Title

Optimizing differential equations to fit data and predict outcomes

Permalink

https://escholarship.org/uc/item/65p1p33b

Journal

Ecology and Evolution, 13(3)

ISSN

2045-7758

Author

Frank, Steven A

Publication Date

2023-03-01

DOI

10.1002/ece3.9895

Peer reviewed

RESEARCH ARTICLE

Ecology and Evolution
Open Access WILEY

Optimizing differential equations to fit data and predict outcomes

Steven A. Frank [ORCID]

Department of Ecology and Evolutionary Biology, University of California, Irvine, California, USA

Correspondence
Steven A. Frank, Department of Ecology and Evolutionary Biology, University of California, Irvine, CA 92697-2525, USA.
Email: saf@stevefrank.org

Abstract

Many scientific problems focus on observed patterns of change or on how to design a system to achieve particular dynamics. Those problems often require fitting differential equation models to target trajectories. Fitting such models can be difficult because each evaluation of the fit must calculate the distance between the model and target patterns at numerous points along a trajectory. The gradient of the fit with respect to the model parameters can be challenging to compute. Recent technical advances in automatic differentiation through numerical differential equation solvers potentially change the fitting process into a relatively easy problem, opening up new possibilities to study dynamics. However, application of the new tools to real data may fail to achieve a good fit. This article illustrates how to overcome a variety of common challenges, using the classic ecological data for oscillations in hare and lynx populations. Models include simple ordinary differential equations (ODEs) and neural ordinary differential equations (NODEs), which use artificial neural networks to estimate the derivatives of differential equation systems. Comparing the fits obtained with ODEs versus NODEs, representing small and large parameter spaces, and changing the number of variable dimensions provide insight into the geometry of the observed and model trajectories. To analyze the quality of the models for predicting future observations, a Bayesian-inspired preconditioned stochastic gradient Langevin dynamics (pSGLD) calculation of the posterior distribution of predicted model trajectories clarifies the tendency for various models to underfit or overfit the data. Coupling fitted differential equation systems with pSGLD sampling provides a powerful way to study the properties of optimization surfaces, raising an analogy with mutation-selection dynamics on fitness landscapes.

KEYWORDS
artificial neural networks, automatic differentiation, Bayesian analysis, differential equation models, ecological and evolutionary dynamics, Julia programming language, Langevin dynamics, optimization

TAXONOMY CLASSIFICATION
Theorectical ecology

1 | INTRODUCTION

Much of science describes or predicts how things change over time. Differential equations provide a common model for fitting data and predicting future observations. Optimizing a differential equation model is challenging. Each observed or predicted point along a temporal trajectory is influenced by the potentially large set of parameters that define the model. Optimizing a model means improving the match between the model's trajectory and the observed or desired temporal path at many individual points in time.

Recent advances in machine learning have greatly improved the potential to optimize differential equation models (Bonnaffé et al., 2021; Chen et al., 2018; Rackauckas et al., 2020). However, when applying the methods to actual data, it is often not so easy to realize the promise of the new conceptual advances and software packages.

This article illustrates how to fit ordinary differential equation (ODE) models to noisy time series data. The fitted models are also sampled to develop an approximate Bayesian posterior distribution of trajectories. The posterior distribution provides a way to evaluate confidence in the fit to observed data and in the prediction of future observations.

I use the classic data for the fluctuations of lynx and hare populations, an example of predator–prey dynamics (Odum & Barrett, 1971). This example illustrates the challenges that arise when fitting models for any scientific problem that can be analyzed by simple deterministic ODEs.

My work follows on the excellent recent article by Bonnaffé et al. (2021). They fit neural ODEs (NODEs) to the hare-lynx data. NODEs use neural networks to fit time series data to the temporal derivatives of variables, in other words, NODEs estimate ODEs by using modern neural networks (Chen et al., 2018). Bonnaffé et al. (2021) emphasized that NODEs have the potential to advance many studies of ecological and evolutionary dynamics. However, they encountered several practical challenges in their application of NODEs to the hare-lynx data, ultimately concluding that "it is our view that the training of these models remains nonetheless intensive."

Challenges with Bonnaffé et al.'s method include long computation times, limited flexibility with regard to studying alternative or larger models within the same computational framework, and lack of connection to rapidly developing technical advances in automatic differentiation (Frank, 2022a). A recent update by Bonnaffé and Coulson (2022) provides an alternative solution for some of these challenges.

2 | MATERIALS AND METHODS

Extending Bonnaffé et al. (2021), I advance practical aspects of fitting ODE and NODE models. I show that several computational techniques provide a relatively easy way to fit and interpret such models. The computer code provides the specific methods by which I achieve each advance. Here, I emphasize six points.

2.1 | Comparing NODE and ODE models

First, my computer code provides a switch between fitting a high-dimensional NODE or a simple low-dimensional ODE to the same data. *Dimensionality* here refers to the size of the parameter space. The switch makes it easy to compare the two types of model.

Conceptually, both NODE and ODE models are basic systems of ordinary differential equations. In practice, the difference concerns the typically much greater dimensionality of the NODE models and the wide variety of high-quality tools available to build, compute, and evaluate complex neural network architectures. The NODE models usually have much greater flexibility and power to fit complex patterns but also suffer from computational complexity and a tendency to overfit data.

2.2 | Dummy variables

Second, I evaluate the costs and benefits of adding dummy variables to the fitting process (Dupont et al., 2019). For example, there are two variables in the case of hare and lynx. To those two variables, we may add additional variables to the system. We may think of those additional dummy variables as unobserved factors (Bonnaffé et al., 2021). For example, if there is one additional factor that significantly influences the dynamics, then trying to fit a two-dimensional model to the data will be difficult because the actual trajectories trace pathways in three dimensions. *Dimensionality* here refers to the number of variables in the system.

