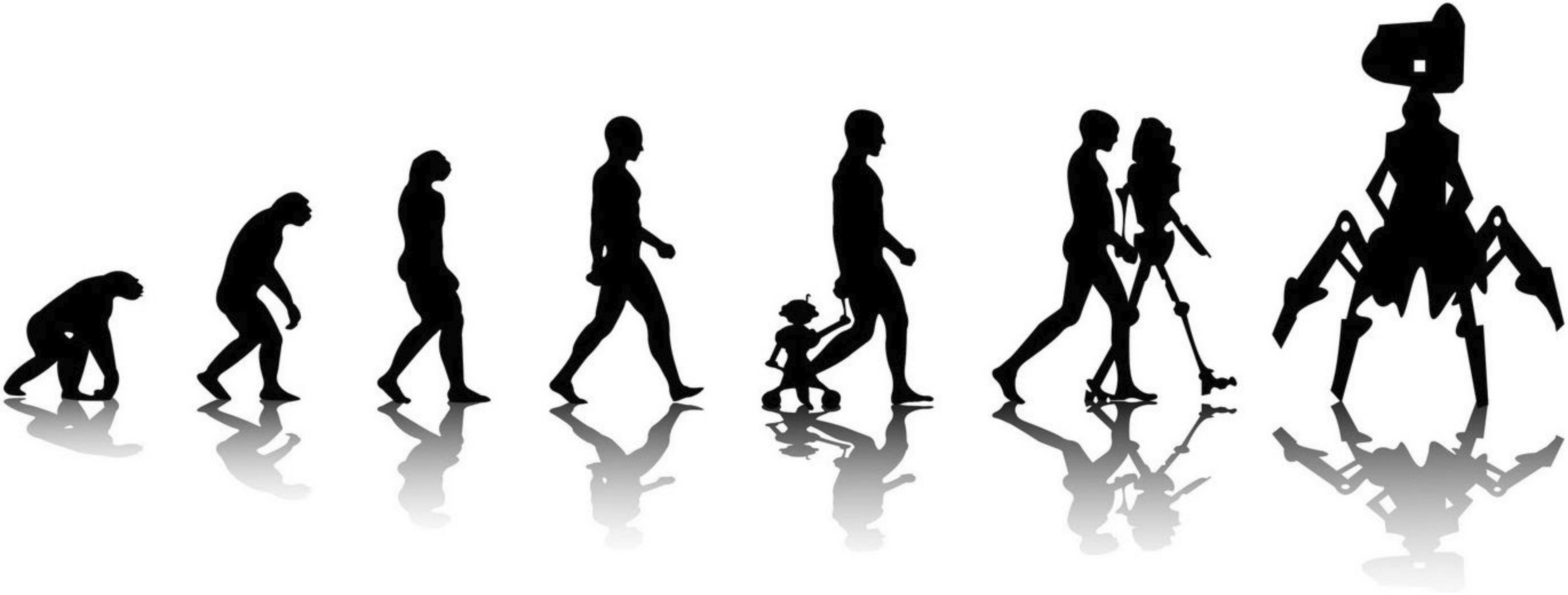


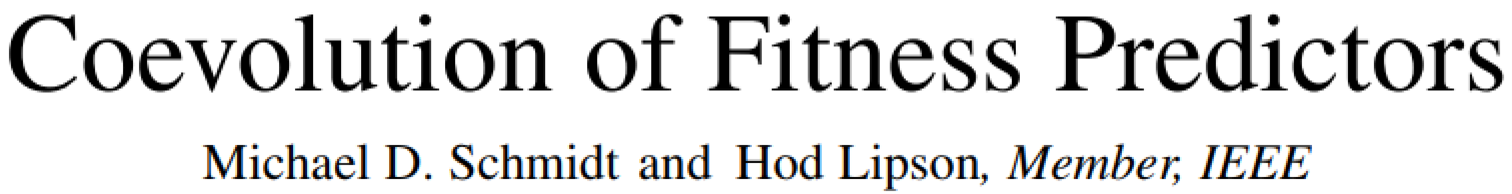
Modern Robotics: Evolutionary Robotics

COSC 4560 / COSC 5560

Professor Cheney

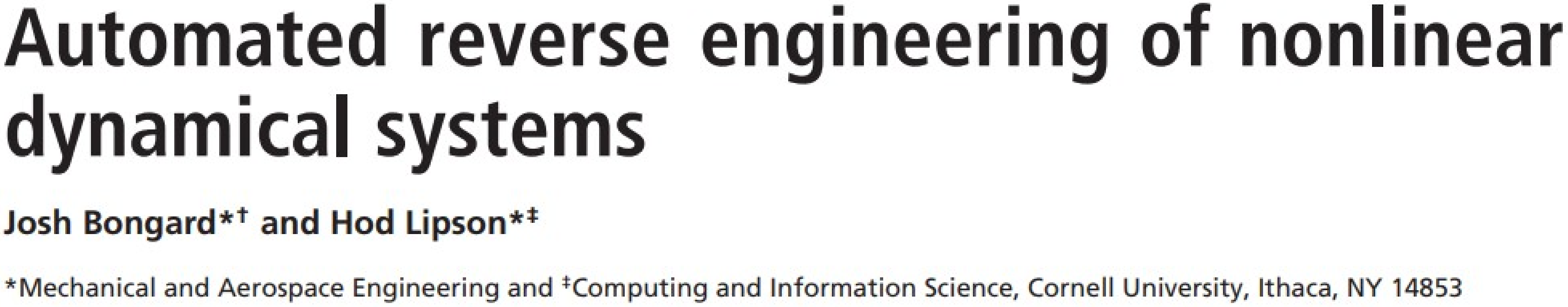
4/11/18

Surrogate Models for Fitness Evaluations



Coevolution of Fitness Predictors

Michael D. Schmidt and Hod Lipson, *Member, IEEE*



Automated reverse engineering of nonlinear dynamical systems

Josh Bongard*† and Hod Lipson*‡

*Mechanical and Aerospace Engineering and ‡Computing and Information Science, Cornell University, Ithaca, NY 14853

1) Motivation: There are several reasons for utilizing fitness approximation through modeling.

— **Reducing complexity**: Many applications of evolutionary algorithms are in high-complexity or intractable domains, where the fitness calculation can be prohibitively time consuming. For example, fitness modeling has been applied to structural design optimization [1], [2], [21]–[25] that often requires time-consuming finite-element calculations. Often the resolution provided by the exact fitness objective is unnecessary for evolutionary progress.

— **No explicit fitness**: Many domains do not have a computable fitness. For example, in human interactive evolution [26] (e.g., evolution of art and music), a human user must select favorable individuals. Fitness models have been applied in these domains to reduce user fatigue and define a computable fitness landscape that can be searched, while waiting for the user to give more feedback [11], [27], [28].

— **Noisy fitness**: Some fitness functions are very noisy. To produce stable fitness rankings, algorithms typically average many evaluations, but this can greatly increase the computational cost [29]. An alternative approach may be to develop a statistical model [30].

— **Smoothing landscapes**: Almost all evolutionary domains suffer from multimodal landscapes that are often dense with local optima. Fitness approximation can greatly reduce the frequency and severity of local optima. Landscape smoothing has been observed with interpolation, kernels, and fitness clustering [24], [25], [31], [32].

— **Promoting diversity**: When models smooth fitness landscapes, they often flatten local optima or produce different regions with similar fitness. While this is undesirable when using a single model throughout evolution, it can be advantageous for producing diversity as long as the fitness model continuously adapts, as is proposed in this paper.

Subsample Fitness Predictors

1) Fitness Predictor Encoding: Training data in symbolic regression typically consists of hundreds to thousands of data points (e.g., experimental measurements) providing output values for a sample of inputs. In our symbolic regression experiments, the fitness predictor is a small subset of these points. Instead of measuring the exact objective fitness of candidate solutions, a subjective fitness is obtained by measuring the error on the select handful of data points of a given fitness predictor.

The fitness predictor is encoded as a small array of indexes to the full training data set (size discussed in the next section). Each index in the predictor's array is free to reference any points in the training data examples and can repeatedly sample point if it likes (thus over emphasizing an area). The predicted fitness is calculated as

$$\text{predicted\_fitness}(s) = \frac{1}{n}\sum_{i=1}^{n} |s(x_i) - y_i|$$

where s is a candidate solution (algebraic expression), x_i and y_i are training data inputs and outputs in the training dataset indexed by the predictor, and n is the number of samples in the predictor.

Genetic Programming

1) heritability of trails
   Genome = expression tree

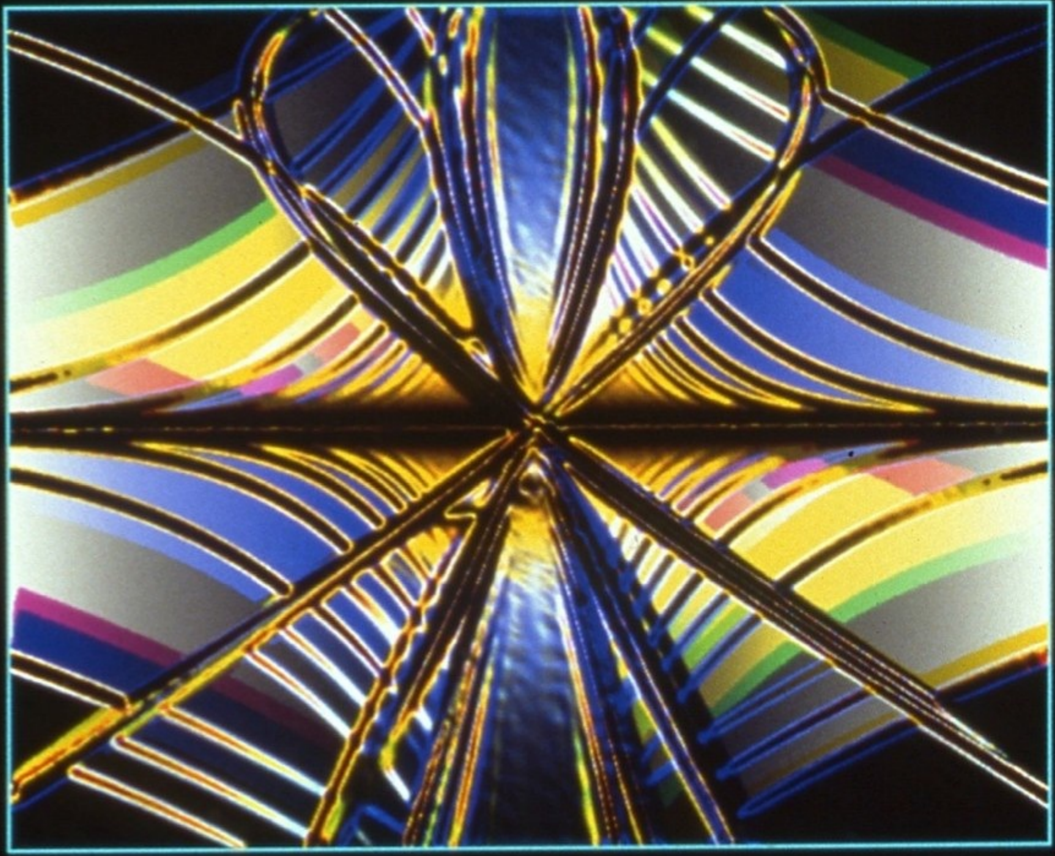
2) genetic diversity through variation
   Mutation = replace node/change weights
   Crossover = swap sub-trees/activation functions

3) competition over scarce resources
   Selection is separate from encoding
   (our task = symbolic regressions)

Images are generated procedurally by symbolic Lisp expressions:

Color <= F(x,y)

Phenotype:
(Image)

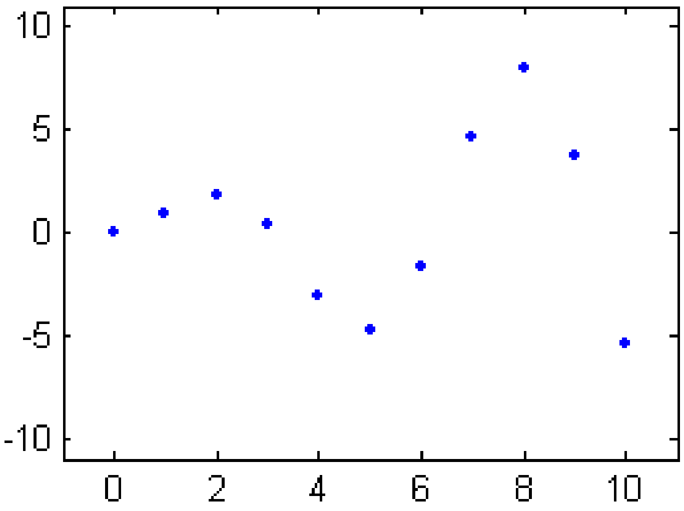


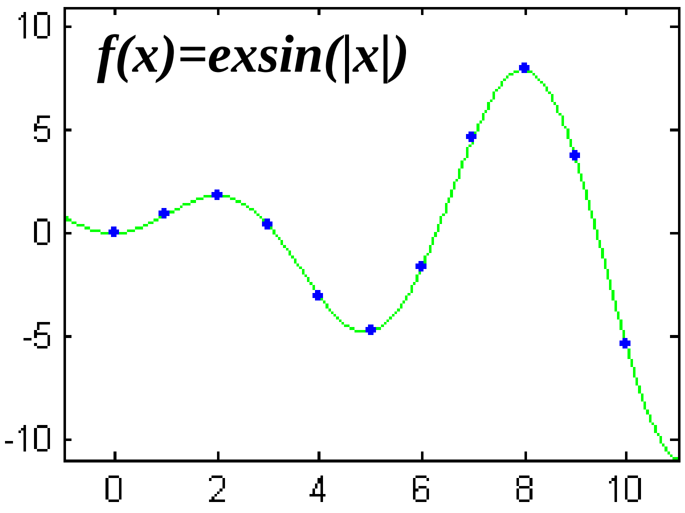
Genotype:
(Lisp code)

```
(round (log (+ y (color-grad (round (+ (abs (round (log
(+ y (color-grad (round (+ y (log (invert y) 15.5)) x) 3.1
1.86 #(0.95 0.7 0.59) 1.35)) 0.19) x)) (log (invert y) 15.5))
x) 3.1 1.9 #(0.95 0.7 0.35) 1.35)) 0.19) x)
```

Genetic Programming Example: Symbolic Regression

What function describes this data?





