



# Systematic tree search for symbolic regression: deterministically searching the space of dimensionally homogeneous models

Marcel Anselment<sup>1</sup> · Moritz Neumaier<sup>1</sup> · Stephan Rudolph<sup>1</sup>

Received: 30 April 2024 / Revised: 27 May 2025 / Accepted: 31 July 2025 / Published online: 18 August 2025  
© The Author(s) 2025

## Abstract

In engineering, the identification of the functional relationship between a set of physical variables is a common objective. In addition to physics, an increasing number of machine learning methods are being utilised to create these models. In particular, a machine learning algorithm, generally referred to as symbolic regression (SR), generates interpretable models in the form of mathematical functions. Thus, a physical interpretation of the relationship found remains possible. While the majority of contemporary SR algorithms do not employ a systematic approach, incorporating stochastic elements, this paper proposes a novel implementation of symbolic regression. This implementation takes the form of a systematic tree search, which is deterministically spanning and searching the space of possible symbolic models. For this purpose, new mathematical models are successively generated by small modifications of an existing model, whereby the search tree grows iteratively and with it the complexity of the models. The control of the search process, and thus the selection of the model to be changed is based on a specially designed dynamic heuristic. Through the additional use of dimensional analysis, the dimensional homogeneity of the models created can be guaranteed. The efficiency of the method to arrive at interpretable models from synthetic data is illustrated by finding each of the 12 Nguyen benchmark data sets. The robustness of the approach is shown by reconstructing Breguet's range formula from data with varying degrees of noise.

**Keywords** Symbolic regression · Artificial intelligence · Dimensional analysis

## 1 Introduction

In the field of engineering, functional relationships between physical variables are very often required, for example, in the design process for the dimensioning of components or in the simulation of thermal or fluid mechanical processes. In some special cases, closed models that have been physically derived, usually by solving differential equations, can be used. If this is not the case, the relationship between parameters can be investigated experimentally. The determined measurement or simulation data can then be used to create a mathematical model using a suitable regression method.

A common approach is to specify a parameterized symbolic model whose coefficients are then fitted to the data using the least squares method. Well-known examples are the Nusselt relationships in heat transfer [1]. However, in this case, the engineer must provide a suitable parameterized model with their expertise. Alternatively, a very general model can be used, such as a polynomial of high degree or a fully connected neural network. The latter, in particular, have been shown to be able to approximate any function [2]. However, the models represented by high-order polynomials or neural networks are difficult to interpret and deviate greatly from the handmade formulae known from physics. This is where symbolic regression (SR) can help. It attempts to automatically determine compact, physics-based symbolic models from data that best describe the relationships in the data.

The concept of symbolic regression has gained significant attention in the machine learning (ML) community and is rapidly evolving as ML itself expands. Despite its growing interest, the idea of symbolic regression is not new. In the early 1990s, Koza addressed the issue of SR by employing a genetic programming approach [3]. This method involves

✉ Marcel Anselment  
anselment@ifb.uni-stuttgart.de

Moritz Neumaier  
neumaier@ifb.uni-stuttgart.de

Stephan Rudolph  
rudolph@ifb.uni-stuttgart.de

<sup>1</sup> Institute of Aircraft Design, University of Stuttgart,  
Pfaffenwaldring 31, 70569 Stuttgart, Germany

the evolution of a population of models through a process of selection, crossover, and mutation. This approach is to this day 1 of probably the most common techniques for addressing SR and was taken up and further developed by various researchers. So did, for example, Rudolph in the late 1990s, who applied symbolic regression in combination with genetic programming to make early neural networks explainable for engineering applications [4]. In the early 2000s, even a commercial software product named Eureqa emerged to address SR.

With increased computational power and the emergence of modern AI algorithms, symbolic regression has expanded into various approaches. For example, Peterson et al. [5] use a sequence-based representation of models and train a reinforcement learning agent, which is then able to iteratively predict a good next operator or operand of the sequence to maximise the quality of the model. Cranmer et al. [6] use graph neural networks to divide symbolic regression into sub-problems, each of which is then solved again via genetic programming. Furthermore, transformer-based neural networks, which are commonly used in a wide range of applications, have been employed to output models, including coefficient values, directly from a data set [7]. For a more comprehensive overview of symbolic regression research, Angelis et al. [8] offer an extensive examination of various methodologies and their applications.

However, despite the variety of approaches, a significant degree of randomness persists in many symbolic regression methods. Consequently, more systematic and deterministic approaches are relatively rare. One of them is AI-Feynman, proposed by Udrescu et al. [9], which attempts to address SR by subdividing the problem into smaller sub-problems and then solving them through a brute-force algorithm. However, the brute-force approach is designed for the small sub-problems and would quickly reach the limits of the available computing power for somewhat larger models.

Rivero et al. [10] also use a deterministic approach in their method. They develop a very simple model iteratively through evolutionary adjustments. However, these changes are not determined randomly, but are selected on the basis of clear conditions.

Sun et al. [11] explored the use of Monte Carlo tree search (MCTS), a technique that gained a lot of popularity due to its successful use in game AIs such as AlphaGo. They start with a simple model in the root node and then develop it along the tree path in the simulation phase of MCTS until a valid symbolic model is created. As the name Monte Carlo implies, this method includes a significant degree of randomness.

Olivetti de Franca et al. [12] take a different approach by constraining the model's form and, therefore, enable a more systematic description of the model. This is then used to do a tree search, which is, however, also using genetic programming to create new models. The main disadvantage

lies in the constraint of the model's form, which prohibits the generation and, therefore, finding of certain solutions.

This paper attempts to fill this gap by presenting a systematic and deterministic way to search the space of all possible models. To this end, the tree search presented here is a method that spans the entire search space and searches it systematically. Since symbolic regression is considered NP-hard [13] it is in general not possible to analyse the space of eligible models in full, thus a brute force search is not sufficient. Consequently, a heuristic is incorporated into the search process to ensure its effective control, and duplicate detection is implemented as a fundamental component of the tree search process. In addition, dimensional analysis is used to reduce the search space and simultaneously ensure the dimensional homogeneity of every model found.

A major advantage of the method is that you end up with a comprehensive list of suitable models that can be weighed against each other in a Pareto analysis, taking into account accuracy and complexity. Furthermore, the search is scalable and the number of models to be analysed is only limited by memory and computing resources.

## 2 Related work

This section provides an overview of prior research that is relevant to the present work, focussing on the representation of symbolic functions and general tree search methods.

### 2.1 Representation of symbolic functions

Symbolic functions are usually represented as a string chain and decoded from left to right, with brackets and rules, such as dot-before-dash, specifying which operator is evaluated

Fig. 1 Model tree of the function  $f(x) = c + c \sin(x)$

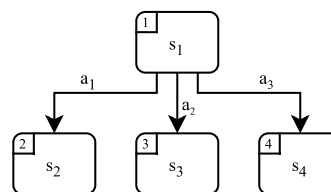
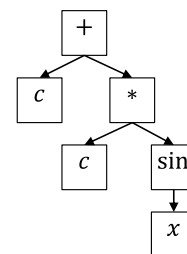
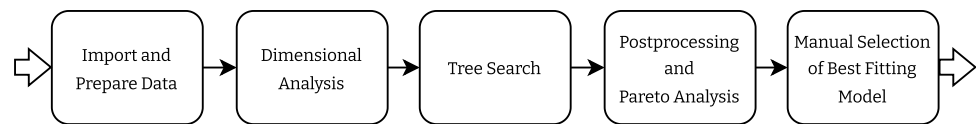


Fig. 2 General search tree

Fig. 3 Overview



first. This representation is not favourable for the machine processing of function expressions, which is why common computer algebra systems express them in a nested list containing the operators and operands. The operators considered in this paper are addition (+), multiplication (\*), exponentiation ( $\wedge$ ) and transcendental functions (T). Division is formed indirectly through exponentiation with negative exponents and subtraction via addition with negative values. Operands can be variables ( $x_i$ ), fixed numbers in  $\mathbb{R}$  (e.g. 2) and coefficients ( $c_i \in \mathbb{R}$ ). While the coefficients are determined by optimisation at a later stage, fixed numbers can no longer be changed.

Nested lists can be better visualised in a tree structure. Every node of the tree then contains an operator or an operand. The operators store their operands (which in turn can be operators) in the child nodes. The calculation sequence is inherently defined in the structure of the tree. Depending on the operator, the number of child nodes varies. While addition and multiplication can have an unlimited number of children (but at least two), exponentiation must have exactly two and transcendental functions must have exactly one child node. Operands are the leaf nodes of the tree and thus cannot have child nodes.

