A Additional figures

In this section, we present some of the figures that were excluded from the main text for brevity.

Figure 6 below highlights the efficiency of our training approach with respect to the number of training epochs by plotting the trajectory predictions of the second-order models during training.

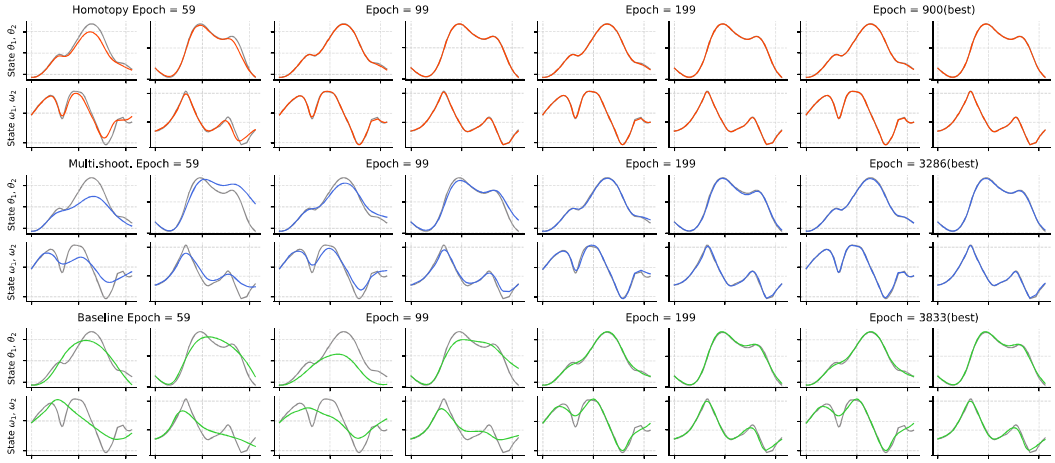

Figure 6: Second-order model predictions for the double pendulum data during training. Blue and orange lines indicate data and model predictions respectively. **(Top)** Results from our homotopy approach. **(Middle)** Results from multiple shooting. **(Bottom)** Results from vanilla gradient descent.

Figures 7 to 9 show model prediction trajectories corresponding to our benchmark results of Figure 4. One intriguing point is that even though both the gray-box model (Lotka-Volterra system, Figure 7) and the second-order model (double pendulum, Figure 8) contain more prior information about their respective systems, this does not necessarily translate to better predictive capabilities if an effective training method is not used. This suggests that introducing physics into the learning problem can obfuscate optimization, something that has been reported for physics-informed neural networks. It also highlights the effectiveness of our homotopy training algorithm, as our method can properly train such difficult-to-optimize models.

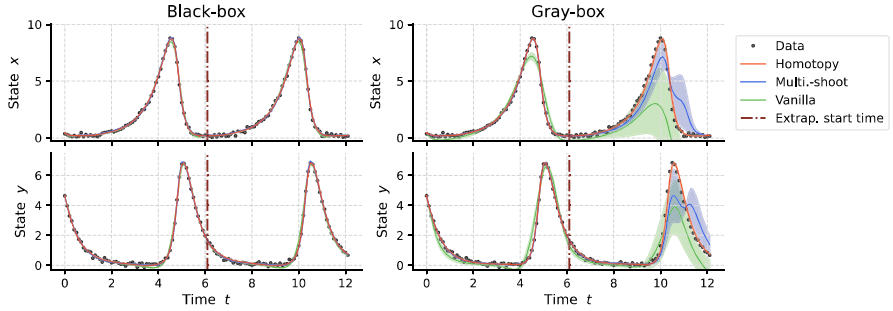

Figure 7: Trajectory predictions for the Lotka-Volterra system from models trained with different methods. Dashed vertical line indicates the start of extrapolation.

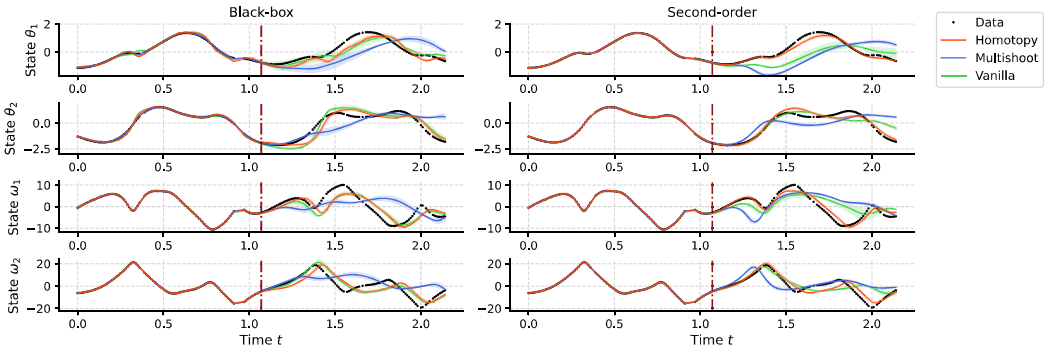

Figure 8: Trajectory predictions for the double pendulum from models trained with different methods. Dashed vertical line indicates the start of extrapolation.

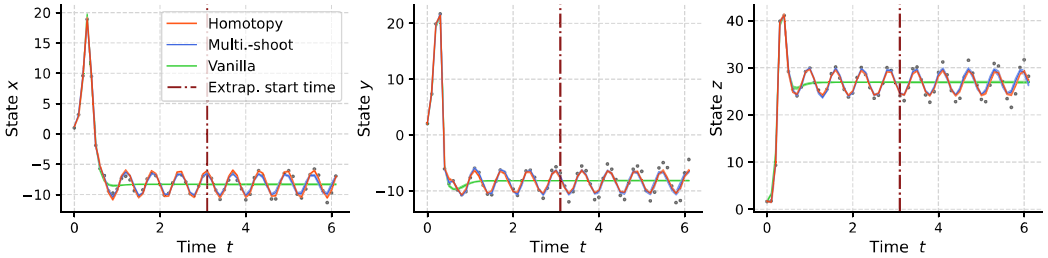

Figure 9: Trajectory predictions for the Lorenz system from models trained with different methods. Dashed vertical line indicates the start of extrapolation.

B Loss landscape and synchronization in ODE parameter estimation

Here, we illustrate the similarity between the ODE parameter estimation problem and the NeuralODE training problem by repeating analyses analogous to those of Section 4 in the setting of estimating the unknown coefficients of a given ordinary differential equation.

