10 Data-Driven Control

As described in Chapter 8, control design often begins with a model of the system being controlled. Notable exceptions include model-free adaptive control strategies and many uses of PID control. For mechanical systems of moderate dimension, it may be possible to write down a model (e.g., based on the Newtonian, Lagrangian, or Hamiltonian formalism) and linearize the dynamics about a fixed point or periodic orbit. However, for modern systems of interest, as are found in neuroscience, turbulence, epidemiology, climate, and finance, typically there are no simple models suitable for control design. Chapter 9 described techniques to obtain control-oriented reduced-order models for high-dimensional systems from data, but these approaches are limited to *linear* systems. Real-world systems are usually nonlinear and the control objective is not readily achieved via linear techniques. Nonlinear control can still be posed as an optimization problem with a high-dimensional, nonconvex cost function landscape with multiple local minima. Machine learning is complementary, as it constitutes a growing set of techniques that may be broadly described as performing nonlinear optimization in a high-dimensional space from data. In this chapter we describe emerging techniques that use machine learning to characterize and control strongly nonlinear, high-dimensional, and multi-scale systems, leveraging the increasing availability of high-quality measurement data.

Broadly speaking, machine learning techniques may be used to 1) characterize a system for later use with model-based control, or 2) directly characterize a control law that effectively interacts with a system. This is illustrated schematically in Fig. 10.1, where data-driven techniques may be applied to either the *System* or *Controller* blocks. In addition, related methods may also be used to identify good sensors and actuators, as discussed previously in Section 3.8. In this chapter, Section 10.1 will explore the use of machine learning to identify nonlinear input–output models for control, based on the methods from Chapter 7. In Section 10.2 we will explore machine learning techniques to directly identify controllers from input–output data. This is a rapidly developing field, with many powerful methods, such as reinforcement learning, iterative learning control, and genetic algorithms. Here we provide a high-level overview of these methods and then explore an example using genetic algorithms. However, it is important to emphasize the breadth and depth of this field, and the fact that any one method may be the subject of an entire book. Finally, in Section 10.3 we describe the adaptive extremum-seeking control strategy, which optimizes the control signal based on how the system responds to perturbations.

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

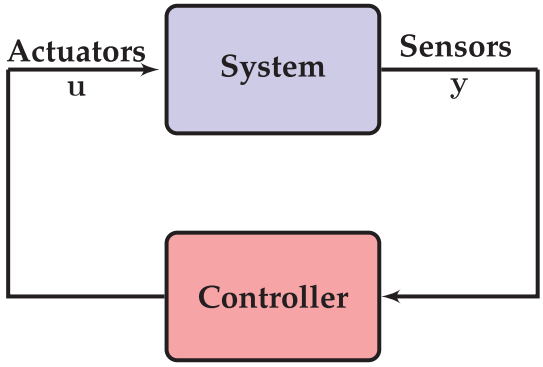

Figure 10.1 In the standard control framework from Chapter 8, machine learning may be used 1) to develop a model of the system or 2) to learn a controller.

10.1 Nonlinear System Identification for Control

The data-driven modeling and control of complex systems is undergoing a revolution, driven by the rise of big data, advanced algorithms in machine learning and optimization, and modern computational hardware. Despite the increasing use of equation-free and adaptive control methods, there remains a wealth of powerful model-based control techniques, such as linear optimal control (see Chapter 8) and model predictive control (MPC) [195, 107]. Increasingly, these model-based control strategies are aided by data-driven techniques that characterize the input–output dynamics of a system of interest from measurements alone, without relying on first principles modeling. Broadly speaking, this is known as *system identification*, which has a long and rich history in control theory going back decades to the time of Kalman. However, with increasingly powerful data-driven techniques, such as those described in Chapter 7, nonlinear system identification is the focus of renewed interest.

The goal of system identification is to identify a low-order model of the input–output dynamics from actuation $\mathbf{u}$ to measurements $\mathbf{y}$. If we are able to measure the full state $\mathbf{x}$ of the system, then this reduces to identifying the dynamics $\mathbf{f}$ that satisfy:

$$\frac{d}{dt}\mathbf{x} = \mathbf{f}(\mathbf{x}, \mathbf{u}). \tag{10.1}$$

This problem may be formulated in discrete-time, since data is typically collected at discrete instances in time and control laws are often implemented digitally. In this case, the dynamics read:

$$\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k, \mathbf{u}_k). \tag{10.2}$$

When the dynamics are approximately linear, we may identify a linear system

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k, \tag{10.3}$$

which is the approach taken in the DMD with control (DMDc) algorithm below.

It may also be advantageous to identify a set of measurements $\mathbf{y} = \mathbf{g}(\mathbf{x})$, in which the unforced nonlinear dynamics appear linear:

$$\mathbf{y}_{k+1} = \mathbf{A}_\mathbf{Y}\mathbf{y}_k. \tag{10.4}$$

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

This is the approach taken in the Koopman control method below. In this way, nonlinear dynamics may be estimated and controlled using standard textbook linear control theory in the intrinsic coordinates $\mathbf{y}$ [302, 276].

Finally, the nonlinear dynamics in (10.1) or (10.2) may be identified directly using the SINDY with control algorithm. The resulting models may be used with model predictive control for the control of fully nonlinear systems [277].

DMD with Control

Proctor et al. [434] extended the DMD algorithm to include the effect of actuation and control, in the so-called DMD with control (DMDc) algorithm. It was observed that naively applying DMD to data from a system with actuation would often result in incorrect dynamics, as the effects of internal dynamics are confused with the effects of actuation. DMDc was originally motivated by the problem of characterizing and controlling the spread of disease, where it is unreasonable to stop intervention efforts (e.g., vaccinations) just to obtain a characterization of the unforced dynamics [435]. Instead, if the actuation signal is measured, a new DMD regression may be formulated in order to disambiguate the effect of internal dynamics from that of actuation and control. Subsequently, this approach has been extended to perform DMDc on heavily subsampled or compressed measurements by Bai et al. [30].

The DMDc method seeks to identify the best-fit linear operators $\mathbf{A}$ and $\mathbf{B}$ that approximately satisfy the following dynamics on measurement data:

$$\mathbf{x}_{k+1} \approx \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k. \tag{10.5}$$

In addition to the snapshot matrix $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_m \end{bmatrix}$ and the time-shifted snapshot matrix $\mathbf{X}' = \begin{bmatrix} \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_{m+1} \end{bmatrix}$ from (7.23), a matrix of the actuation input history is assembled:

$$\boldsymbol{\Upsilon} = \begin{bmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_m \\ | & | & & | \end{bmatrix}. \tag{10.6}$$

The dynamics in (10.5) may be written in terms of the data matrices:

$$\mathbf{X}' \approx \mathbf{A}\mathbf{X} + \mathbf{B}\boldsymbol{\Upsilon}. \tag{10.7}$$

As in the DMD algorithm (see Section 7.2), the leading eigenvalues and eigenvectors of the best-fit linear operator $\mathbf{A}$ are obtained via dimensionality reduction and regression. If the actuation matrix $\mathbf{B}$ is known, then it is straightforward to correct for the actuation and identify the spectral decomposition of $\mathbf{A}$ by replacing $\mathbf{X}'$ with $\mathbf{X}' - \mathbf{B}\boldsymbol{\Upsilon}$ in the DMD algorithm:

$$\left(\mathbf{X}' - \mathbf{B}\boldsymbol{\Upsilon}\right) \approx \mathbf{A}\mathbf{X}. \tag{10.8}$$

When $\mathbf{B}$ is unknown, both $\mathbf{A}$ and $\mathbf{B}$ must be simultaneously identified. In this case, the dynamics in (10.7) may be recast as:

$$\mathbf{X}' \approx \begin{bmatrix} \mathbf{A} & \mathbf{B} \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ \boldsymbol{\Upsilon} \end{bmatrix} = \mathbf{G}\boldsymbol{\Omega}, \tag{10.9}$$

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

and the matrix $\mathbf{G} = \begin{bmatrix}\mathbf{A} & \mathbf{B}\end{bmatrix}$ is obtained via least-squares regression:

$$\mathbf{G} \approx \mathbf{X}'\boldsymbol{\Omega}^{\dagger}. \tag{10.10}$$

The matrix $\boldsymbol{\Omega} = \begin{bmatrix}\mathbf{X}^* & \boldsymbol{\Upsilon}^*\end{bmatrix}^*$ is generally a high-dimensional data matrix, which may be approximated using the SVD:

$$\boldsymbol{\Omega} = \tilde{\mathbf{U}}\tilde{\boldsymbol{\Sigma}}\tilde{\mathbf{V}}^*. \tag{10.11}$$

The matrix $\tilde{\mathbf{U}}$ must be split into two matrices, $\tilde{\mathbf{U}} = \begin{bmatrix}\tilde{\mathbf{U}}_1^* & \tilde{\mathbf{U}}_2^*\end{bmatrix}^*$, to provide bases for $\mathbf{X}$ and $\boldsymbol{\Upsilon}$. Unlike the DMD algorithm, $\tilde{\mathbf{U}}$ provides a reduced basis for the *input space*, while $\hat{\mathbf{U}}$ from

$$\mathbf{X}' = \hat{\mathbf{U}}\hat{\boldsymbol{\Sigma}}\hat{\mathbf{V}}^* \tag{10.12}$$

defines a reduced basis for the *output space*. It is then possible to approximate $\mathbf{G} = \begin{bmatrix}\mathbf{A} & \mathbf{B}\end{bmatrix}$ by projecting onto this basis:

$$\tilde{\mathbf{G}} = \hat{\mathbf{U}}^*\mathbf{G}\begin{bmatrix}\hat{\mathbf{U}} \\ \mathbf{I}\end{bmatrix}. \tag{10.13}$$

The resulting projected matrices $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$ in $\tilde{\mathbf{G}}$ are:

$$\tilde{\mathbf{A}} = \hat{\mathbf{U}}^*\mathbf{A}\hat{\mathbf{U}} = \hat{\mathbf{U}}^*\mathbf{X}'\tilde{\mathbf{V}}\tilde{\boldsymbol{\Sigma}}^{-1}\tilde{\mathbf{U}}_1^*\hat{\mathbf{U}} \tag{10.14a}$$

$$\tilde{\mathbf{B}} = \hat{\mathbf{U}}^*\mathbf{B} \quad = \hat{\mathbf{U}}^*\mathbf{X}'\tilde{\mathbf{V}}\tilde{\boldsymbol{\Sigma}}^{-1}\tilde{\mathbf{U}}_2^*. \tag{10.14b}$$

More importantly, it is possible to recover the DMD eigenvectors $\boldsymbol{\Phi}$ from the eigendecomposition $\tilde{\mathbf{A}}\mathbf{W} = \mathbf{W}\boldsymbol{\Lambda}$:

$$\boldsymbol{\Phi} = \mathbf{X}'\tilde{\mathbf{V}}\tilde{\boldsymbol{\Sigma}}^{-1}\tilde{\mathbf{U}}_1^*\hat{\mathbf{U}}\mathbf{W}. \tag{10.15}$$

Ambiguity in Identifying Closed-Loop Systems

For systems that are being actively controlled via feedback, with $\mathbf{u} = \mathbf{K}\mathbf{x}$,

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{10.16a}$$

$$= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{K}\mathbf{x}_k \tag{10.16b}$$

$$= (\mathbf{A} + \mathbf{B}\mathbf{K})\mathbf{x}_k, \tag{10.16c}$$

it is impossible to disambiguate the dynamics $\mathbf{A}$ and the actuation $\mathbf{BK}$. In this case, it is important to add perturbations to the actuation signal $\mathbf{u}$ to provide additional information. These perturbations may be a white noise process or occasional impulses that provide a kick to the system, providing a signal to disambiguate the dynamics from the feedback signal.