For example, the function  $f(x) = c + c \sin(x)$  can be written in a nested list as follows:  $[+ [c, * [c, \sin [x]]]]$ . Figure 1 shows the corresponding model tree.<sup>1</sup>

A major advantage of this data structure is that the operators and operands can be easily accessed and modifications to the function can be implemented using graph transformations. In addition, the isomorphism of trees can be used to detect duplicated models, which is further described in Sect. 3.2.3.

## 2.2 Tree search

Search trees can be used to explore a state space. These could involve navigation problems or finding the best strategy in games, such as chess or Go. Each node of the tree contains a state, whereas the paths between the nodes indicate the actions that lead from one state to another. The tree's growth process typically involves the expansion of a single node, accompanied by the execution of potential actions

within that node, thereby generating new states. These new states are then designated as children of the expanded node. As illustrated in Fig. 2, a minimal tree with a single expanded node is presented.

A common method of categorising tree search algorithms is by how they decide which node to expand next [14]. This can be done in an uninformed way, as in a breadth-first or depth-first search. In this case, the node at the lowest or deepest level of the tree is expanded first. Another way is to use problem-specific information. This is often achieved by creating a heuristic function  $H(n)$  to evaluate each node. In this approach, generally known as best-first search, the next node to be expanded is the node with the best value of  $H$ .

## 3 Automated model creation based on tree search

While conventional regression methods specify a model and only adjust the coefficients to the data, symbolic regression goes one step further and also adjusts the model itself. The symbolic models are built based on elementary functions<sup>2</sup>, which is particularly advantageous when compact models are to be found. The difficulty lies in finding suitable models from the infinite number of possible models. Even with restrictions on the models, the combinatorial possibilities for assembling the models from the building blocks presented in Sect. 2.1 quickly reach a level, where it is no longer feasible to test all possibilities in an acceptable amount of time. For this reason, an intelligent algorithm is inevitably required to select the models to be analysed.

Figure 3 shows the process chain discussed in this paper. First, the data from which the model is to be built is imported and the required pre-processing is performed. Then a dimensional analysis is performed to be able to do the model creation based on the dimensionless invariants. This model creation is performed using a tree search algorithm. The results are post-processed and an automated Pareto analysis is performed. If all trade-offs, such as complexity vs. error, are known in advance, the final result is available;

<sup>1</sup> In this paper, the tree concerned is referred to as a model tree, in contrast to the search tree presented later. In other papers, the terms syntax tree, expression tree, and parse tree are also frequently used.

<sup>2</sup> Elementary functions are functions "that can be realised by algebraic operations, concatenations and inversions from algebraic functions and the exponential function. This includes all rational functions, exponential and logarithmic functions, the trigonometric functions and hyperbolic functions, as well as their inverse functions." [15] (translation by the authors)

otherwise, a user can select the best one for the given task from the filtered results.

### 3.1 Data preprocessing using dimensional analysis

The data for a  $n$ -dimensional regression problem (consisting of  $n - 1$  input dimensions and 1 output dimension) with  $K$  data points typically is given in the form:

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{2,1} & \cdots & x_{n-1,1} \\ x_{1,2} & x_{2,2} & \cdots & x_{n-1,2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1,K} & x_{2,K} & \cdots & x_{n-1,K} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{bmatrix} \quad (1)$$

with the input data matrix  $\mathbf{X}$  and the output data vector  $\mathbf{y}$ . The goal is to find a model  $f$  that predicts the output of a given input:

$$y = f(x_1, x_2, \dots, x_{n-1}). \quad (2)$$

If the user does not provide two different data sets for training and validation, the data is split automatically in an 80:20 ratio. Therefore, every fifth point is used as a validation data point.

However, models for the relationship between dimensional quantities are subject to some restrictions. After all, the model must adhere to the principle of dimensional homogeneity and ensure that the same unit is to the right and left of the equal sign.<sup>3</sup> If a formula is dimensionally homogeneous, it is independent of the chosen fundamental units [17, p. 79]. Furthermore, only dimensionless expressions can be processed within transcendental functions. To ensure dimensional homogeneity for each model, it makes sense to search for the model in the dimensionless space. For this purpose, a dimensional analysis of all variables occurring in the model is carried out. A problem with  $n$  dimensioned variables can thus be simplified to a problem with fewer dimensionless variables. The number of variables of the new dimensionless problem  $m$  depends on the exact dimensions of the original variables. The way to examine the dimensional analysis is written in detail, for example, in [17] and [18]. The data set can automatically be transferred to the dimensionless data set with fewer variables.

$$\pi_m = f(\pi_1, \pi_2, \dots, \pi_{m-1}), \quad (3)$$

where the invariants are a polynomial product of the dimensioned variables. The exact exponents  $d_{i,j}$  of the dimensioned

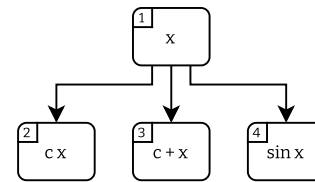


Fig. 4 Search tree (exemplary)

variable  $x_i$  for each dimensionless variable  $\pi_j$  are calculated in the dimensional analysis.

$$\pi_j = x_1^{d_{1,j}} x_2^{d_{2,j}} \dots x_n^{d_{n,j}} \quad (4)$$

The symbolic regression algorithm then uses the dimensionless data set, which ensures that every model found complies with the condition of dimensional homogeneity. Furthermore, this has the advantage of significantly reducing the number of possible models and thus the search space.

The dimensional analysis is automated using matrix multiplication, as described in [19]. By permuting the rows of the dimension matrix, it is possible to generate different sets of dimensionless variables and to provide a set with meaningful exponents  $\mathbf{d}$ . Once the search for the model has been completed, the dimensioned model can be calculated using a back-transformation.

### 3.2 Tree search methodology

The algorithm proposed in this paper is based on a tree search, which is used to span and search the space of possible models. Each node in the search tree contains one independent symbolic model and, therefore, one corresponding model tree, as shown in Sect. 2.1. During the search process, the tree grows iteratively by expanding one node and adding new nodes with new models. The details of which nodes are to be expanded and how the new models are to be created are described in this section. An exemplary search tree, consisting of four nodes, is shown in Fig. 4.

The search tree begins with a single root node that contains the basic model  $f(\mathbf{x}) = x_1$ . The tree is then expanded iteratively, with the models becoming more complex as the depth increases. Each iteration step can be divided into seven steps:

1. Choose the next node to expand using a heuristic
2. Create new models by minimally modifying the model in the selected node
3. Remove potential duplicate models
4. Optimize the remaining new models
5. Evaluate the new models
6. Expand the tree
7. Check the termination criteria

<sup>3</sup> “For physical equations to remain valid for any chosen unit system, each summand of such an equation must carry the same dimension. Equations that satisfy this property are called dimensionally homogeneous.” [16]

The search continues until one of the termination criteria is met. Figure 5 shows an example of the expansion of the search tree within one iteration step. Of all the unexpanded nodes (2, 3, 4), node 4 contains the best model with a heuristic value of 47.0. How this value is determined is discussed in Sect. 3.2.1. Then three new variants of this model are created and saved in the child nodes (5, 6, 7). The coefficients of the new models are optimised, and their deviation and heuristic values are calculated. In the next iteration, node 5 will be expanded, because it has the new lowest heuristic value.

### 3.2.1 Choice of the node

The selection of the next node to be expanded is a very important aspect of the tree search, especially when it comes to searching very large state spaces. The reason is that then it is not possible to expand every node and the selection of the node determines where to search and which areas are neglected.

The node selection in this work should be done in such a way that, on the one hand, good models should be found quickly while not leading in to dead ends, and, on the other hand, the search space should be expanded broadly so that the best models are not overlooked.

**Heuristic function** An uninformed tree search does not make sense in this case, which is why we propose a heuristic tree search that uses information from existing models to evaluate the nodes. In the style of a best-first search, the best-valued node is then selected by the heuristic for expansion in each iteration. This tree search uses a heuristic function for each node  $N$  that consists of the loss of the model  $L$ , the complexity of the model  $C$  and a penalty parameter  $p$ .

$$H_N = L_N + p C_N \tag{5}$$

In the following the parts of the heuristic function gets explained in detail.

**Loss of the model** The basic idea behind the heuristic is that the paths whose nodes already contain good models are pursued further. Therefore, the error of the models is directly included in the heuristic value. If the model error of the node is low, this also reduces the heuristic value, and the node is favoured.

**Complexity** When generating new models, there are many modifications that basically always lead to an improved or at least unchanged model error (for example, adding a coefficient). If only the model error were considered, many of the newly generated models would be evaluated by the heuristic function as the new best nodes. The current path would thus be followed further and further until a potential model limitation (see Sect. 3.2.2) is reached.

To counteract this phenomenon and also support the general preference for compact symbolic models, the complexity of the models is also included in the heuristic. As a result, if there is no noticeable improvement in model prediction (this means  $L_N \approx \text{const}$ ), the heuristic value of the model increases with increasing tree depth. This is because complexity increases, and thus  $C_N$ , which in turn ensures a higher heuristic value  $H_N$ . The exploitation of the paths with no improvement in loss is, therefore, automatically cancelled.

