Evol. Intel. (2015) 8:149–164
DOI 10.1007/s12065-015-0126-x

RESEARCH PAPER

A hybrid evolutionary algorithm for the symbolic modeling of multiple-time-scale dynamical systems

Theodore W. Cornforth · Hod Lipson

Received: 18 August 2014 / Revised: 29 November 2014 / Accepted: 13 January 2015 / Published online: 31 January 2015
© Springer-Verlag Berlin Heidelberg 2015

Abstract Natural and artificial dynamical systems in the real world often have dynamics at multiple time scales. Such dynamics can contribute substantially to the complexity of a dynamical system and increase the difficulty with which it can be analyzed. Although evolutionary algorithms have been proposed that are amenable to the automated modeling of dynamical systems, none have explicitly taken into account multiple time scales or leveraged the information about these dynamics that is inherent in experimental observations. We propose a hybrid approach to the design of models for multiple-time-scale dynamical systems that combines an evolutionary algorithm with other metaheuristics and conventional nonlinear regression. With only minimal human-supplied domain knowledge, the algorithm automates the process of analyzing raw experimental observations and creating an interpretable symbolic model of the system under study. We describe the algorithm in detail and demonstrate its applicability to a variety of both physical and simulated systems. In addition, we study the performance and scalability of the algorithm under different types of dynamics, varying levels of experimental noise, and other factors relevant to the practical application of the algorithm.

T. W. Cornforth (✉)
Biological Statistics and Computational Biology, Cornell University, 1198 Comstock Hall, Ithaca, NY 14853-2601, USA
e-mail: twc63@cornell.edu

H. Lipson
Mechanical and Aerospace Engineering, Cornell University, 239 Upson Hall, Ithaca, NY 14853-7501, USA

Keywords Dynamical systems · Multiple time scales · Evolutionary computation · Genetic programming · Modeling · Symbolic identification

1 Introduction

The mathematical modeling of dynamical systems plays a central role in science, yet it is increasingly difficult for our human ability to construct models to keep pace with the growth in complexity of the systems we study [11, 47, 48]. A common feature of many dynamical systems that contributes to their complexity is the presence of simultaneous dynamics at multiple time scales. Dynamical systems with multiple time scales are those in which different patterns of behavior predominate when the system is viewed at different temporal scales [23]. For example, the forced Van der Pol oscillator is a theoretical system used to model many natural and synthetic oscillating systems (Fig. 1). The relatively fast self-oscillations and the slower forced oscillations are both readily apparent in the measurements for the dynamic variable x as shown in Fig. 1A. The frequency spectrum of these data confirms the qualitative observation that the dynamics in the system consist predominantly of those at two different time scales (Fig. 1B).

Phenomena with multiple time scales arise in many different areas of science and engineering (Figs. 2, 3). Some examples include: bursting activity in neurons [18], the reaction kinetics of enzyme catalysis [19], the motion of the cytoskeleton within cells [14], and the motion of flapping structures [10]. Dynamics at multiple spatio-temporal scales are also frequently observed, such as in fluid mechanics [8, 27] and patterns of ecological diversity [6, 38]. The gap between our ability to collect data from complex systems such as these and our ability to distill that

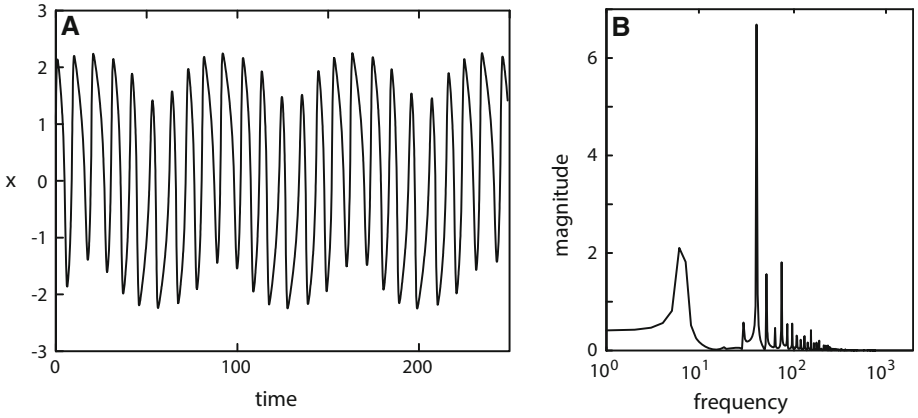


Fig. 1 A forced Van der Pol oscillator time series (**A**) and its corresponding frequency spectrum (**B**)

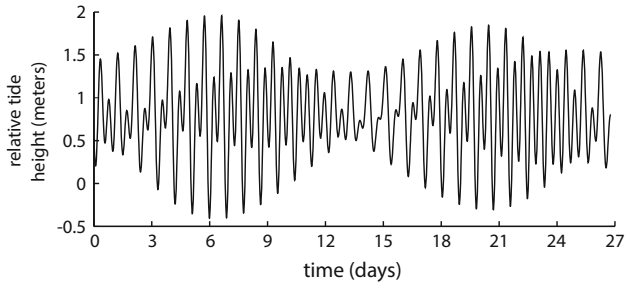


Fig. 2 A natural dynamical system with multiple time scales. These data show the tide levels recorded at a beach over the course of approximately one month. Data from http://tidesandcurrents.noaa.gov

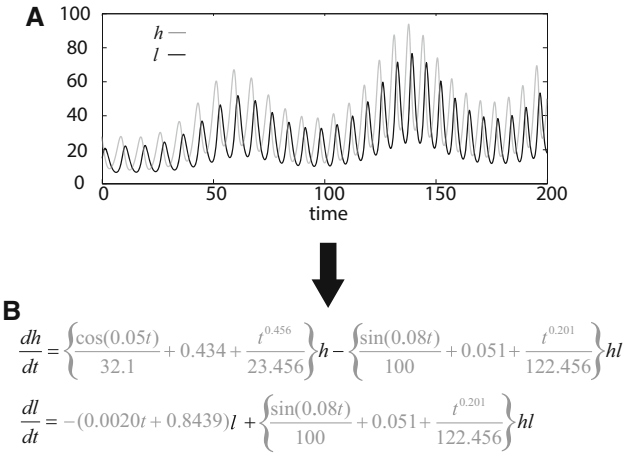


Fig. 3 Reverse-engineering multiple-time-scale dynamical systems. **A** A simulation of a predator–prey system in which hare (h) and lynx (l) populations display fast oscillations as well as slower overall changes in the magnitude of those oscillations. While the faster oscillations represent short term cycles in food supply, the slower changes might represent, for example, gradual evolutionary changes. **B** The goal of the present work is to supply data such as that shown in panel A to an algorithm that can automatically infer an algebraic model of the system. The model here is equivalent to a classic Lotka–Volterra predator–prey dynamical system except that the constant coefficients of the dynamic variables are replaced by complex time-dependent functions (*gray*). These time-dependent coefficients are responsible for the slow changes in population size

data into useful scientific models is rapidly widening. Practitioners frequently note that automation will be of increasing importance to close this gap by assisting scientists in the creation of parsimonious, interpretable models [16, 25, 51].

In fact, a wide variety of methods have been proposed for automating the modeling process. Familiar techniques such as linear and nonlinear regression are relatively easy to apply to empirical data, yet they assume that the structure of the model is substantially fixed in advance [15]. More powerful techniques such as support vector machines and neural networks can approximate arbitrary functions to an arbitrary degree of accuracy, yet they tend to produce models that are difficult to interpret [4]. To avoid such restrictions, "robot scientist" approaches related to expert systems can use inductive reasoning over databases of human-supplied domain knowledge [26, 30]. In the present work, we consider genetic programming, an evolutionary algorithm that can represent models as human-interpretable algebraic expressions and that places few constraints on the form of those expressions [28]. In addition, genetic programming requires minimal human-supplied domain knowledge and automatically designs models de novo directly from empirical data.

Different varieties of genetic programming have been successfully used to find explicit linear and nonlinear models [28, 29], implicit equations [43], and differential equations [5] for problems in many fields of science and engineering. In addition, many studies have focused specifically on applications of genetic programming to the analysis of time series data. The goals of these studies include forecasting [17, 35] as well as reverse-engineering explanatory models [5, 40]. As with other types of data, genetic programming is typically used to model time series with linear or nonlinear yet explicit algebraic equations

[22, 50], but recent advances have improved the capabilities of genetic programming for the modeling of time series as invariant equations [43] and ordinary differential equations [46].

Although these previously proposed genetic programming approaches are, in theory, suitable for application to multiple time scale systems, they do not take advantage of the fact that dynamics present at multiple time scales are potentially separable. This discards important information about the system that could be used to construct models with improved accuracy and explanatory power, while requiring less computational effort.

