UC Santa Cruz

UC Santa Cruz Previously Published Works

Title

Algorithms of data generation for deep learning and feedback design: A survey

Permalink

https://escholarship.org/uc/item/83t5r5nf

Authors

Kang, Wei
Gong, Qi
Nakamura-Zimmerer, Tenavi
et al.

Publication Date

2021-11-01

DOI

10.1016/j.physd.2021.132955

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Algorithms of Data Generation For Deep Learning and Feedback Design *

Wei Kang† Qi Gong‡ Tenavi Nakamura-Zimmerer§ Fariba Fahroo, ¶

April 18, 2021

Abstract

Recent research reveals that deep learning is an effective way of solving high dimensional Hamilton-Jacobi-Bellman equations. The resulting feedback control law in the form of a neural network is computationally efficient for real-time applications of optimal control. A critical part of this design method is to generate data for training the neural network and validating its accuracy. In this paper, we provide a survey of existing algorithms that can be used to generate data. All the algorithms surveyed in this paper are causality-free, i.e., the solution at a point is computed without using the value of the function at any other points. An illustrative example is given for the optimal feedback design using supervised learning in which the data is generated using causality-free algorithms.

1 Introduction

A critical part in feedback controllers of dynamical systems is a mathematical feedback law, which is a function whose output is the control value and its input is the value of sensory information or estimated states. The control law is designed to meet performance requirements based on a given system model. The performance of feedback controls includes, but not limited to, minimizing a cost function, stabilization, tracking, synchronization, etc. After decades of active research, tremendous progress has been made in the field of control theory that has a huge literature of feedback design methodologies. For linear systems, there are a well developed theory and commercially available computational tools, such as MATLAB toolboxes, that help to implement the linear control theory in practical applications. For nonlinear control systems, rigorous theory and design methods have been studied and developed. However, their applications to many real-life problems have been limited by some fundamental challenges. Lacking effective computational algorithms for nonlinear feedback design is one of the main bottlenecks. For instance, ideally one can solve the Hamilton-Jacobi-Bellman (HJB) equation that leads to a simple feedback law of optimal control. However,

*This work was supported in part by U.S. Naval Research Laboratory - Monterey, CA

†Department of Applied Mathematics, Naval Postgraduate School, Monterey, CA, USA; wkang@nps.edu

‡Department of Applied Mathematics,University of California at Santa Cruz, Santa Cruz, CA, USA

§Department of Applied Mathematics,University of California at Santa Cruz, Santa Cruz, CA, USA

¶Air Force Office of Scientific Research, Arlington, Virginia, USA

solving the HJB equation is a problem that suffers the curse-of-dimensionality. For systems with even a moderate dimension such as $n \geq 4$, finding a numerical solution for the HJB equation is extremely difficult, if not impossible. The curse-of-dimensionality affects many areas of nonlinear feedback design, such as differential games, reachable sets, the PDE (or FBI equation) for output regulation, stochastic control, etc.

Some recent publications reveal that neural networks can be used as an effective tool to overcome the curse-of-dimensionality. In [20, 32, 23], neural network approximations of optimal controls are found for examples with high dimensions from $n = 6$ to $n = 100$. In these examples, a large number of numerical data is generated using computational algorithms and the system model. Then a neural network is trained by minimizing a loss function. These examples show that a neural network tends to be less sensitive to the increase of the dimension, but more dependent on the quality of the data. The methods in [20, 32, 23] share something in common: They are all model-based and data-driven, i.e., the control law is designed through the training of a neural network based on data that is generated using a numerical model. After decades of research, a large variety of computational methods have been developed for optimal control. These methods are buried in a huge number of research papers. Some of them are well-known only to some specific communities. In this paper, we give a survey of some existing algorithms of open-loop optimal control that either have been applied or have a great potential to be applied for the purpose of generating data for deep learning.

Optimal control has many different formulations, finite/infinite horizon, fixed/free terminal time, unconstrained/control-constrained/mixed state-control constrained, etc. A computational method that is appropriate for a certain type of problems may face challenge in solving other types. This paper not only surveys computational methods but also highlights their special properties (as shown in Table 1). Such result helps the interested researchers and practitioners to identify appropriate numerical methods for their problems. The survey does not include those results, such as [37] for output regulation, [35] for general PDEs and [3] for multiscale stochastic systems, that are not focused on optimal control although they share similar ideas of deep learning for dynamic systems.

For background information, we briefly outline the process in [32] of training a neural network to approximate an optimal control. Similar steps are followed in several other papers mentioned above. Consider the following problem

$$\begin{cases} \underset{\boldsymbol{u}\in\mathcal{U}}{\text{minimize}} & \displaystyle\int_{t_0}^{t_f} L(t,\boldsymbol{x},\boldsymbol{u})dt + \psi(\boldsymbol{x}(t_f)), \\ \text{subject to} & \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t,\boldsymbol{x},\boldsymbol{u}), \\ & \boldsymbol{x}(t_0) = \boldsymbol{x}_0. \end{cases} \tag{1}$$

Here $\boldsymbol{x}(t) : [t_0, t_f] \to \mathcal{X} \subseteq \mathbb{R}^n$ is the state, $\boldsymbol{u}(t,\boldsymbol{x}) : [0, t_f] \times \mathcal{X} \to \mathcal{U} \subseteq \mathbb{R}^m$ is the feedback control to be designed, $\boldsymbol{f}(t,\boldsymbol{x},\boldsymbol{u}) : [0, t_f] \times \mathcal{X} \times \mathcal{U} \to \mathbb{R}^n$ is a Lipschitz continuous vector field, $\psi(\boldsymbol{x}(t_f)) : \mathcal{X} \to \mathbb{R}$ is the terminal cost, and $L(t,\boldsymbol{x},\boldsymbol{u}) : [0, t_f] \times \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is the running cost, or the Lagrangian. The optimal cost, as a function of $(t_0, \boldsymbol{x}_0)$, is called the value function, which is denoted by $V(t_0, \boldsymbol{x}_0)$ or simply $V(t,\boldsymbol{x})$. The following approach is from [32]. An illustrative example is given in Section 2

1. *Initial data generation:* For supervised learning, a data set must be generated. It contains the value of $V(t,\boldsymbol{x})$ at random points in a given region. A key feature desired for the

algorithm is that the computation should be *causality-free*, i.e. the solution of $V(t_0, \boldsymbol{x}_0)$ is computed without using an approximated value of $V(t, \boldsymbol{x})$ at nearby points. For instance, finite difference methods for solving PDEs are not causality-free (in space) because the solution is propagated over a set of grid points. The Causality-free property is important for several reasons: (1) the algorithm does not require a grid so that the computation can be applied to high dimensional problems; (2) data can be generated at targeted region for adaptive data generation; (3) the accuracy of the trained neural network can be checked empirically in a selected region; (4) data can be generated in parallel in a straightforward manner.

2. *Training:* Given this data set, a neural network is trained to approximate the value function $V(t, \boldsymbol{x})$. The accuracy of the neural network can be empirically checked using a new data set. If necessary, an adaptive deep learning loop can be applied. In each round, one can check the approximation error and then expand the data set in regions where the value function is likely to be steep or complicated, and thus difficult to learn.

3. *Validation:* The training process stops when it satisfies the convergence criteria. Then, the accuracy of the trained neural network is checked on a new set of validation data computed at Monte Carlo sample points. Once again, a causality-free algorithm is needed here.

4. *Feedback control:* Causality-free algorithms may not be fast or reliable enough for real-time feedback control. However, one can compute the optimal feedback control online by evaluating the gradient of the trained neural network and applying Pontryagin's maximum principle. Notably, evaluation of the gradient is computationally cheap even for large n, enabling implementation in high-dimensional systems.

Physics laws and first principle models are fundamental and critical in control system designs. Guaranteed system properties by physics laws and mathematics analysis are invaluable. These properties should be carried through the design, rather than reinventing the wheel by machine learning. In a model-based data-driven approached outlined in Steps 1 - 4, we take the advantage of existing models and design methodologies that have been developed for decades, many with guaranteed performance. The deep neural network is used focusing on the curse-of-dimensionality only, an obstacle that classical analysis or existing numerical methods have failed to overcome. Control systems designed in this way should have the performance as proved in classical and modern control theory, however be curse-of-dimensionality free. Demonstrated in [32, 20, 23], some advantages of the model-based data-driven approach include: the optimal feedback can be learned from data over given semi-global domains, rather than a local neighborhood of an equilibrium point; the level of accuracy of the optimal control and value function can be empirically validated; generating data using causality-free algorithms has perfect parallelism; the inherent capacity of neural networks for dealing with high-dimensional problems makes it possible to solve HJB equations that have high dimensions.

