Jinkun Liu

Intelligent Control Design and MatLab Simulation

Jinkun Liu
Beihang University
Beijing
China

ISBN 978-981-10-5262-0 ISBN 978-981-10-5263-7 (eBook)
https://doi.org/10.1007/978-981-10-5263-7

Jointly published with Tsinghua University Press, Beijing

Library of Congress Control Number: 2017951417

This Springer imprint is published by Springer Nature
The registered company is Springer Nature Singapore Pte Ltd.
The registered company address is:
152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Preface

Recent years have seen a rapid development of intelligent control techniques and their successful applications. Numerous theoretical studies and actual industrial implementations demonstrate that artificial intelligent control is a good candidate for control system design in solving the control problems of complex nonlinear systems in the presence of different kinds of uncertainties. Many control approaches/methods, reporting inventions and control applications within the fields of adaptive control, neural control, and fuzzy systems, have been published in various books, journals, and conference proceedings. In spite of these remarkable advances in neural control field, due to the complexity of nonlinear systems, the present research on intelligent control is still focused on the development of fundamental methodologies.

The advantage of intelligent control is that neural network and fuzzy system can model any (sufficiently smooth) continuous nonlinear function in a compact set and the modeling error is becoming smaller. Thus, an adaptive intelligent controller is most suitable in an environment where system dynamics are significantly changing, highly nonlinear, and in principle not completely known.

This book is motivated by the need for systematic design approaches for intelligent control system design using neural network and fuzzy-based techniques. The main objectives of the book are to introduce the concrete design method and MATLAB simulation of intelligent control strategies.

It is our goal to accomplish these objectives:

- Offer a catalog of implementable intelligent control design methods for engineering applications;
- Provide advanced intelligent controller design methods and their stability analysis methods;
- For each intelligent control algorithm, we offer its simulation example and MATLAB program.

This book provides the reader with a thorough grounding in the intelligent control system design. Typical intelligent controller design is verified using MATLAB simulation. In this book, concrete case studies, which present the results of intelligent controller implementations, are used to illustrate the successful application of the theory.

The book is structured as follows. The book starts with a brief introduction of intelligent control in Chap. 1, expert control algorithm and design remarks are given in Chap. 2, fuzzy sets and membership function are introduced in Chap. 3, fuzzy logic controller design is introduced in Chap. 4, fuzzy T-S modeling and control is introduced in Chap. 5, adaptive fuzzy controller design and analysis are given in Chap. 6, Neural network theory are introduced in Chap. 7, in this Chapter, several typical neural networks such as BP neural network and RBF neural network are introduced, the basic design method of adaptive RBF neural network control and adaptive sliding mode RBF neural network control are introduced in Chaps. 8 and 9, respectively. Discrete RBF neural network controller design and analysis are given in Chap. 10. Intelligent optimization algorithms are recommended in Chap. 11, and at last, iterative learning control algorithm and applications are given in Chap. 12. For each chapter, several engineering application examples are given. The contents of each chapter in this book are independent, so that readers can their own needs.

In this book, all the control algorithms and their programs are described separately and classified by the chapter name, which can be run successfully in MATLAB 7.5.0.342 version or in other more advanced versions. In addition, all the programs can be downloaded via http://shi.buaa.edu.cn/liujinkun. If you have questions about algorithms and simulation programs, please E-mail:ljk@buaa.edu.cn.

Beijing, China Jinkun Liu

Contents

Abstract

The advantage of intelligent control is that neural network and fuzzy system can model any (sufficiently smooth) continuous nonlinear function in a compact set and the modeling error is becoming smaller. Thus, an adaptive intelligent controller is most suitable in an environment where system dynamics are significantly changing, highly nonlinear, and in principle not completely known.

The book is structured as follows. The book starts with a brief introduction of intelligent control in Chap. 1, expert control algorithm and design remarks are given in Chap. 2, fuzzy sets and membership function are introduced in Chap. 3, fuzzy logic controller design is introduced in Chap. 4, fuzzy T-S modeling and control is introduced in Chap. 5, adaptive fuzzy controller design and analysis are given in Chap. 6, neural network theory is introduced in Chap. 7, and in this chapter, several typical neural networks such as BP neural network and RBF neural network are introduced; the basic design method of adaptive RBF neural network control and adaptive sliding mode RBF neural network control are introduced in Chaps. 8 and 9, respectively. Discrete RBF neural network controller design and analysis are given in Chap. 10. Intelligent optimization algorithms are recommended in Chap. 11, and at last, iterative learning control algorithm and applications are given in Chap. 12. For each chapter, several engineering application examples are given. The contents of each chapter in this book are independent, so that readers can do research by their own needs.

This book provides the reader with a thorough grounding in the intelligent controller design. Typical intelligent controller design is emphasized using MATLAB simulation.

Each chapter of the book is interrelated and mutually independent, and the readers can choose to learn according to their own needs. This book is suitable for the readers who engage in the field of production process automation, computer application, electronic machinery, and electrical automation, especially can be used for professional teaching book.

Chapter 1
Introduction to Intelligent Control

The term "intelligent control" may be loosely used to denote a control technique that can be carried out using the "intelligence" of a human who is knowledgeable in the particular domain of control. In this definition, constraints pertaining to limitations of sensory and actuation capabilities and information processing speeds of humans are not considered. It follows that if a human in the control loop can properly control a plant, then that system would be a good candidate for intelligent control. Information abstraction and knowledge-based decision making that incorporates abstracted information are considered important in intelligent control. Unlike conventional control, intelligent control techniques possess capabilities of effectively dealing with incomplete information concerning the plant and its environment, and unexpected or unfamiliar conditions. The term "adaptive control" is used to denote a class of control techniques where the parameters of the controller are changed (adapted) during control, utilizing observations on the plant (i.e., with sensory feedback), to compensate for parameter changes, other disturbances, and unknown factors of the plant. Combining these two terms, one may view "intelligent adaptive control" as those techniques that rely on intelligent control for proper operation of a plant, particularly in the presence of parameter changes and unknown disturbances.

There are several artificial intelligent techniques that can be used as a basis for the development of intelligent systems, namely expert control, fuzzy logic, neural network, and intelligent search algorithms.

In this class, we will study some fundamental techniques and some application examples of expert control, fuzzy logic, neural networks, and intelligent search algorithms. The main focus here will be their use in intelligent control.

The artificial intelligent techniques should be integrated with modern control theory to develop intelligent control systems.

In this class, we study intelligent control in four parts: expert control, fuzzy logic and control, neural network and control, and genetic algorithm.

1.1 Expert Control

Expert control is control tactics to use expert knowledge and experience. Expert control comes from expert system, it was proposed by K.J. Astrom in 1986 [1], and its main idea is to design control tactics with expert knowledge and experience.

1.2 Fuzzy Logic Control

Fuzzy logic is useful in representing human knowledge in a specific domain of application, and in reasoning with that knowledge to make useful inferences or actions.

In particular, fuzzy logic may be employed to represent, as a set of "fuzzy rules," the knowledge of a human controlling a plant. This is the process of knowledge representation. Then, a rule of inference in fuzzy logic may be used according to this "fuzzy" knowledge base, to make control decisions for a given set of plant observations. This task concerns "knowledge processing." In this sense, fuzzy logic in intelligent control serves to represent and process the control knowledge of a human in a given plant.

There are two important ideas in fuzzy systems theory:

- The real world is too complicated for precise descriptions to be obtained; therefore, approximation (or fuzziness) must be introduced in order to obtain a reasonable model.
- As we move into the information era, human knowledge becomes increasingly important. We need a theory to formulate human knowledge in a systematic manner and put it into engineering systems, together with other information like mathematical models and sensory measurements.

From the fuzzy universal approximation theorem [2], fuzzy system can approximate any nonlinear function, which can be used to design adaptive fuzzy controller. By adjusting a set of weighting parameters of a fuzzy system, it may be used to approximate an arbitrary nonlinear function to a required degree of accuracy.

1.3 Neural Network and Control

Artificial neural networks are massively connected networks that can be trained to represent complex nonlinear functions at a high level of accuracy. They are analogous to the neuron structure in a human brain.

It is well known that biological systems can perform complex tasks without recourse to explicit quantitative operations. In particular, biological organisms are

capable of learning gradually over time. This learning capability reflects the ability of biological neurons to learn through exposure to external stimuli and to generalize. Such properties of nervous systems make them attractive as computation models that can be designed to process complex data. For example, the learning capability of biological organisms from examples suggests possibilities for machine learning.

Neural networks, or more specifically, artificial neural networks, are mathematical models inspired from our understanding of biological nervous systems.

They are attractive as computation devices that can accept a large number of inputs and learn solely from training samples. As mathematical models for biological nervous systems, artificial neural networks are useful in establishing relationships between inputs and outputs of any kind of system. Roughly speaking, a neural network is a collection of artificial neurons. An artificial neuron is a mathematical model of a biological neuron in its simplest form. From our understanding, biological neurons are viewed as elementary units for information processing in any nervous system. Without claiming its neurobiological validity, the mathematical model of an artificial neuron is based on the following theses:

(1) Neurons are the elementary units in a nervous system at which information processing occurs.
(2) Incoming information is in the form of signals that are passed between neurons through connection links.
(3) Each connection link has a proper weight that multiplies the signal transmitted.
(4) Each neuron has an internal action, depending on a bias or firing threshold, resulting in an activation function being applied to the weighted sum of the input signals to produce an output signal.

Since the idea of the computational abilities of networks composed of simple models of neurons was introduced in the 1940s, neural network techniques have undergone great developments and have been successfully applied in many fields such as learning, pattern recognition, signal processing, modeling, and system control. Their major advantages of highly parallel structure, learning ability, nonlinear function approximation, fault tolerance, and efficient analog VLSI implementation for real-time applications greatly motivate the usage of neural networks in nonlinear system identification and control.

In many real-world applications, there are many nonlinearities, unmodeled dynamics, unmeasurable noise, and multiloop, which pose problems for engineers to implement control strategies.

BP or RBF neural network can approximate any nonlinear function [3], which can be used to design adaptive neural network controller. By adjusting a set of weighting parameters of a neural network, it may be used to approximate an arbitrary nonlinear function to a required degree of accuracy.

1.4 Intelligent Search Algorithm

There are several intelligent search algorithms, classical intelligent search algorithms include GA, PSO, and DE.

Genetic algorithms (GA) are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover, and selection. The basic principle of GA was first laid down by Holland in 1962 [4]. GA simulates those processes in natural populations that are essential to evolution. Genetic algorithms belong to the area of evolutionary computing. They represent an optimization approach where a search is made to "evolve" a solution algorithm that will retain the "most fit" components, in a procedure that is analogous to biological evolution through natural selection, crossover, and mutation. It follows that GAs are applicable in intelligent control, particularly when optimization is an objective.

Particle swarm optimization (PSO) is originally attributed to Kennedy, Eberhart [5] and was first intended for simulating social behavior. Particle swarm optimization (PSO) is an evolutionary computation technique. The basic idea of particle swarm optimization (PSO) is to find the optimal solution through collaboration and information sharing among individuals in a swarm. The advantages of PSO are simplicity, ease of implementation, and no adjustment of many parameters. At present, it has been widely used in function optimization, neural network training, fuzzy system control, etc.

Differential evolution (DE) is originally due to Storn and Price [6]. In evolutionary computation, DE is a method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. DE is used for multidimensional real-valued functions but does not use the gradient of the problem being optimized, which means DE does not require for the optimization problem to be differentiable as is required by classic optimization methods such as gradient descent and quasi-newton methods. DE can therefore also be used on optimization problems that are not even continuous, are noisy, change over time, etc.

DE optimizes a problem by maintaining a population of candidate solutions and creating new candidate solutions by combining existing ones according to its simple formulae, and then keeping whichever candidate solution has the best score or fitness on the optimization problem at hand. In this way, the optimization problem is treated as a black box that merely provides a measure of quality given a candidate solution and the gradient is therefore not needed. DE has been applied in parallel computing, multiobjective optimization, constrained optimization, etc.

Summarizing, the biological analogies of fuzzy, neural, and intelligent search algorithms can be described as follows: Fuzzy techniques attempt to approximate human knowledge and the associated reasoning process; neural networks are a simplified representation of the neuron structure of a human brain; and intelligent search algorithms follow procedures that are crudely similar to the process of evolution in biological species.

Modern industrial plants and technological products are often required to perform complex tasks with high accuracy, under ill-defined conditions. Conventional control techniques may not be quite effective in these systems, whereas intelligent control has a tremendous potential. The emphasis of the class is on practical applications of intelligent control, primarily using fuzzy logic, neural network, and intelligent search algorithms techniques. The remainder of the class will give an introduction to some fundamental techniques of fuzzy logic, neural networks, and intelligent search algorithms.

References

1. K.J. Astrom, J.J. Anton, K.E. Arzen, Expert control. Automatica **22**(3), 277–286 (1986)
2. L.X. Wang, Fuzzy systems are universal approximators, in *Proceedings of IEEE Conference on Fuzzy Systems* (1992), pp. 1163–1170
3. K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximator. Neural Networks **2**(5), 359–366 (1989)
4. F. Jin, W. Chen, The father of the genetic algorithms—Holland and his scientific work. J. Dialect. Nat. (2007)
5. J. Kennedy, R. Eberhart, Particle swarm optimization, in *Proceedings of IEEE International Conference on Neural Networks* (1995), pp. 1942–1948
6. R. Storn, K. Price, Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. **11**, 341–359 (1997)

Chapter 2
Expert PID Control

Expert control is a control tactics to use expert knowledge and experience. Expert control was proposed firstly by Astrom in 1986 [1].

2.1 Expert PID Control

The expert PID control is to design PID parameters with characteristics of the plant and experience of the control expert, no need of modeling information.

The experience of the expert is mainly based on the error response of the system. Typical error response for a second transfer function is shown in Fig. 2.1; for the area I, III, V, VII, …, where the absolute value of error tends to smaller, we can use unloop control;for the area II, IV, VI, VIII, …, where the absolute value of error tends to bigger, we can use strong control or general control.

At time k, we consider the ideal position signal as $y_d(k)$, the output as $y(k)$, and then the tracking error is $e(k) = y_d(k) - y(k)$ at time k, and $e(k-1)$ and $e(k-2)$ represent the error at time $k-1$ and $k-2$, respectively, then we have

$$\begin{aligned} \Delta e(k) &= e(k) - e(k-1) \\ \Delta e(k-1) &= e(k-1) - e(k-2) \end{aligned} \tag{2.1}$$

According to Fig. 2.1, we can do the following analysis:

(1) When $|e(k)| > M_1$, we can use unloop controller to minimize error quickly.
(2) When $e(k)\Delta e(k) > 0$ or $\Delta e(k) = 0$,we consider two conditions as follows:
If $|e(k)| \geq M_2$, we use strong PID controller as

$$u(k) = u(k-1) + k_1\{k_p[e(k) - e(k-1)] + k_i e(k) + k_d[e(k) - 2e(k-1) + e(k-2)]\} \tag{2.2}$$

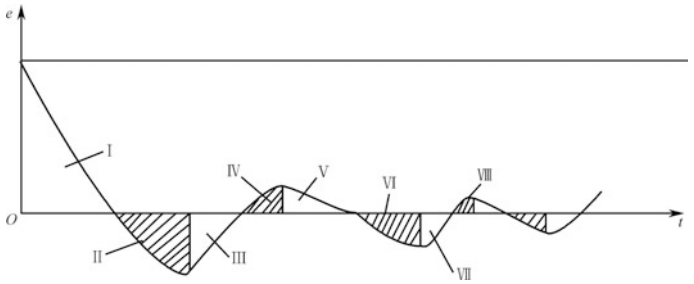


Fig. 2.1 Typical error response for a second transfer function

If $|e(k)| < M_2$, we use weak PID controller as

$$u(k) = u(k-1) + k_p[e(k) - e(k-1)] + k_i e(k) + k_d[e(k) - 2e(k-1) + e(k-2)] \tag{2.3}$$

(3) When $e(k)\Delta e(k) < 0$, $\Delta e(k)\Delta e(k-1) > 0$, or $e(k) = 0$,which indicates the absolute value of error tends to smaller or constant value, we can hold the control input.

(4) When $e(k)\Delta e(k) < 0$, $\Delta e(k)\Delta e(k-1) < 0$, which indicates the value of error is in extremism state. We consider the two conditions as follows:
If the absolute value of error is big, $|e(k)| \geq M_2$, we can use strong controller as

$$u(k) = u(k-1) + k_1 k_p e(k) \tag{2.4}$$

If the absolute value of error is small, $|e(k)| < M_2$, we can adopt weak controller as

$$u(k) = u(k-1) + k_2 k_p e(k) \tag{2.5}$$

(5) When $|e(k)| \leq \varepsilon$, which indicates the absolute value of error tends to very small, we can use PI controller to decrease the static error,
where

$u(k)$—control input at time k;
$u(k-1)$—control input at time $k-1$;
k_1—gain coefficient, $k_1 > 1$;
k_2—gain coefficient, $0 < k_2 < 1$;
M_1, M_2—limit values, $M_1 > M_2$;
ε—positive value.

2.2 Simulation Example

Consider a plant as

$$G_{\mathrm{p}}(s) = \frac{523500}{s^3 + 87.35s^2 + 10470s}$$

The sampling time is 1 ms, using MATLAB command "c2d", the plant can be discrete as

$$\begin{aligned} y(k) = &-\mathrm{den}(2)y(k-1) - \mathrm{den}(3)y(k-2) - \mathrm{den}(4)y(k-3) \\ &+ \mathrm{num}(2)u(k-1) + \mathrm{num}(3)u(k-2) + \mathrm{num}(4)u(k-3) \end{aligned}$$

where `num() and den() can be gotten by the command tfdata.`

The ideal position signal is $y_{\mathrm{d}}(k) = 1.0$. In the simulation program, due to the discretization, there is one delay in control input.

The simulation program of traditional PID controller is chap2_1.m, and the simulation results are shown in Figs. 2.2 and 2.3. The simulation program of expert PID controller is chap2_2.m, and the simulation results are shown in Figs. 2.4 and 2.5.

(1) **Program of traditional PID Controller: chap2_1.m**.

```
%Expert PID Controller
clear all;
close all;
ts=0.001;
sys=tf(5.235e005,[1,87.35,1.047e004,0]); %Plant
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');
```

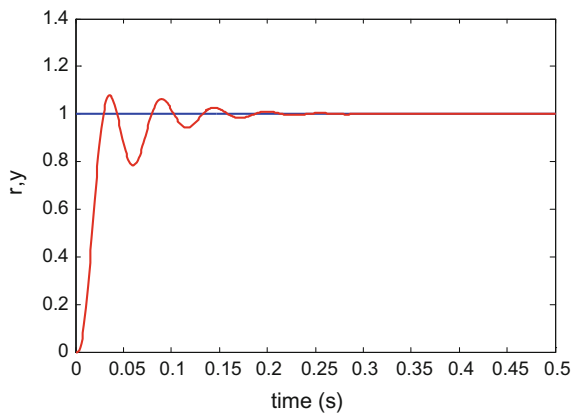


Fig. 2.2 Step response with traditional PID control

Fig. 2.3 Error response with traditional PID control

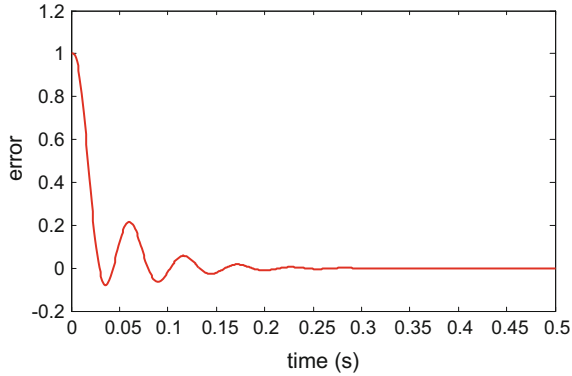


Fig. 2.4 Step response with expert PID control

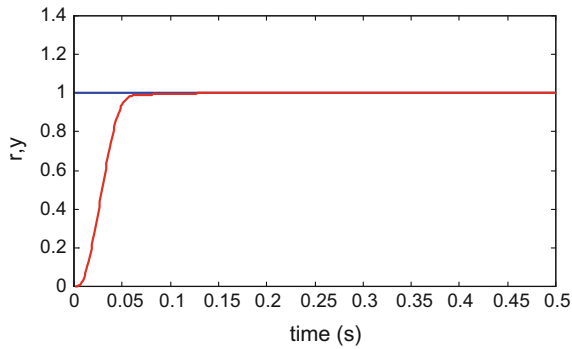
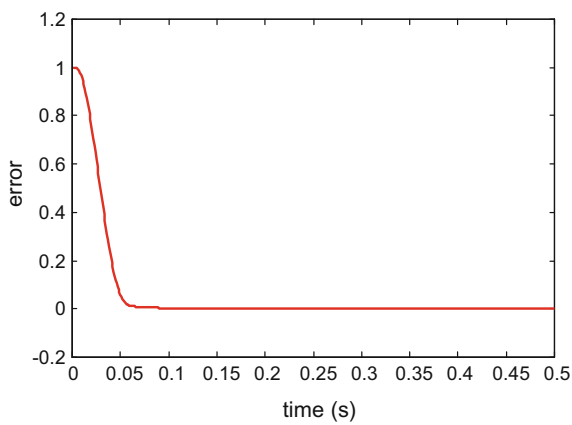


Fig. 2.5 Error response with expert PID control



```
u_1=0;u_2=0;u_3=0;u_4=0;
y_1=0;y_2=0;y_3=0;
ei=0;

kp=0.96;ki=0.03;kd=0.01;

error_1=0;
for k=1:1:500
time(k)=k*ts;

yd(k)=1.0;
y(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(1)*u_1+num(2)*u_2+num(3)
*u_3+num(4)*u_4;

error(k)=yd(k)-y(k);          % Calculating P
derror(k)=error(k)-error_1;     % Calculating D
ei=ei+error(k)*ts;

u(k)=kp*error(k)+kd*derror(k)/ts+ki*ei; %PID Controller

u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=y(k);
error_1=error(k);
end
figure(1);
plot(time,yd,'b',time,y,'r','linewidth',2);
xlabel('time(s)');ylabel('r,y');
figure(2);
plot(time,yd-y,'r','linewidth',2);
xlabel('time(s)');ylabel('error');
```

(2) **Program of expert PID control: chap2_2.m**.

```
%Expert PID Controller
clear all;
close all;
ts=0.001;

sys=tf(5.235e005,[1,87.35,1.047e004,0]); %Plant
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;u_3=0;u_4=0;
y_1=0;y_2=0;y_3=0;
ei=0;
error_1=0;derror_1=0;
```

```
kp=0.6;ki=0.03;kd=0.01;
for k=1:1:500
time(k)=k*ts;

yd(k)=1.0;              %Tracing Step Signal
%Linear model
y(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(1)*u_1+num(2)*u_2+num(3)
*u_3+num(4)*u_4;

error(k)=yd(k)-y(k);          % Calculating P
derror(k)=error(k)-error_1;     % Calculating D
ei=ei+error(k)*ts;           % Calculating I
u(k)=kp*error(k)+kd*derror(k)/ts+ki*ei;  %PID Controller

%Expert control rule
if abs(error(k))>0.8    %Rule1:Unclosed control rule
  u(k)=0.45;
elseif abs(error(k))>0.40
  u(k)=0.40;
elseif abs(error(k))>0.20
  u(k)=0.12;
elseif abs(error(k))>0.01
  u(k)=0.10;
end

if error(k)*derror(k)>0|(derror(k)==0)     %Rule2
  if abs(error(k))>=0.05
    u(k)=u_1+2*kp*error(k);
  else
    u(k)=u_1+0.4*kp*error(k);
  end
end

if (error(k)*derror(k)<0&derror(k)*derror_1>0)|(error(k)==0)  %Rule3
   u(k)=u(k);
end

if error(k)*derror(k)<0&derror(k)*derror_1<0  %Rule4
  if abs(error(k))>=0.05
    u(k)=u_1+2*kp*error(k);
  else
    u(k)=u_1+0.6*kp*error(k);
  end
end

if abs(error(k))<=0.001  %Rule5:Integration separation PI control
  u(k)=0.5*error(k)+0.010*ei;
end
```

```
u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=y(k);
error_1=error(k);
derror_1=derror(k);
end
figure(1);
plot(time,yd,'r',time,y,'b:','linewidth',2);
xlabel('time(s)');ylabel('r,y');
legend('Ideal position','Practical position');
```

Reference

1. K.J. Astrom, J.J. Anton, K.E. Arzen, Expert control, Automatica **22**(3), 277–286 (1986)

Chapter 3
Foundation of Fuzzy Mathematics

Fuzzy theory was initiated by L.A. Zadeh in 1965 with his seminal paper "Fuzzy Sets" [1]. In the early 1960s, he thought that classical control theory had put too much emphasis on precision and therefore could not handle the complex systems. As early as 1962, he wrote that to handle biological systems "we need a radically different kind of mathematics, the mathematics of fuzzy or cloudy quantities which are not describable in terms of probability distributions." Later, he formalized the ideas into the paper "fuzzy sets."

Fuzzy sets are the mathematic foundation of fuzzy control.

3.1 Characteristic Function and Membership Function

(1) Characteristic function

$$\mu_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases} \tag{3.1}$$

(2) Membership function

$$\mu_A(x) = \begin{cases} 1 & x \in A \\ (0,1) & x \in A \text{ partly} \\ 0 & x \notin A \end{cases} \tag{3.2}$$

where A is a fuzzy set, which consists of 0, 1, and $\mu_A(x)$, the range of $\mu_A(x)$ is $[0, 1]$.

Membership function is the foundation of fuzzy sets.

3.2 Fuzzy Set Expression

There are two kinds of expression as follows:

(1) Fuzzy set $\boldsymbol{A}$ consists of discrete element:

$$\boldsymbol{A} = \mu_1/x_1 + \mu_2/x_2 + \cdots + \mu_i/x_i + \cdots \tag{3.3}$$

or

$$\boldsymbol{A} = \{(x_1, \mu_1), (x_2, \mu_2), \ldots, (x_i, \mu_i), \ldots\} \tag{3.4}$$

(2) Fuzzy set A consists of continuous function (Membership Function) $\mu_A(x)$:

$$A = \int \mu_A(x)/x \tag{3.5}$$

Ex. 3.1 To fuzzy the age, we assume the scope of the age is $\boldsymbol{X} = [0, 200]$; Zadeh gave "Young" fuzzy set Y as

$$Y(x) = \begin{cases} 1.0 & 0 \le x \le 25 \\ \left[1 + \left(\frac{x-25}{5}\right)^2\right]^{-1} & 25 < x \le 100 \end{cases}$$

MF is shown in Fig. 3.1.

Program of MF for "Young": chap3_1.m.

```
%Membership function for Young People
clear all;
close all;


for k=1:1:1001
   x(k)=(k-1)*0.10;
if x(k)>=0&x(k)<=25
  y(k)=1.0;
else
  y(k)=1/(1+((x(k)-25)/5)^2);
end
end
plot(x,y,'k');
xlabel('X Years');ylabel('Degree of membership');
```

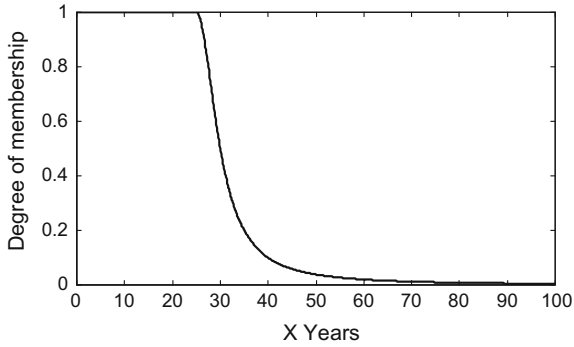


Fig. 3.1 Membership function of "Young"

3.3 Calculation Method of Fuzzy Set

3.3.1 Basic Calculation Method of Fuzzy Set

For fuzzy set $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$, basic calculation methods are given as follows:

(1) Null set

$$\boldsymbol{A} = \varphi \Leftrightarrow \mu_A(u) = 0 \tag{3.6}$$

(2) Full set

$$\boldsymbol{A} = \boldsymbol{E} \Leftrightarrow \mu_A(u) = 1 \tag{3.7}$$

(3) Equal set

$$\boldsymbol{A} = \boldsymbol{B} \Leftrightarrow \mu_A(u) = \mu_B(u) \tag{3.8}$$

(4) Complement set
If $\bar{\boldsymbol{A}}$ is complement set of $\boldsymbol{A}$, then

$$\bar{\boldsymbol{A}} \Leftrightarrow \mu_{\bar{A}}(u) = 1 - \mu_A(u) \tag{3.9}$$

(5) Subset
If $\boldsymbol{B}$ is subset of $\boldsymbol{A}$, then

$$\boldsymbol{B} \subseteq \boldsymbol{A} \Leftrightarrow \mu_B(u) \leq \mu_A(u) \tag{3.10}$$

(6) Fuzzy union set
If $\boldsymbol{C}$ is union set of $\boldsymbol{A}$ and $\boldsymbol{B}$, then

$$\boldsymbol{C} = \boldsymbol{A} \cup \boldsymbol{B}$$

$$\boldsymbol{A} \cup \boldsymbol{B} = \mu_{A \cup B}(u) = \max(\mu_A(u), \mu_B(u)) = \mu_A(u) \vee \mu_B(u) \tag{3.11}$$

(7) Intersection set
If $\boldsymbol{C}$ is intersection set of $\boldsymbol{A}$ and $\boldsymbol{B}$, then

$$\boldsymbol{C} = \boldsymbol{A} \cap \boldsymbol{B}$$

$$\boldsymbol{A} \cap \boldsymbol{B} = \mu_{A \cap B}(u) = \min(\mu_A(u), \mu_B(u)) = \mu_A(u) \wedge \mu_B(u) \tag{3.12}$$

Ex. 3.2 $\boldsymbol{A} = \frac{0.9}{u_1} + \frac{0.2}{u_2} + \frac{0.8}{u_3} + \frac{0.5}{u_4}$, $\boldsymbol{B} = \frac{0.3}{u_1} + \frac{0.1}{u_2} + \frac{0.4}{u_3} + \frac{0.6}{u_4}$
Then,

$$\boldsymbol{A} \cup \boldsymbol{B} = \frac{0.9}{u_1} + \frac{0.2}{u_2} + \frac{0.8}{u_3} + \frac{0.6}{u_4},\ \boldsymbol{A} \cap \boldsymbol{B} = \frac{0.3}{u_1} + \frac{0.1}{u_2} + \frac{0.4}{u_3} + \frac{0.5}{u_4}$$

Ex. 3.3 If $\mu_A(u) = 0.4$, then

$$\mu_{\bar{A}}(u) = 1 - 0.4 = 0.6$$

$$\mu_A(u) \vee \mu_{\bar{A}}(u) = 0.4 \vee 0.6 = 0.6 \neq 1$$
$$\mu_A(u) \wedge \mu_{\bar{A}}(u) = 0.4 \wedge 0.6 = 0.4 \neq 0$$

3.3.2 Fuzzy Operator

For fuzzy set $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$, there are three kinds of conventional operators as follows.

1. Fuzzy intersection operator
For $\boldsymbol{C} = \boldsymbol{A} \cap \boldsymbol{B}$, there are three intersection operators as follows:

(1) Basic intersection operator

$$\mu_c(x) = \text{Min}\{\mu_A(x), \mu_B(x)\} \tag{3.13}$$

(2) Algebra product operator

$$\mu_c(x) = \mu_A(x) \cdot \mu_B(x) \tag{3.14}$$

(3) Limitary product operator

$$\mu_c(x) = \text{Max}\{0, \mu_A(x) + \mu_B(x) - 1\} \tag{3.15}$$

2. Fuzzy union operator
For $\boldsymbol{C} = \boldsymbol{A} \cup \boldsymbol{B}$, there are three fuzzy union operators as follows:

(1) Basic union operator

$$\mu_c(x) = \text{Max}\{\mu_A(x), \mu_B(x)\} \tag{3.16}$$

(2) Probability OR operator

$$\mu_c(x) = \mu_A(x) + \mu_B(x) - \mu_A(x) \times \mu_B(x) \tag{3.17}$$

(3) Limitary sum operator

$$\mu_c(x) = \text{Min}\{1, \mu_A(x) + \mu_B(x)\} \tag{3.18}$$

3. Fuzzy balanceable operator
For $\boldsymbol{C} = \boldsymbol{A} \circ \boldsymbol{B}$, the balanceable operator is

$$\mu_c(x) = [\mu_A(x) \cdot \mu_B(x)]^{1-\gamma} \cdot [1 - (1 - \mu_A(x)) \cdot (1 - \mu_B(x))]^{\gamma} \tag{3.19}$$

where γ is in $[0 \quad 1]$. When $\gamma = 0$, $\mu_c(x) = \mu_A(x) \cdot \mu_B(x)$, that is, $\boldsymbol{C} = \boldsymbol{A} \cap \boldsymbol{B}$.
When $\gamma = 1$, $\mu_c(x) = \mu_A(x) + \mu_B(x) - \mu_A(x) \times \mu_B(x)$, that is, $\boldsymbol{C} = \boldsymbol{A} \cup \boldsymbol{B}$.

3.3.3 Typical Membership Function

In fuzzy system, we often use six kinds of typical MF to fuzzify a variable, which are given as follows:

(1) Gaussian MF

$$f(x, \sigma, c) = \mathrm{e}^{-\frac{(x-c)^2}{2\sigma^2}} \tag{3.20}$$

where σ is a positive constant.
The MATLAB function for Gaussian MF is gaussmf(x, $[\sigma, c]$).

(2) Campanulate MF

$$f(x, a, b, c) = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}} \tag{3.21}$$

(3) S-type MF

$$f(x,a,c)=\frac{1}{1+\mathrm{e}^{-a(x-c)}} \tag{3.22}$$

The MATLAB function for S-type MF is sigmf$(x,[a,c])$.

(4) Trapezoid MF

$$f(x,a,b,c,d)=\begin{cases}0 & x\le a\\ \frac{x-a}{b-a} & a\le x\le b\\ 1 & b\le x\le c\\ \frac{d-x}{d-c} & c\le x\le d\\ 0 & x\ge d\end{cases} \tag{3.23}$$

The MATLAB function for trapezoid MF is trapmf$(x,[a,b,c,d])$.

(5) Triangle MF

$$f(x,a,b,c)=\begin{cases}0 & x\le a\\ \frac{x-a}{b-a} & a\le x\le b\\ \frac{c-x}{c-b} & b\le x\le c\\ 0 & x\ge c\end{cases} \tag{3.24}$$

The MATLAB function for triangle MF is trimf$(x,[a,b,c])$.

(6) Z-type MF

The MATLAB function for Z-type MF is zmf$(x,[a,b])$.

Ex. 3.4 Six kinds of MF simulations, $x\in[0,10]$.

The simulation results for the above six types of MFs are shown in Figs. 3.2, 3.3, 3.4, 3.5, 3.6, and 3.7.

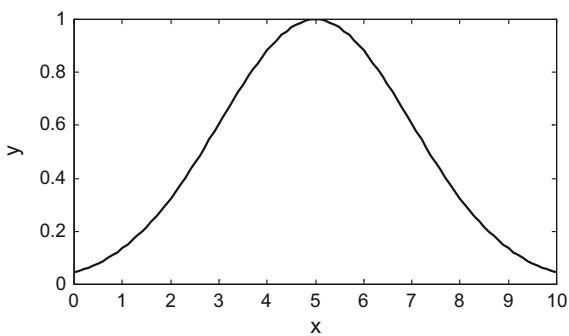


Fig. 3.2 Gaussian MF (M = 1)

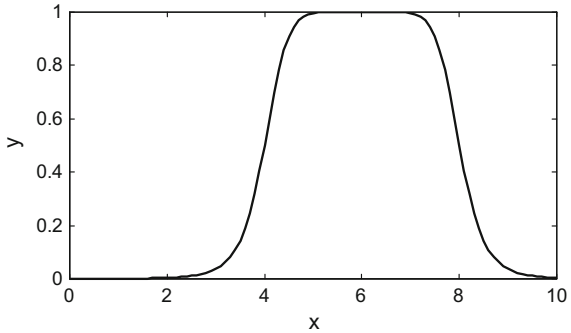


Fig. 3.3 Campanulate MF (M = 2)

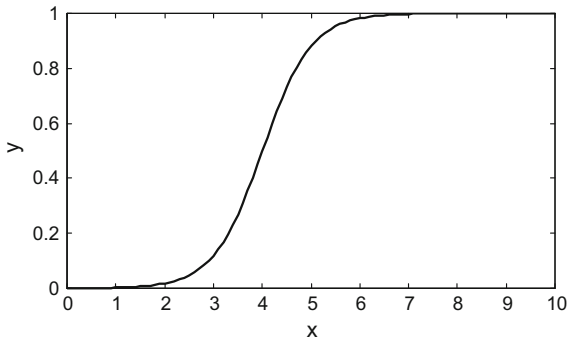


Fig. 3.4 S-type MF ($M = 3$)

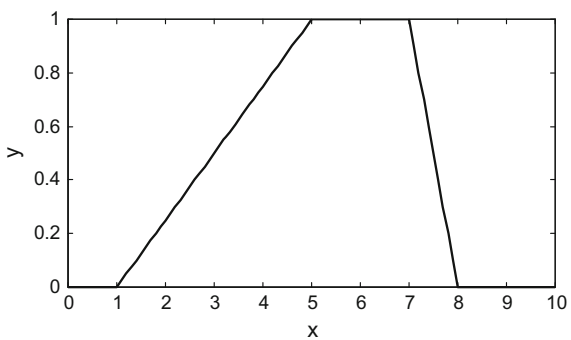


Fig. 3.5 Trapezoid MF ($M = 4$)

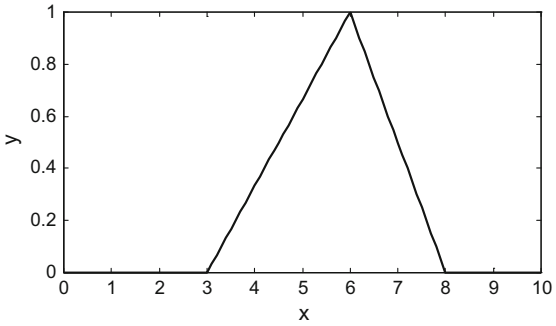


Fig. 3.6 Triangle MF ($M = 5$)

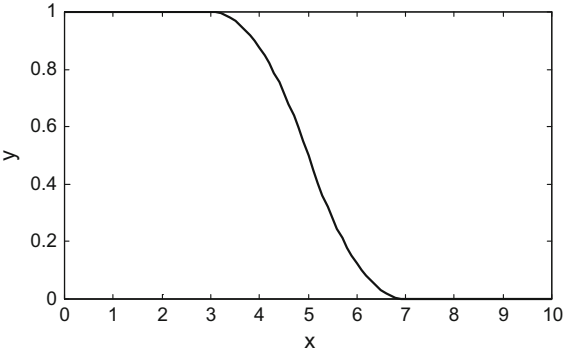


Fig. 3.7 Z-type MF ($M = 6$)

MF design program: chap3_2.m

```
%Membership function
clear all;
close all;
M=6;


if M==1       %Guassian membership function
   x=0:0.1:10;
   y=gaussmf(x,[2 5]);
   plot(x,y,'k');
   xlabel('x');ylabel('y');
elseif M==2    %General Bell membership function
   x=0:0.1:10;
   y=gbellmf(x,[2 4 6]);
   plot(x,y,'k');
   xlabel('x');ylabel('y');
elseif M==3    %S membership function
   x=0:0.1:10;
```

```
   y=sigmf(x,[2 4]);
   plot(x,y,'k');
   xlabel('x');ylabel('y');
elseif M==4    %Trapezoid membership function
   x=0:0.1:10;
   y=trapmf(x,[1 5 7 8]);
   plot(x,y,'k');
   xlabel('x');ylabel('y');
elseif M==5    %Triangle membership function
   x=0:0.1:10;
   y=trimf(x,[3 6 8]);
   plot(x,y,'k');
   xlabel('x');ylabel('y');
elseif M==6    %Z membership function
   x=0:0.1:10;
   y=zmf(x,[3 7]);
   plot(x,y,'k');
   xlabel('x');ylabel('y');
end
```

3.3.4 Design of Fuzzy System

We can use several fuzzy sets to fuzzify a variable.

Ex. 3.5 To describe the error from negative big to positive big with seven fuzzy sets.

To fuzzify the error e, we can define seven fuzzy sets as

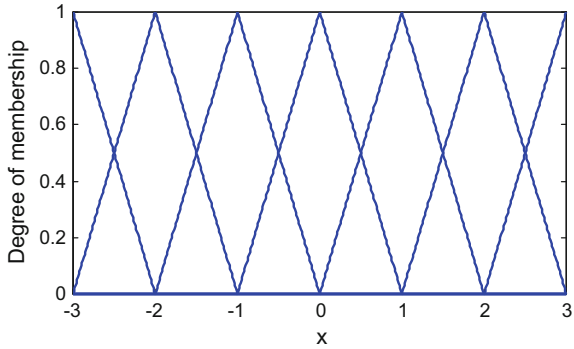
$$\underset{\sim}{e} = \{\text{NB}, \text{NM}, \text{NS}, \text{ZO}, \text{PS}, \text{PM}, \text{PB}\}$$

Use triangle MF, consider variable e varies in the range $[-3, 3]$, we can describe a fuzzy system by seven fuzzy sets. The simulation result is shown in Fig. 3.8.

Program of fuzzy system design: chap3_3.m

```
%Define N+1 triangle membership function
clear all;
close all;
N=6;


x=-3:0.01:3;
for i=1:N+1
   f(i)=-3+6/N*(i-1);
```

Fig. 3.8 Fuzzy system with triangle MF




```
end
u=trimf(x,[f(1),f(1),f(2)]);


figure(1);
plot(x,u);
for j=2:N
   u=trimf(x,[f(j-1),f(j),f(j+1)]);
   hold on;
   plot(x,u);
end
u=trimf(x,[f(N),f(N+1),f(N+1)]);
hold on;
plot(x,u);
xlabel('x');
ylabel('Degree of membership');
```

3.4 Fuzzy Matrix Calculation

3.4.1 Fuzzy Matrix

As an example, Fuzzy matrix is defined to describe the level of different students.

Ex. 3.6 Consider three students and four lessons, $X = \{\text{Zhang, Li, Wang}\}$, $Y = \{\text{English, Math, Physics, Chemistry}\}$, the scores are given in Table 3.1.

To fuzzify the scores, we can define MF as $\mu(u) = \frac{u}{100}$, where u is score, and then, we can get fuzzy relation matrix $\boldsymbol{R}$ as shown in Table 3.2.

Table 3.1 Scores

Name	Lesson			
	English	Math	Physics	Chemistry
Zhang	70	90	80	65
Wang	90	85	76	70
Li	50	95	85	80

Table 3.2 Fuzzy matrix **R**

Name	Lesson			
	English	Math	Physics	Chemistry
Zhang	0.70	0.90	0.80	0.65
Li	0.90	0.85	0.76	0.70
Wang	0.50	0.95	0.85	0.80

From Table 3.2, we can get fuzzy matrix $\boldsymbol{R}$ as

$$\boldsymbol{R} = \begin{bmatrix} 0.70 & 0.90 & 0.80 & 0.65 \\ 0.90 & 0.85 & 0.76 & 0.70 \\ 0.50 & 0.95 & 0.85 & 0.80 \end{bmatrix}$$

3.4.2 *Fuzzy Matrix Calculation*

For fuzzy matrix $\boldsymbol{A}$ and $\boldsymbol{B}$, $\boldsymbol{A} = (a_{ij})$, $\boldsymbol{B} = (b_{ij})$, $i,j = 1, 2, \cdots, n$, we can define several fuzzy matrix calculation methods.