We describe a new method to leverage information about multiple time scales in genetic programming algorithms for symbolic modeling. The overall goal is to take measurements from a system, such as the observed hare and lynx population sizes shown in Fig. 3A, and then use this data to reverse-engineer an algebraic model for the system (Fig. 3B). The proposed method is a divide-and-conquer approach that decomposes dynamics at different scales in a system and models them independently. A complete model describing dynamics at all scales can then be recomposed from these independent models. Some preliminary results were previously published demonstrating this approach for simulated data [12]. Here we address modeling of both simulated and physical systems as well as investigate in more detail the factors that influence performance of the algorithm.

2 Methods

A summary of the proposed algorithm is shown in pseudocode in Fig. 4. The algorithm involves three main steps. In Step 1, an observed time series is divided into small windows such that slower dynamics in the data are approximately constant. The data in each window are modeled with genetic programming and the models obtained from several different windows are stored and ranked. In Step 2, the best models from Step 1 are considered one at a time. Any numerical constants in each model are replaced with "placeholder" variables. Ordinary nonlinear regression is then used to find values of the placeholder variables such that the expression fits the observed data for a small portion of the original data set. This process of finding values for the placeholder variables is repeated for several samples taken from different portions of the original data set. In Step 3, the values of the placeholder variables found in Step 2 are processed with an additional round of genetic programming. As the fitted values of the placeholder variables are time-dependent, they represent time series data sets that can themselves be modeled with genetic programming. The models resulting from this step are finally substituted back into the corresponding placeholder variables from which they were derived. This three step process of separating and independently modeling dynamics at different time scales can be repeated as needed for data with more than two time scales or with additional spatio-temporal scales.

In the next three subsections, we illustrate the algorithm informally using the multiple-time-scale dynamical system shown in Fig. 3. For the purposes of this explanation, it will be assumed that both the hare (h) and lynx (l) data are measured and that the task of finding a model $dh/dt = f(h, l, t)$ can be considered separately from the task of finding a model $dl/dt = f(h, l, t)$. In other words, the modeling problem is partitioned and numerical solution of the system of differential equations is not necessary as part of the modeling process [5]. Also for purposes of illustration, we will consider only the task of using the observed hare and lynx data to find an algebraic model for dl/dt as a function of h, l, and t.

2.1 Algorithm Step 1

The algorithm begins by numerical calculation of the time derivative of the observed lynx data. These time derivative data are divided into small windows as shown in Fig. 5A. The windows may or may not overlap, but their size is chosen such that all but the fastest dynamics are approximately constant over the time span of the window. We found simple inspection of the data to be sufficient for choosing the size of the window in this example and in the examples presented in the Results section below. However, more sophisticated methods for identifying an appropriate window size are considered in the Discussion section below.

In the example shown in Fig. 5, one of these windows consists of the data between the dashed lines. Within this window, the time-dependent coefficients of the l and hl terms are roughly constant, and so the complex ground truth model can be approximated by a much simpler model (Fig. 5B). A genetic programming algorithm can then be used to easily find this relatively simple expression when given the dl/dt data from within the window. Repeating this process for each window in the original data results in several different candidate models of the simpler form shown in Fig. 5B. We refer to this set of candidate models as the "bag" of models produced by Step 1 of the algorithm.

2.2 Algorithm Step 2

In the next step of the algorithm, the bag of models generated in Step 1 is processed. Although models in the bag could be merged or otherwise processed in many different

Begin Algorithm_1(input **DATA**)

Step 1:

Divide **DATA** into a set **W** of (possibly overlapping) windows.

For each window in **W**, run genetic programming and add output candidate models to bag of models **B**.

Rank models in bag **B** and select a subset **M** of the best models. Denote the number of coefficients in model $\mathbf{M}_i$ as k_i.

Step 2:

For each model $\mathbf{M}_i$

Replace the k_i coefficients with k_i placeholder variables $\{c_1, c_2, c_3, ..., c_{k_i}\}$.

Divide **DATA** into a set **V** of (possibly overlapping) windows.

For each window $\mathbf{V}_j$

Fit model $\mathbf{M}_i$ to the data in the window with nonlinear regression.

Store fitted placeholder variables in row j, columns 1 ... k_i of a table $\mathbf{T}_i$.

Store the time value at the midpoint of window $\mathbf{V}_j$ in row j, column $(k_i + 1)$ of table $\mathbf{T}_i$.

End For

End For

Step 3:

For each table $\mathbf{T}_i$

For each column $q \in \{1, 2, 3, ..., k_i\}$ in table $\mathbf{T}_i$

Run genetic programming with column q as the dependent variable and the time value in column $(k_i + 1)$ as the independent variable.

Rank the candidate output models and choose the best to replace coefficient c_q in model $\mathbf{M}_i$.

End For

End For

Return **M**

End Algorithm_1

Fig. 4 Proposed algorithm pseudocode

ways, here we simply select the individual candidate models with lowest error without regard to the window from which they were obtained. Once a model is selected from the bag (Fig. 6A), all numerical constants in the model are replaced with placeholder variables $\{a, b, c, \ldots\}$. We call the resulting model a "general form expression" (Fig. 6B). The placeholder variables are then taken to be the adjustable parameters in a series of nonlinear regression problems in which the general form expression is locally fit to the *dI/dt* data at several different points (Fig. 6C). For these experiments, the size of the local windows for which these local nonlinear regression fits were performed was set to the size of the windows used in Step 1 of the algorithm. Again, more sophisticated methods for selecting the size of the window are considered in the Discussion section below.

We found the Levenberg–Marquardt algorithm [36] with random-restarts to be an efficient means of performing the large number of nonlinear regression runs that are necessary. To accelerate the rate of achieving good local fits, the initial estimates used at the start of each nonlinear regression run were set to the fitted values of the placeholder variables found with nonlinear regression in the

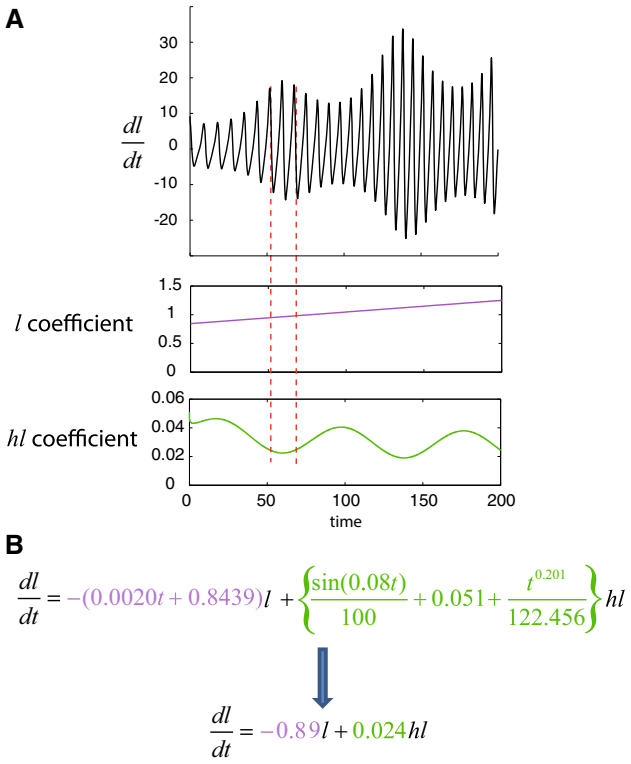


Fig. 5 Algorithm Step 1: Modeling fast dynamics. **A** The time derivative *dl/dt* is estimated from the original lynx data shown in Fig. 3. Windows are then used to divide *dl/dt* into smaller data sets over which slower components of the dynamics are approximately constant. **B** Genetic programming is used to find the relatively simple model that approximates the lynx dynamics within the window bounded by the *two dashed lines*. This is repeated for several windows, giving a bag of models

previous segment of the *dl/dt* data. The initial estimates for the first nonlinear regression run were randomly chosen. Overall, we found the computational effort required for these repeated nonlinear regression fits to be negligible compared to that required for portions of the algorithm involving genetic programming, especially on a per-CPU-core basis.

The result of performing local nonlinear regression for several windows spanning the observed *dl/dt* data can be visualized as a table as shown in Fig. 6D. Each row of this table contains the results of fitting the general form expression to one small window within the data. The first N columns of a given row in the table are the fitted values of the N placeholder variables and column $(N + 1)$ is the time point at the center of the window for which the local nonlinear regression was performed.