2.3 | Data smoothing

Third, I smoothed the original data before fitting the models. Deterministic models can only fit the general trends in the data. Stochastic fluctuations may interfere with the fitting process, which typically gains from a smoother cost function, in which the cost function decreases as the quality of the fit improves. To reduce large fluctuations, I first log-transformed the data and normalized by subtracting the mean for each variable of the times series (Bonnaffé et al., 2021). Then, to emphasize trends in the data, I smoothed the time series with a cubic spline. I added an interpolated time point between each pair of observed time points. I then smoothed the augmented data with a Gaussian filter. Figure 1 shows the original and the smoothed data.

2.4 | Sequential fitting

Fourth, simultaneously fitting all points in the time series may prevent finding a good fit. The complexity of the optimization surface may be too great when starting from random parameters. Sequential fitting can help, first fitting the initial part of the time series, then adding later time points in a stepwise manner. However, when adding additional points, weighted equally with prior points, a strong
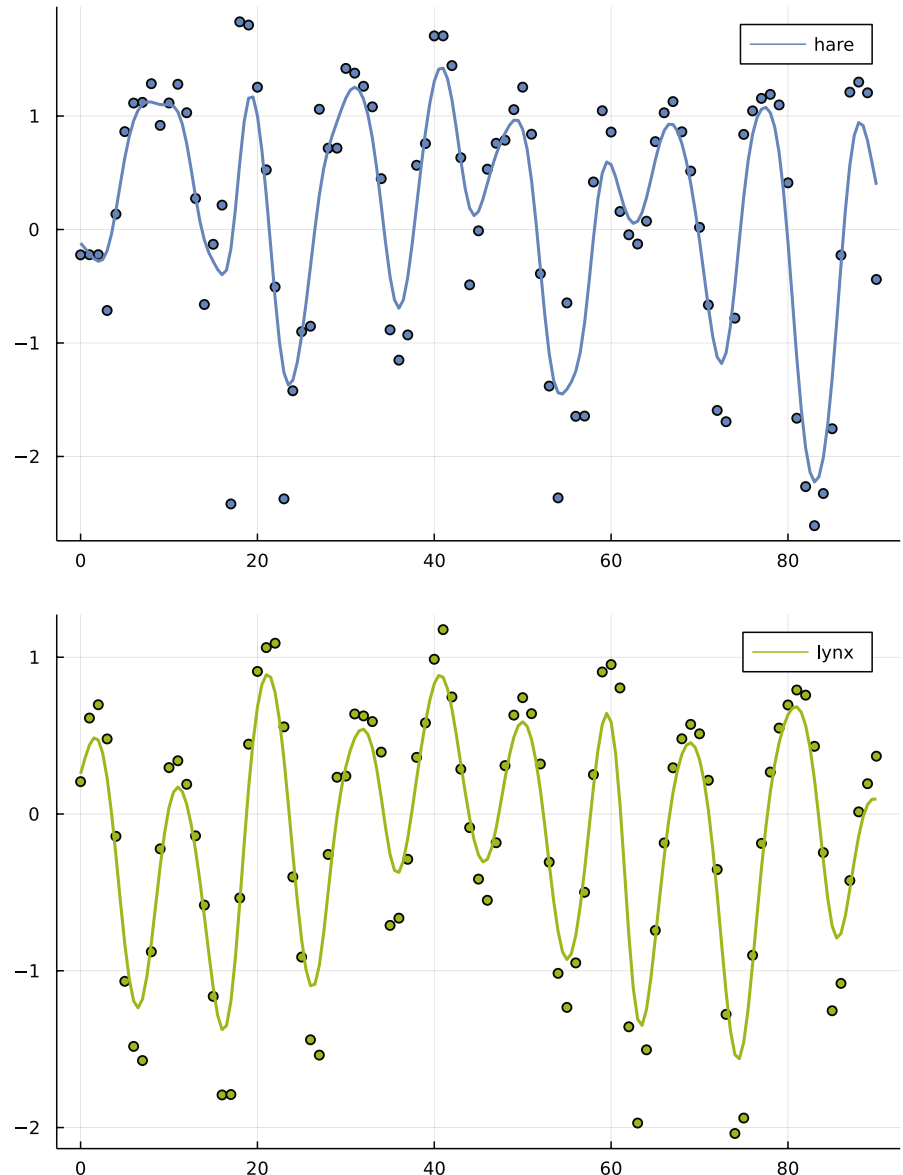
discontinuity may arise in the fitting process. That discontinuity may push the fitting process too far away to recover a smooth approach to a good fit. To fix that issue, I slowly increased the weighting of later points, which provided greater continuity in the optimization process and better convergence to relatively good fits.

2.5 | Approximate Bayesian posterior

Fifth, I estimate a distribution of temporal trajectories for a fitted model by analogy with sampling the Bayesian posterior distribution of the model parameters. The distribution of trajectories provides a measure of the confidence in the quality of a fit to the data and of predictions for future unobserved observations.

To sample the posterior distribution of fitted parameters and associated trajectories, I first fit the model by standard gradient descent methods, using the Adam learning algorithm (Kingma & Ba, 2014). Then, with the fitted model as an initial condition, I

calculated the preconditioned stochastic gradient Langevin dynamics (pSGLD) (Li et al., 2015). In essence, a deterministic force moves each parameter toward a locally better fit, and a stochastic force causes parameter fluctuations.

