Supplementary Materials for

Symbolic regression for empirically realistic population dynamic time series

Cheyenne N. Jarman, Taal Levi & Mark Novak

Contents

A. Symbolic regression and PySR

Symbolic regression

Symbolic regression aims to find simple yet accurate equations to explain a given dataset. Using a process akin to biological evolution, generations of equations are evolved, where individual equations with a better fit to the data have a higher chance of producing offspring equations in the following generation. Symbolic regression begins with the creation of an initial generation of equations. Two equations with high fitness are selected as "parental equations". These parental equations produce an offspring equation for the next generation of equations through additions, deletions, recombinations, or mutations of their components. Multiple pairs of parental equations are selected within any given generation. Additionally, equations from previous generations may move forward into subsequent generations. This equation evolution process occurs for a set amount of time or generations. This algorithm can be implemented in various ways – the four previous ecological studies utilized custom-built algorithms. We implemented symbolic regression with PySR (26), an open-source Python library with a Julia backend. We will describe symbolic regression through the PySR lens, noting that while there are overarching similarities in the workflow, specific implementation details differ from other algorithms.

PySR begins with the user providing a list of predictor variables, a single response variable, and a suite of mathematical operators (both built-in and custom, user-defined operators) to combine terms (Fig. 2b). PySR differs from other symbolic regression algorithms in that evolution can occur within and between populations of equations (Fig. 2c). The number of populations is pre-specified as a user-input argument. PySR initiates the specified number of populations, each containing an initial generation of equations created through random assemblages of predictor variables, parameters, and mathematical operators. Within each population, equation evolution occurs through tournament selection, where the population is randomly subsampled and the individuals with the highest fitness values are selected as parental candidates (26). Mean squared error (MSE) is commonly used as the fitness metric, thus the equation with the highest fitness will have the lowest MSE. The two parental candidates exchange subexpressions of their equation to produce a new equation via crossover. Mutation can also occur and only needs a single parental candidate. In this case, a subexpression of the parental equation is modified (e.g., a change in a parameter value or the operator used). Equation evolution is also done between populations (i.e., gene flow) where equations from one population randomly migrate to

another population and vice versa, allowing for potentially new equation genes to enter a given population (Fig. 2c) (26). PySR's output is a file containing the best-performing equations and the respective MSE across a range of complexity levels, where complexity is defined as the sum of the numbers of parameters, variables, and operators present in a given equation (Fig. 2d). To help with equation selection, the user can plot the equation's MSE against the complexity level (Fig. 2e).

PySR

PySR allows for the specification and customization of many different arguments. We specified that the following operators could be used: addition, subtraction, multiplication, division, and the exponential function. We also created a custom function that matched the non-linear function, $[\,]_+$. This was the only custom function we used. PySR's *populations* argument defines how many populations of equations are evolving. More populations will increase the diversity of equations and run time (26). We used 96 populations in all analyses. The user can also specify how many iterations, or the total number of generations, PySR will run for. The PySR documentation recommends setting this argument to a high value, so we specified a maximum of $1 \cdot 10^{10}$ iterations. Finally, the user can control how much time PySR spends searching by specifying the timeout argument. We specified our search to run for 22 hours. Equation overfitting and bloating are known concerns for symbolic regression algorithms. PySR manages these concerns by defining a maximum equation size (user-specified or default value 30, we used the latter). Additionally, PySR ensures relatively even equation complexity representation by tracking the frequency of particular complexity levels and the complexity levels of the most recent equations (26). While there is no clear recommendation for the optimal number of runs needed to ensure the global optimum is reached, we ran each simulated data set through PySR independently 100 times. Previous studies also conducted (but combined) multiple search runs, with the number ranging from 3–30.

```
X = #array of predictor variables
y = #single response variable (e.g., log.ratio)

model = PySRRegressor(
early_stop_conditions = 2e-09,
timeout_in_seconds = 792000,
niterations = 1e10,
populations = 96,
ncyclesperiteration = 5000,
loss = "L2DistLoss()",
binary_operators=["+", "-", "*", "/"],
unary_operators=["exp","mygreater(x) = x >= 0 ? 1.0f0 : 0.0f0"],
extra_sympy_mappings={
  "mygreater": lambda x:
      Piecewise((1, Eq(x, 0) | (x > 0)), (0, True)), }, )
model.fit(X, y)
```

B. Parameters used to simulate time series

Our modifiication of the (20) model incorporated process noise as

$$\frac{dA}{dt} = (se^{\epsilon_1})e^{-\alpha\tau}\left[1 - a_A A(t-\tau)\right]_+ - (me^{\epsilon_2})A(t) \tag{S.1}$$

where ϵ_1 and ϵ_2 are random variables with $\epsilon_i \sim \mathcal{N}(0, \sigma_i)$ and all parameter values are given in Tables S.1 & S.2.

