System Identification through Expression Optimization

Anonymous Authors[1]

Abstract

Many successful machine learning algorithms find patterns in data using complex and uninterpretable models. This work describes an approach to system identification that requires minimal domain knowledge and discovers governing equations that are parsimonious, generalizable, and interpretable. This approach is enabled by techniques in expression optimization that allow for the automated discovery of mathematical expressions from a combinatorially large set of possibilities. Using simulated data, our approach correctly identifies both linear and nonlinear partial differential equations including the Navier-Stokes equations, as well as discovers exact and approximate Koopman eigenfunctions for nonlinear dynamical systems.

1. Introduction

Over the past several decades, machine learning has made significant strides in learning patterns from large amounts of data. Although accurate, these systems often produce models that are not interpretable by the human researchers who use them. They often do not report a causal explanation for their predictions and can generalize poorly when the task is altered. Constructing intrepretable and generalizable models will allow researchers to better understand and control important physical systems such as complex fluid flows for more efficient energy generation and transportation.

System identification refers to the process of finding models that are simple and generalize well to similar systems. For maximum interpretability, a system identifier should output models expressed in mathematical terms that are understandable by a domain expert. Examples of interpretable models are partial differential equations (PDEs), ordinary differential equations (ODEs), or explicit functions that transform a nonlinear dynamical system into a linear one.

[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.

It is computationally challenging to find an effective and parsimonious mathematical model from data. Finding such a model requires the generation of a set of possible mathematical descriptions (of which there are uncountably many) and then searching that set for the best subset that describes the data.

Current system identification approaches tend to fall into one of three categories: (1) approaches that require little user input but produce results that are difficult to interpret, (2) approaches that produce interpretable output but require significant prior knowledge to use, and (3) approaches that automatically generate mathematical expressions but are hindered by the complexity of that task.

The first category contains methods like Dynamic Mode Decomposition (DMD) (Schmid, 2010; Kutz et al., 2016), which projects the dynamics of a system onto the proper orthogonal decomposition (POD) of the state-space. The result is a low-dimensional, linear model of the dynamics, which can be used for prediction and control. However, the POD modes do not describe the dynamical system in a way that connects with domain-specific concepts, making DMD less interpretable by human researchers. Another approach is used by Morton et al. (2018) where a deep convolutional neural network is used to linearize a high-dimensional nonlinear fluid system. The resulting linearization is contained in a complicated neural network that cannot easily be understood.

The second category includes the method of Brunton et al. (2016) known as sparse identification of nonlinear dynamical systems (SINDy), which has been applied to PDEs (Rudy et al., 2017) and dynamical systems with limited data (Kaiser et al., 2018). SINDy uses sparse regression on a set of features that are provided by the user to determine which are most effective at explaining the data. A similar technique is used in inductive process modeling (Bridewell et al., 2008; Langley & Arvay, 2015), which is typically applied to ODEs with many state variables. The system of Krizman et al. (1995) can generate new feature functions but requires a comprehensive search of all possible features combinations, making it challenging to scale to larger problems. There have also been attempts to linearize nonlinear dynamical systems by explicitly defining a set of basis functions for a nonlinear mapping to linearize a dynamical system (Williams et al., 2015; Johnson & Yeung, 2018). These approaches have been largely successful, but they require the user to define the basis functions.

The third category contains approaches that use genetic algorithms to produce mathematical relationships that explain data from dynamical systems (Iba et al., 1995; J. Gray et al., 1998; Rodriguez et al., 2004; Madár et al., 2005). These methods use genetic programming to evolve an expression tree until a mathematical relationship is discovered that matches the input data. One challenge of these methods is the goal of trying to find the entirety of a relationship in a single expression. Most dynamical systems have individual additive terms (or processes) that contribute to the dynamics of the system. It is challenging for a genetic algorithm to correctly assemble the right set of features when the relationship has many terms. These methods have also not been extended to the challenging domains of PDE identification and system linearization.