(1) Fuzzy equality calculation
For $a_{ij} = b_{ij}$, $\boldsymbol{A} = \boldsymbol{B}$
(2) Fuzzy subset calculation
For $a_{ij} \leq b_{ij}$, $\boldsymbol{A} \subseteq \boldsymbol{B}$
(3) Fuzzy set union calculation
For $c_{ij} = a_{ij} \vee b_{ij}$, $\boldsymbol{C} = (c_{ij})$, $\boldsymbol{C} = \boldsymbol{A} \cup \boldsymbol{B}$
(4) Fuzzy set interaction calculation
For $c_{ij} = a_{ij} \wedge b_{ij}$, $\boldsymbol{C} = (c_{ij})$, $\boldsymbol{C} = \boldsymbol{A} \cap \boldsymbol{B}$
(5) Fuzzy complementary set calculation
For $c_{ij} = 1 - a_{ij}$, $\boldsymbol{C} = (c_{ij})$ is defined as complimentary set of $\boldsymbol{A}$, $\boldsymbol{C} = \bar{\boldsymbol{A}}$.

Ex. 3.7 $\boldsymbol{A} = \begin{bmatrix} 0.7 & 0.1 \\ 0.3 & 0.9 \end{bmatrix}$, $\boldsymbol{B} = \begin{bmatrix} 0.4 & 0.9 \\ 0.2 & 0.1 \end{bmatrix}$

$$\boldsymbol{A} \cup \boldsymbol{B} = \begin{bmatrix} 0.7 \vee 0.4 & 0.1 \vee 0.9 \\ 0.3 \vee 0.2 & 0.9 \vee 0.1 \end{bmatrix} = \begin{bmatrix} 0.7 & 0.9 \\ 0.3 & 0.9 \end{bmatrix}$$

$$\boldsymbol{A} \cap \boldsymbol{B} = \begin{bmatrix} 0.7 \wedge 0.4 & 0.1 \wedge 0.9 \\ 0.3 \wedge 0.2 & 0.9 \wedge 0.1 \end{bmatrix} = \begin{bmatrix} 0.4 & 0.1 \\ 0.2 & 0.1 \end{bmatrix}$$

$$\bar{\boldsymbol{A}} = \begin{bmatrix} 1-0.7 & 1-0.1 \\ 1-0.3 & 1-0.9 \end{bmatrix} = \begin{bmatrix} 0.3 & 0.9 \\ 0.7 & 0.1 \end{bmatrix}$$

3.4.3 Compound of Fuzzy Matrix

If $\boldsymbol{A}$ is defined as fuzzy relation in $x \times y$, $\boldsymbol{B}$ is defined as fuzzy relation in $y \times z$, then $\boldsymbol{C} = \boldsymbol{A} \circ \boldsymbol{B}$ is a compound of fuzzy matrix $\boldsymbol{A}$ and $\boldsymbol{B}$, and $\boldsymbol{A} = (a_{ik})$, $\boldsymbol{B} = (b_{kj})$, $\boldsymbol{C} = (c_{ij})$, $i,j,k = 1,2, \ldots, n$

$$c_{ij} = \bigvee_k \{a_{ik} \wedge b_{kj}\}. \tag{3.25}$$

Ex. 3.8 $\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$, $\boldsymbol{B} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$, then

$$\boldsymbol{C} = \boldsymbol{A} \circ \boldsymbol{B} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$$c_{11} = (a_{11} \wedge b_{11}) \vee (a_{12} \wedge b_{21})$$

$$c_{12} = (a_{11} \wedge b_{12}) \vee (a_{12} \wedge b_{22})$$

$$c_{21} = (a_{21} \wedge b_{11}) \vee (a_{22} \wedge b_{21})$$

$$c_{22} = (a_{21} \wedge b_{12}) \vee (a_{22} \wedge b_{22})$$

When $\boldsymbol{A} = \begin{bmatrix} 0.8 & 0.7 \\ 0.5 & 0.3 \end{bmatrix}$, $\boldsymbol{B} = \begin{bmatrix} 0.2 & 0.4 \\ 0.6 & 0.9 \end{bmatrix}$, we have

$$\boldsymbol{A} \circ \boldsymbol{B} = \begin{bmatrix} 0.6 & 0.7 \\ 0.3 & 0.4 \end{bmatrix}$$

$$\boldsymbol{B} \circ \boldsymbol{A} = \begin{bmatrix} 0.4 & 0.3 \\ 0.6 & 0.6 \end{bmatrix}$$

We can get the conclusion as

$$\boldsymbol{A} \circ \boldsymbol{B} \neq \boldsymbol{B} \circ \boldsymbol{A}$$

Program of fuzzy matrix compound calculation: chap3_4.m

```
clear all;
close all;
A=[0.8,0.7;
  0.5,0.3];
B=[0.2,0.4;
  0.6,0.9];
%Compound of A and B
for i=1:2
  for j=1:2
    AB(i,j)=max(min(A(i,:),B(:,j)'))
   end
end


%Compound of B and A
for i=1:2
  for j=1:2
    BA(i,j)=max(min(B(i,:),A(:,j)'))
   end
end
```

Now, an example of application of fuzzy matrix compound is given as follows. For a family, consider the similarity between grandson and grandparents or between granddaughter and grandparents. The similarity relationship between children and their parents and the similarity relationship between the parents and grandparents are given in Tables 3.3 and 3.4, respectively.

The relationship in Table 3.3 can be expressed by fuzzy matrix as

$$\boldsymbol{R} = \begin{pmatrix} 0.2 & 0.8 \\ 0.6 & 0.1 \end{pmatrix}$$

Table 3.3 Similarity relationship between children and their parents

MF	Father	Mother
Son	0.2	0.8
Daughter	0.6	0.1

Table 3.4 Similarity relationship between the parents and grandparents

MF	Grandfather	Grandmother
Father	0.5	0.7
Mother	0.1	0

The relationship in Table 3.4 can be expressed by fuzzy matrix as

$$\boldsymbol{S} = \begin{pmatrix} 0.5 & 0.7 \\ 0.1 & 0 \end{pmatrix}$$

For this case, the operation of fuzzy relation synthesis between $\boldsymbol{R}$ and $\boldsymbol{S}$ is

$$\begin{aligned} \boldsymbol{R} \circ \boldsymbol{S} &= \begin{pmatrix} 0.2 & 0.8 \\ 0.6 & 0.1 \end{pmatrix} \circ \begin{pmatrix} 0.5 & 0.7 \\ 0.1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} (0.2 \wedge 0.5) \vee (0.8 \wedge 0.1) & (0.2 \wedge 0.7) \vee (0.8 \wedge 0) \\ (0.6 \wedge 0.5) \vee (0.1 \wedge 0.1) & (0.6 \wedge 0.7) \vee (0.1 \wedge 0) \end{pmatrix} = \begin{pmatrix} 0.2 & 0.2 \\ 0.5 & 0.6 \end{pmatrix} \end{aligned}$$

The results indicate that the similarity degree between grandson and grandfather is 0.2, the similarity degree between grandson and grandmother is 0.2. The similarity between granddaughter and grandfather is 0.5, and the similarity between granddaughter and grandmother is 0.5.

3.5 Fuzzy Inference

Consider fuzzy inference for the fuzzy rule "If $\boldsymbol{A}$ AND $\boldsymbol{B}$ then $\boldsymbol{C}$," which means $(\boldsymbol{A} \wedge \boldsymbol{B} \rightarrow \boldsymbol{C})$, $\boldsymbol{A} \in \mathrm{U}$, $\boldsymbol{B} \in \mathrm{U}$, $\boldsymbol{C} \in \mathrm{U}$.

According to Mamdani fuzzy inference method, we can get fuzzy relation matrix $\boldsymbol{R}$ as

$$\boldsymbol{R} = (\boldsymbol{A} \times \boldsymbol{B})^{\mathrm{T}_1} \times \boldsymbol{C} \tag{3.26}$$

where T_1 is the column vector transformation.

For new A,B,C, using fuzzy matrix $\boldsymbol{R}$, we can get $\boldsymbol{C}_1$ by $\boldsymbol{A}_1$ and $\boldsymbol{B}_1$:

$$\boldsymbol{C}_1 = (\boldsymbol{A}_1 \times \boldsymbol{B}_1)^{\mathrm{T}_2} \times \boldsymbol{R} \tag{3.27}$$

where T_2 is the row vector transformation.

Ex. 3.9 $\boldsymbol{X} = \{a_1, \quad a_2, \quad a_3\}$, $\boldsymbol{Y} = \{b_1, \quad b_2, \quad b_3\}$, $\boldsymbol{Z} = \{c_1, \quad c_2, \quad c_3\}$, $\boldsymbol{A} = \frac{0.5}{a_1} + \frac{1}{a_2} + \frac{0.1}{a_3}$, $\boldsymbol{B} = \frac{0.1}{b_1} + \frac{1}{b_2} + \frac{0.6}{a_3}$, $\boldsymbol{C} = \frac{0.4}{c_1} + \frac{1}{c_2}$. The fuzzy relation of "If $\boldsymbol{A}$ AND $\boldsymbol{B}$ then $\boldsymbol{C}$" is $\boldsymbol{R}$.

$$\boldsymbol{A} \times \boldsymbol{B} = \begin{bmatrix} 0.5 \\ 1 \\ 0.1 \end{bmatrix} \circ [0.1 \quad 1 \quad 0.6] = \begin{bmatrix} 0.1 & 0.5 & 0.5 \\ 0.1 & 1.0 & 0.6 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}$$

$$(\boldsymbol{A}\times\boldsymbol{B})^{\mathrm{T}_1}=[0.1\quad 0.5\quad 0.5\quad 0.1\quad 1.0\quad 0.6\quad 0.1\quad 0.1\quad 0.1]^{\mathrm{T}}$$

$$\boldsymbol{R}=(\boldsymbol{A}\times\boldsymbol{B})^{\mathrm{T}_1}\times\boldsymbol{C}=[0.1\quad 0.5\quad 0.5\quad 0.1\quad 1.0\quad 0.6\quad 0.1\quad 0.1\quad 0.1]^{\mathrm{T}}\circ[0.4\quad 1]$$

$$=\begin{bmatrix}0.1 & 0.4 & 0.4 & 0.1 & 0.4 & 0.4 & 0.1 & 0.1 & 0.1\\ 0.1 & 0.5 & 0.5 & 0.1 & 1 & 0.6 & 0.1 & 0.1 & 0.1\end{bmatrix}^{\mathrm{T}}$$

$$\boldsymbol{A}_1\times\boldsymbol{B}_1=\begin{bmatrix}1\\0.5\\0.1\end{bmatrix}\circ[0.1\quad 0.5\quad 1]=\begin{bmatrix}0.1 & 0.5 & 1\\0.1 & 0.5 & 0.5\\0.1 & 0.1 & 0.1\end{bmatrix}$$

$$(\boldsymbol{A}\times\boldsymbol{B})^{\mathrm{T}_2}=[0.1\quad 0.5\quad 1\quad 0.1\quad 0.5\quad 0.5\quad 0.1\quad 0.1\quad 0.1]$$

$$\boldsymbol{C}_1=[0.1\quad 0.5\quad 1\quad 0.1\quad 0.5\quad 0.5\quad 0.1\quad 0.1\quad 0.1]\circ\begin{bmatrix}0.1 & 0.4 & 0.4 & 0.1 & 0.4 & 0.4 & 0.1 & 0.1 & 0.1\\ 0.1 & 0.5 & 0.5 & 0.1 & 1 & 0.6 & 0.1 & 0.1 & 0.1\end{bmatrix}^{T}$$

$$=[0.4\quad 0.5]$$

$$\boldsymbol{C}_1=\frac{0.4}{c_1}+\frac{0.5}{c_2}$$

Fuzzy inference program: chap3_5.m

```
clear all;
close all;

A=[0.5;1;0.1];
B=[0.1,1,0.6];
C=[0.4,1];

%Compound of A and B
for i=1:3
  for j=1:3
    AB(i,j)=min(A(i),B(j));
  end
end
%Transfer to Column
T1=[];
for i=1:3
  T1=[T1;AB(i,:)'];
end
%Get fuzzy R
for i=1:9
  for j=1:2
```

```
    R(i,j)=min(T1(i),C(j));
  end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
A1=[1,0.5,0.1];
B1=[0.1,0.5,1];

for i=1:3
  for j=1:3
    AB1(i,j)=min(A1(i),B1(j));
  end
end
%Transfer to Row
T2=[];
for i=1:3
  T2=[T2,AB1(i,:)];
end
%Get output C1
for i=1:9
  for j=1:2
    D(i,j)=min(T2(i),R(i,j));
    C1(j)=max(D(:,j));
  end
end
```

3.6 Fuzzy Equation

The fuzzy relation equation is an equation of the form $\boldsymbol{A} \circ \boldsymbol{R} = \boldsymbol{B}$, where $\boldsymbol{A}$ and $\boldsymbol{B}$ are fuzzy sets, $\boldsymbol{R}$ is a fuzzy relation, and $\boldsymbol{A} \circ \boldsymbol{R}$ stands for the composition of $\boldsymbol{A}$ with $\boldsymbol{R}$.

Ex. 3.10 Solve $x_i (i = 1, 2, 3)$ for fuzzy equation $\begin{pmatrix} 0.6 & 0.2 & 0.4 \end{pmatrix} \circ \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = 0.4.$

From the equation, we have

$$(0.6 \wedge x_1) \vee (0.2 \wedge x_2) \vee (0.4 \wedge x_3) = 0.4$$

Since $(0.2 \wedge x_2) < 0.4$, we have $x_2 = [0, 1]$; then, we get

$$(0.6 \wedge x_1) \vee (0.4 \wedge x_3) = 0.4$$

We can consider two conditions as follows:

① $0.6 \wedge x_1 = 0.4,\ 0.4 \wedge x_3 \leq 0.4$, then

$$x_1 = 0.4,\ x_3 = [0, 1]$$

② $0.6 \wedge x_1 \leq 0.4,\ 0.4 \wedge x_3 = 0.4$, then

$$x_1 = [0, 0.4],\ x_3 = [0.4, 1]$$

Reference

1. L.A. Zadeh, Fuzzy sets. Inf. Control **8**(3), 338–353 (1965)

Chapter 4
Fuzzy Logic Control

The term fuzzy logic was introduced with the 1965 proposal of fuzzy set theory by Zadeh [1]. Fuzzy logic has been applied to many fields, from control theory to artificial intelligence.

By contrast, in Boolean logic, the truth values of variables may only be the integer values 0 or 1. Fuzzy logic employed to handle the concept of partial truth, where the truth value may range between completely true and completely false. Furthermore, when fuzzy linguistic variables are used, the degrees may be managed by specific (membership) functions [2].

4.1 Design of Fuzzy Logic Controller

(1) **Fuzzify a variable**

To fuzzify the input variable $\underset{\sim}{e}$, we often use three kinds of fuzzy sets as follows:

- $\underset{\sim}{e}$ = {NB, NS, ZO, PS, PB}
- $\underset{\sim}{e}$ = {NB, NM, NS, ZO, PS, PM, PB}
- $\underset{\sim}{e}$ = {NB, NM, NS, NZ, PZ, PS, PM, PB}

For example, a fuzzy system with eight fuzzy sets using triangle MF is shown in Fig. 4.1.

(2) **Rule base—RB**

Rule base consists of several fuzzy rules. For example, in fuzzy logic control, we can design RB with two fuzzy rules as follows:

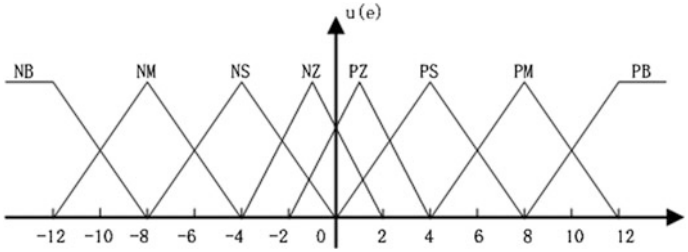


Fig. 4.1 Eight triangle fuzzy sets

R_1 : IF E is NB E EC is NB then U is PB

R_2 : IF E is NB and EC is NS then U is PM

where E represents error, EC represents error change, U represents control input.

(3) **Fuzzy inference and Defuzzy**

From the rule base, we can get fuzzy relation matrix **R**. If we have new premise information **A** and **B**, we can get a new result:

$$\mathbf{C} = (\mathbf{A} \times \mathbf{B}) \circ \mathbf{R} \tag{4.1}$$

The conclusion matrix **C** is fuzzy vector, and it must be defuzzified to exact value for practical use.

4.2 An Example for a Fuzzy Logic Controller Design

Consider the height-level control of water tank as shown in Fig. 4.2, and we can design two kinds of fuzzy rules as:

"if level is higher than O, then drain";

"if level is lower than O, then effuse";

where O represents ideal level.

We can design a fuzzy controller as in the following steps.

(1) **Define error**

In Fig. 4.2, for the height-level control of water tank, the error is defined as

$$e = \Delta h = h_0 - h$$

where h_0 is ideal height and h is practical height.

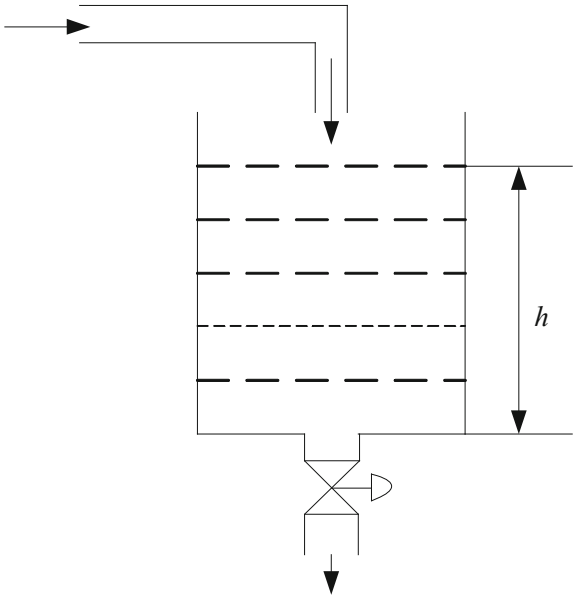


Fig. 4.2 Fuzzy control for height level of water tank

(2) **Fuzzify input and output**

We can use five fuzzy sets to fuzzify error e, that is, negative big (NB), negative small (NS), zero (O), positive small (PS), positive big (PB), and we define the error as seven levels: −3, −2, −1, 0, +1, +2, +3, which is shown in Table 4.1.

To fuzzify control input, we define five fuzzy sets to describe u, that is, negative big (NB), negative small (NS), zero (O), positive small (PS), positive big (PB). The range of control input is divided into nine levels: −4, −3, −2, −1, 0, +1, +2, +3, +4, which is shown in Table 4.2.

(3) **Fuzzy logic rule design**

According to the experience, we can design the fuzzy logic rules as follows:

(1) If e = NB then u = NB
(2) If e = NS then u = NS

Table 4.1 Water level of e

MF degree		Range level						
		−3	−2	−1	0	1	2	3
Fuzzy set	PB	0	0	0	0	0	0.5	1
	PS	0	0	0	0	1	0.5	0
	O	0	0	0.5	1	0.5	0	0
	NS	0	0.5	1	0	0	0	0
	NB	1	0.5	0	0	0	0	0

Table 4.2 Control input u

MF degree		Range level								
		−4	−3	−2	−1	0	1	2	3	4
Fuzzy set	PB	0	0	0	0	0	0	0	0.5	1
	PS	0	0	0	0	0	0.5	1	0.5	0
	O	0	0	0	0.5	1	0.5	0	0	0
	NS	0	0.5	1	0.5	0	0	0	0	0
	NB	1	0.5	0	0	0	0	0	0	0

(3) If e = 0 then u = 0
(4) If e = PS then u = PS
(5) If e = PB then u = PB

Fuzzy rules are given in Table 4.3.

(4) **Fuzzy inference**

From Table 4.3, we can get fuzzy relation matrix as:

$$\boldsymbol{R} = (\mathrm{NB}e \times \mathrm{NB}u) \cup (\mathrm{NS}e \times \mathrm{NS}u) \cup (\mathrm{O}e \times \mathrm{O}u) \cup (\mathrm{PS}e \times \mathrm{PS}u) \cup (\mathrm{PB}e \times \mathrm{PB}) \tag{4.2}$$

The inference can be described as:

$$\mathrm{NB}e \times \mathrm{NB}u = \begin{bmatrix} 1 \\ 0.5 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1.0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Table 4.3 Fuzzy rule

(IF) e	NBe	NSe	Oe	PSe	PBe
(THEN) u	NBu	NSu	Ou	PSu	PBu

$$\mathrm{NS}e \times \mathrm{NS}u = \begin{bmatrix} 0 \\ 0.5 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0.5 & 1 & 0.5 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 1.0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathrm{O}e \times Ou = \begin{bmatrix} 0 \\ 0 \\ 0.5 \\ 1.0 \\ 0.5 \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0 & 0.5 & 1 & 0.5 & 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 1.0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathrm{PS}e \times \mathrm{PS}u = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1.0 \\ 0.5 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0.5 & 1.0 & 0.5 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 1.0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathrm{PB}e \times \mathrm{PB}u = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.5 \\ 1.0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 1.0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 1.0 \end{bmatrix}$$

From above fuzzy matrix, we can get:

$$\boldsymbol{R} = \begin{bmatrix} 1.0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 1.0 & 0.5 & 1.0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0.5 & 1.0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 1.0 \end{bmatrix}$$

(5) **Fuzzy decision making**

$$\boldsymbol{u} = \boldsymbol{e} \circ \boldsymbol{R} \tag{4.3}$$

When $\boldsymbol{e}$ is NB, $\boldsymbol{e} = [1.0 \quad 0.5 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$, the control input is

$$\boldsymbol{u} = \boldsymbol{e} \circ \boldsymbol{R} = [1 \quad 0.5 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0] \circ \begin{bmatrix} 1.0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 1.0 & 0.5 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 1.0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0.5 & 1.0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 1.0 \end{bmatrix} = [1 \quad 0.5 \quad 0.5 \quad 0.5 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$$

Table 4.4 Fuzzy response

E	−3	−2	−1	0	1	2	3
U	−3	−2	−1	0	1	2	3

(6) **Defuzzy**

From above, we have

$$\boldsymbol{u} = \frac{1}{-4} + \frac{0.5}{-3} + \frac{0.5}{-2} + \frac{0.5}{-1} + \frac{0}{0} + \frac{0}{+1} + \frac{0}{+2} + \frac{0}{+3} + \frac{0}{+4}$$

Since $\boldsymbol{u}$ is fuzzy, it must be defuzzified to be used in engineering. According to the defuzzification method "MF degree maximum," we can get $u = -4$. The fuzzy response is given in Table 4.4.

Program for water-level control: chap4_1.m

```
%Fuzzy Control for water tank
clear all;
close all;

a=newfis('fuzz_tank');

a=addvar(a,'input','e',[-3,3]);        %Parameter e
a=addmf(a,'input',1,'NB','zmf',[-3,-1]);
a=addmf(a,'input',1,'NS','trimf',[-3,-1,1]);
a=addmf(a,'input',1,'Z','trimf',[-2,0,2]);
a=addmf(a,'input',1,'PS','trimf',[-1,1,3]);
a=addmf(a,'input',1,'PB','smf',[1,3]);

a=addvar(a,'output','u',[-4,4]);       %Parameter u
a=addmf(a,'output',1,'NB','zmf',[-4,-1]);
a=addmf(a,'output',1,'NS','trimf',[-4,-2,1]);
a=addmf(a,'output',1,'Z','trimf',[-2,0,2]);
a=addmf(a,'output',1,'PS','trimf',[-1,2,4]);
a=addmf(a,'output',1,'PB','smf',[1,4]);

rulelist=[1 1 1 1;      %Edit rule base
      2 2 1 1;
      3 3 1 1;
      4 4 1 1;
      5 5 1 1];
a=addrule(a,rulelist);
```

```
a1=setfis(a,'DefuzzMethod','mom'); %Defuzzy
writefis(a1,'tank');          %Save to fuzzy file "tank.fis"
a2=readfis('tank');

figure(1);
plotfis(a2);
figure(2);
plotmf(a,'input',1);
figure(3);
plotmf(a,'output',1);

flag=0;
if flag==1
   showrule(a)        %Show fuzzy rule base
   ruleview('tank');    %Dynamic Simulation
end
disp('————————————————————————————————————————————————————————————');
disp('    fuzzy controller table:e=[-3,+3],u=[-4,+4]     ');
disp('————————————————————————————————————————————————————————————');

for i=1:1:7
   e(i)=i-4;
   Ulist(i)=evalfis([e(i)],a2);
end
Ulist=round(Ulist)

e=-3;     % Error
u=evalfis([e],a2)  %Using fuzzy inference
```

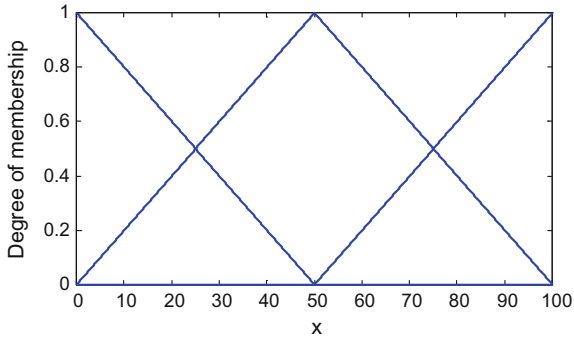
4.3 Fuzzy Logic Control for Washing Machine

For the washing machine, how to set washing time is an important question. Fuzzy logic control is an important method for the washing time setting of the washing machine, which can be described as several steps as follows:

(1) **Fuzzy controller structure**

Consider washing time control of the washing machine, and we can design a fuzzy controller with two inputs and one output. According to our experience, we choose mud and axunge as the inputs and choose washing time as the output.

Fig. 4.3 MF of mud



(2) **Define fuzzy sets**

According to our experience, we can define three fuzzy sets for mud and axunge, respectively, and define five fuzzy sets for washing time.

(3) **Define MF**

Consider MF design of mud, we can define three fuzzy sets: SD (mud small), MD (mud middle), and LD (mud much), the range of mud is in [0, 100], and the MF is designed as follows:

$$\mu_{\text{mud}} = \begin{cases} \mu_{\text{SD}}(x) = (50 - x)/50 & 0 \le x \le 50 \\ \mu_{\text{MD}}(x) = \begin{cases} x/50 & 0 \le x \le 50 \\ (100 - x)/50 & 50 < x \le 100 \end{cases} \\ \mu_{\text{LD}}(x) = (x - 50)/50 & 50 < x \le 100 \end{cases} \tag{4.4}$$

Using triangle MF, the simulation results are shown in Fig. 4.3.

Program of MF of mud: chap4_2.m.m

```
%Define N+1 triangle membership function
clear all;
close all;
N=2;

x=0:0.1:100;
for i=1:N+1
  f(i)=100/N*(i-1);
end

u=trimf(x,[f(1),f(1),f(2)]);
figure(1);
plot(x,u);
```

```
for j=2:N
   u=trimf(x,[f(j-1),f(j),f(j+1)]);
   hold on;
   plot(x,u);
end

u=trimf(x,[f(N),f(N+1),f(N+1)]);
hold on;
plot(x,u);
xlabel('x');
ylabel('Degree of membership');
```

Consider axunge, we can define three fuzzy sets: NG (no axunge), MG (middle axunge), and LG (much axunge), the range value of axunge is set as [0, 100], and the MF is designed as follows:

$$\mu_{\textbf{axunge}} = \begin{cases} \mu_{NG}(y) = (50 - y)/50 & 0 \le y \le 50 \\ \mu_{MG}(y) = \begin{cases} y/50 & 0 \le y \le 50 \\ (100 - y)/50 & 50 < y \le 100 \end{cases} \\ \mu_{LG}(y) = (y - 50)/50 & 50 \le y \le 100 \end{cases} \tag{4.5}$$

MF of axunge is shown in Fig. 4.4, and the program is also chap4_2.m.

For washing time, five fuzzy sets are used: VS (very small), S (small), M (middle), L (ling), and VL (very long), and the value is in the range of [0, 60]. MF of washing time is:

$$\mu_{\text{time}} = \begin{cases} \mu_{VS}(z) = (10 - z)/10 & 0 \le z \le 10 \\ \mu_{S}(z) = \begin{cases} z/10 & 0 \le z \le 10 \\ (25 - z)/15 & 10 < x \le 25 \end{cases} \\ \mu_{M}(z) = \begin{cases} (z - 10)/15 & 10 \le z \le 25 \\ (40 - z)/15 & 25 < z \le 40 \end{cases} \\ \mu_{L}(z) = \begin{cases} (z - 25)/15 & 25 \le z \le 40 \\ (60 - z)/20 & 40 < z \le 60 \end{cases} \\ \mu_{VL}(z) = (z - 40)/20 & 40 \le z \le 60 \end{cases} \tag{4.6}$$

The MF of washing time is shown in Fig. 4.5.

Program of MF of washing time: chap4_3.m

```
%Define N+1 triangle membership function
clear all;
close all;
z=0:0.1:60;
```

Fig. 4.4 MF of axunge

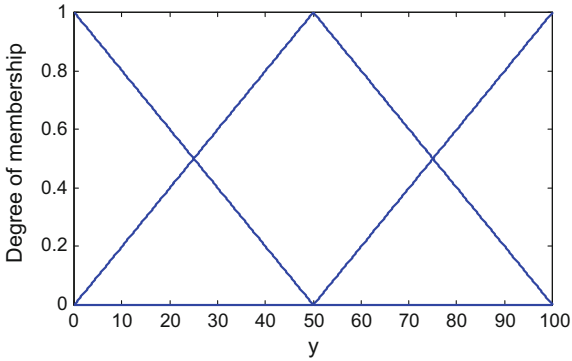
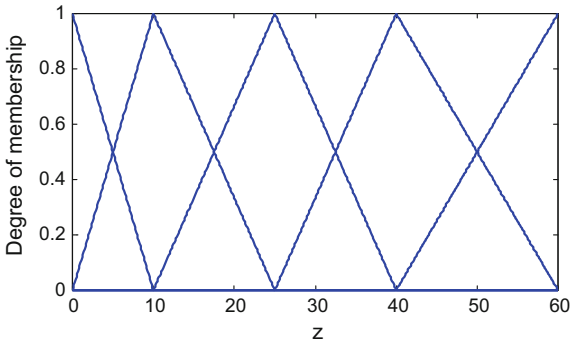


Fig. 4.5 MF of washing time




```
u=trimf(z,[0,0,10]);
figure(1);
plot(z,u);

u=trimf(z,[0,10,25]);
hold on;
plot(z,u);

u=trimf(z,[10,25,40]);
hold on;
plot(z,u);

u=trimf(z,[25,40,60]);
hold on;
plot(z,u);

u=trimf(z,[40,60,60]);
hold on;
```

```
plot(z,u);

xlabel('z');
ylabel('Degree of membership');
```

(4) **Design fuzzy rule**

According to experience, we can design fuzzy rules.

The input is mud and axunge, and the output is washing time. If we design three membership functions for each input, then we can design 9 rules.

The format of the rule is "IF Mud is A AND Axunge is B THEN Washing time is C".

(5) **Design fuzzy rule table**

According to our experience, we can set a fuzzy rule table, which is shown in Table 4.5.

For example, *th fuzzy rule is described as "IF mud is small and axunge is small THEN washing time is very short".

(6) **Fuzzy inference**

① Rule activation

If $x_0(\text{mud}) = 60$, $y_0(\text{axunge}) = 70$, then

$$\mu_{\text{SD}}(60) = 0, \quad \mu_{\text{MD}}(60) = \frac{4}{5}, \quad \mu_{\text{LD}}(60) = \frac{1}{5}$$

$$\mu_{\text{NG}}(70) = 0, \quad \mu_{\text{MG}}(70) = \frac{3}{5}, \quad \mu_{\text{LG}}(70) = \frac{2}{5}$$

Then, four fuzzy rules are activated, and the results are shown in Table 4.6.

② Fuzzy rules inspiration

From Table 4.6, four fuzzy rules are inspired:

Rule 1 IF x is MD and y is MG THEN z is M
Rule 2 IF x is MD and y is LG THEN z is L
Rule 3 IF x is LD and y is MG THEN z is L

Table 4.5 Fuzzy rule of the washing machine

Washing time, z		Mud, x		
		SD	MD	LD
Axunge, y	NG	VS*	M	L
	MG	S	M	L
	LG	M	L	VL

Table 4.6 Fuzzy rule activation

Washing time, z		Mud, x		
		SD	MD (4/5)	LD (1/5)
Axunge, y	NG	0	0	0
	MG (3/5)	0	$\mu_M(z)$	$\mu_L(z)$
	LG (2/5)	0	$\mu_L(z)$	$\mu_{VL}(z)$

Rule 4 IF x is LD and y is LG THEN z is VL

③ Premise inference of each fuzzy rule

(1) Since "AND" is used in the inference, then fuzzy intersection operator can be used and CF for premise of each fuzzy rule can be calculated as follows:

CF of Rule 1 premise: min(4/5, 3/5) = 3/5
CF of Rule 2 premise: min(4/5, 2/5) = 2/5
CF of Rule 3 premise: min(1/5, 3/5) = 1/5
CF of Rule 4 premise: min(1/5, 2/5) = 1/5

The premise CF of each fuzzy rule is shown in Table 4.7.

④ **Inference of each fuzzy rule**

Using fuzzy product operator, the inference of each fuzzy rule is shown in Table 4.8.

⑤ Total output of the system

For different fuzzy rules, fuzzy union operator can be used to calculate the total output as follows:

$$\begin{aligned}\mu_{agg}(z) &= \max\left\{\min\left(\frac{3}{5}, \mu_M(z)\right), \min\left(\frac{1}{5}, \mu_L(z)\right), \min\left(\frac{2}{5}, \mu_L(z)\right), \min\left(\frac{1}{5}, \mu_{VL}(z)\right)\right\} \\ &= \max\left\{\min\left(\frac{3}{5}, \mu_M(z)\right), \min\left(\frac{2}{5}, \mu_L(z)\right), \min\left(\frac{1}{5}, \mu_{VL}(z)\right)\right\}\end{aligned}$$

⑥ **Defuzzy the total output**

$$\mu_M(z) = \frac{z-10}{15} = \frac{3}{5}, \quad \mu_M(z) = \frac{40-z}{15} = \frac{3}{5}$$

Table 4.7 CF of premise of each fuzzy rule

Washing time, z		Mud, x		
		SD	MD (4/5)	LD (1/5)
Axunge, y	NG	0	0	0
	MG (3/5)	0	3/5	1/5
	LG (2/5)	0	2/5	1/5

Table 4.8 Inference of each fuzzy rule

Washing time, z		Mud x		
		SD	MD (4/5)	LD (1/5)
Axunge, y	NG	0	0	0
	MG (3/5)	0	$\min\left(\frac{3}{5}, \mu_M(z)\right)$	$\min\left(\frac{1}{5}, \mu_L(z)\right)$
	LG (2/5)	0	$\min\left(\frac{2}{5}, \mu_L(z)\right)$	$\min\left(\frac{1}{5}, \mu_{VL}(z)\right)$

$$z_1 = 19, \quad z_2 = 31$$

Use the maximum average to defuzzy the output, and we can get precise output as shown in Figs. 4.6 and 4.7.

$$z^* = \frac{z_1 + z_2}{2} = \frac{19 + 31}{2} = 25$$

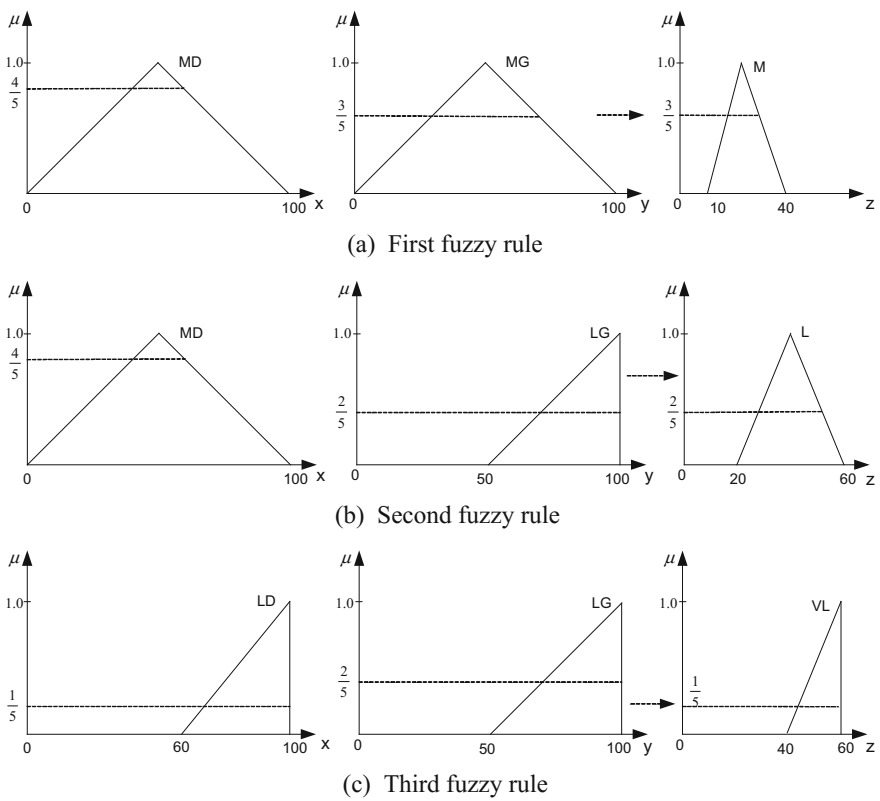


Fig. 4.6 Fuzzy inference of each fuzzy rule

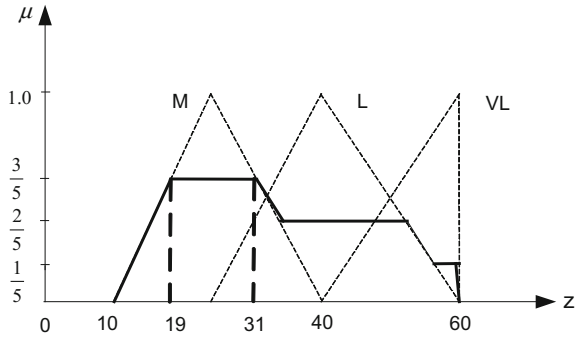


Fig. 4.7 Output and defuzzification of fuzzy inference

(7) **Simulation example**

Choose $x = 60$, $y = 70$, use the above six steps (from step 1 to step 6), and we can get the output as 24.9. The inference process is shown in Fig. 4.8.

Using the order "ruleview", we can watch the dynamic simulation environment.

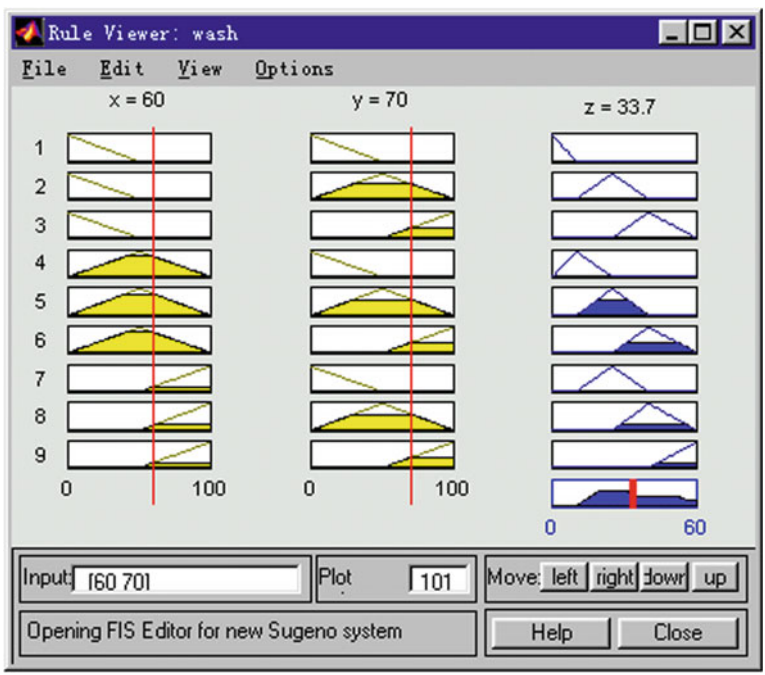


Fig. 4.8 Dynamic simulation

Simulation program of washing time: chap4_4.m

```
%Fuzzy Control for washer
clear all;
close all;

a=newfis('fuzz_wash');

a=addvar(a,'input','x',[0,100]);          %Fuzzy Stain
a=addmf(a,'input',1,'SD','trimf',[0,0,50]);
a=addmf(a,'input',1,'MD','trimf',[0,50,100]);
a=addmf(a,'input',1,'LD','trimf',[50,100,100]);

a=addvar(a,'input','y',[0,100]);          %Fuzzy Axunge
a=addmf(a,'input',2,'NG','trimf',[0,0,50]);
a=addmf(a,'input',2,'MG','trimf',[0,50,100]);
a=addmf(a,'input',2,'LG','trimf',[50,100,100]);

a=addvar(a,'output','z',[0,60]);          %Fuzzy Time
a=addmf(a,'output',1,'VS','trimf',[0,0,10]);
a=addmf(a,'output',1,'S','trimf',[0,10,25]);
a=addmf(a,'output',1,'M','trimf',[10,25,40]);
a=addmf(a,'output',1,'L','trimf',[25,40,60]);
a=addmf(a,'output',1,'VL','trimf',[40,60,60]);

rulelist=[1 1 1 1 1;                %Edit rule base
     1 2 3 1 1;
      1 3 4 1 1;
      2 1 2 1 1;
      2 2 3 1 1;
      2 3 4 1 1;
      3 1 3 1 1;
      3 2 4 1 1;
      3 3 5 1 1];
a=addrule(a,rulelist);
showrule(a)               %Show fuzzy rule base

a1=setfis(a,'DefuzzMethod','mom'); %Defuzzy
writefis(a1,'wash');          %Save to fuzzy file "wash.fis"
a2=readfis('wash');

figure(1);
plotfis(a2);
figure(2);
```

```
plotmf(a,'input',1);
figure(3);
plotmf(a,'input',2);
figure(4);
plotmf(a,'output',1);

ruleview('wash'); %Dynamic Simulation

x=60;
y=70;
z=evalfis([x,y],a2)  %Using fuzzy inference
```

4.4 Fuzzy PI Control

4.4.1 PI Tuning Controller with Fuzzy Logic

Discrete PI algorithm is

$$u(k) = k_p e(k) + k_i T \sum_{j=0}^{k} e(j) \tag{4.7}$$

where T is sampling time and $e(k)$ is error at time k, $k_p > 0$, $k_i > 0$.