Genetic Programming Example: ~~Symbolic~~ Regression

What function describes this data?

$y = a * x + b$, solve for (a, b) given (x, y)

$y = a * x^2 + b * x + c$, solve for (a, b, c) given (x, y)

$y = a * x^3 + b * x^2 + c * x + d$, solve for (a, b, c, d) given (x, y)

$y = a + \sin(x)^2 * (b / c) * \log(x)$, solve for (a, b, c) given (x, y)

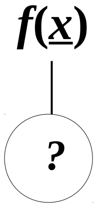
Genetic Programming Example: Symbolic Regression

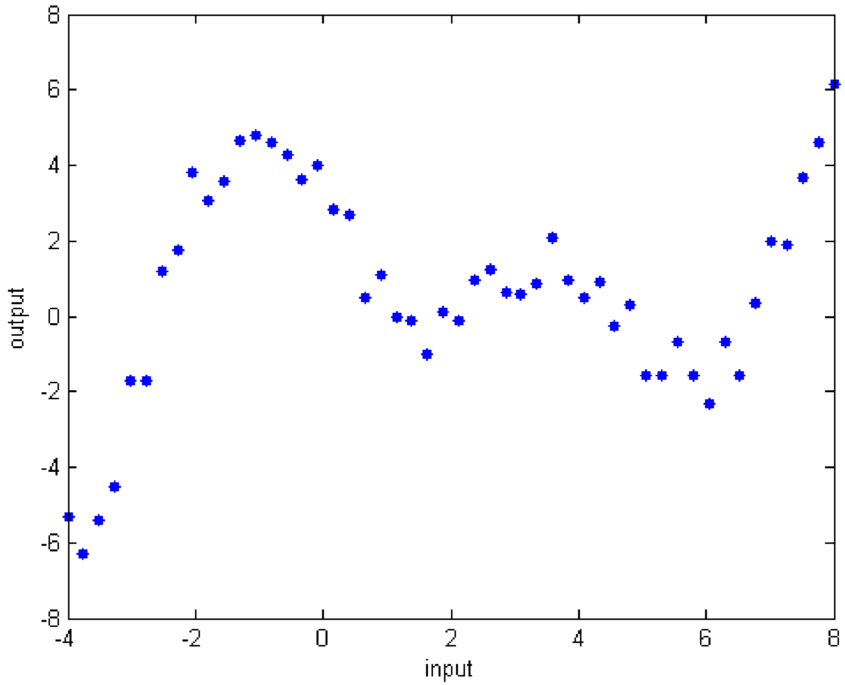
What function describes this data?

y = ? ? ? ? ? ? … , solve for (?, ?, ?, …) given (x, y)

Genetic Programming Example: Symbolic Regression

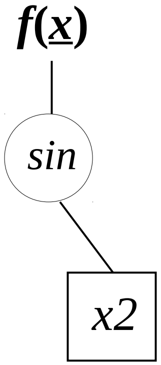
Building Blocks: + - * / sin cos exp log … etc





Genetic Programming Example: Symbolic Regression

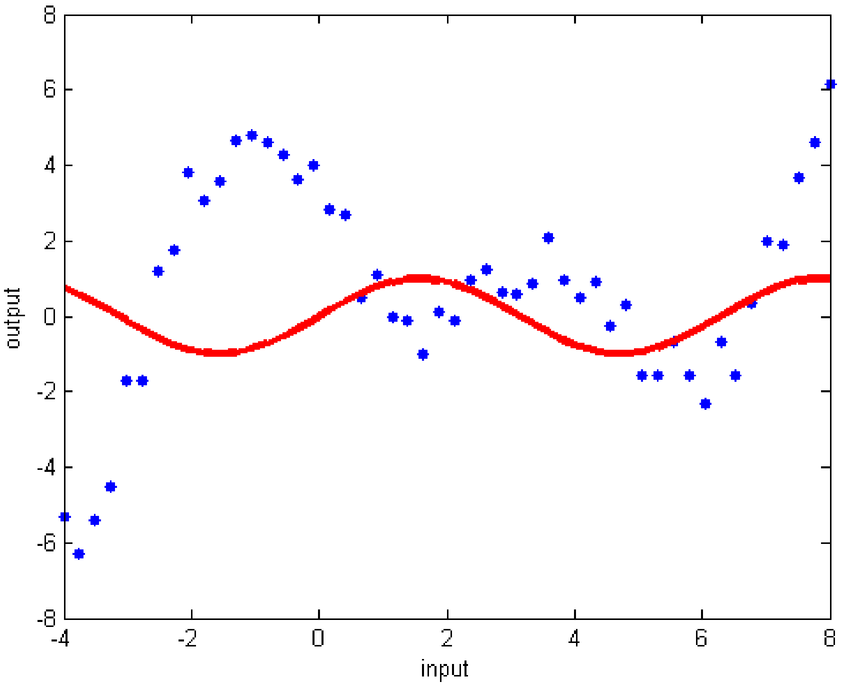
Building Blocks: + - * / sin cos exp log … etc



***f*(*x*)**

sin

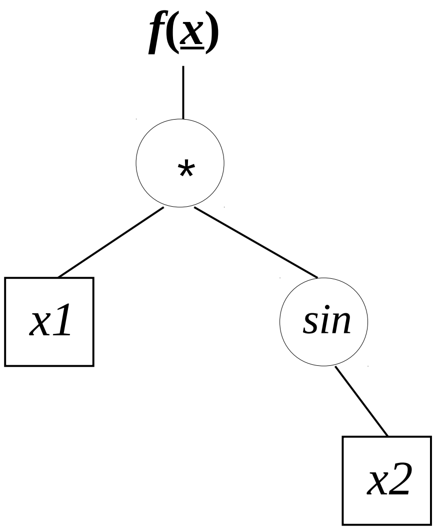
x2

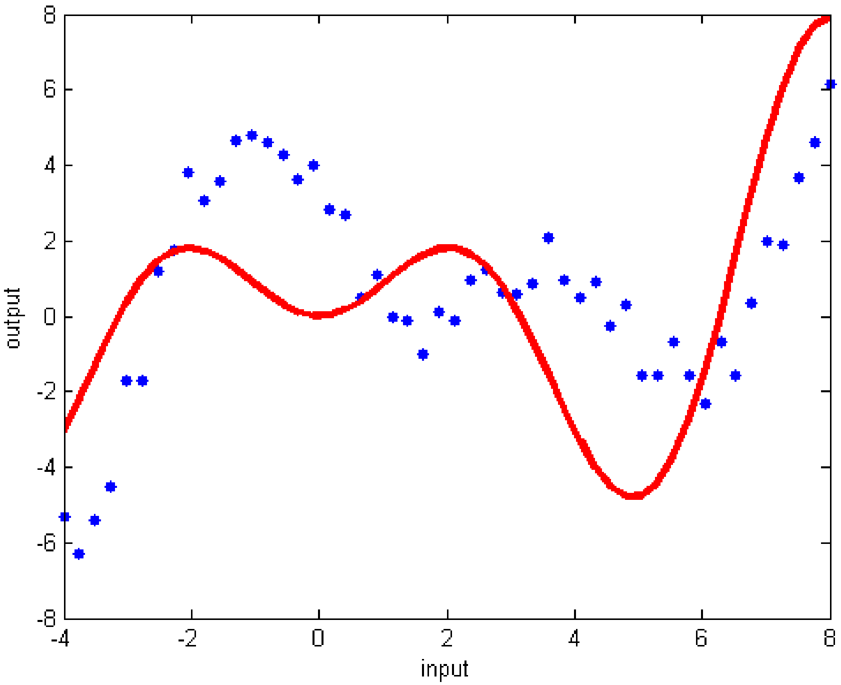


sin(*x2*)

Genetic Programming Example: Symbolic Regression

Building Blocks: + - * / sin cos exp log … etc



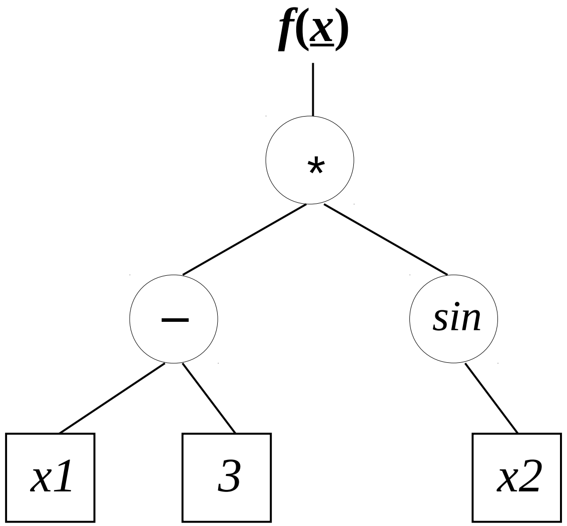


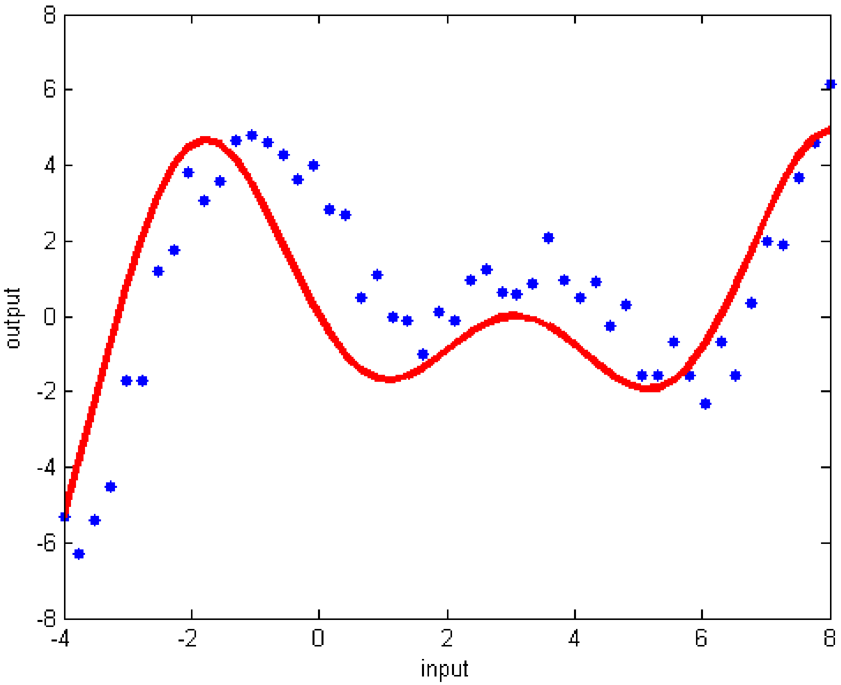
sin(*x2*)

x1*sin(*x2*)**

Genetic Programming Example: Symbolic Regression

Building Blocks: + - * / sin cos exp log … etc





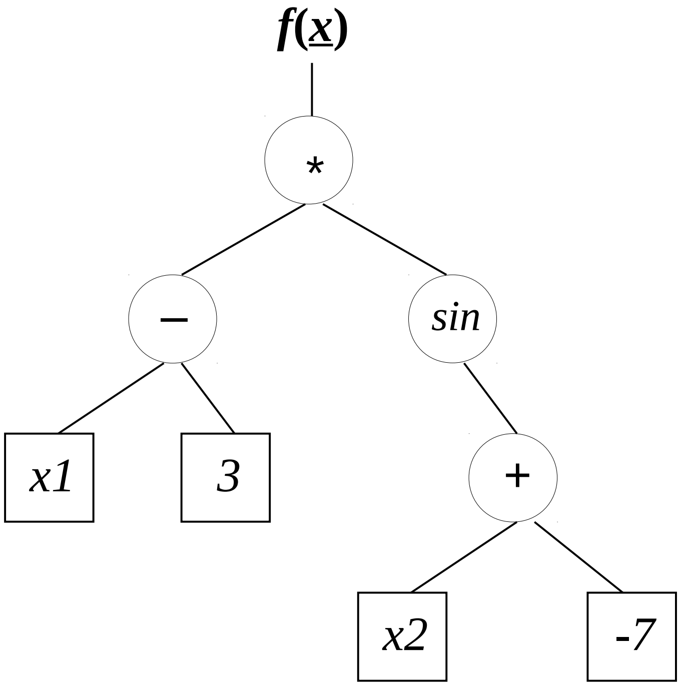
sin(*x2*)

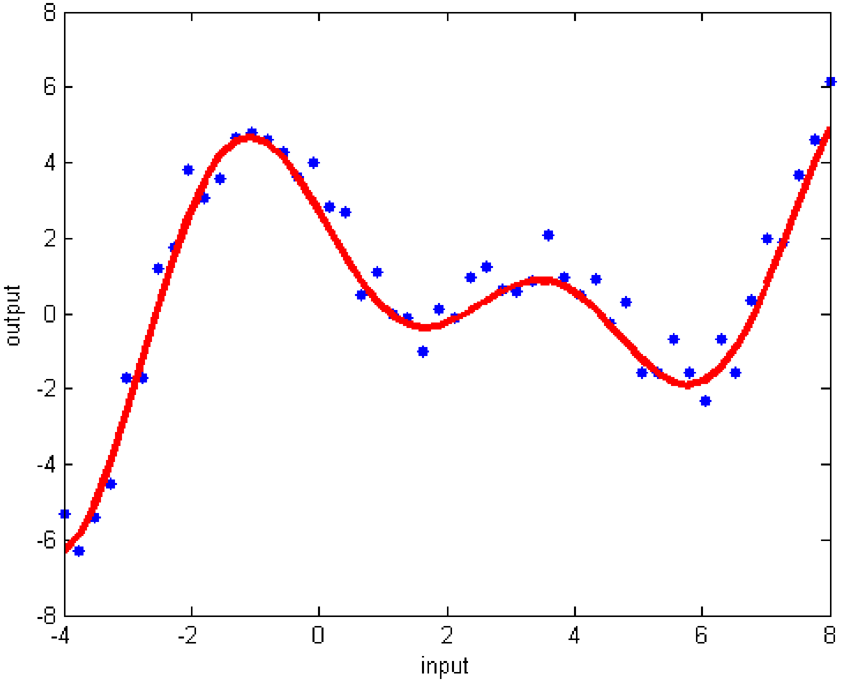
*x1**sin(*x2*)