For a gradient of the loss function with respect to the parameter, g, and a given hyperparameter, ϵ, the stochastic force dominates the deterministic force when $\epsilon g \ll 1$. Thus, when the fit is sufficiently near a local optimum and the associated gradient multiplied by ϵ is small, the model parameter value fluctuates randomly in a way that approximately samples the Bayesian posterior. Here, *hyperparameter* means a parameter that controls the fitting process rather than a fitted parameter of the model, following the common convention in the machine learning literature (Goodfellow et al., 2016).

In this study, I used a standard machine learning quadratic loss function (Goodfellow et al., 2016). In particular, the loss is the sum of squared deviations between each target time point in the smoothed data and the value of the model trajectory at that time point.

A quadratic loss associates with the Bayesian estimator for the mean of a parameter's posterior distribution given a non-informative prior. However, in this study, I used pSGLD to sample the distribution of parameters around a local optimum for a fitted model, in which each observation is a multidimensional parameter vector describing a differential equation model. That distribution provides a rough estimate of the confidence in the fitted parameter values and the associated trajectories, inspired by Bayesian principles rather than adhering strictly to the assumptions of Bayesian analysis.

2.6 | Julia computer language

Finally, I used the Julia programming language (Bezanson et al., 2017). Debate about choice of language often devolves into subjective factors. However, in my interpretation for fitting differential equation models, the current status of Julia provides several clear advantages.

Julia is much faster than popular alternatives, such as Python and R (Perkel, 2019). Speed matters, transforming difficult or essentially undoable optimizations into problems that can easily be solved on a standard desktop computer. I did all of the runs for this article on my daily working desktop computer using only the CPU, completing the most complex runs in at most a few hours, without any special effort to optimize the code or the process. Many useful runs for complex fits could be done in much less than an hour.

The Julia package DifferentialEquations.jl has a very wide array of numerical solvers for differential equations (Rackauckas & Nie, 2017). Using an appropriate solver with the correct tolerances is essential for fitting differential equations. I used the solvers Rodas4P for ODE problems and TRBDF2 for NODE problems. These solvers handle the instabilities that frequently arise when fitting oscillatory time series data. Further experimentation would be useful to test whether other solvers might be faster or handle instabilities better.

Efficient optimization of large models typically gains greatly from automatic differentiation (Baydin et al., 2018; Margossian, 2019). In essence, the computer code automatically analyzes the exact derivatives of the loss function with respect to each parameter, rapidly calculating the full gradient that allows the optimization process to move steadily in the direction that improves the fit (Frank, 2022a).

For fitting differential equations, the special challenge arises because each time point along the target trajectory to be fit must be matched by using the differential equation solver to transform the model parameters into a predicted time point along the calculated trajectory. That means that differentiating the loss function with respect to the parameters must differentiate through the numerical solver for the system of differential equations. It must do so for each target point. In this study, the target trajectory consisted of 181 time points for hare and lynx trajectories, requiring each calculation of a loss function or derivative of the loss to analyze the match between the data and 362 numerically evaluated trajectory points.

The Julia package DiffEqFlux.jl provides automatic differentiation through many different solvers (Rackauckas et al., 2020). Other languages provide similar automatic differentiation, but in my experience, the process is either much slower or more limited in a variety of ways. By contrast, the Julia package works simply and quickly, with many options to adjust the process.

The DiffEqFlux.jl package also provides a broad set of tools to build neural network models. Those models can easily be analyzed as systems that estimate differential equations, NODEs which can be optimized with a few lines of code (Rackauckas et al., 2019).

Documentation of the DiffEqFlux.jl package presents several examples of fitting differential equation models. However, the toy datasets do not bring out many of the challenges one faces when trying to fit the kinds of noisy data that commonly arise in practice. The methods discussed here may be broadly useful for many applications.

2.7 | Overview of the models

Each model has n variables, two for hare and lynx and $n-2$ for dummy variables tracking unobserved factors. The models seek to match the log-transformed and smoothed data shown in data. The differential equation for the vector of variables $\mathbf{u}$ in the ODE models has the form

$$\frac{d\mathbf{u}}{dt} = f(\mathbf{Su} - \mathbf{b}), \tag{1}$$

in which the $n^2 + n$ parameters are in the $n \times n$ matrix, $\mathbf{S}$, and the n vector, $\mathbf{b}$. The function f maps the n dimensional input to an n dimensional output, potentially inducing nonlinearity in the model. One typically selects f from the set of common activation functions used in neural network models. For all runs reported here, I used $f = \tanh$ applied independently to each dimension. I chose tanh because it is a common default in neural networks, can produce both positive and negative values, is centered at zero, and has a relatively steep gradient near zero (Sharma et al., 2020). In practice, it worked well for this study. With regard to Equation (1), it would be easy to study alternative ODE forms. However, in this article, I focus on Equation (1).

For each numerical calculation of a predicted trajectory, I set the initial condition to the data values for the first two dimensions. For the other $n-2$ dimensions, I set a random initial value at the start of an optimization run. My code includes the option to optimize the initial values for the $n-2$ dummy variables by considering those values as parameters of the model. However, in my preliminary studies, optimizing the dummy initial conditions did not provide sufficient advantages. I did not use that option in the runs reported here.

Typical NODE models are neural networks that take n inputs (Chen et al., 2018). The n outputs are the vector of derivatives, $d\mathbf{u}/dt$. Common neural network architectures provide a variety of systems for calculating outputs from inputs. The Julia software packages include simple ways to specify common and custom architectures.

