



RESEARCH ARTICLE | SEPTEMBER 18 2023

## Recent advances in the SISO method and their implementation in the SISO++ code

Special Collection: [Software for Atomistic Machine Learning](#)

Thomas A. R. Purcell ; Matthias Scheffler ; Luca M. Ghiringhelli  

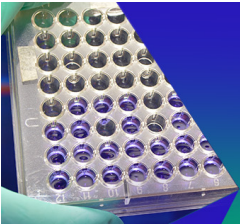
 Check for updates

*J. Chem. Phys.* 159, 114110 (2023)

<https://doi.org/10.1063/5.0156620>





CrossMark



### Biomicrofluidics

Special Topic:  
Microfluidics and Nanofluidics in **India**

**Submit Today**



# Recent advances in the SISO method and their implementation in the SISO++ code

Cite as: J. Chem. Phys. 159, 114110 (2023); doi: 10.1063/5.0156620

Submitted: 1 May 2023 • Accepted: 21 August 2023 •

Published Online: 18 September 2023



View Online



Export Citation



CrossMark

Thomas A. R. Purcell,<sup>1,a)</sup> Matthias Scheffler,<sup>1</sup> and Luca M. Chiringhelli<sup>1,2,b)</sup>

## AFFILIATIONS

<sup>1</sup>The NOMAD Laboratory at the FHI of the Max-Planck-Gesellschaft and IRIS-Adlershof of the Humboldt-Universität zu Berlin, Faradayweg 4–6, D-14195 Berlin, Germany

<sup>2</sup>Physics Department and IRIS-Adlershof, Humboldt Universität zu Berlin, Zum Großen Windkanal 2, D-12489 Berlin, Germany

**Note:** This paper is part of the JCP Special Topic on Software for Atomistic Machine Learning.

<sup>a)</sup>Electronic mail: [purcell@fhi-berlin.mpg.de](mailto:purcell@fhi-berlin.mpg.de)

<sup>b)</sup>Author to whom correspondence should be addressed: [ghiringhelli@fhi-berlin.mpg.de](mailto:ghiringhelli@fhi-berlin.mpg.de)

## ABSTRACT

Accurate and explainable artificial-intelligence (AI) models are promising tools for accelerating the discovery of new materials. Recently, symbolic regression has become an increasingly popular tool for explainable AI because it yields models that are relatively simple analytical descriptions of target properties. Due to its deterministic nature, the sure-independence screening and sparsifying operator (SISO) method is a particularly promising approach for this application. Here, we describe the new advancements of the SISO algorithm, as implemented into SISO++, a C++ code with Python bindings. We introduce a new representation of the mathematical expressions found by SISO. This is a first step toward introducing “grammar” rules into the feature creation step. Importantly, by introducing a controlled nonlinear optimization to the feature creation step, we expand the range of possible descriptors found by the methodology. Finally, we introduce refinements to the solver algorithms for both regression and classification, which drastically increase the reliability and efficiency of SISO. For all these improvements to the basic SISO algorithm, we not only illustrate their potential impact but also fully detail how they operate both mathematically and computationally.

Published under an exclusive license by AIP Publishing. <https://doi.org/10.1063/5.0156620>

## I. INTRODUCTION

Data-centric and artificial-intelligence (AI) approaches are becoming a vital tool for describing physical and chemical properties and processes. The key advantage of AI is its ability to find correlations between different sets of properties without the need to know which ones are important before the analysis. Because of this, AI has become increasingly popular for materials discovery applications with uses in areas such as thermal transport properties,<sup>1,2</sup> catalysis,<sup>3</sup> and quantum materials.<sup>4</sup> Despite the success of these methodologies, creating explainable and physically relevant AI models remains an open challenge in the field.<sup>5–7</sup>

One prevalent set of methods for explainable AI is symbolic regression.<sup>8–11</sup> Symbolic regression algorithms identify optimal nonlinear, analytic expressions for a given target property from a set of input features, i.e., the *primary features*, that are related to the target.<sup>12</sup> Originally, (stochastic) genetic-programming-based approaches were and still are used to find these

expressions,<sup>12–15</sup> but recently, a more diverse set of solvers have been developed.<sup>16–21</sup> The sure-independence screening and sparsifying operator (SISO) approach combines symbolic regression with compressed sensing<sup>22–25</sup> to provide a deterministic way of finding these analytic expressions. This approach has been used to describe numerous properties, including phase stability,<sup>23,26,27</sup> catalysis,<sup>28</sup> and glass transition temperatures.<sup>29</sup> It has also been used in a multi-task<sup>22</sup> and hierarchical fashion.<sup>25</sup>

In this paper, we introduce the new concepts implemented in the recently released SISO++ code<sup>24</sup> and detail their implementation. SISO++ is a new, modular implementation of SISO that provides a more user-friendly interface to run SISO with several methodological updates. Before detailing the updated methodology, we summarize the main aspects of the SISO algorithm and define some of the terminologies we use below. The SISO approach starts with a collection of *primary features* and mathematical unary and binary operators (e.g., addition, multiplication,  $n^{\text{th}}$  root, and logarithms). The first step is the *feature-creation* step, where a pool

of *generated features* is built by exhaustively applying the set of mathematical operators onto the primary features. The algorithm is iteratively repeated by applying the set of operators onto the previously generated features. The number of iterations in this feature-creation step is called the *run*. The generated features, projected onto the training dataset, build the so-called sensing matrix  $\mathbf{D}$ , with one column for each generated feature and one row for each data point.

The subsequent step is the *descriptor identification* step, i.e., compressed sensing is used to identify the best  $D$ -dimensional linear model by performing an  $\ell_0$ -regularized optimization on a subspace  $\mathcal{S}$  of all generated features.  $\mathcal{S}$  is selected using sure-independence screening<sup>30</sup> (SIS), with a suitable projection score, depending on the loss function used to evaluate the models. For a regression problem (see further below for the discussion of classification problems), i.e., the prediction of a continuous property  $\mathbf{P}$  (an array with as many components as the number of training points), one solves  $\text{argmin}_{\mathbf{c}} \|\mathbf{D}\mathbf{c} - \mathbf{P}\|_2 + \lambda \|\mathbf{c}\|_0$ , where  $\|\mathbf{c}\|_0$  is the number of nonzero components in  $\mathbf{c}$ , which we call the dimensionality  $D$  of the model  $\mathbf{D}\mathbf{c}$ . The hyperparameter  $\lambda$  is fixed by cross-validation, i.e., by minimizing the validation error over the values of  $D$ . Within the SISO algorithm, the SIS step works iteratively by ranking all generated features according to their Pearson correlation values to the target property and adding only the most correlated features to  $\mathcal{S}$ . At the next iteration, SIS ranks and adds to  $\mathcal{S}$  the features that Pearson correlate the most with the *residual* of the previous step,  $\Delta_{D-1}$ , i.e., the difference between  $\mathbf{P}$ , and the estimates predicted by the  $(D-1)$ -dimensional model:  $\Delta_{D-1} = \mathbf{D}\mathbf{c}_{D-1} - \mathbf{P}$ . The SIS step is followed by the sparsifying operator (SO) step, where linear-regression models are trained on all subsets of features in  $\mathcal{S}$ , i.e., on all single features, then all pairs, then all triplets, etc. This yields a set of models of dimensions  $D = 1, 2, 3$ , etc. For each dimension, models are ranked by training error and the best model is selected. The model with the lowest validation error across dimensions is the final SISO model. In practice, models at increasing dimension are trained until the validation error starts increasing or reaches a plateau. In summary, the result of the SISO analysis is a  $D$ -dimensional descriptor, which is a vector with components from  $\mathcal{S}$ . For a regression problem, the SISO model is the scalar product of the identified descriptor with the vector of linear coefficients resulting from the  $\ell_0$ -regularized linear regression.