(*x1* – 3)*sin(*x2*)

Genetic Programming Example: Symbolic Regression

Building Blocks: + - * / sin cos exp log … etc



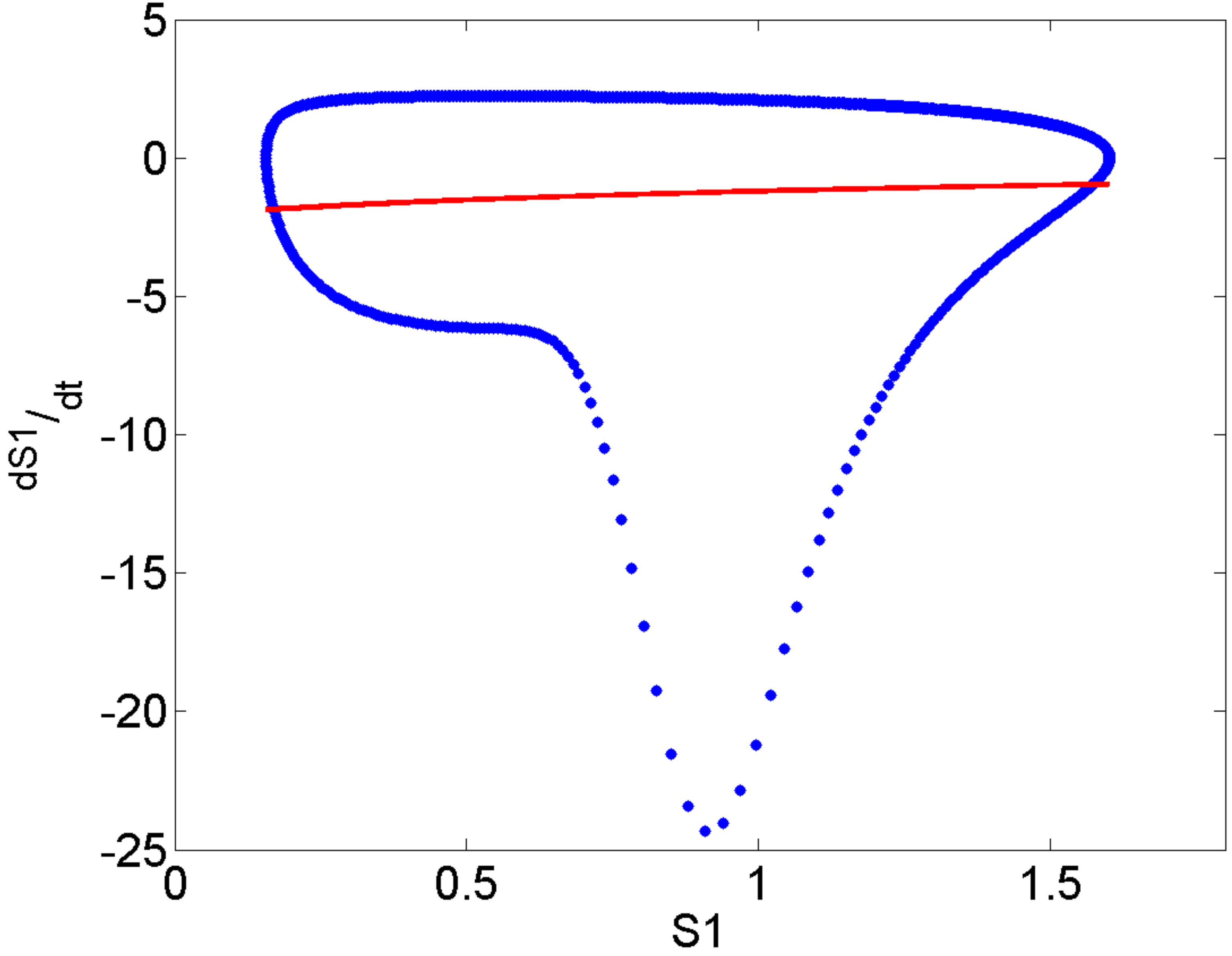


sin(*x2*)

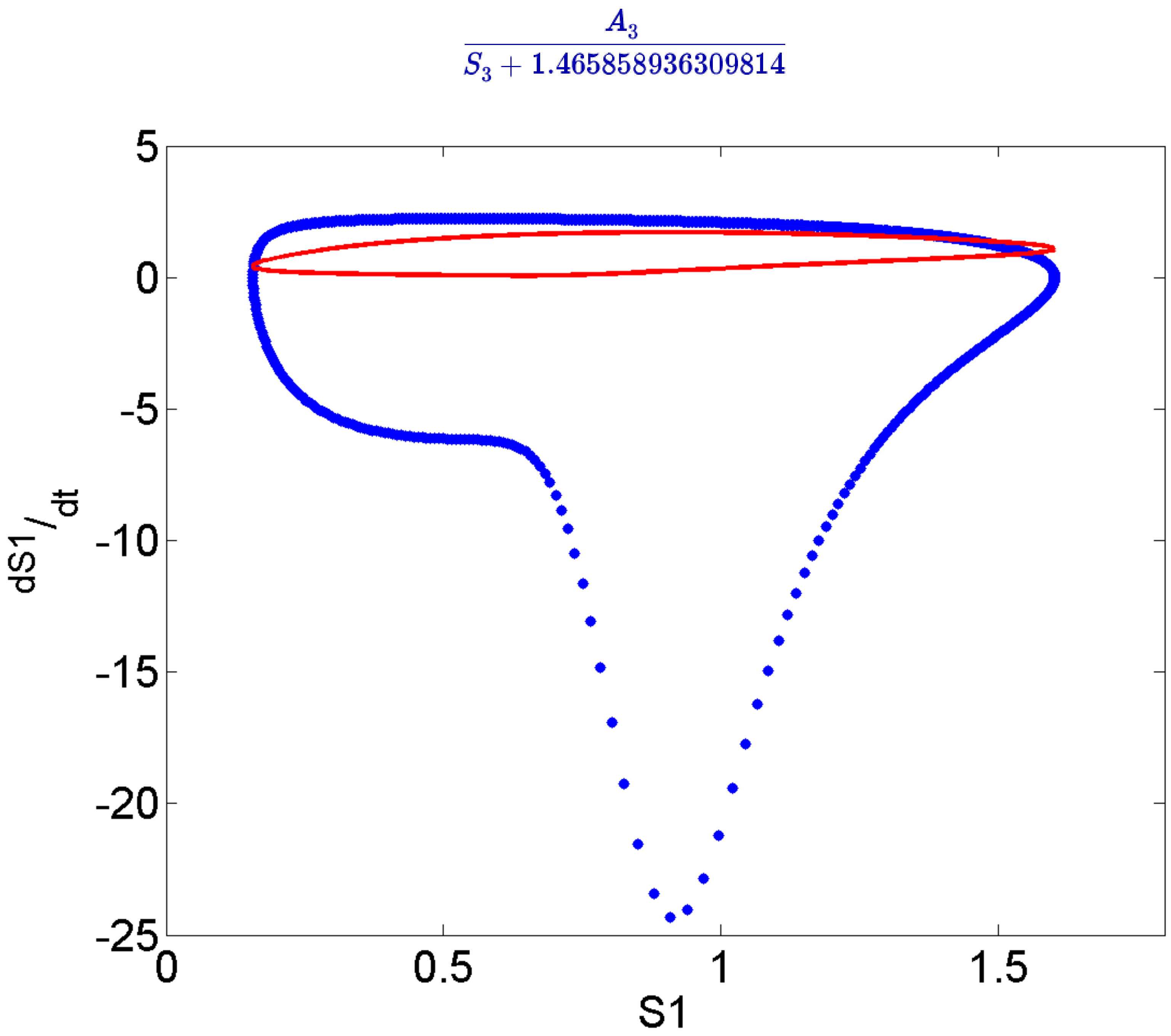
***x1**sin(*x2*)**

(*x1* – 3)*sin(*x2*)

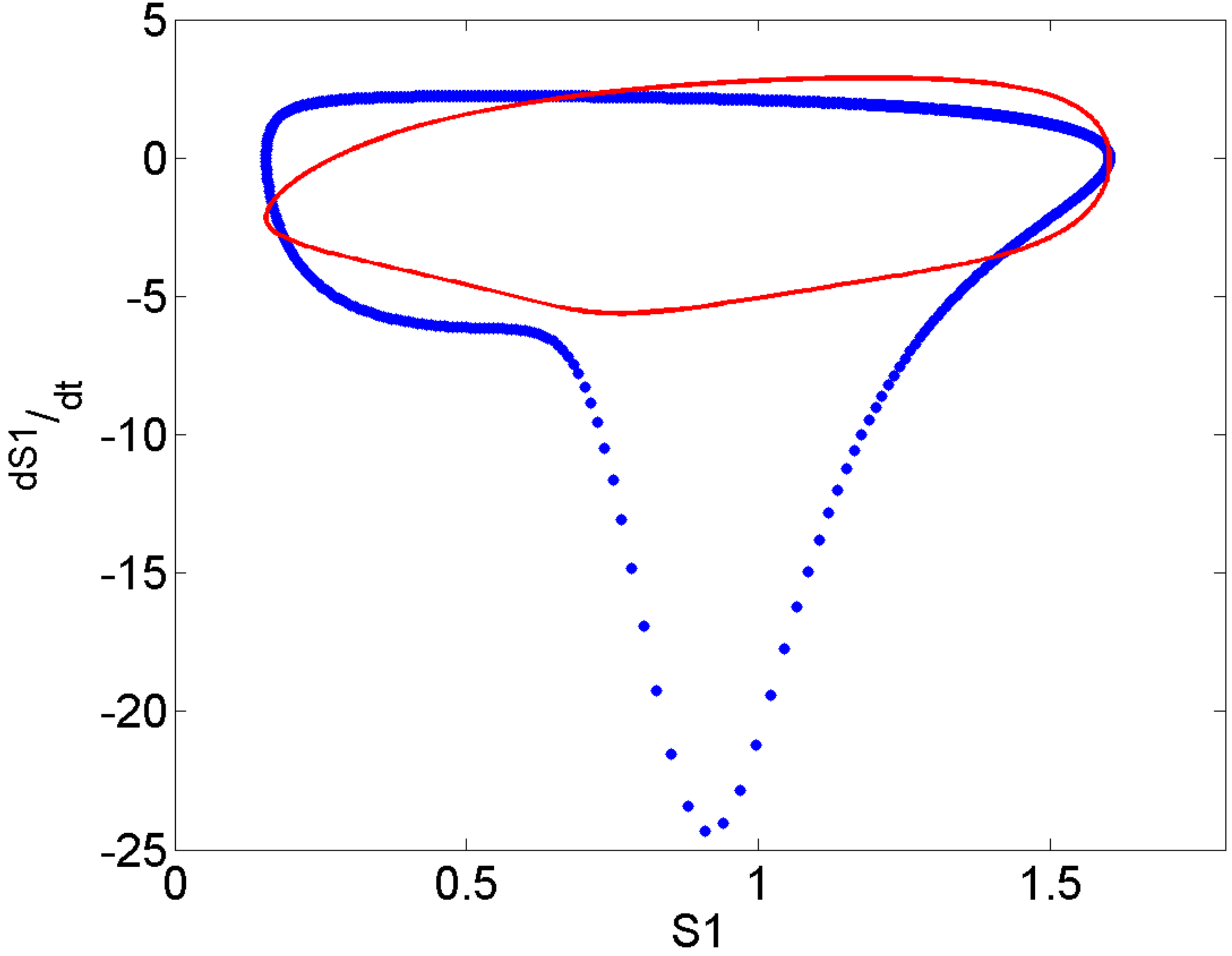
(*x1* – 3)*sin(-7 + *x2*)