I used a simple two-layer architecture, in which each of the n inputs flows to N internal nodes. Each internal node produces an output that is a weighted sum of the inputs, each weight a model

parameter, plus an additional parameter added as a constant. Each of those outputs is transformed by an activation function, f, for which I again chose $f = \tanh$. Those N outputs were then used as inputs into a second layer that produced n outputs as the weighted sum of inputs plus a constant. The second layer did not use an activation function to transform values.

This basic two-layer architecture has $(n + 1)N + n(N + 1)$ parameters. The runs in the following section used $N = 20$ and $n = 2, 3, 4$, yielding 102, 143, and 184 parameters for the increasing values of n.

The parameters and output for all computer models are included with the source code files.

3 | RESULTS

3.1 | ODE versus NODE, varying n

I fit ODE and NODE models for $n = 2, 3, 4$. For each of those six combinations, Figure 2 shows the model fits against the smoothed data for the full 90-year period of observations.

The hare and lynx estimated abundances comprise the $n = 2$ core variables of the analysis. Adding dummy variables to make $n > 2$ improves the fit. That improving fit with rising n can be seen in Figure 2 by the better match between the data and model trajectories as one moves down the fitting sets in each column.

NODE models fit better than ODE models, associated with the greater number of parameters in NODE models. The better fits can be seen by comparing the NODE sets in the right column against their matching ODE sets in the left column.

One expects better fits by adding variable dimensions to increase n or adding parameter dimensions in NODE models. The benefit here is to see exactly how the fits change with the different changes in the models. For example, the phase plots in the next subsection show clearly how the constraints of the relatively low-dimension parameter space of the ODE models limits the fit when compared with the flexibility of the larger parameter space in the NODE models.

3.2 | Phase plots

Figure 2 shows temporal trajectories for each species and each dummy variable, plotting abundance versus time. By contrast, phase plots draw trajectories by mapping the n variables at each time to a point in n-dimensional space. Combining the n-dimensional points at different times traces a trajectory through phase space.

Figure 3 shows phase plots for ODE and NODE models with $n = 3$ variables. The upper plots limit the trajectories to the $n = 2$ dimensions for the hare and lynx data (blue) and model predictions (gold). In two dimensions, the trajectories do not match well. However, one can see that the ODE model in the upper left traces a regular cycle confined to a small part of the two-dimensional phase space, whereas the NODE model in the upper right moves widely over the space.

The three-dimensional phase plots in the lower panels of Figure 3 clarify the differences between the ODE and NODE models. For those three-dimensional plots, I used the third dimension from the models' dummy variable to augment both the data and the model prediction trajectories. For the ODE model in the lower left panel, the model's phase trajectory remains confined to a limited part of a two-dimensional plane, whereas the data wander over the third dimension.

Adding the third dummy variable for the NODE model in the lower right panel greatly enhances the match between the model predictions and the data. One can see the trajectory of that third variable in Figure 2, right column, middle set for NODE and $n = 3$, in the bottom panel of that set. In that case, the dummy variable starts near an abundance of 1 (e^0) and declines toward 0 (e^{-300}).

By spreading the two-dimensional hare and lynx dynamics over a third dimension that declines steadily with time, the messy and visually mismatched data and model trajectories in the two-dimensional phase plot shown in the upper right panel of Figure 3 are transformed into smoothly oscillating and matching three-dimensional trajectories in the lower right panel of that figure.

It could be that the high-dimensional and flexible parameter space of the $n = 3$ NODE model has discovered the simple geometry of the phase dynamics. One could of course find that geometry in other ways. The main advantage of the NODE model is that it does the fit quickly and automatically without any explicit assumptions about the shape of the dynamics.

3.3 | Predicting future observations

Which models do best at predicting future outcomes? Given a single time series, one typically addresses that question by splitting the data. The first training subset provides data to fit the models. The second test subset measures the quality of the predictions.

I trained the six models in Figure 2 on the data from the first 61 yearly observations (0–60). I then compared the predictions of those models to the observed data for the subsequent 30 years (60–90). Figure 4 shows the results.

I obtained predicted values for a model by calculating the model's temporal trajectory for a particular set of fitted parameters. The predictions are the temporal trajectory over the test period, the years 60–90. A single trajectory represents the predictions for one set of fitted parameters.

When making predictions, one wants an estimate of the predicted values and also a measure of confidence in the predictions. How much variability is there in the trajectories when using alternative sets of fitted parameters?