This work describes an approach for system identification that produces interpretable models from data with minimal user input and domain knowledge. The user specifies a small set of basic rules from which possible mathematical descriptions can be generated. Then, an optimization technique is applied to make searching such a vast space of possibilities more tractable, seeking the simplest expressions that can explain the supplied data. Our technique combines the ideas of sparse regression with genetic algorithms to create a hybrid system that has benefits from both areas. Relaxing the requirement to explicitly enumerate possible expressions, allows this new approach to tackle the challenging problem of deriving approximate Koopman eigenfunctions for the linearization of nonlinear systems.

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 take in data and find a parsimonious description of a dynamical system. This section outlines two goals for system identification: (1) discovery of governing equations expressed as PDEs, and (2) the generation of approximate Koopman eigenfunctions for the linearization of nonlinear systems.

2.1. 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 equations can be written as PDEs of the form

$$\frac{\partial \mathbf{x}(\mathbf{r},t)}{\partial t} = \theta_1 \mathbf{g}_1(\mathbf{x}(\mathbf{r},t)) + \ldots + \theta_n \mathbf{g}_n(\mathbf{x}(\mathbf{r},t)) \qquad (1)$$

where $\mathbf{r}$ represents the spatial dimensions, $\mathbf{x}$ contains the state variables, $\mathbf{g}_i$ are feature functions, and θ_i are the corresponding coefficients. Given a mapping $\mathbf{g}_i$, and a state $\mathbf{x}$, the coefficients of the PDE can be found using a linear model of the form

$$\mathbf{y} = \boldsymbol{\theta}^T \tilde{\mathbf{x}} \qquad (2)$$

where $\mathbf{y} = \partial \mathbf{x}/\partial t$ and $\tilde{\mathbf{x}} = \left[\mathbf{g}_1(\mathbf{x}), \ldots, \mathbf{g}_n(\mathbf{x})\right]^T$. An approximation of $\boldsymbol{\theta}$ can be found by sampling many $\mathbf{y}$ and $\tilde{\mathbf{x}}$ points and then performing multiple linear regression. The goal is to find the optimal set of features $\mathbf{g}_i$ (and corresponding θ_i) for producing the best fit between $\mathbf{y}$ and $\tilde{\mathbf{x}}$.

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.

2.2. Koopman Operators

Nonlinear dynamical systems are far more challenging to understand and control than their linear counterparts. One promising area of analysis is Koopman theory (Korda & Mezić, 2018) which states that any nonlinear dynamical system can be converted into a linear system under the state-space mapping

$$\mathbf{x} \mapsto \mathbf{g}(\mathbf{x}) \qquad (3)$$

where $\mathbf{g}(x) = \left[\mathbf{g}_1(x), \mathbf{g}_2(x), \ldots\right]^T$ are the Koopman eigenfunctions. In general, $\mathbf{g}$ can be infinite-dimensional and is often approximated with a finite-dimensional mapping.

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

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

and transform it into the linear system

$$\mathbf{g}(\mathbf{x})^{t+1} = \mathbf{K}\mathbf{g}(\mathbf{x})^t \qquad (5)$$

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 $\mathbf{g}(\mathbf{x}) = \mathbf{x}$ 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 (eq. (5)), without finding an inverse transformation $\mathbf{g}^{-1}$.

Given a finite approximation to the Koopman eigenfunctions $\mathbf{g}(\mathbf{x}) = \left[\mathbf{g}_1(\mathbf{x}), \ldots, \mathbf{g}_n\right]^T$, the matrix $\mathbf{K}$ can be found using a linear model of the form

$$\mathbf{y} = \mathbf{K}\tilde{\mathbf{x}} \tag{6}$$

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

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 an approach to finding these nonlinear mappings.

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

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

3.1. Context-Free Grammar

There are an infinite number of possible mathematical expressions that could be included in a mapping, but researchers do not generally know beforehand which specific expressions will be good candidates. A design goal of SIEO, therefore, is to require minimal prior knowledge from the user while still constraining the possible expressions to be searched.

SIEO relies on a domain-specific *grammar*, provided by the user, that gives the rules for generating feature functions (Kochenderfer & Wheeler, 2019). A grammar is a set of production rules that govern a language (or a set of expressions). Each rule can either be *non-terminal*, where it relates generic expressions together (e.g. multiplication), or *terminal*, where 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 fig. 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

$$u\frac{\partial u}{\partial x} \tag{7}$$

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. The set can be made finite by specifying a maximum expression depth for a grammar.