The complexity of the model is currently determined by the number of operands and operators in the model tree. At this point, it is also possible to incorporate more individual complexity metrics, for example, to weight the operands and operators differently to take account of the different computational effort required by the machine.

**Penalty parameter** The penalty parameter weights the influence of model complexity on the heuristic. In other words, it penalises nodes for the complexity of their models, hence the name. The choice of the exact value for  $p$  has a decisive influence on the search. For high values for  $p$  the search tends to explore the search space similar to a breadth-first search, because the models with low complexity in the upper layers of the tree then have the best heuristic values. Small values for  $p$ , on the other hand, ensure the exploitation of the paths with the most accurate models similar to a depth-first search.

That is why the penalty parameter  $p$  can be used to skillfully control the exploration–exploitation trade-off at runtime. Practice has shown that varying the parameter over the full range from low to high at runtime yields the best search results. By default, the penalty changes every 10 iterations between 100, 10, 1, 0.1 and 0.01. However, it is recommended to avoid setting the parameter to 0, as this usually results in overfitting.

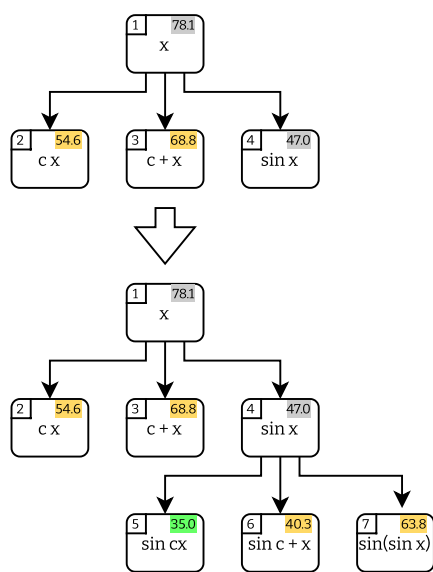


Fig. 5 Expansion of the search tree within one iteration step

**Table 1** Overview of modifications used to create new model variants

ID	Before	After
A	$x_i$	$x_i + c$
B	$x_i$	$c x_i$
C	$x_i$	$x_i^c$
D	$x_i$	$x_i^{-1}$
E	$x_i$	$x_i^2$
F	$x_i$	$x_i^3$
G <sup>1</sup>	$x_i$	$T_n(x_i)$
H <sup>1</sup>	$T_m(x_i)$	$T_m(x_i) T_n(x_i)$
I <sup>2</sup>	$x_i$	$x_i x_j$
J <sup>2</sup>	$x_i$	$x_i + x_j$
K <sup>2</sup>	$x_i$	$x_i^{x_j}$
L <sup>2</sup>	$T_m(x_i)$	$x_j T_m(x_i)$
M <sup>2</sup>	$f(\mathbf{x})$	$f(\mathbf{x}) + x_j$

<sup>1</sup>For every transcendental function  $T_n$

<sup>2</sup>For every input variable  $x_j$

### 3.2.2 Generation of new variants

In the second step, new variants are generated by slightly modifying the previously selected model  $f$ . The primary goal of the new variant generation is to be able to generate potentially any symbolic model within user-defined restrictions. In particular, this requires the non-linear linking of terms. Therefore, these modifications primarily consist of changes to the variables. For example, the variable  $x_i$  in the function expression is replaced by  $x_i + c$ ,  $c x_i$  or  $T(x_i)$ , where  $T$  stands for one of the transcendental functions.<sup>4</sup> Furthermore, there are modifications, such as multiplying the function expression by an additional variable  $x_j$  or adding a new summand. A complete overview of the modifications can be found in Table 1. All modifications are then applied individually to every variable or transcendental function to create new models.

**Model restrictions** However, not every modification creates a meaningful new model. For example, the model  $f_1 = x + c_1$  is transformed to  $f_2 = x + c_1 + c_2$  by applying modification A. In this new model, the additional coefficient  $c_2$  does not offer an advantage. For this reason, unnecessary modifications should be detected and prevented. Fortunately, the data structure of the models presented in Sect. 2.1 makes it quite easy to restrict them. When new variants are created, the system checks whether this variant is permissible. In this particular case, the modification is allowed only if the

variable node in the model tree does not have a coefficient as a sibling.

A further restriction is that, equivalent to the sums mentioned above, each product node may only have one coefficient as a factor. In both cases, there is no advantage in having an extra coefficient, since the coefficients could be merged together. Moreover, sums must not have sums and products must not have products as direct child nodes. Due to the associativity of the operands  $a + b + c$  is the same as  $a + (b + c)$ . However, this condition is crucial for the detection of duplicates in the following step.

In addition, it is often useful for users to constrain the models, for example, using domain knowledge, and thus reduce the search space. Therefore, the settings provide a limit to the number of each variable, the overall number of coefficients, and the total number of nodes of the model tree. Furthermore, the user is free to select the transcendental functions that are used to build the models. Advanced settings also offer the user the option of limiting the nesting of transcendental functions. In this case, the algorithm counts the transcendental functions in the predecessor nodes in the model tree when applying modification G from Table 1. If the limit has already been reached, the variant is not created accordingly.

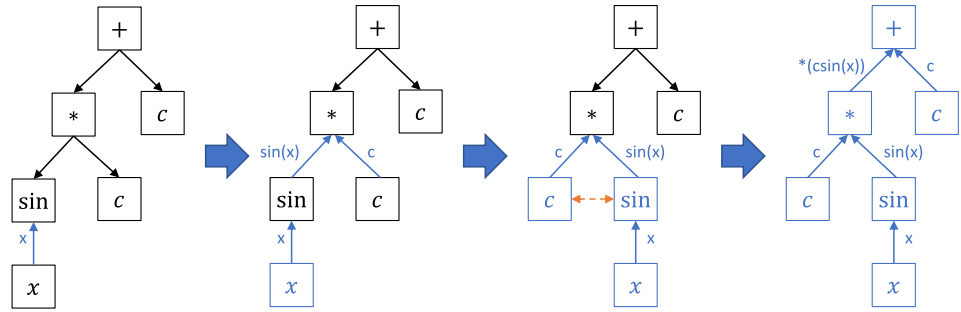
### 3.2.3 Removing of duplicated models

Due to the structure of the search tree, identical models are created on different paths, which is a huge advantage, as the predecessor models are not the same. In particular, if the predecessor nodes of a good model approximate poorly, leading to poor heuristic values, the path remains unconsidered for a long time. On the other hand, the model can maybe be found quickly via a different path, as the predecessor models already have good heuristic values. However, it is immensely important for the functioning of the algorithm to recognise models that occur multiple times, since identical models also generate identical new variants, which in turn do the same. The number of multiple models thus grows exponentially. At the beginning of the search, the effect may still be negligible; in an advanced stage, the search is increasingly concerned with evaluating the same models.

First, the question arises as to when the models are identical at all. Basically, two models are identical if their model trees are identical. However, based solely on this criterion, the two models  $x + c$  and  $c + x$  would not be identical. To solve the problem of commutativity, the trees are brought to a standard form. A recursive algorithm, which is based on one in [20, p. 84] to test two trees for isomorphism, lexically sort the child nodes of each node and returns a string consisting of the contents of the sorted child nodes to the parent node. The parent node itself uses this string as its content, surrounds it with brackets, and concatenates it with its own

<sup>4</sup> This includes the trigonometric functions, as well as their inverse functions and reciprocals and the exponential function and its inverse, the logarithm function.

**Fig. 6** Recursive algorithm for sorting the model trees. On the far left is the model tree before, on the far right after sorting



symbol in string representation. Figure 6 shows an example of this procedure for the model  $f(x) = \sin(x)c_1 + c_2$ . The strings returned to the parent node are coloured blue.

Since the algorithm is recursive, first the string  $x$  is backpropagated to the sine node. Next, in the multiplication node, the returned strings of the child nodes,  $\sin(x)$  and  $c$  are sorted, resulting in the two nodes being swapped. The two strings are joined, bracketed, and propagated back to the parent node. The product node then concatenates its own string  $*$ . In the next step, the procedure is repeated for the addition node. As  $*(c \sin(x))$  is lexically placed before  $c$ , the order remains the same at this point. It is crucial to recognise that the indices of the coefficients are not considered, whereas those of the variables must be incorporated into the string. The full string created in this example is  $+(*(c \sin(x))c)$ . The resulting string is then compared to the string representation of the other models. If the string already exists, the new generated model is ignored.

It is important to note that not all child nodes are reordered. In comparison to the algorithm in [20] the operands of non-commutative operators (power operator) obviously must not be swapped.