The term $e(k)$ is

$$e(k) = y_d(k) - y(k) \tag{4.8}$$

where $y_d(k)$ is the desired value.

In PI controller (4.7), how to tuning k_p and k_i is an important question. To design fuzzy rule, we consider the range of e and ec as $[0, 1]$ and define fuzzy set of e and ec as $\{N,O,P\}$, which represents negative, zero, and positive.

(1) **Fuzzy rules of k_p tuning**

The principle of k_p tuning is: When $e(k)$ is positive, i.e., e is P, we should increase k_p and then Δk_p should be positive; when $e(k)$ is negative, overshoot appears (e is N), we should decrease k_p and then Δk_p should be negative.

When the error is near to zero, i.e., e is Z, we can get three conditions: If ec is N, the overshoot value tends to bigger, Δkp should be negative;if ec is Z, to decrease static error, Δk_p should be positive; if ec is P, the error will tend to be bigger, we should increase k_p, and Δk_p should be positive, which is given in Table 4.9.

Table 4.9 Fuzzy rules of k_p tuning

ec / *e*	N	Z	*P*
N	N	N	*N*
Z	N	P	*P*
P	*P*	*P*	*P*

(2) **Fuzzy rules of k_i tuning**

Using the integrate separation tactics to tune k_i, when the error is very small, we choose big Δk_i, otherwise we choose very small Δk_i, which is given in Table 4.10.

We can tune k_p, k_i online as follows:

$$k_p = k_{p0} + \Delta k_p,\ k_i = k_{i0} + \Delta k_i \tag{4.9}$$

4.4.2 Simulation Example

Consider the following plant

$$G_p(s) = \frac{133}{s^2 + 25s}$$

Choose sampling time as $T = 0.001$, and we can get the discrete plant as:

$$y(k) = -\text{den}(2)y(k-1) - \text{den}(3)y(k-2) + \text{num}(2)u(k-1) + \text{num}(3)u(k-2)$$

The ideal signal is $y_d(k) = 1.0$. Firstly, we run the fuzzy PI tuning program chap4_5.m and we can get the inference system file "fuzzpid.fis". Then, we run the fuzzy control program chap4_6.m. In the program chap4_5.m, according to Tables 4.9 and 4.10, we can get MF of e, ec, k_p, k_i.

We set the range of e, ec, k_p, and k_i according to the ideal signal, the initial error, and our experience.

For the fuzzy system "a", if we use the order "plotmf", we can get the MF of e, ec, k_p, and k_i, which are given from Figs. 4.9, 4.10, 4.11, and 4.12.

Using the command "`showrule(a)`", we can get 9 fuzzy rules as:

1. If (e is N) and (ec is N) then (kp is N)(ki is Z) (1)
2. If (e is N) and (ec is Z) then (kp is N)(ki is Z) (1)
3. If (e is N) and (ec is P) then (kp is N)(ki is Z) (1)
4. If (e is Z) and (ec is N) then (kp is N)(ki is P) (1)
5. If (e is Z) and (ec is Z) then (kp is P)(ki is P) (1)
6. If (e is Z) and (ec is P) then (kp is P)(ki is P) (1)
7. If (e is P) and (ec is N) then (kp is P)(ki is Z) (1)

Table 4.10 Fuzzy rules of k_i tuning

ec / *e*	N	Z	*P*
N	Z	Z	*Z*
Z	P	P	*P*
P	*Z*	*Z*	*Z*

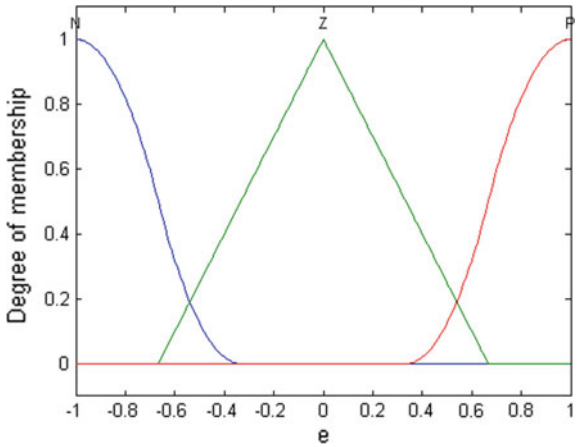


Fig. 4.9 Membership function of error

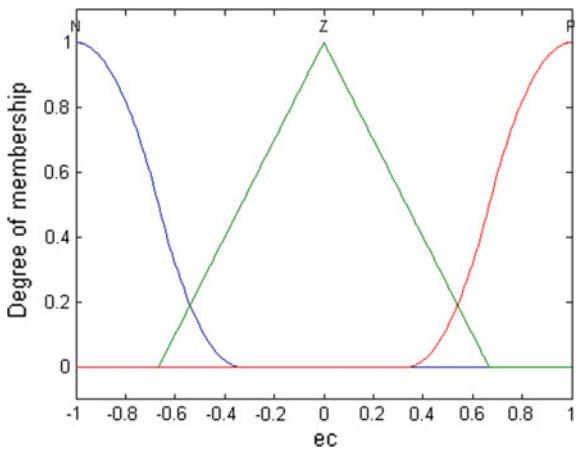


Fig. 4.10 Membership function of error change

8. If (e is P) and (ec is Z) then (kp is P)(ki is Z) (1)
9. If (e is P) and (ec is P) then (kp is P)(ki is Z) (1)

In addition, if we run "fuzzy fuzzpid.fis", we can edit the MF and the rule of the fuzzy system,as shown in Fig. 4.13. If we run "ruleview fuzzpid.fis", we can get the dynamic simulation, as shown in Fig. 4.14.

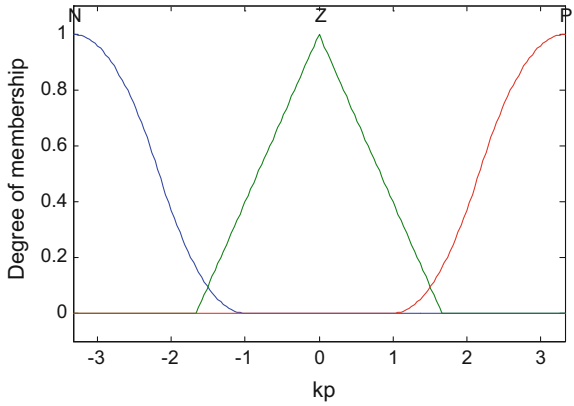


Fig. 4.11 Membership function of k_p

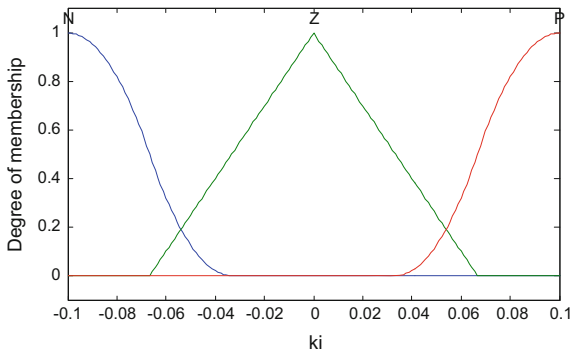


Fig. 4.12 Membership function of k_i

Using the program chap4_6.m, we can realize the tuning of PI controller by using the fuzzy system "fuzzpid.fis".

Choose the initial value of k_p and k_i as zeros, use the fuzzy tuning PI controller, we can get the results, as shown in Figs. 4.15 and 4.16.

Simulation programs:

(1) **Fuzzy logic tuning for PI tuning: chap4_5.m**

```
%Fuzzy Tuning PI Control
clear all;
close all;
a=newfis('fuzzpid');

a=addvar(a,'input','e',[-1,1]);                %Parameter e
a=addmf(a,'input',1,'N','zmf',[-1,-1/3]);
a=addmf(a,'input',1,'Z','trimf',[-2/3,0,2/3]);
```

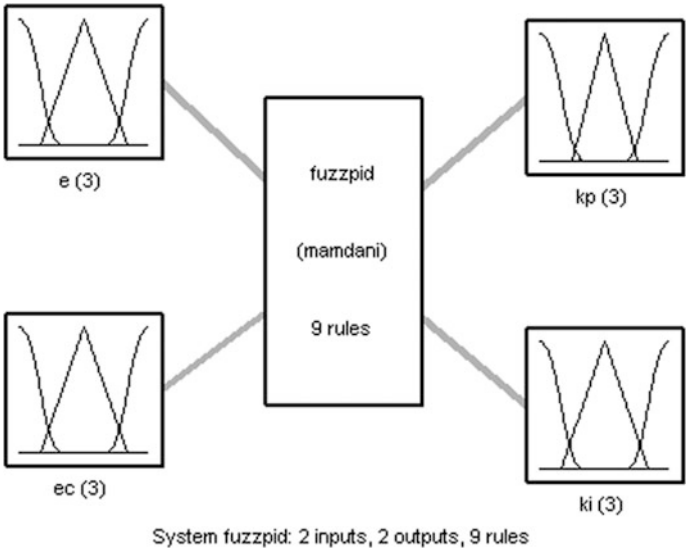


Fig. 4.13 Fuzzy system structure

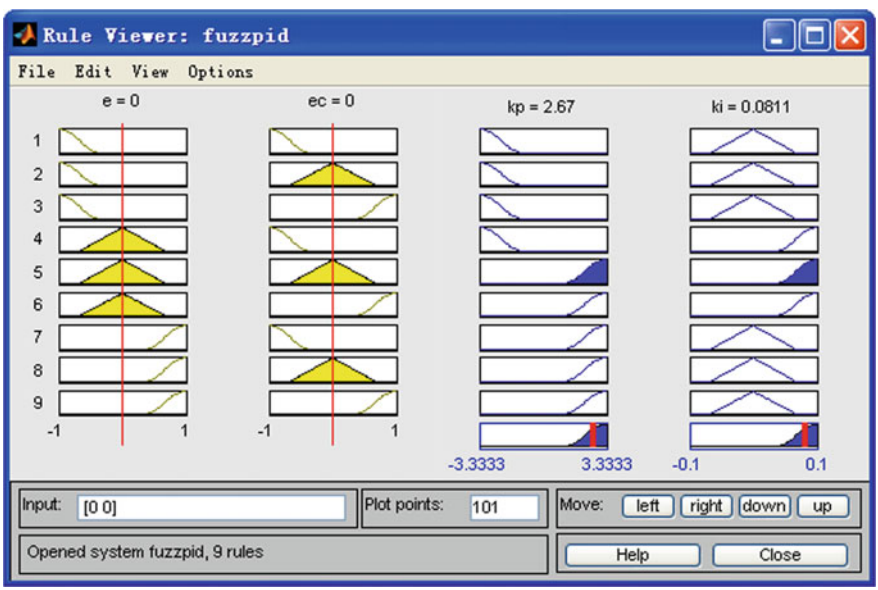


Fig. 4.14 Dynamic simulation environment

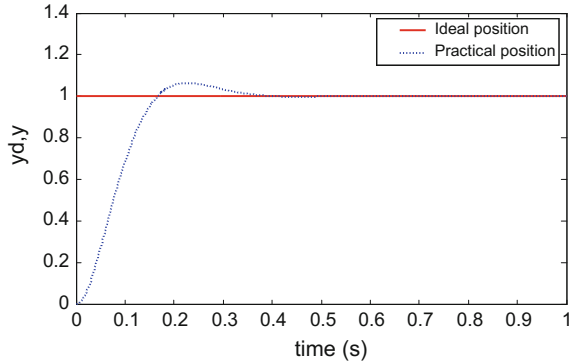


Fig. 4.15 Response with fuzzy PI

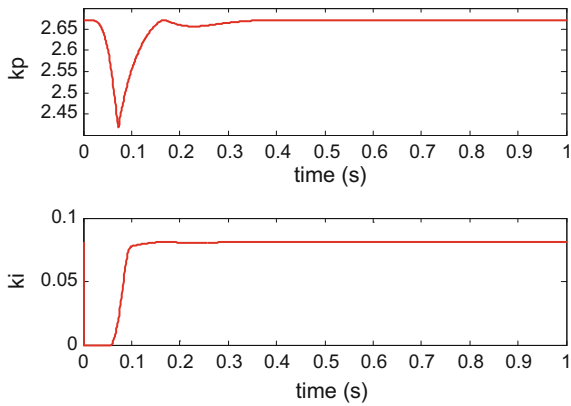


Fig. 4.16 Fuzzy tuning of k_p and k_d

```
a=addmf(a,'input',1,'P','smf',[1/3,1]);

a=addvar(a,'input','ec',[-1,1]);              %Parameter ec
a=addmf(a,'input',2,'N','zmf',[-1,-1/3]);
a=addmf(a,'input',2,'Z','trimf',[-2/3,0,2/3]);
a=addmf(a,'input',2,'P','smf',[1/3,1]);

a=addvar(a,'output','kp',1/3*[-10,10]);          %Parameter kp
a=addmf(a,'output',1,'N','zmf',1/3*[-10,-3]);
a=addmf(a,'output',1,'Z','trimf',1/3*[-5,0,5]);
a=addmf(a,'output',1,'P','smf',1/3*[3,10]);

a=addvar(a,'output','ki',1/30*[-3,3]);           %Parameter ki
a=addmf(a,'output',2,'N','zmf',1/30*[-3,-1]);
a=addmf(a,'output',2,'Z','trimf',1/30*[-2,0,2]);
a=addmf(a,'output',2,'P','smf',1/30*[1,3]);
```

```
rulelist=[1 1 1 2 1 1;
      1 2 1 2 1 1;
      1 3 1 2 1 1;
      2 1 1 3 1 1;
      2 2 3 3 1 1;
      2 3 3 3 1 1;
      3 1 3 2 1 1;
      3 2 3 2 1 1;
      3 3 3 2 1 1];
a=addrule(a,rulelist);
a=setfis(a,'DefuzzMethod','centroid');
writefis(a,'fuzzpid');

a=readfis('fuzzpid');
figure(1);
plotmf(a,'input',1);
figure(2);
plotmf(a,'input',2);
figure(3);
plotmf(a,'output',1);
figure(4);
plotmf(a,'output',2);
figure(5);
plotfis(a);
fuzzy fuzzpid;
showrule(a)
ruleview fuzzpid;
```

(2) **Fuzzy PI control: chap4_6.m**

```
%Fuzzy PI Control
close all;
clear all;
warning off;
a=readfis('fuzzpid');  %Load fuzzpid.fis
ts=0.001;
sys=tf(133,[1,25,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');
u_1=0;u_2=0;
y_1=0;y_2=0;
e_1=0;ec_1=0;ei=0;
```

```
kp0=0;ki0=0;
for k=1:1:1000
time(k)=k*ts;
yd(k)=1;
%Using fuzzy inference to tunning PI
k_pid=evalfis([e_1,ec_1],a);
kp(k)=kp0+k_pid(1);
ki(k)=ki0+k_pid(2);
u(k)=kp(k)*e_1+ki(k)*ei;
y(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
e(k)=yd(k)-y(k);
%%%%%%%%%%%%%%Return of parameters%%%%%%%%%%%%%%%%
u_2=u_1;u_1=u(k);
y_2=y_1;y_1=y(k);
ei=ei+e(k)*ts;   % Calculating I
ec(k)=e(k)-e_1;
e_1=e(k);
ec_1=ec(k);
end
figure(1);
plot(time,yd,'r',time,y,'b:','linewidth',2);
xlabel('time(s)');ylabel('r,y');
legend('Ideal position','Practical position');
figure(2);
subplot(211);
plot(time,kp,'r','linewidth',2);
xlabel('time(s)');ylabel(‘kp');
subplot(212);
plot(time,ki,'r','linewidth',2);
xlabel(‘time(s)');ylabel(‘ki');
figure(3);
plot(time,u,'r','linewidth',2);
xlabel(‘time(s)');ylabel(‘Control input');
```

References

1. L.A. Zadeh, Fuzzy sets. Inf. Control **8**(3), 338–353 (1965)
2. L.A. Zadeh, *Fuzzy Sets, Fuzzy Logic, Fuzzy Systems* (World Scientific Press, 1996)

Chapter 5
Fuzzy T-S Modeling and Control

5.1 Fuzzy T-S Model

The traditional fuzzy model, which belongs to the Mamdani fuzzy model, whose output is fuzzy. The other fuzzy model is Takagi–Sugeno (T-S) fuzzy model, whose output is constant or linear function as follows

$$\begin{aligned} y &= a \\ y &= ax + b \end{aligned} \tag{5.1}$$

The difference between T-S fuzzy model and Mamdani fuzzy model is: (1) the output variable of T-S fuzzy model is constant or linear function; (2) the output of T-S fuzzy model is accurate.

T-S type fuzzy inference system is very suitable for the piecewise linear control system, such as missile control system, aircraft control system.

To design a T-S fuzzy model, for example, we can set the inputs as $X \in [0, 5]$, $Y \in [0, 10]$ and define two fuzzy sets "small" and "large." The output Z can be described as a linear function of the input (x, y) as follows:

$$\begin{aligned} &\text{If } X \text{ is small and } Y \text{ is small then } Z = -\mathrm{x} + \mathrm{y} - 3 \\ &\text{If } X \text{ is small and } Y \text{ is big then } Z = \mathrm{x} + \mathrm{y} + 1 \\ &\text{If } X \text{ is big and } Y \text{ is small then } Z = -2\mathrm{y} + 2 \\ &\text{If } X \text{ is big and } Y \text{ is big then } Z = 2\mathrm{x} + \mathrm{y} - 6 \end{aligned}$$

The input membership function and the input/output of the fuzzy inference system are shown in Figs. 5.1 and 5.2.

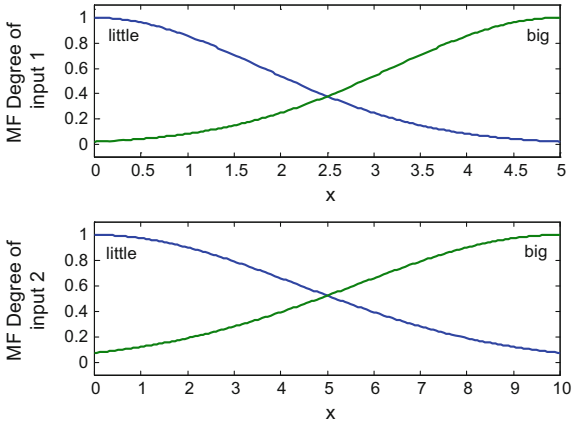


Fig. 5.1 Membership function of T-S fuzzy model inference system

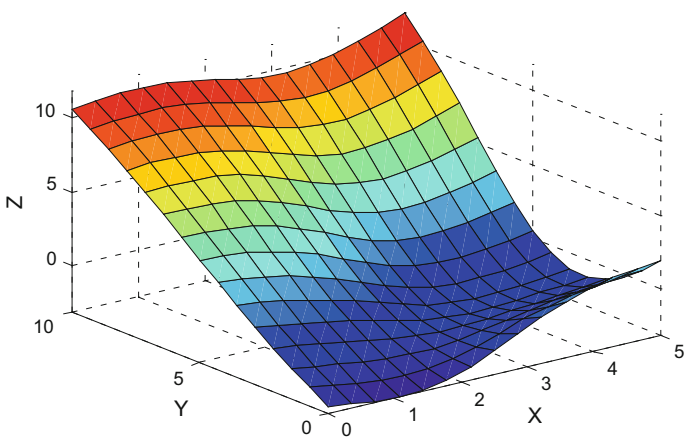


Fig. 5.2 Input and output of T-S type fuzzy inference system

In the simulation program chap5_1.m, by using the command "Showrule (TS2)," fuzzy logic rules can be displayed as the following four fuzzy rules.

(1) If (X is small) and (Y is small) then (Z is first area) (1);
(2) If (X is small) and (Y is big) then (Z is second area) (1);
(3) If (X is big) and (Y is small) then (Z is third area) (1);
(4) If (X is big) and (Y is big) then (Z is fourth area) (1).

Simulation program: chap5_1.m

```
%T-S type fuzzy model
clear all;
close all;
ts2=newfis('ts2','sugeno');
ts2=addvar(ts2,'input','X',[0 5]);
ts2=addmf(ts2,'input',1,'little','gaussmf',[1.8 0]);
ts2=addmf(ts2,'input',1,'big','gaussmf',[1.8 5]);
ts2=addvar(ts2,'input','Y',[0 10]);
ts2=addmf(ts2,'input',2,'little','gaussmf',[4.4 0]);
ts2=addmf(ts2,'input',2,'big','gaussmf',[4.4 10]);
ts2=addvar(ts2,'output','Z',[-3 15]);
ts2=addmf(ts2,'output',1,'first area','linear',[-1 1 -3]);
ts2=addmf(ts2,'output',1,'second area','linear',[1 1 1]);
ts2=addmf(ts2,'output',1,'third area','linear',[0 -2 2]);
ts2=addmf(ts2,'output',1,'fourth area','linear',[2 1 -6]);
rulelist=[1 1 1 1 1;
      1 2 2 1 1;
      2 1 3 1 1;
      2 2 4 1 1];
ts2=addrule(ts2,rulelist);
showrule(ts2);
figure(1);
subplot 211;
plotmf(ts2,'input',1);
xlabel('x'),ylabel('MF Degree of input 1');
subplot 212;
plotmf(ts2,'input',2);
xlabel('x'),ylabel('MF Degree of input 2');
figure(2);
gensurf(ts2);
xlabel('x'),ylabel('y'),zlabel('z');
```

5.2 Fuzzy T-S Modeling and Control Based on LMI

Linear matrix inequality (LMI) is a powerful tool in the field of control domain. Many control theory and synthesis problems can be reduced to the corresponding LMI problem.

It is one of the hot spots in the research of control theory to study the nonlinear system modeling using T-S fuzzy system. It has been proved that the T-S fuzzy model with linear back part can make full use of the local information of the system

and the experience of expert control in the form of fuzzy rules and can approximate any actual plant with arbitrary precision.

The stability condition of T-S fuzzy system can be expressed in the form of linear matrix inequality (LMI). The research on robust stability and adaptive control of T-S fuzzy model is the focus of control theory.

5.2.1 Controller Design of T-S Fuzzy Model Based on LMI

For a continuous nonlinear system with m control inputs and n output states, T-S type fuzzy model can be described as the following fuzzy rules:

$$\text{Fuzzy rule } i \text{ : If } x_1(t) \text{ is } M_1^i \text{ and } x_2(t) \text{ is } M_2^i \text{ and } \cdots x_n(t) \text{ is } M_n^i \tag{5.2}$$

Then $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}_i\boldsymbol{x}(t) + \boldsymbol{B}_i\boldsymbol{u}(t), i = 1, 2, \cdots, r$

where x_j is state, M_j^i is membership function, $\boldsymbol{x}(t)$ is state vector, $\boldsymbol{x}(t) = [x_1(t) \quad \cdots \quad x_n(t)]^{\mathrm{T}} \in R^n$, $\boldsymbol{u}(t)$ is control input vector, $\boldsymbol{u}(t) = [u_1(t) \quad \cdots \quad u_m(t)]^{\mathrm{T}} \in \boldsymbol{R}^m$, $\boldsymbol{A}_i \in \boldsymbol{R}^{n\times n}$, $\boldsymbol{B}_i \in \boldsymbol{R}^{n\times m}$.

According to the defuzzification of fuzzy system, the total output of the fuzzy model is

$$\dot{\boldsymbol{x}}(t) = \frac{\sum_{i=1}^{r} w_i[A_i x(t) + B_i u(t)]}{\sum_{i=1}^{r} w_i} \tag{5.3}$$

where w_i is the membership function for fuzzy rule i, $w_i = \prod_{k=1}^{n} M_k^i(x_k(t))$, Take 4 rules as an example, consider the rule premise as x_1, then we have $k = 1$, $i = 1, 2, 3, 4$, $w_1 = M_1^1(x_1)$, $w_2 = M_1^2(x_1)$, $w_3 = M_1^3(x_1)$, $w_4 = M_1^4(x_1)$.

For each T-S fuzzy rule, the state feedback method can be used to design r kinds of fuzzy control rules as follows

Fuzzy rule i:

$$\text{If } x_1(t) \text{ is } M_1^i \text{ and } x_2(t) \text{ is } M_2^i \text{ and } \cdots x_n(t) \text{ is } M_n^i \tag{5.4}$$

Then $u(t) = \boldsymbol{K}_i\boldsymbol{x}(t), i = 1, 2, \cdots, r$

Parallel distributed compensation (PDC) method is proposed by Sugeno, etc. [1], which is a kind of fuzzy controller design method based on the model, and the proof and analysis are given by [2]. PDC can be applied to solve the control problems for nonlinear system based on T-S fuzzy modeling [3].

Consider the system (5.2) and fuzzy rules (5.4), T-S fuzzy controller can be designed by using PDC method [4] as

$$\boldsymbol{u}(t)=\frac{\sum_{j=1}^{r} w_j \boldsymbol{K}_j \boldsymbol{x}(t)}{\sum_{j=1}^{r} w_j} \tag{5.5}$$

5.2.2 LMI Design and Analysis

Theorem 5.1 [5]: There is a positive definite matrix $\boldsymbol{Q}$. When the following conditions are satisfied, the T-S fuzzy system (5.2) is asymptotically stable.

$$\begin{aligned}
&\boldsymbol{Q}\boldsymbol{A}_i^{\mathrm{T}}+\boldsymbol{A}_i\boldsymbol{Q}+\boldsymbol{V}_i^{\mathrm{T}}\boldsymbol{B}_i^{\mathrm{T}}+\boldsymbol{B}_i\boldsymbol{V}_i<0, \quad i=1,2,\cdots,r\\
&\boldsymbol{Q}\boldsymbol{A}_i^{\mathrm{T}}+\boldsymbol{A}_i\boldsymbol{Q}+\boldsymbol{Q}\boldsymbol{A}_j^{\mathrm{T}}+\boldsymbol{A}_j\boldsymbol{Q}+\boldsymbol{V}_j^{\mathrm{T}}\boldsymbol{B}_i^{\mathrm{T}}+\boldsymbol{B}_i\boldsymbol{V}_j+\boldsymbol{V}_i^{\mathrm{T}}\boldsymbol{B}_j^{\mathrm{T}}+\boldsymbol{B}_j\boldsymbol{V}_i<0, \quad i<j\le r\\
&\boldsymbol{Q}=\boldsymbol{P}^{-1}>0
\end{aligned} \tag{5.6}$$

where $\mathbf{V}_i=\mathbf{K}_i\mathbf{Q}$, that is $\boldsymbol{K}_i=\mathbf{V}_i\mathbf{Q}^{-1}=\mathbf{V}_i\mathbf{P}$, $\mathbf{V}_j=\mathbf{K}_j\mathbf{Q}$, that is $\boldsymbol{K}_j=\boldsymbol{V}_j\boldsymbol{Q}^{-1}=\boldsymbol{V}_j\boldsymbol{P}$.

According to (5.6), K_i in controller (5.5) can be obtained by using the LMI. Theorem 5.1 is given in Ref. [5]. Reference to [5], the concrete proof of theorem 5.1 is given below.

Proof Choose Lyapunov function as

$$\boldsymbol{V}(t)=\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{x}$$

where $\boldsymbol{P}$ is positive and symmetric Matrix.

then

$$\begin{aligned}
\dot{\boldsymbol{V}}(t)&=\frac{1}{2}\left(\dot{\boldsymbol{x}}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{x}+\boldsymbol{x}^{\mathrm{T}}\boldsymbol{P}\dot{\boldsymbol{x}}\right)=\frac{1}{2}\dot{\boldsymbol{x}}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{x}+\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{P}\dot{\boldsymbol{x}}\\
&=\frac{1}{2}\left\{\frac{\sum_{i=1}^{r} w_i[\boldsymbol{A}_i\boldsymbol{x}+\boldsymbol{B}_i\boldsymbol{u}]}{\sum_{i=1}^{r} w_i}\right\}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{x}+\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{P}\left\{\frac{\sum_{i=1}^{r} w_i[\boldsymbol{A}_i\boldsymbol{x}+\boldsymbol{B}_i\boldsymbol{u}]}{\sum_{i=1}^{r} w_i}\right\}
\end{aligned}$$

Submitting controller (5.5) into above, we have

$$\dot{\boldsymbol{V}}(t)=\frac{1}{2}\left\{\frac{\sum_{i=1}^{r}w_i\left[\boldsymbol{A}_i\boldsymbol{x}+\boldsymbol{B}_i\frac{\sum_{j=1}^{r}w_j\boldsymbol{K}_j\boldsymbol{x}}{\sum_{j=1}^{r}w_j}\right]}{\sum_{i=1}^{r}w_i}\right\}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{x}+\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{P}\left\{\frac{\sum_{i=1}^{r}w_i\left[\boldsymbol{A}_i\boldsymbol{x}+\boldsymbol{B}_i\frac{\sum_{j=1}^{r}w_j\boldsymbol{K}_j\boldsymbol{x}}{\sum_{j=1}^{r}w_j}\right]}{\sum_{i=1}^{r}w_i}\right\}$$

$$=\frac{1}{2}\left\{\frac{\sum_{i=1}^{r}w_i\left[\sum_{j=1}^{r}w_j\boldsymbol{A}_i\boldsymbol{x}+\boldsymbol{B}_i\sum_{j=1}^{r}w_j\boldsymbol{K}_j\boldsymbol{x}\right]}{\sum_{i=1}^{r}w_i\sum_{j=1}^{r}w_j}\right\}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{x}+\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{P}\left\{\frac{\sum_{i=1}^{r}w_i\left[\sum_{j=1}^{r}w_j\boldsymbol{A}_i\boldsymbol{x}+\boldsymbol{B}_i\sum_{j=1}^{r}w_j\boldsymbol{K}_j\boldsymbol{x}\right]}{\sum_{i=1}^{r}w_i\sum_{j=1}^{r}w_j}\right\}$$

$$=\frac{1}{2}\left[\frac{\sum_{i=1}^{r}w_i\sum_{j=1}^{r}w_j\left(\boldsymbol{A}_i\boldsymbol{x}+\boldsymbol{B}_i\boldsymbol{K}_j\boldsymbol{x}\right)}{\sum_{i=1}^{r}\sum_{j=1}^{r}w_iw_j}\right]^{\mathrm{T}}\boldsymbol{P}\boldsymbol{x}+\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{P}\left[\frac{\sum_{i=1}^{r}w_i\sum_{j=1}^{r}w_j\left(\boldsymbol{A}_i\boldsymbol{x}+\boldsymbol{B}_i\boldsymbol{K}_j\boldsymbol{x}\right)}{\sum_{i=1}^{r}\sum_{j=1}^{r}w_iw_j}\right]$$

$$=\frac{1}{2}\frac{\sum_{i=1}^{r}\sum_{j=1}^{r}w_iw_j\boldsymbol{x}^{\mathrm{T}}\left(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_j\right)^{\mathrm{T}}}{\sum_{i=1}^{r}\sum_{j=1}^{r}w_iw_j}\boldsymbol{P}\boldsymbol{x}+\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{P}\frac{\sum_{i=1}^{r}\sum_{j=1}^{r}w_iw_j\left(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_j\right)\boldsymbol{x}}{\sum_{i=1}^{r}\sum_{j=1}^{r}w_iw_j}$$

$$=\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\frac{\sum_{i=1}^{r}\sum_{j=1}^{r}w_iw_j\left(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_j\right)^{\mathrm{T}}}{\sum_{i=1}^{r}\sum_{j=1}^{r}w_iw_j}\boldsymbol{P}\boldsymbol{x}+\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{P}\frac{\sum_{i=1}^{r}\sum_{j=1}^{r}w_iw_j\left(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_j\right)}{\sum_{i=1}^{r}\sum_{j=1}^{r}w_iw_j}\boldsymbol{x}$$

$$=\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\left\{\frac{\sum_{i=1}^{r}\sum_{j=1}^{r}w_iw_j\left[\left(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_j\right)^{\mathrm{T}}\boldsymbol{P}+\boldsymbol{P}\left(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_j\right)\right]}{\sum_{i=1}^{r}\sum_{j=1}^{r}w_iw_j}\right\}\boldsymbol{x}$$

Therefore, when the following inequality is satisfied as

$$\left(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_j\right)^{\mathrm{T}}\boldsymbol{P}+\boldsymbol{P}\left(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_j\right)<0$$

We have $\dot{\boldsymbol{V}}(t)<=0$, where $i=1,2,\cdots,r$, $j=1,2,\cdots,r$.
Consider $i=j$ and $i\neq j$, respectively, we have

$$\dot{\boldsymbol{V}}(t)=\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\left\{\frac{\sum\limits_{i=1}^{r}\sum\limits_{j=1}^{r}w_iw_j\left[\left(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_j\right)^{\mathrm{T}}\boldsymbol{P}+\boldsymbol{P}\left(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_j\right)\right]}{\sum\limits_{i=1}^{r}\sum\limits_{j=1}^{r}w_iw_j}\right\}\boldsymbol{x}$$

$$=\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\frac{1}{\sum\limits_{i=1}^{r}\sum\limits_{j=1}^{r}w_iw_j}\sum_{i=1}^{r}w_iw_i\left[\left(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_i\right)^{\mathrm{T}}\boldsymbol{P}+\boldsymbol{P}(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_i)\right]\boldsymbol{x}$$

$$+\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\frac{1}{\sum\limits_{i=1}^{r}\sum\limits_{j=1}^{r}w_iw_j}\sum_{i<j}^{r}w_iw_j\left[\boldsymbol{G}_{ij}^{\mathrm{T}}\boldsymbol{P}+\boldsymbol{P}\boldsymbol{G}_{ij}\right]\boldsymbol{x}$$

where $\boldsymbol{G}_{ij}=\left(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_j\right)+\left(\boldsymbol{A}_j+\boldsymbol{B}_j\boldsymbol{K}_i\right)$.

If the following inequality is satisfied

$$\begin{cases}(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_i)^{\mathrm{T}}\boldsymbol{P}+\boldsymbol{P}(\boldsymbol{A}_i<\boldsymbol{B}_i\boldsymbol{K}_i)<0 & i=j=1,2,\ldots,r\\ \boldsymbol{G}_{ij}^{\mathrm{T}}\boldsymbol{P}+\boldsymbol{P}\boldsymbol{G}_{ij}0 & i<j\le r\end{cases}\tag{5.7}$$

Then we have $\dot{\boldsymbol{V}}(t)\le 0$.

From (5.7), if $\dot{V}\equiv 0$, we have $\boldsymbol{x}\equiv 0$, according to LaSalle invariance principle, when $t\to\infty$, $\boldsymbol{x}\to 0$.

5.2.3 *Transformation of LMI*

Firstly, consider $(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_i)^{\mathrm{T}}\boldsymbol{P}+\boldsymbol{P}(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_i)<0,\quad i=j=1,2,\ldots,r,$ define $\boldsymbol{Q}=\boldsymbol{P}^{-1}$, then $\boldsymbol{Q}$ is also positive and symmetric matrix, let $\boldsymbol{V}_i=\boldsymbol{K}_i\boldsymbol{Q}$, then

$$\boldsymbol{A}_i^{\mathrm{T}}\boldsymbol{P}+\boldsymbol{K}_i^{\mathrm{T}}\boldsymbol{B}_i^{\mathrm{T}}\boldsymbol{P}+\boldsymbol{P}\boldsymbol{A}_i+\boldsymbol{P}\boldsymbol{B}_i\boldsymbol{K}_i<0$$

Multiplied $\boldsymbol{P}^{-1}$ by both sides of above inequality, we have

$$\boldsymbol{P}^{-1}\boldsymbol{A}_i^{\mathrm{T}}+\boldsymbol{P}^{-1}\boldsymbol{K}_i^{\mathrm{T}}\boldsymbol{B}_i^{\mathrm{T}}+\boldsymbol{A}_i\boldsymbol{P}^{-1}+\boldsymbol{B}_i\boldsymbol{K}_i\boldsymbol{P}^{-1}<0$$

That is

$$\boldsymbol{Q}\boldsymbol{A}_i^{\mathrm{T}}+\boldsymbol{V}_i^{\mathrm{T}}\boldsymbol{B}_i^{\mathrm{T}}+\boldsymbol{A}_i\boldsymbol{Q}+\boldsymbol{B}_i\boldsymbol{V}_i<0$$

That is

$$\boldsymbol{Q}\boldsymbol{A}_i^{\mathrm{T}}+\boldsymbol{A}_i\boldsymbol{Q}+\boldsymbol{V}_i^{\mathrm{T}}\boldsymbol{B}_i^{\mathrm{T}}+\boldsymbol{B}_i\boldsymbol{V}_i<0 \tag{5.8}$$

Consider $\boldsymbol{G}_{ij}^{\mathrm{T}}\boldsymbol{P}+\boldsymbol{P}\boldsymbol{G}_{ij}<0$, $\boldsymbol{G}_{ij}=\left(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_j\right)+\left(\boldsymbol{A}_j+\boldsymbol{B}_j\boldsymbol{K}_i\right)$, $i<j\leq r$. Let $\boldsymbol{Q}=\boldsymbol{P}^{-1}$ and define $\boldsymbol{V}_i=\boldsymbol{K}_i\boldsymbol{Q}$, $\boldsymbol{V}_j=\boldsymbol{K}_j\boldsymbol{Q}$, then

$$\left(\left(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_j\right)+\left(\boldsymbol{A}_j+\boldsymbol{B}_j\boldsymbol{K}_i\right)\right)^{\mathrm{T}}\boldsymbol{P}+\boldsymbol{P}\left(\left(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_j\right)+\left(\boldsymbol{A}_j+\boldsymbol{B}_j\boldsymbol{K}_i\right)\right)<0$$

Multiplied $\boldsymbol{P}^{-1}$ by both sides of above inequality, consider $\boldsymbol{Q}=\boldsymbol{Q}^{\mathrm{T}}$, we have

$$\boldsymbol{Q}^{\mathrm{T}}\left(\left(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_j\right)+\left(\boldsymbol{A}_j+\boldsymbol{B}_j\boldsymbol{K}_i\right)\right)^{\mathrm{T}}+\left(\left(\boldsymbol{A}_i+\boldsymbol{B}_i\boldsymbol{K}_j\right)+\left(\boldsymbol{A}_j+\boldsymbol{B}_j\boldsymbol{K}_i\right)\right)\boldsymbol{Q}<0$$

That is

$$\left(\boldsymbol{A}_i\boldsymbol{Q}+\boldsymbol{B}_i\boldsymbol{K}_j\boldsymbol{Q}+\boldsymbol{A}_j\boldsymbol{Q}+\boldsymbol{B}_j\boldsymbol{K}_i\boldsymbol{Q}\right)^{\mathrm{T}}+\boldsymbol{A}_i\boldsymbol{Q}+\boldsymbol{B}_i\boldsymbol{K}_j\boldsymbol{Q}+\boldsymbol{A}_j\boldsymbol{Q}+\boldsymbol{B}_j\boldsymbol{K}_i\boldsymbol{Q}<0$$

then

$$\left(\boldsymbol{A}_i\boldsymbol{Q}+\boldsymbol{B}_i\boldsymbol{V}_j+\boldsymbol{A}_j\boldsymbol{Q}+\boldsymbol{B}_j\boldsymbol{V}_i\right)^{\mathrm{T}}+\boldsymbol{A}_i\boldsymbol{Q}+\boldsymbol{B}_i\boldsymbol{V}_j+\boldsymbol{A}_j\boldsymbol{Q}+\boldsymbol{B}_j\boldsymbol{V}_i<0$$

That is

$$\boldsymbol{Q}\boldsymbol{A}_i^{\mathrm{T}}+\boldsymbol{A}_i\boldsymbol{Q}+\boldsymbol{Q}\boldsymbol{A}_j^{\mathrm{T}}+\boldsymbol{A}_j\boldsymbol{Q}+\boldsymbol{V}_j^{\mathrm{T}}\boldsymbol{B}_i^{\mathrm{T}}+\boldsymbol{B}_i\boldsymbol{V}_j+\boldsymbol{V}_i^{\mathrm{T}}\boldsymbol{B}_j^{\mathrm{T}}+\boldsymbol{B}_j\boldsymbol{V}_i<0 \tag{5.9}$$

5.2.4 LMI Design Example

First example: consider two fuzzy rules, $r=2$, $i=1,2$, then we can get two LMI as follows

$$\begin{aligned}&\boldsymbol{Q}\boldsymbol{A}_1^{\mathrm{T}}+\boldsymbol{A}_1\boldsymbol{Q}+\boldsymbol{V}_1^{\mathrm{T}}\boldsymbol{B}_1^{\mathrm{T}}+\boldsymbol{B}_1\boldsymbol{V}_1<0\\&\boldsymbol{Q}\boldsymbol{A}_2^{\mathrm{T}}+\boldsymbol{A}_2\boldsymbol{Q}+\boldsymbol{V}_2^{\mathrm{T}}\boldsymbol{B}_2^{\mathrm{T}}+\boldsymbol{B}_2\boldsymbol{V}_2<0\end{aligned} \tag{5.10}$$

For $i<j\leq r$, **we have** $i=1$, $j=2$, **two fuzzy rule are interacted, from (5.9), then we have** an LMI as

$$\boldsymbol{Q}\boldsymbol{A}_1^{\mathrm{T}}+\boldsymbol{A}_1\boldsymbol{Q}+\boldsymbol{Q}\boldsymbol{A}_2^{\mathrm{T}}+\boldsymbol{A}_2\boldsymbol{Q}+\boldsymbol{V}_2^{\mathrm{T}}\boldsymbol{B}_1^{\mathrm{T}}+\boldsymbol{B}_1\boldsymbol{V}_2+\boldsymbol{V}_1^{\mathrm{T}}\boldsymbol{B}_2^{\mathrm{T}}+\boldsymbol{B}_2\boldsymbol{V}_1<0 \tag{5.11}$$

For the first example, Matlab program can be written as

$$\begin{aligned}
&\mathrm{L1} = \mathrm{Q} * \mathrm{A1}' + \mathrm{A1} * \mathrm{Q} + \mathrm{V1}' * \mathrm{B1}' + \mathrm{B1} * \mathrm{V1};\\
&\mathrm{L2} = \mathrm{Q} * \mathrm{A2}' + \mathrm{A2} * \mathrm{Q} + \mathrm{V2}' * \mathrm{B2}' + \mathrm{B2} * \mathrm{V2};\\
&\mathrm{L3} = \mathrm{Q} * \mathrm{A1}' + \mathrm{A1} * \mathrm{Q} + \mathrm{Q} * \mathrm{A2}' + \mathrm{A2} * \mathrm{Q} + \mathrm{V2}' * \mathrm{B1}' + \mathrm{B1} * \mathrm{V2} + \mathrm{V1}' * \mathrm{B2}' + \mathrm{B2} * \mathrm{V1};\\
&\mathrm{F} = \mathrm{set}(\mathrm{L1} < 0) + \mathrm{set}(\mathrm{L2} < 0) + \mathrm{set}(\mathrm{L3} < 0) + \mathrm{set}(\mathrm{Q} > 0);
\end{aligned}$$

Second example: consider four fuzzy rules, $r = 4$, **the we have four LMI as follows**

$$\begin{aligned}
&\boldsymbol{Q}\boldsymbol{A}_1^{\mathrm{T}} + \boldsymbol{A}_1\boldsymbol{Q} + \boldsymbol{V}_1^{\mathrm{T}}\boldsymbol{B}_1^{\mathrm{T}} + \boldsymbol{B}_1\boldsymbol{V}_1 < 0\\
&\boldsymbol{Q}\boldsymbol{A}_2^{\mathrm{T}} + \boldsymbol{A}_2\boldsymbol{Q} + \boldsymbol{V}_2^{\mathrm{T}}\boldsymbol{B}_2^{\mathrm{T}} + \boldsymbol{B}_2\boldsymbol{V}_2 < 0\\
&\boldsymbol{Q}\boldsymbol{A}_3^{\mathrm{T}} + \boldsymbol{A}_3\boldsymbol{Q} + \boldsymbol{V}_3^{\mathrm{T}}\boldsymbol{B}_3^{\mathrm{T}} + \boldsymbol{B}_3\boldsymbol{V}_3 < 0\\
&\boldsymbol{Q}\boldsymbol{A}_4^{\mathrm{T}} + \boldsymbol{A}_4\boldsymbol{Q} + \boldsymbol{V}_4^{\mathrm{T}}\boldsymbol{B}_4^{\mathrm{T}} + \boldsymbol{B}_4\boldsymbol{V}_4 < 0
\end{aligned} \tag{5.12}$$

For the second example, Matlab program can be written as

$$\begin{aligned}
&\mathrm{L1} = \mathrm{Q} * \mathrm{A1}' + \mathrm{A1} * \mathrm{Q} + \mathrm{V1}' * \mathrm{B1}' + \mathrm{B1} * \mathrm{V1};\\
&\mathrm{L2} = \mathrm{Q} * \mathrm{A2}' + \mathrm{A2} * \mathrm{Q} + \mathrm{V2}' * \mathrm{B2}' + \mathrm{B2} * \mathrm{V2};\\
&\mathrm{L3} = \mathrm{Q} * \mathrm{A3}' + \mathrm{A3} * \mathrm{Q} + \mathrm{V3}' * \mathrm{B3}' + \mathrm{B3} * \mathrm{V3};\\
&\mathrm{L4} = \mathrm{Q} * \mathrm{A4}' + \mathrm{A4} * \mathrm{Q} + \mathrm{V4}' * \mathrm{B4}' + \mathrm{B4} * \mathrm{V4};
\end{aligned}$$

For $i < j \le r$, from $\boldsymbol{Q}\boldsymbol{A}_i^{\mathrm{T}} + \boldsymbol{A}_i\boldsymbol{Q} + \boldsymbol{Q}\boldsymbol{A}_j^{\mathrm{T}} + \boldsymbol{A}_j\boldsymbol{Q} + \boldsymbol{V}_j^{\mathrm{T}}\boldsymbol{B}_i^{\mathrm{T}} + \boldsymbol{B}_i\boldsymbol{V}_j + \boldsymbol{V}_i^{\mathrm{T}}\boldsymbol{B}_j^{\mathrm{T}} + \boldsymbol{B}_j\boldsymbol{V}_i < 0$, we can design six LMI as follows:

$$i = 1, j = 2;\ i = 1, j = 3;\ i = 1, j = 4;\ i = 2, j = 3;\ i = 2, j = 4;\ i = 3, j = 4.$$

In the design of LMI, the interaction between membership function i and membership function j should be considered.