Generating data is critical in three of the four steps shown above. A reliable, accurate and causality-free algorithm to compute $V(t, \boldsymbol{x})$, and $V_{\boldsymbol{x}}(t, \boldsymbol{x})$ in some cases, is required. Some computational algorithms for open-loop optimal control are suitable for this task. The goal of this paper is to provide a survey of some representative algorithms that have been, or have the potential to be, used for data generation (Sections 3 to 6). In the next section, an example is given to illustrate the key steps outlined above.

2 An example of optimal attitude control

This is an example from [32] in which a neural network is trained using adaptive data generation for the purpose of optimal attitude control of rigid body. The actuators are three pairs of momentum wheels. The state variable is $\boldsymbol{x} = \begin{pmatrix} \boldsymbol{v} & \boldsymbol{\omega} \end{pmatrix}$. Here $\boldsymbol{v}$ represents the Euler angles. Following the definition in [9],

$$\boldsymbol{v} = \begin{pmatrix} \phi & \theta & \psi \end{pmatrix}^T,$$

in which ϕ, θ, and ψ are the angles of rotation around a body frame $\boldsymbol{e}_1'$, $\boldsymbol{e}_2'$, and $\boldsymbol{e}_3'$, respectively, in the order $(1, 2, 3)$. These are also commonly called roll, pitch, and yaw. The other state variable is $\boldsymbol{\omega}$, which denotes the angular velocity in the body frame,

$$\boldsymbol{\omega} = \begin{pmatrix} \omega_1 & \omega_2 & \omega_3 \end{pmatrix}^T.$$

The state dynamics are

$$\begin{pmatrix} \dot{\boldsymbol{v}} \\ \boldsymbol{J}\dot{\boldsymbol{\omega}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{E}(\boldsymbol{v})\boldsymbol{\omega} \\ \boldsymbol{S}(\boldsymbol{\omega})\boldsymbol{R}(\boldsymbol{v})\boldsymbol{h} + \boldsymbol{B}\boldsymbol{u} \end{pmatrix}.$$

Here $\boldsymbol{E}(\boldsymbol{v}), \boldsymbol{S}(\boldsymbol{\omega}), \boldsymbol{R}(\boldsymbol{v}) : \mathbb{R}^3 \to \mathbb{R}^{3\times 3}$ are matrix-valued functions defined as

$$\boldsymbol{E}(\boldsymbol{v}) := \begin{pmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{pmatrix}, \qquad \boldsymbol{S}(\boldsymbol{\omega}) := \begin{pmatrix} 0 & \omega_3 & -\omega_2 \\ -\omega_3 & 0 & \omega_1 \\ \omega_2 & -\omega_1 & 0 \end{pmatrix},$$

and

$$\boldsymbol{R}(\boldsymbol{v}) := \begin{pmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\theta\sin\phi \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\theta\cos\phi \end{pmatrix}.$$

Further, $\boldsymbol{J} \in \mathbb{R}^{3\times 3}$ is a combination of the inertia matrices of the momentum wheels and the rigid body without wheels, $\boldsymbol{h} \in \mathbb{R}^3$ is the total constant angular momentum of the system, and $\boldsymbol{B} \in \mathbb{R}^{3\times m}$ is a constant matrix where m is the number of momentum wheels. To control the system, we apply a torque $\boldsymbol{u}(t, \boldsymbol{v}, \boldsymbol{\omega}) : [0, t_f] \times \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^m$. In this example, $m = 3$. Let

$$\boldsymbol{B} = \begin{pmatrix} 1 & 1/20 & 1/10 \\ 1/15 & 1 & 1/10 \\ 1/10 & 1/15 & 1 \end{pmatrix}, \qquad \boldsymbol{J} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix}, \qquad \boldsymbol{h} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

The optimal control problem is

$$\begin{cases} \underset{\boldsymbol{u}(\cdot)}{\text{minimize}} & \displaystyle\int_t^{t_f} L(\boldsymbol{v}, \boldsymbol{\omega}, \boldsymbol{u}) d\tau + \frac{W_4}{2}\|\boldsymbol{v}(t_f)\|^2 + \frac{W_5}{2}\|\boldsymbol{\omega}(t_f)\|^2, \\ \text{subject to} & \dot{\boldsymbol{v}} = \boldsymbol{E}(\boldsymbol{v})\boldsymbol{\omega}, \\ & \boldsymbol{J}\dot{\boldsymbol{\omega}} = \boldsymbol{S}(\boldsymbol{\omega})\boldsymbol{R}(\boldsymbol{v})\boldsymbol{h} + \boldsymbol{B}\boldsymbol{u}. \end{cases} \tag{2}$$

Here

$$L(\boldsymbol{v}, \boldsymbol{\omega}, \boldsymbol{u}) = \frac{W_1}{2}\|\boldsymbol{v}\|^2 + \frac{W_2}{2}\|\boldsymbol{\omega}\|^2 + \frac{W_3}{2}\|\boldsymbol{u}\|^2,$$

and

$$W_1 = 1, \quad W_2 = 10, \quad W_3 = \frac{1}{2}, \quad W_4 = 1, \quad W_5 = 1, \quad t_f = 20.$$

The HJB equation associated with the optimal control has $n = 6$ state variables and $m = 3$ control variables. Solving the HJB equation using any numerical algorithm based on dense grids in state space is intractable because the size of the grid increases at the rate of N^6, where N is the number of grid points in each dimension. In [27], a time-marching TPBVP solver is applied to compute the optimal control on a set of sparse gridpoints. In [32], this idea is adopted to generate an initial data set from the domain

$$\mathcal{X}_0 = \left\{ \boldsymbol{v}, \boldsymbol{\omega} \in \mathbb{R}^3 \middle| -\frac{\pi}{3} \leq \phi, \theta, \psi \leq \frac{\pi}{3} \text{ and } -\frac{\pi}{4} \leq \omega_1, \omega_2, \omega_3 \leq \frac{\pi}{4} \right\},$$

This is a small data set with $N_d = 64$ randomly selected initial states

$$\boldsymbol{x}^{(i)} = (\boldsymbol{v}^{(i)}, \boldsymbol{\omega}^{(i)}) \text{ for } i = 1, 2, \cdots, N_d.$$

Based on the data, a neural network implemented in TensorFlow [1] is trained to approximate the value function, $V(t, \boldsymbol{x})$, at $t = 0$. The neural network has three hidden layers with 64 neurons in each. The optimization is achieved using the SciPy interface for the L-BFGS optimizer [4, 24]. The loss function has two parts,

$$\mathcal{L} = \frac{1}{N_d} \sum_{i=1}^{N_d} \left[V^{(i)} - V^{NN}(t^{(i)}, \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \right]^2 + \frac{\mu}{N_d} \sum_{i=1}^{N_d} \left\| \boldsymbol{\lambda}^{(i)} - V_{\boldsymbol{x}}^{NN}(t^{(i)}, \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \right\|^2,$$

in which μ is a scalar weight. The optimization variable is $\boldsymbol{\theta}$, the parameter in the neural network. The first part in the loss function penalizes the error of the neural network and the second part penalizes the error of its gradient. The loss function defined in this way takes the advantage of the fact that a TPBVP solver finds both the value of $V(t, \boldsymbol{x})$ and the costate, which equals the gradient of the value function. Due to the small size of the data set, $V^{NN}(t, \boldsymbol{x})$ is an inaccurate approximation of the value function. However, it is often good enough to serve as the initial guess for the TPBVP solver. As a result, new data can be generated significantly faster than using time-marching. This makes adaptive data generation possible. After each training round, the location and number of additional data points are determined following a set of formulae [32]. Then a new data set is generated using a neural network warm start. In this example, a total of seven training rounds are carried out with the final data set containing $N_d = 2110$ samples [32].