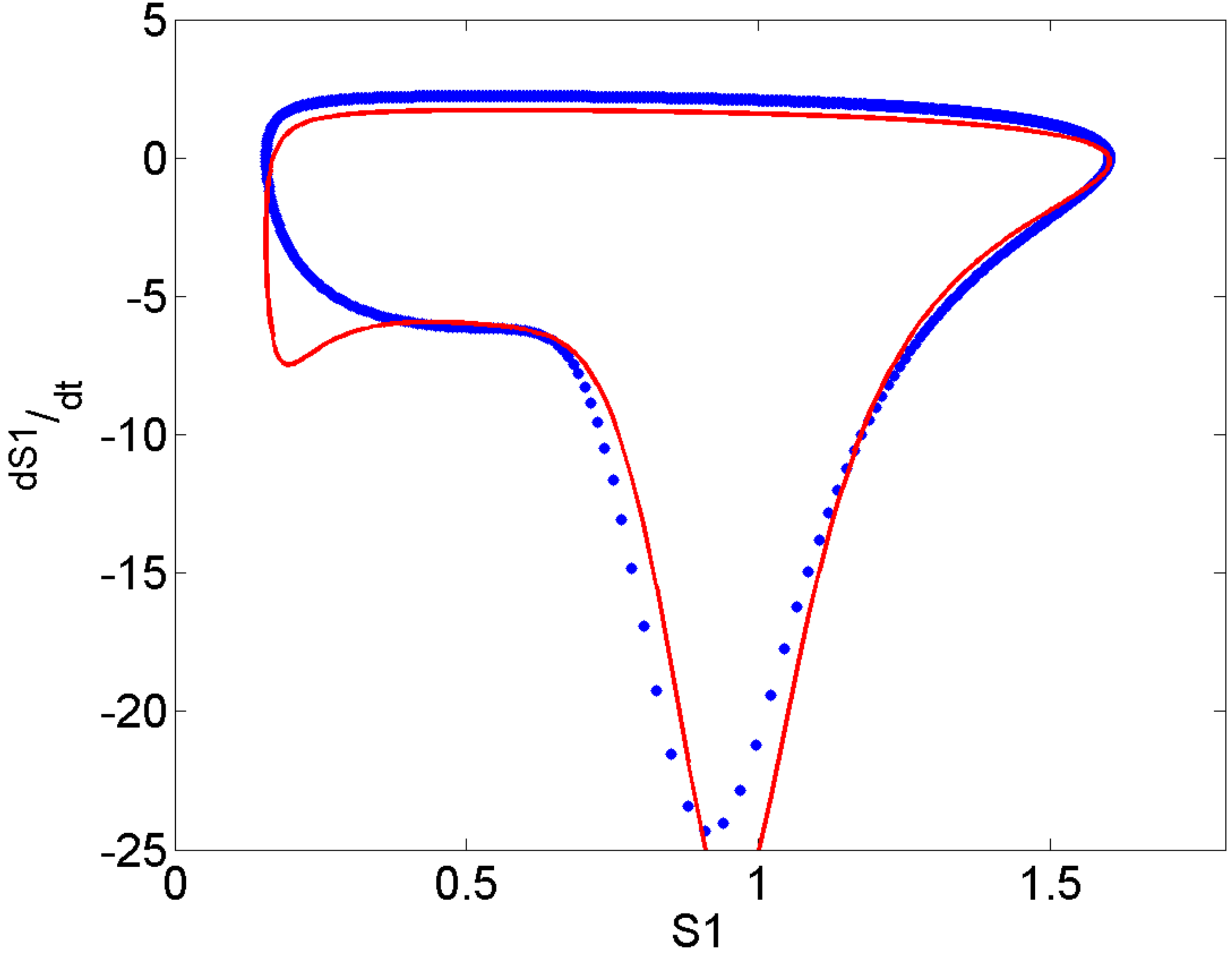
$$\frac{-2.502805233001709}{S_3 + S_1 + 1}$$


$\frac{A_3}{S_3 + 1.465858936309814}$
5
0
-5
-10
-15
-20
-25
0
0.5
1
1.5
$^{dS1}/_{dt}$
S1

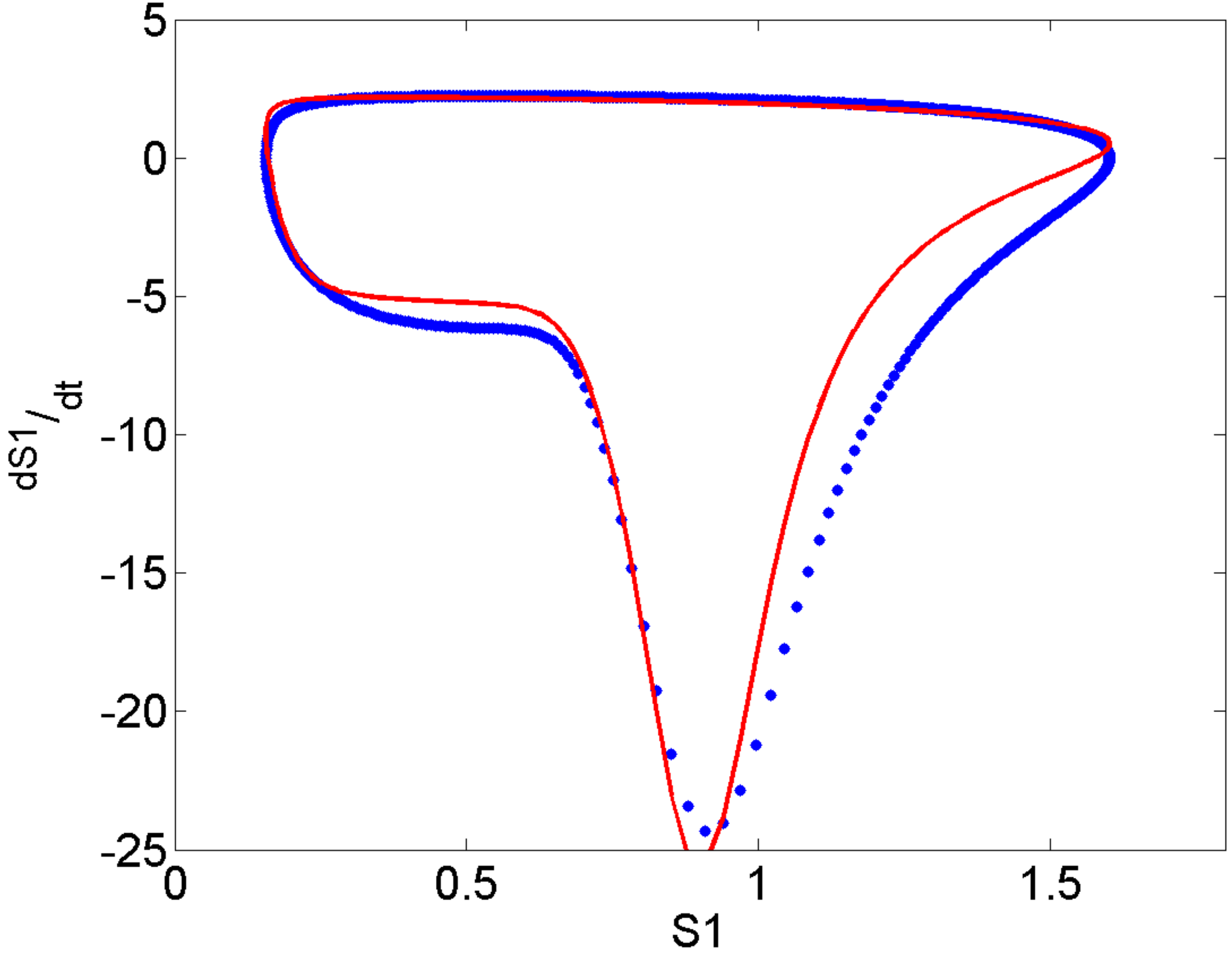
$$\frac{5.931042138244496 \cdot (A_3 - 1.676328420639038)}{S_2 - S_1 + A_3 + 0.158742368221283}$$



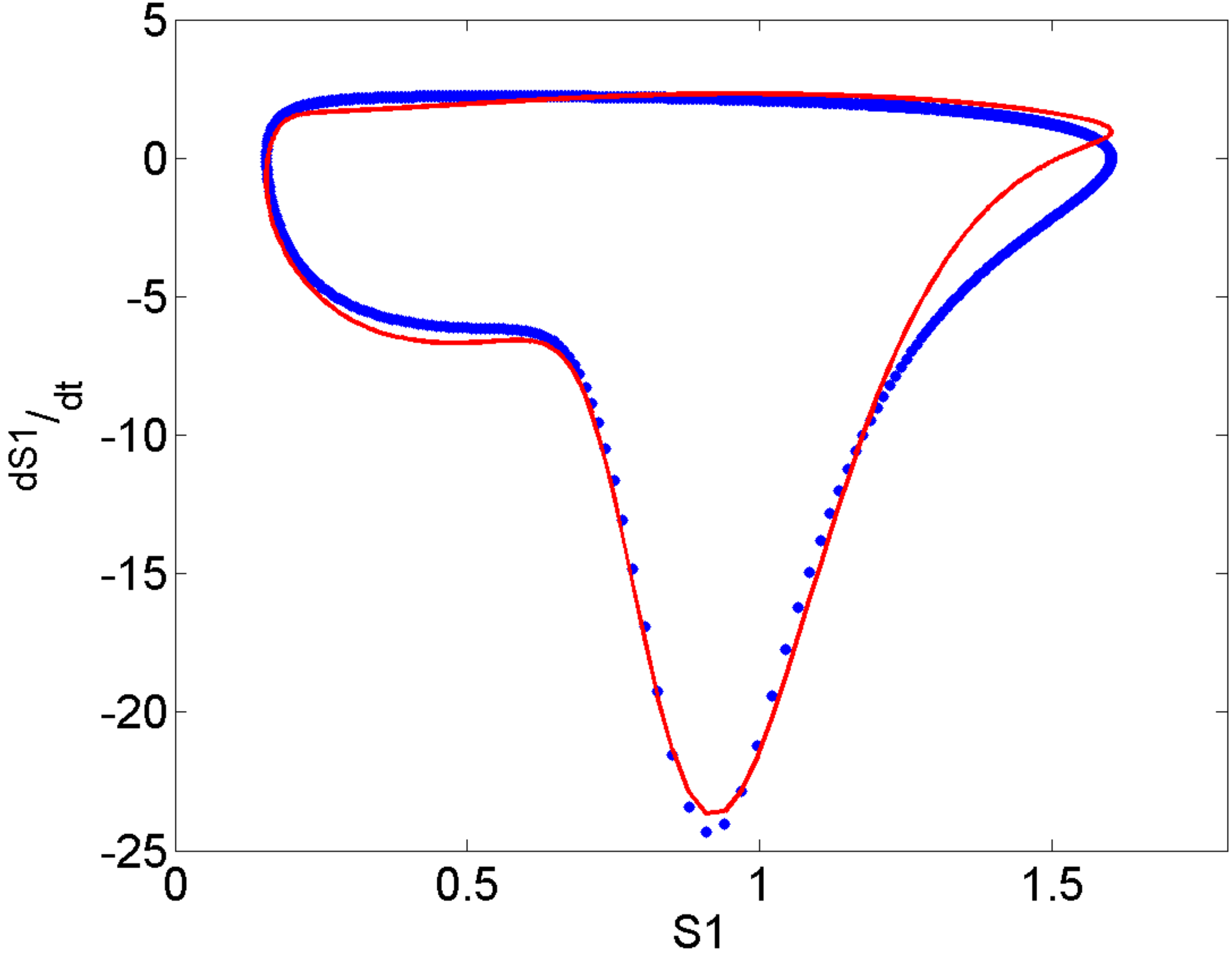
$$\frac{S_4 + 2 \cdot S_1 + 2 \cdot N_2 - (2 \cdot (A_3 - 0.944967806339264)) \cdot A_3 - 0.995008140802383}{(0.944967806339264 - A_3) \cdot A_3 - 0.293821990489960}$$



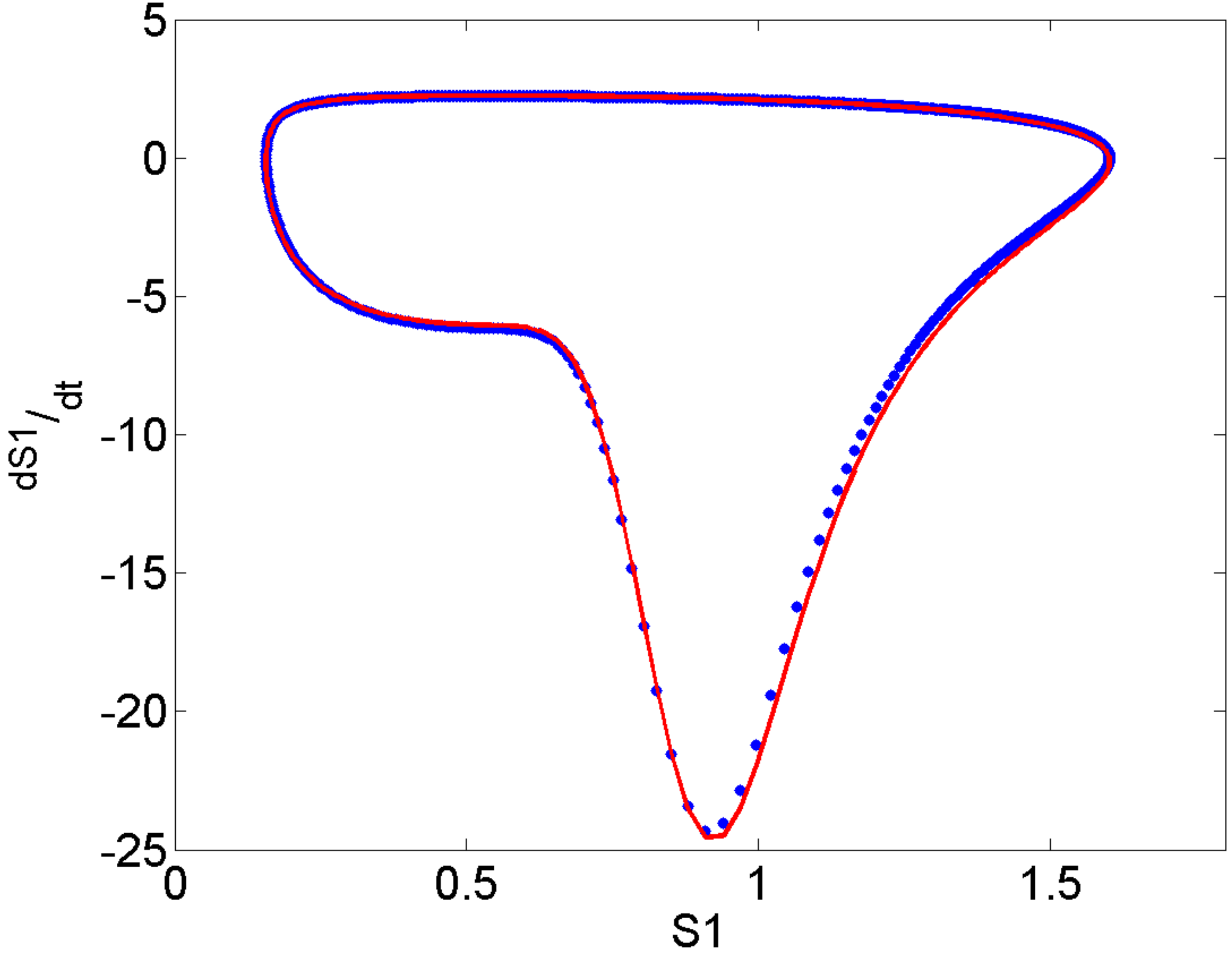
$$2 \cdot S_5 + \frac{2 \cdot S_1}{(0.755706310272217 - A_3) \cdot A_3 - 0.204600989818573} + \frac{2 \cdot N_2}{A_3} + 2.119162559509277$$



$$\left((2.16234540939331 - A_3) \cdot \left(\left(-(2.16234540939331 - A_3)^8\right) \cdot {A_3}^2 - 1.078754663467407\right)\right) \cdot S_1 + 1.82153654098510$$



$$2.53075122833252 - \frac{42.3825454711914 \cdot S_1}{5.432642911755188 \cdot {A_3}^3 + \frac{0.429036676883698}{A_3}}$$