In the left and middle panels of Figure 10, we show the trajectories of the Lotka-Volterra and the Lorenz systems were a single parameter was perturbed. We can easily observe that the original trajectory and the perturbed trajectories evolve independently in time. Furthermore, while trajectories from the simpler Lotka-Volterra system retain the same periodic behavior and differ only in their amplitudes and phases, trajectories from the Lorenz system display much different characteristics with increasing time. This translates over to the shape of the loss landscape (Figure 10, right) with the loss for the Lotka-Volterra system only becoming steeper with increasing data length, whereas the loss for the Lorenz system displays more and more local minima.

Once synchronization is introduced to the perturbed trajectory, its dynamics tends to converge with the reference trajectory as time increases, with the rate of convergence increasing for larger values of the coupling strength k (Figure 11, left and middle panels). In terms of the loss landscape (right panel), larger coupling does result in a smoother landscape, with excessive amount of it resulting in a flat landscape that is also detrimental to effective learning. These results are a direct parallel to our observations in Figure 2 and provide additional justifications as to why techniques developed in the field of ODE parameter estimation also work so well on NeuralODE training.

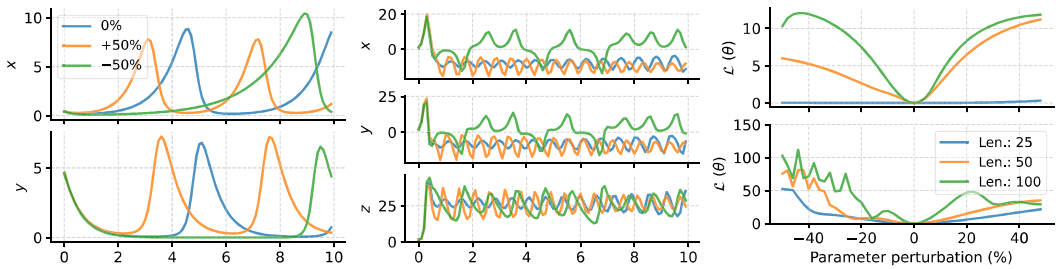

Figure 10: Dynamics of single parameter perturbed systems. The systems and the parameters used are described in Section 6. **(Left)** Solutions for the periodic Lotka-Volterra equations with perturbed α parameter. **(Middle)** Solutions for the chaotic Lorenz system with perturbed β parameter. **(Right)** MSE loss function landscape for the Lotka-Volterra equations **(right, upper)** and Lorenz system **(right, lower)** for different lengths of time series data for the loss calculation.

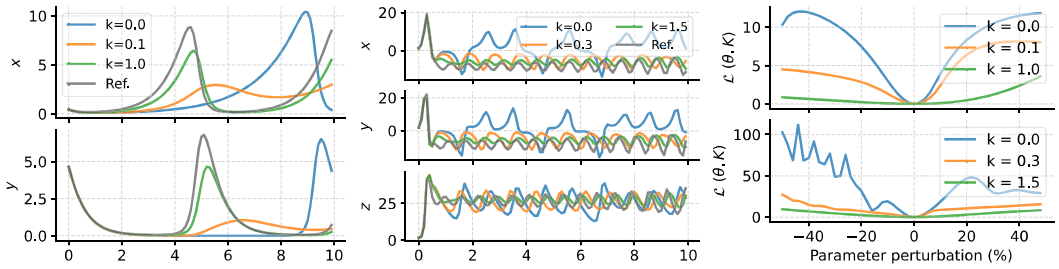

Figure 11: Dynamics of coupled systems with varying coupling strengths. The systems and the parameters used are identical to those of Figure 10. **(Left)** Results for the periodic Lotka-Volterra equations. **(Middle)** Results for the chaotic Lorenz system. **(Right)** Loss function landscape of the coupled systems with different coupling strengths.

C Further description of the homotopy optimization procedure

To implement homotopy optimization, one usually selects the number of discrete steps for optimization, s, as well as a series of positive decrement values for the homotopy parameter $\{\Delta\lambda^{(k)}\}_0^{s-1}$ that sum to 1. Afterwards, optimization starts with an initial λ value of $\lambda^{(0)} = 1$, which gives $\mathcal{H}^{(0)}(\boldsymbol{\theta}) = \mathcal{G}(\boldsymbol{\theta})$. At each step, the objective function at the current iteration is minimized with respect to $\boldsymbol{\theta}$, using the output from the previous step $\boldsymbol{\theta}^{*(k-1)}$ is as the initial guess:

$$\mathcal{H}^{(k)}(\boldsymbol{\theta}) = \mathcal{H}(\boldsymbol{\theta}, \lambda = \lambda^{(k)}) \;\rightarrow\; \boldsymbol{\theta}^{*(k)} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathcal{H}^{(k)}(\boldsymbol{\theta}) \tag{8}$$

Afterwards, λ decremented to its next value, $\lambda^{(k+1)} = \lambda^{(k)} - \Delta\lambda^{(k)}$, and this iteration continues until the final step s where $\lambda^{(s)} = 0$, $\mathcal{H}^{(s)}(\boldsymbol{\theta}) = \mathcal{F}(\boldsymbol{\theta})$, and the final minimizer $\boldsymbol{\theta}^{*(s)} = \boldsymbol{\theta}^*$ is the sought-after solution to the original problem $\mathcal{F}(\boldsymbol{\theta})$.

D Multiple shooting algorithm

Our implementation of the multiple shooting algorithm for training NeuralODEs closely mirrors the example code provided in the `DiffEqFlux.jl` package [39] of the Julia programming language ecosystem.

Given time series data $\{t_i, \hat{\mathbf{u}}_i\}_{i=0}^N$, multiple shooting method partitions the data time points into m overlapping segments:

$$[t_0 = t_0^{(0)}, \ldots, t_n^{(0)}], ..., [t_0^{(m-1)}, \ldots, t_n^{(m-1)} = t_N];\; t_n^{(i-1)} = t_0^{(i)},\; i \in 1 \ldots m-1.$$

During training, the NeuralODE is solved for each time segment, resulting in m segmented trajectories:

$$[\mathbf{u}_0^{(0)}, \ldots, \mathbf{u}_n^{(0)}], ..., [\mathbf{u}_0^{(m-1)}, \ldots, \mathbf{u}_n^{(m-1)}].$$

Taking into account the overlapping time point between segments, these trajectories can then all be concatenated to produce the trajectory prediction:

$$[\mathbf{u}_0^{(0)}, \ldots, \mathbf{u}_{n-1}^{(0)}, \mathbf{u}_0^{(1)}, \ldots, \mathbf{u}_n^{(m-1)}] = [\mathbf{u}_0, \ldots \mathbf{u}_n].$$

From this, the data loss is defined identically to conventional NeuralODE training: $\mathcal{L}_{data}(\boldsymbol{\theta}) = \frac{1}{N+1} \sum_i |\mathbf{u}_i(\boldsymbol{\theta}) - \hat{\mathbf{u}}_i|^2$.