For a classification problem, the SISO model is given as a set of hyperplanes that divide the data points into classes, which are described by the scalar product of the identified descriptor, with a set of coefficients found by linear support vector machines (SVMs). The loss function for classification is based on a convex-hull algorithm, where the convex hulls for each set of points with the same property label, i.e., the same class, are evaluated, and the algorithm tries to minimize the number of data points inside more than one convex hull, i.e., the number of points inside the overlap region. The SIS step ranks features by how well an individual feature separates the remaining unclassified data points by first minimizing the number of misclassified points and then minimizing the hypervolume of the overlap region where they are located. When a feature leaves no misclassified points, the tiebreaker maximizes the minimum separation distance between points in different classes. The residual here is the set of all misclassified points from the  $D-1$ -dimensional descriptor. The SO uses a convex-hull based loss function, i.e., minimizes the

number of points inside the overlap region of a set of  $D$ -dimensional convex hulls.

This paper is organized as follows. We separately describe the recent updates to the SISO methodology for the *feature creation* (Sec. II) and *descriptor identification* steps (Sec. III). The most important advancement of the code is expressing the features as binary expression trees (Subsection II A), instead of strings, allowing us to recursively define all aspects of the generated features from the primary features. With this implementation choice, we are able to keep track of the units for each generated feature, as well as an initial representation of its domain and range. This allows for the creation of grammatically correct expressions, in terms of consistency of the physical units, and the control of numerical issues generated by features going out of their physically meaningful range. In terms of the feature-creation step, we also discuss the implementation of *parametric SISO* (Subsection II B), which introduces the flexibility of nonlinear parameters together with the operators that are optimized against a loss function based on the compressed-sensing-based feature-selection metrics. This procedure was used to describe the thermal conductivity of a material in a recent publication.<sup>31</sup> For the descriptor-identification step, we cover two components: an improved classification algorithm and the multi-residual approach. For classification problems, we generalized the algorithm to work for any problem to an arbitrary dimension and explicitly include a model identification via linear SVM (Subsection III A). The multi-residual approach (Subsection III B), which was previously used in Ref. 25, introduces further flexibility for the identification of models with more than one dimension. Here, we provide an in-depth discussion of its machinery.

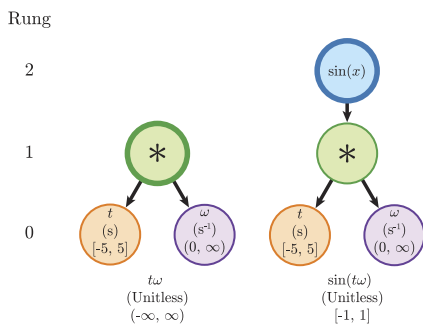
## II. FEATURE CREATION

### A. Binary-expression-tree representation of features

The biggest advancement to the implementation of SISO in SISO++, compared to the original implementation in Ref. 23, is its modified representation of the features as binary expression trees, instead of strings. While the choice of feature representation does not affect the methodology and could be used in any implementation, it does impact the performance and readability of the code base. This representation is illustrated in Fig. 1 and easily allows for all aspects of the generated features to be recursively calculated on the fly from the data stored in the primary features. For certain applications, it is also possible to store the data of higher-rung features to reduce the overall computational cost of the calculations. The individual features are addressed by the root node of the binary expression tree and stored in the code as a `SHARED_PTR` from the C++ standard library. This representation reduces the overall memory footprint of each calculation as the individual features only need to be created once and only copies of shared pointers need to be stored for each new expression. The remainder of this section will be used to describe the various aspects of the new representation, including a description of the units and range of the features, as well as how it is used to generate the feature space.

#### 1. Units

An important upgrade in SISO++ is its generalized and exact treatment of units for the expressions. In physics, dimensional analysis is an important tool when generating physically meaningful



**FIG. 1.** A demonstration of the new representation of the features in the SISSO++ code. The feature is stored as the root of the tree (represented by the thick border), the primary features are the leaves, and the rung corresponds to the height of the tree, i.e., the longest path between each leaf and the root. The unit, range, expressions, and values of the features are necessarily stored only for the primary features, with them defined recursively for all generated features.

expressions, and it is necessary to include it when using symbolic regression for scientific problems. We introduced this into SISSO++ by determining the units for each new expression from the primary features and explicitly checking to ensure that a new expression is physically possible. Within the code, the UNITS are implemented as dictionaries with the key representing the base unit and the value representing the exponent for each key, e.g.,  $m/s^2$  would be stored as  $\{m : 1, s : -2\}$ . Functions exist to transform the UNITS to and from strings to more easily represent the information. We then implemented a multiplication, division, and power operators for these specialized dictionaries, allowing for the units of the generated features to be derived recursively following the rules in Table I. An important caveat is that the current implementation cannot convert between two units for the same physical quantity, e.g., between nanometers, picometers, and Bohr radii for length.

Using this implementation of units, a minimal treatment of dimensional analysis can be performed in the code. The dimensional

**TABLE I.** How the units are calculated for each operation.

Operation	Resulting unit
$A + B$	Unit (A)
$A - B$	Unit (A)
$A * B$	Unit (A) * unit (B)
$A/B$	Unit (A)/unit (B)
$ A - B $	Unit (A)
$ A $	Unit (A)
$\sin(A)$	Unitless
$\cos(A)$	Unitless
$\exp(A)$	Unitless
$\exp(-A)$	Unitless
$\log(A)$	Unitless
$(A)^{-1}$	Unit (A) <sup>-1</sup>
$(A)^2$	Unit (A) <sup>2</sup>
$(A)^3$	Unit (A) <sup>3</sup>
$(A)^6$	Unit (A) <sup>6</sup>
$\sqrt{A}$	Unit (A) <sup>1/2</sup>
$\sqrt[3]{A}$	Unit (A) <sup>1/3</sup>

**TABLE II.** Restrictions for each unit; if an operation is not listed, there are no restrictions.

Operation	Unit restriction
$A + B$	Unit (A) = = unit (B)
$A - B$	Unit (A) = = unit (B)
$ A - B $	Unit (A) = = unit (B)
$\sin(A)$	Unit (A) = = $\emptyset$
$\cos(A)$	Unit (A) = = $\emptyset$
$\exp(A)$	Unit (A) = = $\emptyset$
$\exp(-A)$	Unit (A) = = $\emptyset$
$\log(A)$	Unit (A) = = $\emptyset$

analysis focuses on whether the units are consistent within each expression and for the final model. This check is used to determine the units of the fitted constants in the linear models at the end, which can take arbitrary units and therefore, can do any unit conversion natively. The restrictions, used to reject expressions by dimensional analysis, are outlined in Table II and can be summarized as follows: Addition and subtraction are only allowed to act on features of the same units, and all transcendental operations must act on a unitless quantity. With these two restrictions in place, only physically relevant features can be found, and the choice of units should no longer affect which features are selected. If one wants to revert back to previous descriptions without these restrictions, this can be achieved by providing all primary features with no units associated with them.