The accuracy of the neural network over the course of training is shown in Figure 1. As described in Section 1, accuracy is measured on an *independently generated* set of $N_{d,val} = 1000$ data points. As noted previously, the ability to validate the error in this way is a key advantage of data-driven approaches. Two error metrics are reported. First, the relative mean absolute error (RMAE) of value function prediction, which is defined as

$$\text{RMAE}(\boldsymbol{\theta}) \coloneqq \frac{\sum_{i=1}^{N_{d,val}} \left| V^{(i)} - V^{NN}\left(t^{(i)}, \boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right) \right|}{\sum_{i=1}^{N_{d,val}} \left| V^{(i)} \right|}. \tag{3}$$

Second, the relative mean L^2 error (RML^2) of gradient prediction, which is defined as

$$\text{RM}L^2(\boldsymbol{\theta}) \coloneqq \frac{\sum_{i=1}^{N_{d,val}} \left\| \boldsymbol{\lambda}^{(i)} - V_{\boldsymbol{x}}^{NN}\left(t^{(i)}, \boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right) \right\|_2}{\sum_{i=1}^{N_{d,val}} \left\| \boldsymbol{\lambda}^{(i)} \right\|_2}. \tag{4}$$

Compared to pointwise relative errors, these metrics emphasize predictive accuracy in regions where a lot of control effort is needed. This is important when we are interested in designing nonlinear controllers which are effective and efficient on large domains.

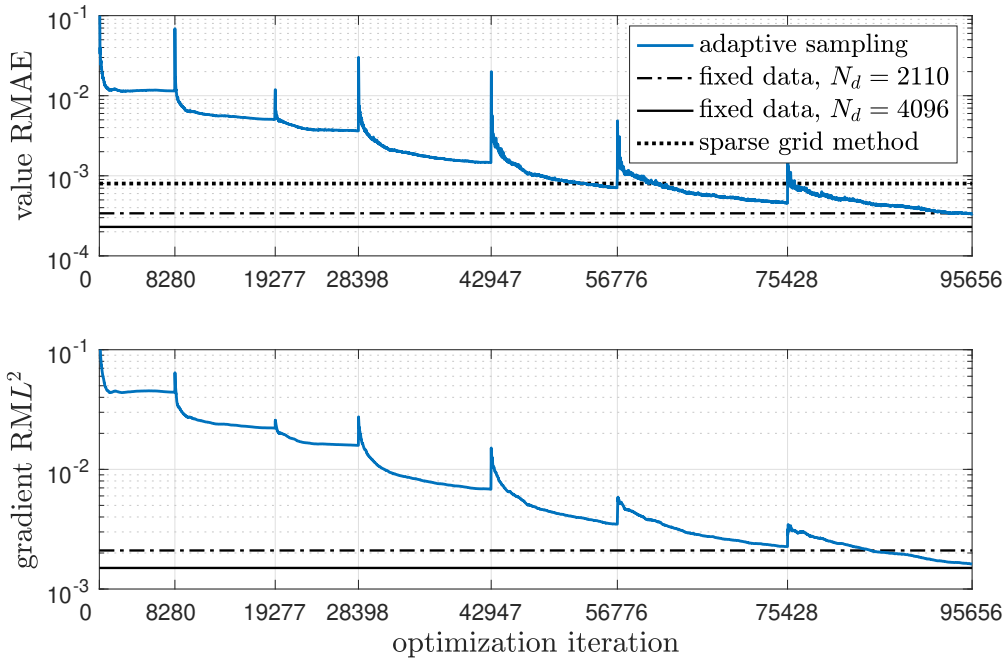

Figure 1: Progress of adaptive sampling and model refinement for the rigid body problem (2), compared to training on fixed data sets and the sparse grid characteristics method. Spikes in the error correspond to the start of new training rounds and expansion of the training data set. Figure from [32].

3 Characteristic methods

In this section, we review a set of causality-free algorithms that are based on characteristic methods. They are used in [32, 23] as the tool of generating data for supervised learning. The survey also includes some other data generating algorithms such as minimization-based algorithms (Section 4), methods for stochastic systems (Section 6), and direct methods for optimal control with constraints (Section 5). Consider the problem of optimal control defined in (1). Let's define the Hamiltonian

$$H(t, \boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{u}) = L(t, \boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{\lambda}^T \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u}), \tag{5}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the state of the control system,

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u}),$$

in which $\boldsymbol{u} \in \mathcal{U} \subseteq \mathbb{R}^m$ is the control variable. In (5), $\boldsymbol{\lambda} \in \mathbb{R}^n$ is the costate and $L(t, \boldsymbol{x}, \boldsymbol{u})$ is the Lagrangian of optimal control. The HJB equation is

$$\begin{cases} V_t(t, \boldsymbol{x}) + \min\limits_{\boldsymbol{u} \in \mathcal{U}} \{L(t, \boldsymbol{x}, \boldsymbol{u}) + V_{\boldsymbol{x}}^T(t, \boldsymbol{x}) \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u})\} = 0, \\ V(t_f, \boldsymbol{x}) = \psi(\boldsymbol{x}), \end{cases} \tag{6}$$

where $\psi(\boldsymbol{x})$ represents the endpoint cost. The optimal feedback control law is

$$(t, \boldsymbol{x}) \to \boldsymbol{u}^*(t, \boldsymbol{x}, V_{\boldsymbol{x}}) = \arg\min_{\boldsymbol{u}\in\mathcal{U}} H\left(t, \boldsymbol{x}, V_{\boldsymbol{x}}, \boldsymbol{u}\right). \tag{7}$$

If we assume $\boldsymbol{\lambda} = V_{\boldsymbol{x}}$, the characteristics of the HJB equation follows Pontryagin's maximum principle (PMP)

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \dfrac{\partial H}{\partial \boldsymbol{\lambda}} = \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u}^*(t, \boldsymbol{x}, \boldsymbol{\lambda})), & \boldsymbol{x}(0) = \boldsymbol{x}_0, \\ \dot{\boldsymbol{\lambda}}(t) = -\dfrac{\partial H}{\partial \boldsymbol{x}}(t, \boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{u}^*(t, \boldsymbol{x}, \boldsymbol{\lambda})), & \boldsymbol{\lambda}(t_f) = \dfrac{\partial \psi}{\partial \boldsymbol{x}}(t_f), \\ \dot{v}(t) = -L(t, \boldsymbol{x}, \boldsymbol{u}^*(t, \boldsymbol{x}, \boldsymbol{\lambda})), & v(t_f) = \psi(\boldsymbol{x}(t_f)). \end{cases} \tag{8}$$

It is a two-point boundary value problem (TPBVP). Computational algorithms of solving TPBVPs have been studied for decades with an extensive literature. Solvers exist in various programming languages and computing platforms such as MATLAB and Python. For instance, bvp5c is a MATLAB boundary value problem solver that controls both the residual and error [30]. In bvp5c, the differential equation is discretized using the four-point Lobatto IIIa formula, which is also stated as an implicit Runge-Kutta formula that has the following Butcher-array ((3.3.21) in [8])

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{5-\sqrt{5}}{10} & \frac{11+\sqrt{5}}{120} & \frac{25-\sqrt{5}}{120} & \frac{25-13\sqrt{5}}{120} & \frac{-1+\sqrt{5}}{120} \\ \frac{5+\sqrt{5}}{10} & \frac{11-\sqrt{5}}{120} & \frac{25+13\sqrt{5}}{120} & \frac{25+\sqrt{5}}{120} & \frac{-1-\sqrt{5}}{120} \\ 1 & \frac{1}{12} & \frac{5}{12} & \frac{5}{12} & \frac{1}{12} \\ \hline & \frac{1}{12} & \frac{5}{12} & \frac{5}{12} & \frac{1}{12} \end{array}$$

The trajectory is evaluated at a sequence of grid points $t_0 = 0 < t_1 < \cdots < t_N = t_f$. If t is not a grid point, bvp5c approximates the solution using a continuous extension, which is a function that interpolates the trajectory and its slope at t_i, $0 \le i \le N$, and the midpoints of subintervals. A relation between the residual and the true error is established for bvp5c in [30]. Solving an implicit Runge-Kutta discretization of boundary value problems requires numerically solving algebraic equations. Most solvers of boundary value problems have algebraic equation algorithms integrated in them, such as bvp5c used for the examples in [27, 28] and the SciPy implementation of bvp4c used in [32]. A challenge in all these examples is that boundary value problem solvers are sensitivity to initial guess. This is a main issue to be addressed in Sections 3.1 and 3.2.