Koopman Operator Nonlinear Control

For nonlinear systems, it may be advantageous to identify data-driven coordinate transformations that make the dynamics appear linear. These coordinate transformations are related to *intrinsic* coordinates defined by eigenfunctions of the Koopman operator (see Section 7.4). Koopman analysis has thus been leveraged for nonlinear estimation [504, 505] and control [302, 276, 423].

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

It is possible to design estimators and controllers directly from DMD or eDMD models, and Korda et al. [302] used model predictive control (MPC) to control nonlinear systems with eDMD models. MPC performance is also surprisingly good for DMD models, as shown in Kaiser et al. [277]. In addition, Peitz et al. [423] demonstrated the use of MPC for switching control between a small number of actuation values to track a reference value of lift in an unsteady fluid flow; for each constant actuation value, a separate eDMD model was characterized. Surana [504] and Surana and Banaszuk [505] have also demonstrated excellent nonlinear estimators based on Koopman Kalman filters. However, as discussed previously, eDMD models may contain many spurious eigenvalues and eigenvectors because of closure issues related to finding a Koopman-invariant subspace. Instead, it may be advantageous to identify a handful of relevant Koopman eigenfunctions and perform control directly in these coordinates [276].

In Section 7.5, we described several strategies to approximate Koopman eigenfunctions, $\varphi(\mathbf{x})$, where the dynamics become linear:

$$\frac{d}{dt}\varphi(\mathbf{x}) = \lambda\varphi(\mathbf{x}). \tag{10.17}$$

In Kaiser et al. [276] the Koopman eigenfunction equation was extended for control-affine nonlinear systems:

$$\frac{d}{dt}\mathbf{x} = \mathbf{f}(\mathbf{x}) + \mathbf{B}\mathbf{u}. \tag{10.18}$$

For these systems, it is possible to apply the chain rule to $\frac{d}{dt}\varphi(\mathbf{x})$, yielding:

$$\frac{d}{dt}\varphi(\mathbf{x}) = \nabla\varphi(\mathbf{x}) \cdot (\mathbf{f}(\mathbf{x}) + \mathbf{B}\mathbf{u}) \tag{10.19a}$$

$$= \lambda\varphi(\mathbf{x}) + \nabla\varphi(\mathbf{x}) \cdot \mathbf{B}\mathbf{u}. \tag{10.19b}$$

Note that even with actuation, the dynamics of Koopman eigenfunctions remain linear, and the effect of actuation is still additive. However, now the actuation mode $\nabla\varphi(\mathbf{x}) \cdot \mathbf{B}$ may be state dependent. In fact, the actuation *will* be state dependent unless the directional derivative of the eigenfunction is constant in the $\mathbf{B}$ direction. Fortunately, there are many powerful generalizations of standard Riccati-based linear control theory (e.g., LQR, Kalman filters, etc.) for systems with a *state-dependent* Riccati equation.

SINDy with Control

Although it is appealing to identify intrinsic coordinates along which nonlinear dynamics appear linear, these coordinates are challenging to discover, even for relatively simple systems. Instead, it may be beneficial to directly identify the nonlinear actuated dynamical system in (10.1) or (10.2), for use with standard model-based control. Using the sparse identification of nonlinear dynamics (SINDy) method (see Section 7.3) results in computationally efficient models that may be used in real-time with model predictive control [277]. Moreover, these models may be identified from relatively small amounts of training data, compared with neural networks and other leading machine learning methods, so that they may even be characterized online and in response to abrupt changes to the system dynamics.

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

The SINDy algorithm is readily extended to include the effects of actuation [100, 277]. In addition to collecting measurements of the state snapshots $\mathbf{x}$ in the matrix $\mathbf{X}$, actuation inputs $\mathbf{u}$ are collected in the matrix $\boldsymbol{\Upsilon}$ from (10.6) as in DMDc. Next, an augmented library of candidate right hand side functions $\boldsymbol{\Theta}(\begin{bmatrix}\mathbf{X} & \boldsymbol{\Upsilon}\end{bmatrix})$ is constructed:

$$\boldsymbol{\Theta}(\begin{bmatrix}\mathbf{X} & \boldsymbol{\Upsilon}\end{bmatrix}) = \begin{bmatrix}\mathbf{1} & \mathbf{X} & \boldsymbol{\Upsilon} & \mathbf{X}^2 & \mathbf{X}\otimes\boldsymbol{\Upsilon} & \boldsymbol{\Upsilon}^2 & \cdots\end{bmatrix}. \tag{10.20}$$

Here, $\mathbf{X}\otimes\boldsymbol{\Upsilon}$ denotes quadratic cross-terms between the state $\mathbf{x}$ and the actuation $\mathbf{u}$, evaluated on the data.

In SINDy with control (SINDYc), the same sparse regression is used to determine the fewest active terms in the library required to describe the observed dynamics. As in DMDc, if the system is being actively controlled via feedback $\mathbf{u} = \mathbf{K}(\mathbf{x})$, then it is impossible to disambiguate from the internal dynamics and the actuation, unless an addition perturbation signal is added to the actuation to provide additional information.

Model Predictive Control (MPC) Example

In this example, we will use SINDYc to identify a model of the forced Lorenz equations from data and then control this model using model predictive control (MPC). MPC [107, 195, 438, 391, 447, 439, 196, 326, 173] has become a cornerstone of modern process control and is ubiquitous in the industrial landscape. MPC is used to control strongly nonlinear systems with constraints, time delays, non-minimum phase dynamics, and instability. Most industrial applications of MPC use empirical models based on linear system identification (see Chapter 8), neural networks (see Chapter 6), Volterra series [86, 73], and autoregressive models [6] (e.g., ARX, ARMA, NARX, and NARMAX). Recently, deep learning and reinforcement learning have been combined with MPC [330, 570] with impressive results. However, deep learning requires large volumes of data and may not be readily interpretable. A complementary line of research seeks to identify models for MPC based on limited data to characterize systems in response to abrupt changes.

Model predictive control determines the next immediate control action by solving an optimal control problem over a receding horizon. In particular, the open-loop actuation signal $\mathbf{u}$ is optimized on a receding time-horizon $t_c = m_c\Delta t$ to minimize a cost J over some prediction horizon $t_p = m_p\Delta t$. The control horizon is typically less than or equal to the prediction horizon, and the control is held constant between t_c and t_p. The optimal control is then applied for one time step, and the procedure is repeated and the receding-horizon control re-optimized at each subsequent time step. This results in the control law:

$$\mathbf{K}(\mathbf{x}_j) = \mathbf{u}_{j+1}(\mathbf{x}_j), \tag{10.21}$$

where $\mathbf{u}_{j+1}$ is the first time step of the optimized actuation starting at $\mathbf{x}_j$. This is shown schematically in Fig. 10.2. It is possible to optimize highly customized cost functions, subject to nonlinear dynamics, with constraints on the actuation and state. However, the computational requirements of re-optimizing at each time-step are considerable, putting limits on the complexity of the model and optimization techniques. Fortunately, rapid advances in computing power and optimization are enabling MPC for real-time nonlinear control.

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

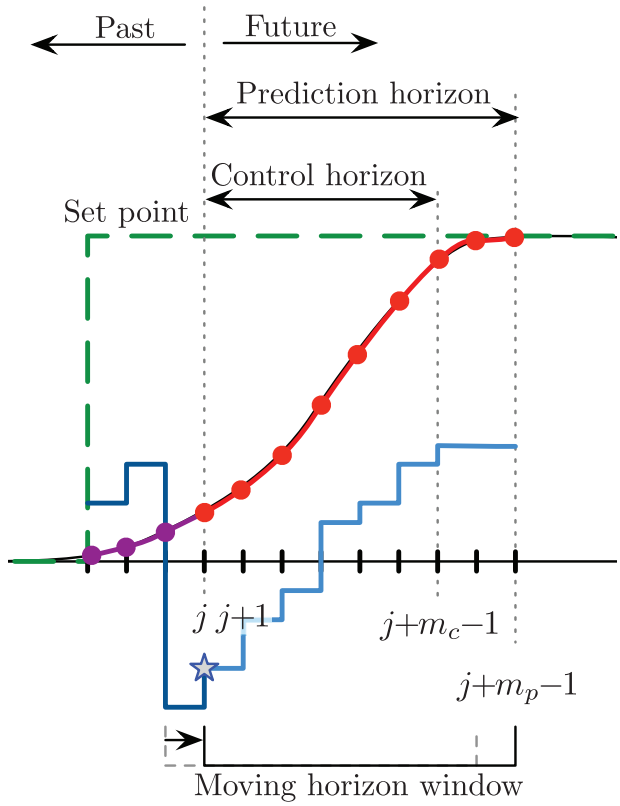

Figure 10.2 Schematic overview of model predictive control, where the actuation input **u** is iteratively optimized over a receding horizon. *Reproduced with permission from Kaiser et al. [277].*

MPC to Control the Lorenz Equations with SINDYc

The following example illustrates how to identify a model with SINDYc for use in MPC. The basic code is the same as SINDy, except that the actuation is included as a variable when building the library $\boldsymbol{\Theta}$.

We test the SINDYc model identification on the forced Lorenz equations:

$$\dot{x} = \sigma(y - x) + g(u) \tag{10.22a}$$

$$\dot{y} = x(\rho - z) - y \tag{10.22b}$$

$$\dot{z} = xy - \beta z. \tag{10.22c}$$

In this example, we train a model using 20 time units of controlled data, and validate it on another 20 time units where we switch the forcing to a periodic signal $u(t) = 50\sin(10t)$. The SINDY algorithm does not capture the effect of actuation, while SINDYc correctly identifies the forced model and predicts the behavior in response to a new actuation that was not used in the training data, as shown in Fig. 10.3.