2.3 Algorithm Step 3

In Step 3 of the algorithm, the tables produced in Step 2 are processed with additional rounds of genetic programming as shown in Fig. 7. In a given table with $(N + 1)$ columns, each of the first N columns represent the value of a placeholder variable as a function of time, where time is given in column $(N + 1)$. Together these two columns constitute a new set of time series data that can be modeled with genetic programming just as the original data was in Step 1 (Fig. 7A). The resulting model can then be substituted into the general form expression where it replaces the corresponding placeholder variable wherever it appears

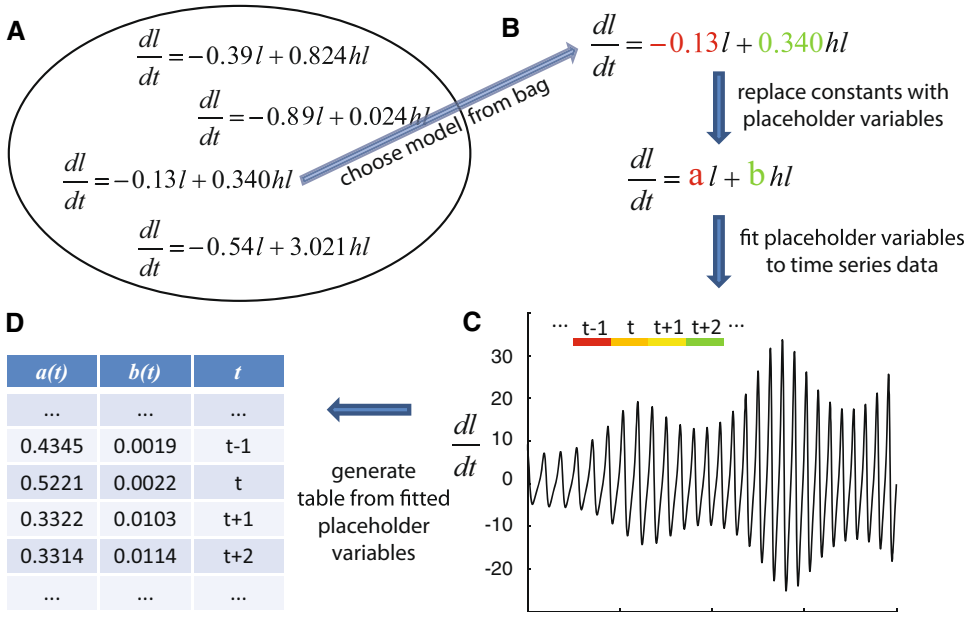


Fig. 6 Algorithm Step 2: Separating slow dynamics from the data. Expressions are taken from the bag of models obtained in Step 1 (**A**). The coefficients in these expressions are replaced with placeholder variables (a, b, c, etc.), resulting in general form expressions (**B**). Ordinary nonlinear regression is used to locally fit these placeholder variables at each of several locations (…, $t - 1$, t, $t + 1$,…) across the time span of the original data (**C**). The result of Step 2 is a table (**D**) that contains the fitted values for each placeholder variable as a function of time t

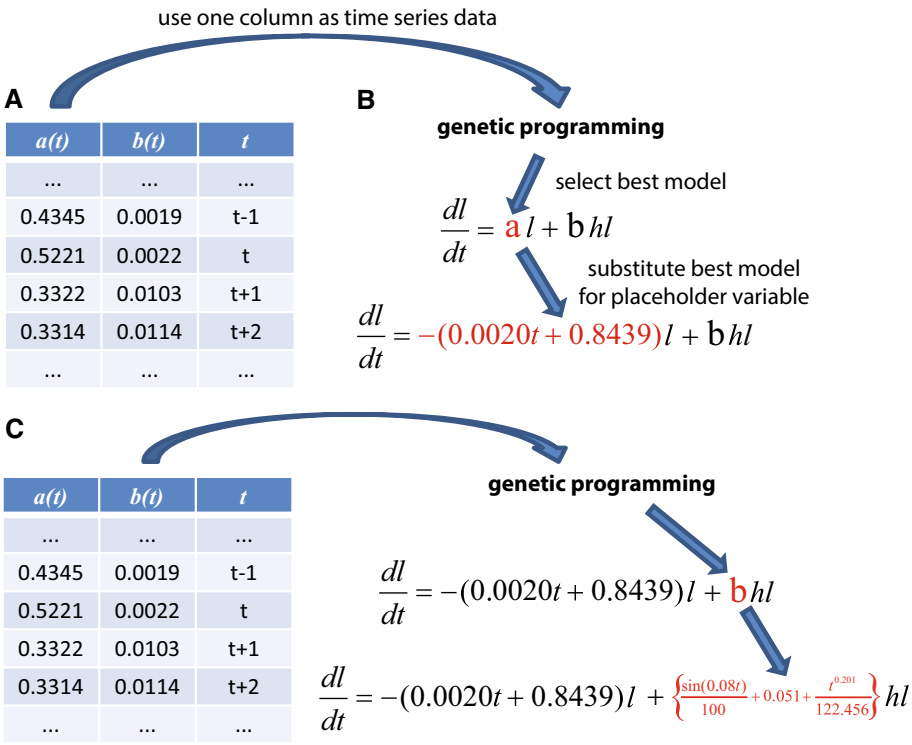


Fig. 7 Algorithm Step 3: Modeling slow dynamics and completing model inference. **A** As each column of a table obtained previously in Step 2 is a new set of time series data, it can be modeled with genetic programming. **B** The best model resulting from this genetic programming step is a model of the slow dynamics that is used to replace the corresponding placeholder variable in the general form expression. **C** Repeating this process for every column in the table results in the completed expression for *dl/dt*

(Fig. 7B). This procedure is repeated for additional columns in the table and finally results in a complete model for the observed lynx data (Fig. 7C).

2.4 Genetic programming for symbolic modeling

The algorithm proposed in this work relies on both nonlinear regression and genetic programming as sub-algorithms. In this section, we describe the genetic programming algorithm used in detail. Like other types of regression, genetic programming applied to symbolic modeling is concerned with the identification of a mathematical model of a system based on experimental observations of the system. However, genetic programming can identify both the algebraic structure and parameters of these models, and this distinguishes it from other methods such as linear and nonlinear regression that are capable only of optimizing the parameters of a model with a predefined structure [28]. The primary downside of genetic programming compared with other types of regression is the computational effort required for the search through the space of possible algebraic expressions. A population-based stochastic optimization technique has proven to be particularly effective means of performing this search, making genetic programming an evolutionary algorithm closely related to genetic algorithms [13, 28].

Genetic programming, as with all evolutionary algorithms, consists of numerous cycles called generations, which repeat until a desired performance goal or time limit is reached. The stopping criteria used here was a hard CPU time limit sufficient for the population to qualitatively achieve convergence for the specific modeling problem. However, the exact quantitative effects of total computational effort were examined for a subset of the test problems as described in more detail below. In each generation, the individuals that comprise a population of solutions to the problem reproduce, are selected to survive based on their fitness, and finally are pooled to form the population in the subsequent generation.

For symbolic modeling in this work we use linear genetic programming, which is distinguished by its use of a linear encoding [7]. With a linear encoding applied to this type of symbolic regression task, an individual member of the population corresponds to an algebraic expression represented as a list of operators and operands (Fig. 8).

id	type	value 1	value 2
id 1	variable	x	
id 2	variable	y	
id 3	add	id 1	id 2
id 4	constant	1.2345	
id 5	subtract	id 3	id 4

Fig. 8 Linear encoding for genetic programming. This individual from a population has a single chromosome of length five and represents the algebraic expression $f(x, y) = x + y - 1.2345$

Each entry or "gene" in the list is similar to a machine language instruction; registers are used to store intermediate values as the list is evaluated from first instruction to last. For example, the individual shown in Fig. 8 corresponds to the algebraic expression $f(x,\ y) = x + y - 1.2345$. Lists of machine language instructions such as these are equivalent to directed acyclic graphs, which have been shown to have advantages over the more traditional tree-based representation for genetic programming [41].

The possible types of entries in the machine language instruction list are drawn from the set of potential mathematical building blocks with which candidate algebraic models can be constructed. In each experiment described below, the building blocks included both operations and operands. The set of operands consisted of the measured variables in the system {x, y, t, etc.} and floating point constants [28]. The set of operations consisted of simple arithmetic functions {+, −, *, /}, trigonometric functions {sin, cos}, and exponentiation functions {exp, ^}. Individuals in the initial population were formed by randomly choosing operators and operands until lists with a length of 128 were formed. In practice, the vast majority of the lists or "chromosomes" constructed in this manner contained genes that performed no direct function, a frequently-observed phenomenon both in genetic programming chromosomes as well as in biological chromosomes [28].