Remark 1 In (8), the TPBVP requires the minimization of the Hamiltonian, i.e., a solution to (7). For some problems whose Hamiltonian is a quadratic polynomial of u, the minimization can be explicitly solved. This is the case for the attitude control problem shown in Section 2. If (7) does not have an explicit solution, $u^*(t, \boldsymbol{x}, \boldsymbol{\lambda})$ in the ODEs in (8) has to be evaluated using a numerical algorithm of minimization, such as Newton's method. In this case, the computational load is increased. Another approach that we recommend is to use a direct method, which is introduced in Section 5. Using a direct method, one applies nonlinear programming to a discretized optimization problem in which a solution to (7) is not necessary.

3.1 Time-marching

TPBVPs have been perceived as very difficult because their numerical algorithms tend to diverge. The single most important factor that affects the convergence is the initial guess. In a worst scenario, the TPBVP solver in [32] converges at only 1% of the sample pints. However, applying a time-marching method, the convergence is improved to 98%. Applying more sophisticated tuning of marching steps in [27, 28], 100% convergence was achieved at more than $40,000$ grid points.

In the time-marching trick, a sequence of solutions is computed that grows from an initially short time interval. More specifically, we choose a time sequence,

$$t_0 < t_1 < t_2 < \cdots < t_K = t_f,$$

in which t_1 is small. For the short time interval $[t_0, t_1]$, the TPBVP solver always converges using an initial guess close to the initial state. Then the resulting trajectory, $(\boldsymbol{x}^1(t), \boldsymbol{\lambda}^1(t))$ is extended over the longer time interval $[t_0, t_2]$. A simple way to extend the trajectory is with a piecewise function

$$\boldsymbol{x}_0^2(t) = \begin{cases} \boldsymbol{x}^1(t), & \text{if } t_0 \le t \le t_1, \\ \boldsymbol{x}^1(t_1), & \text{if } t_1 < t \le t_2, \end{cases}$$

and $\boldsymbol{\lambda}_0^2(t)$ is similarly defined. Or one can try a linear extension

$$\boldsymbol{x}_0^2(t) = \boldsymbol{x}^1\left(t_0 + \frac{t_1 - t_0}{t_2 - t_0}(t - t_0)\right), \quad \text{for } t_0 \le t \le t_2.$$

The trajectory over the extended interval is used as an initial guess to find $(\boldsymbol{x}^2(t), \boldsymbol{\lambda}^2(t))$, a solution of the TPBVP over $[0, t_2]$. Repeating this process until $t_K = t_f$ at which we obtain the full solution. One needs to tune the time sequence $\{t_k\}_{k=1}^K$ to achieve convergence while maintaining acceptable efficiency. The time-marching approach does not require a good initial guess. When converges, the TPBVP solver achieves highly accurate solutions. The algorithm in [30] even provides an estimated error. However, this method is usually slower than the neural network warm start, which is illustrated next.

3.2 Neural network warm start

In [32], a neural network warm start is used to speed up the computation and to improve the convergence. Before a neural network can be trained, we need an initial set of data. This can be generated using the time-marching method in Section 3.1. Then we train a neural network based on the initial data set. The loss function used in [32] takes into consideration both the value functions, $V(t, \boldsymbol{x})$, and the costate, $\boldsymbol{\lambda}(t)$. Suppose

$$V^{(i)} = V(t^{(i)}, \boldsymbol{x}^{(i)})$$

be the optimal cost, i.e. the value function, at sampling points $(t^{(i)}, \boldsymbol{x}^{(i)})$ for $i = 1, 2, \cdots, N_d$. If the value function is evaluated by solving the TPBVP (8), as a byproduct the costate is also known,

$$\boldsymbol{\lambda}^{(i)} = \boldsymbol{\lambda}(t^{(i)}, \boldsymbol{x}^{(i)}),$$

where $\boldsymbol{\lambda}(t, \boldsymbol{x})$ represents the value of costate in the solution of TPBVP with initial time t and initial state $\boldsymbol{x}$. Then we use a neural network to approximate $V(t, \boldsymbol{x})$ by minimizing the following loss function,

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N_d}\left(\sum_{i=1}^{N_d}\left[V^{(i)} - V^{NN}(t^{(i)}, \boldsymbol{x}^{(i)}; \boldsymbol{\theta})\right]^2 + \mu \sum_{i=1}^{N_d}\left\|\boldsymbol{\lambda}^{(i)} - V_{\boldsymbol{x}}^{NN}(t^{(i)}, \boldsymbol{x}^{(i)}; \boldsymbol{\theta})\right\|^2\right),$$

where $\boldsymbol{\theta}$ is the parameter of the neural network, μ is a parameter weighing between the losses of the value function and the costate. The trained neural network, $V^{NN}(t, \boldsymbol{x})$, provides the costate at any given point $(t, \boldsymbol{x})$, which is $V_{\boldsymbol{x}}^{NN}(t, \boldsymbol{x})$. If the size of an initial data set is small, then the neural network approximation is not necessarily accurate. However, it is good enough for the purpose of generating initial guess at any given $(t, \boldsymbol{x})$ to warm start the TPBVP solver so that additional data can be generated at a much faster rate. In [32], neural network warm start exceeds 99% convergence for the rigid body optimal control problem. The initial data used to train the neural network is very small, $N_d = 64$.

3.3 Backward propagation

As its name indicates, the basic idea of backward propagation is to integrate the ODEs in (8) backward in time. In the first step, a solution of TPBVP is solved for a nominal trajectory using a nominal initial state $\boldsymbol{x}_0$. The second step is perturbing the final state and costate around the nominal trajectory, subject to the terminal condition. Then the ODEs in (8) are solved backward in time to propagate a trajectory using the perturbed final state and costate value. In this way, a data set consisting of trajectories around the nominal trajectory is generated. Then, a neural network is trained by minimizing a loss function. In this approach, one avoids solving TPBVP repeatedly. Instead, the data set is generated by integrating differential equations, a task much easier than solving a TPBVP. However, the location of the sample states cannot be fully controlled. Along unstable trajectories (backward in time), integrating the ODE over a relatively long time interval can be numerically challenging.

Backward propagation is used in [23] for the optimal control of spacecraft making interplanetary transfers. The optimal control policy is computed for a spacecraft equipped with nuclear electric propulsion system. The goal is to transfer from the Earth to Venus orbit within about 1.376 years. After generating 45×10^6 data samples, neural networks consisting multiple layers are used to approximate the control policy as well as the value function. Then the accuracy is validated, once again, using data generated by backward propagation.

4 Minimization-based methods - unconstrained optimization

Optimal control is essentially a problem of minimization (or maximization) subject to the constraint of a control system. There exist various ways of transforming the problem into unconstrained optimization. Without constraints, it is relatively easy to find its numerically solution. The methods in this section are different from the direct method illustrated in Section 5 where the problem consists of algebraic constraints in addition to the control system; and the problem is transformed into optimization with constraints, which is solved using nonlinear programming.

4.1 The Hopf formula

Consider a HJ PDE,

$$\begin{cases} V_t(t,\boldsymbol{x}) + \tilde{H}(V_{\boldsymbol{x}}(t,\boldsymbol{x})) = 0, & \text{in } (0,\infty)\times\mathbb{R}^n, \\ V(0,\boldsymbol{x}) = \psi(\boldsymbol{x}), & \boldsymbol{x}\in\mathbb{R}^n, \end{cases} \tag{9}$$

where $\tilde{H}:\mathbb{R}^n\to\mathbb{R}$ is continuous and bounded from below by an affine function, $\psi:\mathbb{R}^n\to\mathbb{R}$ is convex. This equation is associate with a special family of optimal control problems. In [7], control systems in the following form are considered,

$$\begin{aligned} &\dot{\boldsymbol{x}}(s) = f(\boldsymbol{u}(s)), \quad \boldsymbol{x}\in\mathbb{R}^n, \\ &\boldsymbol{x}(t) = \boldsymbol{x}, \end{aligned} \tag{10}$$

where $\boldsymbol{u}:(-\infty,t_f]\to\mathcal{U}\subset\mathbb{R}^n$ is the control input and $\mathcal{U}$ is a compact set. The cost function is

$$J(\boldsymbol{x},t;\boldsymbol{u}) = \int_t^{t_f} L(\boldsymbol{u}(s))ds + \psi(\boldsymbol{x}(t_f)), \tag{11}$$

where $L(\boldsymbol{u})$ and $\psi(\boldsymbol{x})$ are both scalar valued functions. The goal of optimal control design is to find a feedback that minimizes $J(\boldsymbol{x},t;\boldsymbol{u})$ using admissible control inputs. Define

$$\tilde{H}(\boldsymbol{\lambda}) = \min_{\boldsymbol{u}\in\mathcal{U}} \left(L(\boldsymbol{u}) + \boldsymbol{\lambda}^T f(\boldsymbol{u})\right).$$

