Continuous-Time Model-Based Reinforcement Learning Supplementary Material

Çağatay Yıldız [1] Markus Heinonen [1] Harri Lähdesmäki [1]

S1. Technical Discussion

S1.1. The Form of the Differential Function

For notational convenience, we defined the problem as a first-order ODE system:

$$\dot{\mathbf{s}}(t) = \frac{d\mathbf{s}(t)}{dt} = \mathbf{f}(\mathbf{s}(t), \mathbf{a}(t)).$$

We now explain possible extensions to this formulation to inject problem-specific prior knowledge.

Linearity w.r.t. action In many Newtonian systems, the action affects the time differential linearly, a widespread assumption in CT control literature (Doya, 2000). Estimating such systems with an arbitrary function of state and action would tie the action and the dynamics in a nonlinear way, which would render the learning problem unnecessarily complicated. Therefore, we propose to decompose the differential function as follows (Vamvoudakis and Lewis, 2010):

$$\frac{d\mathbf{s}(t)}{dt} = \mathbf{f}(\mathbf{s}(t)) + \mathbf{h}(\mathbf{s}(t)) \cdot \mathbf{a}(t).$$

where $\mathbf{h} : \mathbb{R}^d \mapsto \mathbb{R}^{d\times m}$ and $\cdot$ denotes the standard matrix-vector product. Above formulation assumes an additive differential function of *dynamics* and *control* component. The former aims to learn the evolution of the system under zero force whereas $\mathbf{h}$ defines a manifold the action is projected onto. Since we approximate these functions with neural networks, both the dynamics and the manifold can be arbitrarily complicated.

Second-order dynamics Most dynamical systems can be expressed in terms of position $\mathbf{s} \in \mathbb{R}^d$ and velocity $\mathbf{v} \in \mathbb{R}^d$ components. Such decomposition of the state space is shown to better explain the phenomena of interest if the underlying physics is indeed second-order (Yildiz et al., 2019). Formally, a second-order dynamical system with control is defined as follows:

$$\frac{d\mathbf{s}(t)}{dt} = \mathbf{v}(t), \qquad \frac{d\mathbf{v}(t)}{dt} = \mathbf{f}(\mathbf{s}(t), \mathbf{v}(t), \mathbf{a}(t)).$$

Here, $\mathbf{f} : \mathbb{R}^{2d+m} \to \mathbb{R}^d$ is referred to as *acceleration field*.

Hamiltonian dynamics Zhong et al. (2020) already describes Hamiltonian dynamics in RL context very clearly; however, we include this subsection for completeness. Hamiltonian mechanics reformulate classical physical systems in terms of canonical coordinates $(\mathbf{q}, \mathbf{p})$ with $\mathbf{q}, \mathbf{p} \in \mathbb{R}^d$, where $\mathbf{q}$ denotes generalized coordinates and $\mathbf{p}$ is their conjugate momenta. The time evolution of a Hamiltonian system is defined as

$$\frac{d\mathbf{q}}{dt} = \frac{d\mathcal{H}}{d\mathbf{p}}, \qquad \frac{d\mathbf{p}}{dt} = -\frac{d\mathcal{H}}{d\mathbf{q}}$$

where $\mathcal{H}(\mathbf{q}, \mathbf{p}) : \mathbb{R}^{2d} \to \mathbb{R}$ denotes the Hamiltonian. For simplicity, we assume a time-invariant Hamiltonian, which also corresponds to the total energy of the system. Typically, Hamiltonian is decomposed into a sum of kinetic energy T and potential energy V:

$$\mathcal{H} = T + V, \qquad T = \frac{\mathbf{p}^T M^{-1}(\mathbf{q})\mathbf{p}}{2m}, \qquad V = V(\mathbf{q})$$

where m denotes the mass. In simple systems such as pendulum, the mass matrix $M(\mathbf{q})$ is an identity matrix, implying Euclidean geometry. More complicated systems like cartpole requires learning the geometry. Given a Hamiltonian decomposing like above, the dynamics become

$$\frac{d\mathbf{q}}{dt} = \frac{M^{-1}(\mathbf{q})\mathbf{p}}{m}, \qquad \frac{d\mathbf{p}}{dt} = -\frac{dV}{d\mathbf{q}}$$

The dynamics learning problem reduces to learning a potential energy function $\mathbb{V}(\mathbf{q}) : \mathbb{R}^d \to \mathbb{R}$, whose derivative gives the time evolution of momentum, and estimating the geometry through its Cholesky decomposition L, i.e., $LL^{-1} = M$, via an additional neural network.