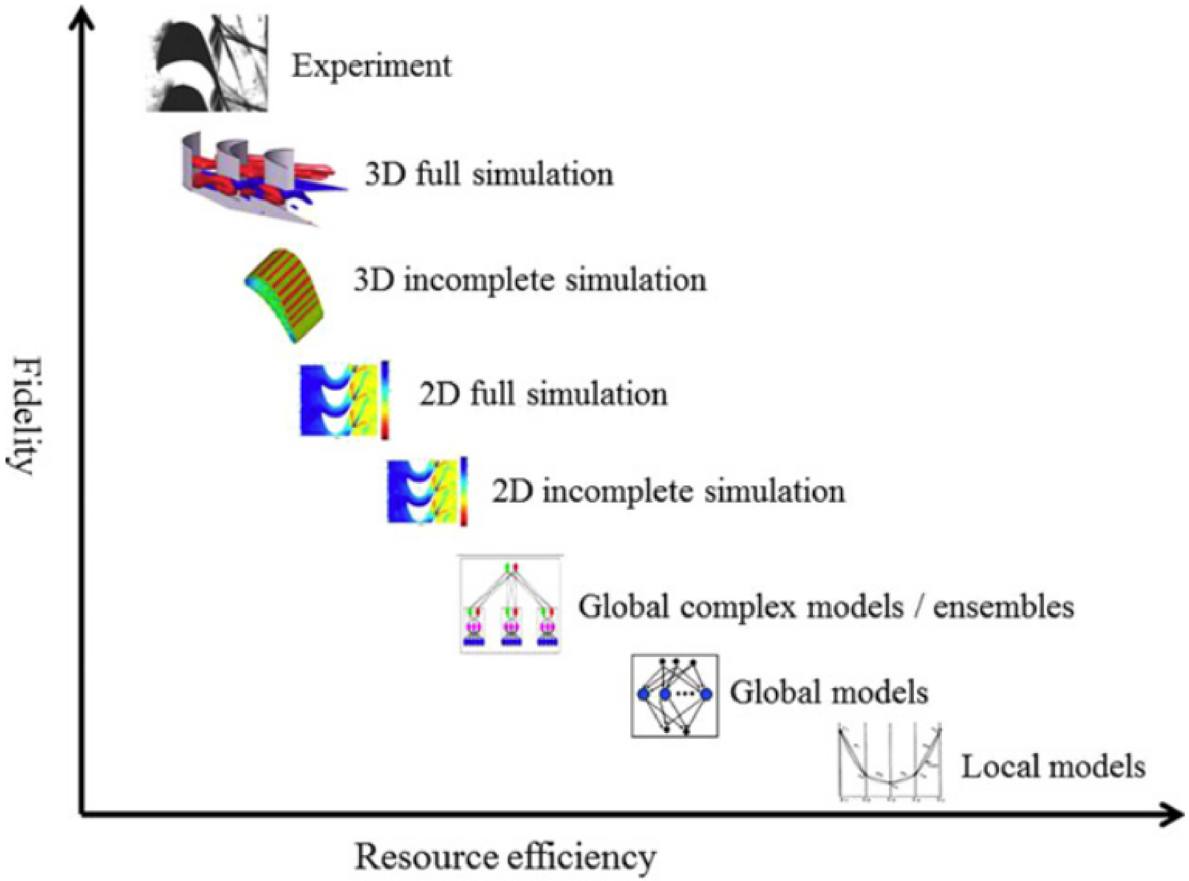
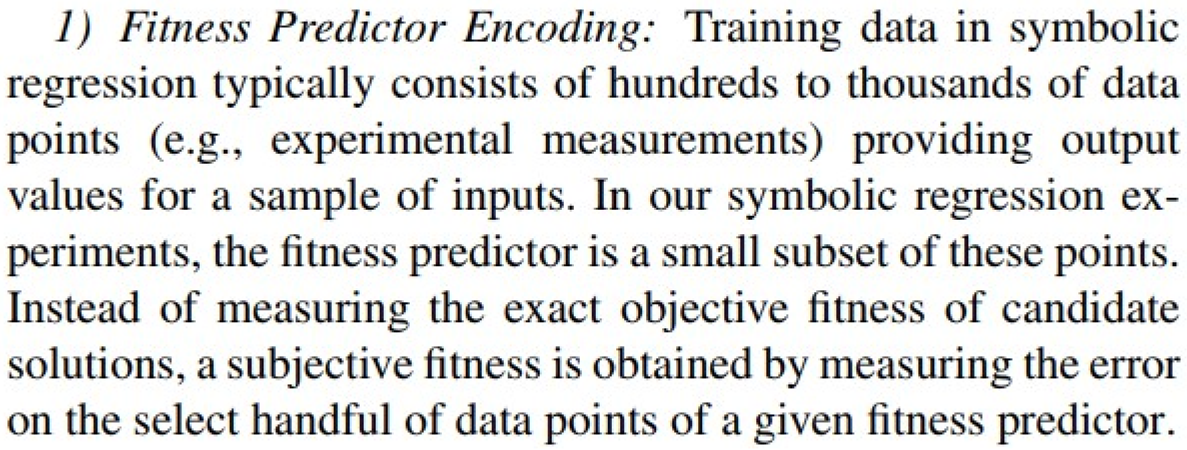
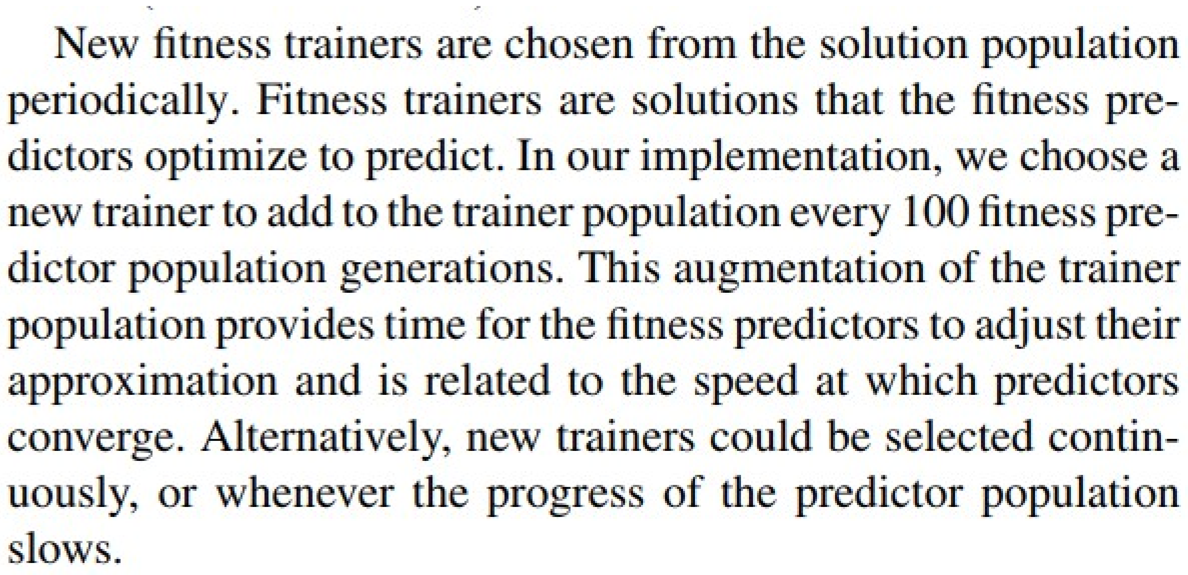


Fig. 1. An illustration of a trade-off between fidelity (approximation accuracy) and computational cost. Usually, high-fidelity fitness evaluations are more time-consuming. By contrast, low-fidelity fitness evaluations are often less time-consuming.



1) Fitness Predictor Encoding: Training data in symbolic regression typically consists of hundreds to thousands of data points (e.g., experimental measurements) providing output values for a sample of inputs. In our symbolic regression experiments, the fitness predictor is a small subset of these points. Instead of measuring the exact objective fitness of candidate solutions, a subjective fitness is obtained by measuring the error on the select handful of data points of a given fitness predictor.



New fitness trainers are chosen from the solution population periodically. Fitness trainers are solutions that the fitness predictors optimize to predict. In our implementation, we choose a new trainer to add to the trainer population every 100 fitness predictor population generations. This augmentation of the trainer population provides time for the fitness predictors to adjust their approximation and is related to the speed at which predictors converge. Alternatively, new trainers could be selected continuously, or whenever the progress of the predictor population slows.

The objectives for each population are summarized below, where asterisks specify an optimal result that is being searched for in each population

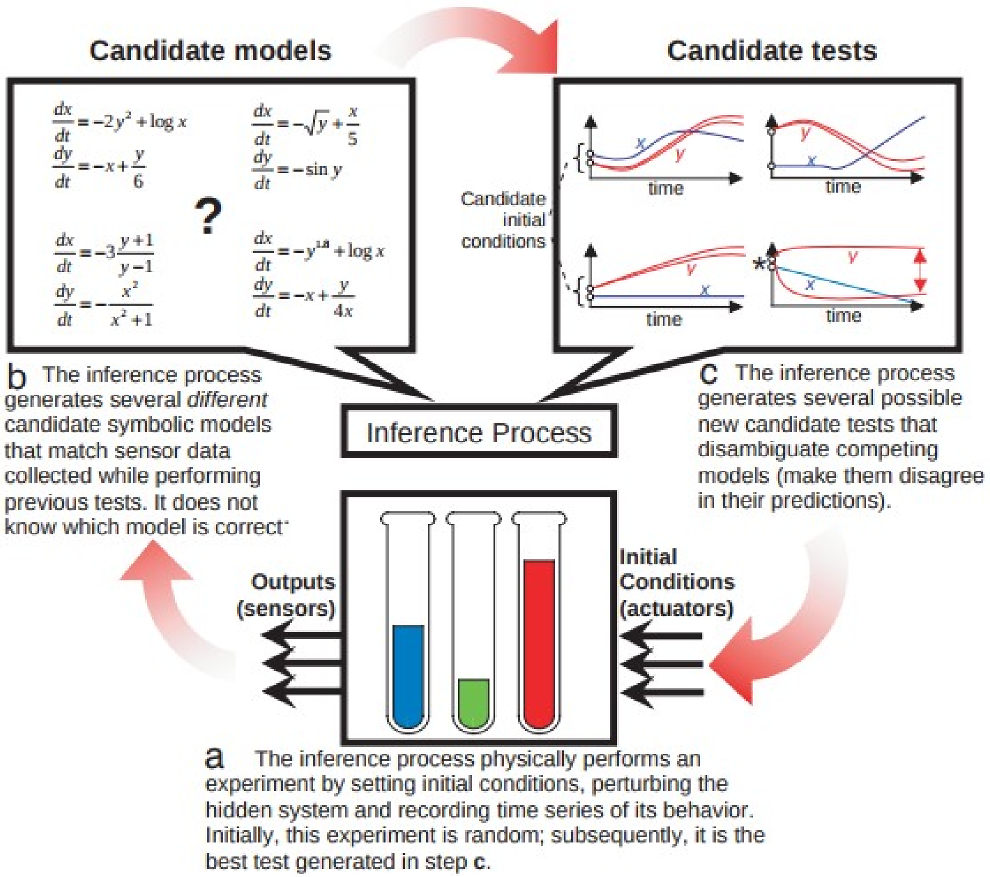
$$s^* = \arg\max_{s\in S} p_{\text{best}}(s) \text{ (Solutions)}$$

$$t^* = \arg\max_{s\in S_c} \frac{1}{N} \sum_{p\in P_{\text{cur}}} (p(s) - \overline{p(s)})^2 \text{ (Trainers)}$$

$$p^* = \arg\min_{p\in P} \frac{1}{N} \sum_{t\in T_{\text{cur}}} |\text{fitness}(t) - p(t)| \text{ (Predictors)}$$

where S is the set of all problem solutions, S_c is the current solution population, P is the set of all possible fitness predictors, P_{cur} is the current predictor population, T_{cur} is the current trainers population, p_{best} is the highest ranked predictor in P_{cur}, and $\overline{p(s)}$ is the average predicted fitness of solution s among the current predictors. It is important to note that all three populations are evolved in parallel and their objectives will be dynamic and changing over each generation.

To summarize the framework, the solution population evolves to maximize the fitness of the current best fitness predictor. Trainers are solutions chosen from the solution population that produce the most variance in predictions among the predictor population. The fitness predictor population evolves to minimize the difference between exact and predicted fitnesses of the current population.


Candidate models
$\frac{dx}{dt} = -2y^2 + \log x$
$\frac{dy}{dt} = -x + \frac{y}{6}$
$\frac{dx}{dt} = -\sqrt{y} + \frac{x}{5}$
$\frac{dy}{dt} = -\sin y$
?
$\frac{dx}{dt} = -3\frac{y+1}{y-1}$
$\frac{dy}{dt} = -\frac{x^2}{x^2+1}$
$\frac{dx}{dt} = -y^{1.8} + \log x$
$\frac{dy}{dt} = -x + \frac{y}{4x}$
Candidate tests
Candidate initial conditions
x
y
time
*
Inference Process
Outputs (sensors)
Initial Conditions (actuators)
b
c
a

b The inference process generates several *different* candidate symbolic models that match sensor data collected while performing previous tests. It does not know which model is correct·

c The inference process generates several possible new candidate tests that disambiguate competing models (make them disagree in their predictions).

a The inference process physically performs an experiment by setting initial conditions, perturbing the hidden system and recording time series of its behavior. Initially, this experiment is random; subsequently, it is the best test generated in step **c**.