Both crossover and mutation were used in our genetic programming algorithm to generate novel individuals during the reproduction process. Each generation, reproduction proceeded by selecting two parents from the population of 64 at random. With 0.75 probability, these parents generated two offspring by one-point crossover of their linear genomes at a randomly determined location. Each instruction in each offspring was then mutated with a probability of 0.01. If mutated, an instruction was replaced with a randomly chosen instruction, or with a new value in the case of instructions that stored floating point constants. Reproduction continued as described above until 64 children were generated.

After reproduction, models in the pooled population of parents and offspring were selected to survive on the basis of a fitness objective that measured the difference between the behavior predicted by the model and that observed in the target system. In this work we defined fitness -or more precisely, the error- as the mean absolute error (MAE):

$$\mathrm{MAE}(f) = \frac{1}{N}\sum_{i=1}^{N} |f(\mathbf{x}_i) - y_i|$$

where f is a candidate model, $\mathbf{x}_i$ and $\mathbf{y}_i$ are the independent and dependent variables in the training data, respectively, and N is the number of examples in the training data.

Evolutionary algorithms are traditionally applied to single-objective optimization problems, with the maximization of fitness or minimization of error as the sole objective. Instead, we implement survival selection as a multiobjective optimization problem, with both MAE and genotypic age as competing objectives to be minimized [45]. This Age-Fitness Pareto optimization approach has been shown to compare favorably with other powerful selection methods that reduce solution bloat and promote population diversity, such as Deterministic Crowding [34] and ALPS [20]. In this work, the Age-Fitness Pareto approach was implemented by first pooling all parents with all the offspring generated during reproduction. This combined population was then culled with a "Pareto tournament" scheme, in which two members of the combined population were selected at random to compete. If one of the pair had both higher age and higher MAE than the other, it was discarded. The survivor was then returned to the pool. This continued until the population size was reduced back to 64. We defined age as the number of generations in which an individual's oldest genes had been present in the population. Offspring age was always set to the parent's age, and offspring inherited the age of the older parent in the case of crossover. In order to ensure a steady flow of diversity into the population, a new solution was added to the population each generation. This new individual was assigned an age of zero and was randomly generated in the same manner as all members of the initial population.

To improve the capability of genetic programming to evolve models with floating point constants, many authors have embedded local search algorithms within a standard genetic programming framework to create a type of memetic algorithm [21]. Here, we use simple hill climbing to perform this local search [39]. Each generation, one hill climbing step was performed whereby all floating point constants in each individual were mutated with a probability of 1.0. The error of each individual was then reevaluated and if the error increased, the old parameter values were restored.

As a further variation on standard genetic programming, we used coevolved rank predictors as part of a meta-algorithm to accelerate evaluation of candidate models. Predictors are a subset of the full training data set and are represented by a linear chromosome of integers, where the integers are indices to points in the training data set. For example, a predictor of length three might be {87, 349, 817}, where the integers refer to the 87th, 349th, and 817th time points in a data set. Predictors can be used as a substitute for evaluating the fitness of a candidate model with the full training data set, thereby greatly accelerating fitness calculations. While the subset of points specified by a predictor can be selected randomly, recent research has shown that coevolving predictors can significantly outperform random selection with minimal computational overhead [42]. In the present work, predictors were coevolved

by attempting to minimize the difference between the relative fitness or rank of the solutions calculated using the full set of training data points and the rank of the same solutions calculated using only the data points specified by the predictor [44]. The predictors had a length of 128 and the size of the predictor population was 16. Predictor evolution was performed with a simple genetic algorithm employing deterministic crowding selection [34], 0.1 point mutation probability, and 0.5 crossover probability. For full details, we refer the reader to our previous work [42, 44].

To evaluate algorithm performance in as unbiased a manner as possible, some experiments described below used random functions as targets. To generate these random functions, lists of operations and operands were

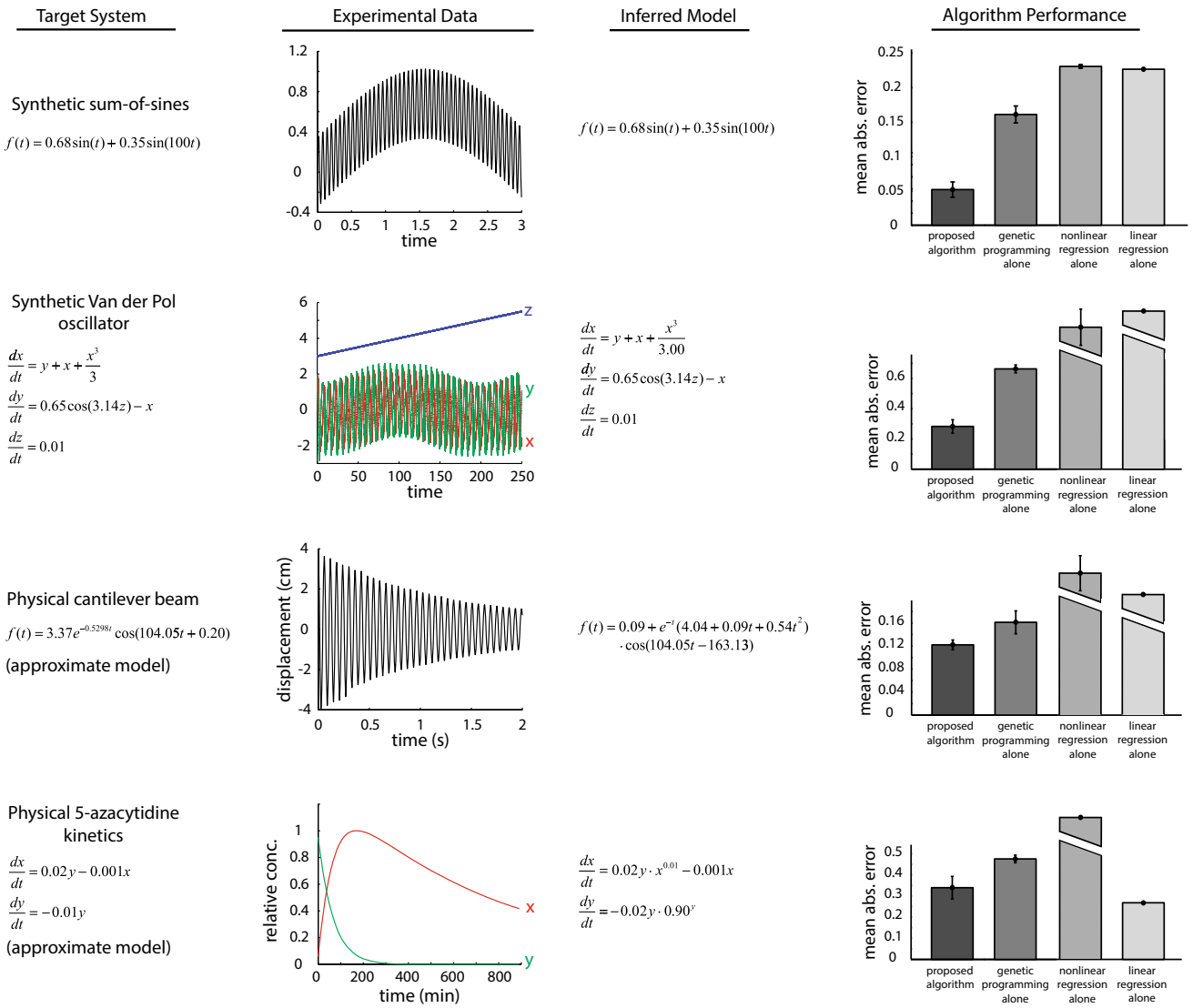


Fig. 9 Four of the multiple-time-scale systems used to test the proposed algorithm. The *first column* shows the target systems, which are only approximate theoretical models in the case of the two physical (non-simulated) systems: the cantilever beam and the 5-azacytidine system. The *second column* shows the data measured from the corresponding system, which is the input to either the algorithm proposed in this work or to genetic programming alone. The *third column* shows the best model inferred by the proposed algorithm. Finally, the *fourth column* summarizes the results of 50 trials of the proposed algorithm and genetic programming alone. The *error bars* represent ± standard error of the mean for 50 independent trials of the algorithms. All differences between the proposed algorithm and genetic programming alone except for the 5-azacytidine kinetics problem are significant by an unpaired *t* test, $p < 0.05$. Values for the 5-azacytidine kinetics problem are multiplied by 1,000 for easier visual comparison with the other results. The *fourth column* also shows performance comparisons with ordinary linear and nonlinear regression. The *bars* with *horizontal whitespace* in between indicate that the associated MAE values are too large to draw on the same meaningful scale as the other results. Note that the *bars* indicating ± standard error of the mean for 50 trials of nonlinear regression are too small to be visible for two out of the four experimental systems and that there are no error bars for linear regression as in this case it is a deterministic procedure

generated at random as with the construction of individuals in an initial genetic programming population. Unlike with population initialization however, random lists were generated until the expression encoded by the list was mathematically valid. The expression was then symbolically simplified using MATLAB's Symbolic Math Toolbox to obtain an accurate measure of the random expression's complexity. Complexity was defined simply as the total number of operations and operands in the expression. This procedure was repeated until a large number of unique functions across a range of several complexities was obtained.