S1.2. Greedy policy

A greedy policy is defined as the one that minimizes the Hamilton–Jacobi–Bellman equation:

$$V^*(\mathbf{s}) = \min_{\mathbf{a}} \left[\frac{dV(\mathbf{s})}{d\mathbf{s}} \cdot \mathbf{f}(\mathbf{s}_t, \mathbf{a}) + r(\mathbf{s}, \mathbf{a}) \right] \qquad (1)$$

[1]Department of Computer Science, Aalto University, Finland. Correspondence to: Çağatay Yıldız <cagatay.yildiz@aalto.fi>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

The greedy policy can be expressed in closed form if *(i)* the reward is of the form $r(\mathbf{s},\mathbf{a}) = r_{\mathbf{s}}(\mathbf{s}) + r_{\mathbf{a}}(\mathbf{a})$ with an invertible action reward and *(ii)* the system dynamics is linear with respect to the action as in eq. (S1.1) (Doya, 2000; Tassa and Erez, 2007):

$$\mathbf{a}_t^* = -\frac{dr_{\mathbf{a}}^{-1}}{d\mathbf{a}}\left(\frac{\mathbf{f}(\mathbf{s}_t,\mathbf{a})}{d\mathbf{a}}^T\frac{dV(\mathbf{s})}{d\mathbf{s}}^T\right) \tag{2}$$

Consequently, given the above assumptions are satisfied, the optimal policy can be expressed in closed form for a given value function, which would obviate the need for an actor. We leave the investigation of model-based greedy policy as an interesting future work.

S2. Experiment Details

This section consists of experimental details which are not included in the main text.

S2.1. Environments

The environment-specific parameters are given in Table S1. In all environments, the actions are continuous and restricted to a range $[-a_{\text{max}}, a_{\text{max}}]$. Similar to Zhong et al. (2020), we experiment with the fully actuated version of the Acrobot environment since no method was able to solve the under-actuated balancing problem.

S2.2. Reward Functions

Assuming that each observation $\mathbf{s} = (\mathbf{q},\mathbf{p})$ consists of state (position) $\mathbf{q}$ and velocity (momentum) $\mathbf{p}$ components, the differentiable reward functions have the following form:

$$r(\mathbf{q},\mathbf{p},\mathbf{a}) = \exp\left(-||\mathbf{q}-\mathbf{s}^{\text{goal}}||_2^2 - c_{\mathbf{p}}||\mathbf{p}||_2^2\right) - c_{\mathbf{a}}||\mathbf{a}||_2^2$$

where $c_{\mathbf{p}}$ and $c_{\mathbf{a}}$ denote environment-specific constants. The exponential function aims to restrict the reward in a range [0,1] minus the action cost, which aids learning (Deisenroth and Rasmussen, 2011). The constants $c_{\mathbf{p}}$ and $c_{\mathbf{a}}$ are set so that they *(i)* penalize large values, and *(ii)* do not enforce the model to stuck at trivial local optima such as the initial state.

Goal states The goal state $[0,\ell]$ in Pendulum environment corresponds to x and y coordinates of pole's tip, with ℓ being the length of the pole. The reward is maximized when the pole is fully upright. In CartPole, this state is concatenated with 0, which represents the cart's target location. Finally in Acrobot, the goal state involves the x and y coordinates of the second link only (hence $2D$).

S2.3. Dataset

Initial dataset Each experiment starts with collecting an initial dataset of N_0 trajectories at observation time points with length $T = 50$. In all environments, the initial state is distributed uniformly:

$$\mathbf{s}_0 \sim \mathcal{U}[-\mathbf{s}^{\text{box}}, \mathbf{s}^{\text{box}}].$$

Initial random actions are drawn from a Gaussian process:

$$\mathbf{a}_t \sim \mathcal{GP}(\mathbf{0}, K(t,t')).$$

The inputs to the GP are the observation time points. We opt for a squared exponential kernel function with $\sigma = 0.5$ and $\ell = 0.5$. The output of the GP is followed by a TANH function and multiplied with a_{max}.