Fig. 1. A concept diagram showing flow of the interrogation algorithm. The cycle (a, b, c) is repeated until a fixed period has elapsed. The dotted arrow in c represents the divergence between two models' predictions about the future behavior of state variable *y*. The best test from c that is executed on the physical system is the set of initial conditions that causes the models to disagree the most in their predictions (marked by an asterisk). The framework converges on accurate models by gradually accumulating information about the system from a series of internally generated, intelligent tests. By periodically discarding older tests and replacing them with newer ones, the algorithm cannot only model static systems, but continuously track changing systems by producing models of its current state.

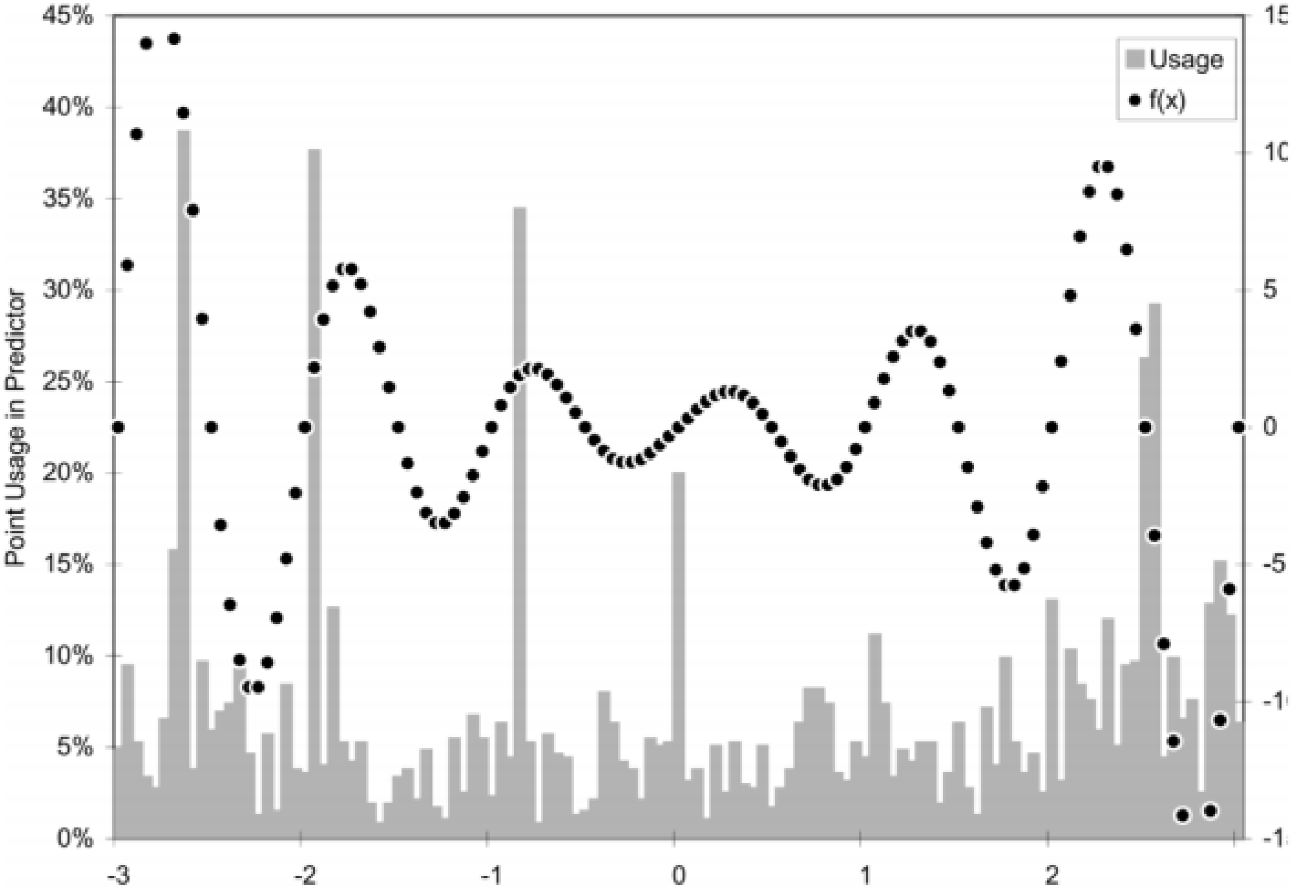
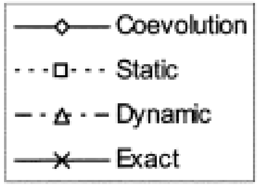
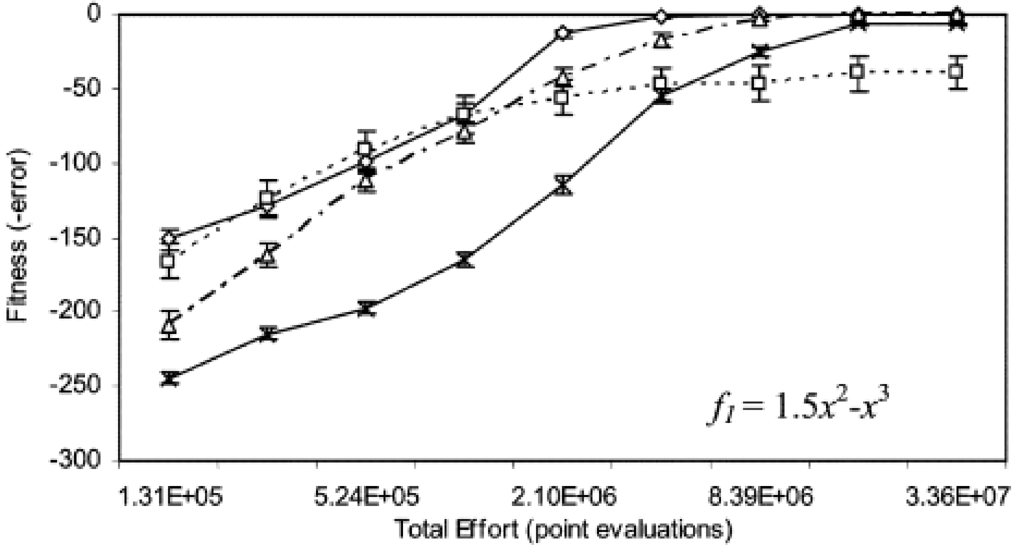


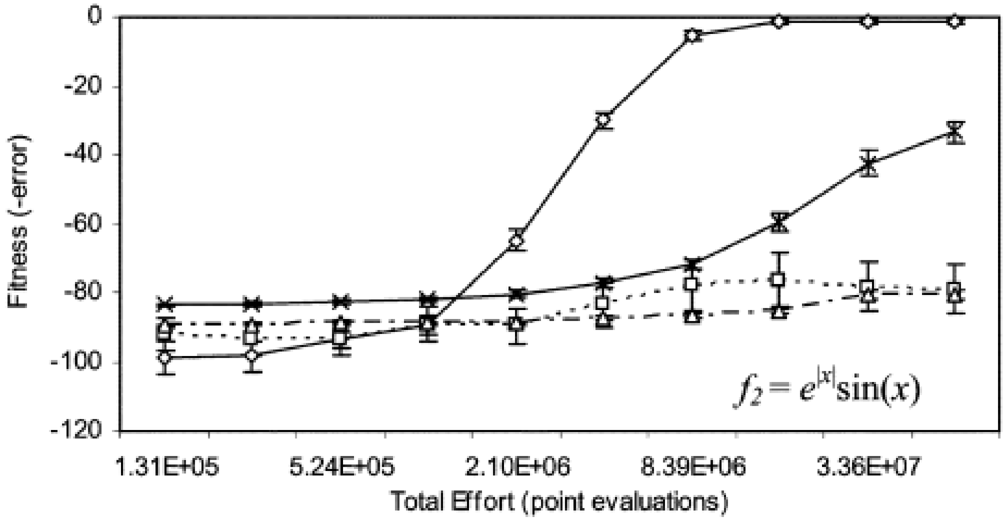
Fig. 4. Histogram of training samples selected by the best fitness predictor during evolution to convergence of $f = e^{|x|}\sin(x)$. Some samples are selected significantly more often that others.

TABLE I
SUMMARY OF THE COMPARED FITNESS PREDICTION STRATEGIES

Strategy	Sample Size	Sample Selection Method
Coevolved Predictor Sample	8	Evolved subset
Static Random Sample	8	Random subset chosen at runtime
Dynamic Random Sample	8	Changing random subset
Exact Fitness	200	Use all training data







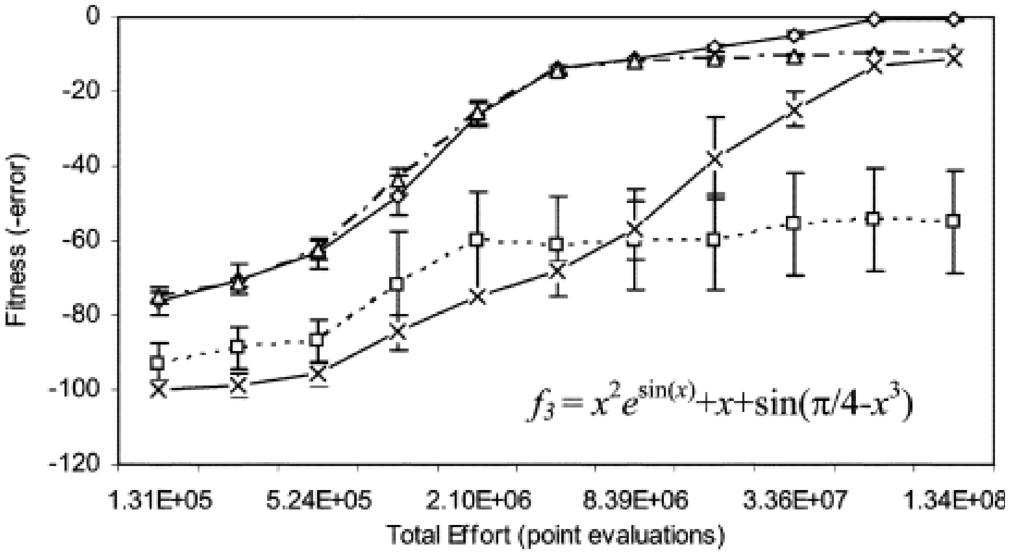
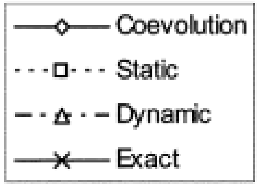
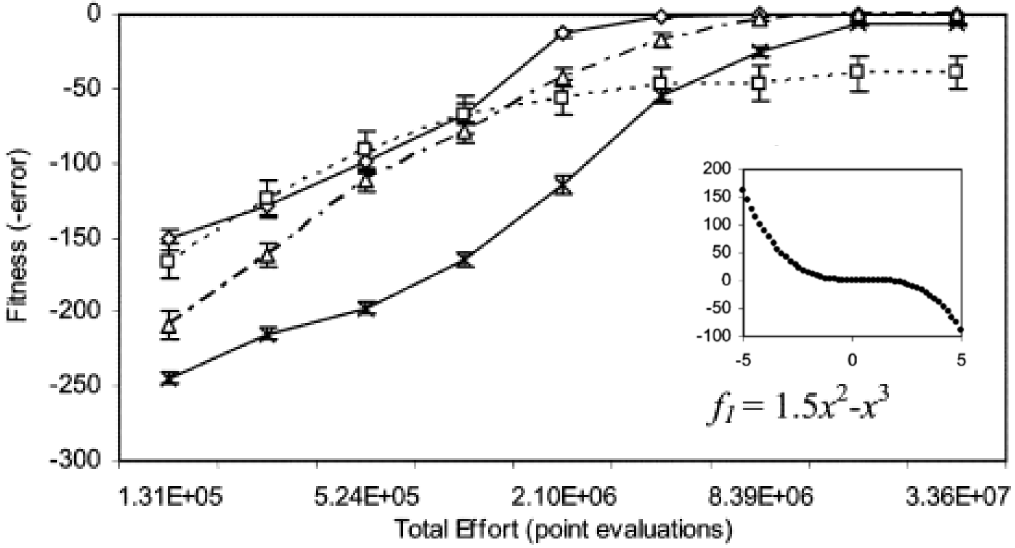
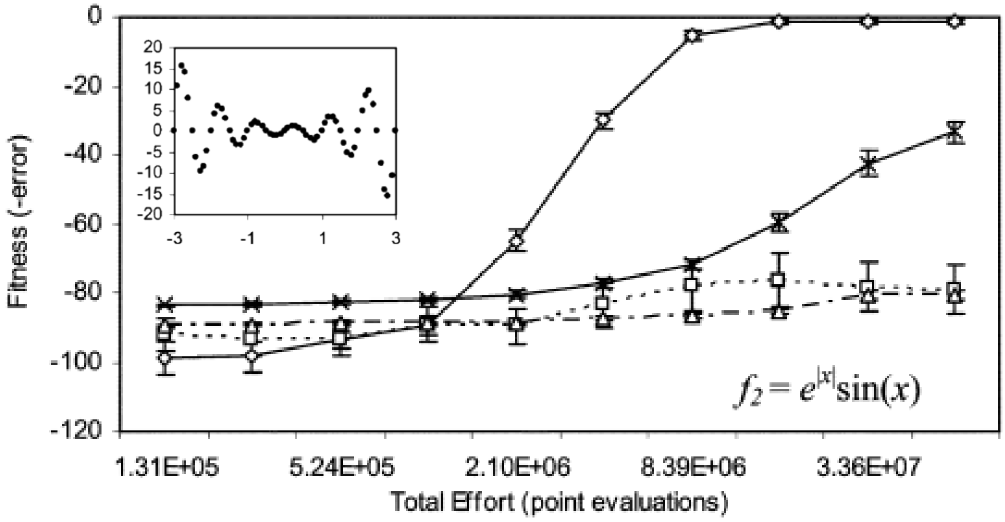