3 Results

To test the proposed algorithm, we applied it to several systems with multiple time scales. Three of these were synthetic systems with simulated data and two were physical systems.

3.1 Application to synthetic systems

Three synthetic systems were studied: the one-dimensional "sum-of-sines" test system, a more complex three-dimensional oscillating system, and randomly-generated functions with multiple time scales. The ground truth equations for the first two synthetic problems and a sample of data generated from them are shown in Fig. 9 (first and second rows).

50 independent trials of the proposed algorithm were run with the goal of reverse-engineering the target expressions. As standard genetic programming algorithms that do not take into account the presence of dynamics at multiple time scales are applicable to the same task, 50 independent trials of genetic programming alone were also run with the same targets as a control to gauge whether the proposed algorithm was able to obtain better models with the same amount of computational effort. All "genetic programming alone" results below were obtained with the algorithm described in Sect. 2.4 above.

In all trials for one of the modeling problems, the total number of candidate model evaluations was the same, regardless of whether the proposed algorithm or genetic programming alone was used. The ground truth equations were used to generate training data, which was used during model inference. These ground truth equations were also used to generate separate test data, which were not used during model inference but were used as the basis for all performance results reported below. Gaussian noise $\sim N(0, 0.01)$ was added to the training data for the Van der Pol oscillator problem and varying amounts of noise were added to the training data for the sum-of-sines problem as described below.

The best models produced in each of the trials (as gauged by performance on the in-sample training data) were pooled and ranked by MAE on the test data set. The target, best, and median models from this pool (as gauged by performance on the out-of-sample test data) from this pool are compared in Table 1. In addition, the best models produced by the proposed algorithm are shown in the third column of Fig. 9. For both synthetic problems, the proposed algorithm outperformed genetic programming alone. Although the inferred models with lowest error found with both algorithms closely matched the target models, modeling with the proposed algorithm appeared to be more robust. In particular, the structure of models with higher error tended to differ from the target to a greater extent with genetic programming alone as can be seen by comparing the median models in Table 1. This is also reflected in the MAE of the models produced by the proposed algorithm and by genetic programming alone as shown in the fourth column of Fig. 9.

In principle, both ordinary linear and nonlinear regression are applicable to the same modeling tasks studied here and can also serve as a baseline against which the performance of the proposed algorithm can be judged. For the two synthetic systems, we tested the performance of linear and nonlinear regression by specifying the structure of linear and nonlinear models that could reasonably be supposed to capture the dynamics of the system under study once parameter values were optimized. For the sum-of-sines problem, the linear model was taken to be a second order polynomial of the form:

$$f(t) = p_1 + p_2 t + p_3 t^2$$

and the nonlinear model was taken to be of the form:

$$f(t) = p_1 \sin(p_2 t + p_3) + p_4 \sin(p_5 t + p_6) + p_7$$

where in both cases, $\mathbf{p}$ is a parameter vector to be optimized. Similarly, for the Van der Pol oscillator problem, the linear model was:

$$\begin{aligned}
\frac{dx}{dt} &= p_1 + p_2 x + p_3 y + p_4 z + p_5 x^2 + p_6 y^2 + p_7 z^2 + p_8 x^3 \\
&\quad + p_9 y^3 + p_{10} z^3 \\
\frac{dy}{dt} &= p_1 + p_2 x + p_3 y + p_4 z + p_5 x^2 + p_6 y^2 + p_7 z^2 + p_8 x^3 \\
&\quad + p_9 y^3 + p_{10} z^3 \\
\frac{dz}{dt} &= p_1 + p_2 x + p_3 y + p_4 z + p_5 x^2 + p_6 y^2 + p_7 z^2 + p_8 x^3 \\
&\quad + p_9 y^3 + p_{10} z^3
\end{aligned}$$

and the nonlinear model was:

Table 1 Models produced by the proposed algorithm and by genetic programming alone for the two synthetic systems: the sum-of-sines system (**A**) and a Van der Pol Oscillator (**B**)

(A) Sum-of-sines
Target
f(t) = 0.67890 * sin(t) + 0.34560 * sin(100 * t)
Best, proposed algorithm
f(t) = 0.67890 * sin(t) + 0.34560 * sin(100 * t)
Median, proposed algorithm
f(t) = 0.67879 * sin(0.00011 + t) + 0.34580 * sin(99.98260 * t − 6.14881)
Best, symbolic regression alone
f(t) = 0.67890 * sin(t) + 0.34560 * sin(100 * t)
Median, symbolic regression alone
f(t) = sin(t) − 0.65618 * sin(t) * sin(−49.55108 * t) * sin(−49.55108 * t)
(B) Van der Pol oscillator
Target
dx/dt = y + x − x^3/3
dy/dt = 0.65430 * cos(3.14159 * z) − x
dz/dt = 0.01000
Best, proposed algorithm
dx/dt = y + x − x^3/3.00049
dy/dt = 0.65432 * cos(−3.14181 * z) − x
dz/dt = 0.00995
Median, proposed algorithm
dx/dt = y + x − x^3/3.00041
dy/dt = −0.65948 * cos(9.13489 − 3.06381 * z) − 0.99125 * x
dz/dt = 0.00997
Best, symbolic regression alone
dx/dt = y + 1.00084 * x − x^3/3.00147
dy/dt = 0.65309 * cos(−3.14095 * z) − x
dz/dt = 0.00996
Median, symbolic regression alone
dx/dt = y + 0.99894 * x − x^3/3.00302
dy/dt = z + 3.73327 * cos(z) − x
dz/dt = 0.01001

$$\frac{dx}{dt} = p_1 \sin(p_2 x + p_3) + p_4 \sin(p_5 y + p_6) + p_7 \sin(p_8 z + p_9) + p_{10}$$

$$\frac{dy}{dt} = p_1 \sin(p_2 x + p_3) + p_4 \sin(p_5 y + p_6) + p_7 \sin(p_8 z + p_9) + p_{10}$$

$$\frac{dz}{dt} = p_1 \sin(p_2 x + p_3) + p_4 \sin(p_5 y + p_6) + p_7 \sin(p_8 z + p_9) + p_{10}$$

For these comparisons, linear regression was performed by QR decomposition and nonlinear regression was performed with the Levenberg–Marquardt algorithm [36] with small uniform-distributed random parameters as starting points for each of 50 trials. Results from these trials are shown in the fourth column of Fig. 9. Under these models, both linear and nonlinear regression performed poorly on test data for both problems, and while they required considerably less computational effort than genetic programming or the proposed algorithm applied to the same task, they suffer from the same problem as all fixed-form modeling methods: their structure must be hand-designed and specified in advance.

Several factors could potentially influence the accuracy and efficiency of the proposed algorithm. These include two factors beyond the control of the investigator: the presence of noise in the training data and the magnitude of differences between time scales inherent in the system. In contrast, the choice of window size and the total amount of computational effort used are two factors under the control of the investigator likely to impact the performance of the algorithm.

To investigate the effects of these four factors, we used the sum-of-sines test problem shown above but systematically varied the influence of the four factors. As before, the performance of the proposed algorithm was compared

with that of genetic programming alone. 50 independent trials were performed for each and the same amount of computational effort used in all trials. A summary of these results is shown in Fig. 10. For relatively low noise levels (Fig. 10A) and a large difference between the two time scales (Fig. 10B), the proposed algorithm outperforms genetic programming alone. However, at higher noise levels and less separation between time scales, this advantage decreases. The impact of noise and separation between time scales is perhaps difficult to appreciate without visual inspection of the corresponding data; Fig. 11 illustrates these factors graphically for several different example time series. For the sum-of-sines problem with no noise and maximum separation between time scales, the advantage of the proposed algorithm over genetic programming alone is robust over a wide range of window sizes (Fig. 10C) and total computational effort (Fig. 10D).

Finally, to test the performance and scalability of the proposed algorithm in as unbiased a manner as possible, we generated hundreds of random target problems with two time scales and a wide variety of different complexities. An example of a random target problem is shown in Fig. 12A. The proposed algorithm outperforms genetic programming alone across a range of target complexities (Fig. 12B).