However, due to the segmented manner in which the above trajectory is generated, optimizing only over this quantity will result in a model that could generate good piecewise predictions, but be unable to generate a proper, continuous global trajectory. Therefore, to ensure the model can generate smooth trajectories, continuity constraints $\mathbf{u}_n^{(i-1)} = \mathbf{u}_0^{(i)}$, $(i \in 1 \ldots m-1)$ must be enforced. How this is achieved is the point of difference in the literature, and our implementation - following that of [39] - introduces a regularization term in the loss:

$$\mathcal{L}_{continuity}(\boldsymbol{\theta}) = \frac{1}{m} \sum_i \left| \mathbf{u}_n^{(i-1)} - \mathbf{u}_0^{(i)} \right|^2.$$

Finally, the train loss for the multiple shooting method is then defined as

$$\mathcal{L}(\boldsymbol{\theta}, \beta) = \mathcal{L}_{data}(\boldsymbol{\theta}) + \beta \cdot \mathcal{L}_{continuity}(\boldsymbol{\theta})$$

and is minimized using gradient descent, where β is a hyperparameter that tunes the relative importance of the two terms during training.

E Experiment details

In all experiments, the AdamW optimizer [23] was used to minimize the respective loss functions for the vanilla and homotopy training. Each experiment was repeated using random seed values of 10, 20, and 30 to compute the prediction means and standard errors.

E.1 Lotka-Volterra system

The Lotka-Volterra system is a simplified model of predator-prey dynamics given by,

$$\frac{dx}{dt} = \alpha x - \beta xy, \quad \frac{dy}{dt} = -\gamma y + \delta xy \tag{9}$$

where the parameters $\alpha, \beta, \gamma, \delta$ characterize interactions between the populations.

Data preparation. Following the experimental design of Rackauckas [40], we numerically integrate Equation (9) in the time interval $t \in [0, 6.1]$, $\Delta t = 0.1$ with the parameter values $\alpha, \beta, \gamma, \delta = 1.3, 0.9, 0.8, 1.8$ and initial conditions $x(0), y(0) = 0.44249296, 4.6280594$. Continuing with the recipe, Gaussian random noise with zero mean and standard deviations with magnitude of 5% of the mean of each trajectory was added to both states. For both data generation and NeuralODE prediction, integration was performed using the adaptive step size `dopri5` solver from the `torchdiffeq` package [7] with an absolute tolerance of 1e-9 and a relative tolerance of 1e-7.

For the experiments of Figure 3, training data were generated in a similar manner, but with specific parameters varied to match the corresponding experiments. Data varying train data length (Figure 3, left panel) were generated using time spans of $t \in [0, 3.1], [0, 6.1], [0.9.1]$ respectively with shared parameter values of $\Delta t = 0.1$ and noise amplitude of 5% of the mean. Data with differing sampling periods (Figure 3, third panel) were generated with a fixed time span of $t \in [0, 6.1]$, noise amplitude of 5% of the mean and sampling periods of $\Delta t = 0.1, 0.3, 0.5$. Data with different noise amplitudes (Figure 3, fourth panel) were generated using a time span of $t \in [0, 6.1]$, sampling period of $\Delta t = 0.1$ and noise amplitudes of 5%, 10%, 20%, 50% of the mean.

Model architecture. We used two types of models for this dataset, a black-box NeuralODE of Equation (1), and a gray-box NeuralODE used in Rackauckas et al. [39] that incorporates partial information about the underlying equation, given by:

$$\frac{dx}{dt} = \alpha x + U_1(x, y; \boldsymbol{\theta}_1), \ \frac{dy}{dt} = -\gamma y + U_2(x, y; \boldsymbol{\theta}_2). \tag{10}$$

In our default setting, the black-box NeuralODE had 3 layers with [2, 32, 2] nodes respectively. The gray-box NeuralODE had 4 layers with [2, 20, 20, 2] nodes, with each node in the output layer corresponding to the output of U_1, U_2 of Equation (10). Following the results from [21], a non-saturating gelu activation was used for all layers except for the final layer, where identity activation was used. For the model capacity experiment (Figure 3, second panel), the number of nodes in the hidden layer was changed accordingly.

E.2 Double pendulum

The double pendulum system is a canonical example in classical mechanics, and has four degrees of freedom $\theta_1, \theta_2, \omega_1, \omega_2$ corresponding to the angles and angular velocities of the two pendulums with respect to the vertical. The governing equation for the system can be derived using Lagrangian formulation of classical mechanics and is given by:

$$\frac{d\theta_1}{dt} = \omega_1, \quad \frac{d\theta_2}{dt} = \omega_2 \tag{11}$$

$$\frac{d\omega_1}{dt} = \frac{m_2 l_1 \omega_1^2 \sin\Delta\theta \cos\Delta\theta + m_2 l_2 \omega_2^2 \sin\Delta\theta + m_2 g \sin\theta_2 \cos\Delta\theta - (m_1+m_2) g \sin\theta_1}{(m_1+m_2) l_1 - m_2 l_1 \cos^2\Delta\theta}, \tag{12}$$

$$\frac{d\omega_2}{dt} = -\frac{m_2 l_2 \omega_2^2 \sin\Delta\theta \cos\Delta\theta + (m_1+m_2)(-l_1\omega_1^2 \sin\Delta\theta + g\sin\theta_1\cos\Delta\theta - g\sin\theta_2)}{(m_1+m_2) l_2 - m_2 l_2 \cos^2\Delta\theta} \tag{13}$$

where $\Delta\theta = \theta_2 - \theta_1$, m_1, m_2 are the masses of each rod, and g is the gravitational acceleration.

Data preparation. While simulated trajectories for the double pendulum can be generated using the equations above, we instead used the experimental data from Schmidt & Lipson [44]. This consists of two trajectories from the double pendulum, captured using multiple cameras. The noise in the data is subdued due to the LOESS smoothing performed by the original authors. For our experiments, we used the first 100 points of the first trajectory for training and the next 100 to evaluate the extrapolation capabilities of the trained model.

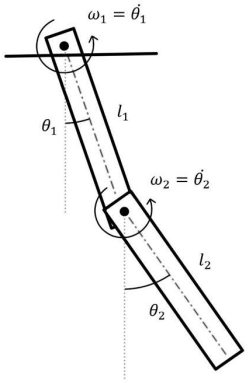

Figure 12: Diagram of a double pendulum.