Consider $i = 3$, $j = 4$, the interaction between the third rule and the fourth rule are considered, we can get a LMI as

$$\boldsymbol{Q}\boldsymbol{A}_3^{\mathrm{T}} + \boldsymbol{A}_3\boldsymbol{Q} + \boldsymbol{Q}\boldsymbol{A}_4^{\mathrm{T}} + \boldsymbol{A}_4\boldsymbol{Q} + \boldsymbol{V}_4^{\mathrm{T}}\boldsymbol{B}_3^{\mathrm{T}} + \boldsymbol{B}_3\boldsymbol{V}_4 + \boldsymbol{V}_3^{\mathrm{T}}\boldsymbol{B}_4^{\mathrm{T}} + \boldsymbol{B}_4\boldsymbol{V}_3 < 0 \tag{5.13}$$

The Matlab program is

$$\begin{aligned}
\mathrm{L} = {} & \mathrm{Q} * \mathrm{A3}' + \mathrm{A3} * \mathrm{Q} + \mathrm{Q} * \mathrm{A4}' + \mathrm{A4} * \mathrm{Q} + \mathrm{V4}' * \mathrm{B3}' + \mathrm{B3} * \mathrm{V4} + \mathrm{V3}'\\
& * \mathrm{B4}' + \mathrm{B4} * \mathrm{V3};
\end{aligned}$$

5.3 Fuzzy T-S Modeling and Control Based on LMI for Inverted Pendulum

5.3.1 System Description

The single inverted pendulum system is a complex nonlinear and uncertain system. The control problem of the inverted pendulum system is a typical problem; the aim is to keep the cart in a predetermined position by applying a control input, and at the same time, the pendulum is kept in the range of a predefined vertical deviation angle.

The single inverted pendulum model is

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{\mathrm{g}\sin x_1 - amlx_2^2 \sin(2x_1)/2 - au\cos x_1}{4l/3 - aml\cos^2 x_1} \end{aligned} \tag{5.14}$$

where x_1 is angle of pendulum, x_2 is angle speed of pendulum, $2l$ is Length of pendulum, u is control input, $a = \frac{1}{M+m}$, M and m are mass of car and pendulum.

5.3.2 Simulation Based on Two Fuzzy Rules Design

When $x_1 \to 0$, $\sin x_1 \to x_1$, $\cos x_1 \to 1$; when $x_1 \to \pm\frac{\pi}{2}$, $\sin x_1 \to \pm 1 \to \frac{2}{\pi}x_1$, from (5.14), we have two T-S type fuzzy rules

Rule 1 IF $x_1(t)$ is about 0, THEN $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}_1\boldsymbol{x}(t) + \boldsymbol{B}_1 u(t)$;
Rule 2 IF $x_1(t)$ is about $\pm\frac{\pi}{2}\left(|x_1| < \frac{\pi}{2}\right)$, then $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}_2\boldsymbol{x}(t) + \boldsymbol{B}_2 u(t)$.

where $\boldsymbol{A}_1 = \begin{bmatrix} 0 & 1 \\ \frac{g}{4l/3-aml} & 0 \end{bmatrix}$, $\boldsymbol{B}_1 = \begin{bmatrix} 0 \\ -\frac{\alpha}{4l/3-aml} \end{bmatrix}$, $\boldsymbol{A}_2 = \begin{bmatrix} 0 & 1 \\ \frac{2\mathrm{g}}{\pi(4l/3-aml\beta^2)} & 0 \end{bmatrix}$, $\boldsymbol{B}_2 = \begin{bmatrix} 0 \\ -\frac{\alpha\beta}{4l/3-aml\beta^2} \end{bmatrix}$, $\beta = \cos(88^\circ)$.

According to theorem 5.1, using (5.10) and (5.11), LMI of the inverted pendulum can be expressed as

$$\begin{aligned} &\boldsymbol{Q}\boldsymbol{A}_1^{\mathrm{T}} + \boldsymbol{A}_1\boldsymbol{Q} + \boldsymbol{V}_1^{\mathrm{T}}\boldsymbol{B}_1^{\mathrm{T}} + \boldsymbol{B}_1\boldsymbol{V}_1 < 0, \\ &\boldsymbol{Q}\boldsymbol{A}_2^{\mathrm{T}} + \boldsymbol{A}_2\boldsymbol{Q} + \boldsymbol{V}_2^{\mathrm{T}}\boldsymbol{B}_2^{\mathrm{T}} + \boldsymbol{B}_2\boldsymbol{V}_2 < 0, \\ &\boldsymbol{Q}\boldsymbol{A}_1^{\mathrm{T}} + \boldsymbol{A}_1\boldsymbol{Q} + \boldsymbol{Q}\boldsymbol{A}_2^{\mathrm{T}} + \boldsymbol{A}_2\boldsymbol{Q} + \boldsymbol{V}_2^{\mathrm{T}}\boldsymbol{B}_1^{\mathrm{T}} + \boldsymbol{B}_1\boldsymbol{V}_2 + \boldsymbol{V}_1^{\mathrm{T}}\boldsymbol{B}_2^{\mathrm{T}} + \boldsymbol{B}_2\boldsymbol{V}_1 < 0 \\ &\boldsymbol{Q} = \boldsymbol{P}^{-1} > 0 \end{aligned} \tag{5.15}$$

where $\boldsymbol{K}_1 = \boldsymbol{V}_1\boldsymbol{P}$, $\boldsymbol{K}_2 = \boldsymbol{V}_2\boldsymbol{P}$, $i = 1, 2$.

From above LMI, the programs are designed as follows:

```
L1=Q*A1'+A1*Q+V1'*B1'+B1*V1;
L2=Q*A2'+A2*Q+V2'*B2'+B2*V2;
L3=Q*A1'+A1*Q+Q*A2'+A2*Q+V2'*B1'+B1*V2+V1'*B2'+B2*V1;
```

Then we can set LMI as follows and get $\boldsymbol{K}_i$ by (5.15)

```
F=set(L1<0)+set(L2<0)+set(L3<0)+set(Q>0);
```

In the simulation, we choose $g = 9.8\,\text{m/s}^2$, $m = 2.0\,\text{kg}$, $M = 8.0\,\text{kg}$, $2l = 1.0\,\text{m}$. According to the experience of inverted pendulum, two fuzzy rules can be designed:

$$\begin{aligned}&\text{Rule 1}: \text{If } x_1(t) \text{ is about } 0 \text{ then } u = \mathbf{K}_1\mathbf{x}(t)\\&\text{Rule 2}: \text{If } x_1(t) \text{ is about } \pm\tfrac{\pi}{2}\left(|x_1(t)| < \tfrac{\pi}{2}\right) \text{ then } u = \mathbf{K}_2\mathbf{x}(t)\end{aligned}$$

According to (5.5), the T-S based fuzzy controller is designed by using PDC method:

$$u = w_1(x_1)\boldsymbol{K}_1\boldsymbol{x}(t) + w_2(x_1)\boldsymbol{K}_2\boldsymbol{x}(t) \tag{5.16}$$

where $w_1 + w_2 = 1$.

According to two fuzzy rules of the inverted pendulum, the membership function should be designed according to Fig. 5.3. Triangular membership function is used to fuzzify $x_1(t)$. The initial states are chosen as $\left[\frac{\pi}{3} \quad 0\right]$.

Using LMI toolbox, YALMIP toolbox, we can get $\boldsymbol{Q}, \boldsymbol{V}_1, \boldsymbol{V}_2$ from the program chap5_2LMI.m, we can get $\boldsymbol{K}_1 = [2400.8 \quad 692.3]$, $\boldsymbol{K}_2 = [5171.6 \quad 1515.3]$. The main program is chap5_2sim.mdl; simulation results are shown in Figs. 5.4, 5.5, and 5.6.

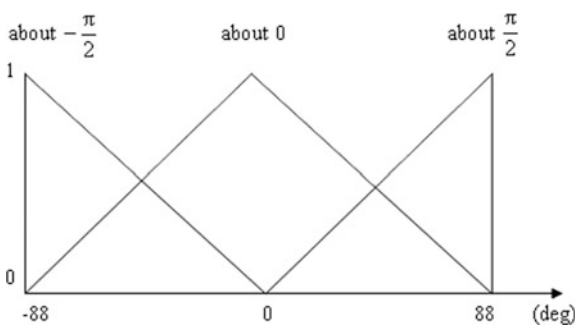


Fig. 5.3 Schematic diagram of fuzzy membership function

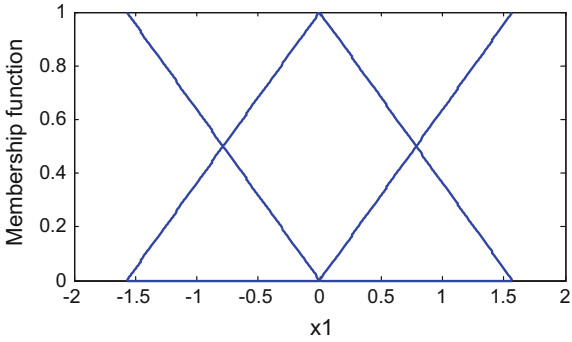


Fig. 5.4 Fuzzy membership function in simulation

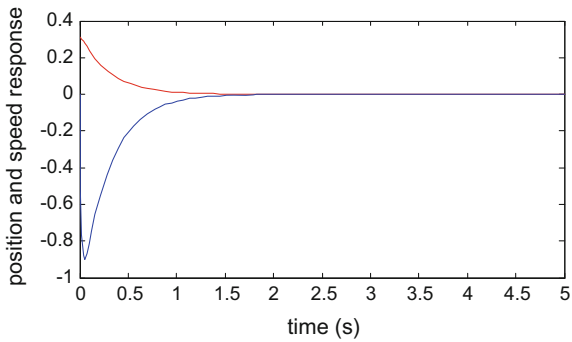


Fig. 5.5 Angle and angle speed response

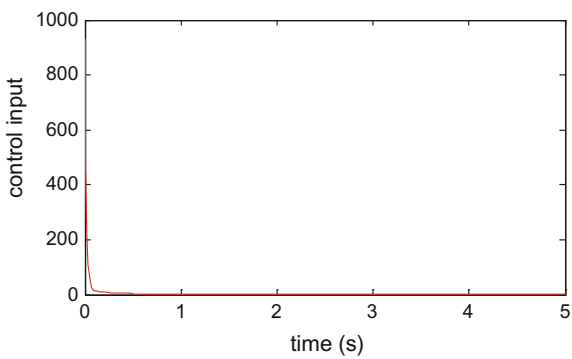


Fig. 5.6 Control input

Matlab Programs:

(1) Controller gain based on LMI: **chap5_2LMI.m;**

```
clearall;
closeall;
g=9.8;m=2.0;M=8.0;l=0.5;
```

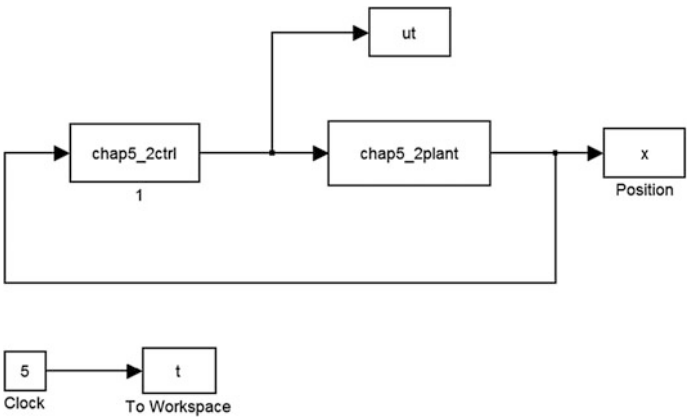
```
a=l/(m+M);beta=cos(88*pi/180);
a1=4*l/3-a*m*l;
A1=[0 1;g/a1 0];
B1=[0 ;-a/a1];
a2=4*l/3-a*m*l*beta^2;
A2=[0 1;2*g/(pi*a2) 0];
B2=[0;-a*beta/a2];
 Q=sdpvar(2,2);
 V1=sdpvar(1,2);
 V2=sdpvar(1,2);
 L1=Q*A1'+A1*Q+V1'*B1'+B1*V1;
 L2=Q*A2'+A2*Q+V2'*B2'+B2*V2;
 L3=Q*A1'+A1*Q+Q*A2'+A2*Q+V2'*B1'+B1*V2+V1'*B2'+B2*V1;
  F=set(L1<0)+set(L2<0)+set(L3<0)+set(Q>0);
solvesdp(F);  %To get Q, V1, V2
  Q=double(Q);
  V1=double(V1);
  V2=double(V2);
  P=inv(Q);
   K1=V1*P
   K2=V2*P
saveK_fileK1K2;
```

(2) Membership function design: **chap5_2mf.m;**

```
clearall;
closeall;
L1=-pi/2;L2=pi/2;
L=L2-L1;
h=pi/2;
N=L/h;
T=0.01;
x=L1:T:L2;
fori=1:N+1
e(i)=L1+L/N*(i-1);
end
u=trimf(x,[e(1),e(2),e(3)]);          %The middle MF
plot(x,u,'r','linewidth',2);
for j=1:N
if j==1
     u=trimf(x,[e(1),e(1),e(2)]);     %The first MF
elseif j==N
     u=trimf(x,[e(N),e(N+1),e(N+1)]); %The last MF
end
```

```
holdon;
plot(x,u,'b','linewidth',2);
end
xlabel('x');ylabel('Membership function');
legend('First Rule','Second rule');
```

(3) Simulink main program: **chap5_2sim.mdl;**



(4) Fuzzy controller program: **chap5_2ctrl.m;**

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
sys=mdlOutputs(t,x,u);
case {2,4,9}
sys=[];
otherwise
error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
```

```
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [0 0];
function sys=mdlOutputs(t,x,u)
x=[u(1);u(2)];
loadK_file;
ut1=K1*x;
ut2=K2*x;
L1=-pi/2;L2=pi/2;
L=L2-L1;
N=2;
fori=1:N+1
e(i)=L1+L/N*(i-1);
end
h1=trimf(x(1),[e(1),e(2),e(3)]);     %The middle
if x(1)<=0
  h2=trimf(x(1),[e(1),e(1),e(2)]);   %The first
else
  h2=trimf(x(1),[e(2),e(3),e(3)]);   %The last
end
%h1+h2
ut=(h1*ut1+h2*ut2)/(h1+h2);
sys(1)=ut;
```

(5) Plot program: **chap5_2plot.m.**

```
closeall;
figure(1);
plot(t,x(:,1),'r',t,x(:,2),'b');
xlabel('time(s)');ylabel('angle and angle speed response');
figure(2);
plot(t,ut(:,1),'r');
xlabel('time(s)');ylabel('control input');
```

5.3.3 Simulation Based on Four Fuzzy Rules Design

In order to control the pendulum in a wide range of initial angle, the number of fuzzy rules should be increased on the basis of the above two rules.

From (5.14), we know if $x_1 \to \pm\frac{\pi}{2}\left(|x_1| > \frac{\pi}{2}\right)$, then $\sin x_1 \to \pm 1 \to \frac{2}{\pi}x_1$, let $\beta = \cos(88^\circ)$, then $\cos(x_1) = \cos(180^\circ - 88^\circ) = -\cos(88^\circ) = -\beta$; if $x_1 \to \pi$, then $\sin x_1 \to 0$, $\cos x_1 \to -1$, and $\dot{x}_2 = \frac{au}{4l/3 - aml}$.

From above, we can get another two T-S type fuzzy rules:

Rule 3: IF $x_1(t)$ is about $\pm\frac{\pi}{2}\left(|x_1| > \frac{\pi}{2}\right)$, THEN $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}_3\boldsymbol{x}(t) + \boldsymbol{B}_3 u(t)$;
Rule 4: IF $x_1(t)$ is about $\pm\pi$, then $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}_4\boldsymbol{x}(t) + \boldsymbol{B}_4 u(t)$

where $\boldsymbol{A}_3 = \begin{bmatrix} 0 & 1 \\ \frac{2g}{\pi(4l/3 - aml\beta^2)} & 0 \end{bmatrix}$, $\boldsymbol{B}_3 = \begin{bmatrix} 0 \\ \frac{\alpha\beta}{4l/3 - aml\beta^2} \end{bmatrix}$, $\boldsymbol{A}_4 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$, $\boldsymbol{B}_4 = \begin{bmatrix} 0 \\ \frac{\alpha}{4l/3 - aml} \end{bmatrix}$.

Then we can design two fuzzy control rules as follows

Rule 3: If $x_1(t)$ is about $\pm\frac{\pi}{2}\left(|x_1| > \frac{\pi}{2}\right)$ then $u = \mathbf{K}_3\mathbf{x}(t)$

Rule 4: If $x_1(t)$ is about $\pm\pi$ then $u = \mathbf{K}_4\mathbf{x}(t)$

According to theorem 5.1, using (5.12), LMI of the inverted pendulum for above fuzzy rules can be expressed as

$$\begin{aligned} &\boldsymbol{Q}\boldsymbol{A}_1^{\mathrm{T}} + \boldsymbol{A}_1\boldsymbol{Q} + \boldsymbol{V}_1^{\mathrm{T}}\boldsymbol{B}_1^{\mathrm{T}} + \boldsymbol{B}_1\boldsymbol{V}_1 < 0, \\ &\boldsymbol{Q}\boldsymbol{A}_2^{\mathrm{T}} + \boldsymbol{A}_2\boldsymbol{Q} + \boldsymbol{V}_2^{\mathrm{T}}\boldsymbol{B}_2^{\mathrm{T}} + \boldsymbol{B}_2\boldsymbol{V}_2 < 0, \\ &\boldsymbol{Q}\boldsymbol{A}_3^{\mathrm{T}} + \boldsymbol{A}_3\boldsymbol{Q} + \boldsymbol{V}_3^{\mathrm{T}}\boldsymbol{B}_3^{\mathrm{T}} + \boldsymbol{B}_3\boldsymbol{V}_3 < 0, \\ &\boldsymbol{Q}\boldsymbol{A}_4^{\mathrm{T}} + \boldsymbol{A}_4\boldsymbol{Q} + \boldsymbol{V}_4^{\mathrm{T}}\boldsymbol{B}_4^{\mathrm{T}} + \boldsymbol{B}_4\boldsymbol{V}_4 < 0, \\ &\boldsymbol{Q} = \boldsymbol{P}^{-1} > 0 \end{aligned} \tag{5.17}$$

where $\boldsymbol{K}_1 = \boldsymbol{V}_1\boldsymbol{P}$, $\boldsymbol{K}_2 = \boldsymbol{V}_2\boldsymbol{P}$, $\boldsymbol{K}_3 = \boldsymbol{V}_3\boldsymbol{P}$, $\boldsymbol{K}_4 = \boldsymbol{V}_4\boldsymbol{P}$, $i = 1, 2, 3, 4$.

From above two LMI, the programs are given as follows:

```
L1=Q*A1'+A1*Q+V1'*B1'+B1*V1;
L2=Q*A2'+A2*Q+V2'*B2'+B2*V2;
L3=Q*A3'+A3*Q+V3'*B3'+B3*V3;
L4=Q*A4'+A4*Q+V4'*B4'+B4*V4;
```

Schematic diagram of membership function with four fuzzy rules are shown in Fig. 5.7, we can see that Rule 1 intersect Rule 2, and Rule 3 intersect Rule 4, therefore, only two LMI can be constructed from (5.9), the corresponding LMI is as follows.

$$\begin{aligned} &\boldsymbol{Q}\boldsymbol{A}_1^{\mathrm{T}} + \boldsymbol{A}_1\boldsymbol{Q} + \boldsymbol{Q}\boldsymbol{A}_2^{\mathrm{T}} + \boldsymbol{A}_2\boldsymbol{Q} + \boldsymbol{V}_2^{\mathrm{T}}\boldsymbol{B}_1^{\mathrm{T}} + \boldsymbol{B}_1\boldsymbol{V}_2 + \boldsymbol{V}_1^{\mathrm{T}}\boldsymbol{B}_2^{\mathrm{T}} + \boldsymbol{B}_2\boldsymbol{V}_1 < 0 \\ &\boldsymbol{Q}\boldsymbol{A}_3^{\mathrm{T}} + \boldsymbol{A}_3\boldsymbol{Q} + \boldsymbol{Q}\boldsymbol{A}_4^{\mathrm{T}} + \boldsymbol{A}_4\boldsymbol{Q} + \boldsymbol{V}_4^{\mathrm{T}}\boldsymbol{B}_3^{\mathrm{T}} + \boldsymbol{B}_3\boldsymbol{V}_4 + \boldsymbol{V}_3^{\mathrm{T}}\boldsymbol{B}_4^{\mathrm{T}} + \boldsymbol{B}_4\boldsymbol{V}_3 < 0 \end{aligned} \tag{5.18}$$

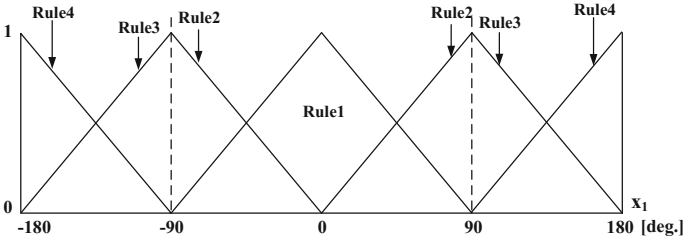


Fig. 5.7 Schematic diagram of membership function

From above two LMI, the programs are given as follows:

$$\mathrm{L5} = \mathrm{Q} * \mathrm{A1}' + \mathrm{A1} * \mathrm{Q} + \mathrm{Q} * \mathrm{A2}' + \mathrm{A2} * \mathrm{Q} + \mathrm{V2}' * \mathrm{B1}' + \mathrm{B1} * \mathrm{V2} + \mathrm{V1}' * \mathrm{B2}' + \mathrm{B2} * \mathrm{V1};$$
$$\mathrm{L6} = \mathrm{Q} * \mathrm{A3}' + \mathrm{A3} * \mathrm{Q} + \mathrm{Q} * \mathrm{A4}' + \mathrm{A4} * \mathrm{Q} + \mathrm{V4}' * \mathrm{B3}' + \mathrm{B3} * \mathrm{V4} + \mathrm{V3}' * \mathrm{B4}' + \mathrm{B4} * \mathrm{V3};$$

From above three LMI, the programs are designed as follows:

```
F=set(L1<0)+set(L2<0)+set(L3<0)+set(L4<0)+set(L5<0)+set
(L6<0)+set(Q>0);
```

According to (5.5), T-S based fuzzy controller is designed by using PDC method:

$$u = w_1(x_1)\boldsymbol{K}_1\boldsymbol{x}(t) + w_2(x_1)\boldsymbol{K}_2\boldsymbol{x}(t) + w_3(x_1)\boldsymbol{K}_3\boldsymbol{x}(t) + w_4(x_1)\boldsymbol{K}_4\boldsymbol{x}(t) \tag{5.19}$$

According to the rules of two T-S fuzzy model of the inverted pendulum, the membership function is designed in Fig. 5.3. The triangular membership function is used to fuzzify $x_1(t)$, and the initial states are $[\pi \quad 0]$.

Using LMI toolbox, YALMIP toolbox, to get $\boldsymbol{K}_i$, the program is chap5_3LMI.m, we have $\boldsymbol{Q}$, $\boldsymbol{V}_1$, $\boldsymbol{V}_2$, $\boldsymbol{V}_3$, $\boldsymbol{V}_4$, then we can get $\boldsymbol{K}_1 = [3301.3 \quad 969.9]$, $\boldsymbol{K}_2 = [6366.3 \quad 1879.7]$, $\boldsymbol{K}_3 = [-6189.6 \quad -1883.7]$, $\boldsymbol{K}_4 = [-3105.2 \quad -969.9]$. Running Simulink main program chap5_3sim.mdl, the simulation results are shown in Figs. 5.8, 5.9, and 5.10.

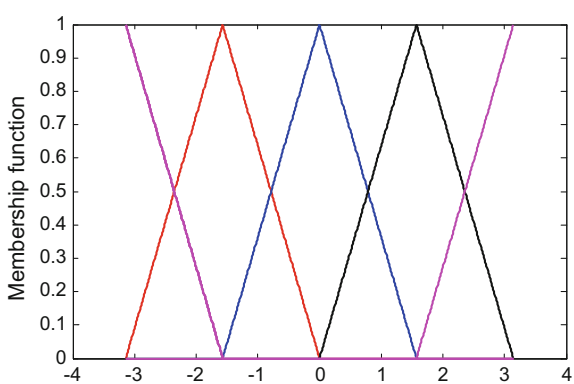


Fig. 5.8
Membership function

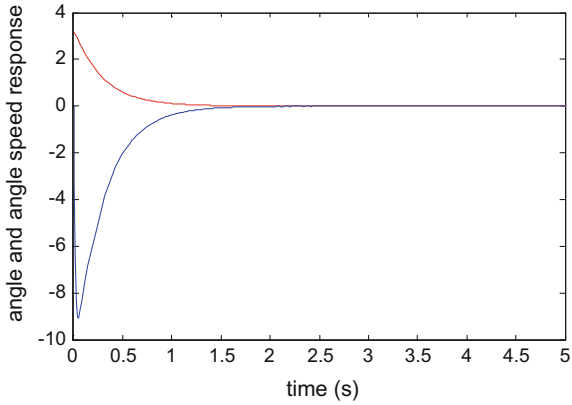


Fig. 5.9 Angle and angle speed response

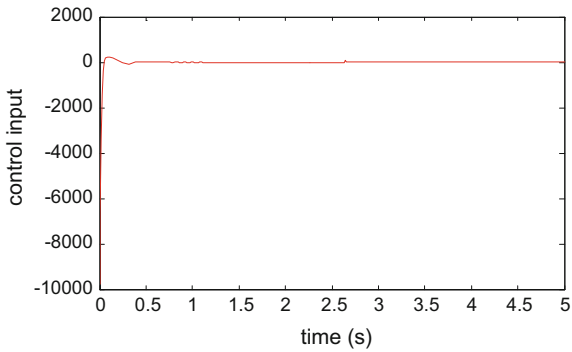


Fig. 5.10 Control input

Matlab Programs:

(1) Controller gain based on LMI: **chap5_3LMI**.m;

```
clear all;
close all;
g=9.8;m=2.0;M=8.0;l=0.5;
a=1/(m+M);beta=cos(88*pi/180);
a1=4*l/3-a*m*l;
A1=[0 1;g/a1 0];
B1=[0 ;-a/a1];
a2=4*l/3-a*m*l*beta^2;
A2=[0 1;2*g/(pi*a2) 0];
B2=[0;-a*beta/a2];
A3=[0 1;2*g/(pi*a2) 0];
B3=[0;a*beta/a2];
A4=[0 1;0 0];
```

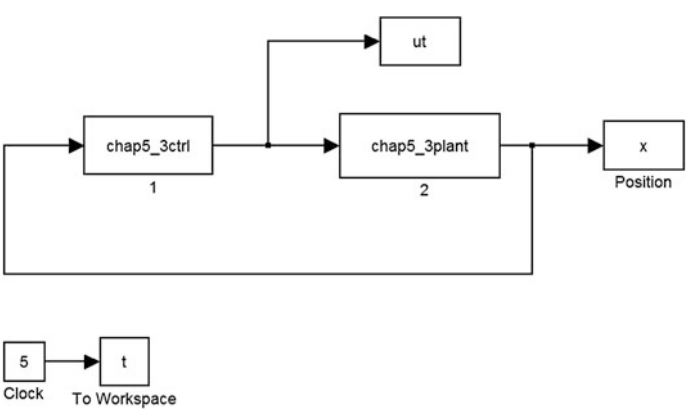
```
B4=[0;a/a1];
 Q=sdpvar(2,2);
 V1=sdpvar(1,2);
 V2=sdpvar(1,2);
 V3=sdpvar(1,2);
 V4=sdpvar(1,2);
L1=Q*A1'+A1*Q+V1'*B1'+B1*V1;
L2=Q*A2'+A2*Q+V2'*B2'+B2*V2;
L3=Q*A3'+A3*Q+V3'*B3'+B3*V3;
L4=Q*A4'+A4*Q+V4'*B4'+B4*V4;
L5=Q*A1'+A1*Q+Q*A2'+A2*Q+V2'*B1'+B1*V2+V1'*B2'+B2*V1; %from R1 and R2
L6=Q*A3'+A3*Q+Q*A4'+A4*Q+V4'*B3'+B3*V4+V3'*B4'+B4*V3; %from R3 and R4
F=set(L1<0)+set(L2<0)+set(L3<0)+set(L4<0)+set(L5<0)+set(L6<0)+set
(Q>0);
solvesdp(F);  %To get Q, V1, V2, V3, V4
 Q=double(Q);
 V1=double(V1);
 V2=double(V2);
 V3=double(V3);
 V4=double(V4);
 P=inv(Q);
  K1=V1*P
  K2=V2*P
  K3=V3*P
  K4=V4*P
saveK_fileK1K2K3K4;
```

(2) Membership function design: **chap5_3mf.m;**

```
clear all;
close all;
L1=-pi;L2=pi;
L=L2-L1;
h=pi/2;
N=L/h;
T=0.01;
x=L1:T:L2;
fori=1:N+1
e(i)=L1+L/N*(i-1);
end
figure(2);
% h1
h1=trimf(x,[e(2),e(3),e(4)]);     %Rule 1:x1 is to zero
plot(x,h1,'r','linewidth',2);
```

```
% h2, Rule 2: x1 is about +-pi/2,but smaller
%if x<=0
  h2=trimf(x,[e(2),e(2),e(3)]);
holdon
plot(x,h2,'b','linewidth',2);
%else
  h2=trimf(x,[e(3),e(4),e(4)]);
holdon
plot(x,h2,'b','linewidth',2);
%end
% h3, Rule 3: x1 is about +-pi/2,but bigger
%if x<0
   h3=trimf(x,[e(1),e(2),e(2)]);
holdon;
plot(x,h3,'g','linewidth',2);
%else
   h3=trimf(x,[e(4),e(4),e(5)]);
holdon;
plot(x,h3,'g','linewidth',2);
%end
% h4, Rule 4: x1 is about +-pi
%if x<0
   h4=trimf(x,[e(1),e(1),e(2)]);
holdon;
plot(x,h4,'k','linewidth',2);
% else
   h4=trimf(x,[e(4),e(5),e(5)]);
holdon;
plot(x,h4,'k','linewidth',2);
%end
```

(3) Simulink Program: **chap5_3**sim.mdl;



(4) S function for controller design: **chap5_3**ctrl.m;

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
sys=mdlOutputs(t,x,u);
case {2,4,9}
sys=[];
otherwise
error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [0 0];
function sys=mdlOutputs(t,x,u)
x=[u(1);u(2)];
loadK_file;
ut1=K1*x;
ut2=K2*x;
ut3=K3*x;
ut4=K4*x;
L1=-pi;L2=pi;
L=L2-L1;
h=pi/2;
N=L/h;
fori=1:N+1
e(i)=L1+L/N*(i-1);
end
% h1
h1=trimf(x(1),[e(2),e(3),e(4)]);      %Rule 1:x1 is to zero
% h2, Rule 2: x1 is about +-pi/2,but smaller
if x(1)<=0
  h2=trimf(x(1),[e(2),e(2),e(3)]);
```

```
else
  h2=trimf(x(1),[e(3),e(4),e(4)]);
end
% h3, Rule 3: x1 is about +-pi/2,but bigger
if x(1)<0
   h3=trimf(x(1),[e(1),e(2),e(2)]);
else
   h3=trimf(x(1),[e(4),e(4),e(5)]);
end
% h4, Rule 4: x1 is about +-pi
if x(1)<0
  h4=trimf(x(1),[e(1),e(1),e(2)]);
else
  h4=trimf(x(1),[e(4),e(5),e(5)]);
end
h1+h2+h3+h4;
ut=(h1*ut1+h2*ut2+h3*ut3+h4*ut4)/(h1+h2+h3+h4);
sys(1)=ut;
```

(5) Plot program: **chap5_3**plot.m.

```
closeall;
figure(1);
plot(t,x(:,1),'r',t,x(:,2),'b');
xlabel('time(s)');ylabel('angle and angle speed response');
figure(2);
plot(t,ut(:,1),'r');
xlabel('time(s)');ylabel('control input');
```

5.4 Simulation Example of YALMIP Toolbox

YALMIP is an independent Matlab toolbox, it has a strong ability to optimize the solution, and the toolbox has the following features:

(1) YALMIP is a toolbox based on symbolic computing toolbox;
(2) YALMIP is a modeling language for defining and solving advanced optimization problems;
(3) YALMIP toolbox is used to solve linear programming, integer programming, nonlinear programming, mixed programming, and other standard optimization problems and LMI problems.

YALMIP toolbox can be used to solve the LMI problem. LMI constraints can be described by the command "set," without specific description of the location and content of the inequality, the results can be used to view by "double."

YALMIP toolbox can be downloaded from the network for free; the toolbox name is "yalmip.rar".

For example, consider a LMI as

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{P}+\boldsymbol{F}^{\mathrm{T}}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{P}+\boldsymbol{P}\boldsymbol{A}+\boldsymbol{P}\boldsymbol{B}\boldsymbol{F}<0 \tag{5.20}$$

set $\boldsymbol{A}=\begin{bmatrix}-2.548 & 9.1 & 0\\ 1 & -1 & 0\\ 0 & -14.2 & 0\end{bmatrix}$, $\boldsymbol{B}=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$, $\boldsymbol{P}=\begin{bmatrix}1000000 & 0 & 0\\ 0 & 1000000 & 0\\ 0 & 0 & 1000000\end{bmatrix}$, solve the LMI by YALMIP toolbox, we can get $\boldsymbol{F}=\begin{bmatrix}-492.4768 & -5.05 & 0\\ -5.05 & -494.0248 & 6.6\\ 0 & 6.6 & -495.0248\end{bmatrix}$.

Program: chap5_4.m

```
clear all;
close all;
%First example
A = [-2.548 9.1 0;1 -1 1;0 -15.2 0];
B = [1 0 0;0 1 0;0 0 1];
F = sdpvar(3,3);
P = 1,000,000*eye(3);
FAI = (A' + F'*B')*P + P*(A + B*F);
%LMI description
L = set(FAI < 0);
solvesdp(L);
F = double(F)
```

References

1. M. Sugeno, G.T. Kang, Fuzzy modeling and control of multilayer incinerator. Fuzzy Sets Syst. **18**, 329–346 (1986)
2. K. Tanaka, M. Sugeno, Stability analysis and design of fuzzy control systems. Fuzzy Sets Syst. **45**(2), 135–156 (1992)

3. H.O. Wang, K. Tanaka, M.F. Griffin, Parallel distributed compensation of nonlinear systems by Takagi-Sugeno fuzzy model, International Joint Conference of the Fourth IEEE International Conference on Fuzzy Systems and The Second International Fuzzy Engineering Symposium, 1995. pp. 531–538
4. S. Farinwata, D. Filev, R. Langari, Fuzzy Control: Synthesis and Analysis (Wiley, 2000)
5. H.O. Wang, K. Tanaka, M. Griffin, An analytical framework of fuzzy modeling and control of nonlinear systems: stability and design issues. Am. Control Conf. **3**, 2272–2276 (1995)

Chapter 6
Adaptive Fuzzy Control

6.1 Adaptive Fuzzy Control

Since the idea of fuzzy system universal approximation theorem was introduced [1], adaptive fuzzy control techniques have undergone great developments and have been successfully applied in many fields such as learning, pattern recognition, signal processing, modeling, and system control. The major advantages of adaptive fuzzy control greatly motivate the usage in nonlinear system identification and control [2].

There are several reasons that have motivated vast research interests in the application of adaptive fuzzy control for control purposes, as alternatives to traditional control methods, among which the main points are:

(1) Better performance is usually achieved since adaptive fuzzy controller can adjust itself to the changing environment.
(2) Modeling is not needed, and the adaptive law can help to learn the dynamics of the plant during operation.

6.2 Fuzzy Approximation

6.2.1 Fuzzy System Design

Step 1. Define N_i fuzzy sets A_i^1, A_i^2 ,..., $A_i^{N_i}$ in $[\alpha_i, \beta_i]$, and design membership functions $\mu_{A_i^1}$, ..., $\mu_{A_i^{N_i}}$.

Step 2. Design $M = N_1 \times N_2$ fuzzy rules in the following form:

$R_u^{i_1 i_2}$: if x_1 is $A_1^{i_1}$ and x_2 is $A_2^{i_2}$,then y is $B^{i_1 i_2}$
where $i_1 = 1, 2, \ldots, N_1,\ i_2 = 1, 2, \quad \ldots, N_2$.

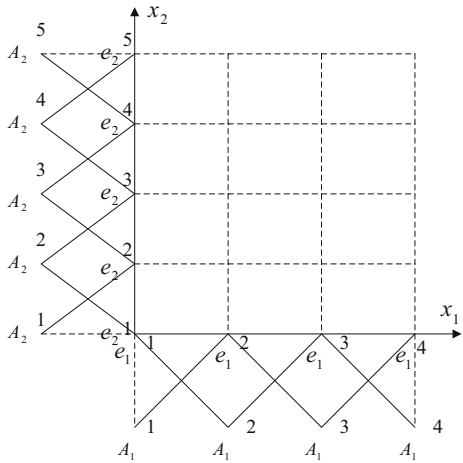


Fig. 6.1 Example of fuzzy sets

The center of fuzzy set $B^{i_1 i_2}$ is designed as

$$\bar{y}^{i_1 i_2} = g\left(e_1^{i_1}, \quad e_2^{i_2}\right) \tag{6.1}$$

Step 3. Design fuzzy system $f(x)$ from the above $N_1 \times N_2$ fuzzy rules by using product inference engine, singleton fuzzifier, and center average defuzzifier:

$$f(x) = \frac{\sum_{i_1=1}^{N_1} \sum_{i_2=1}^{N_2} \bar{y}^{i_1 i_2} \left(\mu_{A_1^{i_1}}(x_1) \mu_{A_2^{i_2}}(x_2)\right)}{\sum_{i_1=1}^{N_1} \sum_{i_2=1}^{N_2} \left(\mu_{A_1^{i_1}}(x_1) \mu_{A_2^{i_2}}(x_2)\right)} \tag{6.2}$$

Figure 6.1 shows an example of fuzzy sets with $N_1 = 4$, $N_2 = 5$, $\alpha_1 = \alpha_2 = 0$, and $\beta_1 = \beta_2 = 1$.

6.2.2 *Fuzzy System Approximation*

Fuzzy system approximation is based on universal approximation theorem as follows.

Theorem 6.1 Universal approximation theorem [1, 2]

Let $f(x)$ be the fuzzy system (6.2); if $g(x)$ is continuously differentiable on $U = [\alpha_1, \quad \beta_1] \times [\alpha_1 \quad \beta_2]$, then we can get the approximation accuracy as

$$\|g-f\|_\infty \le \left\|\frac{\partial g}{\partial x_1}\right\|_\infty h_1 + \left\|\frac{\partial g}{\partial x_2}\right\|_\infty h_2 \tag{6.3}$$

where

$$h_i = \max_{1\le j\le N_i-1}\left|e_i^{j+1} - e_i^j\right| (i=1,\quad 2) \tag{6.4}$$

where $\|*\|_\infty$ is defined as $\|d(x)\|_\infty = \sup_{x\in U}|d(x)|$.

From (6.4), we can get a conclusion: If the number of fuzzy sets of x_i is N_i ,and the length of its range is L_i, then the approximation accuracy of fuzzy system is $h_i = \frac{L_i}{N_i-1}$,that is $N_i \ge \frac{L_i}{h_i} + 1$.

From the theorem, we can get the following conclusions:

(1) In the universal approximation (6.3), for $\varepsilon > 0$, if we design h_1 and h_2 as small enough, we can get $\left\|\frac{\partial g}{\partial x_1}\right\|_\infty h_1 + \left\|\frac{\partial g}{\partial x_2}\right\|_\infty h_2 < \varepsilon$, and $\sup_{x\in U}|g(x)-f(x)| = \|g-f\|_\infty < \varepsilon$ can be ensured.

(2) Since $N_i \ge \frac{L_i}{h_i} + 1$, the more fuzzy sets designed are, the smaller value of h_i is. That is, to get more approximation accuracy of fuzzy system, we must design more fuzzy sets.