3.2 Application to physical systems

Two physical systems were studied: a vibrating cantilever beam modeled as a one-dimensional function of time [32], and the two-dimensional 5-azacytidine chemical reaction system [2, 3]. The application of the proposed algorithm to the vibrating cantilever beam was straightforward as the vibrations of the beam represent the faster component of the dynamics and the gradual decline of the magnitude of these vibrations due to damping represents the slower component of the dynamics. Data for this experiment was collected by manually displacing a simple steel beam and optically tracking the motion of its tip for several seconds. The first 2 s of motion tracking data from one such trial are shown in Fig. 9, row 3, column 2.

In contrast, the 5-azacytidine system displayed a distinctive type of behavior known as fast-slow dynamics [9, 37]. The "fast" phase dominates very early in the chemical reaction while the "slow" phase dominates later. This contrasts with the systems considered so far in which the

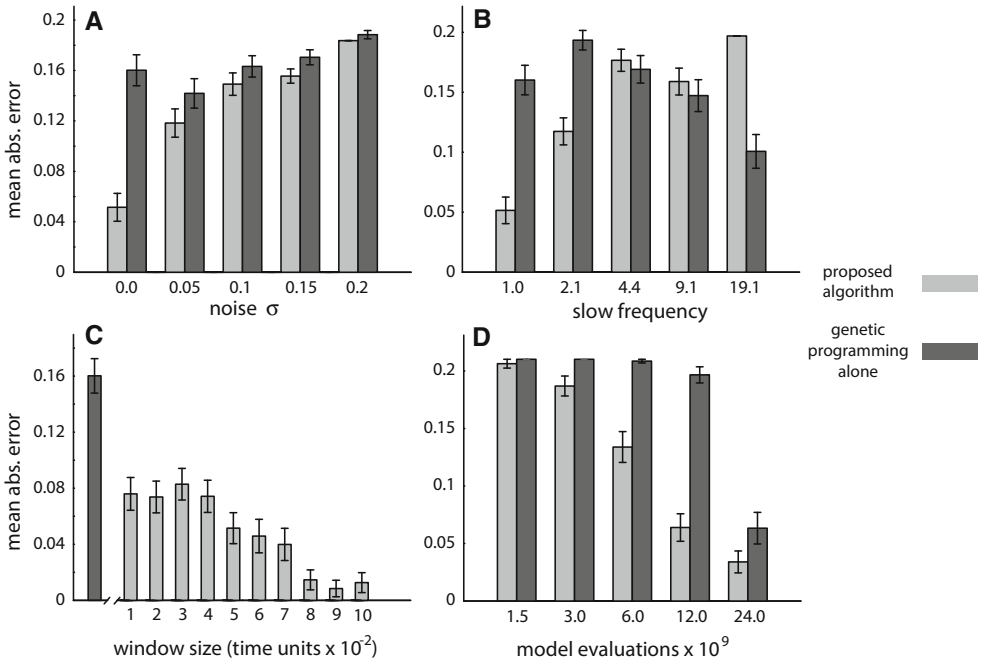


Fig. 10 Effects of noise, magnitude of time scale differences, window size, and computational effort on algorithm performance. The target for the sum-of-sines problem was changed from that shown in Fig. 9 to: $f(t) = 0.67890 \cdot \sin(\Omega \cdot t) + 0.34560 \cdot \sin(100 \cdot t) + N(0, \sigma)$ where Ω is the frequency of the slow component of the dynamics and σ is the standard deviation of Gaussian noise added to the signal. The effect of varying σ on the mean absolute error of the best generated models is shown in (**A**) and the effect of varying Ω is shown in (**B**). For the sum-of-sines problem with no noise and with maximum separation between time scales, the effect on mean absolute error of varying window size is shown in (**C**) and the effect of varying the amount of computational effort used is shown in (**D**). *Light gray bars* are results for the proposed algorithm, *dark gray bars* are for genetic programming alone, and the error bars are ± standard error of the mean for 50 independent trials of the algorithms

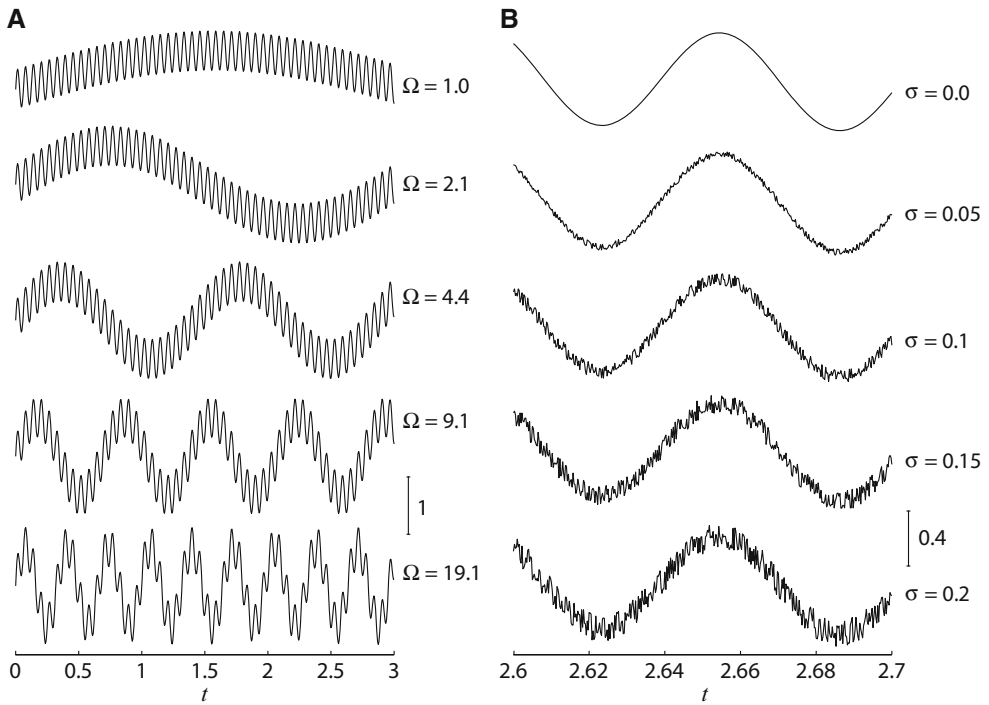


Fig. 11 Illustration of the effects of noise and magnitude of time scale differences in a target function. **A** Data generated from the sum-of-sines function given in Fig. 10's caption with different values of Ω, the frequency of the slow component in the dynamics. The scale of the *vertical axis* is given by the *bar* above and to the *right* of the *bottommost curve*. All curves share the same *vertical* and *horizontal scales*. **B** Data generated from the same sum-of-sines function with different values of σ, the standard deviation of Gaussian noise added to the signal. The scale of the *vertical axis* is again given by the *bar* above and to the *right* of the *bottommost curve*, and all curves share the same *vertical* and *horizontal scales*. The scale of the *horizontal axis* in this panel is smaller than that in panel **A** in order to show the fine details of how noise corrupts the signal

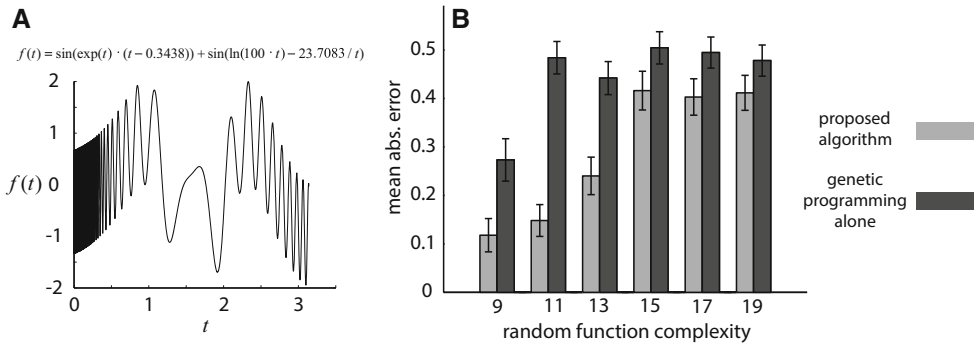


Fig. 12 Random target functions with multiple time scales. **A** An example of a random target function with a complexity of 17. Complexity is quantified as the number of operators and operands in a function. **B** The effects of varying random target function complexity on algorithm performance. *Light gray bars* are results for the proposed algorithm, *dark gray bars* are for genetic programming alone, and the *error bars* are ± standard error of the mean for 50 independent trials of the algorithms. The same number of model fitness evaluations was used in all trials. A unique random target function was generated for each trial within each complexity class

fast and slow dynamics are superimposed throughout the observed time series. However, a simple variation on our algorithm for reverse-engineering multiple-time-scale dynamical systems can be employed for fast-slow systems as well. Instead of using windows to decompose superimposed dynamics at different time scales, the data are decomposed into a very early phase when the fast dynamics are expected to dominate and a later phase when the slow dynamics are expected to dominate. A model of each of these components is inferred separately and then finally combined into a complete model. The raw data for these experiments was collected by Argemí and Saurina as part of their detailed kinetic analysis of this chemical reaction system [2].