Model architecture. Two types of models were used for this dataset: a black-box NeuralODE (Equation (1)) and a NeuralODE with second-order structure[16], which for this system, takes the form:

$$\frac{d\theta_i}{dt} = \omega_i, \ \frac{d\omega_i}{dt} = U_i(\theta_1, \theta_2, \omega_1, \omega_2; \boldsymbol{\theta}_i); \quad i = 1, 2.$$

The black-box NeuralODE had 4 layers with [4, 50, 50, 4] nodes, with the input and output node numbers corresponding to the degrees of freedom of the system. The second-order model also had 4 layers with [4, 50, 50, 2] nodes. Note that there are now 2 output nodes instead of 4 since incorporating second-order structure requires the neural network to only model the derivatives of ω_1 and ω_2. For both models, gelu activations were used for all layers except the last, which used identity activation. Identical to the previous dataset, the NeuralODEs were integrated using the `dopri5` solver from the `torchdiffeq` package [7] with an absolute tolerance of 1e-9 and a relative tolerance of 1e-7.

E.3 Lorenz system

The Lorenz system is given by the equations

$$\frac{dx}{dt} = \sigma(y - x), \quad \frac{dy}{dt} = x(\rho - z) - y, \quad \frac{dz}{dt} = xy - \beta z. \tag{14}$$

Data preparation. To generate the data, we followed experimental settings of Vyasarayani et al. [51] and used parameter values $\sigma, \rho, \beta = 10, 28, 8/3$ and the initial condition $x_0, y_0, z_0 = 1.2, 2.1, 1.7$. These conditions, upon integration, gives rise to the well-known "butterfly attractor". The training data was generated in the interval $t \in [0, 3.1]$, and still adhering to the paper, a Gaussian noise of mean 0 and standard deviation 0.25 was added to the simulated trajectories to emulate experimental noise. For data generation and NeuralODE prediction, the adaptive step size `dopri5` solver from the `torchdiffeq` package [7] was used with an absolute tolerance of 1e-9 and a relative tolerance of 1e-7.ODE solver and the tolerance values were kept identical to the previous Lotka-Volterra experiment.

Model architecture. We used a single NeuralODE for this dataset - a black-box model having 4 layers with [3, 50, 50, 3] nodes. The number of nodes for the input and output layers correspond to the three degrees of freedom of the state vector $\mathbf{u} = [x, y, z]^T$ of the system. The activations for the model was kept identical to the previous experiments - gelu activation for all layers, except for the last layer which used identity activation.

F Hyperparameter selection

F.1 Overview of the hyperparameters

Our final implementation of the homotopy training algorithm has five hyperparameters. Here, we briefly describe the effects and the tips for tuning each.

- **Number of homotopy steps** (s) : This determines how many relaxed problem the optimization process will pass through to get to the final solution. Similar to scheduling the temperature in simulated annealing, fewer steps results in the model becoming stuck in a sharp local minima, and too many steps makes the optimization process unnecessarily long. We find using values in the range of 6-8 or slightly larger values for more complex systems yields satisfactory results.
- **Epochs per homotopy steps** (n_{epoch}) : This determines how long the model will train on a given homotopy parameter value λ. Too small, and the model lacks the time to properly converge on the loss function; too large, and the model overfits on the simpler loss landscape of $\lambda \neq 0$, resulting in a reduced final performance when $\lambda = 0$. We find for simpler monotonic or periodic systems, values of 100-150 work well; for more irregular systems, 200-300 are suitable.
- **Coupling strength** (k) : This determines how trivial the auxillary function for the homotopy optimization will be. Too small, and even the auxillary function will have a jagged landscape; too large, and the initial auxillary function will become flat (Figures 2 and 11, left panel, $k = 1.0, 1.5$) resulting in very slow parameter updates. We find good choices of k tend to be comparable to the scale of the measurement values.
- **Homotopy parameter decrement ratio** (κ) : This determines how the homotopy parameter λ is decremented for each step. Values close to 1 cause λ to decrease in nearly equal decrements, whereas smaller values cause a large decrease of λ in the earlier parts of the training, followed by subtler decrements later on. We empirically find that κ values near 0.6 tends to work well.
- **Learning rate** (η) : This is as same as in conventional NeuralODE training. We found values in the range of 0.002-0.1 to be adequate for our experiments.

F.2 Details on hyperparameter sweep & selected values

For all combinations of models and datasets used, hyperparameters for the optimization were chosen by running sweeps prior to the experiment with a fixed random seed of 10, then selecting hyperparameter values that resulted in the lowest mean squared error value during training.

Note that for the experiment of Figure 3, the same hyperparameters used in Figure 4 was used for all experiments, due to the sheer cost of sweeping for the hyperparameters for every change of independent variables.

Vanilla gradient descent. Vanilla gradient descent has learning rate as its only hyperparameter. Due to finite computation budget, we set the maximum training epochs to 4000 for all sweeps and experiments. We list the values used in the hyperparameter sweep as well as the final selected values for each dataset in Table 1. We found that larger learning rates lead to numerical underflow in the adaptive ODE solver while training on the more difficult Lorenz and double datasets; hence, the sweep values for the learning rate parameters were taken to be lower than those for the Lotka-Volterra dataset.

Dataset	Model	Hyperparameter	Sweep values	Selected value
Lotka-Volterra	Black-box	Learning rate	0.005, 0.01, 0.02, 0.05, 0.1	0.05
	Gray-box			0.02
Double pendulum	Black-box	Learning rate	0.002, 0.005, 0.01, 0.02	0.02
	Second-order			0.05
Lorenz	Black-box	Learning rate	0.002, 0.005, 0.01, 0.02	0.005

Table 1: Hyperparameter sweep and selected values for vanilla gradient descent

Multiple shooting. The multiple shooting method has three hyperparameters: learning rate, number of segments, and continuity penalty. For the majority of experiments, we set number of segments to 5, and performed hyperparameter sweep on the remaining two parameters.

Dataset	Model	Hyperparameter	Sweep values	Selected value
Lotka-Volterra	Black-box	Learning rate	0.005, 0.01, 0.02, 0.05	0.05
		Continuity penality	0.005, 0.002, 0.01	0.01
	Gray-box	Learning rate	0.005, 0.01, 0.02, 0.05	0.05
		Continuity penalty	0.005, 0.002, 0.01	0.002
Double pendulum	Black-box	Learning rate	0.005, 0.01, 0.02, 0.05	0.02
		Continuity penalty	0.005, 0.002, 0.01	0.002
	Second-order	Learning rate	0.005, 0.01, 0.02, 0.05	0.02
		Continuity penalty	0.005, 0.002, 0.01	0.002
Lorenz	Black-box	Learning rate	0.005, 0.01, 0.02, 0.05	0.02
		Continuity penalty	0.005, 0.002, 0.01	0.01

Table 2: Hyperparameter sweep and selected values for multiple shooting

Homotopy optimization. Our homotopy-based NeuralODE training has five hyperparameters: learning rate (η), coupling strength (k), number of homotopy steps (s), epochs per homotopy steps (n_{epoch}), and homotopy parameter decrement ratio (κ).