(3) To design a fuzzy system with a specified accuracy, we must get $\left\|\frac{\partial g}{x_1}\right\|_\infty$ and $\left\|\frac{\partial g}{x_2}\right\|_\infty$, and we must also get the value of $g(x)$ at $x = (e_1^{i_1}, e_2^{i_2})$, $(i_1 = 1,\quad 2,\quad \cdots,\quad N_1, i_2 = 1,\quad 2,\quad \cdots,\quad N_2)$.

6.2.3 Simulation Example

6.2.3.1 One Dimension Function Approximation

Consider one dimension function as

$$g(x) = \sin x \tag{6.5}$$

where $x \in [-3, +3]$.

Define triangle membership function in $[L_1 \quad L_2]$ as shown in Fig. 6.2. The fuzzy system is designed as

$$f(x) = \frac{\sum_{j=1}^{N} \sin(e^j)\,\mu_{A^j}(x)}{\sum_{j=1}^{N} \mu_{A^j}(x)} \tag{6.6}$$

If we choose $N = 30$, then MF design and function approximation are shown in Figs. 6.3 and 6.4. If we choose N = 50, then the function approximation error is

Fig. 6.2 Membership function

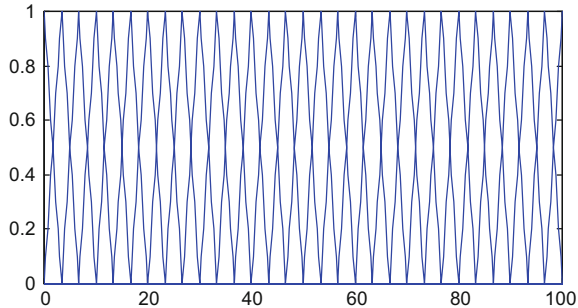


Fig. 6.3 Function approximation

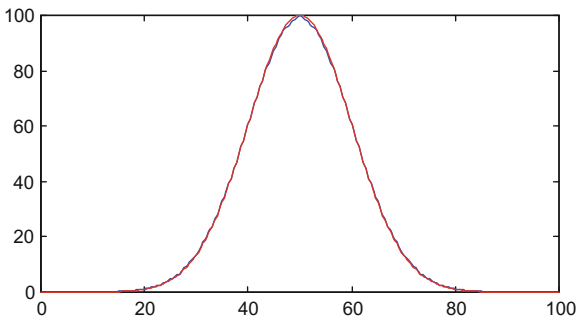
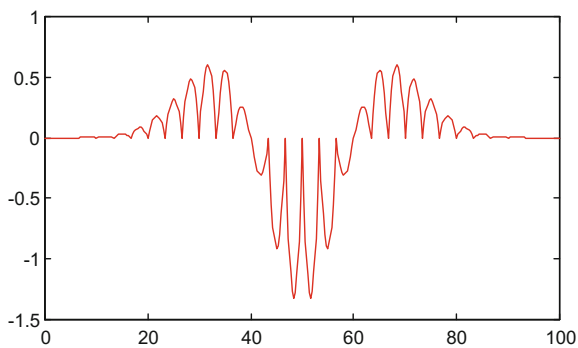


Fig. 6.4 Function approximation error ($N = 30$)

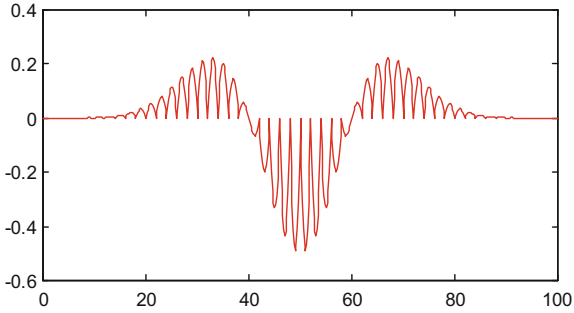


shown in Fig. 6.5. We can see that the more fuzzy sets are designed, the smaller the approximation error is gotten.

Matlab program: chap6_1.m

```
%Fuzzy approximation
clear all;
close all;
```

Fig. 6.5 Function approximation error ($N = 50$)




```
L1=-3;L2=3;
L=L2-L1;

h=0.2;
N=L/h+1;
T=0.01;

x=L1:T:L2;
for i=1:N
   e(i)=L1+L/(N-1)*(i-1);
end

c=0;d=0;
for j=1:N
  if j==1
     u=trimf(x,[e(1),e(1),e(2)]);    %The first MF
  elseif j==N
     u=trimf(x,[e(N-1),e(N),e(N)]); %The last MF
   else
     u=trimf(x,[e(j-1),e(j),e(j+1)]);
  end
  hold on;
  plot(x,u);
  c=c+sin(e(j))*u;
  d=d+u;
end
xlabel('x');ylabel('Membership function');

for k=1:L/T+1
   f(k)=c(k)/d(k);
end
```

```
y=sin(x);
figure(2);
plot(x,f,'b',x,y,'r');
xlabel('x');ylabel('fx approximation');
figure(3);
plot(x,f-y,'r');
xlabel('x');ylabel('approximation error');
```

6.2.3.2 Two Dimension Function Approximation

Consider two dimension function as

$$g(x) = 0.52 + 0.1\,x_1^2 + 0.28\,x_2^2 - 0.06\,x_1 x_2 \tag{6.7}$$

Define triangle membership function in $U = [-1 \quad 1] \times [-1 \quad 1]$ as shown in Figs. 6.6 and 6.7; the fuzzy system is designed as

$$f(x) = \frac{\sum_{i_1=1}^{N1} \sum_{i_2=1}^{N2} g(e^{i_1}, e^{i_2})\, \mu_{A^{i_1}}(x_1)\, \mu_{A^{i_2}}(x_2)}{\sum_{i_1=1}^{N1} \sum_{i_2=1}^{N2} \mu_{A^{i_1}}(x_1)\, \mu_{A^{i_2}}(x_2)} \tag{6.8}$$

Choose $N1$=$N2$=11, the results are given in Figs. 6.8 and 6.9.

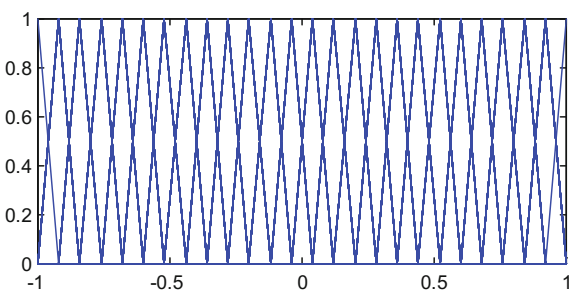


Fig. 6.6 Membership function of x_1

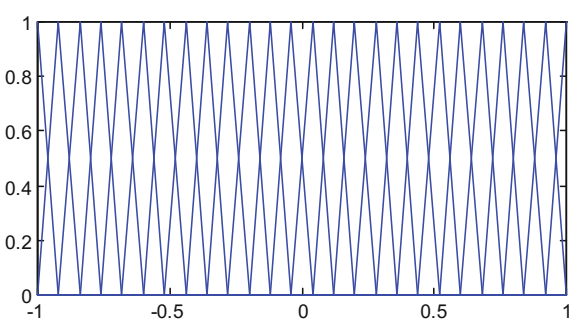


Fig. 6.7 Membership function of x_2

Fig. 6.8 Function approximation

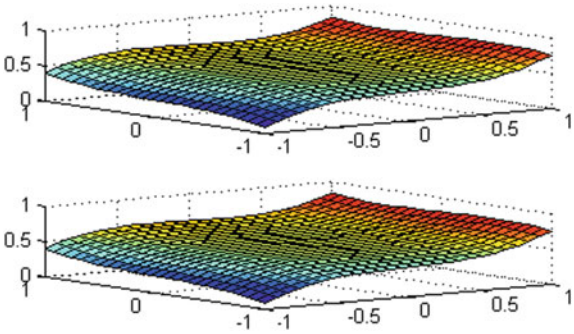
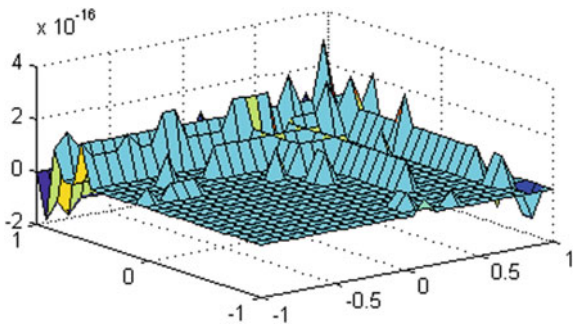


Fig. 6.9 Function approximation error



Program name: chap6_2.m

```
%Fuzzy approximation
clear all;
close all;

T=0.1;
x1=-1:T:1;
x2=-1:T:1;

L=2;
h=0.1;
N=L/h+1;

for i=1:1:N    %N MF
 for j=1:1:N
  e1(i)=-1+L/(N-1)*(i-1);
  e2(j)=-1+L/(N-1)*(j-1);
   gx(i,j)=0.52+0.1*e1(i)^3+0.28*e2(j)^3-0.06*e1(i)*e2(j);
  end
end
```

```
df=zeros(L/T+1,L/T+1);
cf=zeros(L/T+1,L/T+1);
for m=1:1:N              %u1 change from 1 to N
  if m==1
     u1=trimf(x1,[-1,-1,-1+L/(N-1)]);  %First u1
  elseif m==N
     u1=trimf(x1,[1-L/(N-1),1,1]);    %Last u1
  else
   u1=trimf(x1,[e1(m-1),e1(m),e1(m+1)]);
  end
figure(1);
hold on;
plot(x1,u1);
xlabel('x1');ylabel('Membership function');

for n=1:1:N                   %u2 change from 1 to N
  if n==1
    u2=trimf(x2,[-1,-1,-1+L/(N-1)]);  %First u2
  elseif n==N
    u2=trimf(x2,[1-L/(N-1),1,1]);    %Last u2
  else
    u2=trimf(x2,[e2(n-1),e2(n),e2(n+1)]);
  end
figure(2);
hold on;
plot(x2,u2);
xlabel('x2');ylabel('Membership function');

   for i=1:1:L/T+1
    for j=1:1:L/T+1
     d=df(i,j)+u1(i)*u2(j);
     df(i,j)=d;
     c=cf(i,j)+gx(m,n)*u1(i)*u2(j);
     cf(i,j)=c;
    end
  end
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:1:L/T+1
   for j=1:1:L/T+1
     f(i,j)=cf(i,j)/df(i,j);
     y(i,j)=0.52+0.1*x1(i)^3+0.28*x2(j)^3-0.06*x1(i)*x2(j);
   end
end
```

```
figure(3);
subplot(211);
surf(x1,x2,f);
title('f(x)');
subplot(212);
surf(x1,x2,y);
title('g(x)');
figure(4);
surf(x1,x2,f-y);
title('Approximation error');
```

6.3 Adaptive Fuzzy Controller Design

6.3.1 Problem Description

Consider a dynamic system as

$$\ddot{\theta} = f\left(\theta, \dot{\theta}\right) + u \tag{6.9}$$

where θ is angle and u is control input.

We can rewrite (6.9) as

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x) + u \end{aligned} \tag{6.10}$$

where $x_1 = \theta, f(x) = f(x_1, x_2) = f\left(\theta, \dot{\theta}\right)$ is unknown.

Assuming ideal angle is x_d, then we get

$$e = x_1 - x_d,\ \dot{e} = x_2 - \dot{x}_d$$

Define error function as

$$s = ce + \dot{e},\ c > 0 \tag{6.11}$$

then

$$\dot{s} = c\dot{e} + \ddot{e} = c\dot{e} + \dot{x}_2 - \ddot{x}_d = c\dot{e} + f(x) + u - \ddot{x}_d$$

From (6.11), we have if $s \to 0$,then $e \to 0$ and $\dot{e} \to 0$.

6.3.2 Fuzzy Approximation

Using the universal approximation theorem of fuzzy system, we design fuzzy system $\hat{f}(\boldsymbol{x}|\boldsymbol{\theta})$ to approximate $f(\boldsymbol{x})$.

Consider the input x_1 and x_2, we design five MF, then we get $n = 2$, $i = 1, 2$, $p_1 = p_2 = 5$, and we can get $p_1 \times p_2 = 25$ fuzzy rules.

We use two steps to construct fuzzy system $\hat{f}(\boldsymbol{x}|\boldsymbol{\theta})$ as follows:

Step 1: For the variable x_i $(i = 1, 2)$, define p_i fuzzy sets $A_i^{l_i}$ $(l_i = 1, 2, 3, 4, 5)$;

Step 2: Use $\prod_{i=1}^{n} p_i = p_1 \times p_2 = 25$ fuzzy rules to construct fuzzy system $\hat{f}(\boldsymbol{x}|\boldsymbol{\theta})$. The jth fuzzy rule is expressed as

$$\mathrm{R}^{(j)}: \text{if } x_1 \text{ is } A_1^{l_1} \text{ and } x_2 \text{ is } A_1^{l_2} \text{ then } \hat{f} \text{ is } B^{l_1 l_2} \tag{6.12}$$

where $l_i = 1, 2, 3, 4, 5$, $i = 1, 2$, $j = 1, 2, \cdots, 25$, $B^{l_1 l_2}$ is the fuzzy sets of $\hat{f}$.

Then, the first and the twenty-fifth fuzzy rule can be expressed as

$$\mathrm{R}^{(1)}: \text{if } x_1 \text{ is } A_1^1 \text{ and } x_2 \text{ is } A_2^1 \text{ then } \hat{f} \text{ is } E^1$$

$$\mathrm{R}^{(25)}: \text{if } x_1 \text{ is } A_1^5 \text{ and } x_2 \text{ is } A_1^5 \text{ then } \hat{f} \text{ is } E^{25}$$

The fuzzy inference is designed as follows:

(1) Using product inference engine for the premise of fuzzy rule, we can get $\prod_{i=1}^{2} \mu_{A_i^{l_i}}(x_i)$.

(2) Use singleton fuzzifier to get $\bar{y}_{\mathrm{f}}^{l_1 l_2}$, where $\bar{y}_{\mathrm{f}}^{l_1 l_2} = f(x_1, x_2)$ is the point $[x_1, x_2]$ at which $\mu_{B^{l_1 l_2}}\left(\bar{y}_{\mathrm{f}}^{l_1 l_2}\right)$ achieves its maximum value, and we assume that $\mu_{B^{l_1 l_2}}\left(\bar{y}_{\mathrm{f}}^{l_1 l_2}\right) = 1.0$.

(3) Using product inference engine for the premise and conclusion of fuzzy rule, then we get $\bar{y}_{\mathrm{f}}^{l_1 l_2}\left(\prod_{i=1}^{2} \mu_{A_i^{l_i}}(x_i)\right)$, and using the union operator for different fuzzy rules, then we can get the output of fuzzy system as $\sum_{l_1=1}^{5}\sum_{l_2=1}^{5} \bar{y}_{\mathrm{f}}^{l_1 l_2}]$ $\left(\prod_{i=1}^{2} \mu_{A_i^{l_i}}(x_i)\right)$.

(4) Using the center average defuzzifier, we can get output of the fuzzy system.

$$\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}) = \frac{\sum_{l_1=1}^{5}\sum_{l_2=1}^{5} \bar{y}_{\mathrm{f}}^{l_1 l_2}\left(\prod_{i=1}^{2} \mu_{A_i^{l_i}}(x_i)\right)}{\sum_{l_1=1}^{5}\sum_{l_2=1}^{5}\left(\prod_{i=1}^{2} \mu_{A_i^{l_i}}(x_i)\right)} \tag{6.13}$$

Let $\bar{y}_f^{l_1 l_2}$ to be freedom parameter, $\boldsymbol{\theta} = \left[\bar{y}_f^1 \quad \cdots \quad \bar{y}_f^{25}\right]^{\mathrm{T}}$ is a parameter vector, introduce the fuzzy basis vector $\boldsymbol{\xi}(\boldsymbol{x})$, then (6.13) becomes

$$\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}) = \hat{\boldsymbol{\theta}}^{\mathrm{T}}\boldsymbol{\xi}(\boldsymbol{x}) \tag{6.14}$$

where $\boldsymbol{\xi}(\boldsymbol{x})$ is fuzzy basis vector with $\prod_{i=1}^{n} p_i = p_1 \times p_2 = 25$ elements, its $l_1 l_2$ th element is

$$\xi_{l_1 l_2}(x) = \frac{\prod_{i=1}^{2} \mu_{A_i^{l_i}}(x_i)}{\sum_{l_1=1}^{5}\sum_{l_2=1}^{5}\left(\prod_{i=1}^{2} \mu_{A_i^{l_i}}(x_i)\right)} \tag{6.15}$$

6.3.3 *Adaptive Fuzzy Control Design and Analysis*

We set the optimum parameter as

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Omega}}\left[\sup_{x\in\boldsymbol{R}^2}\left|\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}) - f(\boldsymbol{x})\right|\right] \tag{6.16}$$

Then,

$$f(x) = \boldsymbol{\theta}^{*\mathrm{T}}\boldsymbol{\xi}(\boldsymbol{x}) + \varepsilon$$

where ε is the approximation error.

$$f(x) - \hat{f}(x) = \boldsymbol{\theta}^{*\mathrm{T}}\boldsymbol{\xi}(\boldsymbol{x}) + \varepsilon - \hat{\boldsymbol{\theta}}\xi(\boldsymbol{x}) = -\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\boldsymbol{\xi}(\boldsymbol{x}) + \varepsilon$$

Define Lyapunov function as

$$V = \frac{1}{2}s^2 + \frac{1}{2\gamma}\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\tilde{\boldsymbol{\theta}} \tag{6.17}$$

where $\gamma > 0$, $\tilde{\boldsymbol{\theta}} = \hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^*$.

Then, $\dot{\tilde{\boldsymbol{\theta}}} = \dot{\hat{\boldsymbol{\theta}}}$, and

$$\dot{V} = s\dot{s} + \frac{1}{\gamma}\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\dot{\hat{\boldsymbol{\theta}}} = s(c\dot{e} + f(x) + u - \ddot{x}_{\mathrm{d}}) + \frac{1}{\gamma}\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\dot{\hat{\boldsymbol{\theta}}}$$

Design control law as

$$u = -c\dot{e} - \hat{f}(x) + \ddot{x}_{\mathrm{d}} - \eta \mathrm{sgn}(s) \tag{6.18}$$

Then,

$$\begin{aligned}\dot{V} &= s\big(f(x) - \hat{f}(x) - \eta \mathrm{sgn}(s)\big) + \frac{1}{\gamma}\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\dot{\hat{\boldsymbol{\theta}}} \\ &= s\big(-\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\boldsymbol{\xi}(\boldsymbol{x}) + \varepsilon - \eta \mathrm{sgn}(s)\big) + \frac{1}{\gamma}\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\dot{\hat{\boldsymbol{\theta}}} \\ &= \varepsilon s - \eta|s| + \tilde{\boldsymbol{\theta}}^{\mathrm{T}}\left(\frac{1}{\gamma}\dot{\hat{\boldsymbol{\theta}}} - s\boldsymbol{\xi}(\boldsymbol{x})\right)\end{aligned}$$

Choosing $\eta > |\varepsilon|_{\max} + \eta_0$, $\eta_0 > 0$, then adaptive law is

$$\dot{\hat{\boldsymbol{\theta}}} = \gamma s \boldsymbol{\xi}(\boldsymbol{x}) \tag{6.19}$$

Then, $\dot{V} = \varepsilon s - \eta|s| \leq -\eta_0|s| \leq 0$.

From above analysis, we can see that fuzzy system approximation error can be overcome by the robust term $\eta\,\mathrm{sgn}(s)$.

From $\dot{V} \leq -\eta_0|s| \leq 0$, we have

$$\int_0^t \dot{V}dt \leq -\eta_0 \int_0^t |s|dt, \text{i.e. } V(t) - V(0) \leq -\eta_0 \int_0^t |s|dt$$

Then V is limited, s and $\tilde{\boldsymbol{\theta}}$ are all limited, from $\dot{s}$ expression, $\dot{s}$ is limited, the $\int_0^\infty |s|dt$ is limited. From Barbalat Lemma [3], when $t \to \infty$, we have $s \to 0$, then $e \to 0$, $\dot{e} \to 0$.

Since V is limited as $t \to \infty$, thus $\hat{\boldsymbol{\theta}}$ is limited. Since when $\dot{V} \equiv 0$, we cannot get $\tilde{\boldsymbol{\theta}} \equiv 0$, $\hat{\boldsymbol{\theta}}$ will not converge to $\boldsymbol{\theta}^*$.

6.3.4 Simulation Example

Consider the plant as

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x) + u\end{aligned}$$

where $f(x) = 10x_1x_2$.

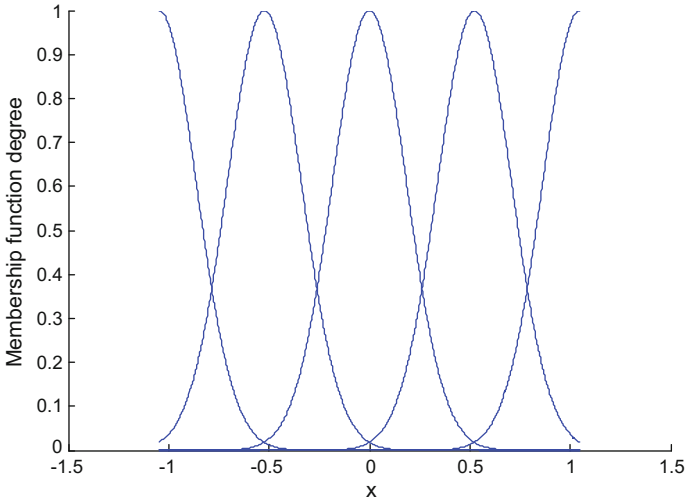


Fig. 6.10 MF of x_i

We consider that ideal position signal is $x_{\mathrm{d}}(t) = \sin t$, and choose five MF to fuzzy x_i as follows:

$$\mu_{\mathrm{NM}}(x_i) = \exp\left[-((x_i + \pi/3)/(\pi/12))^2\right],$$

$$\mu_{\mathrm{NS}}(x_i) = \exp\left[-((x_i + \pi/6)/(\pi/12))^2\right],$$

$$\mu_{\mathrm{Z}}(x_i) = \exp\left[-(x_i/(\pi/12))^2\right],$$

$$\mu_{\mathrm{PS}}(x_i) = \exp\left[-((x_i - \pi/6)/(\pi/12))^2\right],$$

$$\mu_{\mathrm{PM}}(x_i) = \exp\left[-((x_i - \pi/3)/(\pi/12))^2\right].$$

Then, we can get 25 fuzzy rules to construct fuzzy system $\hat{f}$. The MF is given as Fig. 6.10.

We use FS_2, FS_1 and FS to express the $\boldsymbol{\xi}(\boldsymbol{x})$ in the program. The initial states of the plant are $[0.15, 0]$, we use the control law (6.18) and adaptive law(6.19), the initial value of $\hat{\boldsymbol{\theta}}$ is set as 0.10, and choose $c = 15$, $\gamma = 5000$, $\eta = 0.50$. The simulation results are shown in Figs. 6.11 and 6.12.

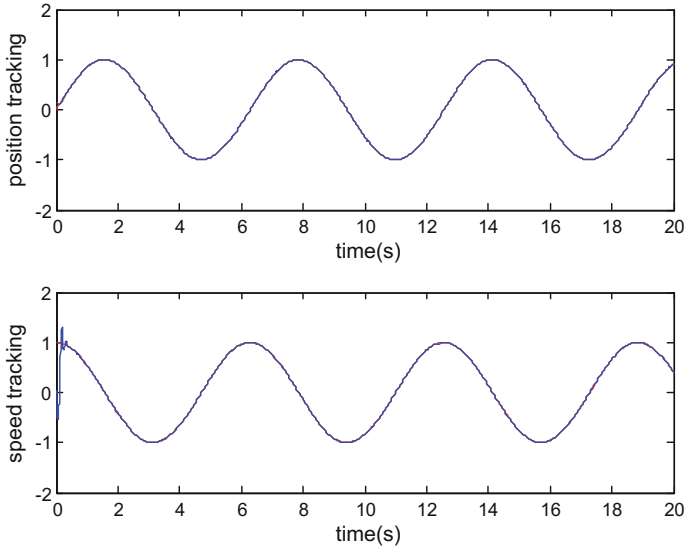


Fig. 6.11 Position and speed tracking

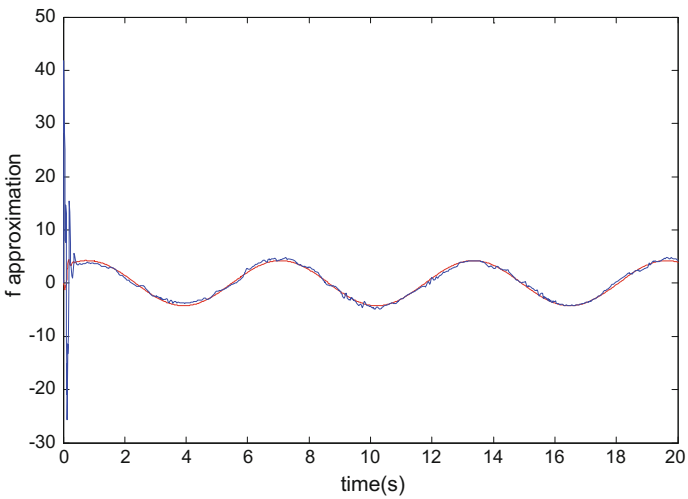


Fig. 6.12 $f(x)$ and $\hat{f}(x)$

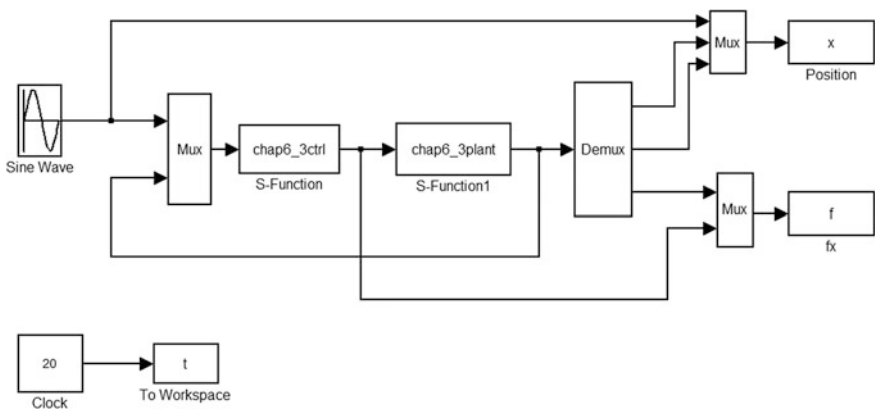
Simulation programs:

(1) **Membership function design: chap6_3mf.m**

```
clear all;
close all;
```

```
L1=-pi/3;
L2=pi/3;
L=L2-L1;

T=L*1/1000;

x=L1:T:L2;
figure(1);
for i=1:1:5
  gs=-[(x+pi/3-(i-1)*pi/6)/(pi/12)].^2;
   u=exp(gs);
   hold on;
   plot(x,u);
end
xlabel('x');ylabel('Membership function degree');
```

(2) **Simulink main program: chap6_3sim.md**



(3) **S function of control law: chap6_3ctrl.m**

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
   sys=mdlDerivatives(t,x,u);
case 3,
   sys=mdlOutputs(t,x,u);
case {2,4,9}
```

```
  sys=[];
otherwise
  error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 25;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 4;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0=[0.1*ones(25,1)];
str=[];
ts=[0 0];
function sys=mdlDerivatives(t,x,u)
xd=sin(t);
dxd=cos(t);

x1=u(2);
x2=u(3);
e=x1-xd;
de=x2-dxd;
c=15;
s=c*e+de;

xi=[x1;x2];

FS1=0;
for l1=1:1:5
  gs1=-[(x1+pi/3-(l1-1)*pi/6)/(pi/12)]^2;
  u1(l1)=exp(gs1);
end

for l2=1:1:5
  gs2=-[(x2+pi/3-(l2-1)*pi/6)/(pi/12)]^2;
  u2(l2)=exp(gs2);
end
for l1=1:1:5
  for l2=1:1:5
    FS2(5*(l1-1)+l2)=u1(l1)*u2(l2);
    FS1=FS1+u1(l1)*u2(l2);
  end
end
FS=FS2/(FS1+0.001);
```

```
for i=1:1:25
  thta(i,1)=x(i);
end
gama=5000;
S=gama*s*FS;

for i=1:1:25
  sys(i)=S(i);
end
function sys=mdlOutputs(t,x,u)
xd=sin(t);
dxd=cos(t);
ddxd=-sin(t);

x1=u(2);
x2=u(3);
e=x1-xd;
de=x2-dxd;
c=15;
s=c*e+de;

xi=[x1;x2];

FS1=0;
for l1=1:1:5
 gs1=-[(x1+pi/3-(l1-1)*pi/6)/(pi/12)]^2;
 u1(l1)=exp(gs1);
end
for l2=1:1:5
 gs2=-[(x2+pi/3-(l2-1)*pi/6)/(pi/12)]^2;
 u2(l2)=exp(gs2);
end
for l1=1:1:5
  for l2=1:1:5
    FS2(5*(l1-1)+l2)=u1(l1)*u2(l2);
    FS1=FS1+u1(l1)*u2(l2);
  end
end
FS=FS2/(FS1+0.001);

for i=1:1:25
  thta(i,1)=x(i);
end
fxp=thta'*FS';
xite=0.50;
ut=-c*de+ddxd-fxp-xite*sign(s);
```

```
sys(1)=ut;
sys(2)=fxp;
```

(4) S function of plant: chap6_3plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
   sys=mdlDerivatives(t,x,u);
case 3,
   sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
   sys = [];
otherwise
   error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 3;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.15;0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
ut=u(1);

f=3*(x(1)+x(2));
sys(1)=x(2);
sys(2)=f+ut;
function sys=mdlOutputs(t,x,u)
f=3*(x(1)+x(2));

sys(1)=x(1);
sys(2)=x(2);
sys(3)=f;
```

(5) **Plot program: chap6_3plot.m**

```
close all;

figure(1);
subplot(211);
plot(t,x(:,1),'r',t,x(:,2),'b');
xlabel('time(s)');ylabel('position tracking');
subplot(212);
plot(t,cos(t),'r',t,x(:,3),'b');
xlabel('time(s)');ylabel('speed tracking');

figure(2);
plot(t,f(:,1),'r',t,f(:,3),'b');
xlabel('time(s)');ylabel('f approximation');
```

6.4 Adaptive Fuzzy Control Based on Fuzzy System Compensator

6.4.1 System Description

A typical manipulator is described as shown in Fig. 6.13.

The dynamic equation with n-joint manipulator can be described as

$$\boldsymbol{D}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) + \boldsymbol{F}(\dot{\boldsymbol{q}}) + \boldsymbol{\tau}_{\mathrm{d}} = \boldsymbol{\tau} \tag{6.20}$$

Fig. 6.13 A 8-joint manipulator

where $\boldsymbol{q} \in \mathrm{R}^n$ is the generalized coordinates; $\boldsymbol{D}(\boldsymbol{q}) \in \mathrm{R}^{n\times n}$ is the symmetric, bounded, positive definite inertia matrix; $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} \in \mathrm{R}^n$ presents the centripetal and Coriolis torques; $\boldsymbol{G}(\boldsymbol{q}) \in \mathrm{R}^n$, $\boldsymbol{F}(\dot{\boldsymbol{q}}) \in \mathrm{R}^n$, $\boldsymbol{\tau}_\mathrm{d} \in \mathrm{R}^n$, and $\boldsymbol{\tau} \in \mathrm{R}^n$ represent the gravitational torques, friction, disturbance, and applied joint torques, respectively.

The dynamic equation with n-joint manipulator is characterized by the following structural properties.

Property 1: $\boldsymbol{D}(\boldsymbol{q})$ is the symmetric, bounded, positive definite inertia matrix; for known positive constant m_1 and m_2, there exists $m_1\boldsymbol{I} \le \boldsymbol{D}(\boldsymbol{q}) \le m_2\boldsymbol{I}$;

Property 2: Using a proper definition of matrix $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$, both $\boldsymbol{D}(\boldsymbol{q})$ and $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ are not independent and satisfy

$$\boldsymbol{x}^\mathrm{T}\left(\dot{\boldsymbol{D}} - 2\boldsymbol{C}\right)\boldsymbol{x} = 0 \tag{6.21}$$

that is, $\boldsymbol{x}^\mathrm{T}\left(\dot{\boldsymbol{D}} - 2\boldsymbol{C}\right)\boldsymbol{x} = 0$ is a skew-symmetric matrix.

This property is simply a statement that the so-called fictitious forces, defined by $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$, do not work on the system. This property is utilized in stability analysis.

Property 3: $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ is limited, that is, for known $c_b(q)$, there exists $\|\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\| \le c_b(\boldsymbol{q})\|\dot{\boldsymbol{q}}\|$;

Property 4: For the unknown disturbance $\boldsymbol{\tau}_\mathrm{d}$, $\|\boldsymbol{\tau}_\mathrm{d}\| \le \tau_\mathrm{M}$, τ_M is a positive constant.

A typical rigid two-joint manipulator is shown in Fig. 6.14.

Just like the Eq. (6.20), we consider that the dynamic equation with n-joint manipulator can be described as

$$\boldsymbol{D}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) + \boldsymbol{F}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}) = \boldsymbol{\tau} \tag{6.22}$$

where $\boldsymbol{F}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})$ consists of friction force $\boldsymbol{F}_\mathrm{r}$, disturbance $\boldsymbol{\tau}_\mathrm{d}$, and uncertainties.

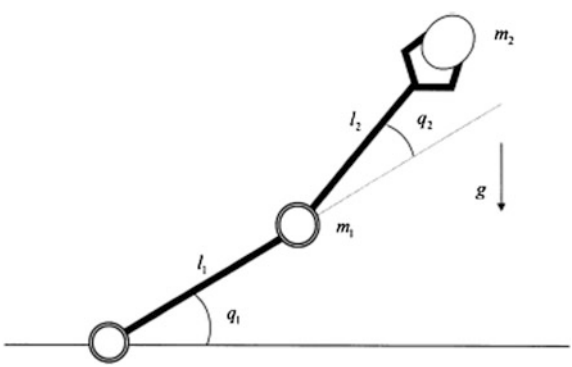


Fig. 6.14 A Two-joint manipulator

6.4.2 Adaptive Fuzzy Control Design and Analysis

Assume $\boldsymbol{D}(\boldsymbol{q})$, $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ and $\boldsymbol{G}(\boldsymbol{q})$ are known, and define sliding mode function as

$$\boldsymbol{s}=\dot{\tilde{\boldsymbol{q}}}+\Lambda\tilde{\boldsymbol{q}} \tag{6.23}$$

where $\boldsymbol{\Lambda}$ is a positive definite matrix, $\tilde{\boldsymbol{q}}(t)$ is the tracking error, $\tilde{\boldsymbol{q}}(t)=\boldsymbol{q}(t)-\boldsymbol{q}_{\mathrm{d}}(t)$, and $\boldsymbol{q}_{\mathrm{d}}(t)$ is the ideal angle.

Define

$$\dot{\boldsymbol{q}}_{\mathrm{r}}(t)=\dot{\boldsymbol{q}}_{\mathrm{d}}(t)-\Lambda\tilde{\boldsymbol{q}}(t) \tag{6.24}$$

Considering $\boldsymbol{F}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})$ as unknown nonlinear function, we design the fuzzy system $\hat{\boldsymbol{F}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}|\boldsymbol{\Theta})$ to approximate $\boldsymbol{F}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})$.

The fuzzy system $\hat{\boldsymbol{F}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}|\boldsymbol{\Theta})$ can be described as

$$\hat{\boldsymbol{F}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}|\boldsymbol{\Theta})=\begin{bmatrix}\hat{F}_1(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}|\boldsymbol{\Theta}_1)\\ \hat{F}_2(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}|\boldsymbol{\Theta}_2)\\ \vdots\\ \hat{F}_n(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}|\boldsymbol{\Theta}_n)\end{bmatrix}=\begin{bmatrix}\boldsymbol{\Theta}_1^{\mathrm{T}}\boldsymbol{\xi}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})\\ \boldsymbol{\Theta}_2^{\mathrm{T}}\boldsymbol{\xi}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})\\ \vdots\\ \boldsymbol{\Theta}_n^{\mathrm{T}}\boldsymbol{\xi}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})\end{bmatrix} \tag{6.25}$$

where $\boldsymbol{\xi}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})$ is fuzzy basis function vector.

Define Lyapunov function as

$$V(t)=\frac{1}{2}\left(\boldsymbol{s}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{s}+\sum_{i=1}^{n}\tilde{\boldsymbol{\Theta}}_i^{\mathrm{T}}\boldsymbol{\Gamma}_i\tilde{\boldsymbol{\Theta}}_i\right) \tag{6.26}$$

where $\tilde{\boldsymbol{\Theta}}_i=\boldsymbol{\Theta}_i^{*}-\boldsymbol{\Theta}_i$, $\boldsymbol{\Theta}_i^{*}$ is ideal weight value, $\boldsymbol{\Gamma}_i>0$.

Since $\boldsymbol{s}=\dot{\tilde{\boldsymbol{q}}}+\Lambda\tilde{\boldsymbol{q}}=\dot{\boldsymbol{q}}-\dot{\boldsymbol{q}}_{\mathrm{d}}+\Lambda\tilde{\boldsymbol{q}}=\dot{\boldsymbol{q}}-\dot{\boldsymbol{q}}_{\mathrm{r}}$,then

$$\boldsymbol{s}=\dot{\tilde{\boldsymbol{q}}}+\Lambda\tilde{\boldsymbol{q}}=\dot{\boldsymbol{q}}-\dot{\boldsymbol{q}}_{\mathrm{d}}+\Lambda\tilde{\boldsymbol{q}}=\dot{\boldsymbol{q}}-\dot{\boldsymbol{q}}_{\mathrm{r}}$$

$$\boldsymbol{D}\dot{\boldsymbol{s}}=\boldsymbol{D}\ddot{\boldsymbol{q}}-\boldsymbol{D}\ddot{\boldsymbol{q}}_{\mathrm{r}}=\boldsymbol{\tau}-\boldsymbol{C}\dot{\boldsymbol{q}}-\boldsymbol{G}-\boldsymbol{F}-\boldsymbol{D}\ddot{\boldsymbol{q}}_{\mathrm{r}}$$

and then,

$$\begin{aligned}\dot{V}(t)&=\boldsymbol{s}^{\mathrm{T}}\boldsymbol{D}\dot{\boldsymbol{s}}+\frac{1}{2}\boldsymbol{s}^{\mathrm{T}}\dot{\boldsymbol{D}}\boldsymbol{s}+\sum_{i=1}^{n}\tilde{\boldsymbol{\Theta}}_i^{\mathrm{T}}\Gamma_i\dot{\tilde{\boldsymbol{\Theta}}}_i\\&=-\boldsymbol{s}^{\mathrm{T}}(-\boldsymbol{\tau}+\boldsymbol{C}\dot{\boldsymbol{q}}+\boldsymbol{G}+\boldsymbol{F}+\boldsymbol{D}\ddot{\boldsymbol{q}}_{\mathrm{r}}-\boldsymbol{C}\boldsymbol{s})+\sum_{i=1}^{n}\tilde{\boldsymbol{\Theta}}_i^{\mathrm{T}}\boldsymbol{\Gamma}_i\dot{\tilde{\boldsymbol{\Theta}}}_i\\&=-\boldsymbol{s}^{\mathrm{T}}(\boldsymbol{D}\ddot{\boldsymbol{q}}_{\mathrm{r}}+\boldsymbol{C}\dot{\boldsymbol{q}}_{\mathrm{r}}+\boldsymbol{G}+\boldsymbol{F}-\boldsymbol{\tau})+\sum_{i=1}^{n}\tilde{\boldsymbol{\Theta}}_i^{\mathrm{T}}\Gamma_i\dot{\tilde{\boldsymbol{\Theta}}}_i\end{aligned} \tag{6.27}$$

To overcome the approximation error, adaptive fuzzy control law with robust term is designed as

$$\boldsymbol{\tau} = \boldsymbol{D}(\boldsymbol{q})\ddot{\boldsymbol{q}}_{\mathrm{r}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}_{\mathrm{r}} + \boldsymbol{G}(\boldsymbol{q}) + \hat{\boldsymbol{F}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}|\boldsymbol{\Theta}) - \boldsymbol{K}_{\mathrm{D}}\boldsymbol{s} - \boldsymbol{W}\,\mathrm{sgn}(\boldsymbol{s}) \tag{6.28}$$

where $\boldsymbol{K}_{\mathrm{D}} = \mathrm{diag}(K_i), \quad K_i > 0,$
$\boldsymbol{W} = \mathrm{diag}[w_{\mathrm{M}_1}, \cdots, w_{\mathrm{M}_n}], w_{\mathrm{M}_i} \geq |\omega_i|, i = 1, 2, \cdots, n.$

The fuzzy approximation error is

$$\boldsymbol{\omega} = \boldsymbol{F}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}) - \hat{\boldsymbol{F}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}|\boldsymbol{\Theta}^*) \tag{6.29}$$

Define the adaptive law as

$$\dot{\boldsymbol{\Theta}}_i = -\Gamma_i^{-1} s_i \boldsymbol{\xi}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}), i = 1,\ 2,\ \cdots,\ n \tag{6.30}$$

Substituting control law (6.28) and adaptive law (6.30) into (6.27), we can get

$$\dot{V}(t) \leq -\boldsymbol{s}^{\mathrm{T}}\boldsymbol{K}_{\mathrm{D}}\boldsymbol{s}$$

From above analysis, we can see that fuzzy system approximation error can be overcome by the robust term $\boldsymbol{W}\mathrm{sgn}(\boldsymbol{s})$.

From $\dot{V}(t) \leq -\boldsymbol{s}^{\mathrm{T}}\boldsymbol{K}_{\mathrm{D}}\boldsymbol{s}$, we have

$$\int_0^t \dot{V}\mathrm{d}t \leq -\lambda_{\min}(\boldsymbol{K}_{\mathrm{D}})\int_0^t \|\boldsymbol{s}\|\mathrm{d}t, \text{i.e.} V(t) - V(0) \leq -\lambda_{\min}(\boldsymbol{K}_D)\int_0^t \|\boldsymbol{s}\|\mathrm{d}t$$

Then V is limited, $\boldsymbol{s}$ and $\tilde{\boldsymbol{\Theta}}_i$ are all limited, from $\dot{\boldsymbol{s}}$ expression, $\dot{\boldsymbol{s}}$ is limited, the $\int_0^\infty \|\boldsymbol{s}\|dt$ is limited. From Barbalat Lemma [3], when $t \to \infty$, we have $\boldsymbol{s} \to 0$, then $\boldsymbol{e} \to 0$, $\dot{\boldsymbol{e}} \to 0$.

Since V is limited as $t \to \infty$, thus $\boldsymbol{\Theta}_i$ is limited. Since when $\dot{V} \equiv 0$, we cannot get $\tilde{\boldsymbol{\Theta}}_i \equiv 0$, $\boldsymbol{\Theta}$ will not converge to $\boldsymbol{\Theta}^*$.

