A Repository and documentation

The repository for this project can be found at `https://github.com/nikihowe/myriad`.

The documentation for this project can be found at `https://nikihowe.github.io/myriad/html/myriad`.

B Implemented in Myriad

Here we give scores for 17 real-world environments, as well as for a "simple case" environment, which is a toy setting to be used for initial algorithm testing.

B.1 Environments

Name	Brief Description	Fixed $\boldsymbol{x}_T$	Terminal Cost
Bacteria*	Manage bacteria population levels	No	Yes
Bear Populations*	Manage metapopulation of bears	No	No
Bioreactor*	Grow bacteria population	No	No
Cancer Treatment*	Decrease tumour size via chemotherapy	No	No
Cart-Pole Swing-Up	Swing up pendulum by translating pivot	Yes	No
Epidemic*	Control epidemic via vaccination	No	No
Glucose*	Manage blood glucose via insulin injections	No	No
Harvest*	Maximize harvest yield	No	No
HIV Treatment*	Manage HIV via chemotherapy	No	No
Mould Fungicide*	Control mould population via fungicide	No	No
Mountain Car	Drive up valley with limited force	Yes	No
Pendulum	Swing up pendulum by rotating pivot	Yes	No
Predator Prey*	Minimize pest population	Yes	Yes
Rocket Landing	Land a rocket	Yes	No
Simple Case	Use for initial algorithm testing	No	No
Timber Harvest*	Optimize tree harvesting	No	No
Tumour*	Block tumour blood supply	No	Yes
Van Der Pol	Forced Van der Pol oscillator	Yes	No

Table 1: The environments currently available in the Myriad repository. Environments with an (*) were inspired by corresponding examples described by Lenhart and Workman (2007). Some environments require that the system end up in a specific state at the end of an episode. Also, some environments impose a final cost in addition to the instantaneous cost, calculated based on the system's final state. For a detailed description and motivation of each environment, see the documentation linked to in the Myriad repository.

B.2 Optimizers and integration methods

Method Name	Direct / Indirect	Sequential / Parallel	Integration Method
Single Shooting	Direct	Sequential	Any
Multiple Shooting	Direct	Partially Parallel	Any
Trapezoidal Collocation	Direct	Parallel	Trapezoidal Rule
Hermite-Simpson Collocation	Direct	Parallel	Simpson's Rule
Forward-Backward Sweep	Indirect	Sequential	Runge-Kutta 4th Order

Table 2: The trajectory optimization techniques available in the Myriad repository. Direct techniques discretize the control problem and solve the resulting nonlinear program. Indirect methods first augment the problem with an adjoint equation, before discretizing and solving.

Integration Method	Explicit / Implicit	Gradient Evaluations
Euler	Explicit	1
Heun	Explicit	2
Midpoint	Explicit	2
Runge-Kutta 4th Order	Explicit	4
Trapezoidal	Implicit	NA
Simpson	Implicit	NA

Table 3: The integration methods available in the Myriad repository.

C Benchmark scores

We provide tables of benchmark scores for several different algorithms.

C.1 Trajectory optimization on true dynamics

To begin with, in Table 4 we show the scores achieved by a trajectory optimization algorithm on all 17 real-world environments, as well as for the "simple case" environment. The cost is the integrated cost over the time horizon of the problem, plus any terminal cost. The defect is the difference between the final state and the desired final state of the system. To reproduce these results, choose a system in `config.py` and uncomment `run_trajectory_opt` in `run.py`. Note that for this setting, no training occurs: we simply run trajectory optimization on the true dynamics model.