Table S.1: Parameter values used to generate time series.

Parameter	Symbol	Symmetric	Asymmetric
Juvenile settlement rate	s	55	55
Juvenile mortality rate	α	0.35	0.15
Space occupied by individual adult	a_A	0.1	0.35
Adult mortality rate	m	1	0.2
Juvenile maturation delay	τ	2	2

Table S.2: Parameter values used to introduce process noise.

Parameter	σ_1	σ_2
No noise	0	0
Low	0.05	0.2
High	0.1	0.8

C. Time series period lengths and aliasing

The period length of cycles produced by the model for each of the four cases was determined by Fourier analysis using *SciPy*. In the deterministic cases, the period lengths were 5.5 and 13.5 for the symmetric and asymmetric cases, respectively. For the stochastic cases, we simulated the time series for 1000 time steps and took the last 500 time steps to average the period length, which was 13.2 and 12.5 time steps for the symmetric stochastic cases under low and high process noise, respectively.

Since in every case we examined the cycles had periods of at least five sampling intervals — i.e., we observed a minimum of five time points per cycle — their fundamental frequencies were well below the Nyquist-Shannon cutoff (38). Consequently, the down-sampled series are unlikely to suffer aliasing; only if very high-frequency harmonics exceeded half the sampling rate would they have risked being reflected at lower apparent frequencies.

D. Asymptotic equivalence of discrete and continuous per capita growth rates.

The log ratio $\frac{N_{t+\Delta t}}{\Delta t}$ is asymptotically equivalent to the instantaneous per capita growth rate $\frac{1}{N}\frac{dN}{dt}$ as Δt approaches 0. That is,

$$\lim_{\Delta t \to 0} \frac{\ln\left(\frac{N_{t+\Delta t}}{N_t}\right)}{\Delta t} = \frac{1}{N}\frac{dN}{dt}\,. \tag{S.2}$$

This may be shown as follows.

Using the quotient rule for logs, we can rewrite the log ratio as the difference between $N_{t+\Delta t}$ and N_t,

$$\lim_{\Delta t \to 0} \frac{\ln\left(\frac{N_{t+\Delta t}}{N_t}\right)}{\Delta t} = \lim_{\Delta t \to 0} \frac{\ln(N_{t+\Delta t}) - \ln(N_t)}{\Delta t}\,. \tag{S.3}$$

The term on the right-hand side is equivalent to the definition of a derivative, thus we can rewrite the expression as

$$\lim_{\Delta t \to 0} \frac{\ln(N_{t+\Delta t}) - \ln(N_t)}{\Delta t} = \frac{d}{dt}\ln(N_t)\,. \tag{S.4}$$

Lastly, applying the chain rule shows how the time derivative of the logarithm of population size relates to the instantaneous per capita growth rate. Specifically, since $\ln(N_t)$ is a function of N_t, which itself varies with time, we differentiate $\ln(N_t)$ with respect to N_t and then multiply by the rate at which N_t is changing over time. Because the derivative of $\ln(N_t)$ is $\frac{1}{N_t}$, this yields

$$\frac{d}{dt}\ln(N_t) = \frac{1}{N}\frac{dN}{dt}\,. \tag{S.5}$$

E. Equation recovery when using only $A(t)$ and $A(t-2)$

We simulated the Bence-Nisbet model using parameter values that generated asymmetric cycles (as in Table S.1) and provided PySR with data downsampled to a sampling density of 100 time points per cycle with the per capita growth rate using either the continuous-time or the discrete-time preprocessing approach. We performed 100 independent runs of PySR for each time series. For each run, we evaluated all resulting equations and ranked them according to their MSE. We then examined the rankings to determine whether the Bence-Nisbet model was present (Fig. S.1). In both the continuous- and discrete-time approaches, the Bence-Nisbet model was recovered among the top 10 ranked models.