## 2. Range

Another important advancement of the feature-creation step is the introduction of ranges for the primary features, which act as a domain for future operations during feature creation. One of the challenges associated with symbolic regression, especially with smaller datasets, is that the selected expressions can sometimes contain discontinuities that are outside of the training data, but still within the relevant input space for a given problem. For example, this can lead to an expression taking the logarithm of a negative number, resulting in an undefined prediction. SISSO++ solves this problem by including an option for describing the range of a primary feature in a standard mathematical notation, e.g.,  $[0, \infty)$ , and then using that to calculate the range for all generated formulas using that primary feature, following the rule specified in Table III. In the code, the ranges are referenced as the DOMAIN because the range for a feature of rung  $n - 1$  is the domain for a possible expression of rung  $n$  that is using that feature. While all ranges in Table III assume inclusive endpoints, the implementation can handle both exclusive endpoints and a list of values explicitly excluded from the range, e.g., point discontinuities inside the primary features themselves.

Table IV lists the cases where the range of a feature is used to prevent a new expression from being generated. In all cases, this prevents an operation from occurring where a mathematical operation would be not defined, such as taking the square-root of a negative number. In cases where the range of values for a primary feature is not defined, then these checks are not performed and the original assumption that all operations are safe is used.

## B. Parametric SISSO

Parametric SISSO extends the feature creation step of SISSO to automatically include scale and bias terms for each operation, as

TABLE III. How the range for each operation is calculated.

Operation	Resulting range
$A + B$	$[\min(A) + \min(B), \max(A) + \max(B)]$
$A - B$	$[\min(A) - \max(B), \max(A) - \min(B)]$
$A * B$	$[\min(\min(A) * \min(B), \min(A) * \max(B), \max(A) * \min(B), \max(A) * \max(B)), \max(\min(A) * \min(B), \min(A) * \max(B), \max(A) * \min(B), \max(A) * \max(B))]$
$A/B$	$[\text{Range}(A) * \text{Range}(B^{-1})]$
$\sin(A)$	$[-1, 1]$
$\cos(A)$	$[-1, 1]$
$\exp(A)$	$[\exp(\min(A)), \exp(\max(A))]$
$\exp(-A)$	$[\exp(-\max(A)), \exp(-\min(A))]$
$\log(A)$	$[\log(\min(A)), \log(\max(A))]$
$(A)^{-1}$	if $(0 \in \text{Range}(A))$ : and $\min(A) \neq 0$ and $\max(A) \neq 0$ : $(-\infty, 0) \cup (0, \infty)$ else if $(0 \in \text{Range}(A))$ and $\min(A) == 0$ : $(0, \infty)$ else if $(0 \in \text{Range}(A))$ and $\max(A) == 0$ : $(-\infty, 0)$ else: $[(\max(A))^{-1}, (\min(A))^{-1}]$
$(A)^3$	$[(\min(A))^3, (\max(A))^3]$
$\sqrt{A}$	$[\sqrt{\min(A)}, \sqrt{\max(A)}]$
$\sqrt[3]{A}$	$[\sqrt[3]{\min(A)}, \sqrt[3]{\max(A)}]$
$ A $	$[\max(0, \min(A)), \max( \max(A) ,  \min(A) )]$
$(A)^2$	$[\max(0, \min(A))^2, \max( \max(A) ,  \min(A) )^2]$
$(A)^6$	$[\max(0, \min(A))^6, \max( \max(A) ,  \min(A) )^6]$
$ A - B $	$[\max(0, \min(A) - \max(B)), \max( \max(\max(A) - \min(B)) ,  \min(\max(A) - \min(B)) )]$

TABLE IV. Domain restrictions for each operation; if an operation is not listed, there are no restrictions.

Operation	Domain restriction
$A/B$	$0 \notin \text{Range}(B)$
$(A)^{-1}$	$0 \notin \text{Range}(A)$
$\log(A)$	$\min(\text{Range}(A)) > 0$
$\sqrt{A}$	$\min(\text{Range}(A)) \geq 0$

used by Purcell *et al.*<sup>31</sup> For a general operator,  $\hat{h}(\mathbf{x}) \in \hat{\mathcal{H}}$ , with a set of scale and bias parameters,  $\hat{\mathcal{P}}$ , the parameterization scheme updates the operator to be

$$\hat{h}(\mathbf{x}) \rightarrow \hat{h}^{\hat{\mathcal{P}}}(\alpha\mathbf{x} + \beta), \quad (1)$$

where  $\alpha$  is the scale parameter,  $\beta$  is the bias term, and  $\mathbf{x}$  is a vector containing all input data. For binary operators, the scale and shift parameters for both input features can be set, leading to

$$\hat{h}(\mathbf{x}_0, \mathbf{x}_1) \rightarrow \hat{h}(\alpha_0\mathbf{x}_0 + \beta_0, \alpha_1\mathbf{x}_1 + \beta_1). \quad (2)$$

These new operators can then be used to create a new feature,  $\hat{\phi}^{\hat{\mathcal{P}}}(\mathbf{x})$ , as is normally done in SISSO, where each feature has its own set of parameters stored as a vector. However, it is important to note that when the operations are combined or incorporated into a linear model, as is done in SISSO, some of the parameters will become linearly dependent on each other. For example, the multiplication operation would be defined as

$$(\alpha_0\mathbf{x}_0 + \beta_0)(\alpha_1\mathbf{x}_1 + \beta_1), \quad (3)$$

which expands to

$$\alpha_0\alpha_1\mathbf{x}_0\mathbf{x}_1 + \alpha_0\beta_1\mathbf{x}_0 + \alpha_1\beta_0\mathbf{x}_1 + \beta_0\beta_1. \quad (4)$$

By factoring out  $\alpha_0\alpha_1$  from the expression and defining  $\beta'_0 = \frac{\beta_0}{\alpha_1}$  and  $\beta'_1 = \frac{\beta_1}{\alpha_0}$ , we can rewrite this expression to be

$$\alpha_0\alpha_1(\mathbf{x}_0\mathbf{x}_1 + \beta'_1\mathbf{x}_0 + \beta'_0\mathbf{x}_1 + \beta'_0\beta'_1). \quad (5)$$

**TABLE V.** The free parameters and updated equations for each operation used in the updated feature creation step of SISO. For each operation, parameter restrictions are shown in the third column.

Operation	Parameterized expressions	Fixed parameters
$A + B$	$A + \alpha_1 B$	$\beta_0 = 0; \beta_1 = 0; \alpha_0 = 1$
$A - B$	$A - \alpha_1 B$	$\beta_0 = 0; \beta_1 = 0; \alpha_0 = 1$
$A * B$	$(A + \beta_0) * (B + \beta_1)$	$\alpha_0 = 1; \alpha_1 = 1$
$A/B$	$(A + \beta_0)/(B + \beta_1)$	$\alpha_0 = 1; \alpha_1 = 1$
$ A - B $	$ A - (\alpha_1 B + \beta_1) $	$\beta_0 = 0; \alpha_0 = 1$
$ A $	$ A + \beta $	$\alpha = 1$
$\sin(A)$	$\sin(\alpha A + \beta)$	
$\cos(A)$	$\cos(\alpha A + \beta)$	
$\exp(A)$	$\exp(\alpha A)$	$\beta = 0; \alpha > 0$
$\exp(-A)$	$\exp(-\alpha A)$	$\beta = 0; \alpha > 0$
$\log(A)$	$\log(\alpha A + \beta)$	$\alpha = \pm 1$
$(A)^{-1}$	$(A + \beta)^{-1}$	$\alpha = 1$
$(A)^2$	$(A + \beta)^2$	$\alpha = 1$
$(A)^3$	$(A + \beta)^3$	$\alpha = 1$
$(A)^6$	$(A + \beta)^6$	$\alpha = 1$
$\sqrt{A}$	$\sqrt{\alpha A + \beta}$	$\alpha = \pm 1$
$\sqrt[3]{A}$	$\sqrt[3]{A + \beta}$	$\alpha = 1$