Finally, SINDYc and neural network models of Lorenz are both used to design model predictive controllers, as shown in Fig. 10.4. Both methods identify accurate models that

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

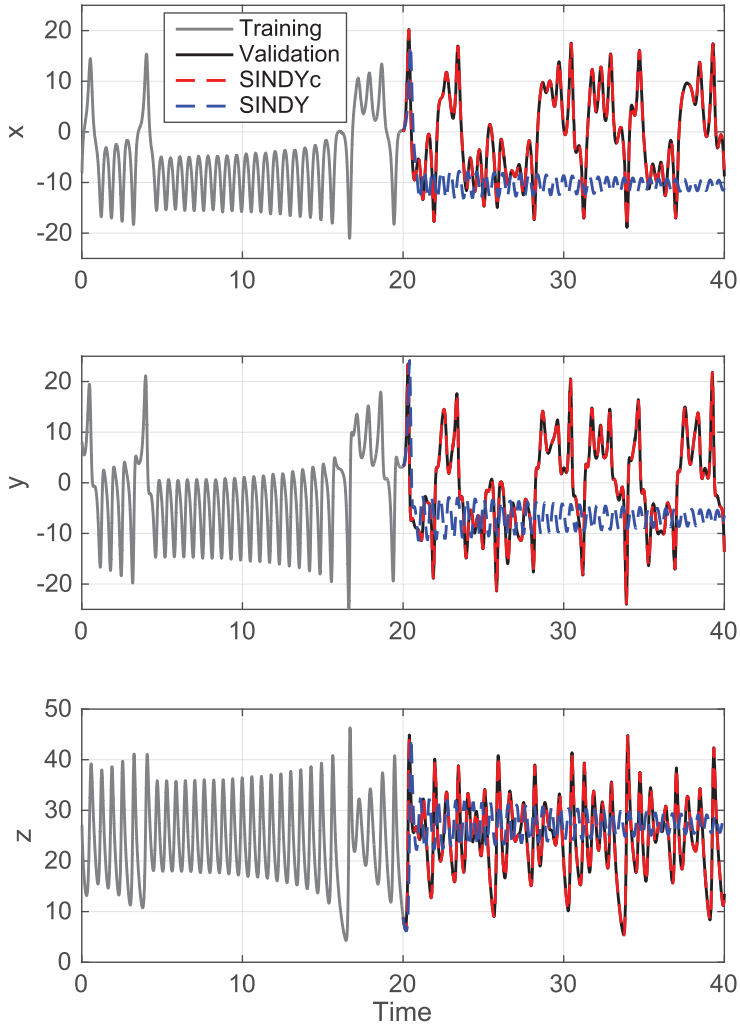

Figure 10.3 SINDY and SINDYc predictions for the controlled Lorenz system in (10.22). Training data consists of the Lorenz system with state feedback. For the training period the input is $u(t) = 26 - x(t) + d(t)$ with a Gaussian disturbance d. Afterward the input u switches to a periodic signal $u(t) = 50\sin(10t)$. *Reproduced with permission from [100].*

capture the dynamics, although the SINDYc procedure requires less data, identifies models more rapidly, and is more robust to noise than the neural network model. This added efficiency and robustness is due to the sparsity promoting optimization, which regularizes the model identification problem. In addition, identifying a sparse model requires less data.

10.2 Machine Learning Control

Machine learning is a rapidly developing field that is transforming our ability to describe complex systems from observational data, rather than first-principles modeling [382, 161, 64, 396]. Until recently, these methods have largely been developed for static data, although

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

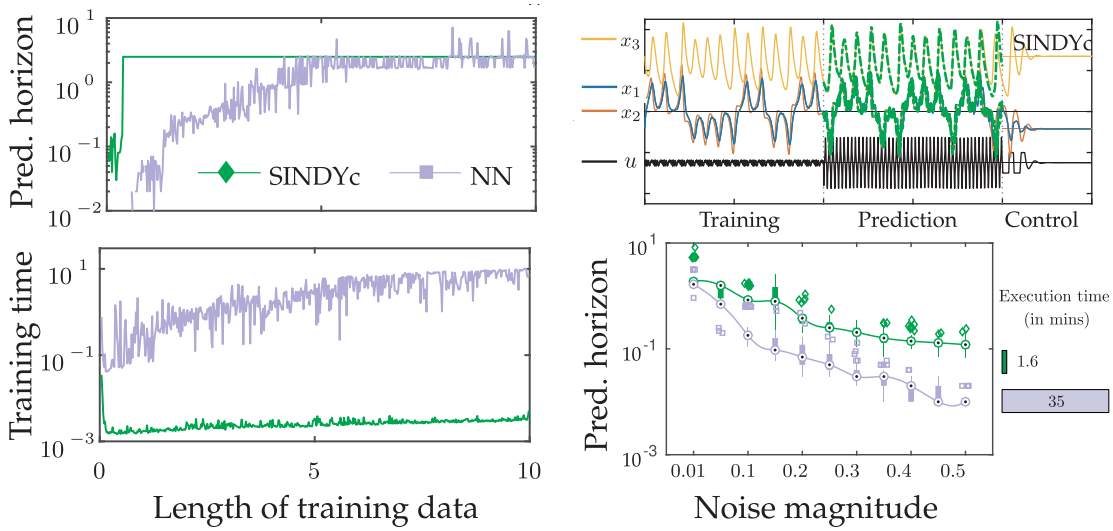

Figure 10.4 Model predictive control of the Lorenz system with a neural network model and a SINDy model. *Reproduced with permission from Kaiser et al. [277].*

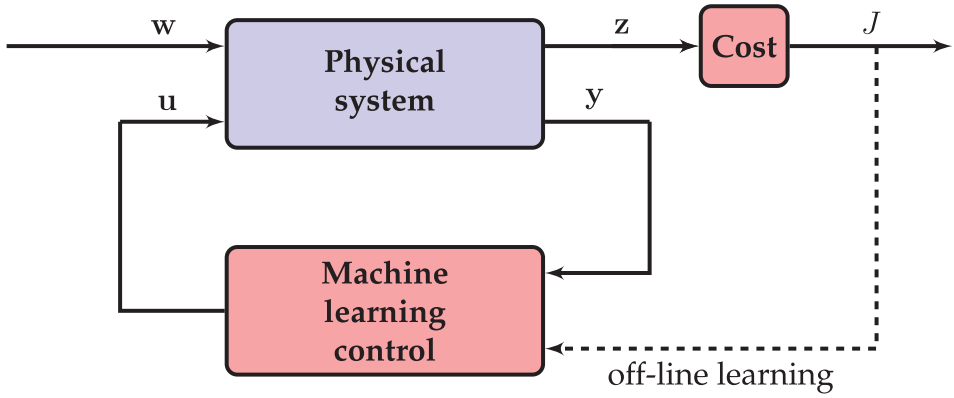

Figure 10.5 Schematic of machine learning control wrapped around a complex system using noisy sensor-based feedback. The control objective is to minimize a well-defined cost function J within the space of possible control laws. An off-line learning loop provides experiential data to train the controller. Genetic programming provides a particularly flexible algorithm to search out effective control laws. The vector $\mathbf{z}$ contains all of the information that may factor into the cost.

there is a growing emphasis on using machine learning to characterize dynamical systems. The use of machine learning to learn control laws (i.e., to determine an effective map from sensor outputs to actuation inputs), is even more recent [184]. As machine learning encompasses a broad range of high-dimensional, possibly nonlinear, optimization techniques, it is natural to apply machine learning to the control of complex, nonlinear systems. Specific machine learning methods for control include adaptive neural networks, genetic algorithms, genetic programming, and reinforcement learning. A general machine learning control architecture is shown in Fig. 10.5. Many of these machine learning algorithms are based on biological principles, such as neural networks, reinforcement learning, and evolutionary algorithms.

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

It is important to note that model-free control methodologies may be applied to numerical or experimental systems with little modification. All of these model-free methods have some sort of macroscopic objective function, typically based on sensor measurements (past and present). Some challenging real-world example objectives in different disciplines include:

Fluid dynamics: In aerodynamic applications, the goal is often some combination of drag reduction, lift increase, and noise reduction, while in pharmaceutical and chemical engineering applications the goal may involve mixing enhancement.

Finance: The goal is often to maximize profit at a given level of risk tolerance, subject to the law.

Epidemiology: The goal may be to effectively suppress a disease with constraints of sensing (e.g., blood samples, clinics, etc.) and actuation (e.g., vaccines, bed nets, etc.).

Industry: The goal of increasing productivity must be balanced with several constraints, including labor and work safety laws, as well as environmental impact, which often have significant uncertainty.

Autonomy and robotics: The goal of self-driving cars and autonomous robots is to achieve a task while interacting safely with a complex environment, including cooperating with human agents.

In the examples above, the objectives involve some minimization or maximization of a given quantity subject to some constraints. These constraints may be hard, as in the case of disease suppression on a fixed budget, or they may involve a complex multi-objective tradeoff. Often, constrained optimizations will result in solutions that live at the boundary of the constraint, which may explain why many companies operate at the fringe of legality. In all of the cases, the optimization must be performed with respect to the underlying dynamics of the system: fluids are governed by the Navier-Stokes equations, finance is governed by human behavior and economics, and disease spread is the result of a complex interaction of biology, human behavior, and geography.

These real-world control problems are extremely challenging for a number of reasons. They are high-dimensional and strongly nonlinear, often with millions or billions of degrees of freedom that evolve according to possibly unknown nonlinear interactions. In addition, it may be exceedingly expensive or infeasible to run different scenarios for system identification; for example, there are serious ethical issues associated with testing different vaccination strategies when human lives are at stake.

Increasingly, challenging optimization problems are being solved with machine learning, leveraging the availability of vast and increasing quantities of data. Many of the recent successes have been on static data (e.g., image classification, speech recognition, etc.), and marketing tasks (e.g., online sales and ad placement). However, current efforts are applying machine learning to analyze and control complex systems with dynamics, with the potential to revolutionize our ability to interact with and manipulate these systems.

The following sections describe a handful of powerful learning techniques that are being widely applied to control complex systems where models may be unavailable. Note that the relative importance of the following methods are not proportional to the amount of space dedicated.

Reinforcement Learning

Reinforcement learning (RL) is an important discipline at the intersection of machine learning and control [507], and it is currently being used heavily by companies such as Google for generalized artificial intelligence, autonomous robots, and self-driving cars. In reinforcement learning, a control policy is refined over time, with improved performance achieved through experience. The most common framework for RL is the Markov decision process, where the dynamics of the system and the control policy are described in a probabilistic setting, so that stochasticity is built into the state dynamics and the actuation strategy. In this way, control policies are probabilistic, promoting a balance of optimization and exploration. Reinforcement learning is closely related to optimal control, although it may be formulated in a more general framework.

Reinforcement learning may be viewed as partially supervised, since it is not always known immediately if a control action was effective or not. In RL, a control policy is enacted by an *agent*, and this agent may only receive partial information about the effectiveness of their control strategy. For example, when learning to play a game like tic-tac-toe or chess, it is not clear if a specific intermediate move is responsible for winning or losing. The player receives binary feedback at the end of the game as to whether or not they win or lose. A major challenge that is addressed by RL is the development of a *value function*, also known as a quality function Q, that describes the value or quality of being in a particular state and making a particular control policy decision. Over time, the agent learns and refines this Q function, improving their ability to make good decisions. In the example of chess, an expert player begins to have intuition for good strategy based on board position, which is a complex value function over an extremely high-dimensional state space (i.e., the space of all possible board configurations). Q-learning is a model-free reinforcement learning strategy, where the value function is learned from experience. Recently, deep learning has been leveraged to dramatically improve the Q-learning process in situations where data is readily available [336, 385, 386, 384]. For example, the Google DeepMind algorithm has been able to master many classic Atari video games and has recently defeated the best players in the world at Go. We leave a more in-depth discussion of reinforcement learning for other books, but emphasize its importance in the growing field of machine learning control.

