Chapter 4
Benchmarking MLC against linear control

"All stable processes we shall predict. All unstable processes we shall control."

- John von Neumann

We have now developed two powerful approaches to design control laws: the machine learning control (MLC) approach from Chapter 2, and the classical optimal linear control theory from Chapter 3. Both approaches have benefits and tradeoffs. Linear control theory yields concise control laws that are solutions to optimization problems, providing the best possible control laws for linear systems that are well characterized by accurate input–output models. In fact, we may view the MLC approach as an alternative optimization procedure to determine these classical controllers that generalizes naturally to nonlinear problems in a model-free context.

In this chapter, we demonstrate the use of genetic programming for MLC on linear systems where optimal control laws are known. In particular, we benchmark MLC against the linear quadratic regulator (LQR) for full-state feedback in Sec. 4.1, the Kalman filter for noisy state estimation in Sec. 4.2, and linear quadratic Gaussian (LQG) for optimal sensor-based feedback in Sec. 4.3. As an example system, we consider an unstable linear oscillator, which mimics many instabilities that occur in fluid dynamics. Next, we compare MLC with linear optimal control on systems with increasing nonlinearity in Sec. 4.4. Exercises are provided in Sec. 4.5. We conclude the chapter in Sec. 4.6 with an interview of Professor Shervin Bagheri who is a pioneer and a leading scholar in model-based closed-loop flow control.

4.1 Comparison of MLC with LQR on a linear oscillator

As discussed in Chapter 3, control design often begins with the implementation of a full-state feedback regulator. The assumption of full-state measurements will later be relaxed, necessitating dynamic estimation from limited sensor measurements.

Machine learning control using genetic programming provides a *symbolic regression* approach to optimize controller design given a well-posed cost function. Thus, it is possible to test the ability of MLC to discover the known optimal linear quadratic regulator (LQR) solution from Sec. 3.2. In particular, we consider an unstable linear oscillator with two states:

$$\begin{aligned} \frac{d}{dt}a_1 &= \sigma a_1 - \omega a_2 \\ \frac{d}{dt}a_2 &= \omega a_1 + \sigma a_2 + b. \end{aligned} \tag{4.1}$$

We assume full-state measurements, $\mathbf{s} = \mathbf{a}$, although we will relax this assumption in the following sections. In the state space form of Eq. (3.1), this system is given by the following system matrices:

$$\mathbf{A} = \begin{bmatrix} \sigma & -\omega \\ \omega & \sigma \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \tag{4.2a}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad \mathbf{D} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{4.2b}$$

For this example $\sigma = \omega = 1$, corresponding to an unstable linear growth rate. The following LQR cost function weights are used:

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R = 1. \tag{4.3}$$

The LQR optimal controller from Eq. (3.10) is given by $b = -\mathbf{K}_r\mathbf{a}$ with

$$\mathbf{K}_r = \begin{bmatrix} -4.8783 & 4.4288 \end{bmatrix}. \tag{4.4}$$

The cost is monitored as a function of time, as in Chapter 2:

$$J(t) = \int_0^t \left[\mathbf{a}^T(\tau)\,\mathbf{Q}\,\mathbf{a}(\tau) + R\,b^2(\tau) \right] d\tau. \tag{4.5}$$

If time is omitted as an argument, the converged value $\lim_{t\to\infty} J(t)$ is used.

The Matlab® implementation is shown in Code 4.1, and the closed-loop LQR response is shown in Fig. 4.1 (solid curves). Notice that there are two large periods of growth in the cost function J corresponding to large magnitude of control expenditure. Eventually, the cost function converges to a final value after the state has been stabilized and the actuation input shrinks to zero.

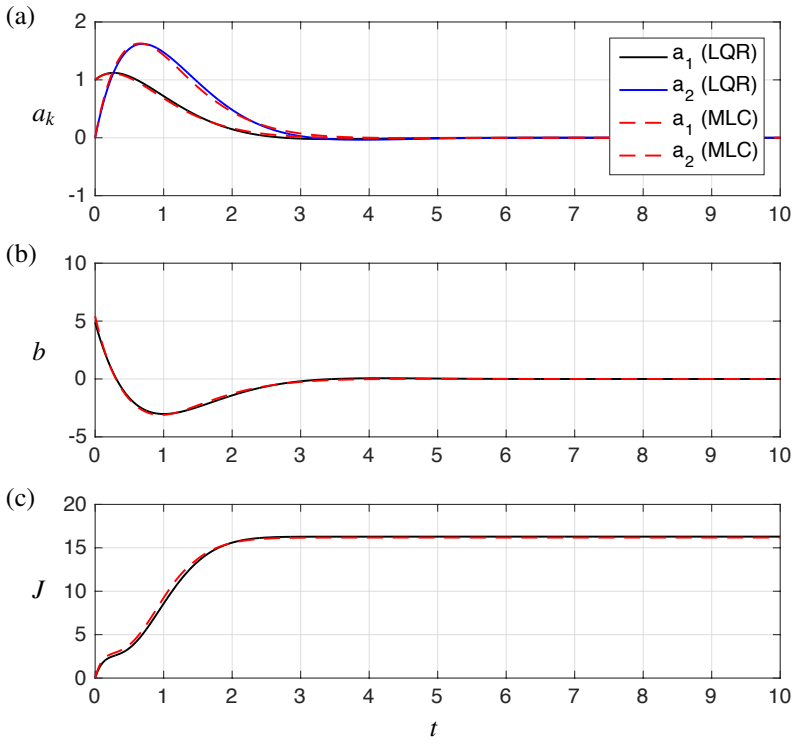

Fig. 4.1 Response of the unstable oscillator system in Eq. (4.2) with LQR control (solid lines). The MLC response is also shown (dashed red lines), and it agrees with the LQR controller. (a) The state initially grows and is eventually stabilized. (b) The actuation signal b is aggressive at the beginning until the system is driven closer to the origin. (c) The cost J (4.5) initially rises because of a large actuation expenditure b, and then continues to rise until the state decays.

An MLC controller is constructed using genetic programming with simple operations $(+, -, \times)$ and the same cost function as in Eq. (4.5). The optimal controller is simply a proportional feedback on the state measurements $\mathbf{a}$, so this presents a simple test case for the MLC architecture. The MLC controller is also shown in Fig. 4.1 (dashed curves), and the response is extremely close to that of the LQR controller.

Code 4.1 LQR implementation in Matlab®.

```
clear all, close all, clc

% Define system matrices
sigma=1, omega=1;    % Unstable oscillator parameters
A = [sigma -omega;   % Dynamics
     omega sigma];
B = [0; 1];          % Actuation on second state
C = [1 0;            % Full-state measurements
     0 1];
D = [0; 0];          % No feedthrough term
```

```
sys = ss(A,B,C,D);  % Continuous-time state-space system

% Compute LQR controller
Q = eye(2);         % State cost is 2x2 identity matrix
R = 1;              % Actuation cost is 1
[K,S,e] = lqr(A,B,Q,R);  % Optimal gain matrix K from LQR

% Simulate closed-loop system
dt = 0.001;
Acl = A-B*K;        % Closed-loop dynamics
Bcl = [0; 0];       % No input after closing-loop
sysK = ss(Acl,Bcl,C,D);  % Closed-loop system
[s,t] = initial(sysK,[1; 0],0:dt:10);  % Intial condition
    response

% Compute cost function
b = -K*s';          % Actuator signal
J(1) = 0;           % Initialize cost J=0
% For each dt, integrate cost function
for k=2:length(t)
    J(k) = J(k-1)+dt*(s(k-1,:)*Q*s(k-1,:)' + R*b(k-1)^2);
end
```

The MLC implementation is given below with the following parameters: The MLC implementation is given below with the parameters of Tab. 4.1.

Table 4.1 Main parameters used for MLC solution of LQR problem.