While sweeping over all five hyperparameters would yield the best possible results, searching such a high dimensional space can pose a computation burden. In practice, we found that sweeping over only the first two hyperparameters and fixing the rest at predecided values still returned models that functioned much better than their vanilla counterparts. We list the predecided hyperparameter values for each dataset in Table 3. Note that we increased the epochs per homotopy step from 100 the Lotka-Volterra model to 300 for the Lorenz system and double pendulum datasets to account for the increased difficulty of the problem.

Table 4 shows the sweeped hyperparameters as well as the final chosen values for each dataset. The total number of training epochs is given by multiplying the number of homotopy steps with epochs per homotopy steps. This resulted in 600 epochs for the Lotka-Volterra dataset, and 1800 epochs for both the Lorenz system and the double pendulum datasets.

Dataset	Model	Hyperparameter	Predecided value
Lotka-Volterra	Black-box Gray-box	Number of homotopy steps	6
		Epochs per step	100
		λ decay ratio	0.6
Double pendulum	Black-box	Number of homotopy steps	7
		Epochs per step	300
		λ decay ratio	0.6
	Second-order	Number of homotopy steps	6
		Epochs per step	300
		λ decay ratio	0.6
Lorenz	Black-box	Number of homotopy steps	8
		Epochs per step	300
		λ decay ratio	0.6

Table 3: Predecided hyperparameter values for homotopy optimization

Dataset	Model	Hyperparameter	Sweep values	Selected value
Lotka-Volterra	Black-box	Learning rate	0.01, 0.02, 0.05	0.05
		Control strength	2, 3, 4, 5, 6	6
	Gray-box	Learning rate	0.01, 0.02, 0.05	0.02
		Control strength	2, 3, 4, 5, 6	4
Double pendulum	Black-box	Learning rate	0.01, 0.02, 0.05	0.05
		Control strength	3, 4, 5, 6, 7, 8, 9, 10	10
	Second-order	Learning rate	0.01, 0.02, 0.05	0.02
		Control strength	3, 4, 5, 6, 7, 8, 9, 10	10
Lorenz	Black-box	Learning rate	0.005, 0.01, 0.02	0.02
		Control strength	6, 7, 8, 9, 10	6

Table 4: Hyperparameter sweep and selected values for homotopy optimization

G Further results

G.1 Table of the benchmark results

Here, we present the benchmark results of Figure 4 in a table format as an alternative data representation.

Table 5: Experimental results for various NeuralODE models trained with three different methods. The values for best train epochs correspond to random seed values of 10, 20, and 30, except for runs marked with * where the backup random seed was used due to unstable training in the original random seed. MSE values are reported with the mean and standard error from three different runs.

Dataset	Lotka-Volterra		Double Pendulum		Lorenz System
Model Type	Black Box	Gray Box	Black Box	Second Order	Black Box
Best train epochs					
Baseline	(3932, 3818, 3134)	(3999, 3980*, 3992)	(3998, 3731, 3633)	(2890, 3914, 3964)	(3960, 3996, 3958)
Multi-. Shoot.	(3644, 1482, 1359)	(3911, 2127, 1198)	(3438, 3454, 3646)	(3286, 2207, 3831)	(3710, 3666*, 3976)
Homotopy	**(299, 208, 227)**	**(291, 265, 216*)**	**(1201, 2087, 1502)**	**(1200, 600, 900)**	**(2399, 2399, 2377*)**
Mean Squared Error ($\times 10^{-2}$)					
Baseline (interp.)	1.76±0.13	27.3±8.68	1.61±0.12	1.03±0.12	259±6.05
Multi-. Shoot.(interp.)	0.95±0.02	0.74±0.02	**0.81±0.23**	0.28±0.03	**12.9±1.03**
Homotopy (interp.)	**0.78±0.05**	**0.72±0.01**	0.98±0.08	**0.18±0.02**	18.6±1.75
Baseline (extrap.)	2.73±0.17	6254±5026	944.9±159.7	1619±172.9	603±1.36
Multi-. Shoot.(extrap.)	1.42±0.01	262.2±212.4	2676±391.7	2365±105.1	138±37.0
Homotopy (extrap.)	**1.22±0.09**	**7.656±4.557**	**1031±103.2**	**428.5±51.90**	**92.8±2.59**

G.2 Example of the training curves

Here, we include some of the training curves for our experiments. Figure 13 displays the averaged training curves for the black-box model trained on the Lotka-Volterra dataset. From the right panel, it can clearly be seen that our homotopy method optimizes the MSE much rapidly than the other methods, arriving at the noise floor in less than 500 epochs. The abrupt jumps in the MSE curve for homotopy is due to the discontinous change in the train loss that occurs every the the homotopy parameter is adjusted. To make this connection clearer, we show the train loss as well as the homotopy parameter for our method in the left panel of the same figure.

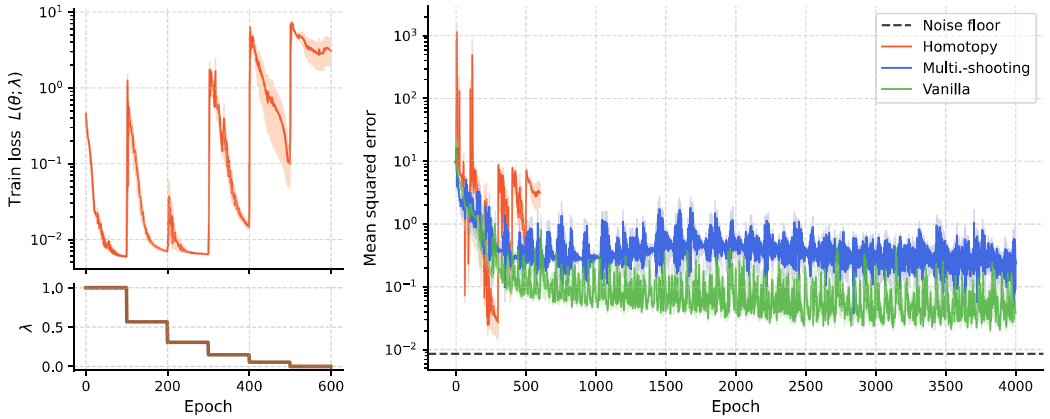

Figure 13: Training curves for the black-box model on the Lotka-Volterra dataset, corresponding to the benchmark results of Figure 4. **(Left)** Train loss $\mathcal{L}(\boldsymbol{\theta}, \lambda)$ and the homotopy parameter λ. **(Right)** Mean squared error as a function of training epochs. Note that the curves for homotopy stops early due to the choice of the algorithm hyperparameters (see Appendix F.2 for further details).