Then the solution of the HJ PDE (9) defines an optimal feedback

$$(t,\boldsymbol{x}) \to \boldsymbol{u}^*(t,\boldsymbol{x},V_{\boldsymbol{x}}) = \arg\min_{\boldsymbol{u}\in\mathcal{U}} \left(L(\boldsymbol{u}) + V_{\boldsymbol{x}}^T f(\boldsymbol{u})\right).$$

The solution of the HJ equation (9) can be expressed as follows (the Hopf formula [22]),

$$V(t,\boldsymbol{x}) = (\psi^* + tH)^*(\boldsymbol{x}), \tag{12}$$

where the superscript '$*$' represents the Fenchel-Legendre transform. Specifically, $f^*:\mathbb{R}^n\to\mathbb{R}\cup\{\infty\}$ of a function (convex, proper, lower semicontinuous) $f:\mathbb{R}^n\to\mathbb{R}\cup\{\infty\}$ is defined by

$$f^*(\boldsymbol{z}) = \sup_{\boldsymbol{x}\in\mathbb{R}^n}\{\boldsymbol{x}^T\boldsymbol{z} - f(\boldsymbol{x})\}.$$

The solution given in (12) is causality-free. It can be computed by solving the minimization problem

$$V(t,\boldsymbol{x}) = -\min_v\{\psi^*(\boldsymbol{v}) + tH(\boldsymbol{v}) - \boldsymbol{x}^T\boldsymbol{v}\}.$$

In [7], it is solved by using the split Bregman iterative approach. Within each iteration, two minimization problems are solved numerically where Newton's method is applicable under some smoothness assumptions. In addition to optimal control, the level set method is also addressed in [7] for the viscosity solution of the eikonal equation.

4.2 Minimization along characteristics

In this approach, the value function is computed by minimizing the cost along trajectories of the Hamiltonian system (8). Rather than a TPBVP, consider the initial value problem

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \dfrac{\partial H}{\partial \boldsymbol{\lambda}} = \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u}^*(t, \boldsymbol{x}, \boldsymbol{\lambda})), & \boldsymbol{x}(t_0) = \boldsymbol{x}_0, \\ \dot{\boldsymbol{\lambda}}(t) = -\dfrac{\partial H}{\partial \boldsymbol{x}}(t, \boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{u}^*(t, \boldsymbol{x}, \boldsymbol{\lambda})), & \boldsymbol{\lambda}(t_0) = \boldsymbol{\lambda}_0, \\ \boldsymbol{u}^*(t, \boldsymbol{x}, \boldsymbol{\lambda}) = \arg\min\limits_{\boldsymbol{u} \in \mathcal{U}} H(t, \boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{u}). \end{cases} \tag{13}$$

For fixed initial state $\boldsymbol{x}_0$, the cost along a characteristic is a function of $\boldsymbol{\lambda}_0$,

$$J(t_0, \boldsymbol{x}_0, \boldsymbol{\lambda}_0) = \int_{t_0}^{t_f} L(t, \boldsymbol{x}, \boldsymbol{u}^*(t, \boldsymbol{x}, \boldsymbol{\lambda}))dt + \psi(t_f). \tag{14}$$

Then the solution, $V(t_0, \boldsymbol{x}_0)$, of the HJB equation (6) is the minimum value of (14) along trajectories satisfying (13), i.e.,

$$V(t_0, \boldsymbol{x}_0) = \min_{\boldsymbol{\lambda}_0} J(t_0, \boldsymbol{x}_0, \boldsymbol{\lambda}_0). \tag{15}$$

Different from the TPBVP in Section 3, (13) is an initial value problem. The solution is computed by solving (15), a problem of unconstrained optimization. Under convexity type of assumptions, the existence and uniqueness of solutions have been studied and proved (see, for instance, [5, 38]). Similar approaches are also applicable to the HJI equation of differential games. In numerical computation, algorithms of unconstrained optimization are applicable. For instance, Powell's algorithm is used in [38]. In [5], coordinate descent is used with multiple initial guesses to perform the optimization. Some examples in [5] show fast convergence that may justify real-time computation. If this is the case, then a neural network training becomes unnecessary. On the other hand, algorithms with guaranteed fast convergence for real-time optimization are still an open problem in general.

Different from the characteristic based approach, in [31] the problem of optimal control is discretized using finite difference. Then the resulting finite dimensional optimization with constraints is transformed to an unconstrained optimization based upon Lagrangian duality. The unconstrained optimization is then solved using a splitting algorithm. This approach can be classified as a direct method, a family of computational methods based on discretization of the original problem. Some algorithms of direct methods are discussed in Section 5.

5 Direct methods - constrained optimization using nonlinear programming

In addition to ODEs, a control system may subject to constraints in the form of algebraic equations or inequalities, such as state constraints, control saturation, or state-control mixed constraints. A family of computational methods, so called direct methods, is particularly effective for finding optimal control with constraints. The basic idea is to first discretize the control system as well as the cost, then numerically compute the optimal trajectory using nonlinear programming. These methods do not use characteristics. Instead, the optimal control is found by directly

optimizing a discretized cost function, thus the name direct method. Different from indirect methods based on characteristics, direct methods do not involve the costate of the Hamiltonian dynamics in computation. Interconnections between direct and indirect methods were studied for some algorithms such as the covector mapping theorem of pseudospectral optimal control [17]. Without a thorough review of various direct methods, interested readers are referred to [2, 6, 10, 12, 13, 18, 14, 16, 17, 26, 25] and references therein. In [36], a direct method is applied to generate data for supervised learning. The method is exemplified by the optimal control of a quadcopter system.

5.1 Finite time optimal control

For the purpose of generating data, direct methods are causality-free. As an example, in the the following we outline the basic ideas of pseudospectral (PS) optimal control. Consider the following problem

$$\min_{\boldsymbol{u}(\cdot)} \mathcal{J}[\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot)] = \int_{-1}^{1} F(\boldsymbol{x}(t), \boldsymbol{u}(t))\ dt + E(\boldsymbol{x}(-1), \boldsymbol{x}(1)), \tag{16}$$

subject to

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)), \\ \boldsymbol{e}(\boldsymbol{x}(-1), \boldsymbol{x}(1); \boldsymbol{x}_0, \boldsymbol{x}_f) = \boldsymbol{0}, \\ \boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \le \boldsymbol{0}, \end{cases} \tag{17}$$

where $F : \mathbb{R}^{N_x} \times \mathbb{R}^{N_u} \to \mathbb{R}$, $E : \mathbb{R}^{N_x} \times \mathbb{R}^{N_x} \to \mathbb{R}$, $\boldsymbol{f} : \mathbb{R}^{N_x} \times \mathbb{R}^{N_u} \to \mathbb{R}^{N_x}$, $\boldsymbol{e} : \mathbb{R}^{2N_x} \times \mathbb{R}^{2N_x} \to \mathbb{R}^{N_e}$ and $\boldsymbol{h} : \mathbb{R}^{N_x} \times \mathbb{R}^{N_u} \to \mathbb{R}^{N_h}$ are continuously differentiable with respect to their arguments and their gradients are Lipschitz continuous. In (17), the endpoint condition, $\boldsymbol{e}(\boldsymbol{x}(-1), \boldsymbol{x}(1); \boldsymbol{x}_0, \boldsymbol{x}_f) = \boldsymbol{0}$, is given in a general form. For the purpose of generating data at random initial points, the endpoint condition is simplified to an initial value problem, $\boldsymbol{x}(-1) = \boldsymbol{x}_0$, where $\boldsymbol{x}_0$ is the random initial state of the data set.