This expression can then be refactored, resulting in

$$\alpha_0 \alpha_1 (\mathbf{x}_0 + \beta'_0)(\mathbf{x}_1 + \beta'_1), \quad (6)$$

with the  $\alpha_0 \alpha_1$  term getting set by either the linear model or an operator further up the tree. For the log operator,  $\alpha$  is always set to  $\pm 1$  to also avoid these linear dependencies as

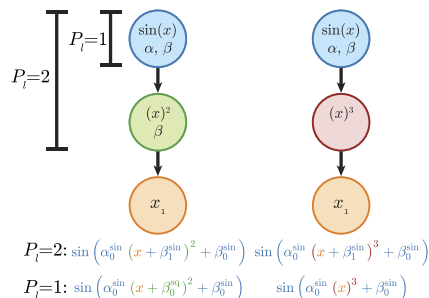
$$\log(\pm|\alpha|\mathbf{x} + \beta) = \log\left(\pm\mathbf{x} + \frac{\beta}{|\alpha|}\right) + \log(|\alpha|). \quad (7)$$

Although this does leave a unit dependency, it can be removed with

$$\ln(\pm\mathbf{x} + \beta) \rightarrow \ln(\alpha_{\text{unit}}(\pm\mathbf{x} + \beta)) - \ln(\alpha_{\text{unit}}), \quad (8)$$

where  $\alpha_{\text{unit}}$  is the unit conversion factor. Table V defines all the free parameters for each operation in parametric SISO.

This parameterization scheme is created at the root node and is recursively defined for all other operators in the binary expression tree at any level below the root. As a result, we introduce a new hyperparameter, the parameterization level,  $P_l$ , to specify the maximum number of levels that can be included within the parameters. For example, if  $P_l = 1$ , then only the root node and its associated parameters will be optimized, but if  $P_l = 2$ , then the parameters associated with the root node and its children will be optimized. This is best illustrated in Fig. 2, where a parameterized-sine operator (blue node) is acting on a parameterized square expression (left graph/green and orange nodes),  $(x + \beta_0^{\text{sq}})^2$  and an unparameterized cube expression (right graph/red and orange nodes),  $(x)^3$ . In both cases when  $P_l = 1$ , only the scale and shift parameters associated with the sine operator are optimized, with all other previously found or not included parameters fixed to their current values, with the resulting expressions shown in the lower row of Fig. 2. However, if  $P_l = 2$ , then the parameters for the square and cube operations ( $\beta_1^{\text{sin}}$ ) are added to those of the sine operator, and the entire expression is



**FIG. 2.** A graphical representation of the effect of the parameterization level for the case where a new parameterized operator is added to a feature that was (left) or was not (right) previously parameterized. If  $P_l = 1$ , then only the parameters associated with the root node (blue, sine operator),  $\alpha_0^{\text{sin}}$  and  $\beta_0^{\text{sin}}$ , get optimized. If the non-root operations were previously parameterized, then those parameters,  $\beta_0^{\text{sq}}$ , remain fixed. If  $P_l = 2$ , then additional parameters associated with child nodes ( $\beta_1^{\text{sin}}$ ) are added to the set that are to be optimized regardless of if the previous child node had optimized parameters.

optimized. The resulting equations in this case are shown in the upper row of Fig. 2.

Once  $\hat{\phi}^{\hat{\mathcal{P}}}(\mathbf{x})$  is defined, all parameters  $\hat{p} \in \hat{\mathcal{P}}$  are optimized using the nonlinear optimization library NLOpt.<sup>32</sup> We use the Cauchy loss function as the objective for the optimization,

$$\min_{\hat{\mathcal{P}}} f(\mathbf{P}, \hat{\phi}^{\hat{\mathcal{P}}}) \quad (9a)$$

$$f(\mathbf{P}, \hat{\phi}^{\hat{\mathcal{P}}}) = \sum_i^{n_{\text{samp}}} \frac{c^2}{n_{\text{samp}}} \log\left(1 + \left(\frac{P_i - \hat{\phi}^{\hat{\mathcal{P}}}(\mathbf{x}_i)}{c}\right)^2\right), \quad (9b)$$

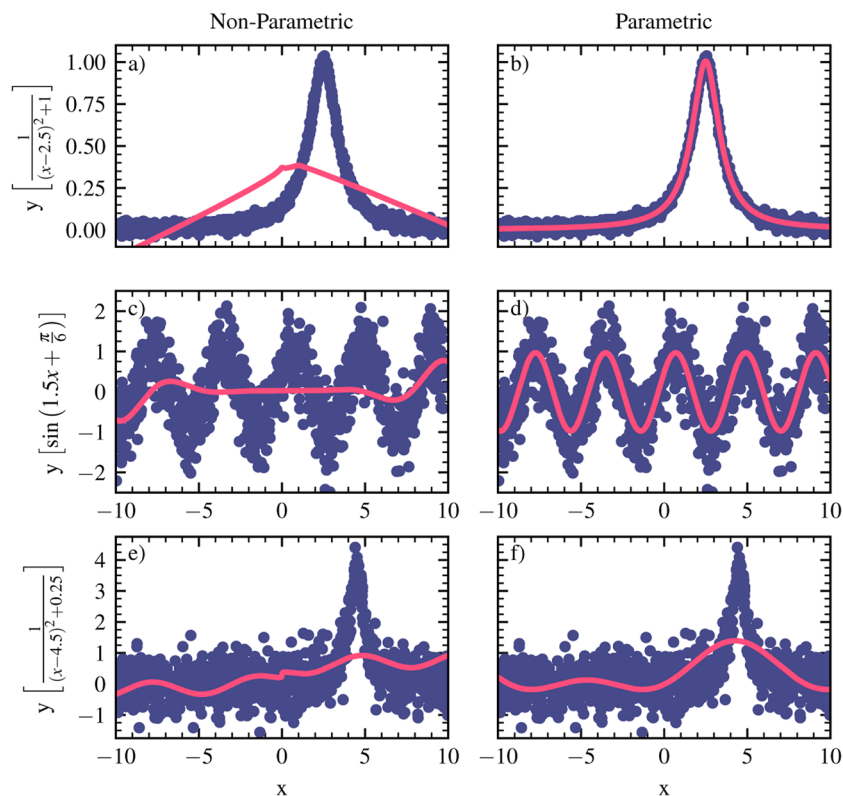
where  $\mathbf{P}$  is a property vector,  $c$  is a scaling factor set to 0.5 for all calculations, and  $n_{\text{samp}}$  is the number of samples. We use the Cauchy loss function over the mean square error to make the nonlinear optimization more robust against outliers in the dataset. Because Eq. (9b) is not scale or bias invariant, additional external parameters  $\alpha_{\text{ext}}$  and  $\beta_{\text{ext}}$  are introduced to, respectively, account for these effects. For the case of multi-task SISO,<sup>22</sup> each task has its own external bias and scale parameters to account for the individual linear-regression solutions. As an example for the features illustrated in Fig. 2 ( $P_l = 2$ ), the functions that are optimized would be

$$\hat{\phi}^{\hat{\mathcal{P}}}(\mathbf{x}) = \alpha_{\text{ext}} \sin(\alpha_0^{\text{sin}}(x + \beta_1^{\text{sin}})^2 + \beta_0^{\text{sin}}) + \beta_{\text{ext}}, \quad (10a)$$

$$\hat{\phi}^{\hat{\mathcal{P}}}(\mathbf{x}) = \alpha_{\text{ext}} \sin(\alpha_0^{\text{sin}}(x + \beta_1^{\text{sin}})^3 + \beta_0^{\text{sin}}) + \beta_{\text{ext}}. \quad (10b)$$

To initialize the parameters in  $\hat{\mathcal{P}}$ , we set all internal  $\alpha$  and  $\beta$  terms to 1.0 and 0.0, respectively, and  $\alpha_{\text{ext}}$  and  $\beta_{\text{ext}}$  are set to the solution of the least squares regression problem for each task. In some cases,  $\beta$  can be set to a nonzero value if leaving it at zero would include values outside the domain of the operator. In these cases,  $\beta$  is set to  $\min(\text{sign}(\alpha)\mathbf{x}) + 10^{-10}$ .