Furthermore, the models  $f_1 = x^2$  and  $f_2 = xx$ , for example, are intentionally not recognised as duplicates, even though they are mathematically the same. This is because their model trees are not isomorphic to each other, resulting in different variants being created from them. Modification  $x \rightarrow \sin(x)$ , for example, turns model 1 into  $\sin(x)^2$  and model 2 into  $\sin(x)x$ .

### 3.2.4 Optimizing of coefficients

The newly generated models contain free coefficients  $\mathbf{c} = [c_1, c_2, \dots]$ , which are fitted to the data in the current step. To do this, an objective function<sup>5</sup> is first set up. It has proven useful to add up the squared deviation of the measurement and the prediction of the model at all  $K$  measurement points.

$$O_N = \sum_{k=1}^K (f(\mathbf{c}, \mathbf{x}_k) - y_k)^2 \tag{6}$$

The input data vectors  $\mathbf{x}_k = [x_{1,k}, x_{2,k}, \dots, x_{n-1,k}]$  and the output data vector  $\mathbf{y}$  are known; therefore, the objective function  $O_N$  is only dependent on the coefficients  $\mathbf{c}$ . As these are generally incorporated into the objective function in a non-linear manner, non-linear optimisation methods are also required to find suitable values for the coefficients. Due to the consideration of the quadratic deviation, the optimisation is also referred to as a nonlinear least squares method. A serious characteristic of non-linear optimisation problems is that in general and in finite computing time it cannot be guaranteed to find a global minimum, i.e. the best set of coefficients  $\mathbf{c}$  [21]. Only local optima are found in the vicinity of an initial value estimate for the coefficients.

Currently, a so-called trust region method is used as an optimisation algorithm, in which a quadratic replacement problem is solved iteratively from the initial value estimate in a trust region [22].

For general models with numerous coefficients, such as artificial neural networks, there are usually many local optima that enable good model adjustments. For compact models with very few coefficients, there is often only one or a few reasonable solutions. It is, therefore, very important to find one of these optima and, in the best case, the global optimum. This is only possible if the initial values are already close to one of these optima. However, if the initial value estimate is poor, it is possible that no optimum is found at all. The tree search has proven to be an advantage in this respect. Due to the incremental structure of the models, the already adjusted coefficients of the predecessor model in the parent node can be used as initial value estimates. New additional coefficients added to the model by the modification are set to 1 by default. Unfortunately, this initial value estimation still does not guarantee finding a good optimum. To increase the chances of success, it has been found useful to perform the optimisation procedure for models of low complexity several times with different starting values. For this purpose, new starting values are specified in the range between  $-10$  and  $10$  for models with 3 or fewer coefficients.

<sup>5</sup> In machine learning jargon, this would be referred to as a loss function.

The user can define in the settings how often this process is executed (by default, it is executed twice). As the optimisation of the coefficients currently consumes a significant amount of computing time, it is intended to further develop the initialisation of the initial values in the future.

### 3.2.5 Evaluation of new models

In the last iteration step, the new models are evaluated using a validation data set. For the evaluation function, currently the relative deviation of model prediction and measuring values is used.

$$L_N = \frac{100}{K} \sum_{k=1}^K \left( \min \left[ 1, \frac{|f(\mathbf{c}, \mathbf{x}_k) - y_k|}{\bar{y}} \right] \right) \quad (7)$$

$$\bar{y} = \frac{1}{K} \sum_{k=1}^K |y_k|. \quad (8)$$

Relativization with the mean absolute value  $\bar{y}$  ensures a certain independence of the evaluation from the specific data set. Otherwise, the impact of the evaluation value in the heuristic would depend on the scale of the measuring values. It has been shown that using the absolute values of  $\mathbf{y}$  results in an improvement of the reference value. This is because otherwise positive and negative values can cancel each other out, resulting in a very small reference value that, in turn, makes the relative error too high.

Since the evaluation value later is used in the search heuristic, the deviation in one single measuring point is limited to 1. Without this limitation, some models never get expanded due to an exorbitant high heuristic value. However, since models, especially compact ones, often improve drastically with simple modifications, this behaviour should be prevented.

The use of a validation data set in this step is crucial to prevent the symbolic regression algorithm from overfitting, which in this case is driven by two phenomena. On the one hand, periodic functions, even simple ones, such as trigonometric functions or function blocks that tend to oscillate, such as higher order polynomials, can cause the model to deviate significantly from reality between training data points. In this case, the validation points between the training points then lead to large deviations in the evaluation function. On the other hand, models that become more complex with search depth can increasingly adapt to the training data. If the evaluation is also carried out with this data, the evaluation will continue to improve along the path, while it will deteriorate when overfitting occurs using validation data, and the path is, therefore, aborted.

### 3.2.6 Expansion of search tree by new models

Finally, all new models are saved in the search tree as child nodes of the node selected in the first iteration step. The expanded node is marked as expanded and is no longer in the best node selection, even if its heuristic value remains best.

### 3.2.7 Check of termination criteria

After each iteration, the algorithm checks if the termination criteria are met. The user can set three criteria: the maximum number of models to evaluate, the maximum search time, and a target accuracy. In the case of the target accuracy, the algorithm stops when a model with an accuracy equal to or less than the target accuracy is found. However, after one of the criteria is met, the termination criteria can be adjusted, and the search can be resumed.

## 3.3 Post processing and Pareto analysis

When the search process is finished, a multitude of models is available. They can be ordered by the relative mean squared error  $L$  or the complexity  $C$ . If the trade-off between these criteria is known in advance, the final model selection can be carried out automatically. As this is often not the case, a Pareto analysis can be performed automatically. The user can then select the best-fitting model for the given task.

## 4 Examples

In this section, the symbolic regression algorithm is tested using various data sets. First, the power of dimensional analysis in the context of symbolic regression is demonstrated using an example from aviation. Second, the often used Nguyen data sets are used to benchmark the algorithm. All tests in this paper were performed on a PC with Windows 10, Intel® Xeon® W-11855 M CPU 3.20 GHz with 64 GB RAM. Moreover, parallel computing is not currently being used.

### 4.1 Breguet formula for range estimation of aircraft

The Breguet formula is used in aeronautics to estimate the range of aircraft during cruise flights. It can be found, for example, in [23, p. 1164]. For turbojet aircraft, the formula reads as follows:

$$r = E \frac{v}{b_f g} \ln \left( \frac{m_0}{m_1} \right) \quad (9)$$

The range  $r$  is calculated based on the aerodynamic glide ratio  $E$ , the gravitational constant  $g$ , the velocity of aircraft

$v$ , the specific fuel consumption  $b_f$  and the mass  $m_0$  at the beginning of the cruise flight as well as the mass  $m_1$  at the end of the cruise flight.

Equation (9) is then used to create a synthetic data set consisting of 500 randomised data points at predetermined intervals for each variable. With help of the dimensional analysis (see Sect. 3.1) the 7 dimensional variables can be reduced to 4 dimensionless variables

$$R = \frac{g r}{v^2}, \quad M = \frac{m_0}{m_1}, \quad V = v b_f, \quad E$$

The regression problem  $r = f(E, v, b_f, g, m_0, m_1)$  thus is simplified to

$$\frac{g r}{v^2} = f\left(\frac{m_0}{m_1}, v b_f, E\right) \tag{10}$$

Using the dimensionless variables, Eq. (9) becomes:

$$R = \frac{E}{V} \ln(M). \tag{11}$$

### 4.1.1 Noise-free data

The algorithm is now trained using the previous described data set. Therefore, it is first divided into training and validation data in the ratio of 80:20. Then the symbolic regression algorithm tries to find the Breguet formula by only using these data sets. The model restrictions were set to three variable nodes for each variable and seven coefficient nodes. Sine, cosine, tangent, exponential and logarithm functions were used as transcendental functions, where their nesting was not allowed. The algorithm is set to terminate after evaluating 100,000 models.

With these settings, the entire search process took around 26 min. After 8 min and evaluating 29, 805 models, the SR algorithm finds the Breguet formula in the desired expression 11, a model with an error less than  $10^{-5}\%$ .<sup>6</sup> However, is already found after around 20 s, In addition, after the search is finished, the algorithm has found 3442 more models with an error less than  $10^{-5}\%$ . To finally decide on a model, it is prudent to additionally consider the complexity of the model in question, which is also a widely accepted practice in the field of symbolic regression.

Graphically, this can be done by plotting the models in a so-called pareto analysis, where the complexity of the model is plotted against the accuracy. Typically, the best models, which are simple and accurate, are in the top right quadrant. The most accurate models of each accuracy build a so-called Pareto front. In Fig. 7 this is done for the Breguet example.

<sup>6</sup> Due to the optimisation of the coefficients, even exact models are left with a small numerical error.