Considering fuzzy system $\hat{\boldsymbol{F}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}|\boldsymbol{\Theta})$, if we select k fuzzy labels on each input variable of the FLS for n-link robot manipulator, the fuzzy compensator needs k^{3n} fuzzy rules [4].

For example, considering two-joint manipulator, we use $\hat{\boldsymbol{F}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}|\boldsymbol{\Theta})$ to approximate $\boldsymbol{F}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})$, then we have $n = 2$,and there are three input variables for each joint; if we design 5 MF for each input variable, the fuzzy rules will be $5^{3\times 2} = 5^6 = 15625$, which will cost much more calculation time.

To solve this problem, therefore, we need to consider the methods to reduce the number of fuzzy rules; one way is to consider the properties of robot dynamics and uncertainties.

6.4.3 Only Consider Friction

If we only consider friction force, then we can get $\boldsymbol{F}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) = \boldsymbol{F}(\dot{\boldsymbol{q}})$. For this condition, we can consider to design one input fuzzy system $\hat{\boldsymbol{F}}(\dot{\boldsymbol{q}}|\boldsymbol{\theta})$ to approximate $\boldsymbol{F}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$.

Then from (6.28), the adaptive robust fuzzy control law becomes

$$\boldsymbol{\tau} = \boldsymbol{D}(\boldsymbol{q})\ddot{\boldsymbol{q}}_{\mathrm{r}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}_{\mathrm{r}} + \boldsymbol{G}(\boldsymbol{q}) + \hat{\boldsymbol{F}}(\dot{\boldsymbol{q}}|\boldsymbol{\theta}) - \boldsymbol{K}_{\mathrm{D}}\boldsymbol{s} - \boldsymbol{W}\mathrm{sgn}(\boldsymbol{s}) \tag{6.31}$$

And the adaptive law (6.30) becomes

$$\dot{\boldsymbol{\theta}}_i = -\boldsymbol{\Gamma}_i^{-1} s_i \boldsymbol{\xi}(\dot{\boldsymbol{q}}), i = 1, 2, \cdots, n \tag{6.32}$$

The fuzzy system is designed as

$$\hat{\boldsymbol{F}}(\dot{\boldsymbol{q}}|\boldsymbol{\theta}) = \begin{bmatrix} \hat{F}_1(\dot{\boldsymbol{q}}_1) \\ \hat{F}_2(\dot{\boldsymbol{q}}_2) \\ \vdots \\ \hat{F}_n(\dot{\boldsymbol{q}}_n) \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}_1^{\mathrm{T}}\boldsymbol{\xi}^1(\dot{\boldsymbol{q}}_1) \\ \boldsymbol{\theta}_2^{\mathrm{T}}\boldsymbol{\xi}^2(\dot{\boldsymbol{q}}_2) \\ \vdots \\ \boldsymbol{\theta}_n^{\mathrm{T}}\boldsymbol{\xi}^n(\dot{\boldsymbol{q}}_n) \end{bmatrix}$$

6.4.4 Simulation Example

Consider a two-joint rigid manipulator dynamic Eq. (6.20) as

$$\begin{pmatrix} D_{11}(q_2) & D_{12}(q_2) \\ D_{21}(q_2) & D_{22}(q_2) \end{pmatrix} \begin{pmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{pmatrix} + \begin{pmatrix} -C_{12}(q_2)\dot{q}_2 & -C_{12}(q_2)(\dot{q}_1 + \dot{q}_2) \\ C_{12}(q_2)\dot{q}_1 & 0 \end{pmatrix} \begin{pmatrix} g_1(q_1 + q_2)g \\ g_2(q_1 + q_2)g \end{pmatrix} + F(q, \dot{q}, \ddot{q}) = \begin{pmatrix} \tau_1 \\ \tau_2 \end{pmatrix}$$

where

$$D_{11}(q_2) = (m_1 + m_2)r_1^2 + m_2 r_2^2 + 2m_2 r_1 r_2 \cos(q_2)$$

$$D_{12}(q_2) = D_{21}(q_2) = m_2 r_2^2 + m_2 r_1 r_2 \cos(q_2)$$

$$D_{22}(q_2) = m_2 r_2^2$$

$$C_{12}(q_2) = m_2 r_1 r_2 \sin(q_2)$$

Let $\boldsymbol{y} = [q_1, q_2]^{\mathrm{T}}$, $\boldsymbol{\tau} = [\tau_1, \tau_2]^{\mathrm{T}}$, $\boldsymbol{q} = [q_1\ \dot{q}_1\ q_2\ \dot{q}_2]^{\mathrm{T}}$, choose $r_1 = 1.0$, $r_2 = 0.8$, $m_1 = 1.0$, $m_2 = 1.5$.

Consider ideal trajectory as $y_{\mathrm{d}1} = 0.3\sin t$ and $y_{\mathrm{d}2} = 0.3\sin t$. Define member function as

$$\mu_{A_i^l}(x_i) = \exp\left(-\left(\frac{x_i - \bar{x}_i^l}{\pi/24}\right)^2\right)$$

where $i = 1, 2, 3, 4, 5$, $\bar{x}_i^l$ is chosen as $-\pi/6$, $-\pi/12$,0, $\pi/12$,and $\pi/6$, respectively, and A_i is designed as NB, NS, ZO, PS, PB.

Choose parameters of the control law as $\lambda_1 = 10$, $\lambda_2 = 10$, $\boldsymbol{K}_{\mathrm{D}} = 20\boldsymbol{I}$, $\Gamma_1 = \Gamma_2 = 0.0001$, the initial states of the plant are chosen as $q_1(0) = q_2(0) = \dot{q}_1(0) = \dot{q}_2(0) = 0$,the friction model is $\boldsymbol{F}(\dot{\boldsymbol{q}}) = \begin{bmatrix} 10\dot{q}_1 + 3\mathrm{sgn}(\dot{q}_1) \\ 10\dot{q}_2 + 3\mathrm{sgn}(\dot{q}_2) \end{bmatrix}$, and the disturbance is $\boldsymbol{\tau}_{\mathrm{d}} = \begin{bmatrix} 0.05\ \sin(20t) \\ 0.1\ \sin(20t) \end{bmatrix}$.

Using the robust adaptive control law (6.31) with the adaptive law (6.32), in the fuzzy system, the inputs are chosen as $[\dot{q}_1\ \dot{q}_2]$. Choose $\boldsymbol{\Theta}_i(0) = 0.10$, and let $\boldsymbol{W} = \mathrm{diag}[1.5, 1.5]$; the simulation results are shown in Figs. 6.15, 6.16, 6.17, and 6.18.

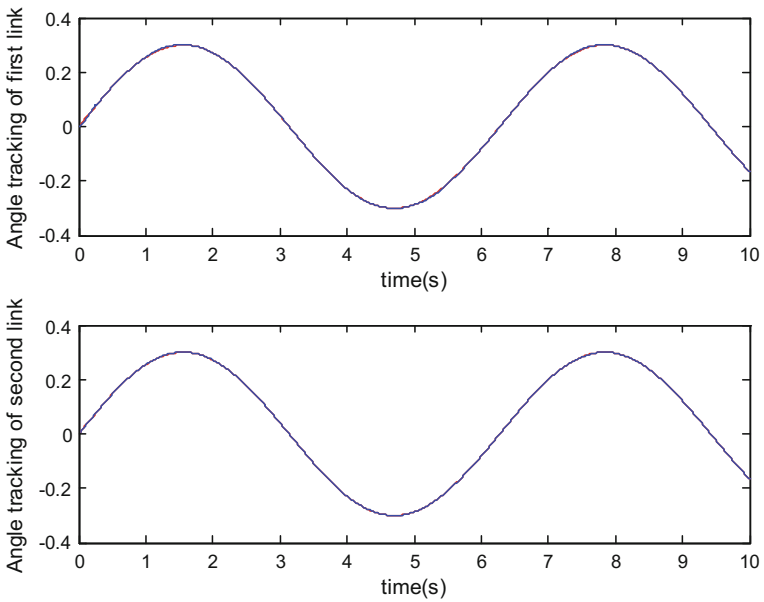


Fig. 6.15 Angle tracking

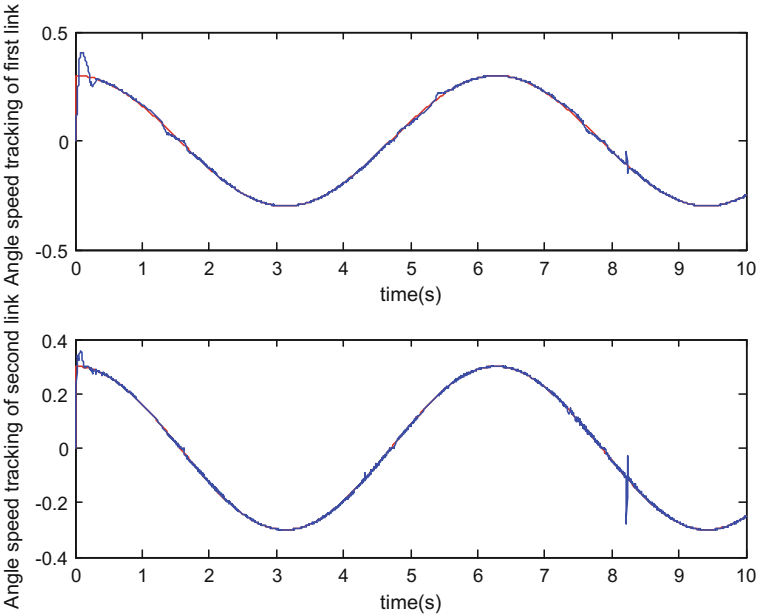


Fig. 6.16 Angle speed tracking

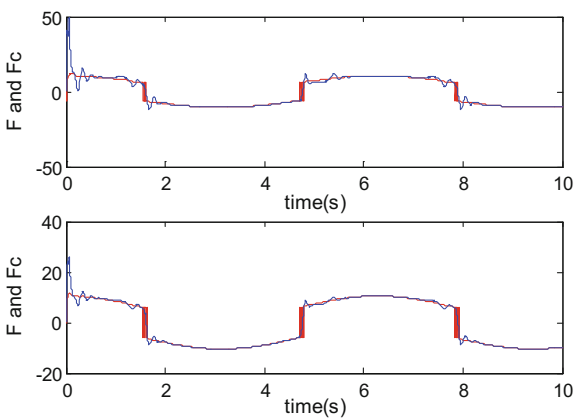
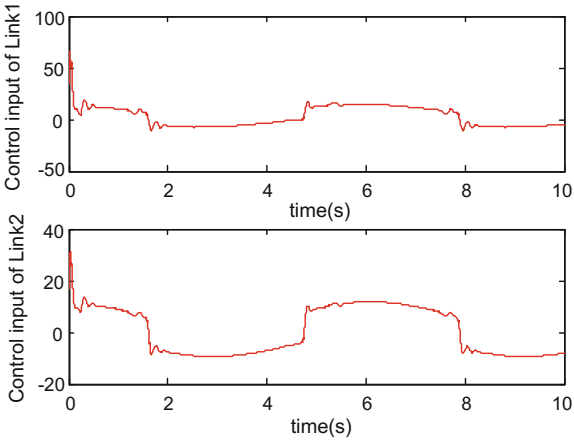


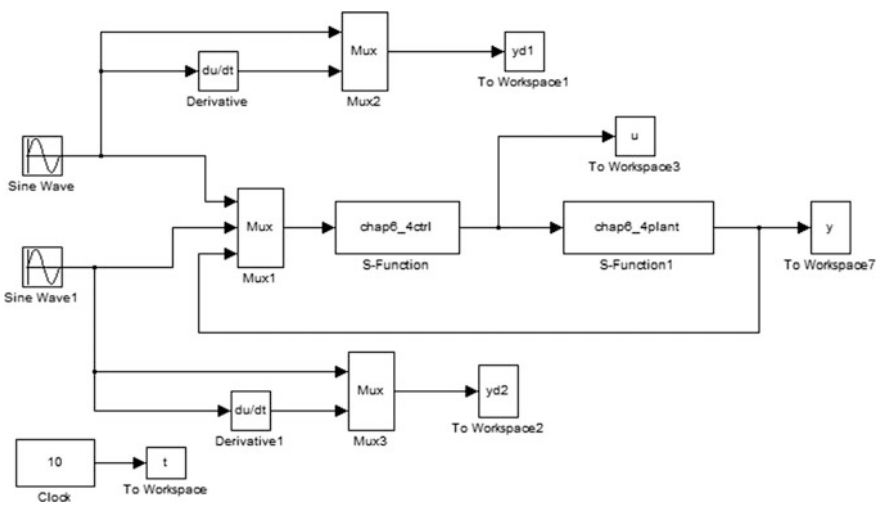
Fig. 6.17 Friction force and compensation

Fig. 6.18 Control input



Simulation programs:

(1) **Simulink main program: chap6_4sim.mdl**



(2) **S function of Control law: chap6_4ctrl.m**

```
function [sys,x0,str,ts] = MIMO_ctrl(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
```

```
case 1,
  sys=mdlDerivatives(t,x,u);
case 3,
  sys=mdlOutputs(t,x,u);
case {2,4,9}
  sys=[];
otherwise
  error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
global nmn1 nmn2 Fai
nmn1=10;nmn2=10;
Fai=[nmn1 0;0 nmn2];
sizes = simsizes;
sizes.NumContStates  = 10;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 4;
sizes.NumInputs      = 8;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = [0.1*ones(10,1)];
str = [];
ts  = [];
function sys=mdlDerivatives(t,x,u)
global nmn1 nmn2 Fai
qd1=u(1);
qd2=u(2);
dqd1=0.3*cos(t);
dqd2=0.3*cos(t);
dqd=[dqd1 dqd2]';

ddqd1=-0.3*sin(t);
ddqd2=-0.3*sin(t);
ddqd=[ddqd1 ddqd2]';

q1=u(3);dq1=u(4);
q2=u(5);dq2=u(6);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fsd1=0;
for l1=1:1:5
  gs1=-[(dq1+pi/6-(l1-1)*pi/12)/(pi/24)]^2;
  u1(l1)=exp(gs1);
end
fsd2=0;
```

```
for l2=1:1:5
  gs2=-[(dq2+pi/6-(l2-1)*pi/12)/(pi/24)]^2;
  u2(l2)=exp(gs2);
end
for l1=1:1:5
  fsu1(l1)=u1(l1);
  fsd1=fsd1+u1(l1);
end
for l2=1:1:5
  fsu2(l2)=u2(l2);
  fsd2=fsd2+u2(l2);
end
fs1=fsu1/(fsd1+0.001);
fs2=fsu2/(fsd2+0.001);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
e1=q1-qd1;
e2=q2-qd2;
e=[e1 e2]';
de1=dq1-dqd1;
de2=dq2-dqd2;
de=[de1 de2]';

s=de+Fai*e;
Gama1=0.0001;Gama2=0.0001;

S1=-1/Gama1*s(1)*fs1;
S2=-1/Gama2*s(2)*fs2;
for i=1:1:5
  sys(i)=S1(i);
end
for j=6:1:10
  sys(j)=S2(j-5);
end

function sys=mdlOutputs(t,x,u)
global nmn1 nmn2 Fai
q1=u(3);dq1=u(4);
q2=u(5);dq2=u(6);

r1=1;r2=0.8;
m1=1;m2=1.5;

D11=(m1+m2)*r1^2+m2*r2^2+2*m2*r1*r2*cos(q2);
D22=m2*r2^2;
D21=m2*r2^2+m2*r1*r2*cos(q2);
D12=D21;
```

```
D=[D11 D12;D21 D22];

C12=m2*r1*sin(q2);
C=[-C12*dq2 -C12*(dq1+dq2);C12*q1 0];

g1=(m1+m2)*r1*cos(q2)+m2*r2*cos(q1+q2);
g2=m2*r2*cos(q1+q2);
G=[g1;g2];

qd1=u(1);
qd2=u(2);
dqd1=0.3*cos(t);
dqd2=0.3*cos(t);
dqd=[dqd1 dqd2]';

ddqd1=-0.3*sin(t);
ddqd2=-0.3*sin(t);
ddqd=[ddqd1 ddqd2]';

e1=q1-qd1;
e2=q2-qd2;
e=[e1 e2]';
de1=dq1-dqd1;
de2=dq2-dqd2;
de=[de1 de2]';

s=de+Fai*e;

dqr=dqd-Fai*e;
ddqr=ddqd-Fai*de;

for i=1:1:5
  thta1(i,1)=x(i);
end
for i=1:1:5
  thta2(i,1)=x(i+5);
end

fsd1=0;
for l1=1:1:5
  gs1=-[(dq1+pi/6-(l1-1)*pi/12)/(pi/24)]^2;
  u1(l1)=exp(gs1);
end
fsd2=0;
for l2=1:1:5
  gs2=-[(dq2+pi/6-(l2-1)*pi/12)/(pi/24)]^2;
  u2(l2)=exp(gs2);
end
```

```
for l1=1:1:5
   fsu1(l1)=u1(l1);
   fsd1=fsd1+u1(l1);
end
for l2=1:1:5
   fsu2(l2)=u2(l2);
   fsd2=fsd2+u2(l2);
end
fs1=fsu1/(fsd1+0.001);
fs2=fsu2/(fsd2+0.001);

Fp(1)=thta1'*fs1';
Fp(2)=thta2'*fs2';

KD=20*eye(2);
W=[1.5 0;0 1.5];

tol=D*ddqr+C*dqr+G+1*Fp'-KD*s-W*sign(s);  %(4.134)

sys(1)=tol(1);
sys(2)=tol(2);
sys(3)=Fp(1);
sys(4)=Fp(2);
```

(3) S function of Plant: chap6_4plant.m

```
function [sys,x0,str,ts]=MIMO_plant(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
   sys=mdlDerivatives(t,x,u);
case 3,
   sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
   sys = [];
otherwise
   error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 4;
sizes.NumDiscStates  = 0;
```

```
sizes.NumOutputs   = 6;
sizes.NumInputs    = 4;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0 0 0 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
r1=1;r2=0.8;
m1=1;m2=1.5;

D11=(m1+m2)*r1^2+m2*r2^2+2*m2*r1*r2*cos(x(3));
D22=m2*r2^2;
D21=m2*r2^2+m2*r1*r2*cos(x(3));
D12=D21;
D=[D11 D12;D21 D22];

C12=m2*r1*sin(x(3));
C=[-C12*x(4) -C12*(x(2)+x(4));C12*x(1) 0];

g1=(m1+m2)*r1*cos(x(3))+m2*r2*cos(x(1)+x(3));
g2=m2*r2*cos(x(1)+x(3));
G=[g1;g2];

Fr=[10*x(2)+3*sign(x(2));10*x(4)+3*sign(x(4))];
told=[0.05*sin(20*t);0.1*sin(20*t)];

tol=[u(1) u(2)]';
S=inv(D)*(tol-C*[x(2);x(4)]-G-Fr);

sys(1)=x(2);
sys(2)=S(1);
sys(3)=x(4);
sys(4)=S(2);
function sys=mdlOutputs(t,x,u)
Fr=[10*x(2)+3*sign(x(2));10*x(4)+3*sign(x(4))];

sys(1)=x(1);
sys(2)=x(2);
sys(3)=x(3);
sys(4)=x(4);
sys(5)=Fr(1);
sys(6)=Fr(2);
```

(4) **Plot program: chap6_4plot.m**

```
close all;

figure(1);
subplot(211);
plot(t,yd1(:,1),'r',t,y(:,1),'b');
xlabel('time(s)');ylabel('Angle tracking of first link');
subplot(212);
plot(t,yd2(:,1),'r',t,y(:,3),'b');
xlabel('time(s)');ylabel('Angle tracking of second link');

figure(2);
subplot(211);
plot(t,yd1(:,2),'r',t,y(:,2),'b');
xlabel('time(s)');ylabel('Angle speed tracking of first link');
subplot(212);
plot(t,yd2(:,2),'r',t,y(:,4),'b');
xlabel('time(s)');ylabel('Angle speed tracking of second link');

figure(3);
subplot(211);
plot(t,y(:,5),'r',t,u(:,3),'b');
xlabel('time(s)');ylabel('F and Fc');
subplot(212);
plot(t,y(:,6),'r',t,u(:,4),'b');
xlabel('time(s)');ylabel('F and Fc');

figure(4);
subplot(211);
plot(t,u(:,1),'r');
xlabel('time(s)');ylabel('Control input of Link1');
subplot(212);
plot(t,u(:,2),'r');
xlabel('time(s)');ylabel('Control input of Link2');
```

References

1. L.X.Wang, *A Course in Fuzzy Systems and Control*, (Prentice-Hall International, Inc., 1996)
2. L.X. Wang, Stable adaptive fuzzy control of nonlinear systems. IEEE Trans. Fuzzy Syst. **1**(2), 146–155 (1993)
3. P.A. Ioannou, J. Sun, *Robust Adaptive Control*, (PTR Prentice-Hall, 1996), pp. 75–76
4. B.K. Yoo, W.C. Ham, Adaptive control of robot manipulator using fuzzy compensator. IEEE Trans. Fuzzy Syst. **8**(2), 186–199 (2000)

Chapter 7
Neural Networks

7.1 Introduction

Neural networks are networks of nerve cells (neurons) in the brain. The human brain has billions of individual neurons and trillions of interconnections. Neurons are continuously processing and transmitting information to one another.

In 1909, Cajal found that the brain consists of a large number of highly connected neurons which apparently can send very simple excitatory and inhibitory messages to each other and can update their excitations on the basis of these simple messages [1]. A neuron has three major regions: the cell body, the axon (send out messages), and the dendrites (receive messages). The cell body provides the support functions, the structure of the cell. The axon is a branching fiber which carries signals away from the neurons. The dendrites consist of more branching fibers which receive signals from other nerve cells.

The historical reviews of neural networks are as follows:

(1) In 1943, McCulloch and Pitts proposed first mathematical model of the neurons and showed how neuron-like networks could be computed.
(2) The first set of ideas of learning in neural networks was contained in Hebb's book entitled The Organization of Behaviour in 1949.
(3) In 1951, Edmonds and Minsky built their learning machine using Hebb's idea.
(4) The real beginning of a meaningful neuron-like network learning can be traced to the work of Rosenblatt in 1962. Rosenblatt invented a class of simple neuron-like learning networks which is called perceptron neural network.
(5) In a breakthrough paper published in 1982, Hopfield introduced a neural network architecture which is called Hopfield network. This NN can be used to solve optimization problems such as the traveling salesman problem.
(6) An important NN which has been widely used in NN is the back-error propagation or backpropagation (BP). BP NN was first presented in 1974 by Werbos and then was independently reinvented in 1986 by Rumelhart et al. [2].

Their book, Parallel Distributed Processing, introduced a broad perspective of the neural network approaches.

(7) RBF neural networks were addressed in 1988 [3], which have recently drawn much attention due to their good generalization ability and a simple network structure that avoids unnecessary and lengthy calculation as compared to the multilayer feed-forward network (MFN). Past research of universal approximation theorems on RBF have shown that any nonlinear function over a compact set with arbitrary accuracy can be approximated by RBF neural network [4]. There have been significant research efforts on RBF neural control for nonlinear systems.

RBF neural network has three layers: the input layer, the hidden layer, and the output layer. Neurons at the hidden layer are activated by a radial basis function. The hidden layer consists of an array of computing units called hidden nodes. Each hidden node contains a center $\boldsymbol{c}$ vector that is a parameter vector of the same dimension as the input vector $\boldsymbol{x}$, the Euclidean distance between the center and the network input vector $\boldsymbol{x}$ is defined by $\|x(t) - c_j(t)\|$.

7.2 Single Neural Network

From Fig. 7.1, the algorithm of single neural network can be described as

$$Net_i = \sum_j w_{ij} x_j + s_i - \theta_i \tag{7.1}$$

$$u_i = f(Net_i) \tag{7.2}$$

$$y_i = g(u_i) = h(Net_i) \tag{7.3}$$

where $g(u_i) = u_i$, $y_i = f(Net_i)$.

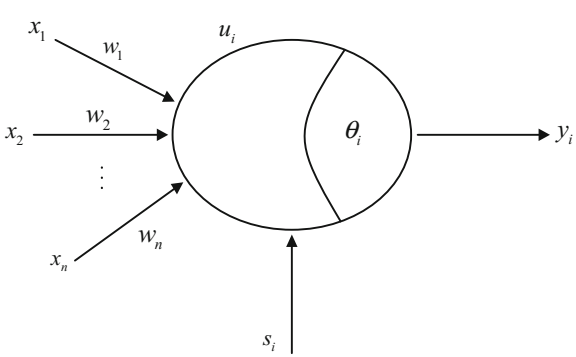


Fig. 7.1 Single NN model

Nonlinearity characteristic function $f(Net_i)$ can be divided as three kinds as follows:

(1) Threshold value

$$f(Net_i) = \begin{cases} 1 & Net_i > 0 \\ 0 & Net_i \leq 0 \end{cases} \tag{7.4}$$

The threshold function is shown in Fig. 7.2.

(2) Linear function

$$f(Net_i) = \begin{cases} 0 & Net_i \leq Net_{i0} \\ kNet_i & Net_{i0} < Net_i < Net_{i1} \\ f_{\max} & Net_i \geq Net_{i1} \end{cases} \tag{7.5}$$

Choose $Net_{i0} = 30$, $Net_{i1} = 70$, $f_{\max} = 5.0$, the linearity function is shown in Fig. 7.3.

(3) Nonlinear function

Sigmoid function and Gaussian function are often used in neural network. Sigmoid type is expressed as

$$f(Net_i) = \frac{1}{1 + e^{-\frac{Net_i}{T}}} \tag{7.6}$$

Choose $T = 1.0$, the sigmoid function is shown in Fig. 7.4.

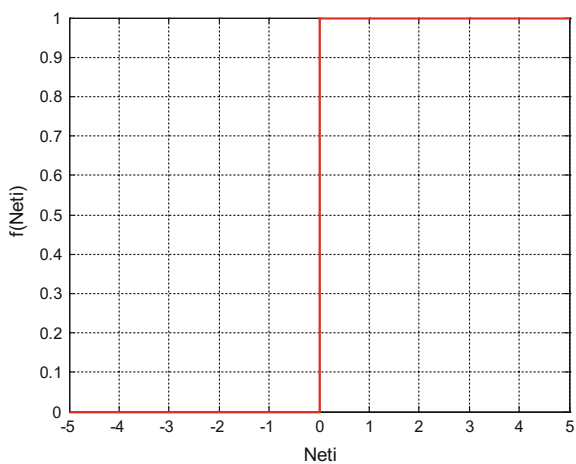


Fig. 7.2 Threshold function

Fig. 7.3 Linearity function

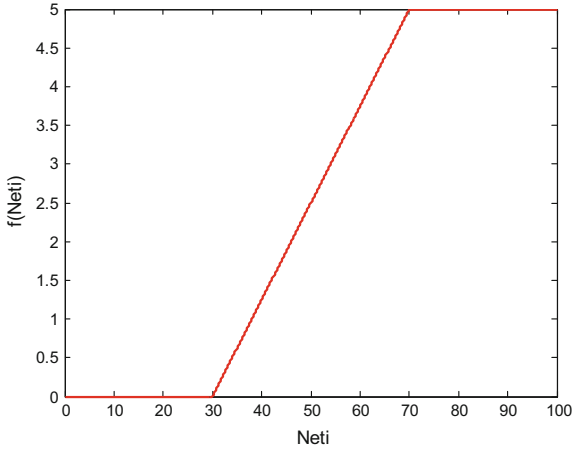
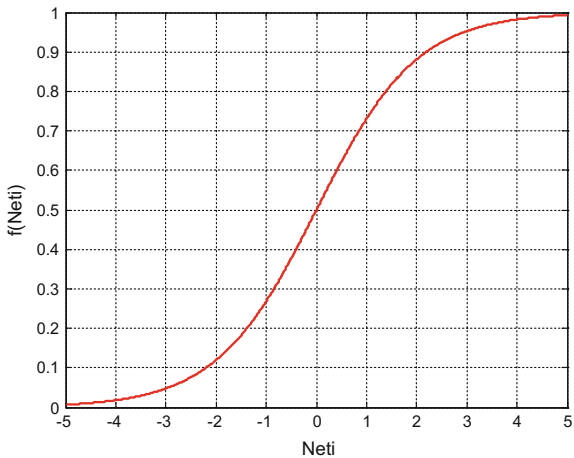


Fig. 7.4 Sigmoid function



7.3 BP Neural Network Design and Simulation

The backpropagation (BP) neural network is a multilayered neural network. Thus, the BP algorithm employs three or more layers of processing unit (neurons).

7.3.1 BP Network Structure

Figure 7.5 shows a structure of a typical three-layered network for the BP algorithm. The leftmost layer of units is the input layer to which the input data is supplied. The layer after it is the hidden layer where the processing units are

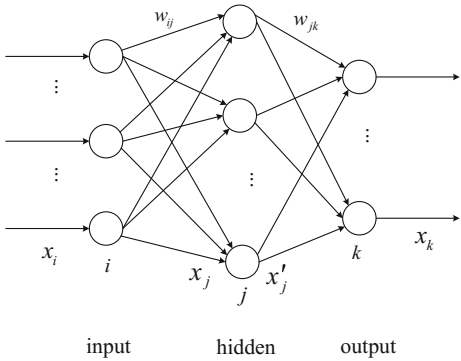


Fig. 7.5 BP NN structure

interconnected to the layers before and after it. The rightmost layer is the output layer. The layers shown in Fig. 7.5 are fully interconnected, which means that each processing unit is connected to every unit in the previous layer and in the succeeding layer. However, units are not connected to other units in the same layer.

7.3.2 *Approximation of BP Neural Network*

BP neural network scheme for approximation is shown in Fig. 7.6.

BP neural network structure for approximation is shown in Fig. 7.7.

Classical BP neural network algorithm is described as follows:

(1) Feed-forward calculation

Input of hidden layer is

$$x_j = \sum_i w_{ij} x_i \tag{7.7}$$

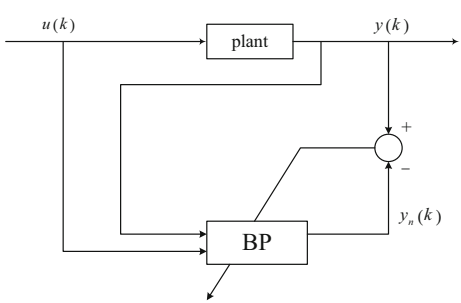


Fig. 7.6 BP approximation scheme

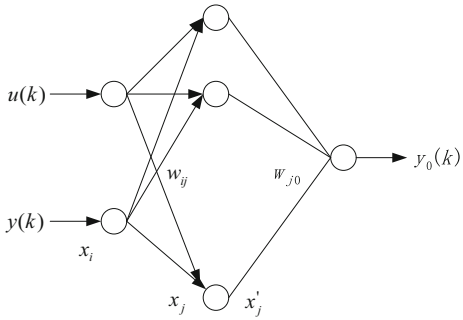


Fig. 7.7 BP neural network structure for approximation

Output of hidden layer is

$$x_j' = f(x_j) = \frac{1}{1 + e^{-x_j}} \tag{7.8}$$

then

$$\frac{\partial x_j'}{\partial x_j} = x_j'(1 - x_j')$$

Output of output layer is

$$y_{\mathrm{o}}(k) = \sum_j w_{j\mathrm{o}} x_j' \tag{7.9}$$

Then, the approximation error is

$$e(k) = y(k) - y_n(k)$$

Error index function is designed as

$$E = \frac{1}{2} e(k)^2 \tag{7.10}$$

(2) Learning algorithm of BP

According to the steepest descent (gradient) method, the learning of weight value w_{jo} is

$$\Delta w_{j\mathrm{o}} = -\eta \frac{\partial E}{\partial w_{j\mathrm{o}}} = \eta \cdot e(k) \cdot \frac{\partial y_{\mathrm{o}}}{\partial w_{j\mathrm{o}}} = \eta \cdot e(k) \cdot x_j'$$

The weight value at time $k+1$ is

$$w_{jo}(k+1) = w_{jo}(k) + \Delta w_{jo}$$

The learning of weight value w_{ij} is

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta \cdot e(k) \cdot \frac{\partial y_o}{\partial w_{ij}}$$

where the chain rule is used, $\frac{\partial y_o}{\partial w_{ij}} = \frac{\partial y_o}{\partial x_j'} \cdot \frac{\partial x_j'}{\partial x_j} \cdot \frac{\partial x_j}{\partial w_{ij}} = w_{jo} \cdot \frac{\partial x_j'}{\partial x_j} \cdot x_i = w_{jo} \cdot x_j'(1 - x_j') \cdot x_i$.

The weight value at time $k+1$ is

$$w_{ij}(k+1) = w_{ij}(k) + \Delta w_{ij}$$

Considering the effect of previous weight value change, the algorithm of weight value is

$$w_{jo}(k+1) = w_{jo}(k) + \Delta w_{jo} + \alpha\big(w_{jo}(k) - w_{jo}(k-1)\big) \tag{7.11}$$

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij} + \alpha(w_{ij}(t) - w_{ij}(t-1)) \tag{7.12}$$

where η is learning rate, α is momentum factor, $\eta \in [0, 1]$, $\alpha \in [0, 1]$.

By using BP neural network approximation, Jacobian value can be calculated as follows:

$$\frac{\partial y(k)}{\partial u(k)} \approx \frac{\partial y_o(k)}{\partial u(k)} = \frac{\partial y_o(k)}{\partial x_j'} \times \frac{\partial x_j'}{\partial x_j} \times \frac{\partial x_j}{\partial x(1)} = \sum_j w_{jo} x_j' \Big(1 - x_j'\Big) w_{1j} \tag{7.13}$$

7.3.3 Simulation Example

The plant is as follows

$$y(k) = u(k)^3 + \frac{y(k-1)}{1 + y(k-1)^2}$$

Input signal is chosen as $u(k) = 0.5\sin(6\pi t)$, let RBF neural network input vector as $\boldsymbol{x} = [\,u(k)\ y(k)\,]$, NN structure is chosen as 2-6-1, the initial value of $\boldsymbol{W}_{jo}$, $\boldsymbol{W}_{ij}$ is chosen as random value in $[\,-1 \quad +1\,]$, $\eta = 0.50$, $\alpha = 0.05$.

The program is chap7_1.m, and the results are shown from Figs. 7.8, 7.9, and 7.10.

Fig. 7.8 BP approximation

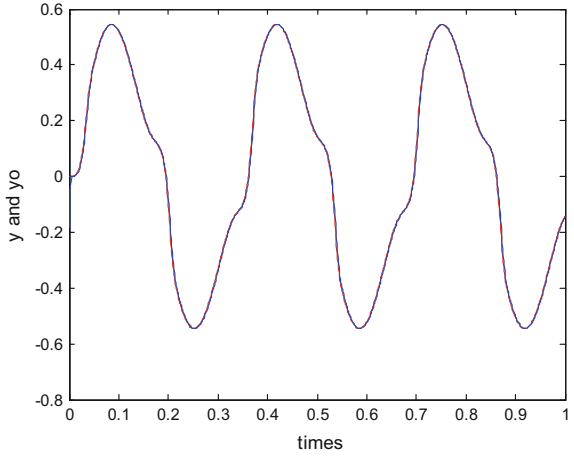


Fig. 7.9 BP approximation error

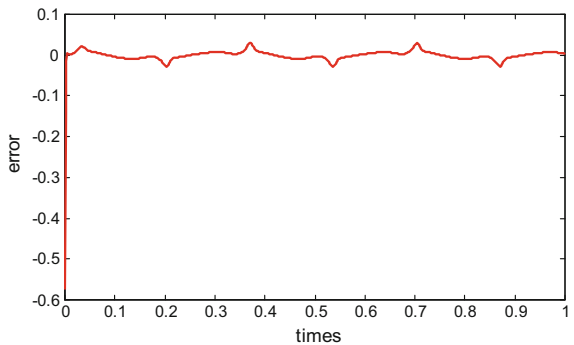
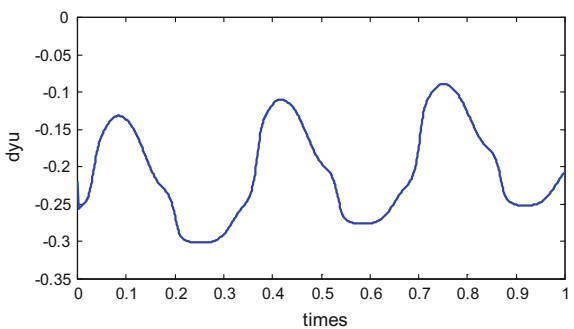


Fig. 7.10 Jacobian value identification



Simulation program: chap7_1.m

```
%BP approximation
clear all;
close all;

xite=0.50;
alfa=0.05;

wjo=rands(6,1);
wjo_1=wjo;wjo_2=wjo_1;

wij=rands(2,6);
wij_1=wij;wij_2=wij;

dwij=0*wij;

x=[0,0]';

u_1=0;
y_1=0;

I=[0,0,0,0,0,0]';
Iout=[0,0,0,0,0,0]';
FI=[0,0,0,0,0,0]';

ts=0.001;
for k=1:1:1000

time(k)=k*ts;
u(k)=0.50*sin(3*2*pi*k*ts);
y(k)=u_1^3+y_1/(1+y_1^2);

x(1)=u(k);
x(2)=y(k);

for j=1:1:6
   I(j)=x'*wij(:,j);
   Iout(j)=1/(1+exp(-I(j)));
end

yo(k)=wjo'*Iout;      % Output of NNI networks

e(k)=y(k)-yo(k);   % Error calculation

wjo=wjo_1+(xite*e(k))*Iout+alfa*(wjo_1-wjo_2);
```

```
for j=1:1:6
  FI(j)=exp(-I(j))/(1+exp(-I(j)))^2;
end

for i=1:1:2
  for j=1:1:6
    dwij(i,j)=e(k)*xite*FI(j)*wjo(j)*x(i);
  end
End
wij=wij_1+dwij+alfa*(wij_1-wij_2);

%%%%%%%%%%%%%%Jacobian%%%%%%%%%%%%%%%%%
yu=0;
for j=1:1:6
  yu=yu+wjo(j)*wij(1,j)*FI(j);
end
dyu(k)=yu;

wij_2=wij_1;wij_1=wij;
wjo_2=wjo_1;wjo_1=wjo;
u_1=u(k);
y_1=y(k);
end
figure(1);
plot(time,y,'r',time,yo,'b');
xlabel('times');ylabel('y and yo');
figure(2);
plot(time,y-yo,'r');
xlabel('times');ylabel('error');
figure(3);
plot(time,dyu);
xlabel('times');ylabel('dyu');
```

7.4 RBF Neural Network Design and Simulation

The radial basis function (RBF) neural network is a multilayered neural network. Like BP neural network structure, RBF algorithm also employs three layers of processing unit (neurons).

The difference between BP and RBF is that RBF have only output layer, and activation function is Gaussian function instead of S function in hidden layer, which will simplify the algorithm and decrease computational burden.

7.4.1 RBF Algorithm

The structure of a typical three-layer RBF neural network is shown as Fig. 7.11.

In RBF neural network, $\boldsymbol{x} = [x_i]^{\mathrm{T}}$ is input vector. Assuming there are mth neural nets, and radial basis function vector in hidden layer of RBF is $\boldsymbol{h} = [h_j]^{\mathrm{T}}$, h_j is Gaussian function value for neural net j in hidden layer, and

$$h_{\mathrm{j}} = \exp\left(-\frac{\|\boldsymbol{x} - c_j\|^2}{2b_j^2}\right) \tag{7.14}$$

where $\boldsymbol{c} = [c_{ij}] = \begin{bmatrix} c_{11} & \cdots & c_{1m} \\ \vdots & \ldots & \vdots \\ c_{n1} & \cdots & c_{nm} \end{bmatrix}$ represents the coordinate value of center point of the Gaussian function of neural net j for the ith input, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$. For the vector $\boldsymbol{b} = [b_1, \ldots, b_m]^{\mathrm{T}}$, b_j represents the width value of Gaussian function for neural net j.

The weight value of RBF is

$$\boldsymbol{w} = [w_1, \ldots, w_m]^{\mathrm{T}} \tag{7.15}$$

The output of RBF neural network is

$$y(t) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{h} = w_1h_1 + w_2h_2 + \cdots + w_mh_m \tag{7.16}$$

7.4.2 RBF Design Example with MATLAB Simulation

7.4.2.1 For Structure 1-5-1 RBF Neural Network

Consider a structure 1-5-1 RBF neural network, we have one input as $x = x_1$, and $\boldsymbol{b} = [\,b_1 \quad b_2 \quad b_3 \quad b_4 \quad b_5\,]^{\mathrm{T}}$, $\boldsymbol{c} = [\,c_{11} \quad c_{12} \quad c_{13} \quad c_{14} \quad c_{15}\,]$, $\boldsymbol{h} = [\,h_1 \quad h_2 \quad h_3 \quad h_4 \quad h_5\,]^{\mathrm{T}}$, $\boldsymbol{w} = [\,w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5\,]$, and $y(t) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{h} = w_1h_1 + w_2h_2 + w_3h_3 + w_4h_4 + w_5h_5$.

Choose the input as $\sin t$, the output of RBF is shown in Fig. 7.12, the output of hidden neural net is shown in Fig. 7.13.

The Simulink program of this example is chap7_2sim.mdl, and MATLAB programs of the example are given in the Appendix.