Each optimization follows a two or three step process outlined here. First, a local optimization is performed to find the local minimum associated with the initial parameters. Once, at a local minimum, an optional global optimization is performed to find any minima that are better than the one initially found. For these first two steps, the parameters are optimized to a relative tolerance of  $10^{-3}$  and  $10^{-2}$ , respectively, with a maximum of five thousand



**FIG. 3.** A comparison of the expressions found nonparametric (a), (c), and (e) and parametric (b), (d), and (f) SISO for a Lorentzian (a), (b), (e), and (f) and sine (c) and (d) function. Blue dots represent the training data, and the red line represents the expressions found by SISO. The parameterization scheme either finds the correct (b) and (d) or better (f) model than the nonparametric functions, even when the high noise or bad initial guess of the parameters leads to a nonoptimal solution.

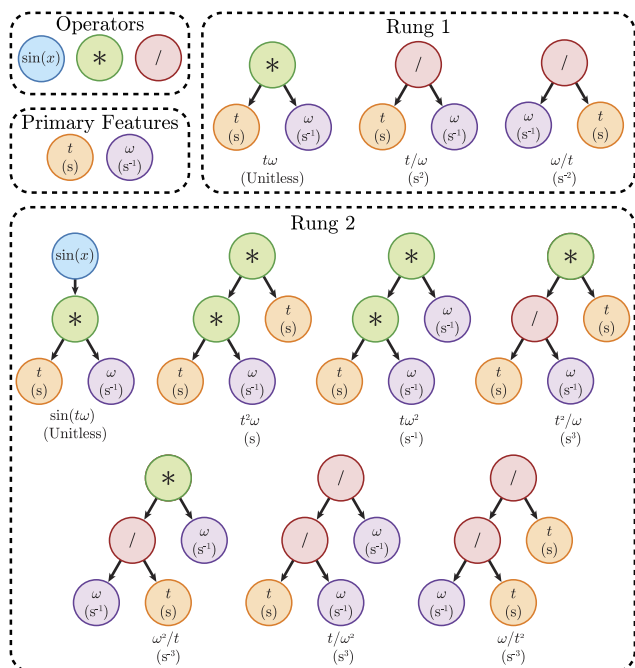
function evaluations for each step. Finally, a more accurate local optimization is done to a relative tolerance of  $10^{-6}$  to find the best parameter set. For this final optimization, ten thousand function evaluations are allowed. Additionally, for both the initial and global optimization steps, the parameters are bounded to be in a range between  $-100$  and  $100$  to improve the efficiency of the optimization, but this restriction is removed for the final optimization. For all local optimizations, the subplex algorithm,<sup>33</sup> a faster and more robust variant of the Nelder–Mead simplex method,<sup>34</sup> is used. The improved stochastic ranking evolution strategy algorithm<sup>35</sup> is used for all global optimizations. Once optimized, only the internal  $\alpha$  and  $\beta$  parameters are stored in  $\hat{\mathcal{P}}$ . This procedure is repeated for all operators specified to be parameterized by the user, so it can increase the cost of feature creation considerably.

Figure 3 illustrates the power of the new parameterization scheme. For both toy problems represented by analytic Lorentzian and sine functions with some white noise, the nonparametric version of SISO cannot find accurate models for the equations as it cannot address the nonlinearities properly. By using this new parameterization scheme, SISO is now able to accurately find the models, as shown in Figs. 3(c) and 3(d). However, it is important to note that the more powerful featurization comes at the cost of a significantly increased time to generate the feature space as the parameterization

becomes the bottleneck for the calculations. Additionally, there can be cases where the parameterization scheme does not find an optimal solution because there is too much noise or the optimal parameters are too far away from the initial guesses, as shown in Figs. 3(e) and 3(f).

### C. Building the complete feature set

With all the new aspects of the feature representation in place, SISO++ has a fully parallelized feature set construction that uses a combination of threads and Message Passing Interface (MPI) ranks for efficient feature set construction. The basic process of creating new features is illustrated in Fig. 4, where each new rung builds on top of existing features by adding a new operation on top of existing binary expression trees for the previous rung. In this step, the operators are separated into parameterized and nonparameterized versions of each other to allow for the optional use of the parametric SISO concepts. Throughout this process, all checks are done to ensure that the units are correct and the domains for each new operation are respected. For example, this is why  $\sin(t)$  and  $\sin(\omega)$  are not included in the rung 1 features as taking the sine of a unit quantity is physically meaningless. If the sine operation was replaced by an algebraic operation or included the parameterization scheme,



**FIG. 4.** Illustration of how the feature space of SISO is created. In this example, the user selects two primary features  $\omega$  (purple) and  $t$  (orange) and three operators  $\sin$  (blue), multiplication (green), and division (red). SISO then builds up a more complicated expression space by applying the operations onto the existing features by increasing the height of the binary expression trees. Throughout this process, the units and ranges of each of the operations are respected.

then two additional rung 1 features would be created with the unary operations acting on each primary feature. Finally, the code checks for invalid values, e.g., NAN or INF, and some basic simplifications for all features, e.g., features such as  $\frac{t\omega}{\omega}$ , are rejected.

### III. DESCRIPTOR IDENTIFICATION

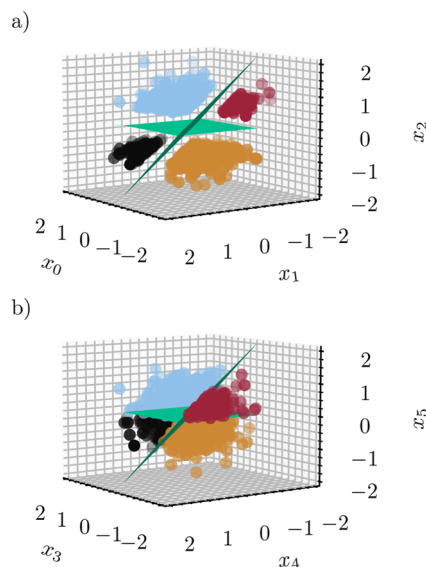
#### A. Linear programming implementation for classification problems

One of the largest updates to the SISO methodology is the new, generalized approach for solving classification problems. In previous implementations, when finding a classification scheme, SISO would explicitly build the convex hull and then calculate the number of points inside the overlap region between different classes and either the normalized overlap volume or separation distance to find the optimal solution. While this works for two dimensions, finding the overlap volume or separation distance becomes intractable for three or more dimensions, and even defining the convex hull becomes intractable for four- or more-dimensional classification. SISO++ replaces these conditions with an algorithm that determines the number of points inside the convex-hull overlap region using linear programming and explicitly creates a model using linear SVM. While for small problem sizes explicitly checking all possible models with linear SVM would be possible, for real-world problems where SISO has to evaluate more than ten million possible combinations of features, this becomes too expensive. To solve this problem,