We present analogous results for the second-order model trained on the double pendulum dataset in Figure 14. Once again, we find that our homotopy method converges much quicker than its competitors.

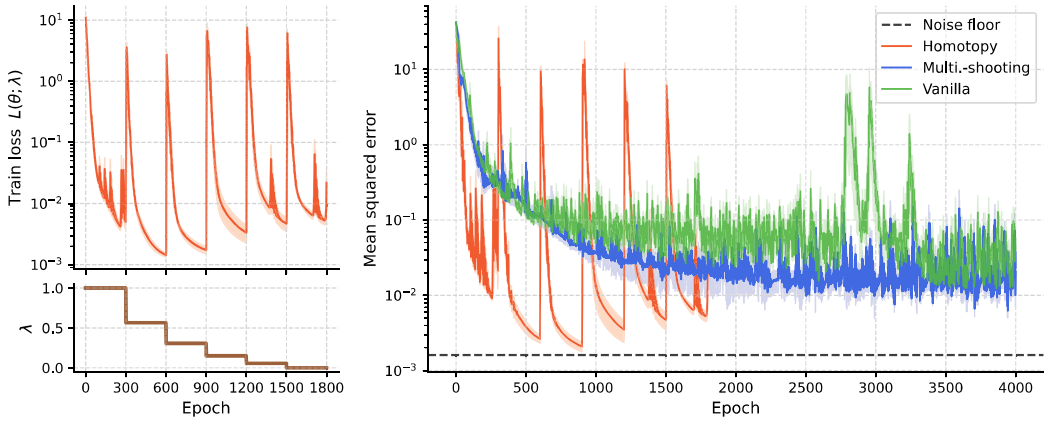

Figure 14: Train loss, homotopy parameter, and mean squared error as a function of epochs for the results of Figures 4, 5 and 6. The layout of the plots, as well as their interpretations are identical to that of Figure 13.

G.3 Further discussion on the Lotka-Volterra system results

In this section, we present additional results regarding our experiment of Figure 3 and continue our discussion.

Increasing data length. Figure 15 depicts the predicted trajectories for differently trained models. For the shortest data, it can be seen that all models did overfit on the training data and failed to capture the periodic nature of the system. This justifies our attributing the large extrapolation error of the models in Figure 3 to overfitting. However, we emphasize that this overfitting is not due to failings in the training algorithms, but rather due to the insufficient information in the training data, as the models cannot be expected to learn periodicity without being given at least a single period worth of data. As the data is increased, we find that both homotopy and multiple shooting properly capture the dynamics of the system, whereas the vanilla method was unable to properly learn the system for the longest data.

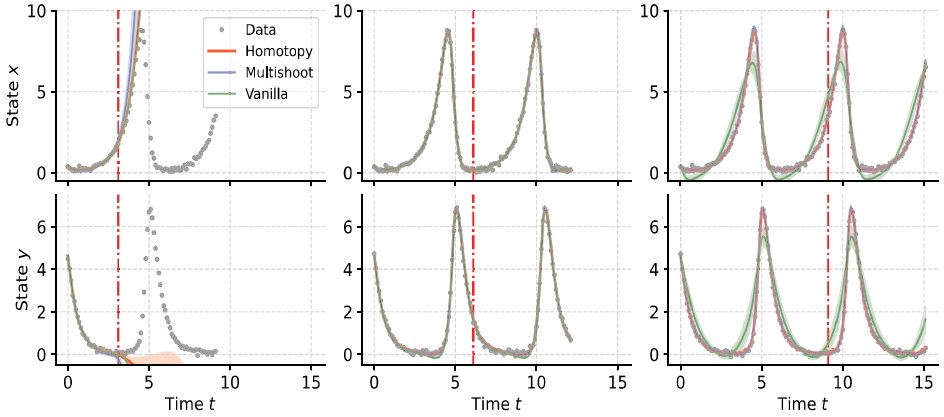

Figure 15: Predicted trajectories for the models with increasing train data length, t=(3.1, 6.1, 9.1). Corresponds to the first panel of Figure 3. The red dashed line indicates the start of extrapolation.

Reducing model capacity. In Figure 16, we show the prediction trajectories corresponding to the model capacity experiment. It is clear from the results that both the homotopy and multiple shooting methods can produce models that accurately portray the dynamics, regardless of the reduction in the model size. In contrast, predictions from vanilla training deteriorate as the model size decreases, with the smallest model inaccurately estimating both the amplitude and the phase of the oscillations.

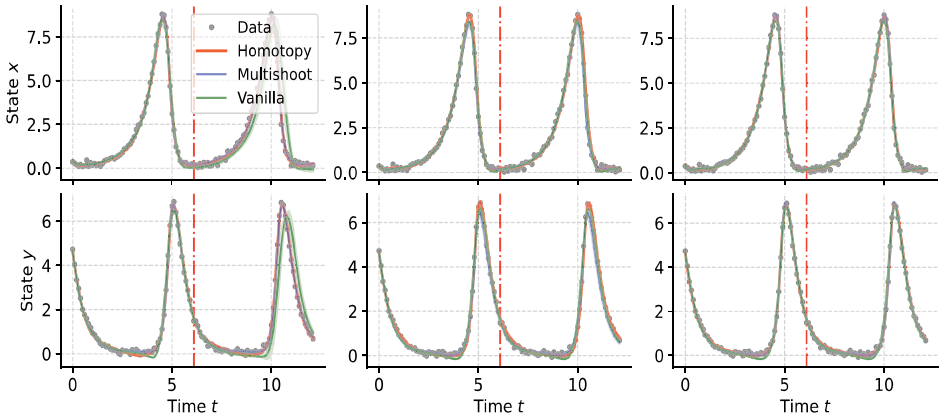

Figure 16: Training curves for each number of nodes, nodes=(8, 16, 32). Corresponds to the second panel of Figure 3.

Increasing sampling period. The model trajectories for decreasing sampling period in the training data is shown below in Figure 17. Intriguingly, we find that the multiple shooting method results suffer greatly with the increased data sparsity. This is a side effect of our experiment setting, where we set the time interval constant while increasing the sampling period. To elaborate, our choice of fixed time interval causes the number of training data points decrease as the period increases. However, as

multiple shooting trains by subdividing the training data, it struggles on small datasets because this leads to each segment containing even less data points that do not convey much information about the data dynamics.