Iterative Learning Control

Iterative learning control (ILC) [5, 67, 83, 130, 343, 390] is a widely used technique that learns how to refine and optimize repetitive control tasks, such as the motion of a robot arm on a manufacturing line, where the robot arm will be repeating the same motion thousands of times. In contrast to the feedback control methods from Chapter 8 which adjust the actuation signal in real-time based on measurements, ILC refines the entire open-loop actuation sequence after each iteration of a prescribed task. The refinement process may be as simple as a proportional correction based on the measured error, or may involve a more sophisticated update rule. Iterative learning control does not require one to know the system equations and has performance guarantees for linear systems. ILC is therefore a mainstay in industrial control for repetitive tasks in a well-controlled environment, such as trajectory control of a robot arm or printer-head control in additive manufacturing.

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

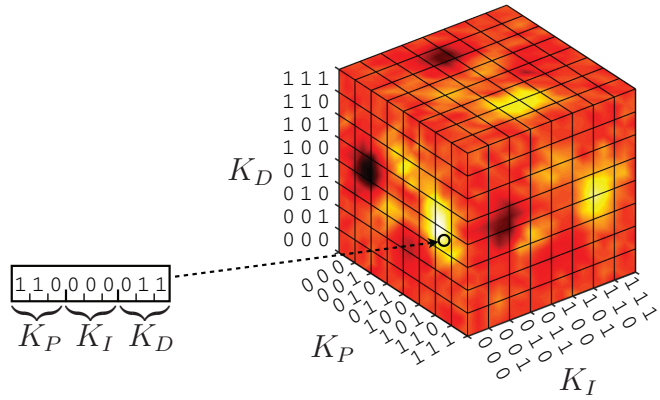

Figure 10.6 Depiction of parameter cube for PID control. The genetic algorithm represents a given parameter value as a *genetic sequence* that concatenates the various parameters. In this example, the parameters are expressed in binary representation that is scaled so that **000** is the minimum bound and **111** is the upper bound. Color indicates the cost associated with each parameter value.

Genetic Algorithms

The genetic algorithm (GA) is one of the earliest and simplest algorithms for parameter optimization, based on the biological principle of optimization through natural selection and fitness [250, 146, 210]. GA is frequently used to tune and adapt the parameters of a controller. In GA, a population comprised of many system realizations with different parameter values compete to minimize a given cost function, and successful parameter values are propagated to future generations through a set of *genetic* rules. The parameters a system are generally represented by a binary sequence, as shown in Fig. 10.6 for a PID control system with three parameters, given by the three control gains K_P, K_I, and K_D. Next, a number of realizations with different parameter values, called *individuals*, are initialized in a population and their performance is evaluated and compared on a given well-defined task. Successful individuals with a lower cost have a higher probability of being selected to advance to the next generation, according to the following genetic operations:

Elitism (optional): A set number of the most fit individuals with the best performance are advanced directly to the next generation.

Replication: An individual is selected to advance to the next generation.

Crossover: Two individuals are selected to exchange a portion of their code and then advance to the next generation; crossover serves to exploit and enhance existing successful strategies.

Mutation: An individual is selected to have a portion of its code modified with new values; mutation promotes diversity and serves to increase the exploration of parameter space.

For the replication, crossover, and mutation operations, individuals are randomly selected to advance to the next generation with the probability of selection increasing with fitness. The genetic operations are illustrated for the PID control example in Fig. 10.7. These generations are evolved until the fitness of the top individuals converges or other stopping criteria are met.

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

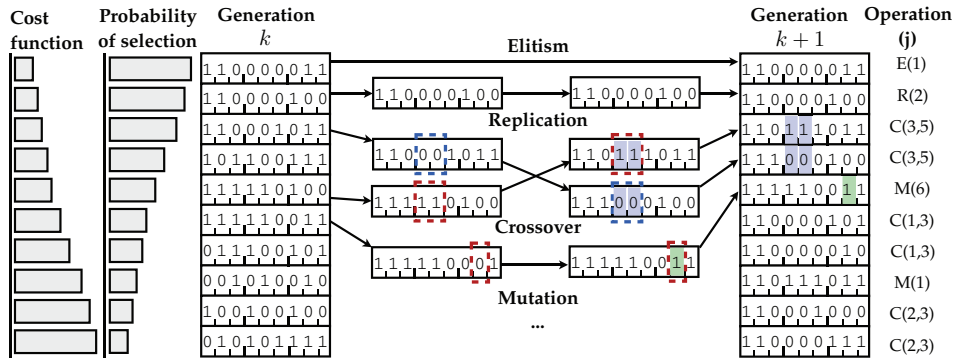

Figure 10.7 Schematic illustrating evolution in a genetic algorithm. The individuals in generation k are each evaluated and ranked in ascending order based on their cost function, which is inversely proportional to their probability of selection for genetic operations. Then, individuals are chosen based on this weighted probability for advancement to generation $k + 1$ using the four operations: elitism, replication, crossover, and mutation. This forms generation $k + 1$, and the sequence is repeated until the population statistics converges or another suitable stopping criterion is reached.

Genetic algorithms are generally used to find nearly globally optimal parameter values, as they are capable of exploring and exploiting local wells in the cost function. GA provides a middle ground between a brute-force search and a convex optimization, and is an alternative to expensive Monte Carlo sampling, which does not scale to high-dimensional parameter spaces. However, there is no guarantee that genetic algorithms will converge to a globally optimal solution. There are also a number of hyper-parameters that may affect performance, including the size of the populations, number of generations, and relative selection rates of the various genetic operations.

Genetic algorithms have been widely used for optimization and control in nonlinear systems [184]. For example, GA was used for parameter tuning in open loop control [394], with applications in jet mixing [304], combustion processes [101], wake control [431, 192], and drag reduction [201]. GA has also been employed to tune an $\mathcal{H}_\infty$ controller in a combustion experiment [233].

Genetic Programming

Genetic programming (GP) [307, 306] is a powerful generalization of genetic algorithms that simultaneously optimizes both the structure and parameters of an input–output map. Recently, genetic programming has also been used to obtain control laws that map sensor outputs to actuation inputs, as shown in Fig. 10.8. The function tree representation in GP is quite flexible, enabling the encoding of complex functions of the sensor signal $\mathbf{y}$ through a recursive tree structure. Each branch is a signal, and the merging points are mathematical operations. Sensors and constants are the leaves, and the overall control signal u is the root. The genetic operations of crossover, mutation, and replication are shown schematically in Fig. 10.9. This framework is readily generalized to include delay coordinates and temporal filters, as discussed in Duriez et al. [167].

Genetic programming has been recently used with impressive results in turbulence control experiments, led by Bernd Noack and collaborators [403, 417, 199, 168, 169, 416].

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

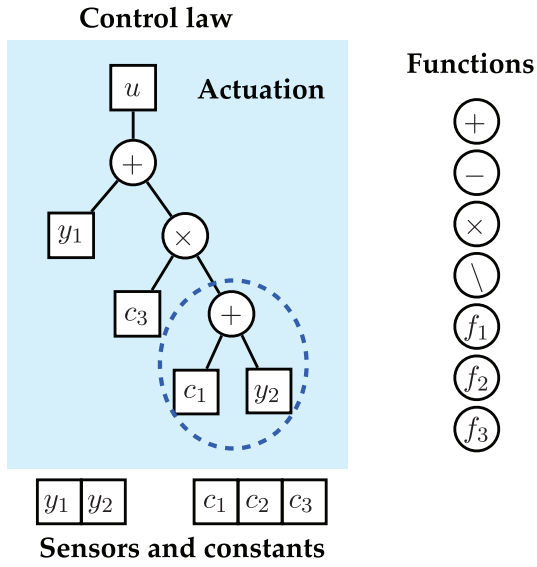

Figure 10.8 Illustration of function tree used to represent the control law **u** in genetic programming control.

This provides a new paradigm of control for strongly nonlinear systems, where it is now possible to identify the structure of nonlinear control laws. Genetic programming control is particularly well-suited to experiments where it is possible to rapidly evaluate a given control law, enabling the testing of hundreds or thousands of individuals in a short amount of time. Current demonstrations of genetic programming control in turbulence have produced several macroscopic behaviors, such as drag reduction and mixing enhancement, in an array of flow configurations. Specific flows include the mixing layer [417, 416, 168, 169], the backward facing step [199, 169], and a turbulent separated boundary layer [169].

Example: Genetic Algorithm to Tune PID Control

In this example, we will use the genetic algorithm to tune a proportional-integral-derivative (PID) controller. However, it should be noted that this is just a simple demonstration of evolutionary algorithms, and such heavy machinery is not recommended to tune a PID controller in practice, as there are far simpler techniques.

PID control is among the simplest and most widely used control architectures in industrial control systems, including for motor position and velocity control, for tuning of various sub-systems in an automobile, and for the pressure and temperature controls in modern espresso machines, to name only a few of the myriad applications. As its name suggests, PID control additively combines three terms to form the actuation signal, based on the error signal and its integral and derivative in time. A schematic of PID control is shown in Fig. 10.10.

In the cruise control example in Section 8.1, we saw that it was possible to reduce reference tracking error by increasing the proportional control gain K_P in the control law $u = -K_P(w_r - y)$. However, increasing the gain may eventually cause instability in some systems, and it will not completely eliminate the steady-state tracking error. The addition