SISO++ uses a linear-programming algorithm that checks for the feasibility of

$$\begin{aligned} &\min 0 \\ &\text{subject to } \sum_{i \in I} \alpha_i x_i = x_j, \quad \sum_{i \in I} \alpha_i = 1, \quad \alpha_i \geq 0, \quad \forall i \in I, \end{aligned} \quad (11)$$

where  $x_i$  is the  $i^{\text{th}}$  point inside the set of all points of a class  $I$ ,  $\alpha_i$  is the coefficient for  $x_i$ , and  $x_j$  is the point to check if it is inside the convex hull. The above problem is only feasible if and only if  $x_j$  lies inside the set of points,  $I$ , representing a class in the problem. Here, we are optimizing a zero function because the actual solution to this optimization does not matter; rather, it is important that such a solution can be found, i.e., the constraints can be fulfilled. The feasibility and linear-programming problem is defined using the CLP library.<sup>36</sup> The classifier then tries to minimize the number of points that are inside more than one of the convex hulls, with the region of feature space that contains the overlap between any of the convex hulls defined as the overlap region. Once the number of points in the overlap region is determined, a linear SVM model is calculated for the best candidates and used as the new tie-breaking procedure. The first tiebreaker is the number of misclassified points by the SVM model, and the second one is the margin distance. The SVM model is calculated using `libsvm`.<sup>37</sup>



**FIG. 5.** A demonstration of the classification algorithm. A set of 1000 points of three features,  $x_0$ ,  $x_1$ , and  $x_2$ , are sampled from a Gaussian distribution with a standard deviation of 0.5 and centered at the origin. The set is separated into four classes (the red, black, yellow, and light blue circles) by the planes  $x_0 + x_1 + x_2 = 0$  and  $x_2 = 0$  (green surfaces). (a) All points where  $|x_0 + x_1 + x_2| < 0.6$  or  $|x_2| < 0.45$  are moved to a random point in the same class, but outside the margin region, and (b) the original data stored as  $x_3$ ,  $x_4$ , and  $x_5$ . The updated classification algorithm correctly determines that  $\{x_0, x_1, \text{ and } x_2\}$  is the superior classifier, while for the original definitions of only the convex overlap region for three and more dimensions, they would be considered equally good. The projections are shown with an elevation angle of  $7.5^\circ$  and an azimuthal angle of  $145^\circ$ .

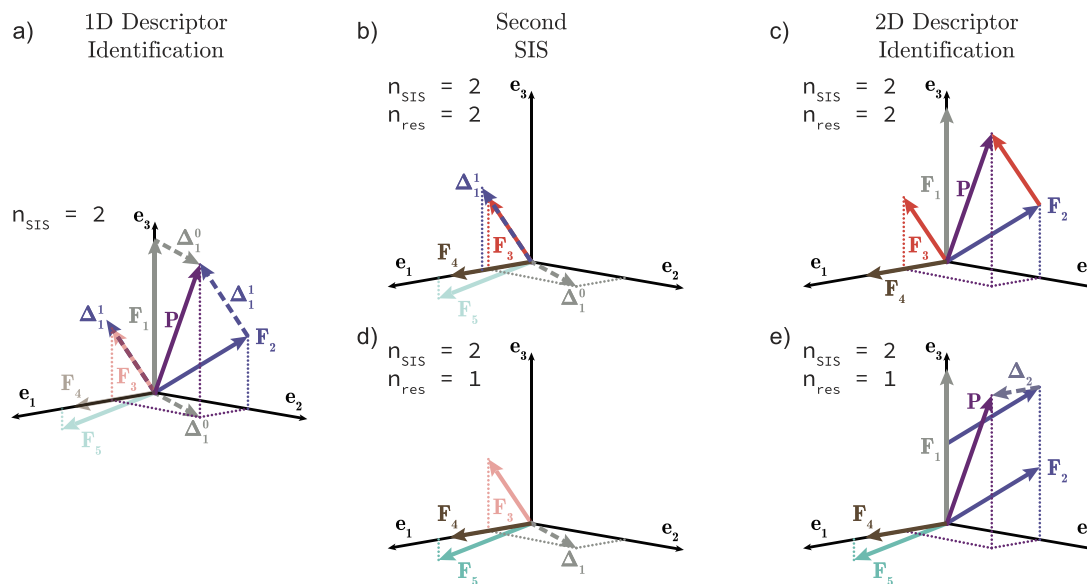


Figure 5 demonstrates the capabilities of the new classification algorithm on a toy problem where a spherical point cloud is separated into four classes by the planes  $x_0 + x_1 + x_2 = 0$  and  $x_2 = 0$ . The choice to use a three-dimensional model was done to demonstrate that the new method can work beyond two dimensions and still be easily visualized, but in principle, this can find a descriptor with an arbitrary dimension. In this example, we randomly sample a Gaussian distribution with a standard deviation of 0.5 and centered at the origin to generate one thousand samples for three features,  $x_0$ ,  $x_1$ , and  $x_2$ , which are then copied and relabeled to  $x_3$ ,  $x_4$ , and  $x_5$ . The samples are then separated into four classes based on whether  $(x_0 + x_1 + x_2)$  is greater than (light blue and black) or less than (red and yellow) zero and whether  $x_2 > 0$  (red and light blue) or  $x_2 < 0$  (black and yellow). To better highlight the separation of the four classes, an artificial margin area is created for  $x_0$ ,  $x_1$ , and  $x_2$  by replacing all points where  $|x_1 + x_2 + x_3| < 0.6$  or  $|x_2| < 0.45$ , with another random point away from the margin, but still within the same class. To focus on the new solver, we do not perform the feature creation step of SISO for this problem; however, the rung two feature of  $(x_0 + x_1) + x_2$  and  $x_2$  would be able to completely separate the classes in two dimensions. Using the updated algorithm, SISO can now easily identify that the set  $\{x_0, x_1, x_2\}$  is the better classifier than the set  $\{x_3, x_4, x_5\}$ , which all previous implementations would fail to do as neither set has any points in the overlap region. More importantly, SISO now provides the  $D$ -dimensional dividing planes found by linear-SVM for all pairs of classes creating an actual classifier automatically.

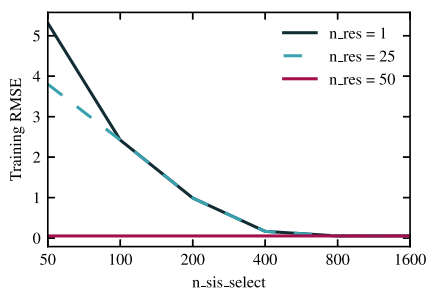
## B. Multiple residuals

The second advancement to the descriptor-identification step of the SISO algorithm is the introduction of a *multiple-residuals* approach to select the features for models with a dimension higher than one. As outlined in the Introduction, in the original SISO algorithm,<sup>38</sup> the residual of the previously found model,  $\Delta_{D-1}^0$ , i.e., the difference between the vector storing the values of the property for each sample,  $\mathbf{P}$ , and the estimates predicted by the  $(D-1)$ -dimensional model ( $\Delta_{D-1}^0 = \mathbf{Dc}_{D-1} - \mathbf{P}$ ), is used to calculate the projection score of the candidate features during the SIS step for the best  $D$ -dimensional model,  $s_j^0 = R^2(\Delta_{D-1}^0, \mathbf{d}_j)$ . Here,  $R$  is the Pearson correlation coefficient, representing a regression problem, and  $j$  corresponds to each expression generated during the feature-creation step of SISO. In SISO++,<sup>24</sup> we extend the residual definition and use the best  $r$  residuals to calculate the projection score:  $\max(s_j^0, s_j^1, \dots, s_j^{r-1})$ . The multiple-residual concept generalizes the descriptor identification step of SISO by using information from an ensemble of models to determine which features to add to the selected subspace. The ensemble is created by using the  $r$  best models from the  $\mathbf{Dc}_{D-1}$  identification step instead of only the top one. The value of  $r$  for a calculation is set as any hyperparameter via cross-validation.