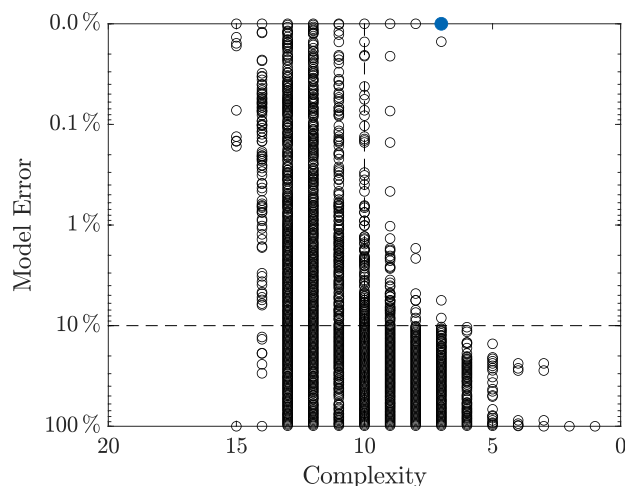


Fig. 7 Pareto analysis of Breguet example with noise-free data

Table 2 Pareto table of Breguet example with noise-free data

C <sup>1</sup>	Error [%]	Model	Model with fitted coefficients
1	100.000	$R = V$	$R = V$
2	99.956	$R = e^V$	$R = e^V$
3	23.723	$R = V^{c_1}$	$R = \frac{1}{V^{15/11}}$
4	23.722	$R = \sin(V)^{c_1}$	$R = \frac{1}{\sin(V)^{15/11}}$
5	15.163	$R = V^{c_1} M$	$R = \frac{M}{V^{59/46}}$
6	10.361	$R = V^{c_1} M^2$	$R = \frac{M^2}{V^{43/36}}$
7	0.000	$R = \frac{E \ln(M)}{V}$	$R = \frac{E \ln(M)}{V}$
8	0.000	$R = \frac{c_1 E \ln(M)}{V}$	$R = \frac{E \ln(M)}{V}$
9	0.000	$R = \frac{E \ln(M^{c_1})}{V}$	$R = \frac{E \ln(M)}{V}$
10	0.000	$R = \frac{c_1 E \ln(M)}{c_2 V}$	$R = \frac{E \ln(M)}{V}$
11	0.000	$R = V^{c_1} E \cos(V^E) \ln(M)$	$R = \frac{E \cos(V^E) \ln(M)}{V}$
12	0.000	$R = c_2 E \ln(M) (c_1 + \frac{1}{2V})$	$R = \frac{E \ln(M)}{V}$
13	0.000	$R = V^{c_1} E \ln(M) (M \cos(V))^{c_2}$	$R = \frac{E \ln(M)}{V}$
14	0.000	$R = V^{c_1} E \ln(M) (e^V \cos(V))^{c_2}$	$R = \frac{E \ln(M)}{V}$
15	0.000	$R = c_2 E + V^{c_1} E \cos(V^E) \ln(M)$	$R = \frac{E \cos(V^E) \ln(M)}{V}$

<sup>1</sup>Complexity: The number of nodes of the corresponding model tree

The correct model (Eq. 11), marked in blue, is clearly distinguishable on the edge of the Pareto front. Models with less complexity exhibit an error of at least 10%. On the other hand, there are numerous models with a higher complexity of 8–15, which were also perfect approximators with an error of 0%.

Further analysis of the models on the Pareto front (see Table 2) shows that the more complex models turn into the correct model by inserting the coefficients. For example, the coefficients  $c_1$  and  $c_2$  of the best model with complexity 10 are determined by the optimiser to 1. Therefore, it can be

simplified to the Breguet formula in the form Eq. 11. The same applies for exponents, as in the model of complexity 9, where the argument of the logarithm  $M^{c_1}$  is simplified to  $M$ . If the coefficients are fitted to zero, entire subterms could be deleted, which happens in the model of complexity 14 to the term  $(e^V \cos(V))^{c_2}$  or in the model of complexity 15 to the summand  $c_2 E$ . The models of complexity 11 and 15 are the only ones that cannot be simplified to the correct model, since both contain the term  $\cos(V^E)$ . The fact that these models also have a 0% error is because the cosine in this case can be approximated as 1, since  $V$  is a positive value much smaller than 1 and  $E$  is between 14 and 20, so  $V^E \approx 0$ .

### 4.1.2 Artificially noised data

In engineering, data is typically noised due to measurement errors. Therefore, it is essential that the SR algorithm performs even with noisy data. To test this, a series of runs with increasingly noisy data was conducted. To introduce artificial noise into the data, a random, uniformly distributed factor was applied to the output variable.

$$\bar{R} = R \cdot (1 + kU(-1, 1)) \tag{12}$$

The noise factor  $k$  between 0 and 100% determines the amplitude of the noise. The SR algorithm is run for a noise factor

of 10–40% in increments of 10%. The settings are identical to those used for the noise-free data.

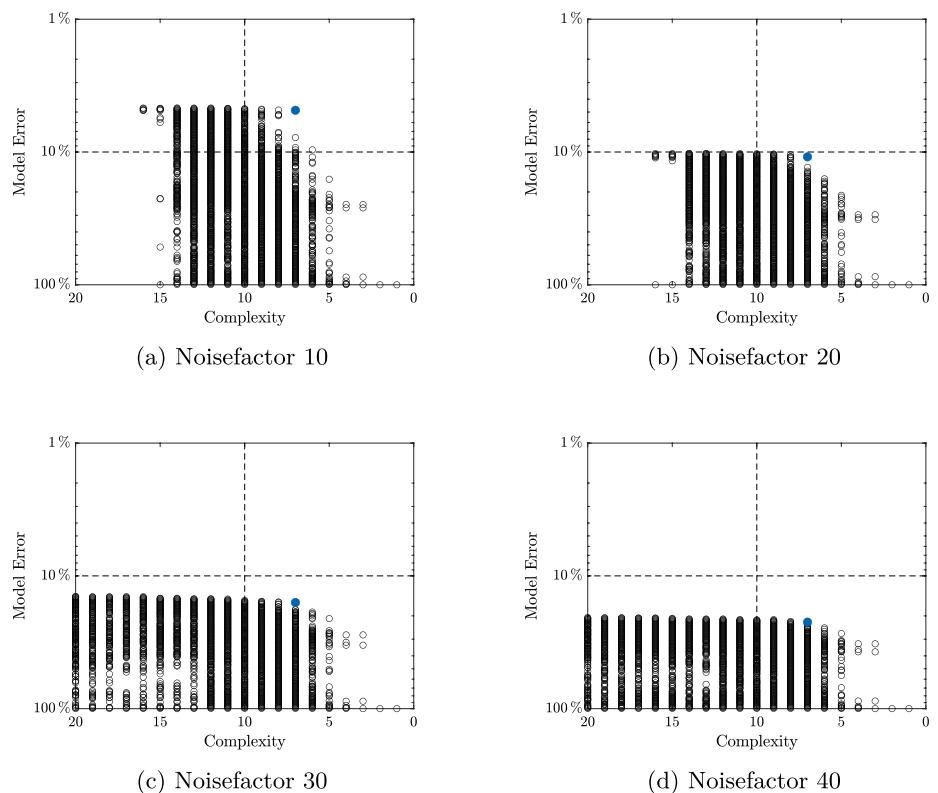
Figure 8 illustrates the Pareto analysis of runs with varying noise factors, from 10 to 40%. As expected, the Pareto front decreases as the noise factor increases. However, the optimal model, highlighted in blue, remains directly on the Pareto front. The SR algorithm appears to be highly robust against uniformly distributed noise. In comparison with the noise-free data, models with complexities higher than those of the correct one are now able to approximate the data more accurately. However, the impact is relatively minor, as evidenced by the low gradient of the Pareto front in this region.

Upon a more detailed analysis of the models on the Pareto front (see Table 3), as demonstrated by the noise factor of 20, it becomes evident that the correct model was once again reconstructed with a complexity of 7. In contrast with the noise-free data, models with a higher complexity now demonstrate a distinct overfitting. This renders them incapable of being transferred to the correct model; instead, the additional terms and parameters are employed to enhance the model’s ability to adapt to the noise.