Another interesting observation that can be made is that vanilla training gives better results when sparser data is used, which is also confirmed by the error values in the third panel of Figure 3. This suggests an alternate training improvement strategy for training NeuralODEs on long time series data - by training on subsampled data, then if necessary, gradually anneal the data intervals back to its original form as training proceeds. Of course, the effectiveness of such a scheme will need to be tested for more complex systems, which is outside the scope of this paper.

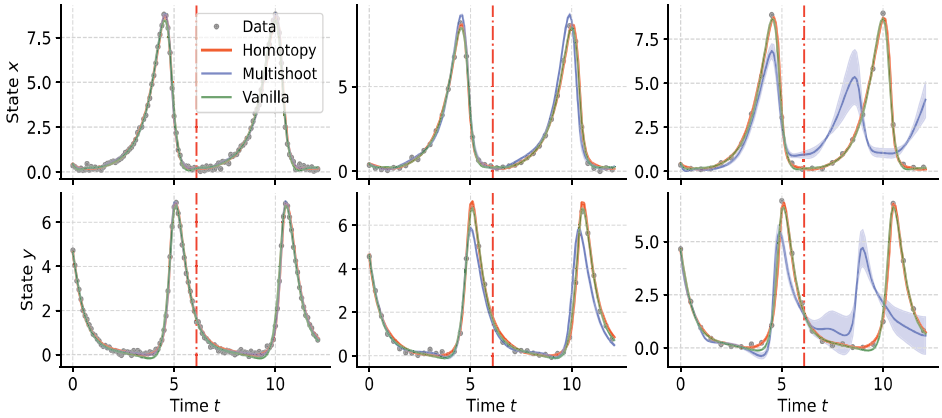

Figure 17: Predicted trajectories for the models with increasing sampling periods, dt = (0.1, 0.3, 0.5). Corresponds to the third panel of Figure 3.

As our homotopy training method utilized a cubic smoothing spline to supply the coupling term, we also inspected the quality of this interpolant as the data period was increased. From Figure 18, we find that the cubic spline relatively resistant to the increased data sparsity, which may be one of the reasons why our homotopy method is very robust against this mode of data degradation.

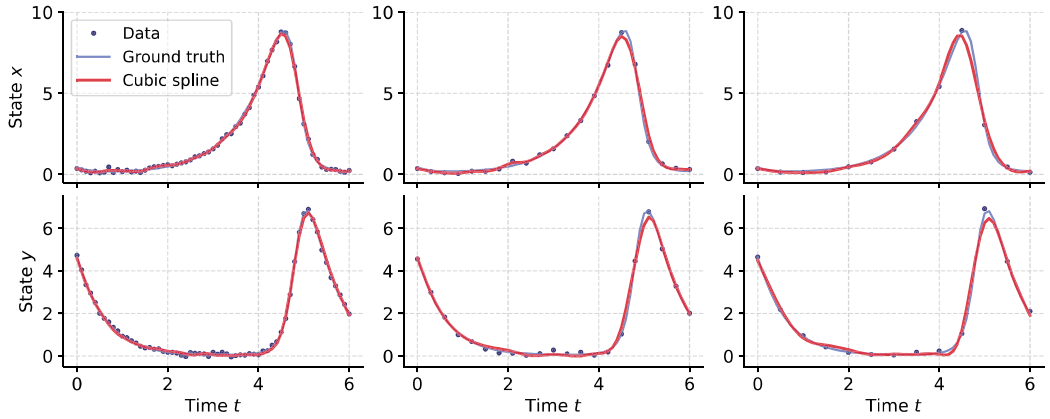

Figure 18: Cubic spline interpolation results for data with increasing sampling periods, dt=(0.1, 0.3, 0.5). Corresponds to the third panel of Figure 3.

Increasing data noise. Finally, Figure 19 shows the model trajectories as the noise in the training data is increased. Here, we do find that while all methods return deteriorating predictions as the noise is increased, our homotopy method is found to be most robust to noise, followed by multiple shooting, and finally vanilla gradient descent. A part of our algorithm's robustness to noise can be explained by the use of the cubic smoothing splines. As our use of synchronization couples the NeuralODE dynamics to the cubic spline trajectory during training, one could argue that the denoising effect of cubic splines is thus directly transferred to the model.

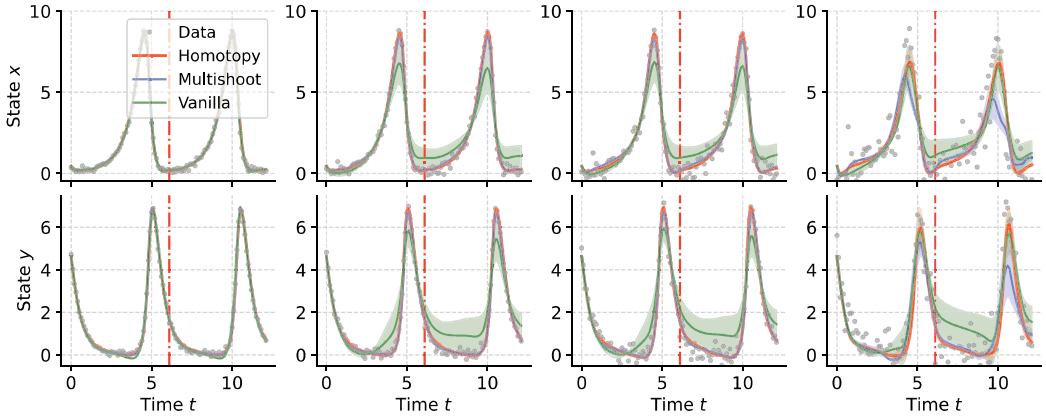

Figure 19: Predicted trajectories for the models with increasing noise, noise=(0.05, 0.1, 0.2, 0.5). Corresponds to the fourth panel of Figure 3.

However, this does not seem to be the full picture. Figure 20 shows the cubic smoothing splines used during modeling training for each of the noise levels. As we held the amount of smoothing constant throughout our experiments, we see that for larger noise amplitudes, the spline fails to reject all noise, and displays irregular high frequency behaviors. On the other hand, the corresponding trained trajectories for the homotopy method (Figure 19, third and fourth panels) do not mirror such irregular oscillations, indicating that the homotopy optimization procedure itself also has an intrinsic robustness to noise.