System	Parameters	Cost	Defect
Bacteria	A: 1, B: 1, C: 1	-7.98	NA
Bear Populations	r: 0.1, K: 0.75, m_f: 0.5, m_p: 0.5	12.28	NA
Bioreactor	G: 1, D: 1	-1.39	NA
Cancer Treatment	δ: 0.45, r: 0.3	20.57	NA
Cart-Pole Swing-Up	g: 9.81, m_1: 1, m_2: 0.3, ℓ: 0.5	87.78	$[0 \quad 0 \quad 0 \quad 0]^\top$
Epidemic	see repository	13.40	NA
Glucose	a: 1, b: 1, c: 1	1354.02	NA
Harvest	A: 5, k: 10, m: 0.2, M: 1	-6.51	NA
HIV Treatment	s: 10, m_1: 0.02, m_2: 0.5, m_3: 4.4, r: 0.03, T_{max}: 1500, k: 0.000024, N: 300, A: 0.05	-823.13	NA
Mould Fungicide	r: 0.3, M: 10	23.50	NA
Mountain Car	g: 0.0025, p: 0.0015	8.57	$[0 \quad 0]^\top$
Pendulum	g: 10, m: 1, ℓ: 1	25.75	$[0 \quad 0]^\top$
Predator Prey	d_1: 0.1, d_2: 0.1	1.79	0
Rocket Landing	g: 9.81, m: 100000, ℓ: 50, w: 10	178.02	$\begin{bmatrix} -172.78 \\ 0 \\ -1413.87 \\ -195.49 \\ -36.04 \\ -2.28 \end{bmatrix}$
Simple Case	A: 1, B: 1, C: 4	-1.35	NA
Timber Harvest	K: 1	-5104.67	NA
Tumour	ξ: 0.084, b: 5.85, d: 0.00873, G: 0.15, μ: 0.02	7571.67	NA
Van Der Pol	a: 1	2.87	$[0 \quad 0]^\top$

Table 4: Summary of performance of direct single shooting on the various environments. For these experiments, we used one shooting trajectory with 100 controls. The Heun method was used for integration. We used `ipopt` to solve the resulting nonlinear program.

C.2 Trajectory optimization on learned dynamics

In addition to performance results using trajectory optimization on the true dynamics models, we also provide benchmark scores of trajectory optimization on models which have been learned from data.

C.2.1 Parametric models

Table 5 shows the results of performing optimization on structured parametric models (known dynamics, unknown coefficients) of which the parameters are learned from data, for 12 of the 17 real-world environments, and for the "simple case" environment. For convenience, the corresponding performance achieved using trajectory optimization on the true model is also provided (to the left of the "/"). To reproduce these results, choose a system in `config.py` and uncomment `run_mle_sysid` in `run.py`. Run with the default hyperparameters in `config.py`, which were chosen via a small amount of trial-and-error. In this setting, we found that a much smaller data regime (train set of 10 trajectories instead of 100) also led to good performance, indicating that using a structured model significantly simplifies the system identification problem.

System	Parameters (true/learned)	Cost (best/achieved)	Defect (best/achieved)
Bacteria	A: 1/1, B: 1/1, C: 1/1	$-7.98/-7.98$	NA
Bear Populations	r: 0.1/0.1, K: 0.75/0.75, m_f: 0.5/0.5, m_p: 0.5/0.5	$12.28/12.28$	NA
Bioreactor	G: 1/1, D: 1/1	$-1.39/-1.39$	NA
Cancer Treatment	δ: 0.45/0.45, r: 0.3/0.3	20.57/20.57	NA
Cart-Pole Swing-Up	g: 9.81/9.81, m_1: 1/1, m_2: 0.3/0.3, ℓ: 0.5/0.5	87.78/87.78	$\begin{bmatrix}0/0\\0/0\\0/0\\0/0\end{bmatrix}$
Glucose	a: 1/1, b: 1/1, c: 1/1	1354.02/1354.02	NA
Mould Fungicide	r: 0.3/0.3, M: 10/10	23.50/23.50	NA
Mountain Car	g: 0.0025/0.0025, p: 0.0015/0.0015	8.57/8.57	$[0/0 \quad 0/0]^\top$
Pendulum	g: 10/11.943, m: 1/0.701, ℓ: 1/1.194	25.75/25.49	$[0/0.04 \quad 0/0.04]^\top$
Predator Prey	d_1: 0.1/0.1, d_2: 0.1/0.1	1.79/1.79	0/0
Timber Harvest	K: 1/1	$-5104.67/-5104.67$	NA
Tumour	ξ: 0.084/0.084, b: 5.85/5.170, d: 0.00873/0.00873, G: 0.15/0.15, μ: 0.02/-0.660	7571.67/7571.73	NA
Van Der Pol	a: 1/1	2.87/2.87	$[0/0 \quad 0/0]^\top$

Table 5: Summary of performance of trajectory optimization on parametric models with parameters learned from data on a variety of environments. The first column lists the environment, followed by the true and learned parameters, and then the cost (and if applicable, defect) resulting from those parameters.