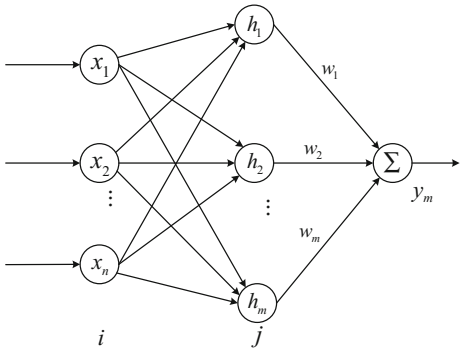


Fig. 7.11 RBF neural network structure

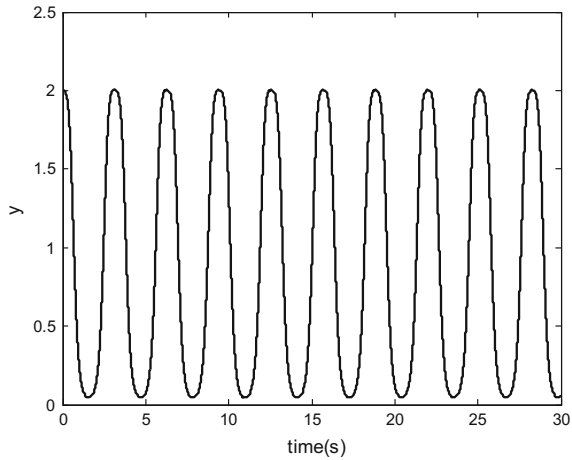


Fig. 7.12 Output of RBF

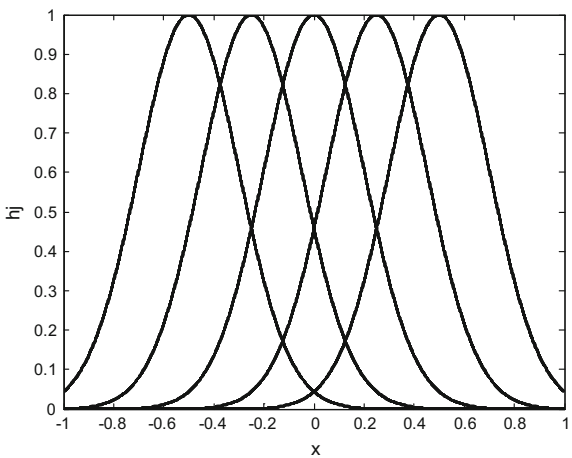
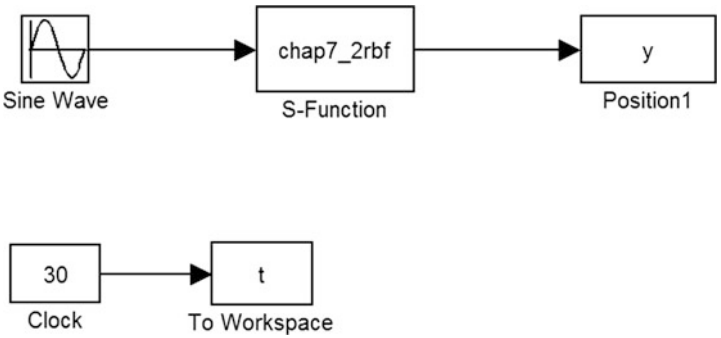


Fig. 7.13 Output of hidden neural net

Simulation programs:

(1) **Simulink main program: chap7_2sim.mdl**



(2) **S function of RBF: chap7_2rbf.m**

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
   sys=mdlOutputs(t,x,u);
case {2,4,9}
   sys=[];
otherwise
   error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 7;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [];
function sys=mdlOutputs(t,x,u)
x=u(1);  %Input Layer
```

```
%i=1
%j=1,2,3,4,5
%k=1
c=[-0.5 -0.25 0 0.25 0.5]; %cij
b=[0.2 0.2 0.2 0.2 0.2]';  %bj

W=ones(5,1);  %Wj
h=zeros(5,1); %hj
for j=1:1:5
  h(j)=exp(-norm(x-c(:,j))^2/(2*b(j)*b(j))); %Hidden Layer
end
y=W'*h;   %Output Layer

sys(1)=y;
sys(2)=x;
sys(3)=h(1);
sys(4)=h(2);
sys(5)=h(3);
sys(6)=h(4);
sys(7)=h(5);
```

(3) Plot program: chap7_2plot.m

```
close all;

% y=y(:,1);
% x=y(:,2);
% h1=y(:,3);
% h2=y(:,4);
% h3=y(:,5);
% h4=y(:,6);
% h5=y(:,7);

figure(1);
plot(t,y(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('y');

figure(2);
plot(y(:,2),y(:,3),'k','linewidth',2);
xlabel('x');ylabel('hj');
hold on;
plot(y(:,2),y(:,4),'k','linewidth',2);
hold on;
plot(y(:,2),y(:,5),'k','linewidth',2);
hold on;
```

```
plot(y(:,2),y(:,6),'k','linewidth',2);
hold on;
plot(y(:,2),y(:,7),'k','linewidth',2);
```

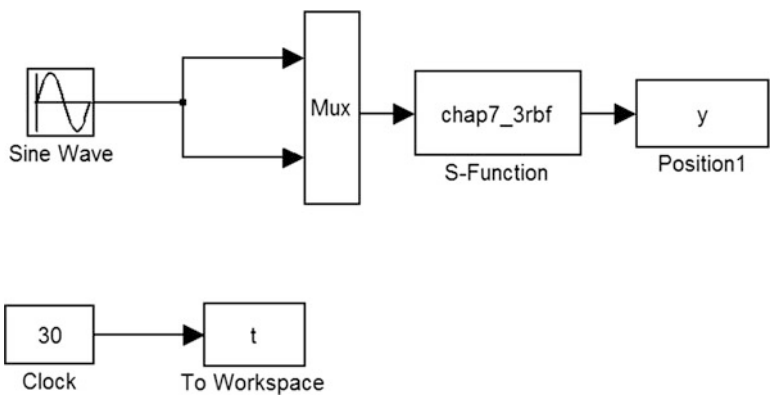
7.4.2.2 For Structure 2-5-1 RBF Neural Network

Consider a structure 2-5-1 RBF neural network, we have $\mathbf{x} = [x_1, x_2]^{\mathrm{T}}$, $\mathbf{b} = [b_1 \quad b_2 \quad b_3 \quad b_4 \quad b_5]^{\mathrm{T}}$, $\mathbf{c} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \end{bmatrix}$, $\mathbf{h} = [h_1 \quad h_2 \quad h_3 \quad h_4 \quad h_5]^{\mathrm{T}}$, $\boldsymbol{w} = [w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5]^{\mathrm{T}}$, and $y(t) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{h} = w_1h_1 + w_2h_2 + w_3h_3 + w_4h_4 + w_5h_5$.

Two inputs are chosen as $\sin t$, the output of RBF is shown in Fig. 7.14, and the output of hidden neural net is shown in Figs. 7.15 and 7.16.

Simulation programs:

(1) **Simulink main program: chap7_3sim.mdl**



(2) **S function of RBF: chap7_3rbf.m**

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
   sys=mdlOutputs(t,x,u);
case {2,4,9}
   sys=[];
```

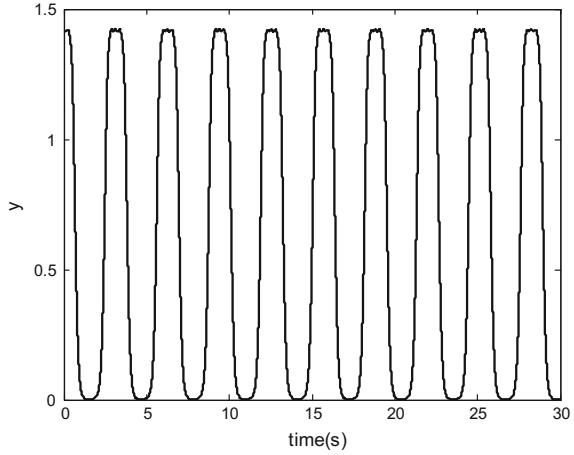


Fig. 7.14 Output of RBF

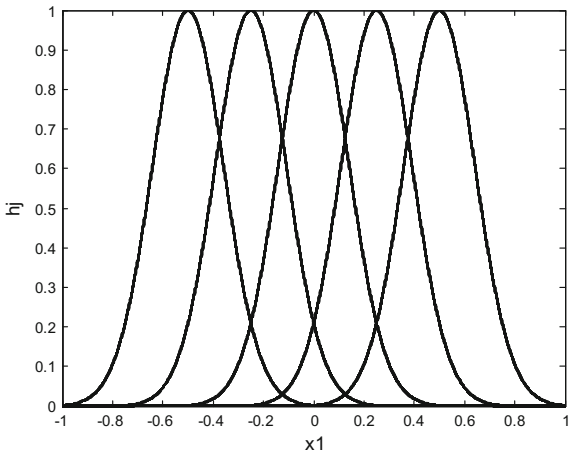


Fig. 7.15 Output of hidden neural net for first input

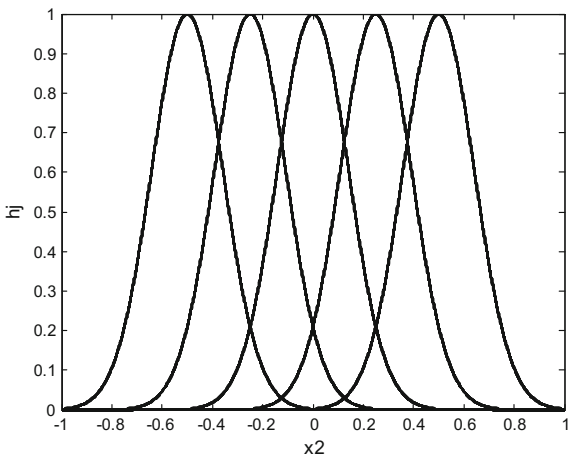


Fig. 7.16 Output of hidden neural net for second input

```
otherwise
  error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 8;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [];
function sys=mdlOutputs(t,x,u)
x1=u(1);  %Input Layer
x2=u(2);
x=[x1 x2]';

%i=2
%j=1,2,3,4,5
%k=1
c=[-0.5 -0.25 0 0.25 0.5;
  -0.5 -0.25 0 0.25 0.5];  %cij
b=[0.2 0.2 0.2 0.2 0.2]';   %bj

W=ones(5,1);  %Wj
h=zeros(5,1); %hj
for j=1:1:5
  h(j)=exp(-norm(x-c(:,j))^2/(2*b(j)*b(j))); %Hidden Layer
end
yout=W'*h;   %Output Layer

sys(1)=yout;
sys(2)=x1;
sys(3)=x2;
sys(4)=h(1);
sys(5)=h(2);
sys(6)=h(3);
sys(7)=h(4);
sys(8)=h(5);
```

(3) **Plot program: chap7_3plot.m**

```
close all;
% y=y(:,1);
% x1=y(:,2);
% x2=y(:,3);
% h1=y(:,4);
% h2=y(:,5);
% h3=y(:,6);
% h4=y(:,7);
% h5=y(:,8);

figure(1);
plot(t,y(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('y');

figure(2);
plot(y(:,2),y(:,4),'k','linewidth',2);
xlabel('x1');ylabel('hj');
hold on;
plot(y(:,2),y(:,5),'k','linewidth',2);
hold on;
plot(y(:,2),y(:,6),'k','linewidth',2);
hold on;
plot(y(:,2),y(:,7),'k','linewidth',2);
hold on;
plot(y(:,2),y(:,8),'k','linewidth',2);

figure(3);
plot(y(:,3),y(:,4),'k','linewidth',2);
xlabel('x2');ylabel('hj');
hold on;
plot(y(:,3),y(:,5),'k','linewidth',2);
hold on;
plot(y(:,3),y(:,6),'k','linewidth',2);
hold on;
plot(y(:,3),y(:,7),'k','linewidth',2);
hold on;
plot(y(:,3),y(:,8),'k','linewidth',2);
```

7.5 RBF Neural Network Approximation Based on Gradient Descent Method

7.5.1 RBF Neural Network Approximation

We use RBF neural network to approximate a plant, the structure is shown in Fig. 7.17.

In RBF neural network, $\boldsymbol{x} = [x_1 \quad x_2 \quad \ldots \quad x_n]^T$ is the input vector, and h_j is Gaussian function for neural net j, then

$$h_j = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{c}_j\|^2}{2b_j^2}\right), j = 1, 2, \ldots, m \tag{7.17}$$

where $\boldsymbol{c}_j = [c_{j1}, \ldots, c_{jn}]$ is the center vector of neural net j.

The width vector of Gaussian function is

$$\boldsymbol{b} = [b_1, \ldots, b_m]^T$$

where $b_j > 0$ represents the width value of Gaussian function for neural net j.

The weight value is

$$\boldsymbol{w} = [w_1, \ldots, w_m]^T \tag{7.18}$$

The output of RBF is

$$y_m(t) = w_1 h_1 + w_2 h_2 + \cdots + w_m h_m \tag{7.19}$$

The performance index function of RBF is

$$E(t) = \frac{1}{2}(y(t) - y_m(t))^2 \tag{7.20}$$

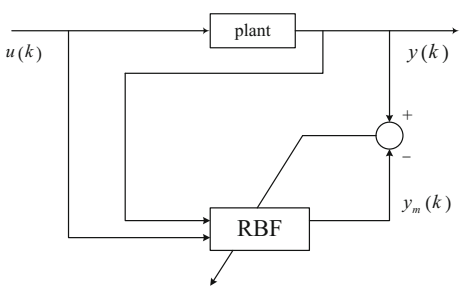


Fig. 7.17 RBF neural network approximation

According to gradient descent method, the parameters can be updated as follows:

$$\Delta w_j(t) = -\eta \frac{\partial E}{\partial w_j} = \eta(y(t) - y_m(t))h_j$$

$$w_j(t) = w_j(t-1) + \Delta w_j(t) + \alpha\big(w_j(t-1) - w_j(t-2)\big) \tag{7.21}$$

$$\Delta b_j = -\eta \frac{\partial E}{\partial b_j} = \eta(y(t) - y_m(t))w_j h_j \frac{\|\boldsymbol{x} - \boldsymbol{c}_j\|^2}{b_j^3} \tag{7.22}$$

$$b_j(t) = b_j(t-1) + \Delta b_j + \alpha\big(b_j(t-1) - b_j(t-2)\big) \tag{7.23}$$

$$\Delta c_{ji} = -\eta \frac{\partial E}{\partial c_{ji}} = \eta(y(t) - y_m(t))w_j \frac{x_j - c_{ji}}{b_j^2} \tag{7.24}$$

$$c_{ji}(t) = c_{ji}(t-1) + \Delta c_{ji} + \alpha\big(c_{ji}(t-1) - c_{ji}(t-2)\big) \tag{7.25}$$

where $\eta \in (0,1)$ is the learning rate, $\alpha \in (0,1)$ is momentum factor.

In RBF neural network approximation, the parameters of $\boldsymbol{c}_i$ and b_i must be chosen according to the scope of the input value. If the parameters $\boldsymbol{c}_i$ and b_i are chosen inappropriately, Gaussian function will not be effectively mapped, and RBF network will be invalid. The gradient descent method is an effective method to adjust $\boldsymbol{c}_i$ and b_i in RBF neural network approximation.

If the initial $\boldsymbol{c}_j$ and $\boldsymbol{b}$ are set in the effective range of inputs of RBF, we can only update weight value with fixed $\boldsymbol{c}_j$ and $\boldsymbol{b}$.

7.5.2 Simulation Example

First example: only update *w*

Using RBF neural network to approximate the following discrete plant

$$G(s) = \frac{133}{s^2 + 25s}$$

Consider a structure 2-5-1 RBF neural network, we choose inputs as $x(1) = u(t)$, $x(2) = y(t)$, and set $\alpha = 0.05$, $\eta = 0.5$. The initial weight value is chosen as random value between 0 and 1.

Choose the input as $u(\mathrm{t}) = \sin t$, consider the range of the first input $x(1)$ is $[0, 1]$, the range of the second input $x(2)$ is about $[0, 10]$, we choose the initial parameters of Gaussian function as $\boldsymbol{c}_j = \begin{bmatrix} -1 & -0.5 & 0 & 0.5 & 1 \\ -10 & -5 & 0 & 5 & 10 \end{bmatrix}^{\mathrm{T}}$, $\boldsymbol{b}_j = 1.5$, $j = 1, 2, 3, 4, 5$.

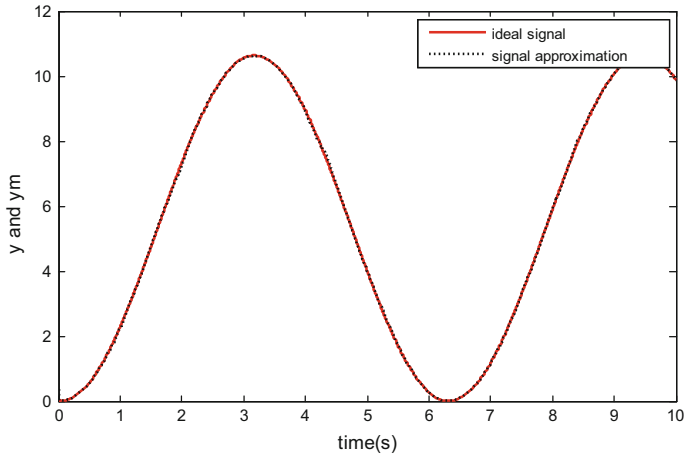
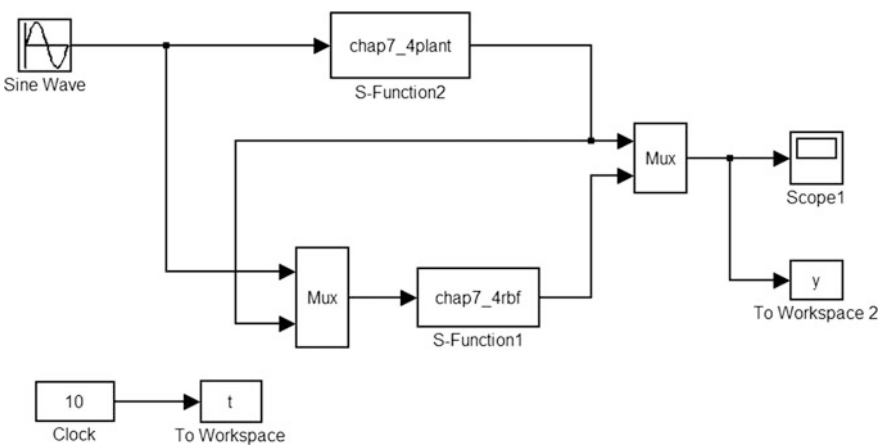


Fig. 7.18 RBF neural network approximation

In the simulation, we only update $\boldsymbol{w}$ with fixed $\boldsymbol{c}_j$ and $\boldsymbol{b}$ in RBF neural network approximation, the results are shown in Fig. 7.18.

Simulation programs:

(1) **Simulink main program: chap7_4sim.mdl**



(2) **S function of RBF: chap7_4rbf.m**

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
```

```
   [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
   sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
   sys = [];
otherwise
   error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[];
str=[];
ts=[];
function sys=mdlOutputs(t,x,u)
persistent w w_1 w_2 b ci
alfa=0.05;
xite=0.5;
if t==0
   b=1.5;
   ci=[-1 -0.5 0 0.5 1;
      -10 -5 0 5 10];
   w=rands(5,1);
   w_1=w;w_2=w_1;
end
ut=u(1);
yout=u(2);
xi=[ut yout]';
for j=1:1:5
   h(j)=exp(-norm(xi-ci(:,j))^2/(2*b^2));
end
ymout=w'*h';

d_w=0*w;
for j=1:1:5  %Only weight value update
  d_w(j)=xite*(yout-ymout)*h(j);
end
```

```
w=w_1+d_w+alfa*(w_1-w_2);

w_2=w_1;w_1=w;
sys(1)=ymout;
```

(3) **Plot program: chap7_4plot.m**

```
close all;

close all;
figure(1);
plot(t,y(:,1),'r',t,y(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('y and ym');
legend('ideal signal','signal approximation');
```

Second example: update w, c_j, b by gradient descent method

Using RBF neural network to approximate the following discrete plant

$$y(k) = u(k)^3 + \frac{y(k-1)}{1+y(k-1)^2}$$

Consider a structure 2-5-1 RBF neural network, and we choose $x(1) = u(k)$, $x(2) = y(k)$, and $\alpha = 0.05$, $\eta = 0.15$. The initial weight value is chosen as random value between 0 and 1. Choose the input as $u(k) = \sin t$, $t = k \times T$, $T = 0.001$, we set the initial parameters of Gaussian function as $\boldsymbol{c}_j = \begin{bmatrix} -1 & -0.5 & 0 & 0.5 & 1 \\ -1 & -0.5 & 0 & 0.5 & 1 \end{bmatrix}^{\mathrm{T}}$, $\boldsymbol{b}_j = 3.0$, $j = 1, 2, 3, 4, 5$.

In the simulation, $M = 1$ indicates only updating $\boldsymbol{w}$ with fixed $\boldsymbol{c}_j$ and $\boldsymbol{b}$ and $M = 2$ indicates updating $\boldsymbol{w}$, $\boldsymbol{c}_j$, $\boldsymbol{b}$, the results are shown from Figs. 7.19 and 7.20.

From the simulation test, we can see that better results can be gotten than only adjusting $\boldsymbol{w}$ by the gradient descent method, especially the initial parameters of Gaussian function $\boldsymbol{c}_j$ and $\boldsymbol{b}$ are chosen not suitably.

Simulation program: chap7_5.m

```
%RBF approximation
clear all;
close all;

alfa=0.05;
xite=0.15;
x=[0,1]';
```

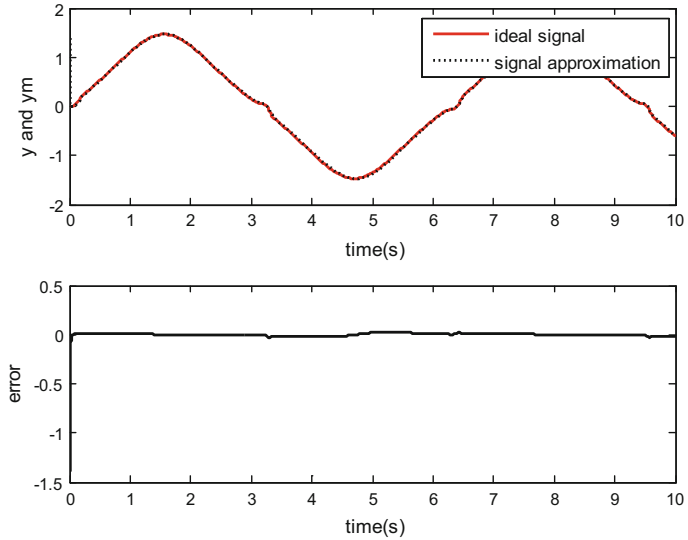


Fig. 7.19 RBF neural network approximation by only updating w (M = 1)

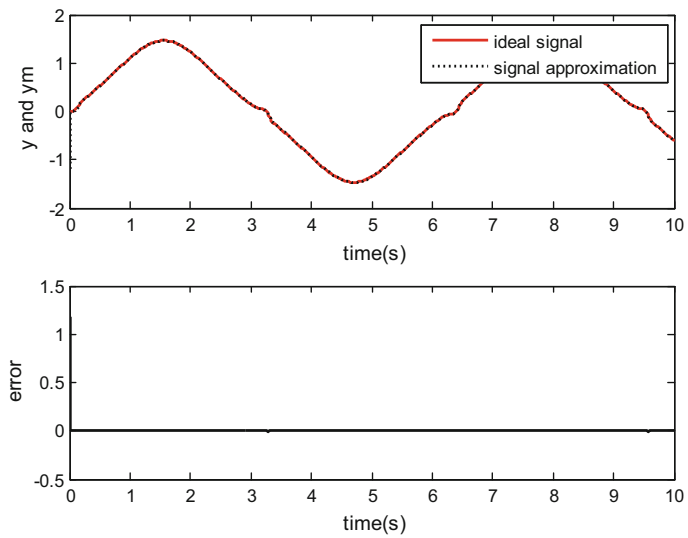


Fig. 7.20 RBF neural network approximation by updating w, b, c (M = 2)

```
b=3*ones(5,1);
c=[-1 -0.5 0 0.5 1;
  -1 -0.5 0 0.5 1];
w=rands(5,1);

w_1=w;w_2=w_1;
c_1=c;c_2=c_1;
b_1=b;b_2=b_1;
d_w=0*w;
d_b=0*b;
y_1=0;

ts=0.001;
for k=1:1:10000

time(k)=k*ts;
u(k)=sin(k*ts);

y(k)=u(k)^3+y_1/(1+y_1^2);

x(1)=u(k);
x(2)=y_1;

for j=1:1:5
   h(j)=exp(-norm(x-c(:,j))^2/(2*b(j)*b(j)));
end
ym(k)=w'*h';
em(k)=y(k)-ym(k);

M=1;
if M==1  %Only weight value update
  d_w(j)=xite*em(k)*h(j);
elseif M==2  %Update w,b,c
   for j=1:1:5
     d_w(j)=xite*em(k)*h(j);
     d_b(j)=xite*em(k)*w(j)*h(j)*(b(j)^-3)*norm(x-c(:,j))^2;
    for i=1:1:2
     d_c(i,j)=xite*em(k)*w(j)*h(j)*(x(i)-c(i,j))*(b(j)^-2);
    end
   end
   b=b_1+d_b+alfa*(b_1-b_2);
   c=c_1+d_c+alfa*(c_1-c_2);
end
w=w_1+d_w+alfa*(w_1-w_2);

y_1=y(k);
```

```
w_2=w_1;
w_1=w;

c_2=c_1;
c_1=c;

b_2=b_1;
b_1=b;
end
figure(1);
subplot(211);
plot(time,y,'r',time,ym,'k:','linewidth',2);
xlabel('time(s)');ylabel('y and ym');
legend('ideal signal','signal approximation');
subplot(212);
plot(time,y-ym,'k','linewidth',2);
xlabel('time(s)');ylabel('error');
```

7.6 Effects of Analysis on RBF Approximation

We consider approximation of the following discrete plant $y(k) = u(k)^3 + \frac{y(k-1)}{1+y(k-1)^2}$

7.6.1 Effects of Gaussian Function Parameters on RBF Approximation

In the simulation, we choose $\alpha = 0.05$, $\eta = 0.3$. The initial weight value is chosen as zeros.

From Gaussian function expression, we know that the effect of Gaussian function is related to the design of center vector c_j, width value b_j, and the number of hidden nets. The principle of c_j and b_j design should be as follows:

(1) Width value b_j represents the width of Gaussian function. The bigger value b_j is, the wider Gaussian function is. The width of Gaussian function represents the covering scope for the network input. The wider the Gaussian function is, the greater the covering scope of the network for the input is, otherwise worse covering scope is. Width value b_j should be designed moderate.

(2) Center vector c_j represents the center coordination of Gaussian function for neural net j. The nearer c_j is to the input value, the better sensitivity of Gaussian function is to the input value, otherwise the worse sensitivity is. Center vector c_j should be designed moderate.

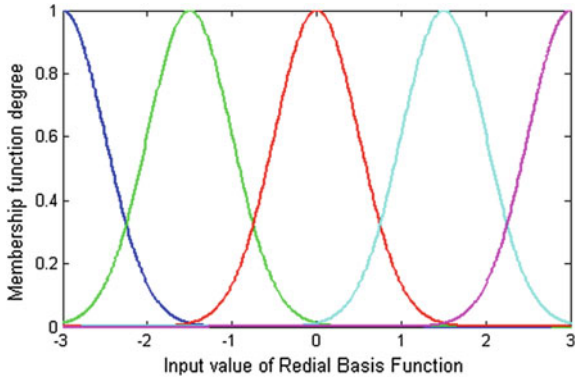


Fig. 7.21 Five Gaussian membership function

(3) The center vector c_j should be designed within the effective mapping of Gaussian membership function. For example, the scope of RBF input value is $[-3, +3]$, and then, the center vector c_j should be set in $[-3, +3]$.

In simulation, we should design the center vector c_j and the width value b_j according to the scope of practical network input value, in other words, the input value must be within the effective mapping of Gaussian membership function. Five Gaussian membership functions are shown in Fig. 7.21.

Simulation program:

Five Gaussian membership function design: chap7_6.m

```
%RBF function
clear all;
close all;

c=[-3 -1.5 0 1.5 3];

M=1;
if M==1
   b=0.50*ones(5,1);
elseif M==2
   b=1.50*ones(5,1);
end

h=[0,0,0,0,0]';

ts=0.001;
for k=1:1:2000

time(k)=k*ts;
```

```
%RBF function
x(1)=3*sin(2*pi*k*ts);

for j=1:1:5
  h(j)=exp(-norm(x-c(:,j))^2/(2*b(j)*b(j)));
end

x1(k)=x(1);
%First Redial Basis Function
h1(k)=h(1);
%Second Redial Basis Function
h2(k)=h(2);
%Third Redial Basis Function
h3(k)=h(3);
%Fourth Redial Basis Function
h4(k)=h(4);
%Fifth Redial Basis Function
h5(k)=h(5);
end
figure(1);
plot(x1,h1,'b');
figure(2);
plot(x1,h2,'g');
figure(3);
plot(x1,h3,'r');
figure(4);
plot(x1,h4,'c');
figure(5);
plot(x1,h5,'m');
figure(6);
plot(x1,h1,'b');
hold on;plot(x1,h2,'g');
hold on;plot(x1,h3,'r');
hold on;plot(x1,h4,'c');
hold on;plot(x1,h5,'m');
xlabel('Input value of Redial Basis Function');ylabel
('Membership function degree');
```

In the simulation, we choose the input of RBF as $0.5\sin(2\pi t)$, and set the structure as 2-5-1. By changing c_j and b_j values, the effects of c_j and b_j on RBF approximation are given.

Now, we analyze the effect of different c_j and b_j on RBF approximation as follows:

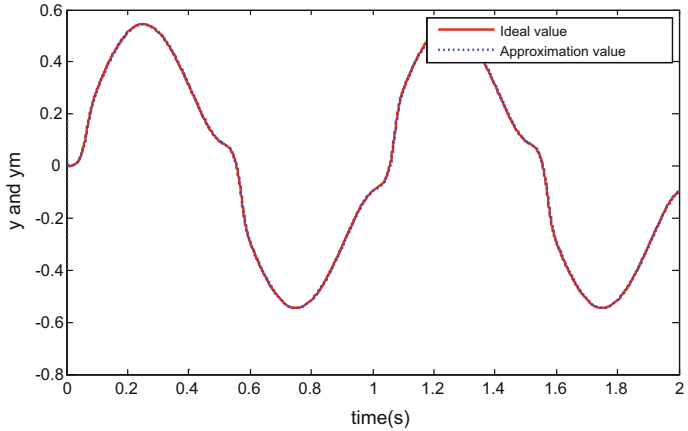


Fig. 7.22 RBF approximation with moderate b_j and c_j (Mb = 1, Mc = 1)

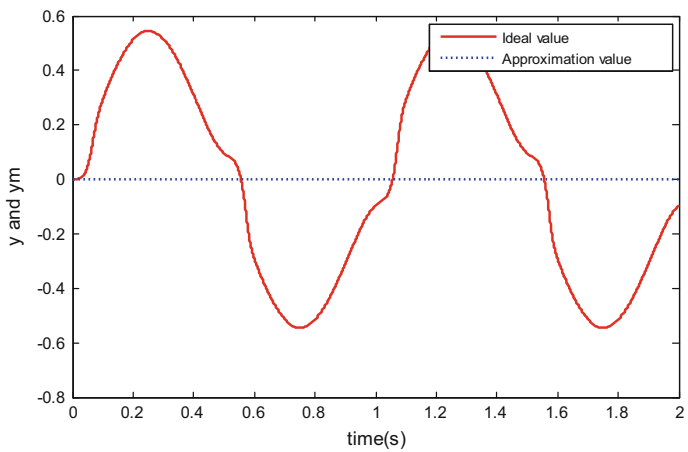


Fig. 7.23 RBF approximation with improper b_j and moderate c_j (Mb = 2, Mc = 1)

(1) RBF approximation with moderate b_j and c_j (Mb = 1, Mc = 1);
(2) RBF approximation with improper b_j and moderate c_j (Mb = 2, Mc = 1);
(3) RBF approximation with moderate b_j and improper c_j (Mb = 1, Mc = 2);
(4) RBF approximation with improper b_j and c_j (Mb = 2, Mc = 2).

The results are shown from Figs. 7.22, 7.23, 7.24, and 7.25. From the results, we can see if we design improper c_j and b_j, the RBF approximation performance will not be ensured.

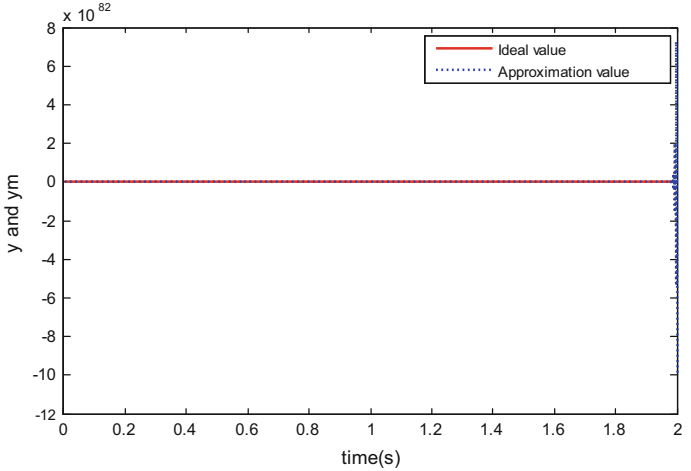


Fig. 7.24 RBF approximation with moderate b_j and improper c_j (Mb = 1, Mc = 2)

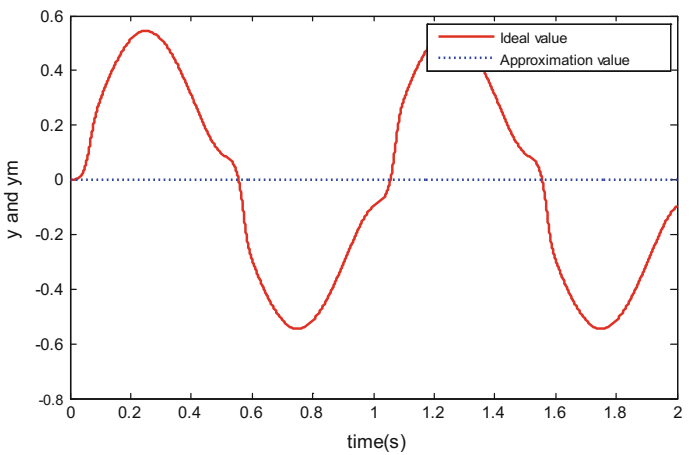


Fig. 7.25 RBF approximation with improper b_j and c_j (Mb = 2, Mc = 2)

Simulation program: chap7_7.m

```
%RBF approximation test
clear all;
close all;

alfa=0.05;
xite=0.5;
```

```
x=[0,0]';

%The parameters design of Guassian Function
%The input of RBF (u(k),y
(k)) must be in the effect range of Guassian function overlay

%The value of b represents the widenth of Guassian function overlay
Mb=1;
if Mb==1      %The width of Guassian function is moderate
  b=1.5*ones(5,1);
elseif Mb==2    %The width of Guassian function is too narrow, most overlap
of the function is near to zero
  b=0.0005*ones(5,1);
end

%The value of c represents the center position of Guassian function overlay
%the NN structure is 2-5-1: i=2; j=1,2,3,4,5; k=1
Mc=1;
if Mc==1  %The center position of Guassian function is moderate
c=[-1.5 -0.5 0 0.5 1.5;
  -1.5 -0.5 0 0.5 1.5];  %cij
elseif Mc==2  %The center position of Guassian function is improper
c=0.1*[-1.5 -0.5 0 0.5 1.5;
    -1.5 -0.5 0 0.5 1.5];  %cij
end
w=rands(5,1);
w_1=w;w_2=w_1;
y_1=0;

ts=0.001;
for k=1:1:2000

time(k)=k*ts;
u(k)=0.50*sin(1*2*pi*k*ts);

y(k)=u(k)^3+y_1/(1+y_1^2);

x(1)=u(k);
x(2)=y(k);

for j=1:1:5
  h(j)=exp(-norm(x-c(:,j))^2/(2*b(j)*b(j)));
end
ym(k)=w'*h';
em(k)=y(k)-ym(k);

d_w=xite*em(k)*h';
w=w_1+ d_w+alfa*(w_1-w_2);
```

```
y_1=y(k);
w_2=w_1;w_1=w;

end
figure(1);
plot(time,y,'r',time,ym,'b:','linewidth',2);
xlabel('time(s)');ylabel('y and ym');
legend('Ideal value','Approximation value');
```

7.6.2 *Effects of Hidden Nets Number on RBF Approximation*

From Gaussian function expression, besides the moderate center vector c_j and width value b_j, the approximation error is also related to the number of hidden nets.

In the simulation, we choose $\alpha = 0.05$, $\eta = 0.3$. The initial weight value is chosen as zeros, and the parameter of Gaussian function is chosen as $\boldsymbol{b}_j = 1.5$. The inputs of RBF are $u(k) = \sin t$ and $y(k)$, set the structure as 2-m-1, where m represents the number of hidden nets. We analyze the effect of different number of hidden nets on RBF approximation as $m = 1$, $m = 3$, and $m = 7$. According to the practical scope of the two inputs $u(k)$ and $y(k)$, for different m, the parameter c_j is chosen $\boldsymbol{c}_j = 0$, $\boldsymbol{c}_j = \frac{1}{3}[-1 \quad 0 \quad 1]^{\mathrm{T}}$, and $\boldsymbol{c}_j = \frac{1}{9}\begin{bmatrix} -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ -3 & -2 & -1 & 0 & 1 & 2 & 3 \end{bmatrix}^{\mathrm{T}}$, respectively.

The results are shown from Figs. 7.26, 7.27, 7.28, 7.29, 7.30, and 7.31. From the results, we can see that the more number the hidden nets is chosen, the smaller the approximation error can be received.

It should be noted that the more number the hidden nets is chosen, to prevent from divergence, the smaller value of η should be designed.

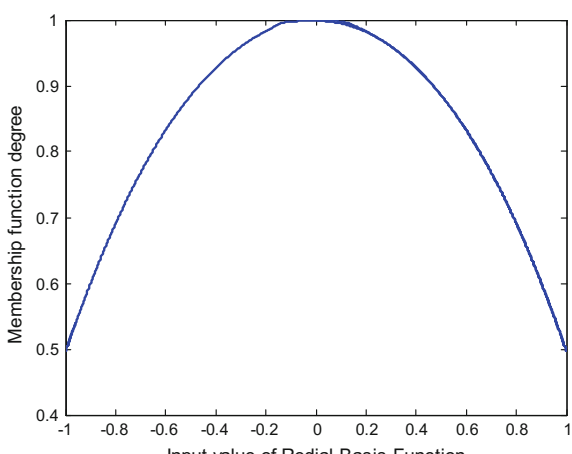


Fig. 7.26 One Gaussian function with only one hidden net $(m = 1)$

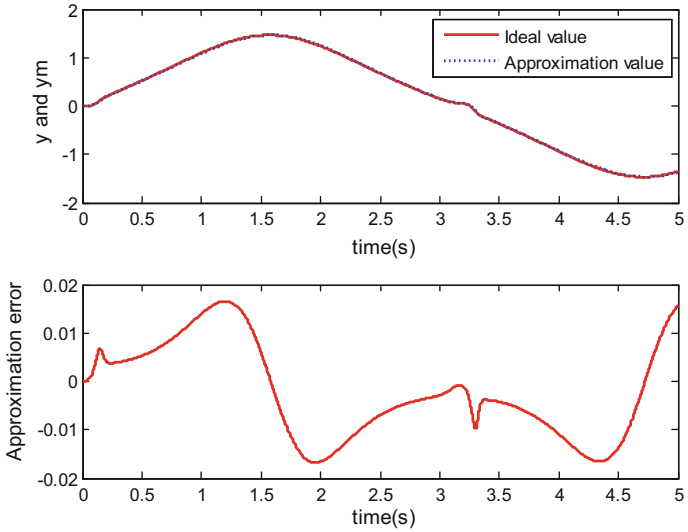
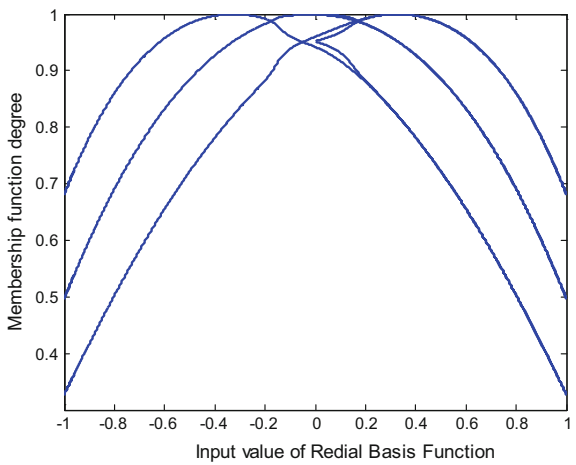


Fig. 7.27 Approximation with only one hidden net ($m = 1$)

Fig. 7.28 Three Gaussian functions with three hidden nets ($m = 3$)



Simulation program: chap7_8.m

```
%RBF approximation test
clear all;
close all;

alfa=0.05;
xite=0.3;
```

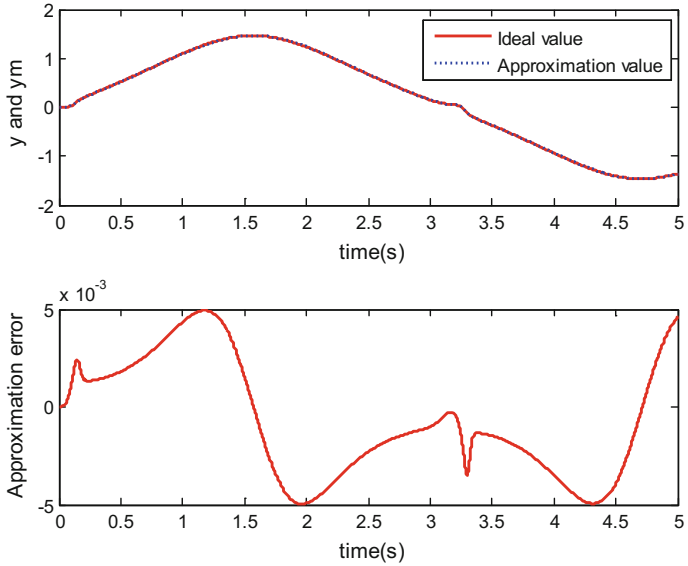
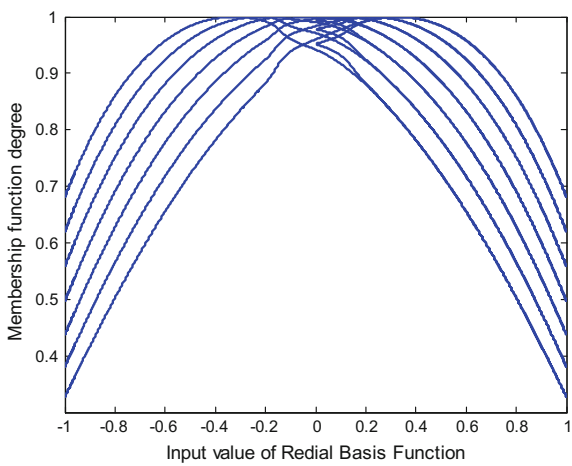


Fig. 7.29 Approximation with three hidden nets ($m = 3$)

Fig. 7.30 Seven Gaussian functions with seven hidden nets ($m = 7$)




```
x=[0,0]';

%The parameters design of Guassian Function
%The input of RBF (u(k),y(k))must be in the effect range of Guassian func-
tion overlay
%The value of b represents the widenth of Guassian function overlay
```

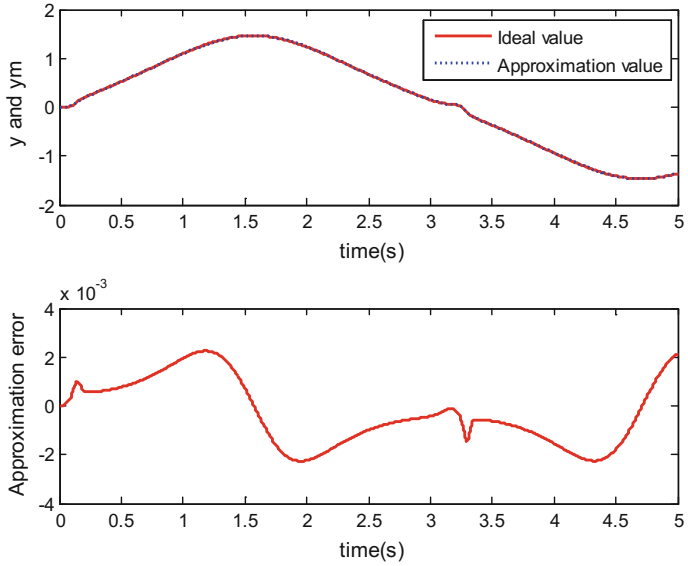


Fig. 7.31 Approximation with seven hidden nets $(m = 7)$