The benefit of this approach is two-fold:

1. For the cost of defining a simple grammar, an infinite number of possible expressions (ranging from simple to complex) can be produced.
2. Researchers can encode into the grammar any information they 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 mappings.

3.2. Choice of Loss Function

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

$$R^2_{\text{adj}} = R^2 - \alpha(1 - R^2)(n - 1) \tag{8}$$

where n is the number of features in the mapping. When $n = 1$ this expression reduces to the traditional R^2 value, but when $n > 1$ the difference between R^2 and 1 is scaled by the number of parameters. Thus, generally, $\alpha \in [0, \infty)$ should be large when agreement between the input and output (i.e. for synthetic, noise-free data) is expected 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

$$\ell_R(\theta, \mathbf{g}) = -R^2_{\text{adj}}(\boldsymbol{\theta}, \mathbf{g}) \tag{9}$$

For the goal of discovering Koopman eigenfunctions, however, parsimony is no longer a consideration so a new loss function is needed. Given a linear system with n_g Koopman eigenfunctions, the loss function associated with the Koopman operator $\mathbf{Kg}$ is

$$\ell_k(\mathbf{Kg}) = \frac{1}{n_g} \|\overline{\mathbf{g}}(\mathbf{x})^{t+1} - \overline{\mathbf{Kg}}(\mathbf{x})^t\|_2^2 \tag{10}$$

where the bar indicates that each feature is normalized to a zero mean and standard deviation of 1. The loss is scaled by $1/n_g$ so the algorithm is not penalized for adding more Koopman eigenfunctions, and instead focuses on closely matching each new state variable when new features are added. This loss function is referred to as the *average normalized sum of squares* loss and should be used when parsimony is not a consideration. The following subsection ties together the use of a grammar and a loss function to explain how expression optimization and heuristic search are used to select features.

3.3. Feature Selection Algorithms

One major challenge of system identification is the selection of a mapping from a group of possible features. If there are N possible features, then there are 2^N possible subsets from which to choose. It is infeasible to exhaustively search the entire space of combinations for even moderate values of N.

Previous approaches have used sparsity-promoting regression techniques such as LASSO (Tibshirani, 1996), but these approaches are brittle when the input data suffers from multicollinearity (Rudy et al., 2017). It was found that the most effective heuristic 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.

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 in the loss function or a specified threshold is met. Then, the feature selection algorithm performs a greedy backward search (algorithm 3) on the mapping 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.

Algorithm 1 Feature Selection

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

Algorithm 2 Forward Search

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

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, and the one chosen for SIEO is Grammatical Evolution (GE), an expression optimization approach introduced by Ryan et al. (1998). In GE, each feature is converted to an unbounded integer array that can be parsed from left to right into an expression tree. To evaluate the fitness of a feature, it is concatenated to the current mapping and assigned the loss associated with the new mapping. The integer array is then transformed in a manner reminiscent of traditional genetic algorithms. The basic operations are

Algorithm 3 Backward Search

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

1. **Selection**: The integer arrays with the best fitness (lowest value of the 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**: 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 from an individual.

These operations are applied to a large number of expressions over many generations in order to produce optimal features from the large space of possibilities. The feature that maximally reduces the loss of the mapping is returned to the forward search algorithm. In the following section, SIEO is applied to four model systems to demonstrate several 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 by either finding them exactly or by finding a good finite approximation.

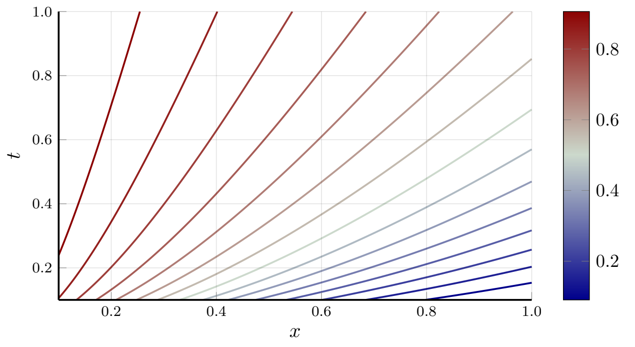


Figure 1. Unsteady data from the 1D advection-diffusion equation

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

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} - v\frac{\partial u}{\partial x} \tag{11}$$