C.2.2 Neural ODE models

In general when acting in real-world environments, we do not know the system dynamics – or even a parametric model describing the general form of the system dynamics – beforehand. As such, it is desirable to be able to learn system dynamics entirely from data. Table 6 shows the performance of trajectory optimization on a Neural ODE model learned from data, for 12 of the 17 real-world environments, and for the "simple case" environment. To reproduce these results, choose a system in `config.py` and uncomment `run_node_mle_sysid` in `run.py`. Use the default hyperparameters, except as indicated in the final two columns of Table 6, which show the number of trajectories per dataset and total number of datasets used during training. To select these hyperparameters, we ran training with train sets of size 10 and 100 with up to 5 datasets trained on sequentially (except in Cart-Pole Swing-Up, where we used up to 10 datasets). We then chose the dataset size and number of datasets which led to the best performance when performing trajectory optimization on the learned model.

System	Cost (best/achieved)	Defect (best/achieved)	# Traj.s/D.set	# D.sets
Bacteria	$-7.98/-2.72$	NA	10	5
Bear Populations	$12.28/12.29$	NA	10	4
Bioreactor	$-1.39/-1.39$	NA	10	1
Cancer Treatment	20.57/20.57	NA	10	1
Cart-Pole Swing-Up	87.78/242.25	$\begin{bmatrix} 0/-0.95 \\ 0/0.23 \\ 0/-0.85 \\ 0/6.63 \end{bmatrix}$	100	7
Glucose	1354.02/1354.02	NA	10	1
Mould Fungicide	23.50/23.50	NA	10	1
Mountain Car	8.57/15.87	$\begin{bmatrix} 0/0.03 \\ 0/-0.01 \end{bmatrix}$	100	5
Pendulum	25.42/20.88	$\begin{bmatrix} -0.01/0.37 \\ -0.01/0.61 \end{bmatrix}$	100	1
Predator Prey	1.79/1.95	0/0.14	100	5
Timber Harvest	$-5104.67/-3928.77$	NA	100	1
Tumour	7571.67/8468.68	NA	100	2
Van Der Pol	2.87/11.12	$\begin{bmatrix} 0/-0.35 \\ 0/-1.76 \end{bmatrix}$	10	3

Table 6: Summary of performance of trajectory optimization on Neural ODE models learned from data on a variety of environments. The layout is the same as the previous table, except that parameters are omitted, since there is no way to directly compare the true parameters with the weights and biases of the neural network. Instead, the fourth column shows the number of trajectories per dataset used in training, and the fifth column shows the total number of datasets used to train the model.

D End-to-end algorithm performance

In Table 7 we present the performance of running the end-to-end implicit planning imitation learning algorithm described in Section 7, for 13 of the 17 real-world environments, and for the "simple case" environment. To reproduce these results, choose a system in `config.py` and uncomment `run_node_endtoend` in `run.py`. Use default hyperparameters, except as indicated in the final table column, which shows the size of the two hidden layers of the Neural ODE. To choose a hidden layer size, we tried running smaller (50, 50) and larger (100, 100) networks, and selected the one that achieved better performance on the given environment. One other important hyperparameter is the number of implicit planning steps to perform at each iteration. We found 5 steps (default in the code; found via manual search) to strike a good balance in providing a planning inductive bias while still enabling fast propagation of gradients.