Parameter	N_i	N_s	N_b	P_r	P_m	P_c	N_p	N_e	Node functions
Value	1000	2	1	0.1	0.4	0.5	7	10	$+, -, \times$

The following code initializes the MLC problem, runs 50 generations, and displays the results.

```
mlc=MLC('MLC_ex_LQE_problem'); % Creates the MLC problem
mlc.go(50);                    % Runs MLC for 50 generations
mlc.show_best_individual       % Displays the results
```

The evaluation function can be displayed by using the command:

```
open('MLC_evaluator_LQR')
```

As in Sec. 2.3.2, the evaluation function implements a dynamical system by representing the time derivatives as functions of the states and the control law representing the individual. Then `ode45` is used to integrate the dynamical system and the cost function value is computed and returned. In case the numerical integration diverges, a predefined high value is returned for J.

Although MLC achieves near-optimal performance in the LQR problem, we had the advantage of full state measurements. To explore the case with limited measurements, we must explore an extension of MLC that includes temporal filters as function blocks to estimate the full state from a time-history of sensor measurements.

4.2 Comparison of MLC with Kalman filter on a noisy linear oscillator

In practice, full-state measurements of the system are often unavailable or may be prohibitively expensive to collect and process in real-time. Instead, it is typically necessary to collect limited sensor measurements and *reconstruct* the relevant state through dynamic estimation, for example using the Kalman filter from Sec. 3.3.

In a high-dimensional fluid, even reconstructing the state through estimation may be computationally expensive, introducing unacceptable time delays in the control loop. Instead reduced-order models are generally used to describe the few states that are most controllable and observable. A more challenging test of MLC involves full-state estimation from limited noisy sensor measurements. As an illustrative example, consider a neutrally stable oscillator with no forcing:

$$\frac{d}{dt}a_1 = \sigma a_1 - \omega a_2 + w_{d,1} \tag{4.6a}$$

$$\frac{d}{dt}a_2 = \omega a_1 + \sigma a_2 + w_{d,2} \tag{4.6b}$$

$$s = a_1 + w_n. \tag{4.6c}$$

Again, this corresponds to a linear system with the following system matrices

$$\mathbf{A} = \begin{bmatrix} \sigma & -\omega \\ \omega & \sigma \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \tag{4.7a}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 \end{bmatrix}, \qquad \mathbf{D} = \begin{bmatrix} 0 \end{bmatrix}. \tag{4.7b}$$

In this example, we will consider $\sigma = 0$ and $\omega = 1$, corresponding to a neutrally stable oscillator. Finally, the disturbance and noise covariance are given by:

$$\mathbf{V}_d = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{V}_n = 0.1. \tag{4.8}$$

The cost function quantifies the accuracy of the estimation, as in Eq. (3.16).

The Matlab® implementation is shown in Code 4.2, and the full-state Kalman filter estimate is shown in Fig. 4.2. The noisy sensor measurement s is shown in the middle panel for a single noise realization. An ensemble of square-summed errors are shown in the bottom panel, and the ensemble average is the cost function J, as in Eq. (3.16). Because there is constantly error introduced through noisy measurements, the cost function continues to increase for all time.

To compare the MLC solution, it is necessary to first generalize the function tree representation beyond a static input–output map. In particular, we envision two methods of generalizing a function tree to achieve dynamic estimation: First, it is possible to have nodes that accumulate information by integration (see Fig. 4.3), and second, it is possible to have function expressions for the numerator and denominator of a transfer function (see Fig. 4.4).

Code 4.2 LQE implementation in Matlab®.

```
% Define system matrices
sigma = 0, omega = 1;     % Neutrally stable oscillator
A = [sigma -omega;        % Dynamics
    omega sigma];
B = [eye(2) [0; 0] ];     % Disturbance plus actuation
C = [1 0];                % Measure first state
D = 0;                    % No feedthrough term
sys = ss(A,B,C,D);        % Continuous state-space system

% Disturbance and noise covariance matrices
Vd = eye(2);              % Disturbance covariance
Vn = .1;                  % Noise covariance
Vdn = [0; 0];             % No cross-terms

% Compute Kalman filter using LQE
[L,P,E] = lqe(A,eye(2),C,Vd,Vn,Vdn);  % L is gain matrix
Aest = A-L*C;               % Estimator dynamics
Best = L;                   % Input to estimator
Cest = eye(2);              % Estimator outputs both states
Dest = [0; 0];              % No feedthrough
sysK = ss(Aest,Best,Cest,Dest);  % Estimator system

% Loop through 50 noise realizations for average
for count = 1:50
    t = 0:0.01:20;            % Duration of simulation
    d1 = 1*randn(size(t));    % Disturbance to state a1
    d2 = 1*randn(size(t));    % Disturbance to state a2
    n = .1*randn(size(t));    % Noise
    b = zeros(size(d1));      % No actuation

    % Simulate noisy system with disturbance
    [s,tout,a] = lsim(sys,[d1; d2; b],t,[1;0]);

    % Simulate clean system for "truth" baseline
    [sclean,tout,aclean] = lsim(sys,[0*d1; 0*d2; b],t,[1;0]);

    % Simulate Kalman filter to estimate ahat
    sn = s+n';
    [ahat,tout] = lsim(sysK,sn,t,[1; 0]);

    % Compute cost function
    for k=1:size(a,1)
        err = a(k,:)-ahat(k,:);  % Choice: use 'a' or 'aclean'
        Jlong(k) = err*err';
    end
    Jindiv = cumtrapz(t,Jlong);  % Trapezoidal integration
    Jall(count,:) = Jindiv; % Store current realization
end
J = mean(Jall,1);           % Average cost across realizations
```

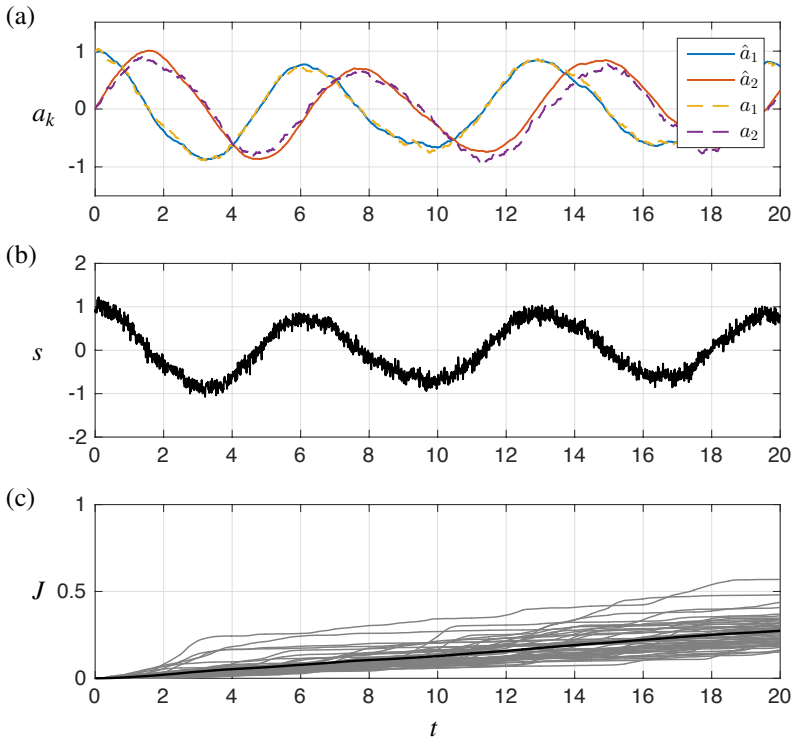

Fig. 4.2 Dynamic full-state estimation of the neutrally stable oscillator system in Eq. (4.6) with a Kalman filter (i.e., LQE). The cost J (3.16) rises because disturbances and noise constantly introduce discrepancies between the estimate $\hat{\mathbf{a}}$ and the true state $\mathbf{a}$. An ensemble average cost function is shown in black, and the individual cost functions for fifty instances are shown in gray.