To obtain a distribution of fitted parameter sets, I used the pSGLD method described earlier. That method provides a Bayesian-motivated notion of the posterior parameter distribution. To draw the predicted gold trajectories in Figure 4, I estimated the posterior parameter distribution and then randomly sampled 30 parameter sets from that distribution.
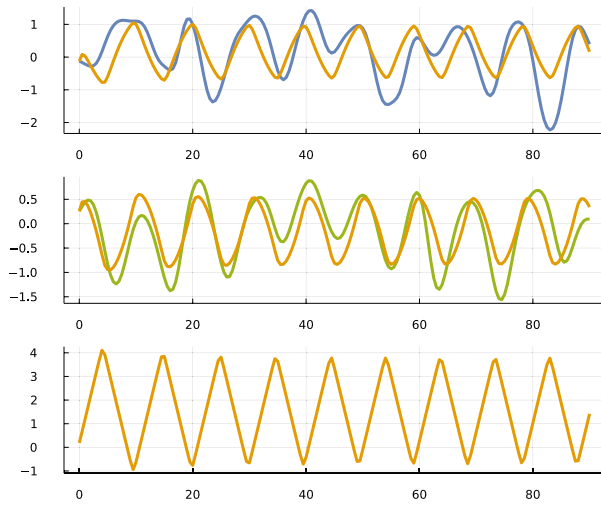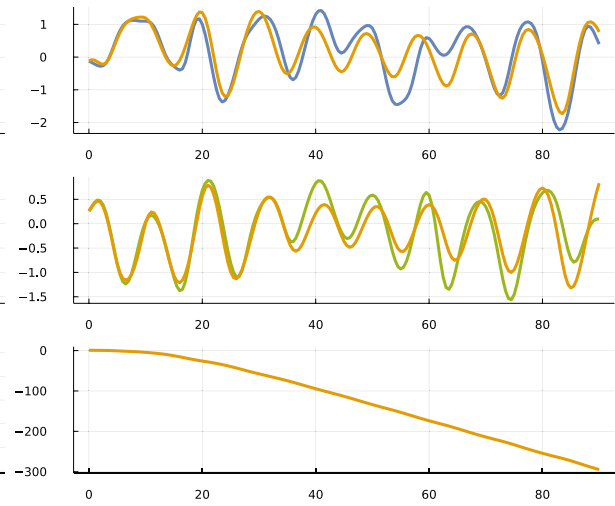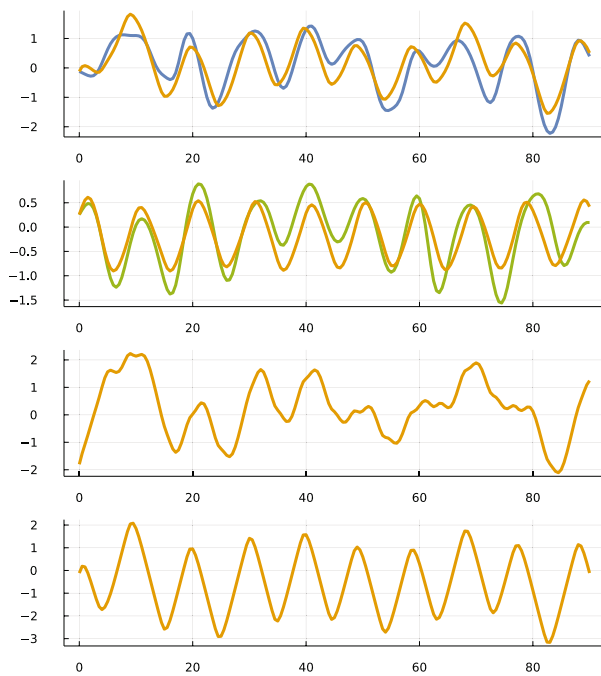
ODE *n* = 2; loss = 156.84
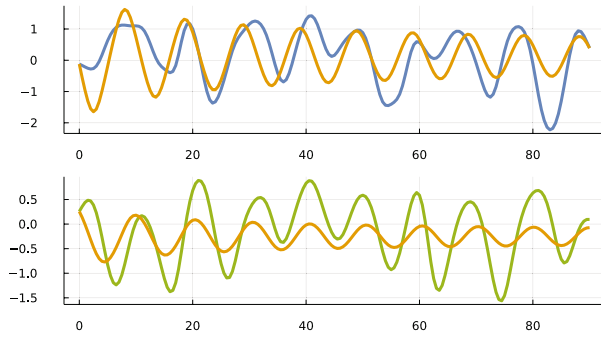
NODE *n* = 2; loss = 122.38
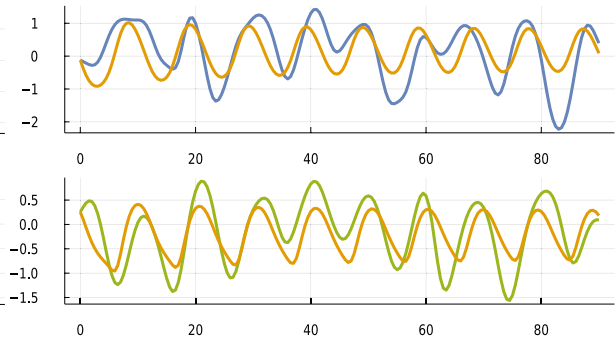
ODE *n* = 3; loss = 107.83

NODE *n* = 3; loss = 55.09

ODE *n* = 4; loss = 71.18
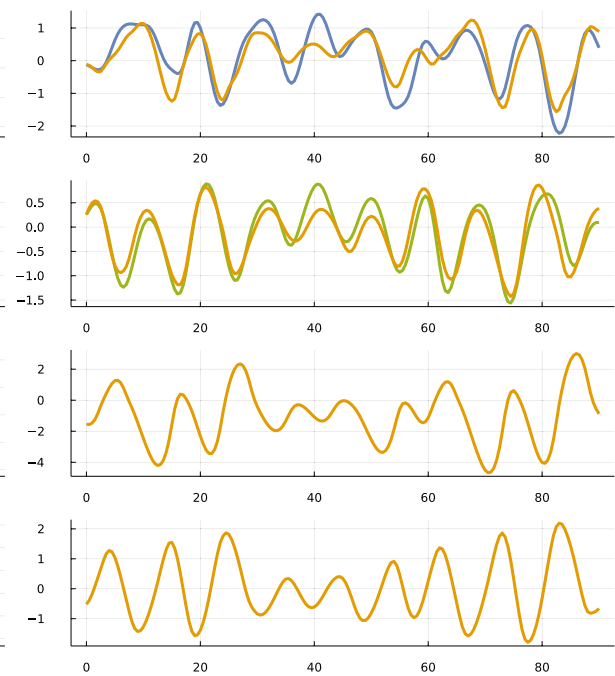
NODE *n* = 4; loss = 47.02