System	Cost (best/achieved)	Defect (best/achieved)	Neurons/Layer
Bacteria	$-7.98/-7.60$	NA	100
Bear Populations	$12.28/12.40$	NA	100
Bioreactor	$-1.39/-1.39$	NA	100
Cancer Treatment	20.57/20.58	NA	50
Cart-Pole Swing-Up	87.78/36.90	$\begin{bmatrix} 0/1.21 \\ 0/-4.88 \\ 0/-0.29 \\ 0/-3.28 \end{bmatrix}$	100
Glucose	1354.02/1354.12	NA	100
HIV Treatment	$-823.13/-822.96$	NA	100
Mould Fungicide	23.50/23.54	NA	50
Mountain Car	8.57/3000	$\begin{bmatrix} 0/-0.39 \\ 0/-0.02 \end{bmatrix}$	50
Pendulum	25.53/1.90	$\begin{bmatrix} 0/-2.69 \\ 0/-0.08 \end{bmatrix}$	100
Predator Prey	1.79/1.04	$0/-1.84$	100
Timber Harvest	$-5104.67/-500.00$	NA	50
Tumour	7571.67/9161.23	NA	50
Van Der Pol	$2.87/16.81$	$\begin{bmatrix} 0/0.54 \\ 0/-1.75 \end{bmatrix}$	100

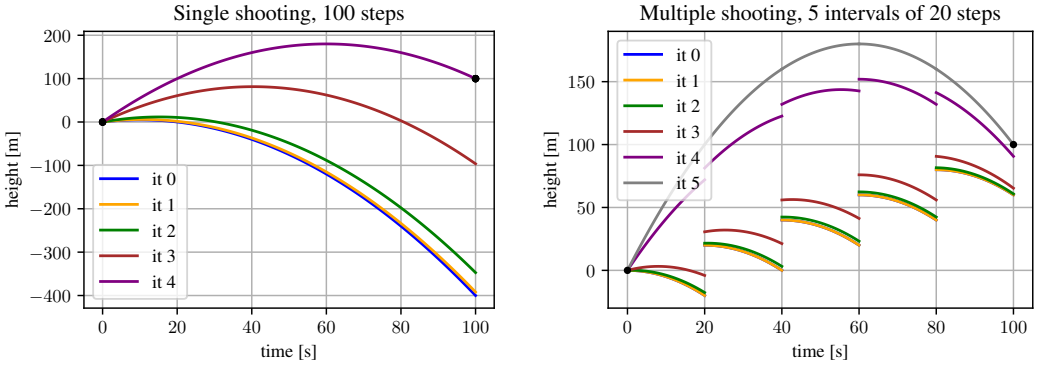
Table 7: Summary of performance of end-to-end learning and planning with Neural ODE models on a variety of environments. The first column lists the environment. The second column indicates the cost of applying the controls solved for with the model, applied in the true environment. The third column shows the defect of the final state from the desired final state, if any. The fourth column shows the size of the hidden layers of the neural network that was used for the model.

E Description of the direct multiple shooting algorithm

The direct single shooting approach does not allow us to impose constraints on the state trajectory, and is inherently sequential. Direct multiple shooting addresses both these shortcomings by breaking the problem into a sequence of *shooting intervals* on which direct single shooting can be applied in parallel. As such, it is sometimes advantageous to employ direct multiple shooting instead of direct single shooting.

The nonlinear program resulting from direct multiple shooting is presented below, followed by a comparison of the two direct shooting techniques in Figure 5, on the same toy problem presented in Section 4.