The performance of MLC for state estimation is shown in Fig. 4.5 for the same conditions as used in Fig. 4.2. The state is estimated despite large disturbances and sensor noise. MLC results in a sum-square error that is about twice as large as the optimal Kalman filter solution.

The MLC implementation is given below with the parameters of Tab. 4.2.

Table 4.2 Main parameters used for MLC solution of LQE problem. $N_b = 4$ indicates that 4 subtrees are generated for each individual.

Parameter	N_i	N_s	N_b	P_r	P_m	P_c	N_p	N_e	Node functions
Value	1000	1	4	0.1	0.4	0.5	7	10	$+, -, \times$

The following code initializes MLC, runs 50 generations, and displays the results.

```
mlc=MLC('MLC_ex_LQE_problem'); %creates the MLC problem
mlc.go(50);                    %runs MLC for 50 generations
mlc.show_best_individual       %displays the results
```

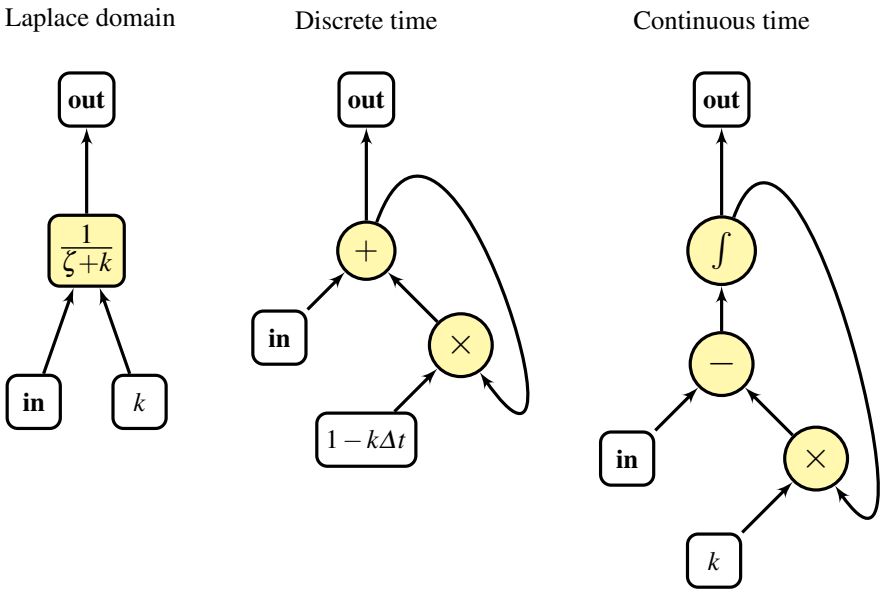

Fig. 4.3 Illustration of a generalized genetic programming transfer function operator in the frequency domain with Laplace variable ζ and filter frequency k. The discrete-time and continuous-time implementations are also shown.

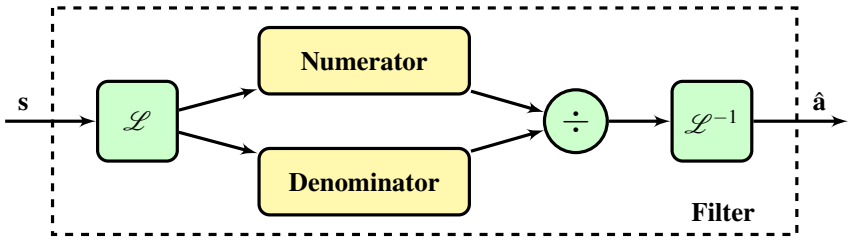

Fig. 4.4 General filter block used in genetic programming for MLC. The inputs **s** are Laplace transformed, and a transfer function is constructed in the frequency domain. This transfer function is achieved by allowing GP to select numerator and denominator polynomials, which are then divided. The output signal is inverse Laplace transformed to bring it back to the time domain.

Each filter is the quotient of two polynomials in frequency domain, requiring one tree for the numerator and one tree for the denominator, as in Fig. 4.4. In this example, two filters are identified for each of a_1 and a_2, resulting in 4 polynomials. All individuals are initialized to contain 4 subtrees, and can be written as the following LISP string:

$$(\text{root}\,(\text{poly}_1)\,(\text{poly}_2)\,(\text{poly}_3)\,(\text{poly}_4))$$

Where each 'poly_i' is a LISP polynomial such as '$(\times\, S0\,(+S0\,2.23)))$', where $S0$ represents the argument. In order to only obtain polynomials, $+$, $-$, and $\times$ are the only operations allowed. The two first subtrees define the denominator and numera-

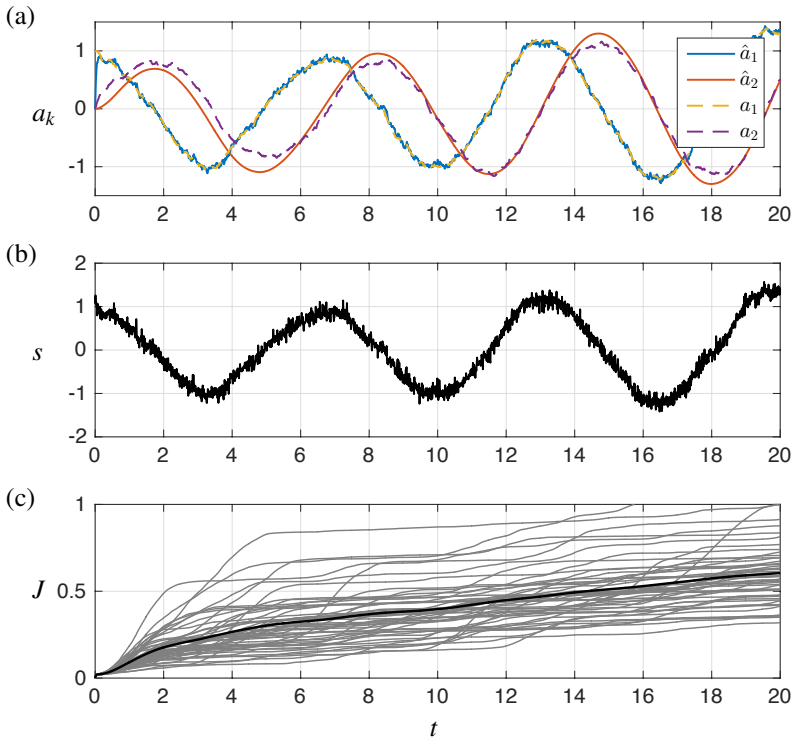

Fig. 4.5 Dynamic full-state estimation of the neutrally stable oscillator system in Eq. (4.6) using genetic programming for MLC. The cost J (3.16) continually rises because disturbances and noise constantly introduce discrepancies between the estimate $\hat{\mathbf{a}}$ and the true state $\mathbf{a}$. An ensemble average cost function is shown in black, and the individual cost functions for fifty random instances are shown in gray.

tor of the first filter. The two later subtrees describe the same quantities of the second filter.

The evaluation function can be displayed by using the command:

```
open('MLC_evaluator_LQE')
```

The evaluation function consists in setting up the Simulink® model with the parameters (initial conditions, noise level etc...) and the polynomials for the transfer functions. The frequency-domain filters need the denominator to be of higher order than the numerator. This is why a pre-evaluation function is called at individual initialization, so that proper transfer functions are enforced. The Simulink® implementation of MLC is shown in Figs. 4.6 and 4.7.