–WILEY

FIGURE 2 Fit of models to the smoothed data. Gold curves show predicted trajectories. In each set, the top plot shows hare data (blue), and the second plot shows lynx data (green). Subsequent plots show dummy variables when $n > 2$. The loss sums the squared deviations between the smoothed data and the model trajectories. I measured the deviations for both species at the 181 half-yearly intervals, yielding 362 squared-deviation components in the loss calculations.
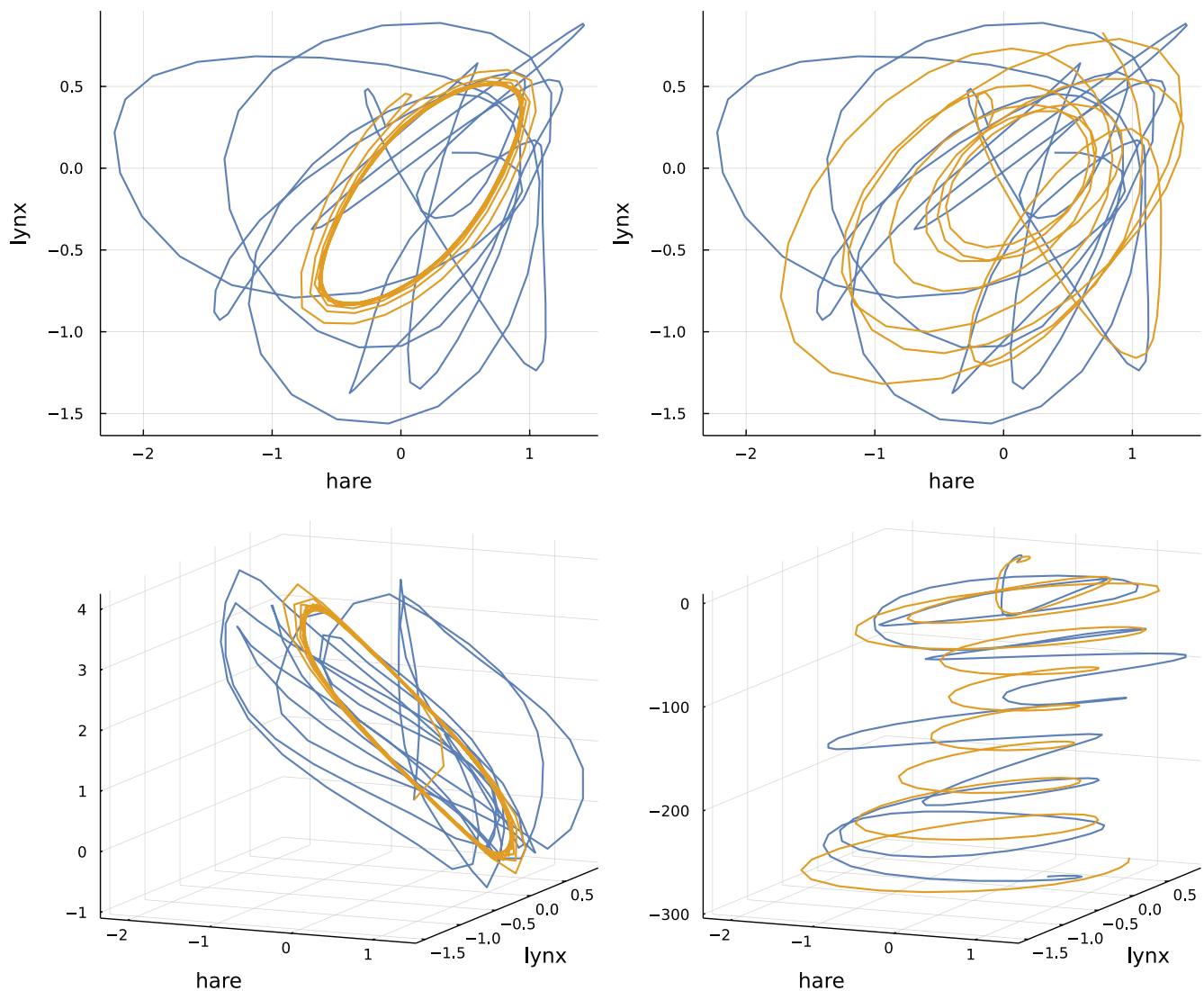

FIGURE 3 Phase plots for ODE (left) and NODE (right) models with $n = 3$. Blue curves trace the data trajectories, and gold curves trace the models' predicted trajectories. The upper plots show only the hare and lynx variables for the data and model, tracing $n = 2$ dimensional trajectories. The lower plots add the third dummy dimension variable from the model to both the data and model trajectories.

How do the different models in Figure 4 compare with regard to the quality of their predictions during the test period, 60–90? Starting at the top left, ODE $n = 2$, the model predictions are precise but inaccurate. The small variation in trajectories during the test period reflects the high precision, whereas the large differences between the predictions and the data with regard to the timing of the oscillations reflect the low accuracy.

The low accuracy (high loss) during the training period and poor fit during the test period suggest that this model is underfit. Here, *underfit* roughly means that the dimensionality of the variables or of the parameters is not sufficient to fit the data.

Next, consider the lower left model in that figure, ODE $n = 4$. That model has high accuracy during the training period, but very low precision during the prediction period. The model seems to be overfit.