$$
\begin{aligned}
&\text{decision variables} && \hat{\boldsymbol{x}}_0, \hat{\boldsymbol{x}}_k, \hat{\boldsymbol{x}}_{2k}, \ldots, \hat{\boldsymbol{x}}_{N-k}, \hat{\boldsymbol{x}}_N, \hat{\boldsymbol{u}}_0, \hat{\boldsymbol{u}}_1, \hat{\boldsymbol{u}}_2, \ldots, \hat{\boldsymbol{u}}_N \\
&\text{objective} && \left[\sum_{j=1}^{N/k} \int_{t_{(j-1)k}}^{t_{jk}} f_{\text{aug}} \left(\begin{bmatrix} \boldsymbol{x}(t) \\ c(t) \end{bmatrix}, \psi(\hat{\boldsymbol{u}}_{(j-1)k:jk}, t), t \right) dt \right] \texttt{[-1]} \\
&\text{equality constraints} && \hat{\boldsymbol{x}}_{ik} = \hat{\boldsymbol{x}}_{(i-1)k} + \int_{t_{(i-1)k}}^{t_{ik}} f(\boldsymbol{x}(t), \psi(\hat{\boldsymbol{u}}_{(i-1)k:ik}, t)) \quad \text{for } i = 1, 2, \ldots, N/k \\
& && \hat{\boldsymbol{x}}_0 = \boldsymbol{x}_s \\
&* && \hat{\boldsymbol{x}}_N = \boldsymbol{x}_f \\
&\text{inequality constraints}* && \boldsymbol{x}_{ik}^{\text{lower}} \leq \hat{\boldsymbol{x}}_{ik} \leq \boldsymbol{x}_{ik}^{\text{upper}} \quad \text{for } i = 0, 1, \ldots, N/k \\
&* && \boldsymbol{u}_i^{\text{lower}} \leq \hat{\boldsymbol{u}}_i \leq \boldsymbol{u}_i^{\text{upper}} \quad \text{for } i = 0, 1, \ldots, N
\end{aligned}
\tag{14}
$$



(a) Trajectories resulting from 0 to 4 iterations of direct single shooting.

(b) Trajectories resulting from 0 to 5 iterations of direct multiple shooting.

Figure 5: Comparison of direct single shooting (a) and direct multiple shooting (b), applied to a toy dynamics problem. While direct single shooting converges in fewer iterations, each iteration takes longer than those performed in direct multiple shooting. Additionally, the parallelism of direct multiple shooting might enable it to tackle problems with longer time horizons.

F Incorporating inequality constraints

For real-world systems, we usually would like to restrict the agent to only take certain paths through state space, avoiding dangerous or otherwise undesirable areas. Such restrictions can be expressed as bounds on the state variables in the nonlinear program resulting from transcription of the trajectory optimization problem. Note that in the single shooting setting, there are no state decision variables; this technique is only possible when using multiple shooting or collocation techniques.

Once we have inequality constraints in the nonlinear program, we must find a solution which satisfies these constraints. When using a Lagrangian-based approach, there are at least two straightforward ways of doing this. The first approach, which is implemented in Myriad, is *projection*. Over the course of optimization, after a gradient step is taken, the resulting iterate is projected back into the feasible set. With fixed bounds, this can be implemented as a `clip` operation (Bertsekas, 1999).

An alternative approach, which can be included directly in the system definition, is that of *reparametrization*. Instead of stepping and then modifying the iterate to satisfy the bounds, we instead modify the space in which we are performing the optimization, so that any point in the space will be feasible (Niculae, 2020). For example, if our feasible set is $x_{\text{lower}} \leq x \leq x_{\text{upper}}$, a viable reparametrization would be to use a sigmoid of the form $\sigma(x, x_{\text{lower}}, x_{\text{upper}}) = (x_{\text{upper}} - x_{\text{lower}})/(1 + e^{-\alpha x}) - x_{\text{lower}}$, where α is a temperature constant which can be decreased over time.

G Compute

All experiments were performed on a personal laptop with the following specifications:

- 2.7 GHz Quad-Core Inter Core i7
- Intel Iris Plus Graphics 665 1536 MB
- 16 GB 2133 MHz LPDDR3
- 500 GB PCI-Express SSD

The average runtime for experiments is presented in Table 8.

Experiment	Runtime
Trajectory Optimization	$\sim$ 1 minute per environment
System Identification (Parametrized)	$\sim$ 2 minutes per environment
System Identification (Neural ODE)	$\sim$ 2 hours per environment
End-to-end Control (Parametrized)	$\sim$ 0.5 hours per environment
End-to-end Control (Neural ODE)	$\sim$ 10 hours per environment
Total for all experiments (above times 18)	$\sim$ 300 hours

Table 8: The approximate amount of compute used for all experiments.