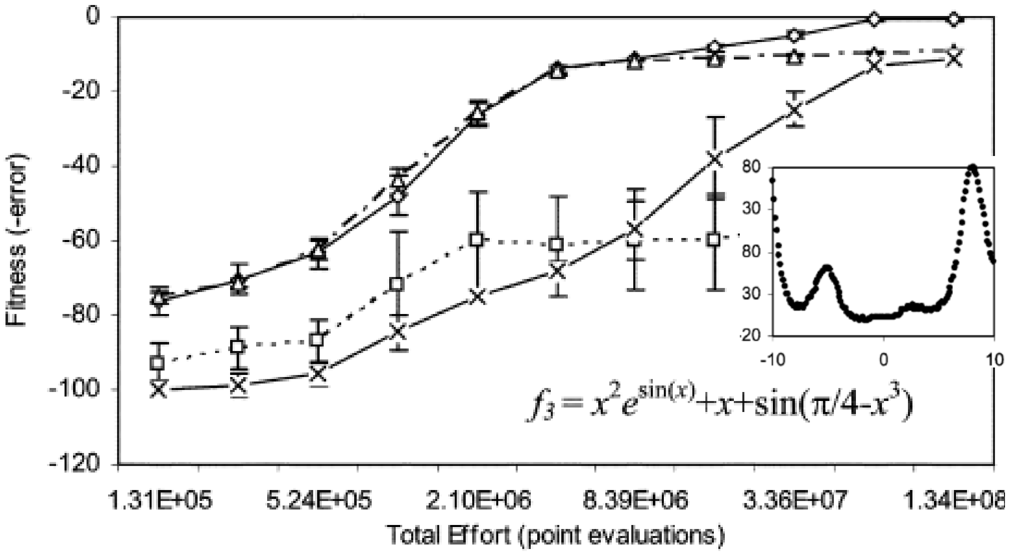
TABLE I
SUMMARY OF THE COMPARED FITNESS PREDICTION STRATEGIES

Strategy	Sample Size	Sample Selection Method
Coevolved Predictor Sample	8	Evolved subset
Static Random Sample	8	Random subset chosen at runtime
Dynamic Random Sample	8	Changing random subset
Exact Fitness	200	Use all training data









Examples of Randomly Generated Functions and Their Complexities

Random Function	Complexity
$f(x) = x$	1
$f(x) = x^2 - x$	5
$f(x) = \sin(\cos(x))\cdot(\exp(x) - \cos(x))$	11
$f(x) = \exp((\lvert x\rvert + \exp(x)))/((\exp(x) + \sin(x)) - \lvert(x/x)\rvert)$	23
$f(x) = \log(\cos(x + (\exp(\sin(x)\cdot\lvert x\rvert)\cdot (\sin(x\cdot\log(x)) + \exp(\cos(x))))))$	37

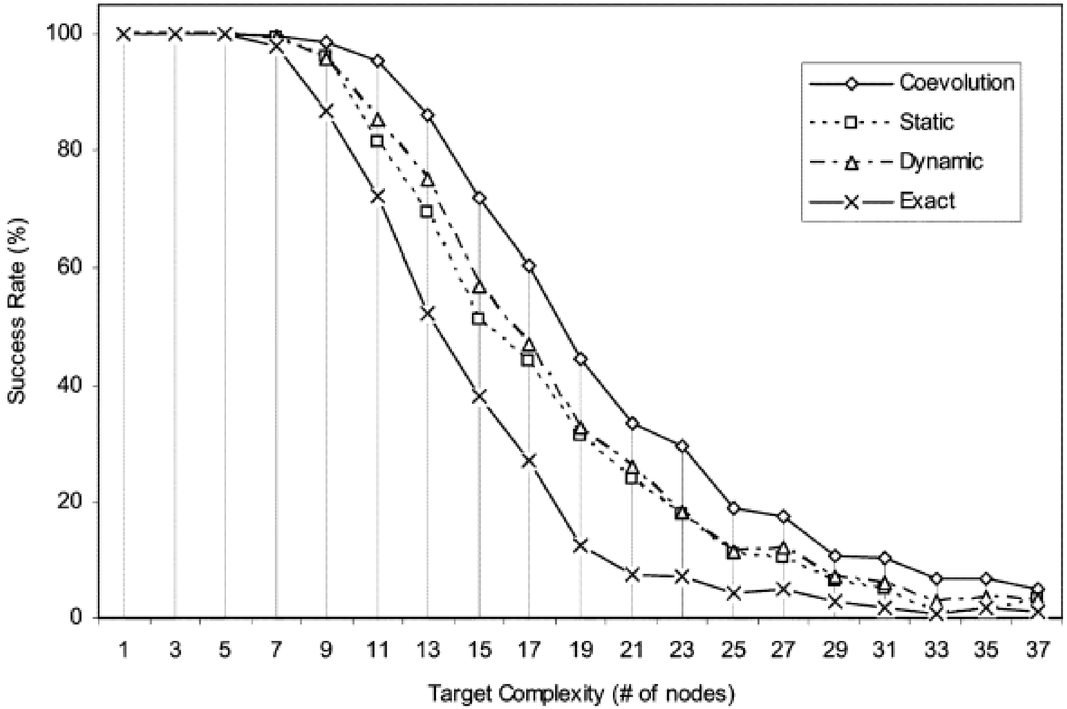


Fig. 9. The percent of successful convergence after 10 M point evaluations versus the target function complexity (the number of nodes in the binary expression tree).

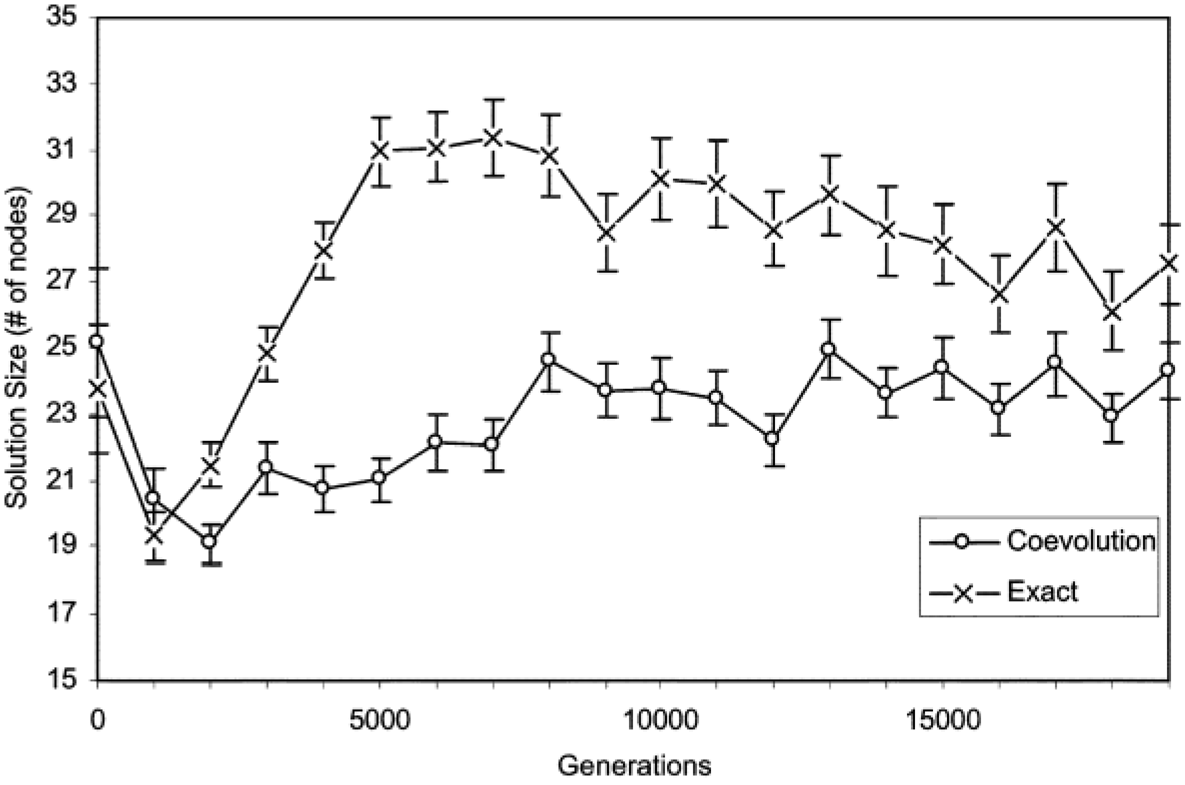
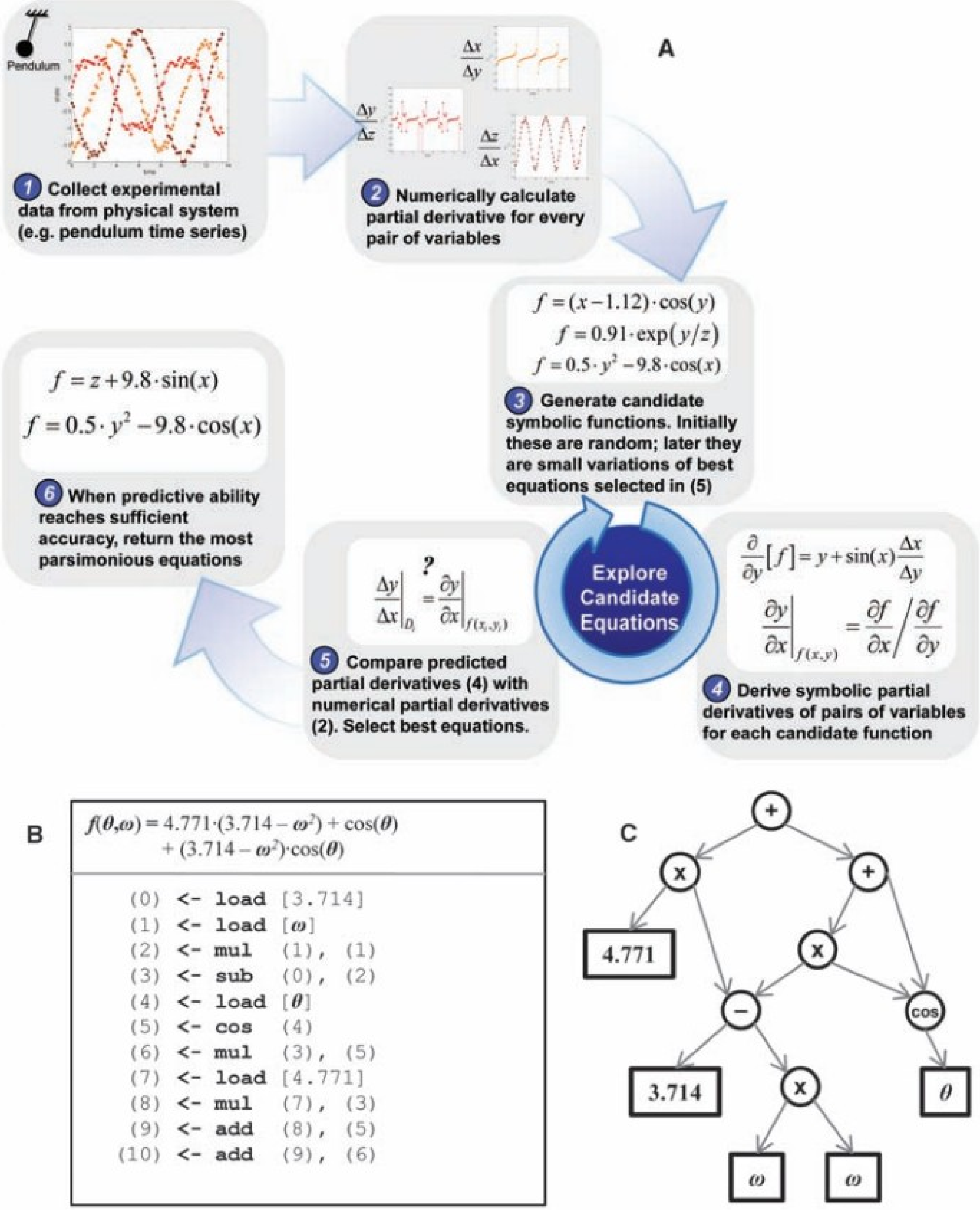


Fig. 12. The size of the best solution during evolution of $f_2(x)$ averaged over 100 test runs. Error bars show the standard error.