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

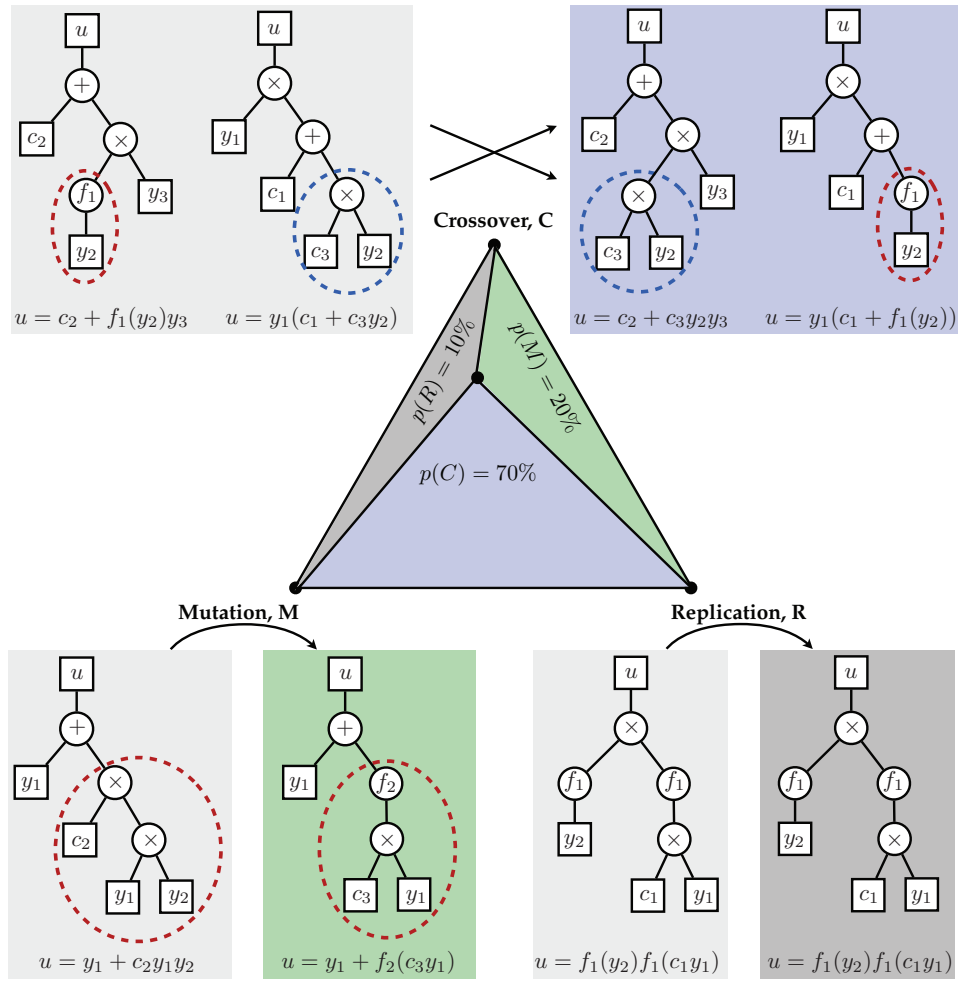

Figure 10.9 Genetic operations used to advance function trees across generations in genetic programming control. The relative selection rates of replication, crossover, and mutation are $p(R) = 0.1$, $p(C) = 0.7$, and $p(M) = 0.2$, respectively.

of an integral control term, $K_I \int_0^t (w_r - y)$ is useful to eliminate steady-state reference tracking error while alleviating the work required by the proportional term.

There are formal rules for how to choose the PID gains for various design specifications, such as fast response and minimal overshoot and ringing. In this example, we explore the use of a genetic algorithm to find effective PID gains to minimize a cost function. We use an LQR cost function

$$J = \int_0^T Q(w_r - y)^2 + Ru^2 \, d\tau$$

with $Q = 1$ and $R = 0.001$ for a step response $w_r = 1$. The system to be controlled will be given by the transfer function

$$G(s) = \frac{1}{s^4 + s}.$$

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

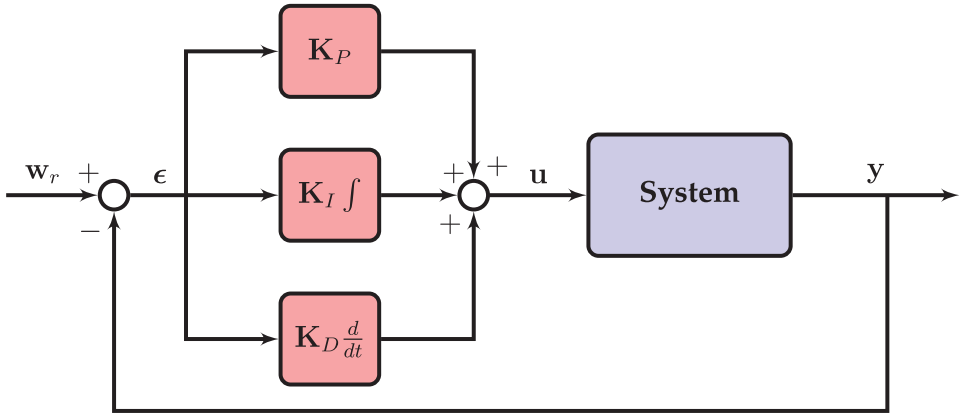

Figure 10.10 Proportional-integral-derivative (PID) control schematic. PID remains ubiquitous in industrial control.

The first step is to write a function that evaluates a given PID controller, as in Code 10.1. The three PID gains are stored in the variable **parms**.

Code 10.1 Evaluate cost function for PID controller.

```
function J = pidtest(G,dt,parms)

s = tf('s');
K = parms(1) + parms(2)/s + parms(3)*s/(1+.001*s);
Loop = series(K,G);
ClosedLoop = feedback(Loop,1);
t = 0:dt:20;
[y,t] = step(ClosedLoop,t);

CTRLtf = K/(1+K*G);
u = lsim(K,1-y,t);
```

Next, it is relatively simple to use a genetic algorithm to optimize the PID control gains, as in Code 10.2. In this example, we run the GA for 10 generations, with a population size of 25 individuals per generation.

Code 10.2 Genetic algorithm to tune PID controller.

```
dt = 0.001;
PopSize = 25;
MaxGenerations = 10;
s = tf('s');
G = 1/(s*(s*s+s+1));

options = optimoptions(@ga,'PopulationSize',PopSize,'
    MaxGenerations',MaxGenerations,'OutputFcn',@myfun);
[x,fval] = ga(@(K)pidtest(G,dt,K),3,-eye(3),zeros(3,1)
    ,[],[],[],[],[],options);
```

The results from intermediate generations are saved using the custom output function in Code 10.3.

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

Code 10.3 Special output function to save generations.

```
function [state,opts,optchanged]=myfun(opts,state,flag)
persistent history
persistent cost
optchanged = false;

switch flag
 case 'init'
        history(:,:,1) = state.Population;
        cost(:,1) = state.Score;
    case {'iter','interrupt'}
        ss = size(history,3);
        history(:,:,ss+1) = state.Population;
        cost(:,ss+1) = state.Score;
    case 'done'
        ss = size(history,3);
        history(:,:,ss+1) = state.Population;
        cost(:,ss+1) = state.Score;
        save history.mat history cost
end
```

The evolution of the cost function across various generations is shown in Fig. 10.11. As the generations progress, the cost function steadily decreases. The individual gains are shown in Fig. 10.12, with redder dots corresponding to early generations and bluer generations corresponding to later generations. As the genetic algorithm progresses, the PID gains begin to cluster around the optimal solution (black circle).

Fig. 10.13 shows the output in response to the PID controllers from the first generation. It is clear from this plot that many of the controllers fail to stabilize the system, resulting in large deviations in y. In contrast, Fig. 10.14 shows the output in response to the PID controllers from the last generation. Overall, these controllers are more effective at producing a stable step response.

The best controllers from each generation are shown in Fig. 10.15. In this plot, the controllers from early generations are redder, while the controllers from later generations

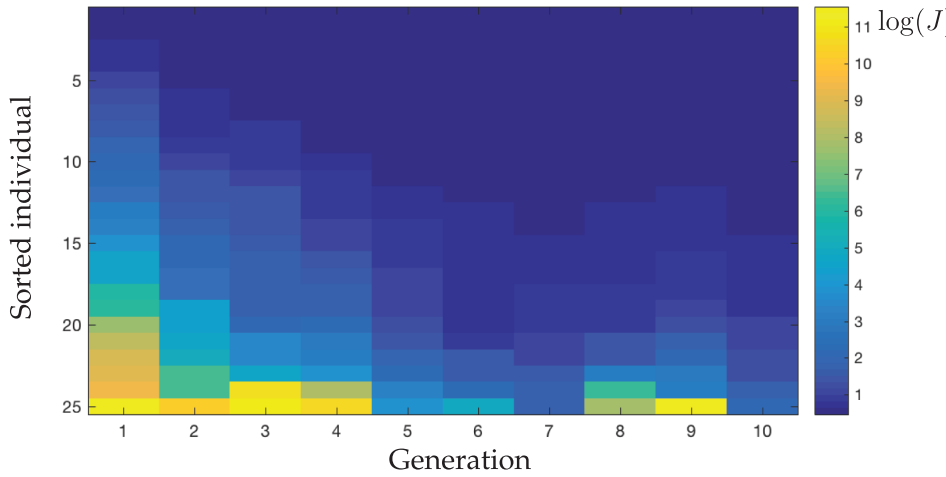

Figure 10.11 Cost function across generations, as GA optimizes PID gains.

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

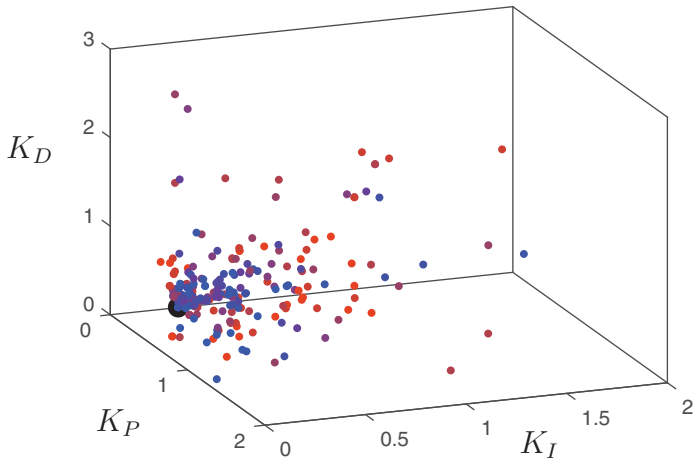

Figure 10.12 PID gains generated from genetic algorithm. Red points correspond to early generations while blue points correspond to later generations. The black point is the best individual found by GA.

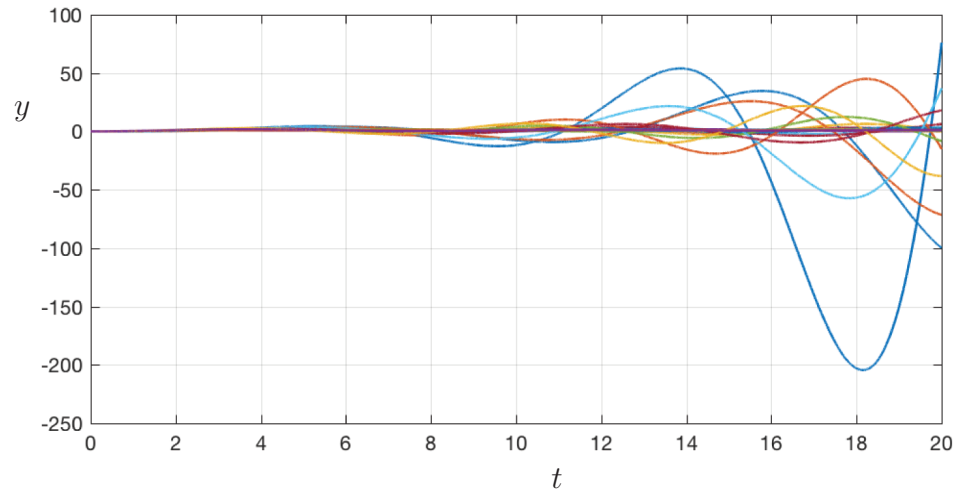

Figure 10.13 PID controller response from first generation of genetic algorithm.

are bluer. As the GA progresses, the controller is able to minimizes output oscillations and achieves fast rise time.