Approximate theoretical models for the two physical systems derived from first principles are shown in Fig. 9, column 1. The experimental data from the systems are shown in Figure 9, column 2. Reverse-engineering was performed as for the synthetic systems with 50 independent trials and the same amount of computational effort was used for the proposed algorithm and genetic programming alone. As with the synthetic systems, the best models produced in each of the 50 trials were pooled and ranked by mean absolute error on the test data. The target, best, and median models obtained by the proposed algorithm and by genetic programming alone are compared in Table 2. Along with the theoretical models, the best models from 50 trials of the proposed algorithm are shown in Fig. 9, column 3. The proposed algorithm outperformed genetic programming alone for both physical systems as shown in Table 2 and Fig. 9, column 4).

We also compared the performance of the proposed algorithm and genetic programming alone with that of ordinary linear and nonlinear regression. For the physical cantilever beam system, the linear model was taken to be of the form:

$$f(t) = p_1 + p_2 t + p_3 t^2$$

and the nonlinear model was of the form:

$$f(t) = p_1 \sin(p_2 t + p_3) + p_4 \sin(p_5 t + p_6) + p_7$$

For the physical 5-azacytidine kinetics system, the linear model was:

$$\frac{dx}{dt} = p_1 + p_2 x + p_3 y + p_4 x^2 + p_5 y^2 + p_6 x^3 + p_7 y^3$$
$$\frac{dy}{dt} = p_1 + p_2 x + p_3 y + p_4 x^2 + p_5 y^2 + p_6 x^3 + p_7 y^3$$

and the nonlinear model was:

$$\frac{dx}{dt} = p_1 \sin(p_2 x + p_3) + p_4 \sin(p_5 y + p_6) + p_7$$
$$\frac{dy}{dt} = p_1 \sin(p_2 x + p_3) + p_4 \sin(p_5 y + p_6) + p_7$$

Table 2 Models produced by the proposed algorithm and by genetic programming alone for the two physical systems: a cantilever beam (**A**) and the 5-azacytidine reaction system (**B**)

(A) Cantilever beam
Approximate theoretical model
f(t) = 3.3672 * exp(−0.5298 * t) * cos(104.0514 * t + 0.19914)
Best, proposed algorithm
f(t) = (0.09382 + exp(−t) * (4.03615 + 0.09041 * t + 0.54199 * t^2)) * cos(104.0481 * t − 163.1311)
Median, proposed algorithm
f(t) = (0.46968 + 3.59458 * exp(−0.98795 * t)) * cos(−104.04810 * t − 31.18420)
Best, symbolic regression alone
f(t) = (1.47585 + 0.97938 * exp(0.99979 − 1.35988 * t)
Median, symbolic regression alone
f(t) = 2.57905/(0.58074 + t) * cos(104.52440 * t) + 0.30671 * cos((cos(0.01356/(t^2)) + 3.86792/t + 104.92515) * t
(B) 5-azacytidine reaction kinetics
Approximate theoretical model
dx/dt = 0.01871 * y − 0.00125 * x
dy/dt = −0.01490 * y
Best, proposed algorithm
dx/dt = 0.01799 * y * $x^{0.00946}$ − 0.00132 * x
dy/dt = −0.01628 * y * 0.90006^y
Median, proposed algorithm
dx/dt = 0.01778 * y − 0.00132 * x
dy/dt = −0.01695 * y * 0.82759^y
Best, symbolic regression alone
dx/dt = (0.12748 * y − 0.00847 * x)/(6.43765 + $y^{2.73063}$)
dy/dt = −0.01660 * y^2/(y − 7.33560) − 0.01660 * y
Median, symbolic regression alone
dx/dt = 0.01906 * y^2 + 0.01906 * x * y − 0.00139 * y − 0.00139 * x
dy/dt = 0.00283 * y^2 − 0.01684 * y

As with the synthetic systems, linear and nonlinear regression here required less computational effort but performed more poorly than the proposed algorithm or genetic programming alone. The exception was linear regression applied to the physical 5-azacytidine system, which performs as well or better than the proposed algorithm. The physical system in this case happens to be well approximated by a very simple linear model, although this would not be known in advance unless the experimenter possessed significant domain knowledge. In contrast, the proposed algorithm performs as well or nearly as well across several trials without the need for any such knowledge, and in fact, with the ability to generate models that convey that knowledge to the experimenter.

4 Discussion

We propose a hybrid evolutionary algorithm for the automated modeling of dynamical systems with multiple time scales. For both synthetic problems in which an exact underlying model is known and for physical systems with only approximate theoretical models, the algorithm consistently found models that matched the accuracy and parsimony of the targets. Although genetic programming is the core of our approach, our results suggest that the evolutionary algorithm proposed here can significantly outperform ordinary genetic programming with little additional computational effort. By improving the scalability of the automated modeling process in this domain, the algorithm proposed here helps to meet the challenge posed by the ever-increasing complexity of systems studied throughout science and engineering [11, 47, 48].

Most theoretical and computational work in the area of multiple-time-scale dynamical systems has focused on a technique known as singular perturbation theory and its related techniques [23, 49]. The goal in this body of work is to decompose models of multiple-time-scale systems into independent single-scale systems that approximate the behavior of one of the scales in the original system [9, 33]. Although singular perturbation and its extensions are powerful techniques, they are used for a fundamentally different purpose than the algorithm we propose here. Singular perturbation starts with a complete, hand-designed model and then proceeds to analyze and reduce that model, whereas the focus of the present work is on the de novo, automated design of the complete symbolic model using only empirical observations. This makes the algorithm proposed here more closely related to the divide-and-conquer curve fitting technique MFIT, in which high dimensional regression problems with several independent variables are reduced to a series of single-dimensional regression problems [31].

The algorithm we propose involves multiple interacting sub-algorithms and stages of analysis, and it is perhaps somewhat opaque as to how overall improvements in modeling efficiency over genetic programming could be gained. Although a detailed theoretical treatment of the proposed algorithm's performance is beyond the scope of this discussion, we present our hypothesized explanation for the performance gains of the algorithm under the general framework of divide-and-conquer problem-solving. Like any divide-and-conquer approach to computation, the algorithm proposed in the present work decomposes the problem at hand into smaller sub-problems, each of which are easier to solve than the original problem and which generate solutions that can be combined into a solution for the original problem. In the present case, the "original problem" is a complex symbolic regression problem and the "sub-problems" are much less complex symbolic regression problems. The novelty of the algorithm lies both in how these sub-problems are defined and in how the solutions to these sub-problems are combined to result in a solution to the original problem. In Steps 1 and 2 of the algorithm, the raw data is processed to create data that serves as targets for simple symbolic regression sub-problems. Then, using the "placeholder variable" idea, it is in Step 3 that the solution to the original problem can be composed from the solutions to the individual symbolic regression sub-problems with negligible computational effort. It is an empirical question as to whether the total effort expended at solving the symbolic regression sub-problems is less than what would be required to solve the original problem directly, and we provide the results of investigations into this question using a variety of synthetic and physical modeling problems, with the primary conclusion being that the proposed algorithm can outperform a baseline free-form symbolic modeling algorithm when both solution accuracy and computational effort are taken into account.

One of our algorithm's main assumptions is that windows of an appropriate size are chosen in Step 1 and Step 2, yet information for making this choice at the start of an experiment would generally not be available. The results shown in Fig. 10C suggest a degree of insensitivity to the exact choice of window size, and in the current work we found that simple inspection of the data was sufficient to determine reasonable window sizes. However, many possibilities exist for a more principled method of identifying window size. For systems with periodic dynamics, one such approach is to find the frequency spectrum of the signal as in Fig. 1B and use it to estimate the optimal window length that would separate faster components of the signal from slower components. More advanced

techniques such as wavelet analysis [1] could also be used to decompose signals into components that dominate at different time scales.

Systems with multiple spatial scales, multiple spatio-temporal scales, or with more than two time scales were not considered in the examples here but should be amenable to analysis by the proposed algorithm with little or no modification. Rather, any limitations of the algorithm would likely be due to noise in the data, the resolution of the data, the magnitude of separation between spatial or temporal scales, and the computational effort expended in the model search. The resolution of the data set and the magnitude of separation between spatial or temporal scales would generally not be under the control of the experimenter, but sophisticated denoising techniques such as unscented Kalman filtering [24] could reduce problems posed by noise in the data.