Distilling Free-Form Natural Laws from Experimental Data

Michael Schmidt[1] and Hod Lipson[2,3]*



1 Collect experimental data from physical system (e.g. pendulum time series)

2 Numerically calculate partial derivative for every pair of variables

$$\frac{\Delta x}{\Delta y} \qquad \frac{\Delta y}{\Delta z} \qquad \frac{\Delta z}{\Delta x}$$

$$f = (x - 1.12) \cdot \cos(y)$$
$$f = 0.91 \cdot \exp(y/z)$$
$$f = 0.5 \cdot y^2 - 9.8 \cdot \cos(x)$$

3 Generate candidate symbolic functions. Initially these are random; later they are small variations of best equations selected in (5)

$$\frac{\partial}{\partial y}[f] = y + \sin(x)\frac{\Delta x}{\Delta y}$$
$$\left.\frac{\partial y}{\partial x}\right|_{f(x,y)} = \frac{\partial f}{\partial x} \Big/ \frac{\partial f}{\partial y}$$

4 Derive symbolic partial derivatives of pairs of variables for each candidate function

Explore Candidate Equations

$$\left.\frac{\Delta y}{\Delta x}\right|_{D_i} \overset{?}{=} \left.\frac{\partial y}{\partial x}\right|_{f(x_i,y_i)}$$

5 Compare predicted partial derivatives (4) with numerical partial derivatives (2). Select best equations.

$$f = z + 9.8 \cdot \sin(x)$$
$$f = 0.5 \cdot y^2 - 9.8 \cdot \cos(x)$$

6 When predictive ability reaches sufficient accuracy, return the most parsimonious equations

$$f(\theta,\omega) = 4.771 \cdot (3.714 - \omega^2) + \cos(\theta) + (3.714 - \omega^2) \cdot \cos(\theta)$$

```
 (0) <- load [3.714]
 (1) <- load [ω]
 (2) <- mul  (1), (1)
 (3) <- sub  (0), (2)
 (4) <- load [θ]
 (5) <- cos  (4)
 (6) <- mul  (3), (5)
 (7) <- load [4.771]
 (8) <- mul  (7), (3)
 (9) <- add  (8), (5)
(10) <- add  (9), (6)
```

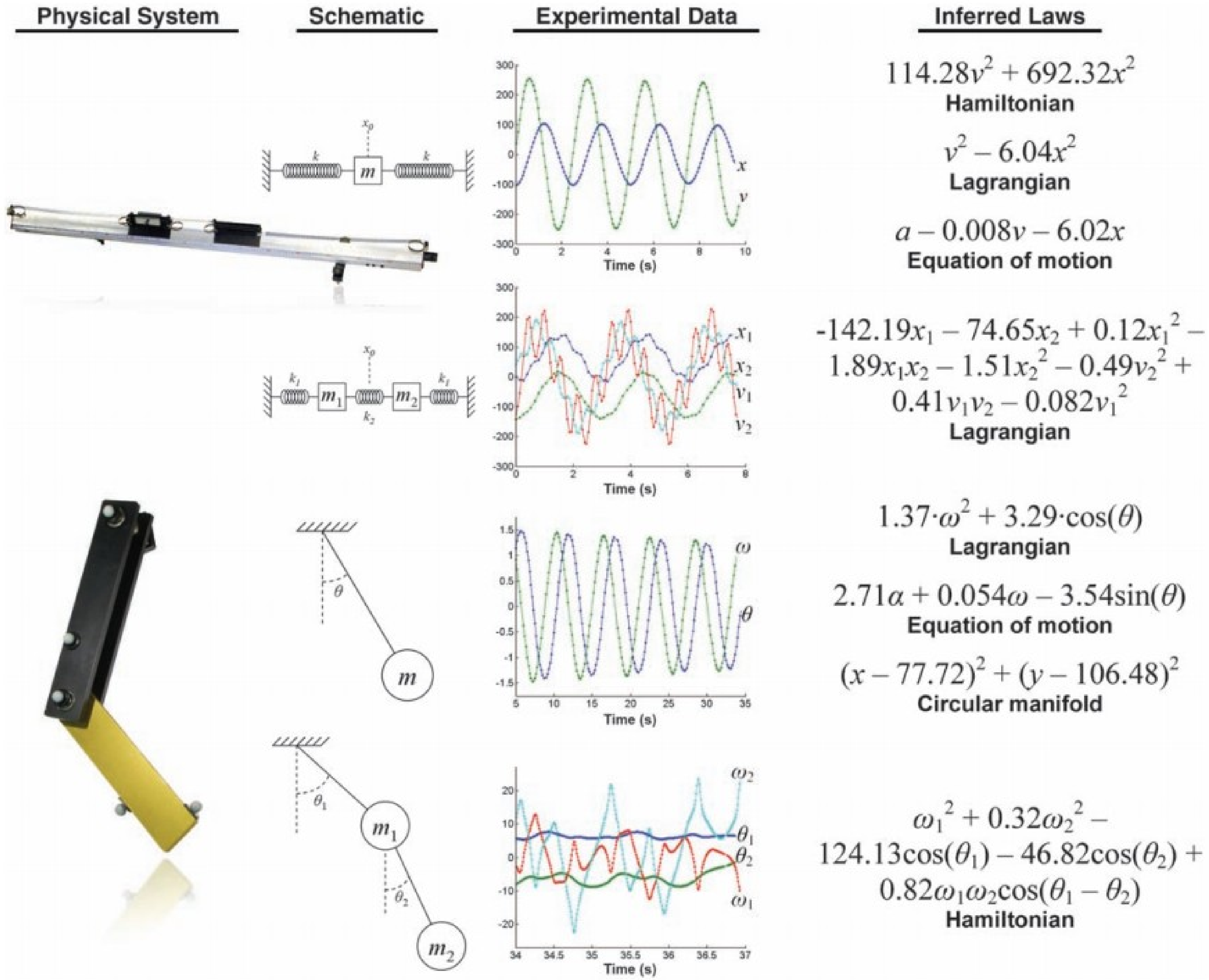


Fig. 3. Summary of laws inferred from experimental data collected from physical systems. Depending on the types of variables provided to the algorithm, it detects different types of laws. Given solely position information, the algorithm detects position manifolds; given velocities, the algorithm detects energy laws; given accelerations, it detects equations of motion and sum of forces laws (θ, angle; ω, angular velocity; α, angular acceleration).

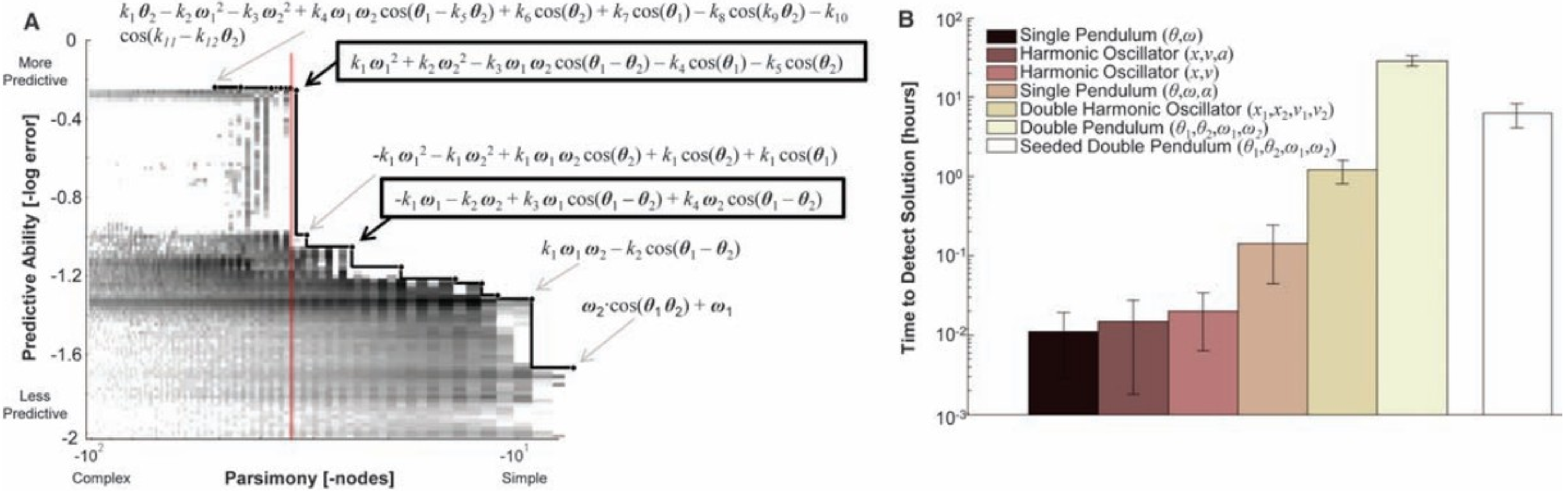
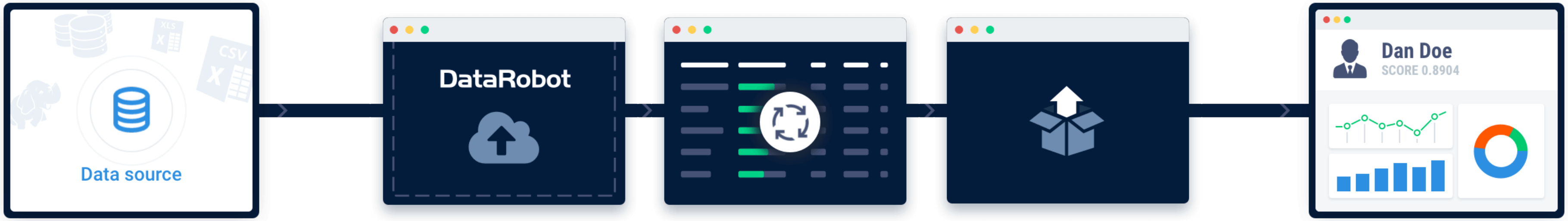


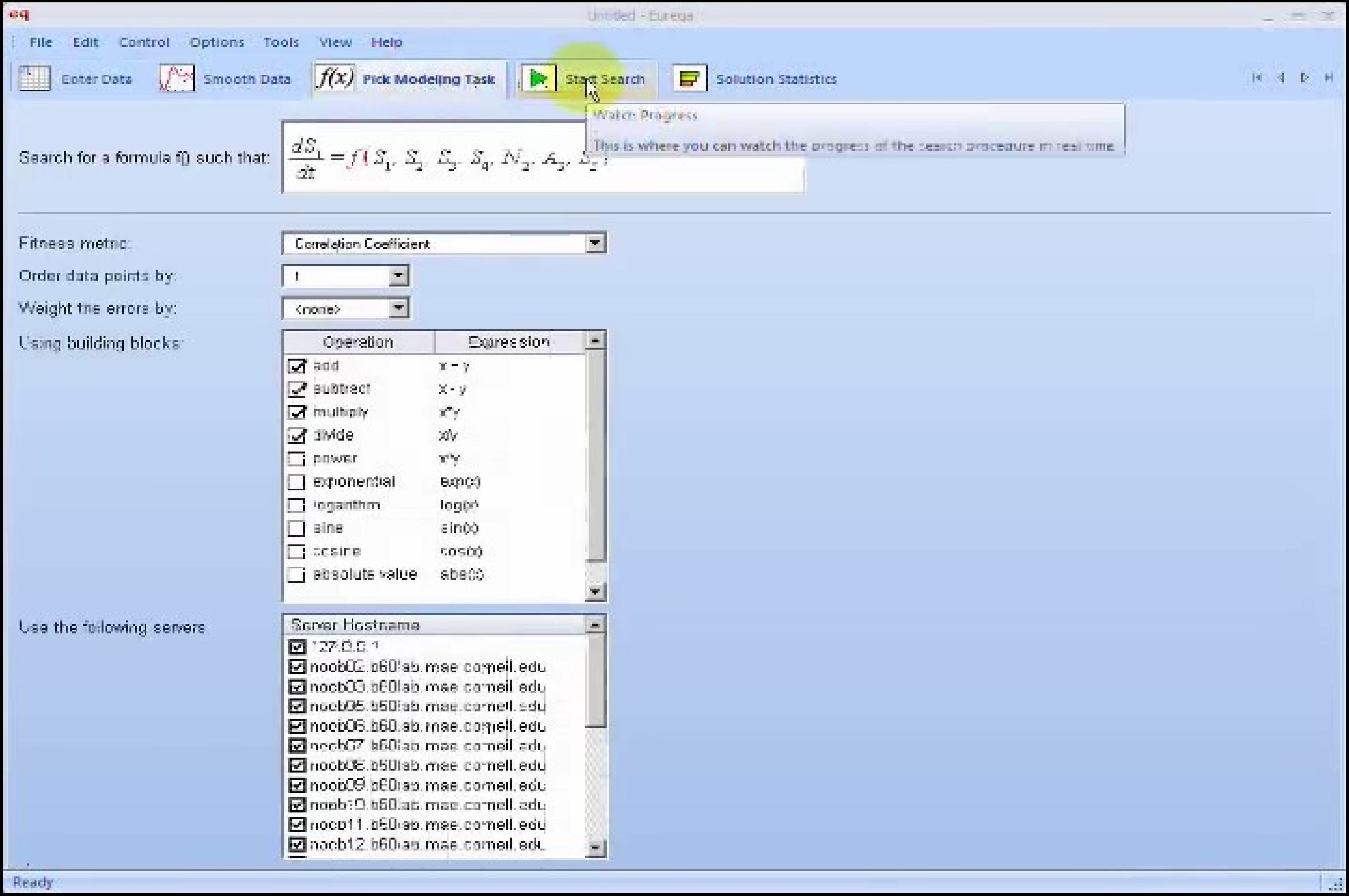
Fig. 4. Parsimony versus accuracy and computation time. (**A**) Pareto front (solid black curve) for physical laws of the double pendulum and the frequency of sampling during the law equation search (grayscale). The equation at the cliff corresponds to the exact energy conservation law of the double pendulum (highlighted in the figure). A second momentum conservation law that we encountered is also highlighted. (**B**) Computation time required to detect different physical laws for several systems. The computation time increases with the dimensionality, law equation complexity, and noise. A notable exception is the bootstrapped double pendulum, where reuse of terms from simpler systems helped reduce computational cost by almost an order of magnitude, suggesting a mechanism for scaling higher complexities.

eureqa

nutonian

DataRobot





Watch Progress

This is where you can watch the progress of the search procedure in real time.

Search for a formula f() such that: $\frac{dS_1}{dt} = f(S_1, S_2, S_3, S_4, N_2, A_3, S_2$

Fitness metric: Correlation Coefficient

Order data points by: t

Weight the errors by: <none>

Using building blocks:

	Operation	Expression
☑	add	x + y
☑	subtract	x - y
☑	multiply	x*y
☑	divide	x/y
□	power	x^y
□	exponential	exp(x)
□	logarithm	log(x)
□	sine	sin(x)
□	cosine	cos(x)
□	absolute value	abs(x)

Use the following servers

	Server Hostname
☑	127.0.0.1
☑	noob02.b60lab.mae.cornell.edu
☑	noob03.b60lab.mae.cornell.edu
☑	noob05.b60lab.mae.cornell.edu
☑	noob06.b60lab.mae.cornell.edu
☑	noob07.b60lab.mae.cornell.edu
☑	noob08.b60lab.mae.cornell.edu
☑	noob09.b60lab.mae.cornell.edu
☑	noob10.b60lab.mae.cornell.edu
☑	noob11.b60lab.mae.cornell.edu
☑	noob12.b60lab.mae.cornell.edu