The problem is discretized at a time sequence

$$t_0 = -1 < t_1 < t_2 < \cdots < t_N = 1.$$

There are various ways of chosing t_k. For example, the Legendre-Gauss-Lobatto (LGL) nodes (see, for instance, [14, 16]) are widely used in which t_k are the roots of the derivative of the Nth order Legendre polynomial. Let the discrete state and control variables be $\bar{\boldsymbol{x}}_k$ and $\bar{\boldsymbol{u}}_k$, then the problem defined in (16–17) is discretized to form a problem of nonlinear programming,

$$\min_{\bar{\boldsymbol{u}}} \bar{J} = \sum_{k=0}^{N} F(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) w_k + E(\bar{\boldsymbol{x}}_0, \bar{\boldsymbol{x}}_N), \tag{18}$$

subject to

$$\begin{cases} \| \sum_{i=0}^{N} \bar{\boldsymbol{x}}_i D_{ki} - \boldsymbol{f}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) \|_\infty \le \epsilon, \quad k = 0, 1, \cdots, N, \\ \| \boldsymbol{e}(\bar{\boldsymbol{x}}_0, \bar{\boldsymbol{x}}_N) \|_\infty \le \epsilon, \\ \| \boldsymbol{h}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) \|_\infty \le \epsilon, \qquad\qquad k = 0, 1, \cdots, N. \end{cases} \tag{19}$$

In this discretization, the constraints are relaxed in which the value of ϵ is chosen to guarantee feasibility. In (18–19), D_{kj} represent the elements in the differentiation matrix and w_k represent the LGL weights [16]. Problem (18–19) can be solved using numerical algorithms of nonlinear programming. Commercial or free software packages are available for this purpose. The continuous time solution is approximated by

$$\boldsymbol{x}(t) \approx \sum_{k=0}^{N} \bar{\boldsymbol{x}}_k \phi_k(t), \quad \boldsymbol{u}(t) \approx \sum_{k=0}^{N} \bar{\boldsymbol{u}}_k \psi_k(t),$$

where $\phi_k(t)$ is the Lagrange interpolating polynomial and $\psi_k(t)$ is any continuous function such that $\psi_k(t_j) = 1$ if $k = j$ and $\psi_k(t_j) = 0$ if $j \neq k$. Different choices of $\psi_k(t)$ are introduced in [16, 17, 25, 26]. The LGL PS method of optimal control are proved to have a high order convergence rate [25]. Direct methods based on Runge-Kutta or finite difference discretization are also widely used in applications. Interested readers are referred to [2] and [18].

5.2 Infinite time optimal control

The methodology described above for solving a nonlinear constrained optimal control over a finite time horizon can be extended to infinite time horizon problems. These type of optimal control problems arise naturally in applications in biology and economics and also in mechanics where stability and behavior of systems are considered over an infinite time interval. To handle these type of optimal control problems using PS methods, a variation of the LGL methods called the Legendre-Gauss-Radau (LGR) PS method was proposed in [15]. A brief description of the problem and the LGR method is introduced as follows. Consider the problem of determining the state-control function pair $[0, \infty) \ni s \mapsto \{\boldsymbol{x} \in \mathbb{R}^{N_x}, \boldsymbol{u} \in \mathbb{R}^{N_u}\}$ that minimize the cost

$$J[\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot)] = \int_0^\infty F(\boldsymbol{x}(s), \boldsymbol{u}(s), s)\, ds \tag{20}$$

subject to the dynamic constraints,

$$\dot{\boldsymbol{x}}(s) = \boldsymbol{f}(\boldsymbol{x}(s), \boldsymbol{u}(s), s) \tag{21}$$

end point *event* constraints,

$$\boldsymbol{e}\big(\boldsymbol{x}(0), \boldsymbol{x}(\infty)\big) = \mathbf{0} \tag{22}$$

and possibly mixed trajectory-control constraints,

$$\mathbf{g}_l \leq \mathbf{g}(\boldsymbol{x}(s), \boldsymbol{u}(s), s) \leq \mathbf{g}_u \tag{23}$$

In the above equations, $\dot{\boldsymbol{x}}$ denotes $d\boldsymbol{x}/ds$ and the various functions are defined as,

$$E: \quad \mathbb{R}^{N_x} \times N_x \to \mathbb{R} \tag{24}$$
$$F: \quad \mathbb{R}^{N_x} \times \mathbb{R}^{N_u} \times \mathbb{R} \to \mathbb{R} \tag{25}$$
$$\boldsymbol{f}: \quad \mathbb{R}^{N_x} \times \mathbb{R}^{N_u} \times \mathbb{R} \to \mathbb{R}^{N_x} \tag{26}$$
$$\boldsymbol{e}: \quad N_x \times \mathbb{R}^{N_x} \to \mathbb{R}^{N_e} \tag{27}$$
$$\mathbf{g}: \quad \mathbb{R}^{N_x} \times \mathbb{R}^{N_u} \times \mathbb{R} \to \mathbb{R}^{N_g} \tag{28}$$

where N_g is the dimension of the path constraint vector and N_e is the dimension of the event constraints. The above optimal control problem is posed on the semi-infinite time domain $[0, \infty)$ and usually this semi-infiniteness of the computational domain poses a challenge for direct numerical simulation of the problem. The usual technique of using the Riccati solution is typically applicable to the linear unconstrained quadratic cost problem. For many problems, linearization of the underlying problem to use the Riccati approach is unsatisfactory. In order to obtain a full solution for the original nonlinear constrained control problem with a general cost function, the following method is suggested. First, map the semi-infinite domain to the finite time domain $[-1, 1]$ and then use the appropriate quadrature nodes for the interpolation polynomials. For $s \in [0, \infty)$ and $t \in [-1, 1]$ we have

$$s = \frac{(1+t)}{1-t} \quad \leftrightarrow \quad t = \frac{s-1}{s+1}$$

Using this mapping we can reformulate the original optimal control problem on the finite interval $[-1, 1]$ using the derivative of the mapping

$$r(t) = \frac{ds}{dt} = \frac{2}{(1-t)^2} \tag{29}$$

Now the transformed optimal control problem is formulated as determining the state-control function pair $[-1, 1) \ni t \mapsto \{\boldsymbol{x} \in \mathbb{R}^{N_x}, \boldsymbol{u} \in \mathbb{R}^{N_u}\}$ that minimize the modified quadratic cost functional,

$$J[\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot)] = \int_{-1}^{1} F(\boldsymbol{x}(t), \boldsymbol{u}(t), s(t))\, r(t)\, dt \tag{30}$$

subject to the mapped dynamic constraints,

$$\frac{d\boldsymbol{x}}{dt} = \dot{\boldsymbol{x}}(t) = r(t)\, \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), s(t)) \tag{31}$$

end point constraints,

$$\boldsymbol{e}\big(\boldsymbol{x}(-1), \boldsymbol{x}(1)\big) = \boldsymbol{0} \tag{32}$$

and mixed trajectory-control constraints,

$$\mathbf{g}_l \leq \mathbf{g}(\boldsymbol{x}(s), \boldsymbol{u}(s), s) \leq \mathbf{g}_u \tag{33}$$

It should be noted that in the formulation above all functional evaluation at $t = 1$ which correspond to the value of the original function at $s = \infty$ is taken in the sense of limit. Now that the problem is posed on the finite time domain, the traditional quadrature interpolation using PS methods can be used in the way similar to the approach illustrated above. The one major modification required is the choice of node points. We choose LGR nodes for collocation. These node points, $t_j, j = 0, \cdots, N$, are defined $t_0 = -1$ and the rest of the nodes are defined as zeros of $L_N + L_{N+1}$ where L_N is the Legendre polynomial of degree N. Different from the LGL nodes that include $t = 1$, in the LGR nodes $t_N < 1$. As $N \to \infty$, t_N approaches $t = 1$ as a limit.