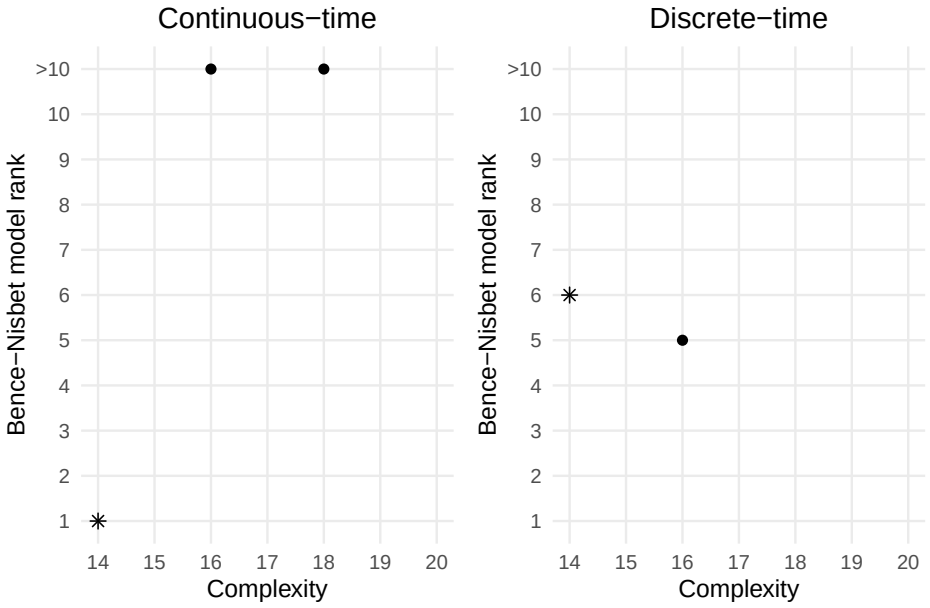

Figure S.1: The maximum MSE-rank of the Bence-Nisbet model when it was recovered, returned by the 100 runs from the asymmetric, 100 points per cycle data, with a continuous-time and discrete-time approach, respectively. We assessed the complexity range from 10 to 20. The lowest complexity that the Bence-Nisbet model occurred was complexity 14, which is the complexity level of the Bence-Nisbet model is 14 (indicated by asterisks). Ranks at complexities greater than 14 (indicated by points) occurred when the equations could be algebraically simplified to the Bence-Nisbet model.

F. Attractor reconstruction

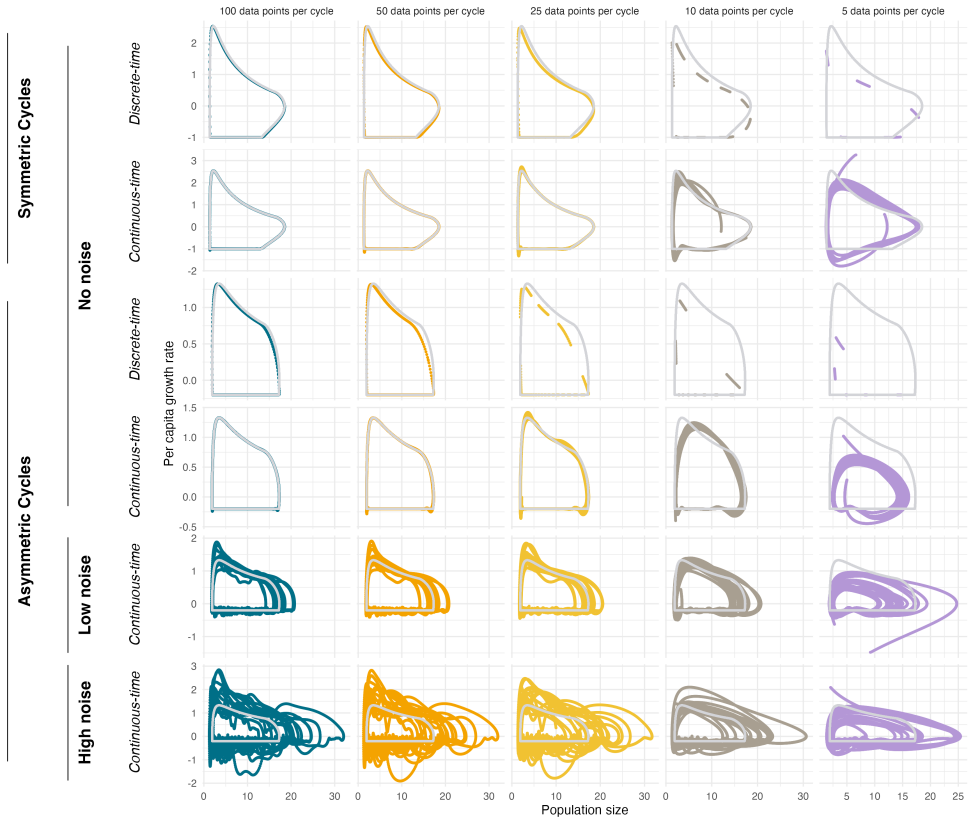

Figure S.2: Phase portraits of population size and per capita growth rate (log ratio or interpolated for discrete-time or continuous-time, respectively) provide insight into the influence of sampling density on symbolic regression's success. As sampling density decreases, the apparent attractor tends to shrink relative to the true attractor of the Bence-Nisbet model (overlaid in light gray), except for the cases with process noise.