This process is illustrated in Fig. 6 for a three-dimensional problem space (three training samples) defined by  $\mathbf{e}_1$ ,  $\mathbf{e}_2$ , and  $\mathbf{e}_3$  with five candidate features  $\mathbf{F}_1, \dots, \mathbf{F}_5$  to describe the property  $\mathbf{P}$ . In principle, there can be an arbitrary number of feature vectors, but for



**FIG. 6.** An illustration of how tracking multiple residuals can improve the performance of SISO. A three-dimensional problem space (three training samples) defined by  $\mathbf{e}_1$ ,  $\mathbf{e}_2$ , and  $\mathbf{e}_3$  with five feature vectors  $\mathbf{F}_1$  (gray),  $\mathbf{F}_2$  (blue),  $\mathbf{F}_3$  (red),  $\mathbf{F}_4$  (brown), and  $\mathbf{F}_5$  (turquoise) for a property vector,  $\mathbf{P}$  (purple). For this example, the size of the SIS subspace,  $n_{\text{SIS}}$ , is two. The residuals for the best ( $\Delta_1^1$ , gray) and second best ( $\Delta_2^1$ , blue) one-dimensional models are shown as dashed lines. Note that both residuals are plotted twice, once connecting the tips of the arrows representing the residuals are difference of and once rigidly translated in order to stem from the origin. If the number of residuals,  $n_{\text{res}}$ , is one, then only  $\Delta_1^1$  is used (d) and (e) and  $\mathbf{F}_3$  will not be selected in the second SIS step. This means that the best two-dimensional model is not found. However, if  $n_{\text{res}} = 2$  (b) and (c), then  $\mathbf{F}_3$  is selected and the best two-dimensional model, a combination of  $\mathbf{F}_2$  and  $\mathbf{F}_3$ , can be found in the second  $\ell_0$  step.



**FIG. 7.** The training RMSE of a two-dimensional model against the size of the SIS subspace for  $y = 5.5 + 0.4158d_1 - 0.0974d_2 + \delta$ , where  $d_1 = x_0^2 \sqrt[3]{x_1}$ ,  $d_2 = |x_2^3|$ , and  $\delta$  is a Gaussian white noise term pulled from a distribution with a standard deviation of 0.05. The dark blue solid line is the error when learning using one residual, the light blue dashed line is the error when learning using 25 residuals, and the red solid line is the error when learning with 50 residuals.

clarity, we only show five. If only a single residual is used, then the selected two-dimensional model will comprise of  $F_1$  and  $F_4$ , as  $F_3$  is never selected because it has the smallest projection score from the residual of the model found using  $F_1$ . However, because  $F_2$  has a component along  $e_2$ , a better two-dimensional model consisting of a linear combination of  $F_2$  and  $F_3$  exists, despite  $F_2$  not being the most correlated feature to  $P$ . When going to higher-dimensional feature spaces, it becomes more likely that the feature vectors similarly correlated with the property contain orthogonal information, thus the need for using multiple residuals in SISSO.

In order to demonstrate the effect of learning over multiple residuals and to get an estimate of the optimal number of residuals and size of the SIS subspace ( $n_{\text{sis}}$ ), we plot the training root-mean-square error of prediction (RMSE) for the two-dimensional models for the function  $y = 5.5 + 0.4158d_1 - 0.0974d_2 + \delta$ , where  $d_1 = x_0^2 \sqrt[3]{x_1}$ ,  $d_2 = |x_2^3|$ , and  $\delta$  is a Gaussian white noise term pulled from a distribution with a standard deviation of 0.05 in Fig. 7. For this problem, the best one-dimensional descriptor is  $\frac{1}{\sqrt[3]{x_3}}$ , with  $x_3$  being explicitly set to  $(y + \Delta)^{-3}$ , and  $\Delta$  is a Gaussian white noise term with a standard deviation of 20.0. This primary feature was explicitly added in order to ensure that the best one-dimensional model would not contain  $d_1$  or  $d_2$  in this synthetic problem. Because of this, when using a single residual, the SIS subspace size has to be increased to over 400, before  $y$  can be reproduced by SISSO. However, by increasing the number of residuals to 50, SISSO can now find which features are most correlated with the residual of  $d_1$  and it immediately finds  $y$ . In a recent paper published by some of us, we further demonstrate that this approaches effectiveness for learning models of the bulk modulus of cubic perovskites.<sup>25</sup>

#### IV. CONCLUSIONS

In this paper, we described recently developed improvements to the SISSO method and their implementation in the SISSO++ code in terms of both their mathematical and computational details, which constitute a large leap forward in terms of the expressivity of the SISSO method. Utilizing these features provides greater flexibility and control over the expressions found by SISSO and acts as a start to introducing “grammatical” rules into SISSO and symbolic

regression. In particular, concepts such as the units and ranges of the formula could be extended to prune the search space of possible expressions for the final models. We have also described the implementation of *parametric SISSO*, which considerably opens up the range of possible expressions found by SISSO. Finally, we discussed two improvements related to the SISSO solver, i.e., a linear programming implementation for the classification problems and the multiple-residuals technique, both providing extended flexibility in the descriptors and models found by SISSO.

#### ACKNOWLEDGMENTS

TARP thanks Christian Carbogno for valuable discussions related to the parametric SISSO scheme and proof reading those parts of the manuscript. TARP thanks Lucas Foppa for discussions related to the multi-residual approach and proof reading those parts of the manuscript. This work was funded by the NOMAD Center of Excellence (European Union’s Horizon 2020 Research and Innovation Program, Grant Agreement No. 951786), the ERC Advanced Grant TEC1p (European Research Council, Grant Agreement No. 740233), BigMax (the Max Planck Society’s Research Network on Big-Data-Driven Materials-Science), and the project FAIRmat (FAIR Data Infrastructure for Condensed-Matter Physics and the Chemical Physics of Solids, German Research Foundation, Project No. 460197019). TARP acknowledges the Alexander von Humboldt (AvH) Foundation for their support through the AvH Postdoctoral Fellowship Program.

#### AUTHOR DECLARATIONS

##### Conflict of Interest

The authors have no conflicts to disclose.

##### Author Contributions

TARP implemented all methods and performed all calculations. TARP ideated all methods with assistance from LMG. MS and LMG supervised the project. All authors wrote the manuscript.

**Thomas A. R. Purcell:** Conceptualization (equal); Data curation (lead); Formal analysis (equal); Funding acquisition (equal); Investigation (lead); Methodology (equal); Software (lead); Writing – original draft (equal); Writing – review & editing (equal). **Matthias Scheffler:** Conceptualization (supporting); Funding acquisition (equal); Supervision (supporting); Writing – original draft (equal); Writing – review & editing (equal). **Luca M. Ghiringhelli:** Conceptualization (equal); Formal analysis (equal); Funding acquisition (equal); Methodology (equal); Supervision (equal); Writing – original draft (equal); Writing – review & editing (equal).