Data collection After each round, the policy is executed once in the environment starting from an initial state drawn from the environment. This is followed by the execution of N_{exp} exploring policy functions. Similar to the idea of perturbing policies with an Ornstein–Uhlenbeck process for exploration (Lillicrap et al., 2016), we add draws from a zero-mean Gaussian process to the policy for smoother perturbations:

$$\boldsymbol{\pi}^{\text{explore}}(\mathbf{s}(t),t) := \boldsymbol{\pi}(\mathbf{s}(t)) + \mathbf{z}(t)$$
$$\mathbf{z}(t) \sim \mathcal{GP}(\mathbf{0}, k(t,t'))$$

where the input to the GP is a set of time points. We again use a squared exponential kernel function with $\sigma = 0.1$ and $\ell = 0.5$. We choose the initial values for the exploring data sequences from the previous experience dataset randomly, where each state has a weight propotional to the dynamics estimator's variance at that state.

S2.4. Training Details

We used ADAM optimizer to train all the model components (Kingma and Ba, 2014). More explanation is as follows:

- **NODE Dynamics:** We initialize the differential function by gradient matching:

  $$\mathbf{f}(\mathbf{s}_i,\mathbf{a}_i) \approx \frac{\mathbf{s}_{i+1}-\mathbf{s}_i}{\mathbf{t}_{i+1}-t_i}$$

  Regardless of how observation time points are distributed, we use subsequences of length $t_s = 5$ to train the dynamics model. We randomly pick 5 subsequences from each data trajectory to reduce gradient stochasticity. While training the neural ODE model, we start with an initial learning rate of 1e-4, gradually increase it to 1e-3 in 100 iterations, and then proceed $N_{\text{dyn}} = 1250$ iterations with the latter learning rate.
- **Discrete Dynamics:** We train PETS and deep PILCO with the algorithms given respective papers.
- **Actor-Critic:** In each round, we form a dataset of initial values from the experience dataset. We chose to

exclude the sequences collected with exploring policy as they may explore states that are undesirably far from the goal. We set $N_{\text{ac}} = 250$.

S2.5. Neural Network Architectures

The dynamics, actor and critic functions are approximated by multi-layer perceptrons. In all methods and environments, we used the same neural network architectures, which are given as follows:

- **Dynamics:** 3-hidden layers, 200 hidden neurons and ELU activations. We experimentally observed that dynamics functions with ELU activations tend to extrapolate better on test seqeunces. Therefore, training the dynamics model with an augmented dataset (after each round) becomes much more robust.
- **Actor:** 2-hidden layers, 200 hidden neurons and RELU activations. Based on the idea that optimal policies can be expressed as a collection of piece-wise linear functions, we opt for RELU activations. Neural network output goes into TANH activation and multiplied with $a_{\max}$.
- **Critic:** 2-hidden layers, 200 hidden neurons and TANH activations. Since the state-value functions must be smooth, TANH activation is more suitable compared to other activations. We empirically observed critic networks with RELU activations easily explode outside the training data, which deteriorates the learning.

S2.6. Additional Results

Predictive dynamic errors on shorter sequences are illustrated in Figure S2. We see that future MSEs are much lower compared to $H = 2$ while they still cannot directly predict overall model performance $V(\mathbf{s}_0)$

S3. ODE Solver Comparison

In this ablation study, we ask two questions in relation with the simulation environment and numerical integration: *(i)* Which numerical ODE solver one should use, *(ii)* To what extent our continuous framework differ from its discrete counterparts? To answer, we have built a simple experiment on CartPole environment where several ODE solvers are compared: three adaptive step solvers (`dopri5` (`RK45`), `RK23` and `RK12`), five fixed step solvers (`RK4` with 1/10 intermediate steps, and `Euler` with 10/100/1000 intermediate steps), as well as discrete transitions. Due to the lack of a closed-form ODE solution, *true* ODE solutions are obtained by Runge-Kutta 7(8) solver, the numerical integrator which achieves the smallest local error to the best of our knowledge (Prince and Dormand, 1981).