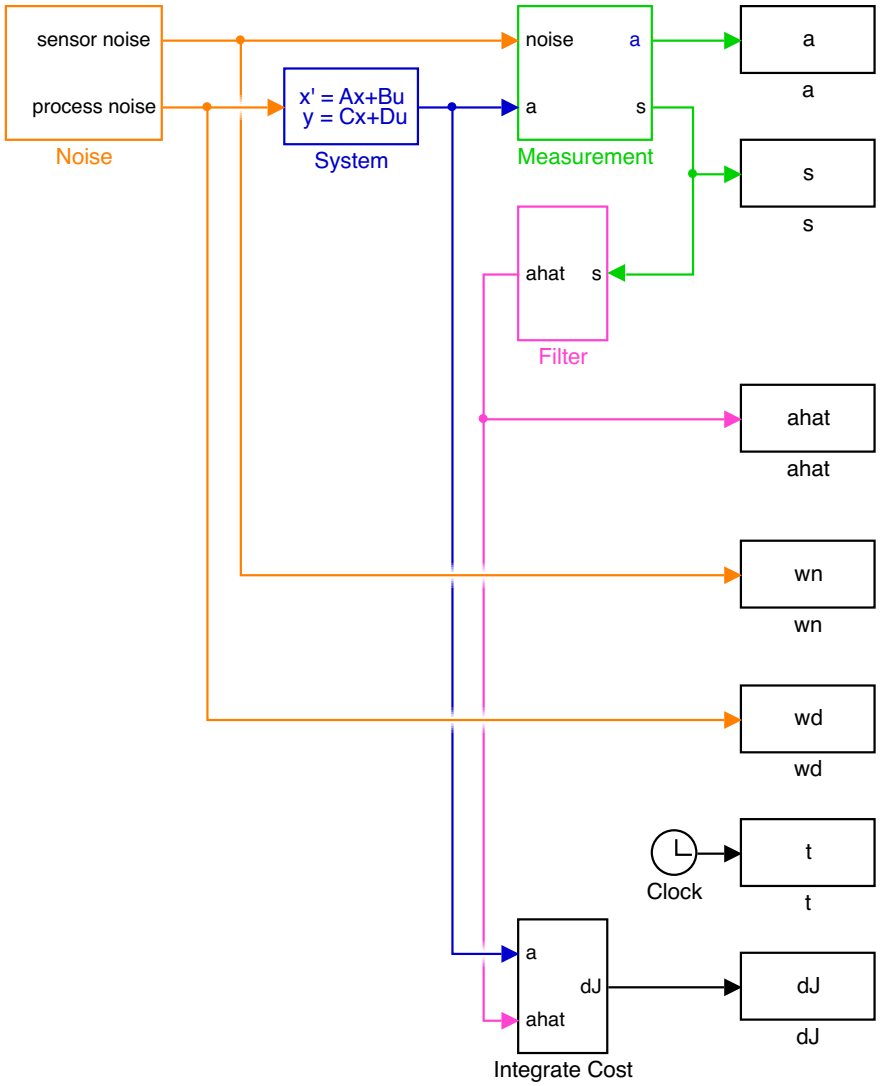

Fig. 4.6 Simulink® model to implement MLC for state estimation based on limited noisy measurements. Individual blocks are shown in Fig. 4.7.

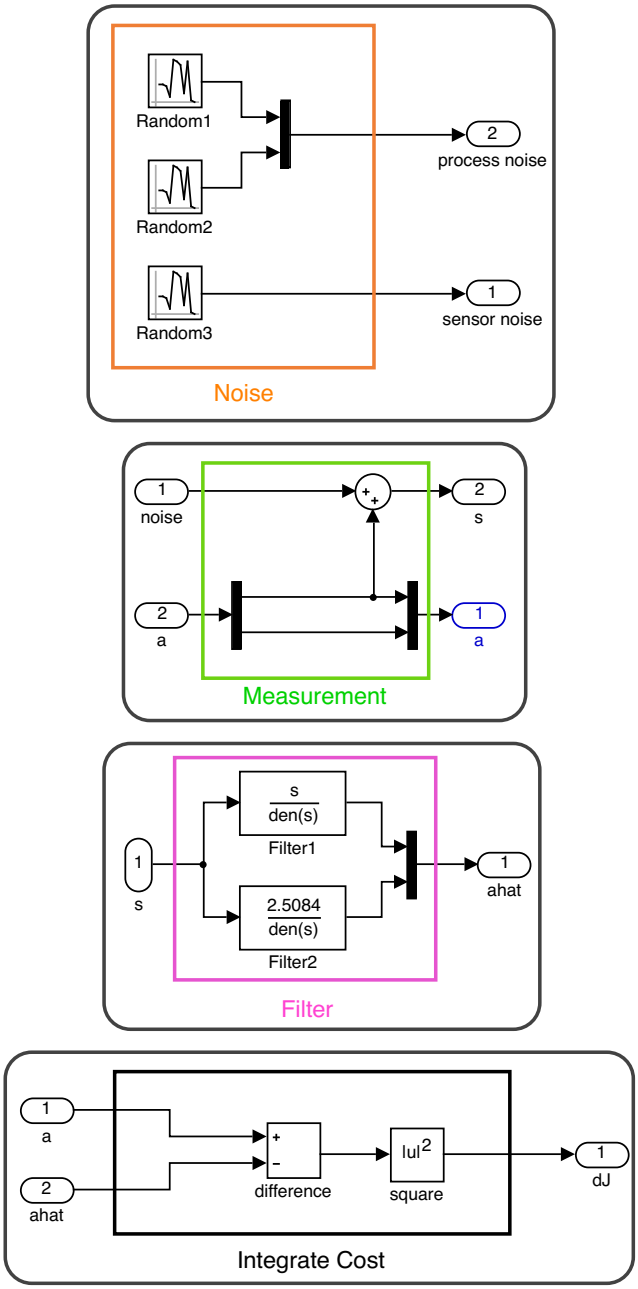

Fig. 4.7 Simulink® blocks for model in Fig. 4.6. This implements MLC for state estimation based on limited noisy measurements.

Time-delay coordinates through spatial measurements of convective flow

Interestingly, it has been shown in convective flow experiments that a finite but large set of spatial sensors have resulted in effective closed-loop control performance, even without dynamic estimation. It is likely that a rake of hot wires is effectively establishing time-delay coordinates, which may be acting as a proxy for a dynamic estimation state. This is a vibrant area of research, with many promising connections to dynamical systems via the Koopman operator [162, 163, 187, 47, 170, 188, 41], although this is beyond the scope of this book.

4.3 Comparison of MLC with LQG for sensor-based feedback

In model-based control design, it is possible to separately design an optimal regulator using LQR and an optimal Kalman filter, and then combine the two to obtain an optimal sensor-based LQG regulator. In practice, we may not have a model of the dynamics, and if the dynamics are unstable it is difficult to run experiments to collect data for system identification of a model. In this section, we demonstrate the ability of MLC to generate a sensor-based stabilizing feedback controller for an unstable system.

Again we consider the unstable system

$$\frac{d}{dt}a_1 = \sigma a_1 - \omega a_2 + w_{d,1} \tag{4.9a}$$

$$\frac{d}{dt}a_2 = \omega a_1 + \sigma a_2 + b + w_{d,2}, \tag{4.9b}$$

and we assume that the sensor only measures the first component:

$$s = a_1 + w_n. \tag{4.10}$$

This corresponds to a linear system with the following system matrices

$$\mathbf{A} = \begin{bmatrix} \sigma & -\omega \\ \omega & \sigma \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \tag{4.11a}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 \end{bmatrix}, \qquad \mathbf{D} = \begin{bmatrix} 0 \end{bmatrix}. \tag{4.11b}$$

In this example we choose $\sigma = \omega = 1$, corresponding to an unstable oscillator. The ensemble-averaged cost function from Eq. (3.20) is used.