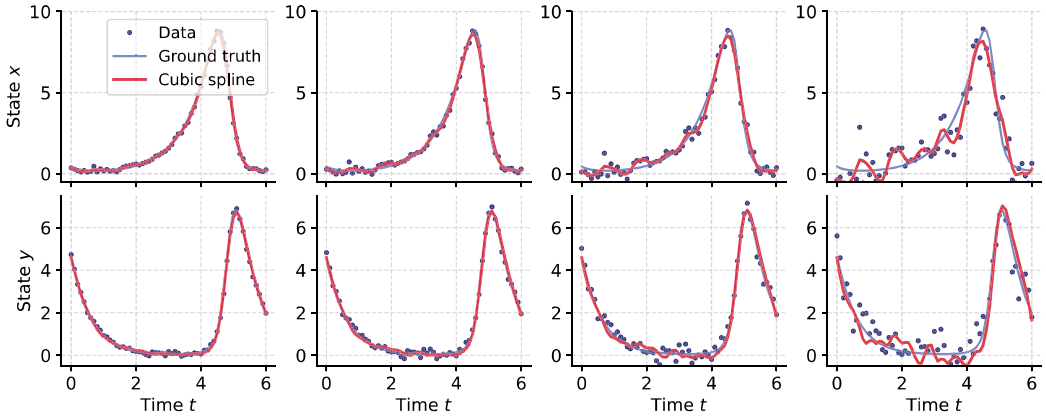

Figure 20: Cubic spline interpolation results for data with increasing noise amplitude, noise=(0.05, 0.1, 0.2, 0.5). Corresponds to the fourth panel of Figure 3.

G.4 Comparison to a non-deep learning algorithm: SINDy

In this section, we briefly compare our algorithm to a more traditional symbolic regression-based method. We choose the well-known SINDy algorithm, which is readily available in the `pysindy` package. This algorithm takes as input, time series measurements of the states of the dynamical system in question, as well as a dictionary of candidate terms that could constitute the governing equation. Afterwards, the time derivative of the states is estimated numerically, and a sparse regression is performed to determine which terms exist in the governing equation, as well as what their coefficient values are.

Compared to neural network-based approaches, the unique feature of SINDy is that its results are given in the form of analytical expressions. Therefore if the governing equation for the data is simple (e.g. consists of low-order polynomial terms), or if the user has sufficient prior information about the system and can construct a very small dictionary that contains all the terms that appear in the true equation, SINDy can accurately recover the ground truth equation.

On the other hand, if the system is high dimensional so that the space of possible candidate terms start to grow drastically, or if the governing equation has a arbitrary, complex form that cannot be easily guessed, SINDy either does not fit the data properly, or even if it does, recovers inaccurate equations that are difficult to assign meaning to. These correspond to situations where NeuralODEs, and hence our method for effectively training them, shines.

To illustrate this point, we choose the double pendulum dataset and compare the predicted trajectories from SINDy and our homotopy algorithm. As one can see from Equation (13), the governing equation of this system has an extremely complicated form that cannot be guessed easily. To reflect this difficulty, we chose the basis set of the dictionary of the candidate terms to be the state variable themselves, their sines and cosines, and the higher harmonics of the sines and cosines.

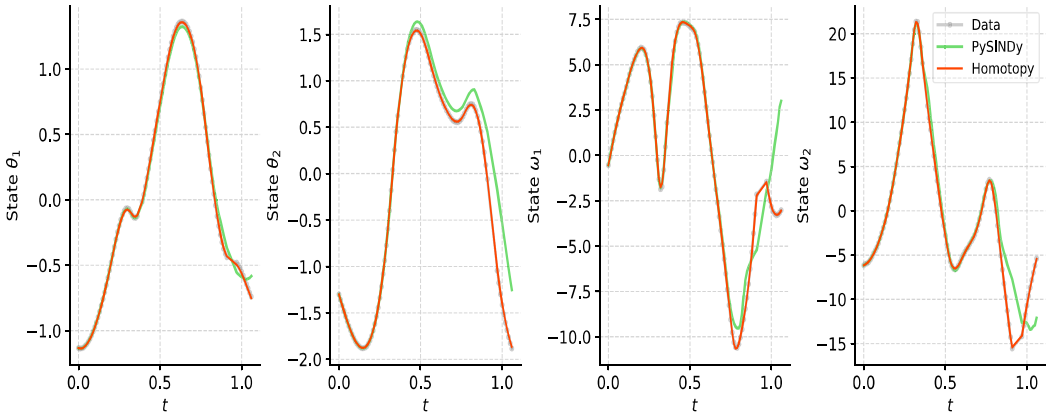

Figure 21: Comparison between our homotopy method and SINDy on the double pendulum dataset.

From Figure 21, we find that SINDy is able to fit the data to a modest accuracy. While this may seem weak compared to the result from our homotopy method, the accuracy of the SInDy prediction is likely to improve if the candidate dictionary is updated to better reflect the actual form of Equation (13). However, this amounts to supplying the algorithm with excessive amounts of prior information about the system, which is rarely available in practice. This is also constrasted with NeuralODE training with our homotopy method - which does not require any additional information about the system to produce the results above.

G.5 Changing the gradient calculation scheme

As we commented in Section 3, methods that focus on other aspects of NeuralODE training, such as alternate gradient calculation schemes, are fully compatible with our homotopy training method. To demonstrate this point, we used the symplectic-adjoint method [25] provided in the `torch-symplectic-adjoint` library as a substitute for the direct backpropagation used for gradient calculation in our experiments.

Table 6: Benchmark results with the gradient calculation scheme changed to the symplectic-adjoint method.

Dataset	Lotka-Volterra		Double Pendulum		Lorenz System
Model Type	Black Box	Gray Box	Black Box	Second Order	Black Box
			Best train epochs		
Baseline	(3999, 3999, 3853)	(3997, 3999, 3992)	(3909, 3384, 3998)	(3983, 3998, 3972)	(3994, 3999, 3990)
Multi-. Shoot.	(3982, 3968, 3670)	(3901, 3958, 3996)	(3805, 3913, 3820)	(3988, 3970, 3942)	(3977, 3418, 3954)
Homotopy	**(299, 208, 228)**	**(598, 265, 291)**	**(1201, 1799, 1774)**	**(900, 1799, 900)**	**(1799, 918, 1799)**
			Mean Squared Error ($\times 10^{-2}$)		
Baseline (interp.)	2.47±0.13	187±133	1.69±0.20	1.02±0.12	184±60.1
Multi-. Shoot.(interp.)	1.20±0.02	**0.95±0.02**	**0.76±0.01**	0.61±0.02	**29.1±15.8**
Homotopy (interp.)	**0.77±0.04**	1.68±0.76	1.08±0.12	**0.25±0.03**	**33.3±5.43**

Comparing the above results to our benchmark results in Figure 4 and Table 5, we see that the overall results remain the same - our homotopy method is able to train the models effectively, in much small number of epochs.