The synthetic and physical systems here were chosen to be relatively tractable so that several trials could be performed as a basis for comparison between the proposed algorithm, genetic programming alone, linear regression alone, and nonlinear regression alone. It is likely that the relative benefits of the proposed algorithm over ordinary genetic programming and other methods will increase as the difficulty of the problem increases. Further work is needed to rigorously test this hypothesis and to gain additional insight into how the performance of algorithms for automated modeling can scale with problem size and difficulty. Such insight will become increasingly important in our modern world as the rate at which data is collected, stored, and analyzed continues to grow.

Acknowledgments Support was provided by the Tri-Institutional Training Program in Computational Biology and Medicine, US National Science Foundation grant ECCS 0941561 on Cyber-enabled Discovery and Innovation (CDI), US National Institutes of Health NIDA grant RC2 DA028981, and the US Defense Threat Reduction Agency (DTRA) grant HDTRA 1-09-1-0013.

References

1. Akansu AN, Haddad PR (2000) Multiresolution signal decomposition: transforms, subbands, and wavelets. Academic Press, Waltham
2. Argemí A, Saurina J (2007) Study of the degradation of 5-aza-cytidine as a model of unstable drugs using a stopped-flow method and further data analysis with multivariate curve resolution. Talanta 74:176–182
3. Beisler JA (1978) Isolation, characterization, and properties of a labile hydrolysis product of the antitumor nucleoside, 5-azacytidine. J Med Chem 21:204–208
4. Bishop CM (2006) Pattern recognition and machine learning. Springer, London
5. Bongard JC, Lipson H (2007) Automated reverse engineering of nonlinear dynamical systems. Proc Natl Acad Sci USA 104:9943–9948
6. Borcard D, Legendre P, Avois-Jacquet C, Tuomisto H (2004) Dissecting the spatial structure of ecological data at multiple scales. Ecology 85:1826–1832
7. Brameier MF, Banzhaf W (2007) Linear genetic programming. Springer, New York
8. Chen W, Fish J (2001) A dispersive model for wave propagation in periodic heterogeneous media based on homogenization with multiple spatial and temporal scales. J Appl Mech 68:153–161
9. Chen WW, Niepel M, Sorger PK (2010) Classic and contemporary approaches to modeling biochemical reactions. Genes Dev 24:1861–1875
10. Chung J, Jung D, Yoo HH (2004) Stability analysis for the flapwise motion of a cantilever beam with rotary oscillation. J Sound Vib 273:1047–1062
11. Clery D, Voss D (2005) All for one and one for all. Science 308:809
12. Cornforth TW, Lipson H (2012) In: proceedings of the genetic and evolutionary computation conference. Symbolic regression of multiple-time-scale dynamical systems, Philadelphia, USA, 2012, pp 735–742
13. DeJong KA (2006) Evolutionary computation: a unified approach. MIT Press, Cambridge
14. Deng L, Trepat X, Butler JP, Millet E, Morgan KG, Weitz DA, Fredberg JJ (2006) Fast and slow dynamics of the cytoskeleton. Nat Mater 5:636–640
15. Draper NR, Smith H (1998) Applied regression analysis. Wiley, New York
16. Evans J, Rzhetsky A (2010) Machine science. Science 329:399–400
17. Guven A (2009) Linear genetic programming for time-series modeling of daily flow rate. J Earth Syst Sci 118:137–146
18. Harris-Warrick RM, Flamm RE (1987) Multiple mechanisms of bursting in a conditional bursting neuron. J Neurosci 7:2113–2128
19. Henzler-Wildman KA, Lei M, Thai V, Kerns SJ, Karplus M, Kern D (2007) A hierarchy of timescales in protein dynamics is linked to enzyme catalysis. Nature 450:913–916
20. Hornby GS (2006) In: proceedings of the genetic and evolutionary computation conference. ALPS: the age-layered population structure for reducing the problem of premature convergence, New York, 206, pp 815–822
21. Iba H, de Garis H, Sato T (1994), In Davidor Y, Schwefel H-P, Männer R (eds) proceedings of the PPSN III. Genetic programming with local hill-climbing, Springer, London, 1994, pp 302–311
22. Jara EC (2011) Long memory time series forecasting by using genetic programming. Genet Program Evol Mach 12:429–456
23. Jones CKRT (2001) Multiple-time-scale dynamical systems. Springer, New York
24. Julier S, Uhlmann J, Durrant-Whyte HF (2000) A new method for the nonlinear transformation of means and covariances in filters and estimators. IEEE Trans Automat Control 45:477–482
25. King RD, Whelan KE, Jones FM, Reiser PGK, Bryant CH, Muggleton SH, Kell DB, Oliver SG (2004) Functional genomic hypothesis generation and experimentation by a robot scientist. Nature 427:247–252
26. King RD, Rowland J, Oliver SG, Young M, Aubrey W, Byrne E, Liakata M, Markham M, Pir P, Soldatova LN, Sparkes A, Whelan KE, Clare A (2009) The automation of science. Science 324:85–89
27. Klein R (2001) Asymptotic adaptive methods for multi-scale problems in fluid mechanics. J Eng Math 39:261–343
28. Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge
29. Koza JR, Bennett FH, Andre D, Keane MA (1999) Genetic programming III: Darwinian invention and problem solving. Morgan Kaufmann, San Francisco

30. Langley P, Simon HA, Bradshaw G, Zytkow JM (1987) Scientific discovery: computational exploration of the creative processes. MIT Press, Cambridge
31. Lawlor FD (1973) MFIT: a multiple nonlinear curve fitting technique. IBM Poughkeepsie laboratory technical report TR 00.2482
32. Le Pourhiet A, Corrège M, Caruana D (2003) Control of self-oscillating systems. IEEE Proc Control Theory Appl 150:599–610
33. Lee CH, Othmer HG (2009) A multi-time-scale analysis of chemical reaction networks: I Deterministic systems. J Math Biol 60:387–450
34. Mahfoud SW (1992), In: Männer R, Manderick R (eds) parallel problem solving from nature—PPSN-II. Crowding and preselection revisited, Elsevier
35. Mulloy BS, Riolo RL, Savit RS (1996) In: Genetic programming 1996: proceedings of the first annual conference. Dynamics of genetic programming and chaotic time series prediction, pp 166–174
36. Nocedal J, Wright SJ (2006) Numerical optimization. Springer, New York
37. Palsson BO (1987) On the dynamics of the irreversible Michaelis-Menten reaction mechanism. Chem Eng Sci 42:447–458
38. Pandolfi JM (2002) Coral community dynamics at multiple scales. Coral Reefs 21:13–23
39. Russell S, Norvig P (2009) Artificial intelligence: a modern approach, 3rd edn. Prentice Hall, Upper Saddle River
40. Sakamoto E, Iba H (2001) In: proceedings of the 2001 congress on evolutionary computation. Inferring a system of differential equations for a gene regulatory network by using genetic programming. pp 720–726
41. Schmidt MD, Lipson H (2007) In: proceedings of the genetic and evolutionary computation conference 2007. Comparison of tree and graph encodings as a function of problem complexity. pp 1674–1679
42. Schmidt MD, Lipson H (2008) Coevolution of fitness predictors. IEEE Trans Evol Comp. 12:736–749
43. Schmidt MD, Lipson H (2009) Distilling free-form natural laws from experimental data. Science 324:81–85
44. Schmidt MD, Lipson H (2010) In: proceedings of the genetic and evolutionary computation conference 2010. Predicting solution rank to improve performance. pp 949–956
45. Schmidt MD, Lipson H (2010) Age-fitness Pareto optimization. Genet Program Theory Pract 8:129–146
46. Schmidt MD, Vallabhajosyula RR, Jenkins JW, Hood JE, Soni AS, Wikswo JP, Lipson H (2011) Automated refinement and inference of analytical models for metabolic networks. Phys Biol 8:055011
47. Strogatz SH (2001) Exploring complex networks. Nature 410:268–276
48. Szalay A, Gray J (2006) 2020 Computing: science in an exponential world. Nature 440:413–414
49. Verhulst F (2005) Methods and applications of singular perturbations: boundary layers and multiple timescale dynamics. Springer, New York
50. Wagner N, Michalewicz Z, Khouja M, McGregor RR (2007) Time series forecasting for dynamic environments: the DyFor genetic program model. IEEE Trans Evolut Comput 11:433–452
51. Waltz D, Buchanan BG (2009) Automating science. Science 324:43–44