The result of classic linear-quadratic-Gaussian (LQG) control is shown in Fig. 4.8. Note that the state **a** is rapidly stabilized despite a single noisy measurement s. How-

ever, because of continual noise and disturbances, the state has small fluctuations about zero for all time. These fluctuations contribute to a slowly increasing cost. The most significant increase in cost J is experienced from $t = 0$ to $t = 5$, where the state is large and there is significant control expenditure required to stabilize the system. The LQG controller represents the optimal sensor-based feedback regulator given a model of the system and of the noise and disturbance covariances.

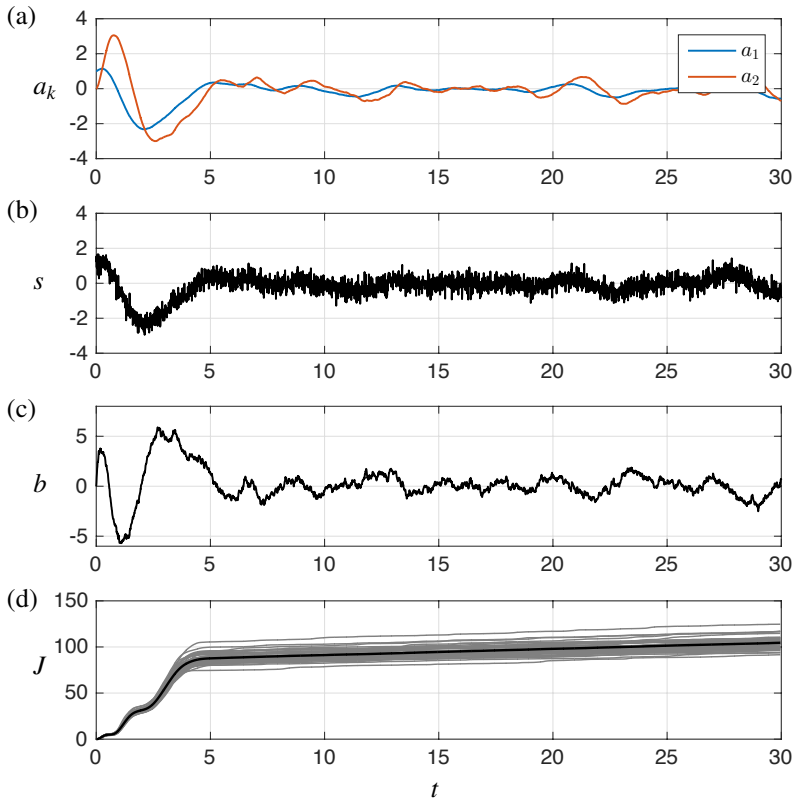

Fig. 4.8 Sensor-based feedback for the unstable oscillator system in Eq. (4.9) using LQG control. The cost J (3.20) continually rises because disturbances and noise constantly push the system away from equilibrium. An ensemble average cost function is shown in black, and the individual cost functions for fifty random instances are shown in gray.

Figure 4.9 shows the performance of the machine learning control approach on the same unstable system. In this example, there is no model of the system dynamics or of the noise and disturbance covariances. Instead, candidate sensor-based control schemes are formulated, tested, evaluated, and evolved to converge on a stabilizing controller. This represents a more challenging and general approach to control design. It is seen that this control also stabilizes the system, although the performance is not optimal as in the LQG case. However, this is reasonable, since MLC is not

based on a model, but instead relied on trial-and-error to stabilize the system. Moreover, since the system is unstable originally, it is quite challenging to find stabilizing control expressions. With more optimization and generations, it is likely that MLC will converge to a solution where less noise propagates through the control and into the state, resulting in a lower cost function.

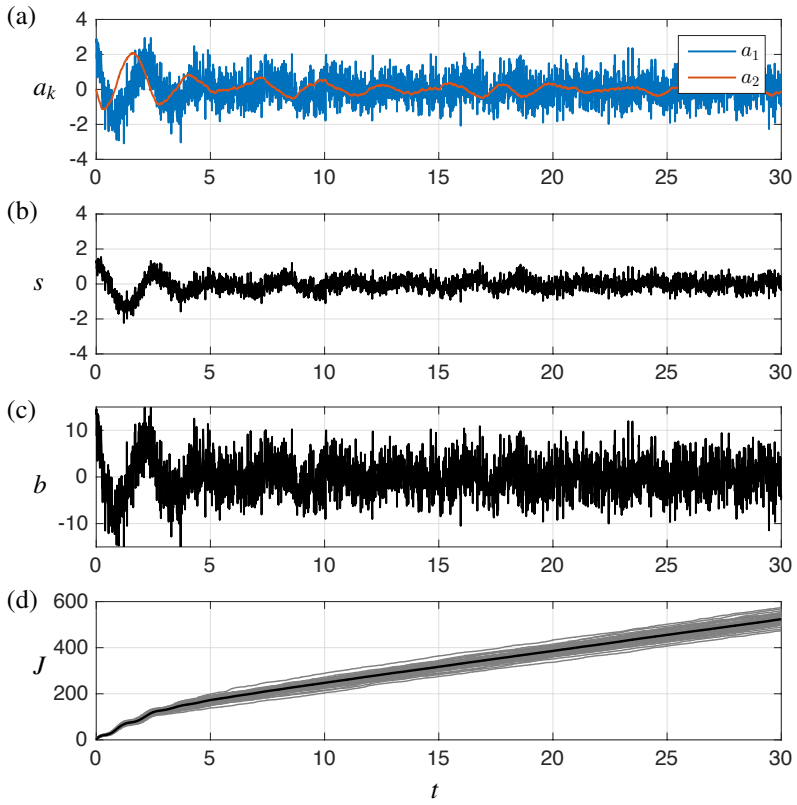

Fig. 4.9 Sensor-based feedback for the unstable oscillator system in Eq. (4.9) using MLC. Parameters and plotted quantities are the same as in Fig. 4.8.

Table 4.3 Main parameters used for MLC solution of LQG problem.

Parameter	N_i	N_s	N_b	P_r	P_m	P_c	N_p	N_e	Node functions
Value	1000	2	5	0.1	0.4	0.5	7	10	$+, -, \times$

The following code initializes the MLC problem, runs 50 generations, and displays the results.

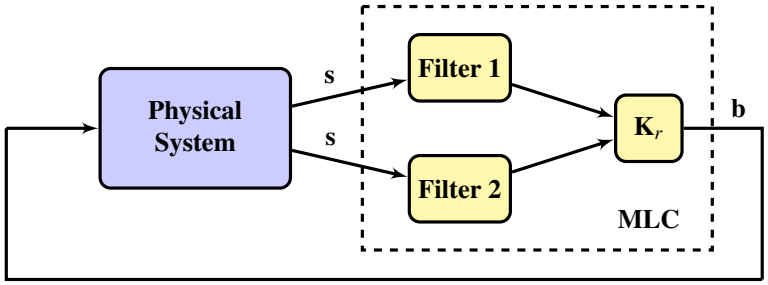

Fig. 4.10 Schematic illustrating the use of filter blocks in genetic programming for MLC on LQG problem. Each filter block is constructed as in Fig. 4.4.

```
mlc=MLC('MLC_ex_LQG_problem'); % Creates the MLC problem
mlc.go(25);                    % Runs MLC for 25 generations
mlc.show_best_individual       % Displays the results
```

In the LQG case, two filters are constructed to estimate the two states a_1 and a_2 from a single noisy sensor. Each filter is composed in the frequency domain as the ratio of a numerator and denominator polynomial, as in Fig. 4.4. Thus, each filter is represented by two expression trees. The controller gain matrix $\mathbf{K}_r$ is also represented as an expression tree, resulting in five trees overall. This can be written as the following LISP string:

$$(\text{root}\,(\text{poly}_1)\,(\text{poly}_2)\,(\text{poly}_3)\,(\text{poly}_4)\,(\text{poly}_5)),$$

where each 'poly_i' is a LISP polynomial such as '$(\times\, S0\,(+S1\,2.23))$)'. In order to only obtain polynomials, $+$, $-$, and $\times$ are the only operations allowed. For the four first subtrees, all sensors ($S0$ or $S1$) are interpreted as the same variable so that they can be used in polynomials as in the LQE implementation. The fifth subtree uses the two estimated states as its sensor input.