### 4.2 Nguyen benchmark data set

To compare the performance of the proposed symbolic regression algorithm, the Nguyen benchmark data sets [24]

**Fig. 8** Pareto analysis for different noise factors



**Table 3** Pareto table of Breguet example with noise factor 20

$C^1$	Error [%]	Model	Model with fitted coefficients
1	100.000	$R = V$	$R = V$
2	99.954	$R = e^V$	$R = e^V$
3	29.704	$R = V^{c_1}$	$R = V^{-1.37}$
4	29.704	$R = \sin(V)^{c_1}$	$R = \sin(V)^{-1.37}$
5	21.182	$R = e^{c_1+V+M}$	$R = e^{V+M+6.14}$
6	15.927	$R = V^{c_1} M^2$	$R = \frac{M^2}{V^{6/5}}$
7	10.859	$R = V^{c_1} E \ln(M)$	$R = \frac{E \ln(M)}{V}$
8	10.487	$R = V^{c_1} E (c_2 + M)$	$R = \frac{E(M - \frac{8}{9})}{V^{0.93}}$
9	10.354	$R = V^{c_1} E (c_2 + 2M)$	$R = \frac{E(2M - \frac{2}{3})}{V^{0.81}}$
10	10.309	$R = V^{c_1} E (c_2 + M + \ln(M))$	$R = \frac{E(M + \ln(M) - 0.93)}{V^{0.84}}$
11	10.292	$R = V^{c_1} E (c_2 + 2M + \sin(M))$	$R = \frac{E(2M + \sin(M) - 2.84)}{V^{0.81}}$
12	10.290	$R = V^{c_1} \ln(M^3) (c_2 + E + M)$	$R = \frac{\ln(M^3)(E + M - 2.25)}{V^{0.82}}$
13	10.262	$R = V^{c_1} \ln(c_2 + M) (2E + \cos(E))$	$R = \frac{\ln(M - 0.03)(2E + \cos(E))}{V^{8/9}}$
14	10.249	$R = V^{c_1} \ln(c_2 + M) (3E + \ln(V))$	$R = \frac{\ln(M - 0.03)(3E + \ln(V))}{V^{5/6}}$
15	10.292	$R = E (\cos(V) \sin(V))^{c_1} (c_2 + 2M + \sin(M))$	$R = \frac{E(2M + \sin(M) - 2.84)}{(\cos(V) \sin(V))^{0.81}}$
16	10.289	$R = c_3 V + V^{c_1} \ln(M^3) (c_2 + E + M)$	$R = \frac{\ln(M^3)(E + M - 2.25)}{V^{0.82}} - 9V$

<sup>1</sup>Complexity: The number of nodes of the corresponding model tree

**Table 4** Nguyen benchmark data sets

Data set	$f(x)$	Boundaries	Number of data points
Nguyen 1	$x^3 + x^2 + x$	$[-1, 1]$	20
Nguyen 2	$x^4 + x^3 + x^2 + x$	$[-1, 1]$	20
Nguyen 3	$x^5 + x^4 + x^3 + x^2 + x$	$[-1, 1]$	20
Nguyen 4	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	$[-1, 1]$	20
Nguyen 5	$\sin(x^2) \cos(x) - 1$	$[-1, 1]$	20
Nguyen 6	$\sin(x^2) + \sin(x + x^2)$	$[-1, 1]$	20
Nguyen 7	$\ln(x + 1) + \ln(x^2 + 1)$	$[0, 2]$	20
Nguyen 8	$\sqrt{x}$	$[0, 2]$	20
Nguyen 9	$\sin(x) + \sin(y^2)$	$[-1, 1] \times [-1, 1]$	$10 \times 10$
Nguyen 10	$2 \sin(x) \cos(y)$	$[-1, 1] \times [-1, 1]$	$10 \times 10$
Nguyen 11	$x^y$	$[0, 1] \times [0, 1]$	$10 \times 10$
Nguyen 12	$x^4 - x^3 + \frac{1}{2}y^2 - y$	$[0, 1] \times [0, 1]$	$10 \times 10$
Nguyen 1 <sup>c</sup>	$3.39x^3 + 2.12x^2 + 1.78x$	$[-1, 1]$	20
Nguyen 2 <sup>c</sup>	$0.48x^4 + 3.39x^3 + 2.12x^2 + 1.78x$	$[-1, 1]$	20
Nguyen 5 <sup>c</sup>	$\sin(x^2) \cos(x) - 0.75$	$[-1, 1]$	20
Nguyen 7 <sup>c</sup>	$\ln(x + 1.4) + \ln(x^2 + 1.3)$	$[0, 2]$	20
Nguyen 8 <sup>c</sup>	$\sqrt{1.23x}$	$[0, 2]$	20
Nguyen 9 <sup>c</sup>	$\sin(1.5x) + \sin(0.5y^2)$	$[-1, 1] \times [-1, 1]$	$10 \times 10$
Nguyen 10 <sup>c</sup>	$2 \sin(1.5x) \cos(0.5y)$	$[-1, 1] \times [-1, 1]$	$10 \times 10$

are used. To also test the integrated coefficient optimiser, extended Nguyen functions containing rational coefficients are used. These functions are found in [5] and [11]. For each of the functions in Table 4, a uniformly distributed data set is

generated between the given boundaries and with the given number of data points.

For each data set, the symbolic regression algorithm should then search until a model with error less than

**Table 5** Symbolic regression results of the Nguyen data sets

Data set	Found model	Duration in h:mm:ss.s	Number of models
Nguyen 1	$c_2 x(x^2 + x + c_1)$	24.2	4749
Nguyen 2	$(c_1 + x)(x^3 + x)$	47.4	8785
Nguyen 3	$(x^3 + x)(x^2 + x + c_1) + c_2 x$	2:45:01.3	968,337
Nguyen 4	$x(x^3 + c_2)(x^2 + x + c_1)$	30:29.8	162,782
Nguyen 5	$c_1 + \sin(x^2) \cos(x)$	20.2	262
Nguyen 6	$c_1 \sin(x(c_2 + x)) \cos(c_3 x)$	17:20.2	94,325
Nguyen 7	$\ln(c_1 + x^3 + (c_3 + x)^{c_2})$	1:41.8	10,532
Nguyen 8	$x^{c_1}$	0.1	14
Nguyen 9	$\sin(x) + c_1 \sin(c_3 y^2 + c_2)$	1:05.2	8754
Nguyen 10	$c_1 \cos(y) \sin(x)$	2.3	466
Nguyen 11	$x^y$	0.1	17
Nguyen 12	$x^{c_3} + c_2 x^3 + c_5 y^2 + c_1 y \cos((x + y)^{c_4})$	2:19:37.1	503,505
Nguyen 1 <sup>c</sup>	$c_3 x + c_1 x^2 (c_2 + x)$	13.9	1434
Nguyen 2 <sup>c</sup>	$c_3 x (c_1 + x) (c_2 + (c_4 + x)^2)$	10:27.7	43,806
Nguyen 5 <sup>c</sup>	$c_1 + \sin(x^2) \cos(x)$	27.0	2875
Nguyen 7 <sup>c</sup>	$\ln(c_1 + c_4 x^2 + (c_3 + x)^{c_2})$	2:20.1	11,607
Nguyen 8 <sup>c</sup>	$(c_2 x)^{c_1}$	0.6	77
Nguyen 9 <sup>c</sup>	$c_2 \sin(c_4 x) + c_1 \sin(c_3 y^2)$	2:06.3	13,181
Nguyen 10 <sup>c</sup>	$c_1 \cos(c_2 y) \sin(c_3 x)$	49.2	5510

$5 \times 10^{-5}\%$  is found. Like already mentioned above, the small tolerance is needed because of the numerical errors due to the optimisation of the coefficients. The duration is determined by measuring the delta between two system time stamps, the first at the start of the algorithm, and the second directly after the termination criterion is fulfilled. The models are restricted to six variable nodes for each variable and five coefficient nodes. Sine, cosine, the exponential, and the logarithm functions are used as transcendental functions; the nesting of these functions is not permitted. For the fitting of the coefficients, only the first optimisation run using the fitted coefficients of the parent model as initial values is executed. As strategy for penalty switching, the default strategy described in Sect. 3.2.1 is used. Furthermore, dimensional analysis was not used, since the data is already dimensionless.

Table 5 shows for each Nguyen data set the best model found and both the time and the number of models required until the desired accuracy is reached. For each of the 19 Nguyen data sets, a model with zero error can be found. With the exception of the model for Nguyen 12, the models are symbolic equivalent to their respective Nguyen function shown in Table 4. Even the model for Nguyen 6 can be transformed into the original Nguyen function by applying trigonometric relationships. The model found for Nguyen 12 is symbolically not exactly the same due to its cosine part. However, fitting the coefficients results in the cosine part always being  $\cos(1) = 0.54$ . Multiplicated with

the coefficient  $c_1 = -1.85$  this leads to the desired  $-1$  as coefficient.<sup>7</sup> If the search is continued, as was done in an additional test run for Nguyen 12, the symbolically correct model will be found. In this run, which was limited to the evaluation of 2 million models, the model searched for is generated at position 983257. Although alternative models with 0 errors are faster to find, when using Pareto analysis as in Sect. 4.1, the original Nguyen function would be preferred due to its lower complexity.

Looking at the running time, it is noticeable that 14 of the 19 models can be found in less than 3 min. The least complex functions (Nguyen 8, 10, 8<sup>c</sup>) are even discovered in less than a second. This is also to be expected, as these functions are already formed in the second or third depth level of the search tree.