where D is the diffusion coefficient, v is the fluid velocity and x is the spatial dimension. Synthetic data is generated using the exact solution derived by Van Genuchten (1982), which, for boundary conditions

$$u(x,0) = u_0 \tag{12}$$
$$u(0,t) = u_L \tag{13}$$
$$\frac{\partial u}{\partial x}(\infty, t) = 0 \tag{14}$$

is found to be

$$\begin{aligned} u(x,t) = u_0 + \frac{1}{2}(u_L - u_0)\Bigg(&\mathrm{erfc}\left[\frac{x - vt}{2\sqrt{Dt}}\right] \\ &+ \exp(vx/D)\mathrm{erfc}\left[\frac{x+vt}{2\sqrt{Dt}}\right]\Bigg) \end{aligned} \tag{15}$$

This solution is plotted in fig. 1 for $x > 0$, $t > 0$, $D = 1$, $v = 0.5$, 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 fig. 2 was used for the expression optimization routine and was limited to a tree depth of 3.

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

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%	49.08 %	19.02 %

The adjusted R^2 loss function was used with the parameter $\alpha = 0.5$ and the solution was denoised before and after the numerical differentiation using total variation filtering (Rudin et al., 1992).

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 of 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.

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

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{16}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \frac{\mu}{\rho}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{17}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \frac{\mu}{\rho}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{18}$$

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.

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 %

Synthetic data was collected from the simulation of vortex shedding behind a circular cylinder at a Reynold's number of 50. The simulation was performed using PyFR (Witherden et al., 2014) 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 fig. 3, with u, v, and p on separate plots. The grammar in fig. 4 was used to produce features up to a depth of 3, yielding 363 features and 2^{363} mappings to be searched.

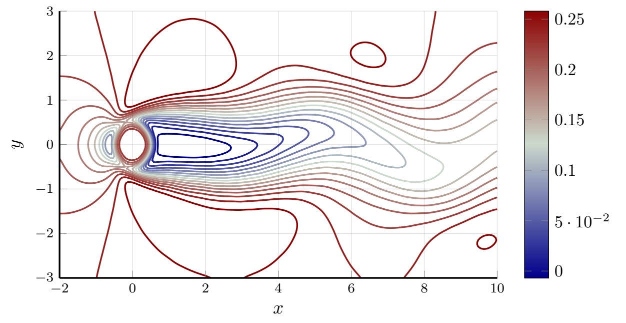
SIEO was performed on the the vortex shedding data with the adjusted R^2 loss function and $\alpha = 0.2$ and it found eqs. (19) and (20) to explain the unsteady Navier-Stokes data

$$\frac{\partial u}{\partial t} = \theta_{x1}\frac{\partial (uv)}{\partial y} + \theta_{x2} v\frac{\partial u}{\partial y} + \theta_{x3}\frac{\partial p}{\partial y} + \theta_{x4}\frac{\partial^2 u}{\partial y^2} \tag{19}$$

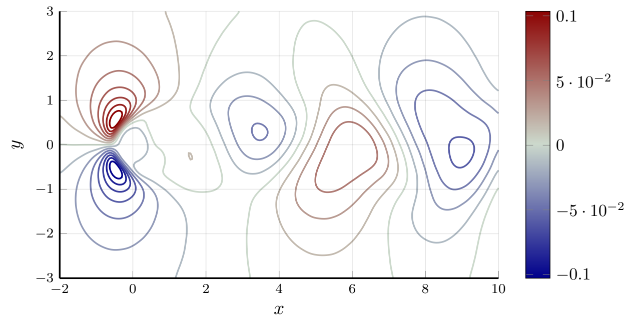
$$\frac{\partial v}{\partial t} = \theta_{y1} u\frac{\partial v}{\partial x} + \theta_{y2} v\frac{\partial v}{\partial y} + \theta_{y3}\frac{\partial p}{\partial y} \tag{20}$$

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. Note also 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 have a sufficiently large signal in the data collected.