The evaluation function can be displayed by using the command:

```
open('MLC_evaluator_LQG')
```

The evaluation function consists in setting up the Simulink® model with the parameters (initial conditions, noise level etc...), the polynomials for the Laplace space filters and finally the control law in the controller box. As the Laplace space filters needs the denominator to be of higher order than the numerator, for both filters, 75% of the individuals randomly generated this way cannot describe correctly such a filter. This is why a pre-evaluation function is called at the individual creation, so that only the individuals that meet the requirement that both denominator polynomials are of higher order than their respective numerator are kept.

4.4 Modifications for small nonlinearity

Often, linear control can be applied to nonlinear systems. Here, we compare the performance of MLC and LQR when nonlinear terms are added to a linear system:

$$\frac{d}{dt}\mathbf{a} = \mathbf{A}\mathbf{a} + \varepsilon \mathbf{F}(\mathbf{a}) + \mathbf{B}\mathbf{b}. \tag{4.12}$$

In this case, the variable ε modulates the strength of the nonlinearity, with $\varepsilon = 0$ corresponding to the linear system in Eq. (3.1). As an example, consider the Hopf system, which adds a cubic nonlinearity to the linear oscillator in Eq. (4.2):

$$\frac{d}{dt}a_1 = \sigma a_1 - \omega a_2 - \varepsilon a_1(a_1^2 + a_2^2) \tag{4.13a}$$

$$\frac{d}{dt}a_2 = \omega a_1 + \sigma a_2 - \varepsilon a_2(a_1^2 + a_2^2) + b. \tag{4.13b}$$

For unstable linear dynamics with $\sigma > 0$, this system has a stable limit cycle with radius $r = \sqrt{a_1^2 + a_2^2} = \sqrt{\sigma/\varepsilon}$. The smaller ε is, the larger the limit cycle radius, and the larger the domain where the linear model is valid. As ε approaches zero, then the limit cycle radius increases to infinity, and the linear system is recovered.

Again, we assume full-state feedback, so that $\mathbf{s} = \mathbf{a}$, and we use the same linear dynamics $\sigma = \omega = 1$, the same LQR controller $b = -\mathbf{K}_r\mathbf{a}$, and the same LQR cost function weights $\mathbf{Q}$ and R from Sec. 4.1.

Figure 4.12 shows the LQR controller response for increasing nonlinearity ε. The system is stabilized for all values, and as ε increases, the nonlinear dynamics actually help the controller, so that the overall cost J decreases. Figure 4.13 shows the LQR performance when ε is negative. In this case, the increasing nonlinearity makes the system more difficult to control, and after a point the LQR controller fails to stabilize the system; these parameter values are marked as diverging solutions.

Figures 4.14 and 4.15 show the machine learning control performance on the same system. In the case of positive ε, the performance is quite similar to the LQR solution, whereas in the case of negative ε, MLC performs with lower cost and over a larger range of ε corresponding to stronger nonlinearities.

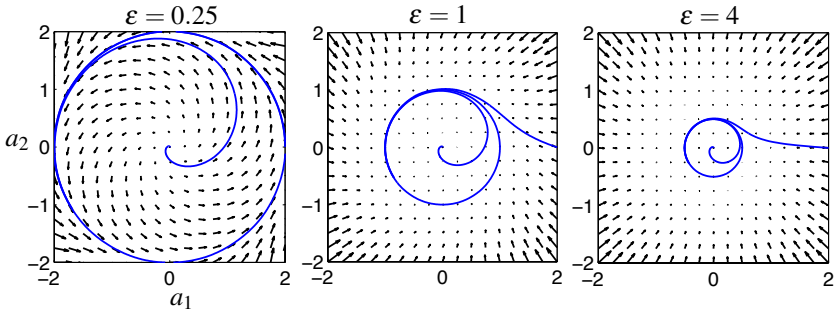

Fig. 4.11 Vector field and trajectories attracting onto limit cycle for the Hopf normal form in Eq. (4.13) with various values of ε.

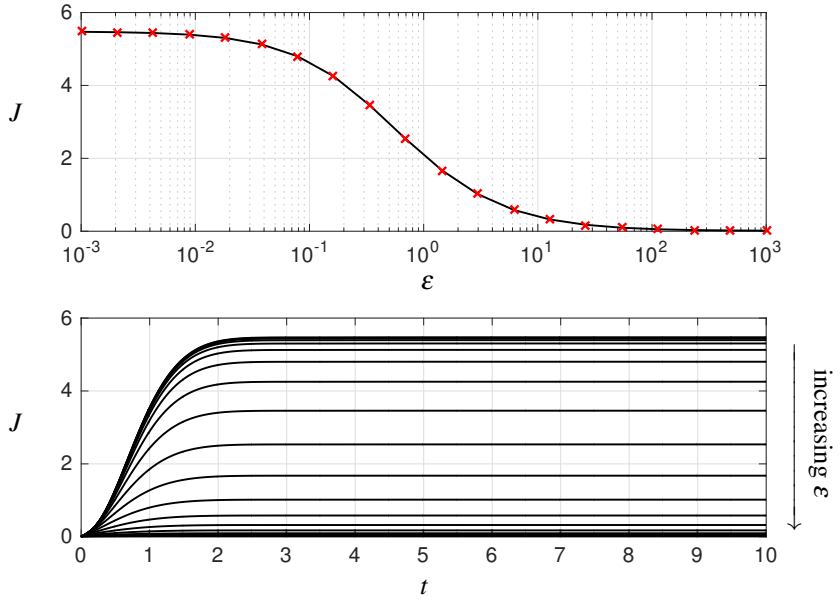

Fig. 4.12 Cost of LQR for Hopf normal form with varying nonlinearity strength ε. The initial condition for each case is $a_1 = a_2 = \sqrt{2}/2$.

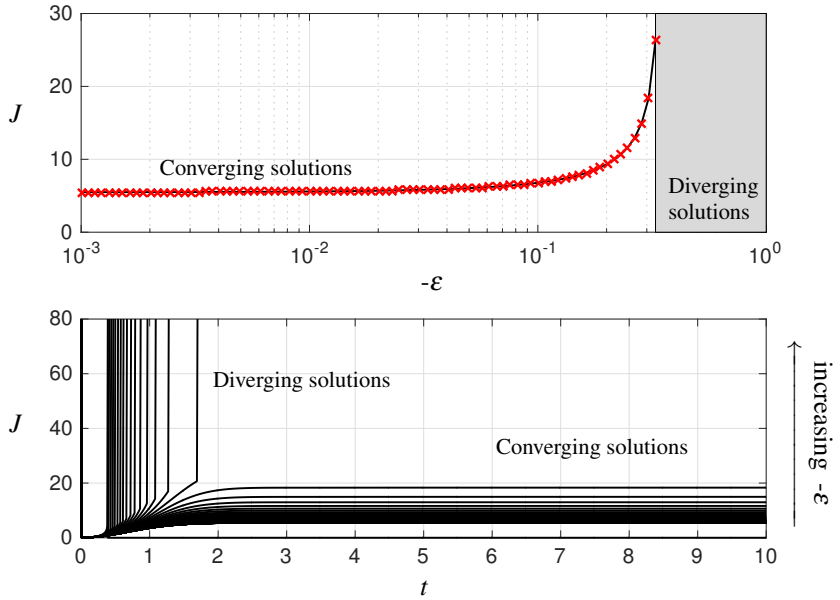

Fig. 4.13 Cost of LQR for unstable Hopf normal form with various magnitude of nonlinearity, $-\varepsilon$. The initial conditions are given by $a_1 = a_2 = \sqrt{2}/2$.