6 Stochastic process

Model-based deep learning for optimal control was first introduced in [19] for stochastic systems. In this approach, the optimal control law is approximated using a neural network. The learning process is based upon a data set generated from the stochastic model of the system, rather than data sets collected from experimentation. More recently, in [11, 20] solutions of the following semilinear parabolic PDEs are approximated using a neural network,

$$\begin{cases} V_t(t,\boldsymbol{x}) + \dfrac{1}{2}\mathrm{Tr}(\sigma\sigma^T \mathrm{Hess}_{\boldsymbol{x}} V)(t,\boldsymbol{x}) + V_{\boldsymbol{x}}(t,\boldsymbol{x})^T \mu(t,\boldsymbol{x}) + H(t,\boldsymbol{x},V(t,\boldsymbol{x}),\sigma^T(t,x)V_{\boldsymbol{x}}(t,x)) = 0, \\ V(t_f,\boldsymbol{x}) = \psi(\boldsymbol{x}). \end{cases} \tag{34}$$

As a special case, HJB equations with viscosity are PDEs in the form of (34). In this section, $\boldsymbol{x} \in \mathbb{R}^n$, $\sigma(t,\boldsymbol{x}) \in \mathbb{R}^{n\times n}$ is a matrix valued function, $\mu(t,\boldsymbol{x})$ is a vector valued function, $\mathrm{Hess}_{\boldsymbol{x}} V$ represents the Hessian of V with respect to $\boldsymbol{x}$, $\mathrm{Tr}(M)$ denotes the trace of a matrix M, $H : (-\infty, t_f] \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a known function. The characteristic of the PDE is a stochastic process satisfying

$$\boldsymbol{x}(t) = \boldsymbol{x}_0 + \int_0^t \mu(s,\boldsymbol{x}(s))ds + \int_0^t \sigma(s,\boldsymbol{x}(s))d\boldsymbol{W}_s, \tag{35}$$

where $\boldsymbol{W}_t$, $0 \le t \le t_f$, is an n-dimensional Brownian motion. The solution of (34) satisfies the following backward stochastic differential equation (BSDE) [33, 34],

$$\begin{cases} V(t,\boldsymbol{x}(t)) = V(0,\boldsymbol{x}_0) - \displaystyle\int_0^t H\left(s,\boldsymbol{x}(s),V(s,\boldsymbol{x}(s)),\sigma(s,\boldsymbol{x}(s))^T V_{\boldsymbol{x}}(s,\boldsymbol{x}(s))\right) ds \\ \qquad\qquad + \displaystyle\int_0^t V_{\boldsymbol{x}}(s,\boldsymbol{x}(s))^T \sigma(s,\boldsymbol{x}(s))d\boldsymbol{W}_s, \\ V(t_f,\boldsymbol{x}(t_f)) = \psi(\boldsymbol{x}(t_f)). \end{cases} \tag{36}$$

Discretize (35) and (36), we have

$$\boldsymbol{x}(t_{n+1}) \approx \boldsymbol{x}(t_n) + \mu(t_n,\boldsymbol{x}(t_n))(t_{n+1}-t_n) + \sigma(t_n,\boldsymbol{x}(t_n))(\boldsymbol{W}_{t_{n+1}} - \boldsymbol{W}_{t_n}), \quad n = 0,1,2,\cdots,N, \tag{37}$$

and

$$\begin{aligned} V(t_{n+1},\boldsymbol{x}(t_{n+1})) \approx\ & V(t_n,\boldsymbol{x}(t_n)) - H\left(t_n,\boldsymbol{x}(t_n),V(t_n,\boldsymbol{x}(t_n)),\sigma(t_n,\boldsymbol{x}(t_n))^T V_{\boldsymbol{x}}(t_n,\boldsymbol{x}(t_n))\right)(t_{n+1}-t_n) \\ & + V_{\boldsymbol{x}}(t_n,\boldsymbol{x}(t_n))^T \sigma(t_n,\boldsymbol{x}(t_n))(\boldsymbol{W}_{t_{n+1}} - \boldsymbol{W}_{t_n}), \end{aligned} \tag{38}$$

for $n = 0,1,2,\cdots,N-1$. In (38), the value of $V(t,\boldsymbol{x})$ depends on its gradient, $V_{\boldsymbol{x}}(t,\boldsymbol{x})$, which is unknown. In [20], a neural network is defined to approximate the function $x \to \sigma^T V_{\boldsymbol{x}}(t_n,\boldsymbol{x})$ at $t = t_n$. In addition to the parameters of the neural network, the initial value and initial gradient are also treated as parameters. For the data set, the first step is to generate a set of Brownian motion paths,

$$\left\{ \{\boldsymbol{W}_{t_n}^i\}_{n=0}^N \middle|\ \ i = 1,2,\cdots,N_s \right\},$$

where N_s is the total number of samples. Then, one can compute all sample paths of the stochastic process by solving (37) (forward in time),

$$\left\{ \{\boldsymbol{x}^i(t_n)\}_{n=0}^N \middle|\ \ i = 1,2,\cdots,N_s \right\}.$$

Integrating (38) along each sample path, one can compute $\hat{V}(t_n, \boldsymbol{x}^{(i)}(t_n))$, an approximation of the value function. Note that this approximation depends on the parameters of the neural network. In [20], the neural network is trained using loss functions that penalize the mean square error of the terminal condition

$$V(t_N, \boldsymbol{x}(t_N)) = \psi(\boldsymbol{x}(t_N)). \tag{39}$$

For instance, one of such loss functions is

$$l(\boldsymbol{\theta}) = \mathbb{E}\left(\left\{ |\psi(\boldsymbol{x}^{(i)}(t_N)) - \hat{V}(t_N, \boldsymbol{x}^{(i)}(t_N))|^2 \middle| \ \ 1 \le i \le N_s \right\}\right), \tag{40}$$

where $\boldsymbol{\theta}$ represents the parameters in the neural network and the unknown initial value and initial gradient of $V(t, \boldsymbol{x})$.

Some results on the error analysis for this type of algorithms are published in a recent paper [21]. In addition to optimal control, neural networks are applicable to solve various types of PDEs. In [11, 20, 35], neural networks are trained to solve PDEs with space dimensions up to $n = 100$.

7 Characteristics for general PDEs

The basic idea discussed in this paper is not limited to optimal control problems. In principal, any PDE that has a causality-free algorithm allows one to generate data for supervised learning and accuracy validation. Illustrated in Section 3, finding solutions along characteristic curves is a causality-free process, i.e., the solution can be found point-by-point without using a grid. For instance, a characteristic method of solving 1D conservation law was introduced in [29]. Because of the causality-free property, the algorithm is able to provide accurate solution for systems with complicated shocks. In general, any quasilinear PDE,

$$\sum_{i=1}^{n} a_i(x_1, \cdots, x_n, u)\frac{\partial u}{\partial x_i} = c(x_1, \cdots, x_n, u), \tag{41}$$

has a characteristic, $(\bar{x}(s), \bar{u}(s))$, defined by a system of ODEs

$$\begin{cases} \dfrac{d\bar{x}_i}{ds} = a_i(\bar{x}_1(s), \cdots, \bar{x}_n(s), \bar{u}(s)), \\ \dfrac{d\bar{u}}{ds} = c(\bar{x}_1(s), \cdots, \bar{x}_n(s), \bar{u}(s)). \end{cases} \tag{42}$$

Then the solution of the PDE satisfies

$$u(\bar{x}_1(s), \cdots, \bar{x}_n(s)) = \bar{u}(s).$$

A challenge of this method is that the characteristic curves may cross each other to form shocks. Also, the curves may not cover the entire region, thus forming rarefaction. Nevertheless, if a unique solution can be defined based on characteristics, the resulting algorithm is causality-free. It can be used to generate data to train a neural network as an approximate solution. The data can also be used to check the accuracy of the neural network solution.

8 Summary

To put the surveyed algorithms in perspective, we summarize them in Table 1. It contains the references where the interested readers can find technical details of the algorithms. It contains brief information about the type of examples shown in the references. The table also contains brief comments on the applicability and limitations of the algorithms.