Each ODE solver takes as input the same set of initial values as well as twenty different policy functions, some of which solve the problem whereas some are sub-optimal. Figure S1 demonstrates the distance between the true state solutions and those given by different ODE solvers. The most striking observation is that discrete transitions of the form $\mathbf{s}_{t+1} - \mathbf{s}_t = h \cdot \mathbf{f}(\mathbf{s}_t, \mathbf{a}_t)$ are highly erroneous. Moreover, adaptive solvers as well as fixed-step solvers with sufficiently many intermediate steps attain practically zero error. Unsurprisingly, approximate state solutions deteriorate over time since the error accumulates. In our experiments, we use `RK78` to mimic the interactions with the real world, and `dopri5` to forward simulate model dynamics.

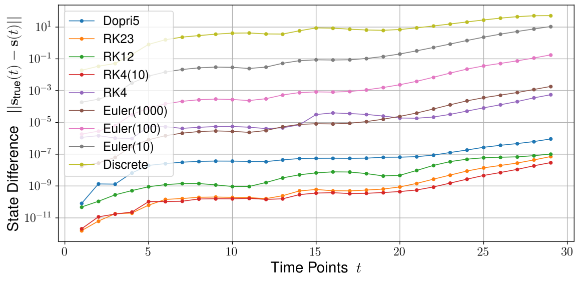

Figure S1. Error estimates of different numerical integration methods plotted against integration time.

S4. Additional Related Work

Model-based RL The majority of model-based reinforcement learning methods assumes auto-regressive transitions, which effectively learn a distribution over the *next* state given the current state and action. Unknown transitions are typically approximated by a Gaussian process (Kocijan et al., 2004; Deisenroth and Rasmussen, 2011; Kamthe and Deisenroth, 2017; Levine et al., 2011), multi-layer perceptron (MLP) (Gal et al., 2016; Depeweg et al., 2017; Nagabandi et al., 2018; Chua et al., 2018) or recurrent neural network (Ha and Schmidhuber, 2018; Hafner et al., 2018; 2020). Such models are typically developed in conjunction with model predictive control (Richards, 2005) used for planning or with a parametric policy.

Continuous-time RL In addition to the works discussed earlier (Baird, 1993; Doya, 2000), Bradtke and Duff (1994) developed Q-functions and temporal different learning in the context of semi-Markov decision processes. Abu-Khalaf and Lewis (2005) proposed a policy-iteration algorithm for the optimal control of CT systems with constrained controllers. An online version of this algorithm was derived in (Vrabie and Lewis, 2009), which was extended in a series of papers (Luo et al., 2014; Modares et al., 2016; Zhu et al., 2016; Lee and Sutton, 2019). A direct least-squares solution to Hamilton-Jacobi-Bellman equation was studied in

Table S1. Environment specifications

Environment	N_0	N_{exp}	$c_{\mathbf{p}}$	$c_{\mathbf{a}}$	a_{max}	$\mathbf{s}^{\text{box}}$	$\mathbf{s}^{\text{goal}}$	Max. execution time (h)
PENDULUM	3	-	1e-2	1e-2	2	$[\pi, 3]$	$[0, \ell]$	12
CARTPOLE	5	2	1e-2	1e-2	3	$[0.05, 0.05, 0.05, 0.05]$	$[0, 0, \ell]$	24
ACROBOT	7	3	1e-4	1e-2	4	$[0.1, 0.1, 0.1, 0.1]$	$[0, 2\ell]$	24