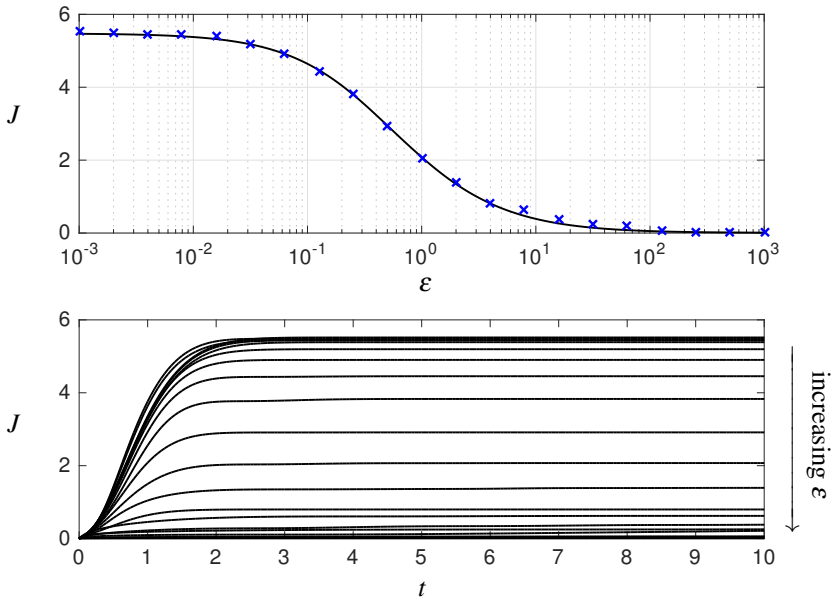

Fig. 4.14 Cost of MLC using $(+,-,*)$ for Hopf normal form with varying nonlinearity strength ε. The initial condition for each case is $a_1 = a_2 = \sqrt{2}/2$. The blue crosses indicate MLC results and the solid black line shows the corresponding LQR cost.

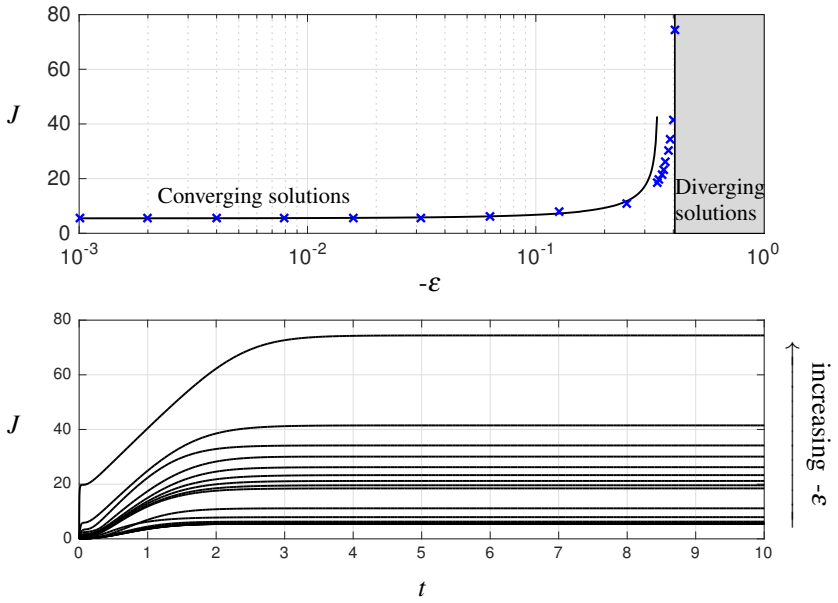

Fig. 4.15 Cost of MLC using $(+,-,*)$ for unstable Hopf normal form with various magnitude of nonlinearity, $-\varepsilon$. The initial conditions are given by $a_1 = a_2 = \sqrt{2}/2$. The blue crosses indicate MLC results while the solid black line shows the corresponding LQR cost.

4.5 Exercises

Exercise 4-1: Consider the same linear system from **Exercise 3-1**:

$$\frac{d}{dt}\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} b. \tag{4.14}$$

(a) Now, use genetic programming to solve for the optimal controller $b = \mathbf{K}(\mathbf{s})$ assuming full state measurements $\mathbf{s} = \mathbf{a}$. This controller should minimize the LQR cost function:

$$J = \int_0^\infty \left[\mathbf{a}^T(\tau)\,\mathbf{Q}\,\mathbf{a}(\tau) + R\,b^2(\tau)\right] d\tau, \quad \mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R = 1. \tag{4.15}$$

(b) Check your MLC expression and compare with the optimal LQR solution. Implement a refinement to select genetic programming expressions with good performance but also with the added constraint of a simple expression. This may be added as a penalty on the cost function, or you may alternatively plot a Pareto front of complexity vs performance for numerous candidate high-performance controllers from the generations.

Exercise 4–2: Consider the neutrally stable system:

$$\frac{d}{dt}\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} + \mathbf{B}b + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} w_{d_1} \\ w_{d_2} \end{bmatrix}. \tag{4.16a}$$

$$s = \mathbf{C}\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} + w_n. \tag{4.16b}$$

(a) For the case with no actuation, $\mathbf{B} = \mathbf{0}$, and a measurement of the first state $\mathbf{C} = \begin{bmatrix} 1 & 0 \end{bmatrix}$, develop a genetic programming expression to estimate the state from noisy measurements using full-state training data. Construct an expression to estimate the state with and without the use of filter blocks as discussed in this chapter. How does the static function perform in the presence of noise?

(b) Now consider the case with actuation $\mathbf{B} = \begin{bmatrix} 0 & 1 \end{bmatrix}^T$, and develop a GP expression to estimate the state with forcing. Use the same cost function as in the formulation of a Kalman filter.

(c) Finally, develop an optimal sensor based feedback control law using genetic programming for MLC that optimizes the LQR cost function.

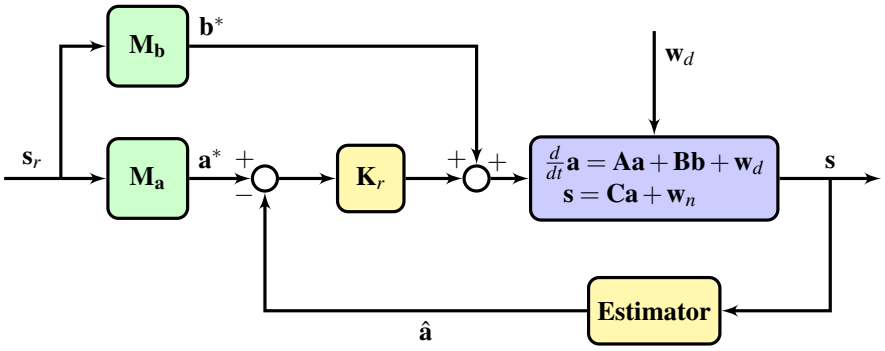

Fig. 4.16 LQG reference tracking schematic for Exercise 4–3.

Exercise 4–3: In many cases, it is desirable to track a reference trajectory with feedback control, as opposed to stabilizing a fixed point. For instance, we may implement a control law to design and track a limit cycle.