The longest time is required for the polynomials Nguyen 3 and 12 at around 2–3 h. What may seem like a disadvantage at first glance turns out to be a major advantage of the tree search. The fact that models can still be discovered after a long runtime indicates that the search is scalable with more computational effort.

**Comparison to other SR algorithms** Comparison of different SR algorithms is a challenging task. The most common way is, as done in this section, to use benchmark data sets and study if the SR algorithms are capable of discovering the underlying function.

<sup>7</sup> Special case: If  $y = 0$ , the cosine part gets 1, but then the whole summand gets 0.

When comparing algorithms, the recovery rate is a commonly used metric. It indicates how often a target function is successfully identified over multiple independent runs. Since many algorithms rely on stochastic components, their performance may vary from one run to another. However, the methodology proposed in this paper is deterministic, meaning that repeated executions under identical hyperparameter settings will always yield the same result. As a consequence, the conventional definition of the recovery rate is not suitable in this context. Instead, a single successful identification of a benchmark function, such as one in the Nguyen data set, is considered equivalent to achieving a 100% recovery rate for that particular instance.

In addition to identifying the ground truth function, other aspects such as computational runtime and the approximation error of the best discovered model are also relevant performance indicators. These factors are not included in the present comparison.

However, Table 6 presents an attempt to benchmark our method against previously published symbolic regression algorithms. It summarises reported recovery rates on the Nguyen benchmark functions as achieved by various research efforts.

Although this comparison does not reflect a standardised benchmark under identical experimental conditions, it nonetheless highlights the strong performance of the systematic tree search. In this setting, the method successfully recovers all functions in the Nguyen suite, outperforming most of the other approaches.

This result underscores a key advantage of the systematic approach: in theory, it is guaranteed to recover any target model, provided certain conditions are met. Specifically, the design of the systematic tree search ensures that a model will be discovered unless one of the following three limitations applies:

- (1) the target function violates user-defined constraints (e.g., restrictions on maximum expression complexity, permitted operators, or nesting depth);
- (2) the function has not been generated due to computational constraints, such as limited runtime or memory capacity; or
- (3) the correct symbolic structure has been found, but the subsequent numerical optimisation step fails to converge to the correct parameter values.

## 5 Impact of heuristic

In Sect. 3.2.1 a heuristic was introduced that controls the search process and, more precisely, the next node to be expanded in every iteration. Of particular importance is the penalty parameter  $p$ , which weights the model complexity. In this section, the impact of this parameter towards the success of the search is investigated. This is done by comparing the duration needed to discover the Nguyen functions while using different strategies of changing  $p$  throughout the search process.

In total, 6 strategies are compared. The first 3 uses a fixed parameter throughout the entire search process, while the last 3 use dynamic values for  $p$ . Table 7 shows the values used for every strategy in detail. The switching points indicate the iterations after which the penalty is changed. Once a full episode has elapsed, the penalty schedule cycles back to its initial value, and the sequence is repeated.

For this study, the symbolic regression is executed for each strategy and Nguyen data set, as in Sect. 4.2 before, until a model with an error of less than  $5 \times 10^{-5}\%$  is found. However, the maximum runtime of the search is limited to 3000 s. Except for the penalty strategy, each search is executed with the identical settings. The limits for variables and coefficients are set to 6 and 3, respectively. As transcendental functions, sine, cosine, logarithm, and exponential functions are used, but nesting them is prohibited. In addition, the overall complexity was limited to 15. The optimisation process is only executed once for each model, with the starting values derived from the predecessor model.

**Table 6** Recovery rates of different SR algorithms for the Nguyen datasets

SR algorithm	Average recovery rate over the 12 Nguyen data sets (%)	Source
STS (Systematic tree search)	100.0	
GSR [25]	100.0 <sup>1</sup>	[25]
SPL [11]	94.5	[11]
NGGP [26]	91.4	[26]
DSR [5]	83.6	[5]
PQT [27]	75.2	[5]
Eureqa	73.9	[5]
GP [28]	60.1	[5]

<sup>1</sup>Only first 11 Nguyen data sets were used

**Table 7** Strategies for penalty switching

ID	Strategy	Penalty values	Switching points
1	Static Intermediate	1	–
2	Static Exploitation	0	–
3	Static Exploration	1000	–
4	Dynamic Intermediate	100, 10, 1, 0.1, 0.01	10, 20, 30, 40, 50
5	Dynamic Exploitation	50, 5, 1, 0.1, 0.01	6, 14, 24, 36, 50
6	Dynamic Exploration	200, 20, 1, 0.1, 0.01	14, 26, 36, 44, 50

The Graph 9 shows the duration needed until the Nguyen data set is discovered for each strategy. The bar reaching the top of the graphic at 3000 s indicates that the Nguyen function was not found yet.

It can clearly be seen that the orange-red-coloured dynamic strategies outperform the blue-green-coloured static strategies in general and especially in the long run.

There is no case after a runtime of 100 s that a model is found faster with a static strategy than with a dynamic. In fact, Nguyen models 3, 4, 6 and 7 are not even found with the static strategies within the specified maximum runtime, whereas the dynamic strategies find every model except for one case (Nguyen model 7 with strategy 6).

For models which in general are found faster (in less than 100 s), it depends on the specific case. Nevertheless, in many cases the static strategies need significantly more time or are not able to discover the model (for example, Nguyen 2 and 5<sup>c</sup>). On the other hand, except for the very fast found model 8 there is no case in which the dynamic strategy needs significantly more time.

The difference between the dynamic strategies is marginal. The optimal strategy depends on the specific data set. Strategy 5 could be argued to be the best in the long run, as it performs best with Nguyen data sets 3, 4, 6 and 7. However, this thesis would require further testing with additional data sets to confirm it.

In conclusion, it can be stated that the variation of the penalty parameter at runtime, especially for longer searches, leads to the discovery of the correct model significantly faster than the use of a static value. This makes the variation of the penalty an essential aspect of the tree search algorithm. The exact variation strategy seems to be of secondary importance. In the future, it is also possible to use an adaptive penalty parameter that changes depending on the success of the search.

## 6 Summary and prospects

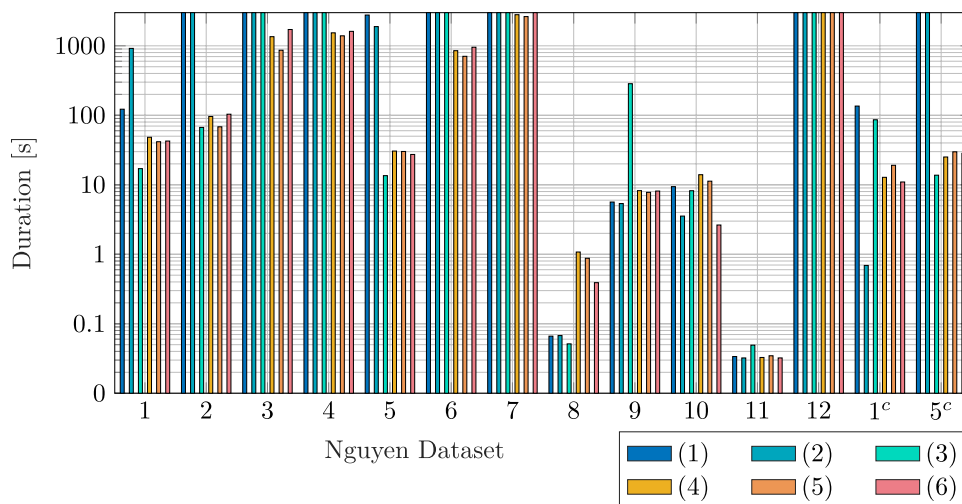
The discussed approach has been successfully applied to different examples; nevertheless, different enhancements could be implemented in the future.

### 6.1 Summary

This paper presents a deterministic algorithm for symbolic regression based on a tree search that systematically scans the entire search space of eligible symbolic models. Therefore, new model variants are iteratively developed, optimised, and evaluated from models that have already been examined. A dynamic heuristic was implemented that switches between exploration and exploitation during the tree search, and at the same time counteracts model overfitting. In addition, a method was presented to recognise the models that are created multiple times during the tree search and are, therefore, redundant to each other. This step is crucial when implementing symbolic regression with a tree search, especially to perform longer searches. Furthermore, dimension analysis was used to simplify the data set and ensure that each model satisfies the necessary condition of dimensional homogeneity. The result is a Pareto diagram densely packed with dimensionally homogeneous models, which allows a well-informed decision and trade-off between model accuracy and simplicity.

This is demonstrated by successfully reconstructing the physical formula from a synthetic data set created from Breguet's range formula. The algorithm is also able to find the correct correlation even with data with different noise levels, showing the robustness of the approach. In addition, the competitiveness of the tree search with other SR algorithms was demonstrated by successfully finding each of the 19 Nguyen Benchmark data sets used.

