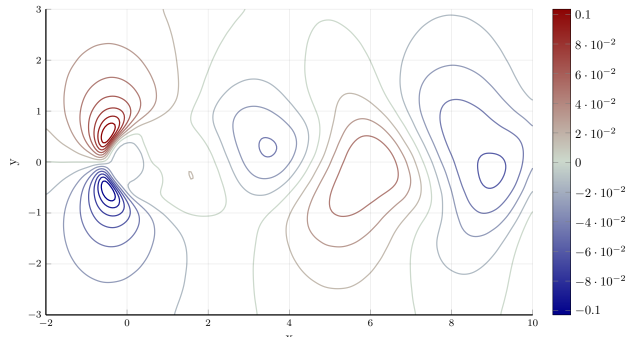
4.3. Exact Koopman Eigenfunction Discovery

A simple nonlinear ODE with an easily found, and finite-dimensional Koopman eigenfunctions is given by [Brunton citation]

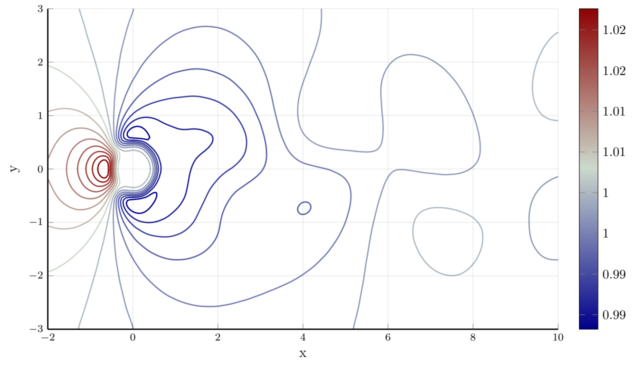
$$\begin{aligned}\frac{dx}{dt} &= \mu x \\ \frac{dy}{dt} &= \lambda(y - x^2)\end{aligned} \quad (18)$$



(a) x-velocity



(b) y-velocity



(c) pressure

Figure 3. Global caption that can reference **??** and **??**

$$\begin{aligned}&\mathbb{R} \mapsto u \quad | \quad v \quad | \quad p \\ &\mathbb{R} \mapsto \frac{d\mathbb{R}}{dx} \quad \Big| \quad \frac{d\mathbb{R}}{dy} \\ &\mathbb{R} \mapsto \mathbb{R} \times \mathbb{R}\end{aligned}$$

Figure 4. Grammar used for the Navier-Stokes equations

The state-space transformation

$$\begin{bmatrix} x \\ y \end{bmatrix} \Rightarrow \begin{bmatrix} x \\ y \\ x^2 \end{bmatrix} \tag{19}$$

linearizes equation 18 to

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ x^2 \end{bmatrix} = \begin{bmatrix} \mu & 0 & 0 \\ 0 & \lambda & -\lambda \\ 0 & 0 & 2\mu \end{bmatrix}\begin{bmatrix} x \\ y \\ x^2 \end{bmatrix} \tag{20}$$

To test the discovery of exact Koopman operators, data from the system (18) was collected data for many initial conditions. The initial data is stored in a vector $\mathbf{x}^0$, then the solution is integrated forward one timestep and the result is stored in another vector $\mathbf{x}^1$. SIEO was then applied with the average sum of squares loss function l_k. The goal was to find a state-inclusive Koopman operator so the loss function was modified to penalize the absence of the primary state variables x and y. The function-rich grammar in figure 5 was used as input (exchanging (θ, ω) for (x, y)). The correct Koopman transformation was discovered (equation 19) and the Koopman operator (equation 20) was recovered to within machine precision.

4.4. Approximate Koopman Eigenfunction Discovery of Nonlinear Pendulum

One of the simplest nonlinear ODEs without a known Koopman operator, is a simple pendulum swinging at large angles. The equation of motion a pendulum is given by

$$\frac{d^2\theta}{dt^2} + \sin\theta = 0$$

This equation can be converted to a system of nonlinear first-order ODEs given by

$$\frac{d\theta}{dt} = \omega$$
$$\frac{d\omega}{dt} = -\sin\theta$$

In order to generate training data, the system was integrated forward in time from a starting position $(\theta_0, \omega_0) = (\pi/2, 0)$ until $T = 10$. Koopman operator was approximated by SIEO using the average sum of squares loss function with the same state-inclusive penalty described in section 4.3. The function-rich grammar shown in figure 5 was used as the input. The resulting expression with 24 features was then compared to a linearized version of the equation with only the primary state variables. Both linear models were initialized at $(\theta_0, \omega_0) = (\pi/2, 0)$ and were allowed to evolve forward in time past the training period. The error between the linear model and the exact solution for each model is shown in figure 6.

$$\begin{aligned}
&\mathbb{R} \mapsto \theta \quad | \quad \omega\\
&\mathbb{R} \mapsto \mathbb{R} \times \mathbb{R}\\
&\mathbb{R} \mapsto \sin(\mathbb{G} \times \mathbb{R}) \quad | \quad \sin(\mathbb{G} \times \mathbb{R} + \mathbb{G})\\
&\mathbb{R} \mapsto \exp(\mathbb{G} \times \mathbb{R})\\
&\mathbb{R} \mapsto 1/(1 + \exp(-\mathbb{G} \times \mathbb{R}))\\
&\mathbb{R} \mapsto \exp(-(\mathbb{R} - \mathbb{G})^2/\mathbb{G})\\
&\mathbb{R} \mapsto \mathrm{imag}(\mathbb{R}^{\mathbb{G}})\\
&\mathbb{R} \mapsto \mathrm{real}(\mathbb{R}^{\mathbb{G}})\\
&\mathbb{G} \mapsto -\mathbb{G} \quad | \quad \mathbb{G} + \mathbb{G} \quad | \quad \mathbb{G}/\mathbb{G}\\
&\mathbb{G} \mapsto \mathrm{logspace}(-5, 2, 20)
\end{aligned}$$

Figure 5. Grammar used for both nonlinear ODEs

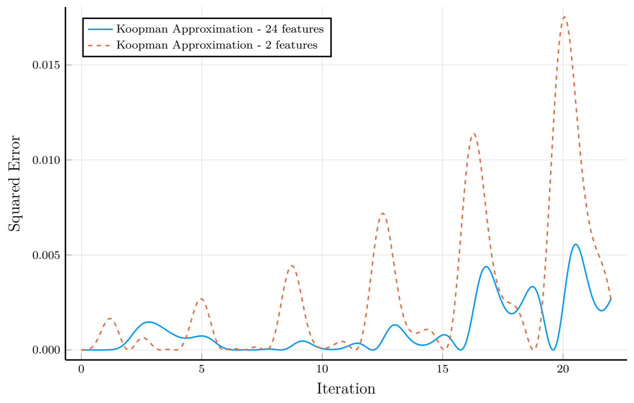


Figure 6. Error in the nonlinear pendulum for the simplest linear model and linear model with more nonlinear features

The figure shows that the linear model with more nonlinear features has significantly lower average error then the linear model with only the two state variables. Additionally, the error more slowly for the Koopman approximation. This shows that it is possible to improve linearizations of nonlinear model by using expression optimization to approximate the Koopman eigenfunctions of the system.

5. Conclusions

- Robustness in finding correct features in the presence of noise and limited data make the resulting model very generalizeable
- It can seach through a vast number of possible features and mappings – even in problems with multiple state variables and multiple dimenions
- Presesnts a new way of finding exact and approximate Koopman operators

[[Summarize and conclude]]