10.3 Adaptive Extremum-Seeking Control

Although there are many powerful techniques for model-based control design, there are also a number of drawbacks. First, in many systems, there may not be access to a model, or the model may not be suitable for control (i.e., there may be strong nonlinearities or the model may be represented in a nontraditional form). Next, even after an attractor has been identified and the dynamics characterized, control may invalidate this model by modifying the attractor, giving rise to new and uncharacterized dynamics. The obvious exception is

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

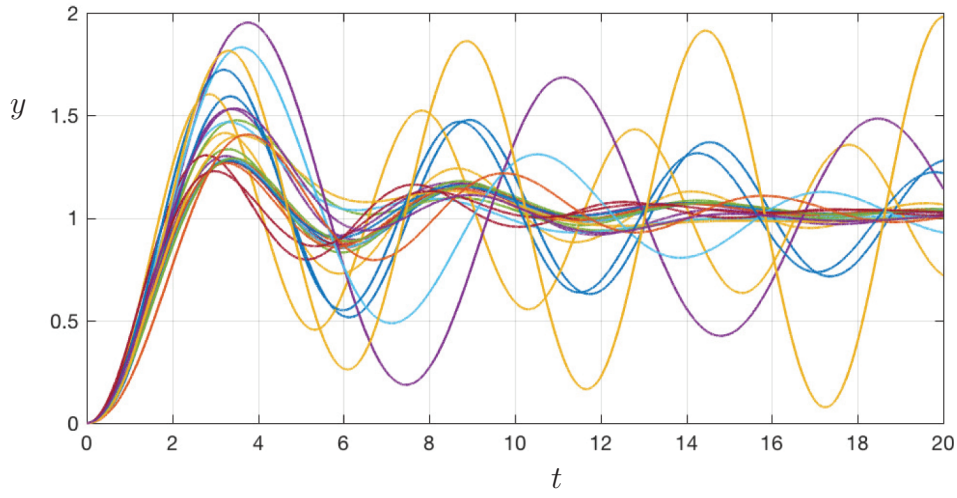

Figure 10.14 PID controller response from last generation of genetic algorithm.

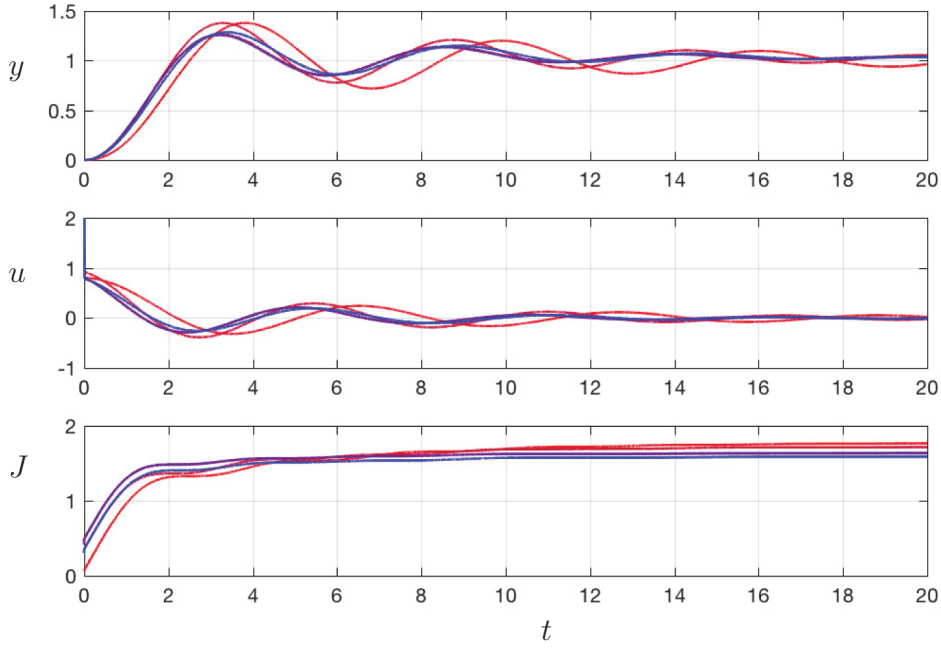

Figure 10.15 Best PID controllers from each generation. Red trajectories are from early generations, and blue trajectories correspond to the last generation.

stabilizing a fixed point or a periodic orbit, in which case effective control keeps the system in a neighborhood where the linearized model remains accurate. Finally, there may be slow changes to the system that modify the underlying dynamics, and it may be difficult to measure and model these effects.

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

The field of *adaptive* control broadly addresses these challenges, by allowing the control law the flexibility to modify its action based on the changing dynamics of a system. Extremum-seeking control (ESC) [312, 19] is a particularly attractive form of adaptive control for complex systems because it does not rely on an underlying model and it has guaranteed convergence and stability under a set of well-defined conditions. Extremum-seeking may be used to track local maxima of an objective function, despite disturbances, varying system parameters, and nonlinearities. Adaptive control may be implemented for in-time control or used for slow tuning of parameters in a working controller.

Extremum-seeking control may be thought of as an advanced *perturb-and-observe* method, whereby a sinusoidal perturbation is additively injected in the actuation signal and used to estimate the gradient of an objective function J that should be maximized or minimized. The objective function is generally computed based on sensor measurements of the system, although it ultimately depends on the internal dynamics and the choice of the input signal. In extremum-seeking, the control variable $\mathbf{u}$ may refer either to the actuation signal or a set of parameters that describe the control behavior, such as the frequency of periodic forcing or the gains in a PID controller.

The extremum-seeking control architecture is shown in Fig. 10.16. This schematic depicts ESC for a scalar input u, although the methods readily generalize for vector-valued inputs $\mathbf{u}$. A convex objective function $J(u)$, is shown in Fig. 10.17 for static plant dynamics (i.e., for $y = u$). The extremum-seeking controller uses an input perturbation to estimate the gradient of the objective function J and steer the mean actuation signal towards the optimizing value.

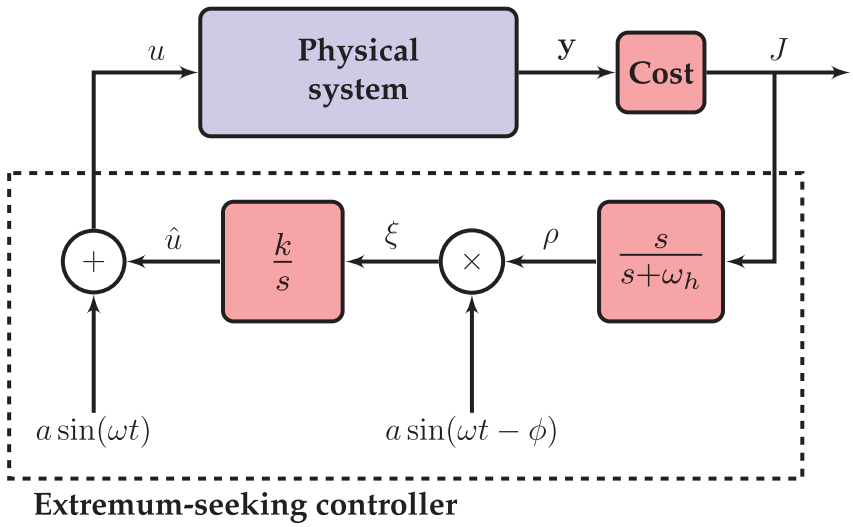

Figure 10.16 Schematic illustrating an extremum-seeking controller. A sinusoidal perturbation is added to the best guess of the input $\hat{u}$, and it passes through the plant, resulting in a sinusoidal output perturbation that may be observed in the sensor signal $\mathbf{y}$ and the cost J. The high-pass filter results in a zero-mean output perturbation, which is then multiplied (demodulated) by the same input perturbation resulting in the signal ξ. This demodulated signal is finally integrated into the best guess $\hat{u}$ for the optimizing input u.

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

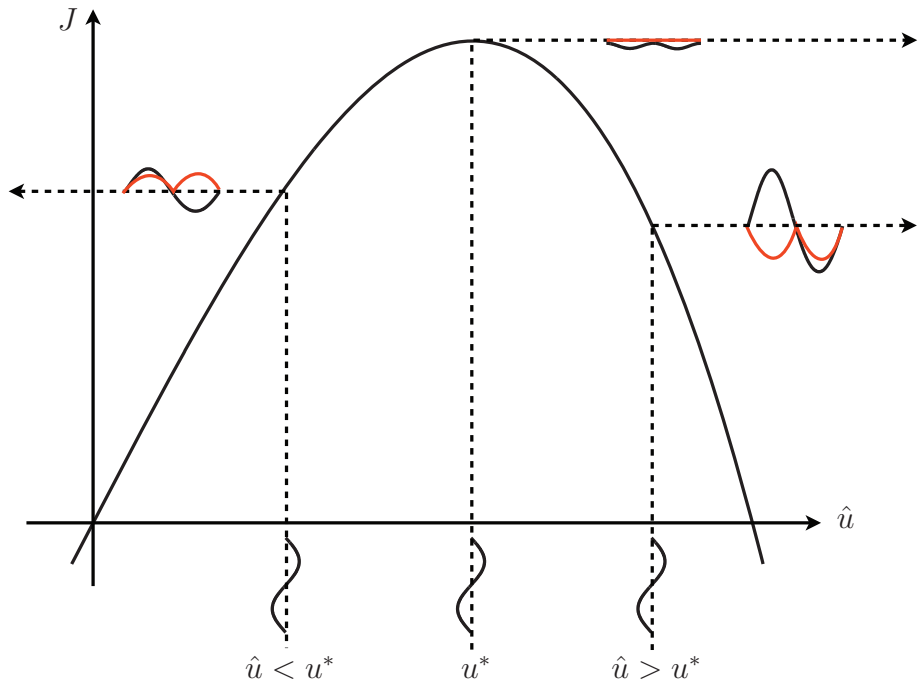

Figure 10.17 Schematic illustrating extremum-seeking control on for a static objective function $J(u)$. The output perturbation (red) is in phase when the input is left of the peak value (i.e. $u < u^*$) and out of phase when the input is to the right of the peak (i.e. $u > u^*$). Thus, integrating the product of input and output sinusoids moves $\hat{u}$ towards u^*.

Three distinct time-scales are relevant for extremum-seeking control:

1. slow – external disturbances and parameter variation;
2. medium – perturbation frequency ω;
3. fast – system dynamics.

In many systems, the internal system dynamics evolve on a fast time-scale. For example, turbulent fluctuations may equilibrate rapidly compared to actuation time-scales. In optical systems, such as a fiber laser [93], the dynamics of light inside the fiber are extremely fast compared to the time-scales of actuation.