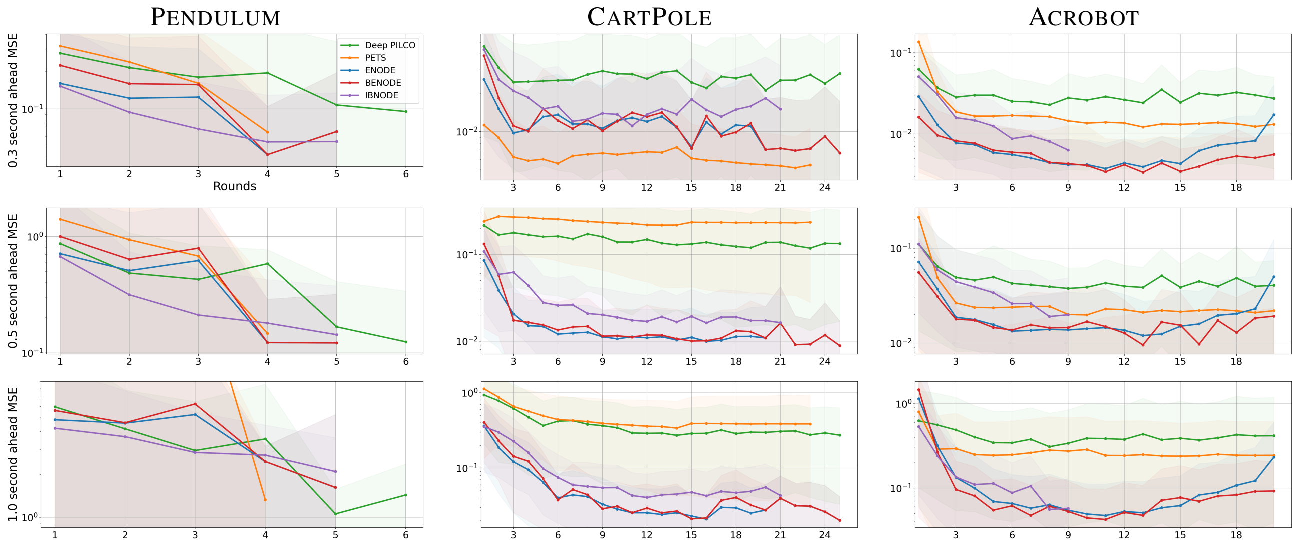

Figure S2. Predictive mean squared error of different dynamics models, computed after each round on a fixed test set.

(Tassa and Erez, 2007), requiring no forward integration for value estimation but a bag of tricks to deal with numerical instabilities. In a related work, Mehta and Meyn (2009) proposed an adaptive controller for nonlinear CT systems via a continuous-time analog of the Q-function. Above-mentioned methods are either built upon known dynamics or they are model-free. In either case, the dynamics are assumed to be linear with respect to the action, a premise needed for closed-form optimal policies.

Neural ODEs In their ground-breaking work, Chen et al. (2018) show that simple multi-layer perceptrons (MLP) can be utilized for learning arbitrary continuous-time dynamics. The resulting model, called Neural ODEs (NODEs), have been shown to outperform its RNN-based, discrete counterparts in interpolation and long-term prediction tasks (Chen et al., 2018). The vanilla NODE model paved the way for advances in continuous-time modeling, such as second-order latent ODE models (Yildiz et al., 2019), augmented systems (Dupont et al., 2019), stochastic differential equations (Jia and Benson, 2019; Li et al., 2020), and so forth. NODE framework also allows encoding prior knowledge about the observed phenomena on the network topology, which leads to Hamiltonian and Lagrangian neural networks that are capable of long-term extrapolations, even when trained from images (Greydanus et al., 2019; Cranmer et al., 2020).

Neural CTRL The NODE breakthrough has opened a new research avenue in CTRL. In particular, physics-informed continuous-time dynamical systems have gained popularity. For example, Lagrangian mechanics are imposed on the architecture presented in (Lutter et al., 2019), which results in near-perfect real-time control of a robot with seven degrees of freedom. Hamiltonian framework is proven useful for inferring controls from generalized coordinates and momenta (Zhong et al., 2020). Later in Zhong and Leonard (2020), an interpretable latent Lagrangian dynamical system and controller were trained from images. In addition to dynamics learning, above-mentioned methods describe model-specific recipes for learning controls.

References

Murad Abu-Khalaf and Frank Lewis. Nearly optimal control laws for nonlinear systems with saturating actuators using a neural network HJB approach. *Automatica*, 41(5):779–791, 2005.

Leemon Baird. Advantage updating. Technical report, Wright Lab, 1993.

Steven Bradtke and Michael Duff. Reinforcement learning methods for continuous-time Markov decision problems. *NIPS*, 7:393–400, 1994.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *NIPS*, pages 6571–6583, 2018.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *NIPS*, pages 4754–4765, 2018.