Method	**Initial Guess Required**	**References**	**Examples**
Time or space-marching	No	[27, 28, 32]	Optimal control of rigid body, $n = 6$. Optimal control of Burgers' equation, $n = 30$.
	Comments: Computational convergence and speed depend on the number of marching steps. It requires TPBVP solvers.		
NN warm start	Yes	[32]	Optimal control of rigid body, $n = 6$. Optimal control of Burgers' equation, $n = 30$.
	Comments: Computational convergence and speed depend on the quality of neural network initial guess. It requires TPBVP solvers.		
Backward propagation	No	[23]	Space system interplanetary transfer, $n = 7$.
	Comments: No convergence issue. Initial states in data cannot be pre-selected.		
Hopf formula	Yes	[7]	Optimal control of (10)-(11), $n = 4, 8, 12, 16$.
	Comments: Limited to control systems in the form of (10)-(11). It requires unconstrained optimization such as Bregman's algorithm.		
Minimization along characteristics	Yes	[38]	Differential game of state affine systems, $n = 4$.
	Comments: Computational convergence and speed depend on the initial guess. It requires unconstrained optimization such as Powell's algorithm or coordinate descent algorithm.		
Direct methods	Yes	[16, 17, 36]	Optimal control with state-control constraints, $n = 4, 5, 7$.
	Comments: The method is effective for problems with state-control constraints. It requires nonlinear programming software or solvers.		
Stochastic process	Yes	[11, 20]	Optimal control of stochastic systems, $n = 100$.
	Comments: The method is applicable to stochastic optimal control. It requires unconstrained optimization for neural network training.		

Table 1: A summary of the surveyed algorithms with references and brief comments on their applicability and limitations. In the column "Examples," n represents the state space dimension of the examples that can be found in the references

As demonstrated in the example of attitude control, causality-free algorithms generate data for not only the training of neural networks but also the validation of their accuracy. A guaranteed error upper bound is often impossible to be mathematically proved for applications of neural networks. In this case, an empirically computed approximate error provides critical information and confidence for practical applications. This is a main advantage of causality-free algorithms for the purpose of deep learning. In addition, the methods surveyed in this paper are all model-based. One has a full control of the location and amount of data to be generated, a property that is

very useful for an adaptive training process. In computation, generating data using causality-free algorithms has perfect parallelism because the solution at each point is computed individually without using the function value at other points.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, et al. TensorFlow: Large-scale machine learning on heterogeneous systems. http://www.tensorflow.org/, 2015–.

[2] John Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*. SIAM, Philadelphia, 2001.

[3] Andreas Bittracher, Stefan Klus, Boumediene Hamzi, and Christof Schütte. A kernel-based method for coarse graining complex dynamical systems. *arXiv:1904.08622v1*, 2019.

[4] Richard H. Byrd, Peihang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16:1190–1208, 1995.

[5] Yat Tin Chow, Jerome Darbon, Stanley Osher, and Wotao Yin. Algorithm for overcoming the curse of dimensionality for state-dependent Hamilton-Jacobi equations. *J. Comput. Phys.*, 387:376–409, 2019.

[6] I. Chryssoverghi, J. Coletsos, and B. Kokkinis. Discretization methods for optimal control problems with state constraints. *J. Comput. Appl. Math.*, 191:1–31, 2006.

[7] Jerome Darbon and Stanley Osher. Algorithms for overcoming the curse of dimensionality for certain Hamilton-Jacobi equations arising in control theory and elsewhere. *Res. Math. Sci.*, 3(1), 2016.

[8] K. Dekker and J. G. Verwer. *Stability of Runge-Kutta Methods for Stiff Nonlinear Differential Equations*. Elsevier Science Publishers B. V., 2001.

[9] James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. https://www.astro.rug.nl/software/kapteyn-beta/_downloads/attitude.pdf, 2006.

[10] A. L. Dontchev and William W. Hager. The Euler approximation in state constrained optimal control. *Math. Comput.*, 70:173–203, 2001.

[11] Weinan E, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017.

[12] G. Elnagar, M. A. Kazemi, and M. Razzaghi. The pseudospectral legendre method for discretizing optimal control problems. *IEEE Trans. Autom. Control*, 40(10):1793–1796, 1995.

[13] Paul J. Enright and Bruce A. Conway. Discrete approximations to optimal trajectories using direct transcription and nonlinear programming. *J. Guid. Control Dyn.*, 15(4):994–1002, 1992.

[14] Fariba Fahroo and I. Michael Ross. Costate estimation by a legendre pseudospectral method. *J. Guid. Control Dyn.*, 24(2):270–277, 2001.

[15] Fariba Fahroo and I. Michael Ross. Pseudospectral methods for infinite horizon optimal control problems. *Journal of Guidance, Control, and Dynamics*, 31(4):927–936, 2008.

[16] Qi Gong, Wei Kang, and I. Michael Ross. A pseudospectral method for the optimal control of constrained feedback linearizable systems. *IEEE Trans. Automat. Control*, 51(7):1115–1129, 2006.

[17] Qi Gong, I. Michael Ross, Wei Kang, and Fariba Fahroo. Connections between the covector mapping theorem and convergence of pseudospectral methods for optimal control. *Comput. Optim. Appl.*, 41(3):307–335, 2008.

[18] William W. Hager. Runge-kutta methods in optimal control and the transformed adjoint system. *Numer. Math.*, 87(2):247–282, 2000.

[19] Jiequn Han and Weinan E. Deep learning approximation for stochastic control problems. *arXiv:1611.07422v1*, 2016.

[20] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[21] Jiequn Han and Jihao Long. Convergence of the deep bsde method for coupled fbsdes. *Probab Uncertain Quant. Risk*, 5(5), 2020.

[22] Eberhard Hopf. Generalized solutions of nonlinear equations of the first order. *J. Math. Mech.*, 14(6):951–973, 1965.

[23] Dario Izzo, Ekin Öztürk, and Marcus Märtens. Interplanetary transfers via deep representations of the optimal policy and/or of the value function. *arXiv:1904.08809*, 2019.

[24] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python. http://www.scipy.org/, 2001–.

[25] Wei Kang. Rate of convergence for the legendre pseudospectral optimal control of feedback linearizable systems. *J. Control Theory Appl.*, 8(4):391–405, 2010.

[26] Wei Kang, Qi Gong, I. Michael Ross, and Fariba Fahroo. On the convergence of nonlinear optimal control using pseudospectral methods for feedback linearizable systems. *International Journal of Robust and Nonlinear Control*, 17:1251–1277, 2007.

[27] Wei Kang and Lucas C. Wilcox. A causality free computational method for HJB equations with application to rigid body satellites. In *AIAA Guidance, Navigation, and Control Conference*, AIAA 2015-2009, Kissimmee, FL, 2015.

[28] Wei Kang and Lucas C. Wilcox. Mitigating the curse of dimensionality: sparse grid characteristic method for optimal feedback control and HJB equations. *Comput. Optim. Appl.*, 68(2):289–315, 2017.

[29] Wei Kang and Lucas C. Wilcox. Solving 1D conservation laws using pontryagin's minimum principle. *J. Sci. Comput.*, 71(1):144–165, 2017.

[30] Jacek Kierzenka and Lawrence F. Shampine. BVP solver that controls residual and error. *Journal of Numerical Analysis, Industrial and Applied Mathematics*, 3(1-2):27–41, 2008.

[31] Alex Tong Lin, Yat Tin Chow, and Stanley Osher. A splitting method for overcoming the curse of dimensionality in Hamilton-Jacobi equations arising from nonlinear optimal control and differential games with applications to trajectory generation. *arXiv:1803.01215*, 2018.

[32] Tenavi Nakamura-Zimmerer, Qi Gong, and Wei Kang. Adaptive deep learning for high-dimensional Hamilton-Jacobi-Bellman equations. *arXiv:1907.05317*, 2019.

[33] Etienne Pardoux and Shige Peng. Backward stochastic differential equations and quasilinear parabolic partial differential equations. In Rozovskii B. L. and Sowers R. B., editors, *Stochastic partial differential equations and their applications*, volume 176 of *Lecture Notes in Control and Information Sciences*, pages 200–217. Springer-Verlag Berlin Heidelberg, 1992.

[34] Etienne Pardoux and Tang Shanjian. Forward-backward stochastic differential equations and quasilinear parabolic pdes. *Probability Theory and Related Fields*, 114(2):123–150, 1999.

[35] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. of Computational Physics*, 378:686–707, 2019.

[36] Dharmesh Tailor and Dario Izzo. Learning the optimal state-feedback via supervised imitation learning. *arXiv:1901.02369v2*, 2019.

[37] Jin Wang, Jie Huang, and Stephen S.T. Yau. Approximate nonlinear output regulation based on the universal approximation theorem. *International Journal of Robust and Nonlinear Control*, 10:439–456, 2000.

[38] Ivan Yegorov and Peter M. Dower. Perspectives on characteristics based curse-of-dimensionality-free numerical approaches for solving Hamilton-Jacobi equations. *Appl. Math. Optim.*, 2018.