**Fig. 9** Computing time required per strategy



## 6.2 Prospects

With both the increasing dimension of the measurement data and the additional provision of transcendental functions, the number of possibilities increases immensely. To continue to find good models in acceptable computing time, it will be necessary to select nodes to be expanded and modifications to be selected even more intelligently. For example, in this study, all modifications are applied to the model to be expanded. In the future, reinforcement learning will be utilised to determine at runtime which modifications are most effective for each model. A Q-learning agent could offer a probability distribution for the optimal modifications based on the chosen model. As runtime increases, the algorithm should be able to make increasingly better decisions in this way.

As stated in Sect. 3.2.4, it is crucial to find the global optimal values for the model coefficients, especially for simple models. In the future, a better guess for the initial values should be implemented, or a global optimisation algorithm should be used. Since optimisation is the most time-consuming part, an algorithm with both a high probability of finding the global optimum and good performance is necessary.

In addition, parallelisation could significantly improve performance, since the tree search is currently running on a single core. However, many parts could be highly parallelised, such as the optimisation of all new models. Another idea would be to run the search in parallel on different paths, although it must be ensured that duplicates are recognised. It is also possible to perform exploration and exploitation simultaneously. Duplicates should not occur, because the exploration path of the models contains simpler models, while the exploitation path contains more complex ones.

After implementing the mentioned extensions the capabilities could be tested on more complex examples, like other benchmark data sets or by modelling physical relationships in higher dimensions.

**Acknowledgements** This work was supported by the Federal Ministry for Economic Affairs and Climate Action as part of the projects “Künstliche Intelligenz Europäisch Zertifizieren unter Industrie 4.0” (KIEZ4-0) and “Erschließung von Ressourcen- und Energieeffizienzpotenzialen für faserverbundkeramische Hochtemperatur-Leichtbausysteme durch digitale Transformation bestehender Fertigungsstrukturen” (CERAHEAT4.0).

Supported by:



Federal Ministry  
for Economic Affairs  
and Climate Action

on the basis of a decision  
by the German Bundestag

**Author contributions** S.R. had the original idea for the approach, M.A. detailed and expanded it. M.A. designed and implemented the algorithms. All authors reviewed and discussed the approach. All authors designed and reviewed the examples. M.A. and M.N. wrote the main manuscript text. All authors reviewed the manuscript.

**Funding** Open Access funding enabled and organized by Projekt DEAL.

**Data availability** No data sets were generated or analysed during the current study.

## Declarations

**Conflict of interest** The authors declare that they have no conflict of interest.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## References

1. Dittus, F.W., Boelter, L.M.K.: Heat transfer in automobile radiators of the tubular type. *Int. Commun. Heat Mass Transf.* **12**(1), 3–22 (1985). [https://doi.org/10.1016/0735-1933\(85\)90003-X](https://doi.org/10.1016/0735-1933(85)90003-X)
2. Cybenko, G.: Approximation by superpositions of a sigmoidal function. *Math. Control Signals Syst.* **2**(4), 303–314 (1989). <https://doi.org/10.1007/BF02551274>
3. Koza, J.R.: Genetic programming as a means for programming computers by natural selection. *Stat. Comput.* **4**, 87–112 (1994)
4. Rudolph, S.: On topology, size and generalization of non-linear feed-forward neural networks. *Neurocomputing* **16**(1), 1–22 (1997). [https://doi.org/10.1016/S0925-2312\(96\)00059-8](https://doi.org/10.1016/S0925-2312(96)00059-8)
5. Petersen, B.K., Landajuela, M., Mundhenk, T.N., Santiago, C.P., Kim, S.K., Kim, J.T.: Deep symbolic regression: recovering mathematical expressions from data via risk-seeking policy gradients. *ICLR (2021)* <https://doi.org/10.48550/arXiv.1912.04871>
6. Cranmer, M., Sanchez-Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel, D., Ho, S.: Discovering symbolic models from deep learning with inductive biases (2020). <https://doi.org/10.48550/arXiv.2006.11287>
7. Kamienny, P.-A., d’Ascoli, S., Lample, G., Charton, F.: End-to-end symbolic regression with transformers (2022). <https://doi.org/10.48550/arXiv.2204.10532>
8. Angelis, D., Sofos, F., Karakasidis, T.E.: Artificial intelligence in physical sciences: symbolic regression trends and perspectives. *Arch. Comput. Methods Eng.* **30**(6), 3845–3865 (2023). <https://doi.org/10.1007/s11831-023-09922-z>
9. Udrescu, S.-M., Tegmark, M.: AI Feynman: a physics-inspired method for symbolic regression. *Sci. Adv.* **6**(16), 2631 (2020). <https://doi.org/10.1126/sciadv.aay2631>

10. Rivero, D., Fernandez-Blanco, E., Pazos, A.: Dome: A deterministic technique for equation development and symbolic regression. *Expert Syst. Appl.* **198**, 116712 (2022). <https://doi.org/10.1016/j.eswa.2022.116712>
11. Sun, F., Liu, Y., Wang, J.-X., Sun, H.: Symbolic physics learner: discovering governing equations via Monte Carlo tree search (2023). <https://doi.org/10.48550/arXiv.2205.13134>
12. Olivetti de França, F.: A greedy search tree heuristic for symbolic regression. *Inf. Sci.* **442–443**, 18–32 (2018). <https://doi.org/10.1016/j.ins.2018.02.040>
13. Virgolin, M., Pissis, S.P.: Symbolic regression is NP-hard. [arxiv:2207.01018](https://arxiv.org/abs/2207.01018) (2022)
14. Russell, S.J., Norvig, P.: *Artificial Intelligence: A Modern Approach*, 3rd edn. Pearson Education, London (2016)
15. Walz, G. (ed.): *Lexikon der Mathematik: Band 2*. Springer, Berlin (2017). <https://doi.org/10.1007/978-3-662-53504-2>
16. Simon, V., Weigand, B., Gomma, H.: *Dimensional Analysis for Engineers*. Springer, Cham (2017). <https://doi.org/10.1007/978-3-319-52028-5>
17. Görtler, H.: *Dimensionsanalyse*. Ingenieurwissenschaftliche Bibliothek. Springer, Berlin (1975)
18. *Applied Dimensional Analysis and Modeling*. Elsevier (2007). <https://doi.org/10.1016/B978-0-12-370620-1.X5000-X>
19. Taylor, M., Diaz, A.I., Jodar-Sanchez, L.A., Villanueva-Mico, R.J.: A matrix generalisation of dimensional analysis: new similarity transforms to address the problem of uniqueness. (2008). <https://api.semanticscholar.org/CorpusID:4492811>
20. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, Reading (1974)
21. Aragón, F.J., Goberna, M.A., López, M.A., Rodríguez, M.M.L.: *Nonlinear Optimization*. Springer undergraduate texts in mathematics and technology, Springer, Cham (2019). <https://doi.org/10.1007/978-3-030-11184-7>
22. Nocedal, J., Wright, S.J.: Trust-region methods. In: *Numerical Optimization*. Springer series in operations research and financial engineering, pp. 66–100. Springer, New York (2006). [https://doi.org/10.1007/978-0-387-40065-5\\_4](https://doi.org/10.1007/978-0-387-40065-5_4)
23. Bender, B., Göhlich, D. (eds.): *Dubbel Taschenbuch Für Den Maschinenbau*. Springer, Berlin (2020). <https://doi.org/10.1007/978-3-662-59715-6>
24. Uy, N.Q., Hoai, N.X., O’Neill, M., McKay, R.I., Galván-López, E.: Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genet. Program Evolvable Mach.* **12**(2), 91–119 (2010). <https://doi.org/10.1007/s10710-010-9121-2>
25. Tohme, T., Liu, D., Youcef-Toumi, K.: GSR: A Generalized Symbolic Regression Approach (2023). [arxiv:2205.15569](https://arxiv.org/abs/2205.15569)
26. Mundhenk, T.N., Landajuela, M., Glatt, R., Santiago, C.P., Faisol, D.M., Petersen, B.K.: Symbolic Regression via Neural-Guided Genetic Programming Population Seeding (2021). [arxiv:2111.00053](https://arxiv.org/abs/2111.00053)
27. Abolafia, D.A., Norouzi, M., Shen, J., Zhao, R., Le, Q.V.: Neural Program Synthesis with Priority Queue Training (2018). [arxiv:1801.03526](https://arxiv.org/abs/1801.03526)
28. Fortin, F.-A., Rainville, F.-M.D., Gardner, M.-A., Parizeau, M., Gagné, C.: DEAP: evolutionary algorithms made easy. *J. Mach. Learn. Res.* **13**(70), 2171–2175 (2012)

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.