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. *ICLR Deep Differential Equations Workshop*, 2020.

Marc Deisenroth and Carl E Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *ICML*, pages 465–472, 2011.

Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Learning and policy search in stochastic dynamical systems with Bayesian neural networks. In *ICLR*, 2017.

Kenji Doya. Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245, 2000.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural ODEs. In *NIPS*, pages 3140–3150, 2019.

Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving PILCO with Bayesian neural network dynamics models. In *ICML Data-Efficient Machine Learning Workshop*, page 34, 2016.

Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In *NIPS*, pages 15379–15389, 2019.

David Ha and Jürgen Schmidhuber. World models. *arXiv:1803.10122*, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv:1811.04551*, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *ICLR*, 2020.

Junteng Jia and Austin R Benson. Neural jump stochastic differential equations. In *NIPS*, pages 9847–9858, 2019.

Sanket Kamthe and Marc Peter Deisenroth. Data-efficient reinforcement learning with probabilistic model predictive control. *arXiv:1706.06491*, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

Jus Kocijan, Roderick Murray-Smith, Carl Edward Rasmussen, and Agathe Girard. Gaussian process model based predictive control. In *Proceedings of the 2004 American control conference*, volume 3, pages 2214–2219. IEEE, 2004.

Jae Young Lee and Richard S Sutton. Policy iterations for reinforcement learning problems in continuous time and space: Fundamental theory and methods. *Automatica*, 2019.

Sergey Levine, Zoran Popovic, and Vladlen Koltun. Nonlinear inverse reinforcement learning with Gaussian processes. In *NIPS*, pages 19–27, 2011.

Xuechen Li, Ting-Kam Leonard Wong, Ricky T. Q. Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. In *AISTATS*, pages 3870–3882, 2020.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

Biao Luo, Huai-Ning Wu, Tingwen Huang, and Derong Liu. Data-based approximate policy iteration for affine nonlinear continuous-time optimal control design. *Automatica*, 50(12):3281–3290, 2014.

Michael Lutter, Christian Ritter, and Jan Peters. Deep Lagrangian networks: Using physics as model prior for deep learning. In *ICLR*, 2019.

Prashant Mehta and Sean Meyn. Q-learning and Pontryagin's minimum principle. In *IEEE Conference on Decision and Control*, pages 3598–3605, 2009.

Hamidreza Modares, Frank L Lewis, and Zhong-Ping Jiang. Optimal output-feedback control of unknown continuous-time linear systems using off-policy reinforcement learning. *IEEE transactions on cybernetics*, 46(11):2401–2410, 2016.

Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

Peter J Prince and John R Dormand. High order embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 7(1):67–75, 1981.

Arthur George Richards. *Robust constrained model predictive control*. PhD thesis, Massachusetts Institute of Technology, 2005.

Yuval Tassa and Tom Erez. Least squares solutions of the HJB equation with neural network value-function approximators. *IEEE transactions on neural networks*, 18(4): 1031–1041, 2007.

Kyriakos G Vamvoudakis and Frank L Lewis. Online actor–critic algorithm to solve the continuous-time infinite horizon optimal control problem. *Automatica*, 46(5):878–888, 2010.

Draguna Vrabie and Frank Lewis. Neural network approach to continuous-time direct adaptive optimal control for partially unknown nonlinear systems. *Neural Networks*, 22(3):237–246, 2009.

Cagatay Yildiz, Markus Heinonen, and Harri Lahdesmaki. ODE2VAE: Deep generative second order ODEs with Bayesian neural networks. In *NIPS*, pages 13412–13421, 2019.

Yaofeng Desmond Zhong and Naomi Leonard. Unsupervised learning of Lagrangian dynamics from images for prediction and control. In *NIPS*, 2020.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ODE-net: Learning Hamiltonian dynamics with control. In *ICLR*, 2020.

Yuanheng Zhu, Dongbin Zhao, and Xiangjun Li. Using reinforcement learning techniques to solve continuous-time non-linear optimal tracking problem without system dynamics. *IET Control Theory & Applications*, 10(12): 1339–1347, 2016.