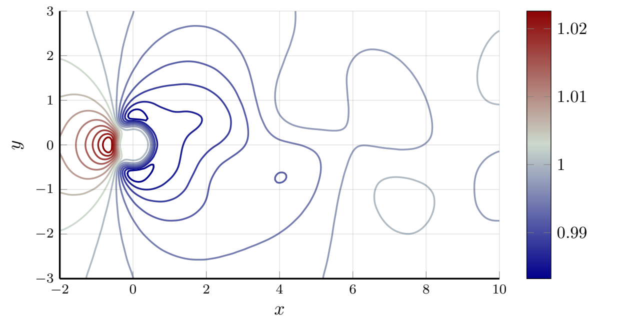
The ratios of the model coefficients were compared to determine the error in the model parameters. The results are shown in table 2. In general, the model parameter error is very small and the discovered equations could be used for prediction or control of other similar types of flow fields (i.e. those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



(a) x-velocity



(b) y-velocity



(c) pressure

Figure 3. One snapshot of vortex-shedding behind a circular cylinder

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

Figure 4. Grammar used for the Navier-Stokes equations

4.3. Exact Koopman Eigenfunction Discovery

A simple nonlinear ODE with finite-dimensional Koopman eigenfunctions that are easily found is given by Kutz et al. (2016) as

$$\frac{dx}{dt} = \mu x \tag{21}$$

$$\frac{dy}{dt} = \lambda(y - x^2) \tag{22}$$

The state-space transformation

$$\begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} x \\ y \\ x^2 \end{bmatrix} \tag{23}$$

linearizes eqs. (21) and (22) to

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ x^2 \end{bmatrix} = \begin{bmatrix} \mu & 0 & 0 \\ 0 & \lambda & -\lambda \\ 0 & 0 & 2\mu \end{bmatrix} \begin{bmatrix} x \\ y \\ x^2 \end{bmatrix} \tag{24}$$

To test the discovery of exact Koopman operators, data from the system (eqs. (21) and (22)) was collected for $\lambda = \mu = 1$ with many initial conditions. The initial data is stored in an vector $\mathbf{x}^0$, then the solution is integrated forward one timestep and the result is stored in an output vector $\mathbf{x}^1$. SIEO was then applied with the average normalized 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 fig. 5 was used as input (exchanging (θ, ω) for (x, y)). The correct Koopman transformation was discovered (eq. (23)) and the Koopman operator (eq. (24)) was recovered to within machine precision after a few seconds of computation running on a single thread of an Intel Core i5 processor.

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 (with parameters normalized to 1) is given by

$$\frac{d^2\theta}{dt^2} + \sin\theta = 0 \tag{25}$$

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

$$\frac{d\theta}{dt} = \omega \tag{26}$$

$$\frac{d\omega}{dt} = -\sin\theta \tag{27}$$

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

Figure 5. Grammar used for both nonlinear ODEs

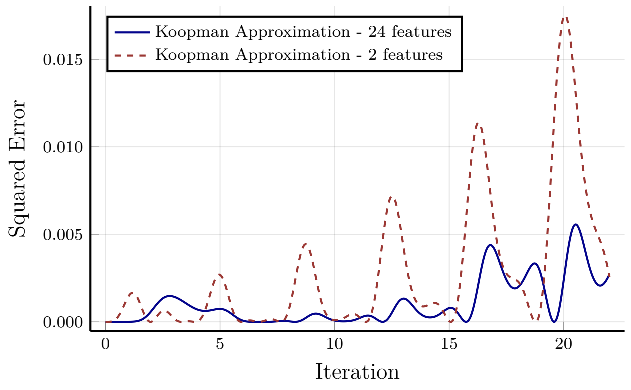


Figure 6. Error in the nonlinear pendulum for the simplest linear model and the linear model with 24 features

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 $T = 10$. The 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 fig. 5 was used as the input.

The resulting Koopman operator approximation 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 fig. 6.