In extremum-seeking control, a sinusoidal perturbation is added to the estimate of the input that maximizes the objective function, $\hat{u}$:

$$u = \hat{u} + a\sin(\omega t). \tag{10.23}$$

This input perturbation passes through the system dynamics and output, resulting in an objective function J that varies sinusoidally about some mean value, as shown in Fig. 10.17. The output J is high-pass filtered to remove the mean (DC component), resulting in the oscillatory signal ρ. A simple high-pass filter is represented in the frequency domain as

$$\frac{s}{s+\omega_h} \tag{10.24}$$

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

where s is the Laplace variable, and ω_h is the filter frequency. The high-pass filter is chosen to pass the perturbation frequency ω. The high-pass filtered output is then multiplied by the input sinusoid, possibly with a phase shift ϕ, resulting in the *demodulated* signal ξ:

$$\xi = a \sin(\omega t - \phi)\rho. \tag{10.25}$$

This signal ξ is mostly positive if the input u is to the left of the optimal value u^* and it is mostly negative if u is to the right of the optimal value u^*, shown as red curves in Fig. 10.17. Thus, the demodulated signal ξ is integrated into $\hat{u}$, the best estimate of the optimizing value

$$\frac{d}{dt}\hat{u} = k\,\xi, \tag{10.26}$$

so that the system estimate $\hat{u}$ is steered towards the optimal input u^*. Here, k is an integral gain, which determines how aggressively the actuation climbs gradients in J.

Roughly speaking, the demodulated signal ξ measures gradients in the objective function, so that the algorithm climbs to the optimum more rapidly when the gradient is larger. This is simple to see for constant plant dynamics, where J is simply a function of the input $J(u) = J(\hat{u} + a\sin(\omega t))$. Expanding $J(u)$ in the perturbation amplitude a, which is assumed to be small, yields:

$$J(u) = J(\hat{u} + a\sin(\omega t)) \tag{10.27a}$$

$$= J(\hat{u}) + \left.\frac{\partial J}{\partial u}\right|_{u=\hat{u}} \cdot a\sin(\omega t) + \mathcal{O}(a^2). \tag{10.27b}$$

The leading-order term in the high-pass filtered signal is $\rho \approx \partial J/\partial u|_{u=\hat{u}} \cdot a\sin(\omega t)$. Averaging $\xi = a\sin(\omega t - \phi)\rho$ over one period yields:

$$\xi_{\text{avg}} = \frac{\omega}{2\pi}\int_0^{2\pi/\omega} a\sin(\omega t - \phi)\rho\, dt \tag{10.28a}$$

$$= \frac{\omega}{2\pi}\int_0^{2\pi/\omega} \left.\frac{\partial J}{\partial u}\right|_{u=\hat{u}} a^2 \sin(\omega t - \phi)\sin(\omega t)\, dt \tag{10.28b}$$

$$= \frac{a^2}{2}\left.\frac{\partial J}{\partial u}\right|_{u=\hat{u}} \cos(\phi). \tag{10.28c}$$

Thus, for the case of trivial plant dynamics, the average signal ξ_{avg} is proportional to the gradient of the objective function J with respect to the input u.

In general, extremum-seeking control may be applied to systems with nonlinear dynamics relating the input u to the outputs $\mathbf{y}$ that act on a faster timescale than the perturbation ω. Thus, J may be time-varying, which complicates the simplistic averaging analysis above. The general case of extremum-seeking control of nonlinear systems is analyzed by Krstić and Wang in [312], where they develop powerful stability guarantees based on a separation of timescales and a singular perturbation analysis. The basic algorithm may also be modified to add a phase ϕ to the sinusoidal input perturbation in (10.25). In [312], there was an additional low-pass filter $\omega_l/(s+\omega_l)$ placed before the integrator to extract the DC component of the demodulated signal ξ. There is also an extension to extremum-seeking called slope-seeking, where a specific slope is sought [19] instead of the standard zero slope corresponding to a maximum or minimum. Slope-seeking is preferred when there is not an extremum, as in the case when control inputs saturate. Extremum-seeking is often

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

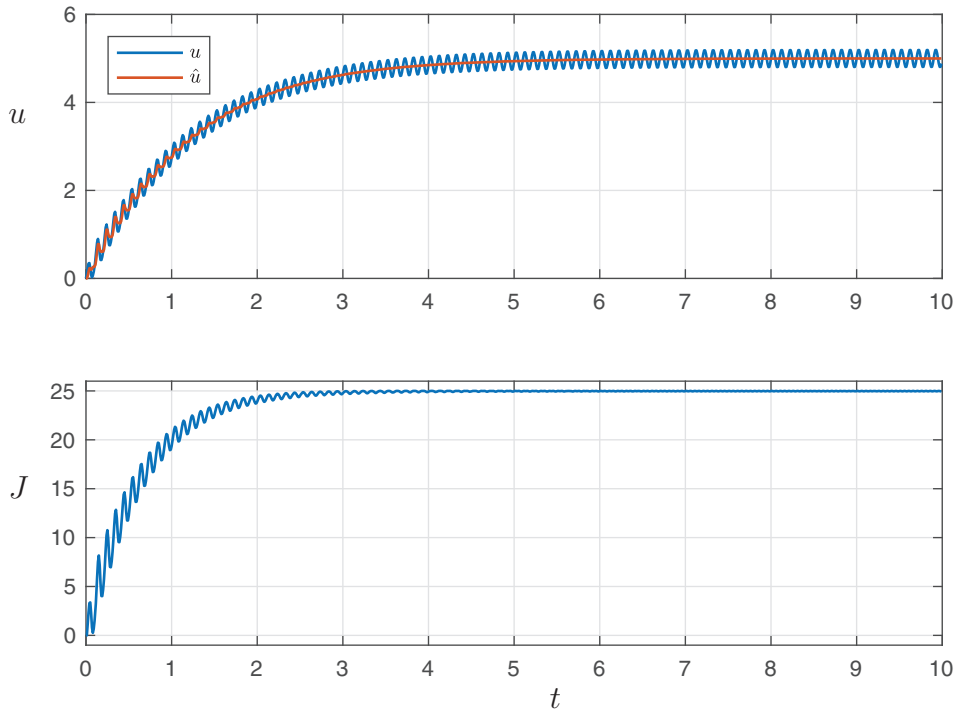

Figure 10.18 Extremum-seeking control response for cost function in (10.29).

used for frequency selection and slope-seeking is used for amplitude selection when tuning an open-loop periodic forcing.

It is important to note that extremum-seeking control will only find local maxima of the objective function, and there are no guarantees that this will correspond to a global maxima. Thus, it is important to start with a good initial condition for the optimization. In a number of studies, extremum-seeking control is used in conjunction with other global optimization techniques, such as a genetic algorithm, or sparse representation for classification [191, 99].

Simple Example of Extremum-Seeking Control

Here we consider a simple application of extremum-seeking control to find the maximum of a static quadratic cost function,

$$J(u) = 25 - (5 - u)^2. \tag{10.29}$$

This function has a single global maxima at $u^* = 5$. Starting at $u = 0$, we apply extremum-seeking control with a perturbation frequency of $\omega = 10\,\text{Hz}$ and an amplitude of $a = 0.2$. Fig. 10.18 shows the controller response and the rapid tracking of the optimal value $u^* = 5$. Code 10.4 shows how extremum-seeking may be implemented using a simple Butterworth high-pass filter.

Notice that when the gradient of the cost function is larger (i.e., closer to $u = 0$), the oscillations in J are larger, and the controller climbs more rapidly. When the input u gets close to the optimum value at $u^* = 5$, even though the input perturbation has the same

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

amplitude a, the output perturbation is nearly zero (on the order of a^2), since the quadratic cost function is flat near the peak. Thus we achieve fast tracking far away from the optimum value and small deviations near the peak.

Code 10.4 Extremum-seeking control code.

```
J = @(u,t)(25-(5-(u)).^2);
y0 = J(0,0);  % u = 0

% Extremum Seeking Control Parameters
freq = 10*2*pi; % sample frequency
dt = 1/freq;
T = 10; % total period of simulation (in seconds)
A = .2;  % amplitude
omega = 10*2*pi; % 10 Hz
phase = 0;
K = 5;   % integration gain

% High pass filter (Butterworth filter)
butterorder=1;
butterfreq=2;  % in Hz for 'high'
[b,a] = butter(butterorder,butterfreq*dt*2,'high')
ys = zeros(1,butterorder+1)+y0;
HPF=zeros(1,butterorder+1);

uhat=u;
for i=1:T/dt
    t = (i-1)*dt;
    yvals(i)=J(u,t);

    for k=1:butterorder
        ys(k) = ys(k+1);
        HPF(k) = HPF(k+1);
    end
    ys(butterorder+1) = yvals(i);
    HPFnew = 0;
    for k=1:butterorder+1
        HPFnew = HPFnew + b(k)*ys(butterorder+2-k);
    end
    for k=2:butterorder+1
        HPFnew = HPFnew - a(k)*HPF(butterorder+2-k);
    end
    HPF(butterorder+1) = HPFnew;

    xi = HPFnew*sin(omega*t + phase);
    uhat = uhat + xi*K*dt;
    u = uhat + A*sin(omega*t + phase);
    uhats(i) = uhat;
    uvals(i) = u;
end
```

To see the ability of extremum-seeking control to handle varying system parameters, consider the time-dependent cost function given by

$$J(u) = 25 - (5 - u - \sin(t))^2. \tag{10.30}$$

The varying parameters, which oscillate at $1/2\pi$ Hz, may be consider slow compared with the perturbation frequency 10 Hz. The response of extremum-seeking control for this slowly varying system is shown in Fig. 10.19. In this response, the actuation signal is able

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

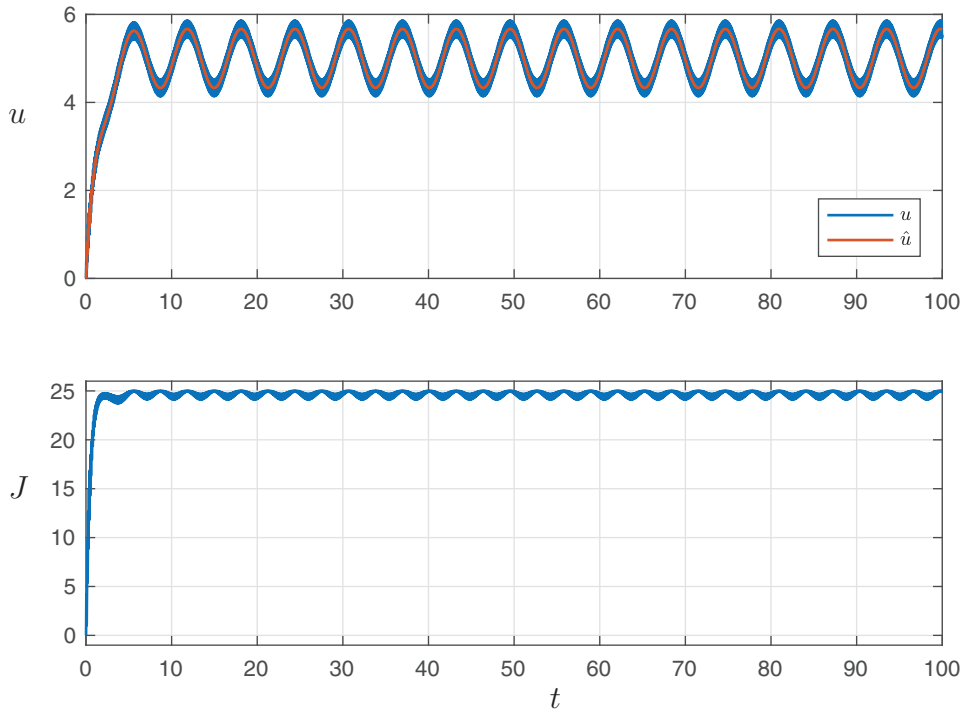

Figure 10.19 Extremum-seeking control response with a slowly changing cost function $J(u, t)$.

to maintain good performance by oscillating back and forth to approximately track the oscillating optimal u^*, which oscillates between 4 and 6. The output function J remains close to the optimal value of 25, despite the unknown varying parameter.