#### DATA AVAILABILITY

The data that support the findings is available in FigShare at <http://dx.doi.org/10.6084/m9.figshare.23813889>. The code used here can be found on gitlab: [https://gitlab.com/sissopp\\_developers/sissopp](https://gitlab.com/sissopp_developers/sissopp).

## REFERENCES

- <sup>1</sup>T. Zhu *et al.*, *Energy Environ. Sci.* **14**, 3559 (2021).
- <sup>2</sup>S. A. Miller *et al.*, *Chem. Mater.* **29**, 2494 (2017).
- <sup>3</sup>K. Tran and Z. W. Ulissi, *Nature Catalysis* **1**, 696 (2018).
- <sup>4</sup>V. Stanev, K. Choudhary, A. G. Kusne, J. Paglione, and I. Takeuchi, *Commun. Mater.* **2**, 105 (2021).
- <sup>5</sup>P. P. Angelov, E. A. Soares, R. Jiang, N. I. Arnold, and P. M. Atkinson, *WIREs Data Min. Knowl. Discovery* **11**, e1424 (2021).
- <sup>6</sup>D. Gunning *et al.*, *Sci. Rob.* **4**, eaay7120 (2019).
- <sup>7</sup>A. Das and P. Rad, [arXiv:2006.11371](https://arxiv.org/abs/2006.11371) (2020).
- <sup>8</sup>F. Xu *et al.*, “Explainable AI: A brief survey on history, research areas, approaches and challenges,” in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Springer, 2019), Vol. 11839 LNAI, pp. 563–574.
- <sup>9</sup>G. S. I. Aldeia and F. O. De França, “Measuring feature importance of symbolic regression models using partial effects,” in *GECCO '21: Proceedings of the Genetic and Evolutionary Computation Conference* (Association for Computing Machinery, 2021), p. 750–758.
- <sup>10</sup>A. Holzinger, A. Saranti, C. Molnar, P. Biecek, and W. Samek, “Explainable AI methods - a brief overview,” in *Lecture Notes in Computer Science (Including Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 13200 LNAI (Springer Science and Business Media Deutschland GmbH, 2022), pp. 13–38.
- <sup>11</sup>Z. Li, J. Ji, and Y. Zhang, [arXiv: 2111.12210](https://arxiv.org/abs/2111.12210) (2021).
- <sup>12</sup>Y. Wang, N. Wagner, and J. M. Rondinelli, *MRS Commun.* **9**, 793 (2019).
- <sup>13</sup>J. R. Koza, *Stat. Comput.* **4**, 87 (1994).
- <sup>14</sup>T. Mueller, E. Johlin, and J. C. Grossman, *Phys. Rev. B* **89**, 115202 (2014).
- <sup>15</sup>F. Yuan and T. Mueller, *Sci. Rep.* **7**, 17594 (2017).
- <sup>16</sup>S.-M. Udrescu and M. Tegmark, *Sci. Adv.* **6**(16), eaay2631 (2020).
- <sup>17</sup>S. Kim *et al.*, *IEEE Trans. Neural Networks Learn. Syst.* **32**, 4166 (2021).
- <sup>18</sup>M. D. Cranmer, R. Xu, P. Battaglia, and S. Ho, *Learning Symbolic Physics with Graph Networks* (Curran, 2019), see Associates [https://ml4physicalsciences.github.io/2019/files/NeurIPS\\_ML4PS\\_2019\\_15.pdf](https://ml4physicalsciences.github.io/2019/files/NeurIPS_ML4PS_2019_15.pdf).
- <sup>19</sup>M. Valipour, B. You, M. Panju, and A. Ghodsi, *Symbolicpt: A Generative Transformer Model for Symbolic Regression* (Curran Associates, 2021), see [https://neurips2022-enlsp.github.io/papers/paper\\_62.pdf](https://neurips2022-enlsp.github.io/papers/paper_62.pdf).
- <sup>20</sup>B. K. Petersen *et al.*, “Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients,” in OpenReview ICLR 2021 Conference (2023), see [https://openreview.net/forum?id=m5Qsh0kBQG&utm\\_source=miragenews&utm\\_medium=miragenews&utm\\_campaign=news](https://openreview.net/forum?id=m5Qsh0kBQG&utm_source=miragenews&utm_medium=miragenews&utm_campaign=news)
- <sup>21</sup>W. Tenachi, R. Ibata, and F. I. Diakogiannis, “Deep symbolic regression for physics guidconstraints: Toward the automated discovery of physical laws,” [arXiv:2303.03192](https://arxiv.org/abs/2303.03192) [astro-ph.IM].
- <sup>22</sup>R. Ouyang, E. Ahmetcik, C. Carbogno, M. Scheffler, and L. M. Ghiringhelli, *J. Phys. Mater.* **2**, 024002 (2019).
- <sup>23</sup>R. Ouyang, S. Curtarolo, E. Ahmetcik, M. Scheffler, and L. M. Ghiringhelli, *Phys. Rev. Mater.* **2**, 83802 (2018).
- <sup>24</sup>T. A. R. Purcell, M. Scheffler, C. Carbogno, and L. M. Ghiringhelli, *J. Open Source Software* **7**, 3960 (2022).
- <sup>25</sup>L. Foppa, T. A. Purcell, S. V. Levchenko, M. Scheffler, and L. M. Ghiringhelli, *Phys. Rev. Lett.* **129**, 55301 (2022).
- <sup>26</sup>C. J. Bartel *et al.*, *Sci. Adv.* **5**, eaav0693 (2019).
- <sup>27</sup>G. R. Schleder, C. M. Acosta, and A. Fazzio, *ACS Appl. Mater. Interfaces* **12**, 20149 (2020).
- <sup>28</sup>Z.-K. Han *et al.*, *Nat. Commun.* **12**, 1833 (2021).
- <sup>29</sup>G. Pilania, C. N. Iverson, T. Lookman, and B. L. Marrone, *J. Chem. Inf. Model.* **59**, 5013 (2019).
- <sup>30</sup>J. Fan and J. Lv, *J. R. Stat. Soc. Ser. B: Stat. Methodol.* **70**, 849 (2008).
- <sup>31</sup>T. A. R. Purcell, M. Scheffler, L. M. Ghiringhelli, and C. Carbogno, *npj Comput. Mater.* **9**, 112 (2023).
- <sup>32</sup>S. G. Johnson, The NLOpt nonlinear-optimization package, 2021, <http://github.com/stevengj/nlopt>.
- <sup>33</sup>T. H. Rowan, “Functional stability analysis of numerical algorithms,” Ph.D. thesis, University of Texas at Austin, 1990.
- <sup>34</sup>J. A. Nelder and R. Mead, *Comput. J.* **7**, 308 (1965).
- <sup>35</sup>T. Runarsson and X. Yao, *IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.)* **35**, 233 (2005).
- <sup>36</sup>J. J. Forrest *et al.*, coin-or/clp: Version 1.17.6.
- <sup>37</sup>C.-C. Chang and C.-J. Lin, *ACM Trans. Intell. Syst. Technol.* **2**(3), 27 (2011), Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- <sup>38</sup>R. Ouyang, S. Curtarolo, E. Ahmetcik, M. Scheffler, and L. M. Ghiringhelli, *Phys. Rev. Mater.* **2**, 083802 (2018).