During the prediction period, the NODE models for $n = 3, 4$ also have relatively low precision and varying but typically not very good accuracy. Those models may also be overfit, for which *overfit* means roughly that the models' high dimensionality caused such a close fit to the fluctuations in the training data that the models failed to capture the general trend in the data sufficiently to predict the outcome in the test period.
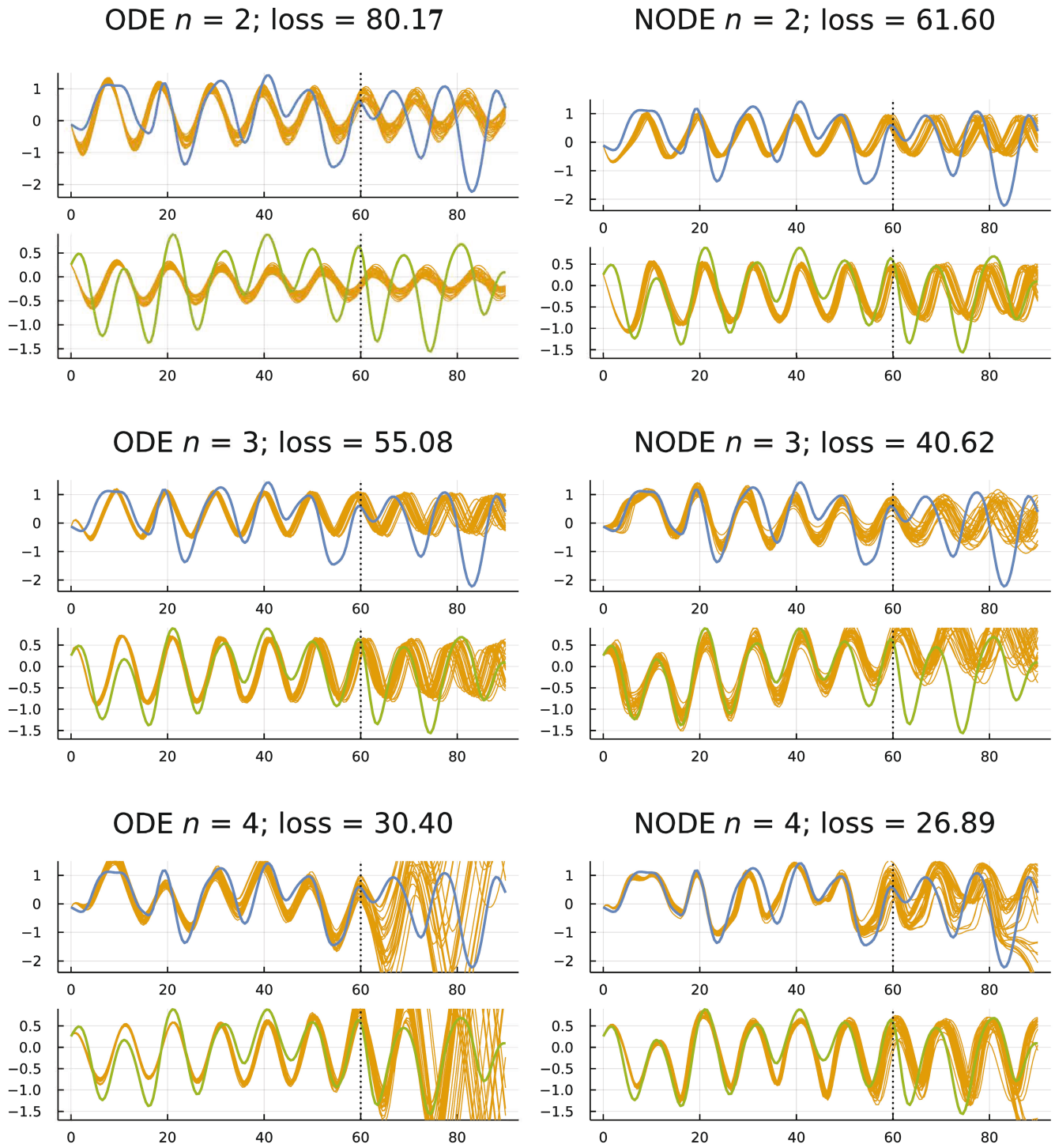
FIGURE 4 Fit of models to the first 61 yearly observations (0–60) of the 91 yearly observations (0–90). The vertical dotted line shows the end of the fitted training period. The subsequent 30 years from 60 to 90 comprise the predictions of the model relative to data not used in fitting. The layout of the plots and colors for the various curves match Figure 2, with blue for smoothed hare data, green for smoothed lynx data, and gold for the predicted trajectories of the models. For the predicted trajectories, I randomly chose 30 parameter combinations from the posterior distribution of parameters obtained by pSGLD sampling. The loss values are calculated for the training period, 0–60.

Finally, consider the two best models with regard to predictions during the test period, ODE $n = 3$ and NODE $n = 2$. Those models have intermediate accuracy during the fitted period, which seemed to avoid underfitting and overfitting. During the test period, both models had moderately good accuracy with regard to the timing of oscillations and

moderately good precision with regard to variation in the predicted trajectories. Although the fits are far from perfect, they are good given the short training period in relation to the complex shape of the dynamics.

To support these conclusions for the quality of the predicted fits during the test period, I calculated loss values for a sample of 3000

trajectories for each of the six plots in Figure 4. The four best fits with the lowest loss values, given as mean±standard deviation, are NODE2 = 43 ± 6, ODE3 = 78 ± 22, ODE2 = 96 ± 15, NODE3 = 180 ± 39. Repeating the procedure yielded similar values for all except ODE3. In 49 of 80 separate samples of 3000 trajectories, the mean ODE3 values were less than the ODE2 mean of 96. Overall, ODE3 appears to be better on average than ODE2 but with a heavier tail of larger values.