Challenging Example of Extremum-Seeking Control

Here we consider an example inspired by a challenging benchmark problem in Section 1.3 of [19]. This system has a time-varying objective function $J(t)$ and dynamics with a right-half plane zero, making it difficult to control.

In one formulation of extremum-seeking [133, 19], there are additional guidelines for designing the controller if the plant can be split into three blocks that define the input dynamics, a time-varying objective function with no internal dynamics, and the output dynamics, as shown in Fig. 10.20. In this case, there are procedures to design the high-pass filter and integrator blocks.

In this example, the objective function is given by

$$J(\theta) = .05\delta(t - 10) + (\theta - \theta^*(t))^2,$$

where δ is the Dirac delta function, and the optimal value $\theta^*(t)$ is given by

$$\theta^* = .01 + .001t.$$

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

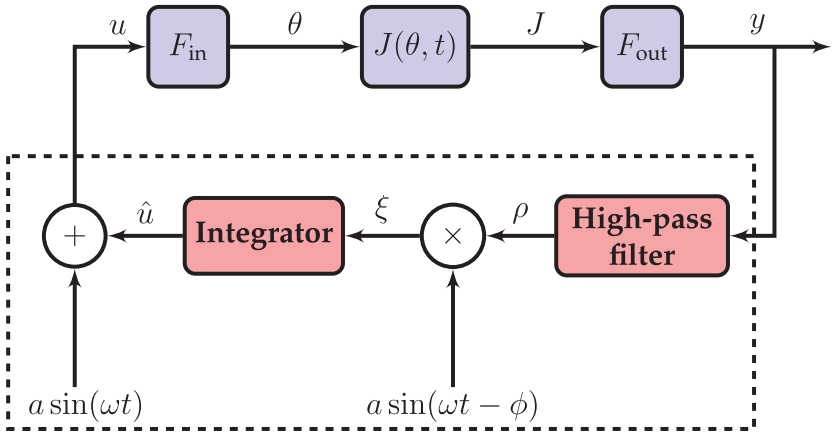

Figure 10.20 Schematic of a specific extremum-seeking control architecture that benefits from a wealth of design techniques [133, 19].

The optimal objective is given by $J^* = .05\delta(t - 10)$. The input and output dynamics are taken from the example in [19], and are given by

$$F_{\text{in}}(s) = \frac{s-1}{(s+2)(s+1)}, \qquad F_{\text{out}}(s) = \frac{1}{s+1}.$$

Using the design procedure in [19], one arrives at the high-pass filter $s/(s+5)$ and an integrator-like block given by $50(s-4)/(s-.01)$. In addition, a perturbation with $\omega = 5$ and $a = 0.05$ is used, and the demodulating perturbation is phase-shifted by $\phi = .7955$; this phase is obtained by evaluating the input function F_{in} at $i\omega$. The response of this controller is shown in Fig. 10.21, along with the Simulink implementation in Fig. 10.22. The controller is able to accurately track the optimizing input, despite additive sensor noise.

Applications of Extremum-Seeking Control

Because of the lack of assumptions and ease of implementation, extremum-seeking control has been widely applied to a number of complex systems. Although ESC is generally applicable for in-time control of dynamical systems, it is also widely used as an online optimization algorithm that can adapt to slow changes and disturbances. Among the many uses of extremum-seeking control, here we highlight only a few.

Extremum-seeking has been used widely for maximum power point tracking algorithms in photovoltaics [331, 178, 75, 97], and wind energy conversion [395]. In the case of photovoltaics, the voltage or current ripple in power converters due to pulse-width modulation is used for the perturbation signal, and in the case of wind, turbulence is used as the perturbation. Atmospheric turbulent fluctuations were also used as the perturbation signal for the optimization of aircraft control [309]; in this example it is infeasible to add a perturbation signal to the aircraft control surfaces, and a natural perturbation is required. ESC has also been used in optics and electronics for laser pulse shaping [450], tuning high-gain fiber lasers [93, 99], and for beam control in a reconfigurable holographic metamaterial antenna array [265]. Other applications include formation flight optimization [60],

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

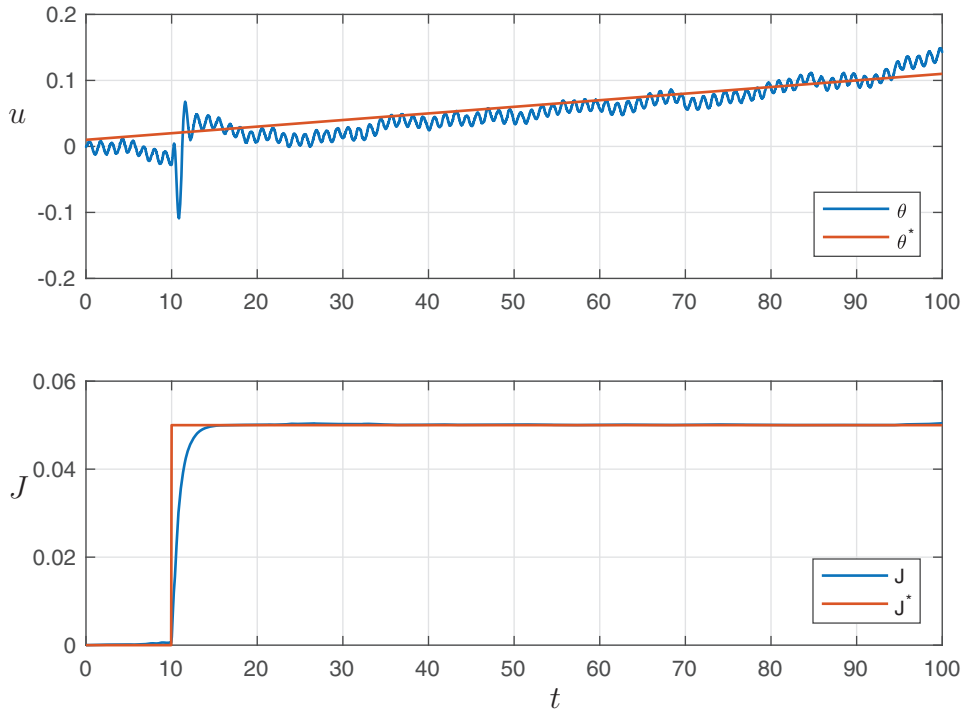

Figure 10.21 Extremum-seeking control response for a challenging test system with a right-half plane zero, inspired by [19].

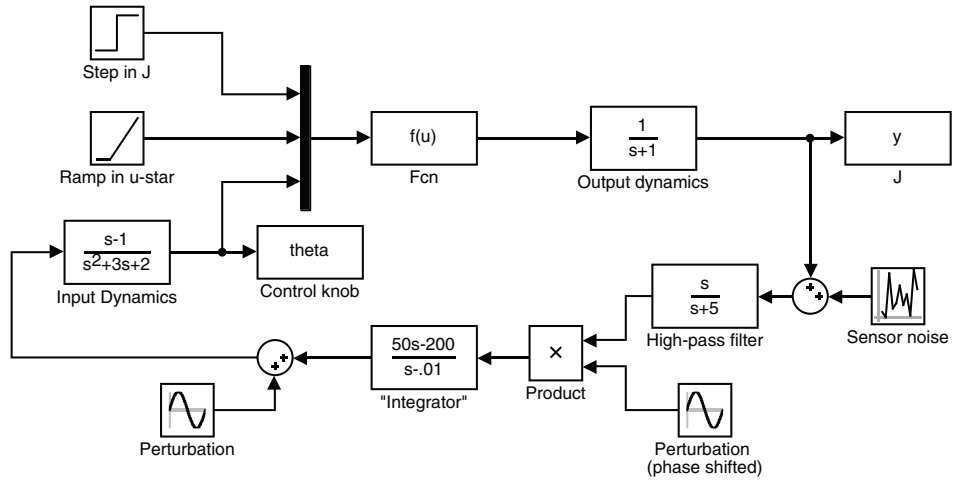

Figure 10.22 Simulink model for extremum-seeking controller used in Fig. 10.21.

bioreactors [546], PID [289] and PI [311] tuning, active braking systems [568], and control of Tokamaks [413].

Extremum-seeking has also been broadly applied in turbulent flow control. Despite the ability to control dynamics in-time with ESC, it is often used as a slow feedback

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press

optimization to tune the parameters of a working open-loop controller. This slow feedback has many benefits, such as maintaining performance despite slow changes to environmental conditions. Extremum-seeking has been used to control an axial flow compressor [547], to reduce drag over a bluff-body in an experiment [45, 46] using a rotating cylinder on the upper trailing edge of the rear surface, and for separation control in a high-lift airfoil configuration [47] using pressure sensors and pulsed jets on the leading edge of a single-slotted flap. There have also been impressive industrial-scale uses of extremum-seeking control, for example to control thermoacoustic modes across a range of frequencies in a 4 MW gas turbine combustor [37, 35]. It has also been utilized for separation control in a planar diffusor that is fully turbulent and stalled [36], and to control jet noise [375].

There are numerous extensions to extremum-seeking that improve performance. For example, extended Kalman filters were used as the filters in [202] to control thermoacoustic instabilities in a combustor experiment, reducing pressure fluctuations by nearly 40 dB. Kalman filters were also used with ESC to reduce the flow separation and increase the pressure ratio in a high-pressure axial fan using an injected pulsed air stream [553]. Including the Kalman filter improved the controller bandwidth by a factor of 10 over traditional ESC.

Suggested Reading

Texts

(1) **Reinforcement learning: An introduction**, by R. S. Sutton and A. G. Barto, 1998 [507].

(2) **Real-time optimization by extremum-seeking control**, by K. B. Ariyur and M. Krstić, 2003 [19].

(3) **Machine learning control: Taming nonlinear dynamics and turbulence**, by T. Duriez, S. L. Brunton, and B. R. Noack, 2016 [167].

(4) **Model predictive control**, by E. F. Camancho, C. B. Alba, 2013 [107].

Papers and Reviews

(1) **Stability of extremum seeking feedback for general nonlinear dynamic systems**, by M. Krstić and H. H. Wang, *Automatica*, 2000 [312].

(2) **Dynamic mode decomposition with control**, by J. L. Proctor, S. L. Brunton, and J. N. Kutz, *SIAM Journal on Applied Dynamical Systems*, 2016 [434].

(3) **Model predictive control: theory and practice – a survey**, by C. E. Garcia, D. M. Prett, and M. Morari, *Automatica*, 1989 [195].

(4) **Closed-loop turbulence control: Progress and challenges**, by S. L. Brunton and B. R. Noack, *Applied Mechanics Reviews*, 2015 [94].

https://doi.org/10.1017/9781108380690.011 Published online by Cambridge University Press