The figure shows that the linear model with more nonlinear features has significantly lower average error than the linear model with only the two state variables. Additionally, the error increases 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 and Future work

From the previous experiments, we conclude that the use of context-free grammars and expression optimization is an effective way of generating mappings that describe observed data. The system is robust to noise and can operate on small amounts of data. Additionally, it is robust to multiple dimensions and variables in the state space. Lastly, it has been demonstrated that expression optimization has the potential to discover exact and approximate analytical Koopman eigenfunctions for the linearization of nonlinear systems.

Further work on this topic will focus on expanding the utility of SIEO by

- Including free-parameter optimization into the selection of expressions.
- Investigating which grammars perform best for equation discovery and Koopman eigenfunction approximation.
- Finding the Koopman eigenfunction approximations for larger dynamical systems and using the resulting models for control.

References

Bridewell, W., Langley, P., Todorovski, L., and Džeroski, S. Inductive process modeling. *Machine learning*, 71(1): 1–32, 2008.

Brunton, S. L., Proctor, J. L., and Kutz, J. N. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, pp. 201517384, 2016.

Iba, H., deGaris, H., and Sato, T. A numerical approach to genetic programming for system identification. *Evolutionary Computation*, 3(4):417–452, 1995.

J. Gray, G., Murray-Smith, D., Li, Y., Sharman, K., and Weinbrenner, T. Nonlinear model structure identification using genetic programming. *Control Engineering Practice*, 6:1341–1352, 11 1998.

Johnson, C. A. and Yeung, E. A class of logistic functions for approximating state-inclusive koopman operators. In *2018 Annual American Control Conference (ACC)*, pp. 4803–4810. IEEE, 2018.

Kaiser, E., Kutz, J. N., and Brunton, S. L. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proceedings of the Royal Society A*, 474(2219), 2018.

Kochenderfer, M. and Wheeler, T. *Algorithms for Optimization*. MIT Press, 2019.

Korda, M. and Mezić, I. Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. *Automatica*, 93:149–160, 2018.

Krizman, V., Dzeroski, S., and Kompare, B. Discovering dynamics from measured data. *Electrotechnical Review*, 62(3-4):191–198, 1995.

Kutz, J. N., Brunton, S. L., Brunton, B. W., and Proctor, J. L. *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems*, volume 149. SIAM, 2016.

Langley, P. and Arvay, A. Heuristic induction of rate-based process models. In *AAAI Conference on Artificial Intelligence*, pp. 537–543, 2015.

Madár, J., Abonyi, J., and Szeifert, F. Genetic programming for the identification of nonlinear inputoutput models. *Industrial & Engineering Chemistry Research*, 44(9): 3178–3186, 2005.

Morton, J., Jameson, A., Kochenderfer, M. J., and Witherden, F. Deep dynamical modeling and control of unsteady fluid flows. In *Advances in Neural Information Processing Systems 31*, pp. 9278–9288. Curran Associates, Inc., 2018.

Rodriguez, K., Fonseca, C., and Fleming, P. Identifying the structure of nonlinear dynamic systems using multiobjective genetic programming. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 34: 531 – 545, 08 2004.

Rudin, L. I., Osher, S., and Fatemi, E. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268, 1992.

Rudy, S. H., Brunton, S. L., Proctor, J. L., and Kutz, J. N. Data-driven discovery of partial differential equations. *Science Advances*, 3(4):e1602614, 2017.

Ryan, C., Collins, J. J., and Neill, M. O. Grammatical evolution: Evolving programs for an arbitrary language. In *European Conference on Genetic Programming*, pp. 83–96. Springer, 1998.

Schmid, P. J. Dynamic mode decomposition of numerical and experimental data. *Journal of fluid mechanics*, 656: 5–28, 2010.

Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

Van Genuchten, M. T. *Analytical Solutions of the One-Dimensional Convective-Dispersive Solute Transport Equation*. Number 1661. US Department of Agriculture, Agricultural Research Service, 1982.

Williams, M. O., Kevrekidis, I. G., and Rowley, C. W. A data-driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346, 2015.

Witherden, F. D., Farrington, A. M., and Vincent, P. E. Pyfr: An open source framework for solving advection-diffusion type problems on streaming architectures using the flux reconstruction approach. *Computer Physics Communications*, 185(11):3028–3040, 2014.