A technical challenge arises when deciding how long to run the sampling period for pSGLD and what hyperparameters to use to control that process. When does one have a sufficient estimate for the posterior distribution of trajectories? In a typical run for this study, I first ran the pSGLD sampling for a warm-up period that created 5000 parameter sets and associated trajectories. I collected 10,000 or more parameter sets and associated trajectories by pSGLD. For each trajectory, I calculated the loss for the full time over both the training and test periods.

I concluded that the sample was sufficient when the distribution of loss values for the first half of the 10,000 generated parameter sets was reasonably close to the distribution of loss values for the second half of the generated parameter sets. As long as the loss distributions were not broadly different, the plotted trajectory distributions typically did not look very different.

One could also study posterior distributions for individual parameters. However, in this study, there was no reason to analyze individual parameters.

4 | DISCUSSION

The various technical advances greatly enhance the ease of fitting alternative differential equation models. New possibilities arise to analyze dynamics, gain insight into process, improve predictions, and enhance control.

In this article, I focused on fitting observed dynamics from a natural system. Alternatively, one could study how to design a system to achieve desired dynamics or to match a theoretical target pattern (Hiscock, 2019).

The pSGLD method to sample parameter combinations near a local optimum also raises interesting possibilities for future study. Technically, it is a remarkably simple and computationally fast method. Conceptually, it creates a kind of random walk near a local optimum on a performance surface, similar to mutation-selection dynamics near a local optimum of a fitness landscape (Neher & Shraiman, 2011). That analogy suggests the potential to gain further understanding of genetic variation and evolutionary dynamics on complex fitness surfaces.

The reviewers asked about what insights into ecology might arise from this study. For example, how do the growth rate and competition parameters compare between the ODE model with $n = 3$ and the NODE model with $n = 2$? What additional information can we learn from the best fitted ODEs beyond the classic Lotka-Volterra model? What ecological meaning could the additional dummy variables have? What are the application prospects and the limitations in applying this kind of approach?

I agree that these are essential questions to answer over the long term. Otherwise, the methods do not advance our understanding of ecological data and ultimately provide little value. However, my intent for this study was primarily methodological. If we are eventually going to improve our ability to deal with such questions, we must initially solve the methodological challenges. In my view, the commonly analyzed hare and lynx data provide a nice test case for exploring new methods but do not provide nearly enough information to say anything significant about ecology. I hesitate to overinterpret. I do believe that as our methods improve, we will be able to apply such methods to broader data sets and to design analyses that will improve our understanding of ecology. But that improvement is a long-term goal. This is an early step toward that goal.

AUTHOR CONTRIBUTIONS

Steven A. Frank: Conceptualization (lead); data curation (lead); formal analysis (lead); funding acquisition (lead); investigation (lead); methodology (lead); project administration (lead); resources (lead); software (lead); supervision (lead); validation (lead); visualization (lead); writing – original draft (lead); writing – review and editing (lead).

CONFLICT OF INTEREST STATEMENT

The author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

DATA AVAILABILITY STATEMENT

All data and code are available at https://github.com/evolbio/FitODE and on Zenodo at https://doi.org/10.5281/zenodo.6463624. The parameters and output used to generate the figures in this article are only available at Zenodo.

ORCID

Steven A. Frank https://orcid.org/0000-0001-7348-7794

REFERENCES

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M. (2018). Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*, *18*, 1–43.

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*, 65–98.

Bonnaffé, W., & Coulson, T. (2022). *Fast fitting of neural ordinary differential equations by Bayesian neural gradient matching to infer ecological interactions from time series data*. arXiv:2209.06184.

Bonnaffé, W., Sheldon, B. C., & Coulson, T. (2021). Neural ordinary differential equations for ecological and evolutionary time-series analysis. *Methods in Ecology and Evolution*, *12*, 1301–1315.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2018). Neural ordinary differential equations. *arXiv:1806.07366*.

Dupont, E., Doucet, A., & Teh, Y. W. (2019). Augmented neural ODEs. *Advances in Neural Information Processing Systems, 32*.

Frank, S. A. (2022a). Automatic differentiation and the optimization of differential equation models in biology. *Frontiers in Ecology and Evolution, 10*, 1010278.

Frank, S. A. (2022b). Optimizing differential equations to fit data and predict outcomes. *arXiv:2204.07833*.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Hiscock, T. W. (2019). Adapting machine-learning algorithms to design gene circuits. *BMC Bioinformatics, 20*, 214.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv:1412.6980*.

Li, C., Chen, C., Carlson, D., & Carin, L. (2015). Preconditioned stochastic gradient Langevin dynamics for deep neural networks. *arXiv:1512.07666*.

Margossian, C. C. (2019). A review of automatic differentiation and its efficient implementation. *WIREs Data Mining and Knowledge Discovery, 9*, e1305.

Neher, R. A., & Shraiman, B. I. (2011). Statistical genetics and evolution of quantitative traits. *arXiv:1108.1630*.

Odum, E. P., & Barrett, G. W. (1971). *Fundamentals of ecology* (3rd ed.). W. B. Saunders.

Perkel, J. M. (2019). Julia: Come for the syntax, stay for the speed. *Nature, 572*, 141–142.

Rackauckas, C., Innes, M., Ma, Y., Bettencourt, J., White, L., & Dixit, V. (2019). DiffEqFlux.jl—A Julia library for neural differential equations. *arXiv:1902.02376*.

Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., Skinner, D., & Ramadhan, A. (2020). Universal differential equations for scientific machine learning. *arXiv:2001.04385*.

Rackauckas, C., & Nie, Q. (2017). DifferentialEquations.jl—A performant and feature-rich ecosystem for solving differential equations in Julia. *Journal of Open Research Software, 5*, 15.

Sharma, S., Sharma, S., & Athaiya, A. (2020). Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology, 4*, 310–316.