With a working LQG controller, it is possible to command reference trajectories, as depicted schematically in Fig. 4.16. In the general case, it is necessary to translate the reference sensor signal $\mathbf{s}_r$ into a reference actuation $\mathbf{b}^*$ and reference state $\mathbf{a}^*$ according to the following formula:

$$\begin{bmatrix} \mathbf{0} \\ \mathbf{s}_r \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{a}^* \\ \mathbf{b}^* \end{bmatrix} \quad \Longrightarrow \quad \begin{bmatrix} \mathbf{a}^* \\ \mathbf{b}^* \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{\dagger} \begin{bmatrix} \mathbf{0} \\ \mathbf{s}_r \end{bmatrix}, \tag{4.17}$$

where the superscript '†' denotes the Moore-Penrose pseudo inverse. In the case of full-state measurements, so that $\mathbf{s} = \mathbf{a}$, then $\mathbf{a}_r = \mathbf{a}^*$, so that $\mathbf{M_a} = \mathbf{I}$ and

$$\mathbf{b}^* = \mathbf{B}^{\dagger}\mathbf{A}\mathbf{a}_r. \tag{4.18}$$

Thus, $\mathbf{M_b} = \mathbf{B}^{\dagger}\mathbf{A}$.

Use MLC to design a controller to force a stable linear system into a limit cycle behavior shown in Fig. 4.17. Implement the controller with actuation $\mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Compare with the full-state LQR controller response in Fig. 4.17.

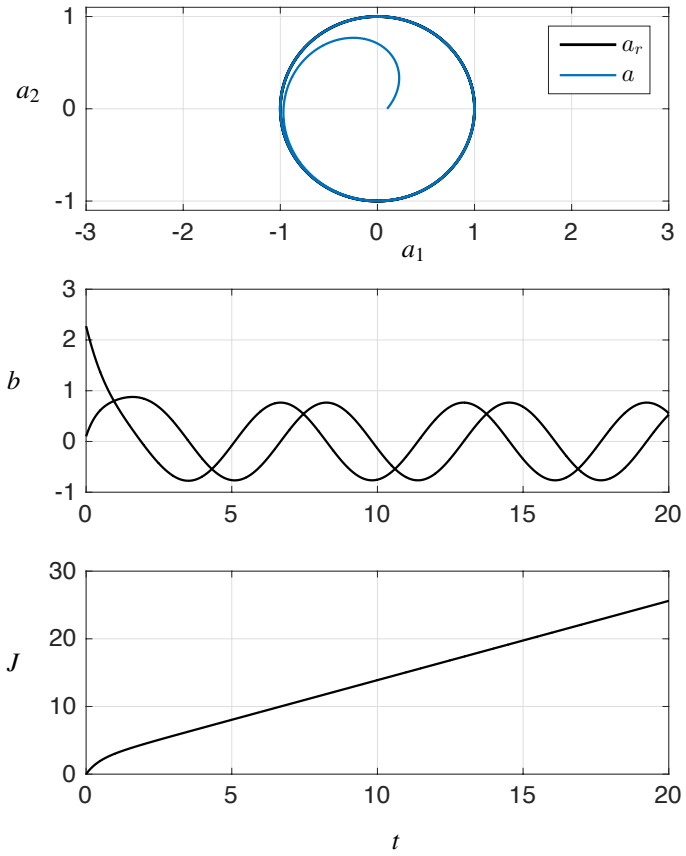

Fig. 4.17 LQR stabilization of a limit cycle with full control authority in Exercise 4–3.

4.6 Interview with Professor Shervin Bagheri

Shervin Bagheri is Associate Professor in fluid mechanics at the Royal Institute of Technology (KTH) and the Linné Flow Centre. He has pioneered model-based closed-loop control with numerical and experimental demonstrations. He is co-inventor of DMD modes, also known as Koopman modes, one of most used ingredients of reduced-order models. In his research, he focusses on the mathematical foundations and physical principles that enable manipulation of fluid flows. These efforts include both passive and active means to decrease drag, to increase mixing and to enhance lift on bodies. His work was published in the leading fori of our field including Nature Communications, the Philosophical Transactions of the Royal Society London A and Physical Review Letters.

Authors: You are a leader in the field of model-based flow control with landmark contributions to linear control design. What are some of the major trends that you have observed in flow control over the past decade?

Prof. Bagheri: It was only 15 years ago that a systems theoretical approach (such as input-output analysis, state-space systems, controllability, observability etc) to analyze and control shear flows became an active field on its own. Since then, we have had a decade of proof-of-concept work, focusing on accommodating and testing many of the powerful systems theoretical tools on fluid flows, and using model reduction as enabler to do this. Last years however, things have changed. We know that many control theorerical tools, albeit expensive, can be applied to control fluid flow instabilities both in convective and globally unstable flows at low Reynolds numbers. The use of these tools to nonlinear and turbulent flows is the next step, and in the last years several groups have made progress. For example, by treating the nonlinear terms of the Navier-Stokes equations as an appropriate stochastic forcing. Another emerging branch in flow control is the use of transfer operators (such as the Koopman operator). These methods can via an appropriate nonlinear transformation of the system provide a linear system, where analysis and control tools can be applied, followed by a transformation back. A third example is the use of online adaptive self-learning techniques, such as machine learning. In summary, the major current trends are to deal with nonlinearity and high-dimensionality at the same time in order to move from simple linear 2D systems at low Reynolds numbers towards more complex systems.

Authors: In recent years, you are moving to nonlinear modeling, statistical closures and machine learning methods. Can you sketch the need for the inclusion

of nonlinear dynamics and noise in model-based flow control? How much can be gained?

Prof. Bagheri: Moving in this direction is necessary, since in nearly all classical engineering applications the Reynolds number is high and the flow physics is sensitive to the external disturbance environment. It is clear that a flow-control technology based on a systematic approach (in contrast to a trial-and-error approach) that can be used in applications, has to deal with turbulence and robustness. However, we should also be realistic (and humble) for this task, since for the coming decades our computational capability is limited to academic complex problems, such as low-Reynolds number turbulent flows.

Authors: It is common for practitioners to collect data to characterize a system, develop a model, and then use this model for control. What are some of the challenges and benefits associated with the online learning and adaptation of controllers from data?

Prof. Bagheri: Indeed, data-driven methods are becoming increasingly important, as large-scale high-performance computations have now taken its rightful place in the community. When it comes to control of high-dimensional nonlinear chaotic systems, in my opinion, an attractive approach is adaptive algorithms that are able to adjust online to new dynamics in uncertain conditions. One of the challenges we have encountered when using adaptive algorithms is that although they may be fast enough to account for slow changes in conditions (e.g. variation in the Reynolds number), they are often not sufficient quick learners to account for changes in the internal changes in the dynamics (e.g. emergence of new length scales during the transition process).

Authors: In the coming decades, what do you envision as the academically most rewarding grand-challenge problems of feedback control in fluid dynamics? You work not only on a model-based control logic but also on model-free bio-inspired actuators. Which evolutions do you foresee in experimental flow control on a hardware and a theoretical level?

Prof. Bagheri: The grand challenge is to efficiently and robustly control turbulence for Reynolds numbers that are relevant for everyday applications. Within the next decade, we will be able to reduce turbulent skin friction drag with 30% using actuation/sensing at the wall at moderate Reynolds numbers. It will probably take another decade or two, to devise both efficient and robust controllers for high-Reynolds number turbulent flows, where the contribution to skin-friction is also significant from large scale structures. In order to achieve these goals we need a multi-disciplinary approach, where advances in fluid mechanics, material science and surface chemistry are combined with applied mathematics, algorithms and computer science.

For example, we are now looking into how soft, porous, lubricated, multi-scale hierarchal materials possibly treated chemically can be used to manipulate an

overlying fluid. Although, mimicking biological surface coatings such shark skin and lotus leaf has proven useful, I believe that active control techniques can provide the right guidance for using innovative surface materials for flow control.

Authors: We look forward to your next breakthroughs in flow control and thank you for this interview!

Prof. Bagheri: Thank you. It was a pleasure.