```
bj=1.5;   %The width of Guassian function
%The value of c represents the center position of Guassian function overlay
%the NN structure is 2-m-1: i=2; j=1,2,...,m; k=1
M=3;  %Different hidden nets number
if M==1  %only one hidden net
m=1;
c=0;
elseif M==2
m=3;
c=1/3*[-1 0 1;
    -1 0 1];
elseif M==3
m=7;
c=1/9*[-3 -2 -1 0 1 2 3;
     -3 -2 -1 0 1 2 3];
end
w=zeros(m,1);
w_1=w;w_2=w_1;
y_1=0;

ts=0.001;
for k=1:1:5000
```

```
time(k)=k*ts;
u(k)=sin(k*ts);

y(k)=u(k)^3+y_1/(1+y_1^2);

x(1)=u(k);
x(2)=y(k);

for j=1:1:m
  h(j)=exp(-norm(x-c(:,j))^2/(2*bj^2));
end
ym(k)=w'*h';
em(k)=y(k)-ym(k);

d_w=xite*em(k)*h';
w=w_1+ d_w+alfa*(w_1-w_2);

y_1=y(k);
w_2=w_1;w_1=w;

x1(k)=x(1);
for j=1:1:m
  H(j,k)=h(j);
end

if k==5000
  figure(1);
  for j=1:1:m
    plot(x1,H(j,:),'linewidth',2);
    hold on;
  end
  xlabel('Input value of Redial Basis Function');ylabel
('Membership function degree');
end
end
figure(2);
subplot(211);
plot(time,y,'r',time,ym,'b:','linewidth',2);
xlabel('time(s)');ylabel('y and ym');
legend('Ideal value','Approximation value');
subplot(212);
plot(time,y-ym,'r','linewidth',2);
xlabel('time(s)');ylabel('Approximation error');
```

7.7 RBF Neural Network Training for System Modeling

7.7.1 RBF Neural Network Training

We can use RBF neural network to train a data vector with multiinput and multi-output or to model a system off-line.

In RBF neural network, $\boldsymbol{x} = [x_1 \quad x_2 \quad \cdots \quad x_{\mathrm{n}}]^{\mathrm{T}}$ is the input vector, and h_{j} is Gaussian function for neural net j, then

$$h_{\mathrm{j}} = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{c}_j\|^2}{2b_j^2}\right), j = 1, 2, \ldots, m \tag{7.22}$$

where $\boldsymbol{c}_j = [c_{j1}, \ldots, c_{jn}]$ is the center vector of neural net j.

The width vector of Gaussian function is

$$\boldsymbol{b} = [b_1, \ldots, b_m]^T$$

where $b_j > 0$ represents the width value of Gaussian function for neural net j.

The weight value is

$$\boldsymbol{w} = [w_1, \ldots, w_m]^{\mathrm{T}} \tag{7.23}$$

The output of RBF is

$$y_l = w_1 h_1 + w_2 h_2 + \cdots + w_{\mathrm{m}} h_{\mathrm{m}} \tag{7.24}$$

Denote y_l^{d} as the ideal output, $l = 1, 2, \ldots, N$.

The error of the lth output is

$$e_l = y_l^{\mathrm{d}} - y_l$$

The performance index function of the training is

$$E(t) = \sum_{l=1}^{N} e_l^2 \tag{7.25}$$

According to gradient descent method, the weight value can be updated as follows:

$$w_j(t) = w_j(t-1) + \Delta w_j(t) + \alpha\big(w_j(t-1) - w_j(t-2)\big) \tag{7.26}$$

where $\eta \in (0, 1)$ is the learning rate, $\alpha \in (0, 1)$ is momentum factor.

7.7.2 Simulation Example

First example: MIMO data sample training.

Consider three-input and two-output data as a training sample, which is shown in Table 7.1.

RBF network structure is chosen as 3-5-1. Gaussian function parameter values c_{ij} and b_j must be chosen according to the scope of practical input value. According to the practical scope of x_1 and x_2, the parameters of $\boldsymbol{c}_i$ and b_i are designed as $\begin{bmatrix} -1 & -0.5 & 0 & 0.5 & 1 \\ -1 & -0.5 & 0 & 0.5 & 1 \\ -1 & -0.5 & 0 & 0.5 & 1 \end{bmatrix}$ and 10, the initial weight value is chosen as random value in the interval of $[-1 \quad +1]$; $\eta = 0.10$ and $\alpha = 0.05$ are chosen.

Firstly, we run chap7_9a.m, set the error index as $E = 10^{-20}$, error index change is shown as Fig. 7.32, the trained weight values are saved as wfile.dat.

Then, we run chap7_9b.m, use wfile.dat, the test results with two samples are shown in Table 7.2. From the results, we can see that good modeling performance can be received.

The programs of this example are chap7_9a.m and chap7_9b.m, which are given in the Appendix.

Table 7.1 One training sample

Input			Output	
1	0	0	1	0

Fig. 7.32 Error index change

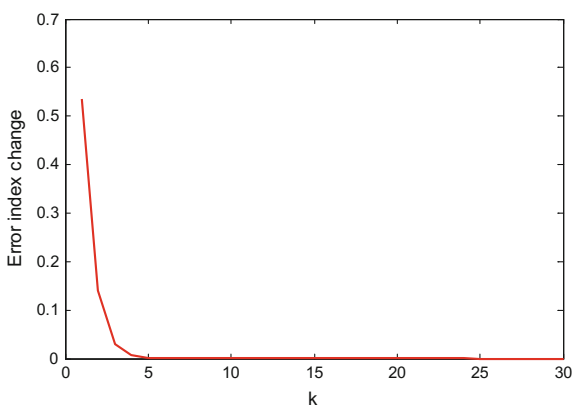


Table 7.2 Test samples and results

Input			Output	
0.970	0.001	0.001	1.0004	−0.0007
1.000	0.000	0.000	1.000	0.0000

Simulation program:

(1) MIMO data sample training: chap7_9a.m

```
%RBF Training for MIMO
clear all;
close all;

xite=0.10;
alfa=0.05;

W=rands(5,2);
W_1=W;
W_2=W_1;
h=[0,0,0,0,0]';

c=2*[-0.5 -0.25 0 0.25 0.5;
   -0.5 -0.25 0 0.25 0.5;
   -0.5 -0.25 0 0.25 0.5];  %cij
b=10;   %bj

xs=[1,0,0];%Ideal Input
ys=[1,0]; %Ideal Output
OUT=2;
NS=1;

k=0;

E=1.0;
while E>=1e-020
%for k=1:1:1000
k=k+1;
times(k)=k;

for s=1:1:NS  %MIMO Samples
x=xs(s,:);

for j=1:1:5
  h(j)=exp(-norm(x'-c(:,j))^2/(2*b^2)); %Hidden Layer
end
yl=W'*h;   %Output Layer

el=0;
y=ys(s,:);
for l=1:1:OUT
  el=el+0.5*(y(l)-yl(l))^2;   %Output error
end
es(s)=el;
```

```
E=0;
if s==NS
  for s=1:1:NS
    E=E+es(s);
  end
end
error=y-yl';
dW=xite*h*error;

W=W_1+dW+alfa*(W_1-W_2);

W_2=W_1;W_1=W;
end  %End of for
Ek(k)=E;
end  %End of while
figure(1);
plot(times,Ek,'r','linewidth',2);
xlabel('k');ylabel('Error index change');
save wfile b c W;
```

(2) MIMO data sample test: chap7_9b.m

```
%Test RBF
clear all;
load wfile b c W;

%N Samples
x=[0.970,0.001,0.001;
  1.000,0.000,0.000];
NS=2;
h=zeros(5,1); %hj

for i=1:1:NS
   for j=1:1:5
     h(j)=exp(-norm(x(i,:)'-c(:,j))^2/(2*b^2)); %Hidden Layer
   end
yl(i,:)=W'*h;   %Output Layer
end
yl
```

Second example: system modeling

Consider a nonlinear discrete-time system as

$$y(k) = \frac{0.5y(k-1)(1-y(k-1))}{1+\exp(-0.25y(k-1))} + u(k-1)$$

To model the system above, we choose RBF neural network. The network structure is chosen as 2-5-1; according to the practical scope of two inputs $u(k)$ and $y(k)$, the parameters of $\boldsymbol{c}_i$ and b_i are designed as $\begin{bmatrix} -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ -3 & -2 & -1 & 0 & 1 & 2 & 3 \end{bmatrix}$ and 1.5, each element of the initial weight vector is chosen as 0.10, and $\eta = 0.50$ and $\alpha = 0.05$ are chosen.

Firstly, we run chap7_10a.m, the input is chosen as $\boldsymbol{x} = [u(k) \quad y(k)]$, $u(k) = \sin t$, and $t = k \times ts$, where $ts = 0.001$ represents sampling time. The number of samples are chosen as $NS = 3000$. After 500-step training off-line, we get the error index change as Fig. 7.33. The trained weight values and Gaussian function parameters are saved as wfile.dat.

Then, we run chap7_10b.m, use wfile.dat, and the test results with input $\sin t$ are shown in Fig. 7.34. From the results, we can see that good modeling performance can be received.

Simulation program:

(1) **System training: chap7_10a.m**

```
%RBF Training for a Plant
clear all;
close all;
```

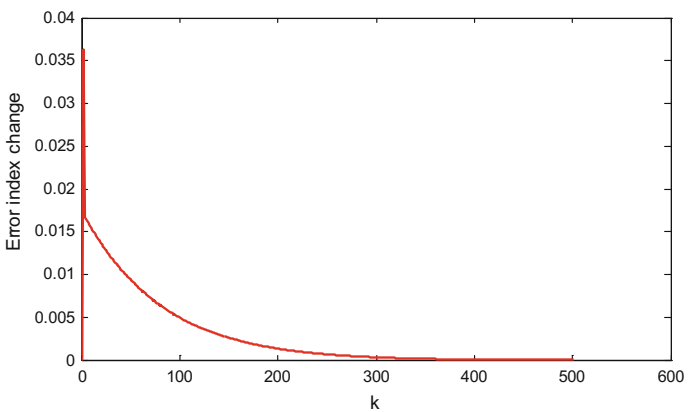
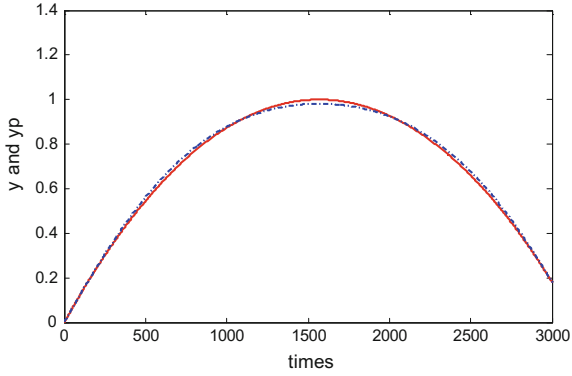


Fig. 7.33 Error index change

Fig. 7.34 Modeling test




```
ts=0.001;
xite=0.50;
alfa=0.05;

u_1=0;y_1=0;
fx_1=0;

W=0.1*ones(1,7);
W_1=W;
W_2=W_1;
h=zeros(7,1);

c1=[-3 -2 -1 0 1 2 3];
c2=[-3 -2 -1 0 1 2 3];
c=[c1;c2];

b=1.5;   %bj

NS=3000;
for s=1:1:NS  %Samples
u(s)=sin(s*ts);

fx(s)=0.5*y_1*(1-y_1)/(1+exp(-0.25*y_1));
y(s)=fx_1+u_1;

u_1=u(s);
y_1=y(s);
fx_1=fx(s);
end
k=0;

for k=1:1:500
k=k+1;
times(k)=k;
```

```
for s=1:1:NS  %Samples
  x=[u(s),y(s)];
for j=1:1:7
  h(j)=exp(-norm(x'-c(:,j))^2/(2*b^2)); %Hidden Layer
end
yl(s)=W*h;   %Output Layer

el=0.5*(y(s)-yl(s))^2;   %Output error

es(s)=el;

E=0;
if s==NS
  for s=1:1:NS
    E=E+es(s);
  end
end
error=y(s)-yl(s);
dW=xite*h'*error;

W=W_1+dW+alfa*(W_1-W_2);

W_2=W_1;W_1=W;
end  %End of for
Ek(k)=E;
end  %End of while
figure(1);
plot(times,Ek,'r','linewidth',2);
xlabel('k');ylabel('Error index change');
save wfile b c W NS;
```

(2) System test: chap7_10b.m

```
%Online RBF Etimation for Plant
clear all;
load wfile b c W NS;

ts=0.001;
u_1=0;y_1=0;
fx_1=0;
h=zeros(7,1);
for k=1:1:NS
  times(k)=k;
  u(k)=sin(k*ts);
```

```
   fx(k)=0.5*y_1*(1-y_1)/(1+exp(-0.25*y_1));
   y(k)=fx_1+u_1;

   x=[u(k),y(k)];
for j=1:1:7
   h(j)=exp(-norm(x'-c(:,j))^2/(2*b^2)); %Hidden Layer
end
yp(k)=W*h;   %Output Layer

u_1=u(k);y_1=y(k);
fx_1=fx(k);
end
figure(1);
plot(times,y,'r',times,yp,'b-.','linewidth',2);
xlabel('times');ylabel('y and yp');
```

7.8 RBF Neural Network Approximation

Since any nonlinear function over a compact set with arbitrary accuracy can be approximated by RBF neural network [4, 5], RBF neural network can be used to approximate uncertainties in the control systems. Many books about neural network control have been published [6–12].

For example, to approximate the function $f(\boldsymbol{x})$, the algorithm of RBF is expressed as

$$\begin{aligned} h_j &= g\left(\left\|\boldsymbol{x}-\boldsymbol{c}_{ij}\right\|^2/b_j^2\right) \\ f &= \boldsymbol{W}^{*\mathrm{T}}\boldsymbol{h}(\boldsymbol{x})+\varepsilon \end{aligned} \tag{7.27}$$

where $\boldsymbol{x}$ is the input vector, i denotes input neural net number in the input layer, j denotes hidden neural net number in the hidden layer, $\boldsymbol{h}=[h_1,h_2,\ldots,h_n]^{\mathrm{T}}$ denotes the output of hidden layer, $\boldsymbol{W}^*$ is the ideal weight vector, and ε is approximation error, $\varepsilon\le\varepsilon_{\mathrm{N}}$.

In the control system, if we use RBF to approximate f, we often choose the system states as the input of RBF neural network. For example, we can choose the tracking error and its derivative value as the input vector, i.e., $\boldsymbol{x}=[e\quad \dot{e}]^{\mathrm{T}}$, and then, the output of RBF is

$$\hat{f}(\boldsymbol{x})=\hat{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) \tag{7.28}$$

where $\hat{\boldsymbol{W}}$ is the estimated weight vector, which can be tuned by the adaptive algorithm in the Lyapunov stability analysis.

In Chaps. 8, 9, and 10, we use RBF approximation to design adaptive RBF controllers.

References

1. D. Graupe, *Principles of Artificial Neural Networks* (World Scientific Publishing, Singapore, 2013), p. 8
2. D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representations by error propagation. Parallel Distrib. Process. **1**, 318–362 (1986)
3. D.S. Broomhead, D. Lowe, Multivariable functional interpolation and adaptive networks. Complex Syst. **2**, 321–355 (1988)
4. J. Park, L.W. Sandberg, Universal approximation using radial-basis-function networks. Neural Comput. **3**(2), 246–257 (1991)
5. J.K. Liu, *RBF Neural Network Control for Mechanical Systems_Design, Analysis and Matlab Simulation* (Tsinghua & Springer Press, 2013)
6. S.S. Ge, T.H. Lee, C.J. Harris, *Adaptive Neural Network Control of Robotic Manipulators* (World Scientific, London, 1998)
7. S.S. Ge, C.C. Hang, T.H. Lee, T. Zhang, *Stable Adaptive Neural Network Control, Boston* (Kluwer, MA, 2001)
8. F.L. Lewis, S. Jagannathan, A. Yęildirek, *Neural Network Control of Robot Manipulators and Nonlinear Systems* (Taylor & Francis, London, 1999)
9. F.L. Lewis, J. Campos, R. Selmic, *Neuro-fuzzy Control of Industrial Systems with Actuator Nonlinearities*. Frontiers in Applied Mathematics (2002)
10. H.A. Talebi, R.V. Patel, K. Khorasani, *Control of Flexible-link Manipulators using Neural Networks, London* (Springer, New York, 2000)
11. Y.H. Kim, F.L. Lewis, *High-Level feedback control with neural networks, Singapore* (World Scientific, River Edge, NJ, 1998)
12. S.G. Fabri, V. Kadirkamanathan, *Functional Adaptive Control: An Intelligent Systems Approach* (Springer, New York, 2001)

Chapter 8
Adaptive RBF Neural Network Control

8.1 Neural Network Control

Since the idea of the computational abilities of networks composed of simple models of neurons was introduced in the 1940s [1], neural network techniques have undergone great developments and have been successfully applied in many fields such as learning, pattern recognition, signal processing, modeling, and system control. Their major advantages of highly parallel structure, learning ability, nonlinear function approximation, fault tolerance, and efficient analog VLSI implementation for real-time applications, greatly motivate the usage of neural networks in nonlinear system identification and control [2].

In many real-world applications, there are many nonlinearities, unmodeled dynamics, unmeasurable noise, and multiloop, etc., which pose problems for engineers to implement control strategies.

During the past several decades, development of new control strategies has been largely based on modern and classical control theories. Modern control theories such as adaptive and optimal control techniques and classical control theory have been based mainly on linearization of systems. In the application of such techniques, development of mathematical models is a prior necessity.

There are several reasons that have motivated vast research interests in the application of neural networks for control purposes, as alternatives to traditional control methods, among which the main points are as follows:

- Neural networks can be trained to learn any function. Thus, this self-learning ability of the neural networks eliminates the use of complex and difficult mathematical analysis which is dominant in many traditional adaptive and optimal control methods.
- The inclusions of activation function in the hidden neurons of multilayered neural networks offer nonlinear mapping ability for solving highly nonlinear control problems where to this end traditional control approaches have no practical solution yet.

- The requirement of vast a priori information regarding the plant to be controlled such as mathematical modeling is a prior necessity in traditional adaptive and optimal control techniques before they can be implemented. Due to the self-learning capability of neural networks such vast information is not required for neural controllers. Thus, neural controllers seem to be able to be applied under a wider range of uncertainty.
- The massive parallelism of neural networks offers very fast multiprocessing technique when implemented using neural chips or parallel hardware.
- Damage to some parts of the neural network hardware may not affect the overall performance badly due to its massive parallel processing architecture.

8.2 Adaptive Control Based on Neural Approximation

Note that using the gradient descent method to design the neural network weights adjustment law, neural network parameters are selected by experience, only local optimization can be guaranteed, closed-loop system stability can not be guaranteed, and closed-loop system control is easy to diverge. To solve this problem, there has been online adaptive neural network control method, the adaptive law is designed based on the Lyapunov stability theory, and the closed-loop system stability can be achieved.

8.2.1 Problem Description

Consider a second-order nonlinear system

$$\ddot{x} = f(x, \dot{x}) + g(x, \dot{x})u \tag{8.1}$$

Where f is unknown nonlinear function, g is known nonlinear function, $u \in R^n$, and $y \in R^n$ is input and output.

Eq. (8.1) can also be written as

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x_1, x_2) + g(x_1, x_2)u \\ y &= x_1 \end{aligned} \tag{8.2}$$

We assume the ideal position signal is y_d, let

$$e = y_d - y = y_d - x_1, \boldsymbol{E} = (\, e \quad \dot{e}\,)^{\mathrm{T}}$$

Design the control law as

$$u^* = \frac{1}{g(\boldsymbol{x})}\left[-f(\boldsymbol{x}) + \ddot{y}_{\mathrm{d}} + \boldsymbol{K}^{\mathrm{T}}\boldsymbol{E}\right] \tag{8.3}$$

Substitute (8.3) into (8.1), we can get the closed control system as

$$\ddot{e} + k_{\mathrm{p}}e + k_{\mathrm{d}}\dot{e} = 0 \tag{8.4}$$

We design $\boldsymbol{K} = \left(k_{\mathrm{p}}, k_{\mathrm{d}}\right)^{\mathrm{T}}$ so that all the roots of the polynomial $s^2 + k_{\mathrm{d}}s + k_{\mathrm{p}} = 0$ are in the left part of the complex plane. Then, we have $t \to \infty$, $e(t) \to 0$, and $\dot{e}(t) \to 0$.

From (8.3), we know if the function $f(\boldsymbol{x})$ is unknown, the control law will not be realized.

8.2.2 Adaptive RBF Controller Design

8.2.2.1 RBF Neural Network Design

In this section, we use RBF to design $\hat{f}(\boldsymbol{x})$ to approximate $f(\boldsymbol{x})$. The algorithm of RBF is described as

$$h_j = g\left(\left\|\boldsymbol{x} - \boldsymbol{c}_{ij}\right\|^2 / b_j^2\right)$$
$$f = \boldsymbol{W}^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) + \varepsilon$$

where $\boldsymbol{x}$ is the input vector, i denotes input neural nets number in the input layer, j denotes hidden neural nets number in the hidden layer, $\boldsymbol{h} = [h_1, h_2, \cdots, h_n]^{\mathrm{T}}$ denotes the output of hidden layer, $\boldsymbol{W}$ is weight value, ε is approximation error, $|\varepsilon| \le \varepsilon_{\mathrm{N}}$.

We use RBF to approximate f, the input vector is chosen as $\boldsymbol{x} = [e \quad \dot{e}]^{\mathrm{T}}$, the output of RBF is

$$\hat{f}(\boldsymbol{x}) = \hat{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) \tag{8.5}$$

8.2.2.2 Control Law and Adaptive Law Design

The fuzzy system approximation algorithm was applied to design indirect adaptive fuzzy controller [3]. Now, we used RBF to replace fuzzy system to design RBF adaptive controller.

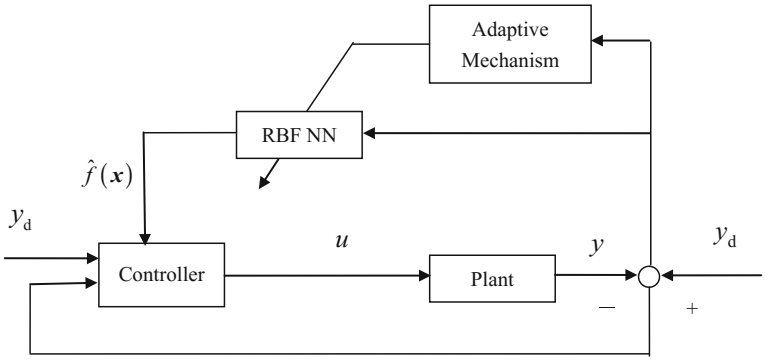


Fig. 8.1 Block diagram of the control scheme

If we use RBF neural network to represent the unknown nonlinear function f, the control law becomes

$$u = \frac{1}{g(\boldsymbol{x})}\left[-\hat{f}(\boldsymbol{x}) + \ddot{y}_d + \boldsymbol{K}^T\boldsymbol{E}\right] \tag{8.6}$$

$$\hat{f}(\boldsymbol{x}) = \hat{\boldsymbol{W}}^T\boldsymbol{h}(\boldsymbol{x}) \tag{8.7}$$

where $\boldsymbol{h}(\boldsymbol{x})$ is Gaussian function, $\hat{\boldsymbol{W}}$ is the estimated parameter for $\boldsymbol{W}$.

Figure 8.1 shows the closed-loop neural-based adaptive control scheme.

We choose the adaptive law as

$$\dot{\hat{\boldsymbol{W}}} = -\gamma\boldsymbol{E}^T\boldsymbol{P}\boldsymbol{b}\boldsymbol{h}(\boldsymbol{x}) \tag{8.8}$$

8.2.2.3 Stability Analysis

Submitting the control law (8.6) into (8.1), the closed-loop system is expressed as

$$\ddot{e} = -\boldsymbol{K}^T\boldsymbol{E} + \left[\hat{f}(\boldsymbol{x}) - f(\boldsymbol{x})\right] \tag{8.9}$$

Let

$$\boldsymbol{\Lambda} = \begin{bmatrix} 0 & 1 \\ -k_p & -k_d \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{8.10}$$

Now, (8.9) can be rewritten as

$$\dot{\boldsymbol{E}} = \boldsymbol{\Lambda E} + B\left[\hat{f}(\boldsymbol{x}) - f(\boldsymbol{x})\right] \tag{8.11}$$

The optimal weight values is

$$W^* = \arg\min_{\boldsymbol{W}\in\Omega}\left[\sup\left|\hat{f}(\boldsymbol{x}) - f(\boldsymbol{x})\right|\right] \tag{8.12}$$

Define the modeling error as

$$\omega = \hat{f}(\boldsymbol{x}|\boldsymbol{W}^*) - f(\boldsymbol{x}) \tag{8.13}$$

where $|\omega| \le \omega_{\max}$.

Then, Eq. (8.11) becomes

$$\dot{\boldsymbol{E}} = \boldsymbol{\Lambda E} + B\left\{\left[\hat{f}(\boldsymbol{x}|) - \hat{f}(\boldsymbol{x}|\boldsymbol{W}^*)\right] + \omega\right\} \tag{8.14}$$

Submit (8.7) into (8.14), we can get closed equation as

$$\dot{\boldsymbol{E}} = \boldsymbol{\Lambda E} + B\left[\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) + \omega\right] \tag{8.15}$$

Choose a Lyapunov function as

$$V = \frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{E} + \frac{1}{2\gamma}\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right) \tag{8.16}$$

where γ is positive constant. $\hat{\boldsymbol{W}} - \boldsymbol{W}^*$ denotes the parameter estimation error, and the matrix **P** is symmetric and positive definite and satisfies the following Lyapunov equation

$$\boldsymbol{\Lambda}^{\mathrm{T}}\boldsymbol{P} + \boldsymbol{P}\boldsymbol{\Lambda} = -\boldsymbol{Q} \tag{8.17}$$

With $\boldsymbol{Q} \ge 0$, $\boldsymbol{\Lambda}$ is given by (8.10).

Choosing $V_1 = \frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{E}$, $V_2 = \frac{1}{2\gamma}\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{T}\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)$, let $\boldsymbol{M} = B\left[\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) + \omega\right]$, Eq. (8.15) becomes

$$\dot{\boldsymbol{E}} = \boldsymbol{\Lambda E} + \boldsymbol{M}$$

Then,

$$\begin{aligned}\dot{V}_1 &= \frac{1}{2}\dot{\boldsymbol{E}}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{E} + \frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\dot{\boldsymbol{E}} = \frac{1}{2}\left(\boldsymbol{E}^{\mathrm{T}}\boldsymbol{\Lambda}^{\mathrm{T}} + \boldsymbol{M}^{\mathrm{T}}\right)\boldsymbol{P}\boldsymbol{E} + \frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}(\boldsymbol{\Lambda}\boldsymbol{E} + M) \\ &= \frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\left(\boldsymbol{\Lambda}^{\mathrm{T}}\boldsymbol{P} + \boldsymbol{P}\boldsymbol{\Lambda}\right)\boldsymbol{E} + \frac{1}{2}\boldsymbol{M}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{E} + \frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{M} \\ &= -\frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{E} + \frac{1}{2}\left(\boldsymbol{M}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{E} + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{M}\right) = -\frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{E} + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{M}\end{aligned}$$

Submitting $\boldsymbol{M}$ into above, noting that $\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) = \left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\left[\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\boldsymbol{h}(\boldsymbol{x})\right]$, we get

$$\begin{aligned}\dot{V}_1 &= -\frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{E} + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\omega \\ &= -\frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{E} + \left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\omega\end{aligned}$$

$$\dot{V}_2 = \frac{1}{\gamma}\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\dot{\hat{\boldsymbol{W}}}$$

Then, the derivative V becomes

$$\dot{V} = \dot{V}_1 + \dot{V}_2 = -\frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{E} + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\omega + \frac{1}{\gamma}\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\left[\dot{\hat{\boldsymbol{W}}} + \gamma\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\boldsymbol{h}(\boldsymbol{x})\right]$$

Submitting the adaptive law (8.8) into above, we have

$$\dot{V} = -\frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{E} + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\omega$$

Since $-\frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{E} \leq 0$, if we can make the approximation error ω very small by using RBF, we can get $\dot{V} \leq 0$. Then we can get that $\boldsymbol{E}$ and $\tilde{\boldsymbol{W}}$ are all limited. The convergence is

$$\|\boldsymbol{E}\| \leq \frac{2\lambda_{\max}(\boldsymbol{P}\boldsymbol{B})\omega_{\max}}{\lambda_{\min}(\boldsymbol{Q})}$$

where $\lambda(\cdot)$ is characteristic value, λ_{max} and λ_{min} are the maximum and minimum value of matrix.

8.2.3 Simulation Examples

8.2.3.1 First Simulation Example: Linear System

Consider a linear plant as follows:

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = f(\boldsymbol{x}) + g(\boldsymbol{x})u$$

where x_1 and x_2 are position and speed, respectively, u is control input, $f(\boldsymbol{x}) = -25x_2$, $g(\boldsymbol{x}) = 133$.

We use ideal position signal as $y_d(t) = \sin t$ and choose the initial states of the plant as $[\pi/60, 0]$. RBF network structure is chosen as 2-5-1. The choice of Gaussian function parameters value c_{ij} and b_j must be chosen according to the scope of practical input value, which have important role in the neural network control. If the parameters value is chosen inappropriately, Gaussian function will not be effectively mapped, and RBF network will be invalid.

According to the practical scope of x_1 and x_2, the parameters of $\boldsymbol{c}_i$ and b_j are designed as $[-1 \quad -0.5 \quad 0 \quad 0.5 \quad 1]$ and $1.\mathbf{0}$, and the initial weight value is chosen as zero. Adopting control law (8.6) and adaptive law (8.8), choose $Q = \begin{bmatrix} 500 & 0 \\ 0 & 500 \end{bmatrix}$, $k_d = 50$, $k_p = 30$, $\gamma = 1000$.

The results are shown as Figs. 8.2 and 8.3.

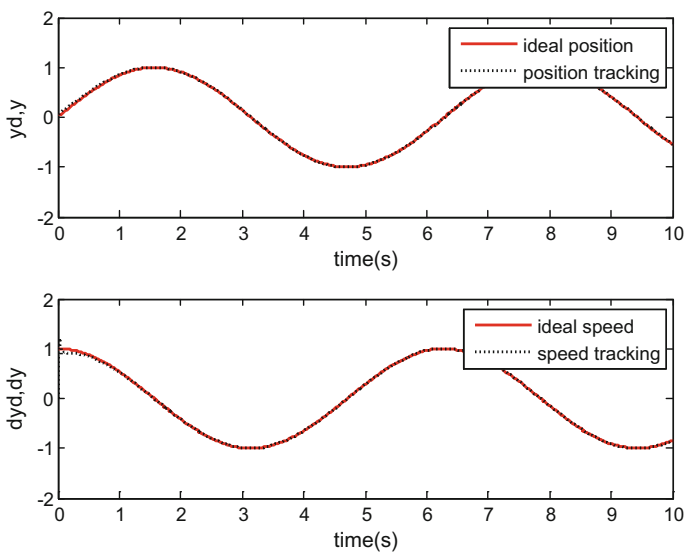


Fig. 8.2 Position and speed tracking

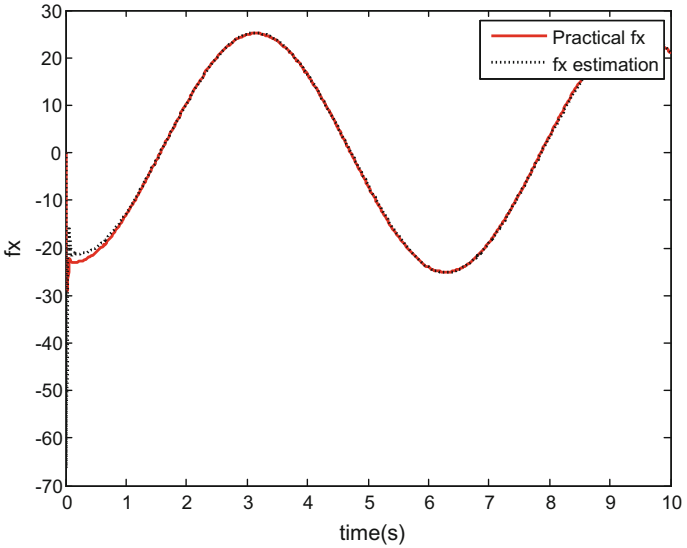
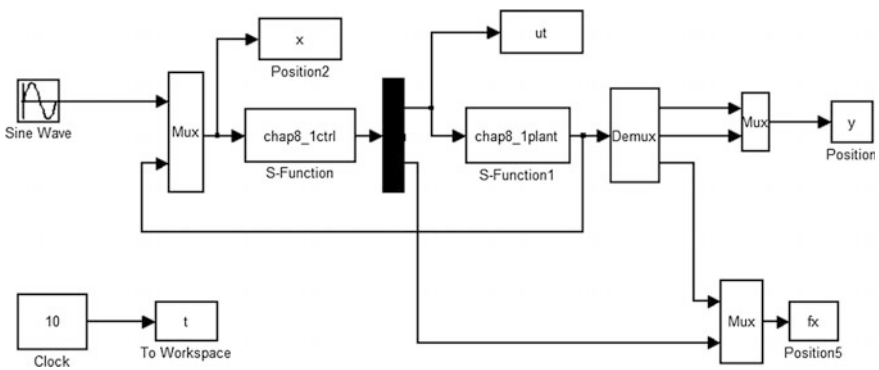


Fig. 8.3 $f(\boldsymbol{x})$ and $\hat{f}(\boldsymbol{x})$

Simulation programs:

1. **Simulink main program: chap8_1sim.mdl**



2. **S function of Control law: chap8_1ctrl.m**

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
```

```
   [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
   sys=mdlDerivatives(t,x,u);
case 3,
   sys=mdlOutputs(t,x,u);
case {2,4,9}
   sys=[];
otherwise
   error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
global c b
sizes = simsizes;
sizes.NumContStates  = 5;
sizes.NumDiscStates  = 0;
sizes.NumOutputs    = 2;
sizes.NumInputs     = 4;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = [0*ones(5,1)];
c= [-1 -0.5 0 0.5 1;
   -1 -0.5 0 0.5 1];
b=1.0;
str = [];
ts  = [];
function sys=mdlDerivatives(t,x,u)
global c b
gama=1000;
yd=sin(t);
dyd=cos(t);
ddyd=-sin(t);

x1=u(2);x2=u(3);

e=yd-x1;
de=dyd-x2;

kp=30;kd=50;
K=[kp kd]';

E=[e,de]';

Fai=[0 1;-kp -kd];
A=Fai';
```

```
Q=[500 0;0 500];
P=lyap(A,Q);

xi=[x1;x2];
h=zeros(5,1);
for j=1:1:5
  h(j)=exp(-norm(xi-c(:,j))^2/(2*b^2));
end
W=[x(1) x(2) x(3) x(4) x(5)]';

B=[0;1];
S=-gama*E'*P*B*h;

for i=1:1:5
  sys(i)=S(i);
end

function sys=mdlOutputs(t,x,u)
global c b
yd=sin(t);
dyd=cos(t);
ddyd=-sin(t);

x1=u(2);x2=u(3);
e=yd-x1;
de=dyd-x2;

kp=30;kd=50;
K=[kp kd]';

E=[e de]';

W=[x(1) x(2) x(3) x(4) x(5)]';
xi=[x1;x2];
h=zeros(5,1);
for j=1:1:5
  h(j)=exp(-norm(xi-c(:,j))^2/(2*b^2));
end
fxp=W'*h;

gx=133;

ut=1/gx*(-fxp+ddyd+K'*E);

sys(1)=ut;
sys(2)=fxp;
```

3. S function of Plant: chap8_1plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
  [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
  sys=mdlDerivatives(t,x,u);
case 3,
  sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
  sys = [];
otherwise
  error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 3;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[pi/60 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
fx=-25*x(2);

sys(1)=x(2);
sys(2)=fx+133*u;
function sys=mdlOutputs(t,x,u)
fx=-25*x(2);

sys(1)=x(1);
sys(2)=x(2);
sys(3)=fx;
```

4. Plot program: chap8_1plot.m

```
close all;

figure(1);
subplot(211);
plot(t,sin(t),'r',t,y(:,1),'k:','linewidth',2);
```

```
xlabel('time(s)');ylabel('yd,y');
legend('ideal position','position tracking');
subplot(212);
plot(t,cos(t),'r',t,y(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('dyd,dy');
legend('ideal speed','speed tracking');

figure(2);
plot(t,ut(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');

figure(3);
plot(t,fx(:,1),'r',t,fx(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('fx');
legend('Practical fx','fx estimation');
```

8.2.3.2 Second Simulation Example: Nonlinear System

Consider a single inverted pendulum system as Fig. 8.4.

The dynamic equation is described as

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = f(\boldsymbol{x}) + g(\boldsymbol{x})u$$

where $f(\boldsymbol{x}) = \frac{g\sin x_1 - mlx_2^2\cos x_1 \sin x_1/(m_c+m)}{l(4/3 - m\cos^2 x_1/(m_c+m))}$, $g(\boldsymbol{x}) = \frac{\cos x_1/(m_c+m)}{l(4/3 - m\cos^2 x_1/(m_c+m))}$, x_1, and x_2 are angle and angle speed value, respectively, g = 9.8 m/s^2, m_c = 1 kg is mass of cart, m = 0.1 kg is mass of the pendulum, l = 0.5 m is the half length of the pendulum, u is control input.

Consider ideal angle signal as $y_d(t) = 0.1\sin t$, the initial states are chosen as $[\pi/60, 0]$. RBF network structure is chosen as 2-5-1.

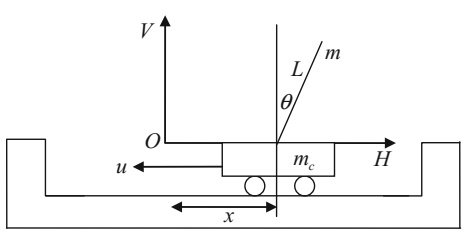


Fig. 8.4 Single inverted pendulum system

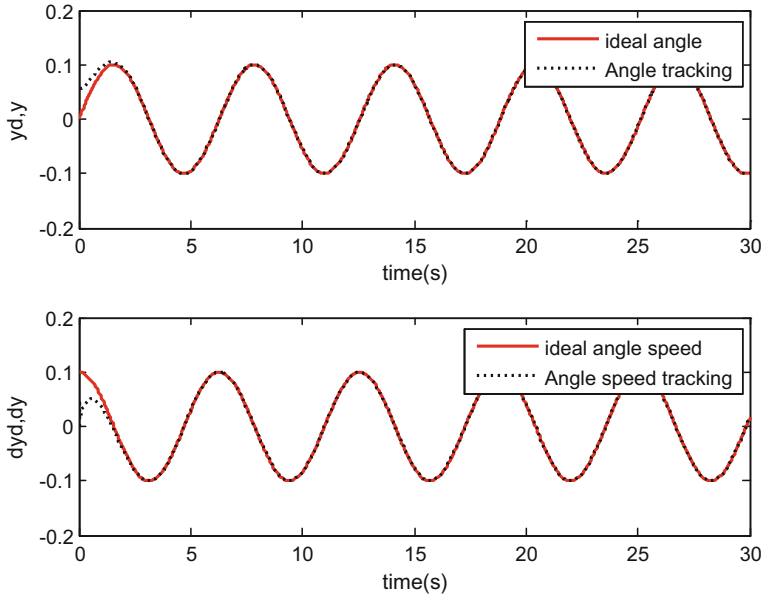


Fig. 8.5 Angle tracking and angle speed tracking

According to the practical scope of x_1 and x_2, the parameters of $\boldsymbol{c}_i$ and b_j are designed as $[-2 \quad -1 \quad 0 \quad 1 \quad 2]$ and $0.\mathbf{20}$, and the initial weight value is chosen as zero. Adopting control law (8.6) and adaptive law (8.8), choose $Q = \begin{bmatrix} 500 & 0 \\ 0 & 500 \end{bmatrix}$, $k_d = 50$, $k_p = 30$, $\gamma = 1200$. The results are shown as Figs. 8.5 and 8.6.

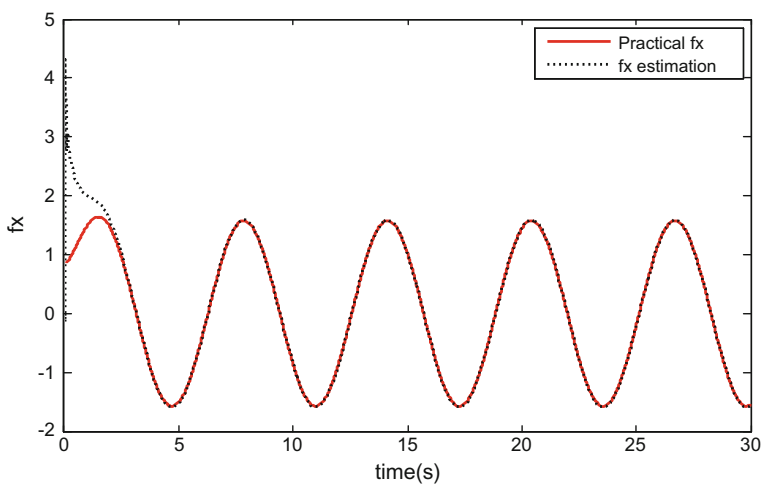
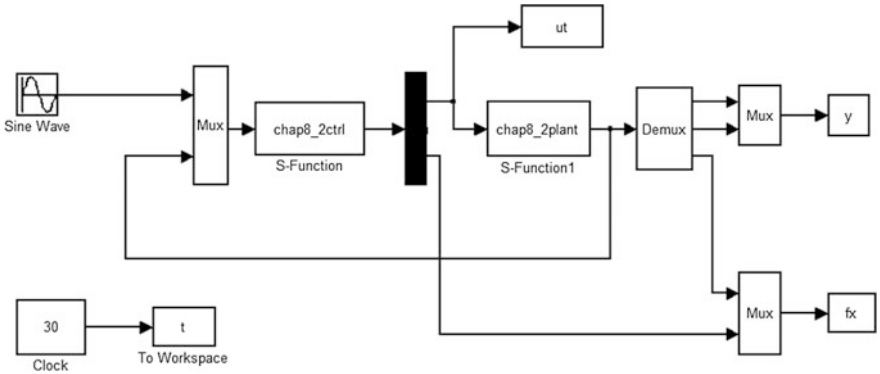


Fig. 8.6 $f(\boldsymbol{x})$ and $\hat{f}(\boldsymbol{x})$

Simulation programs:

1. **Simulink main program: chap8_2sim.mdl**



2. **S function of Control law: chap8_2ctrl.m**

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
   sys=mdlDerivatives(t,x,u);
case 3,
   sys=mdlOutputs(t,x,u);
case {2,4,9}
   sys=[];
otherwise
   error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
global c b
sizes = simsizes;
sizes.NumContStates  = 5;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 4;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = [0*ones(5,1)];
c= [-2 -1 0 1 2;
   -2 -1 0 1 2];
```

```
b=0.20;
str = [];
ts  = [];
function sys=mdlDerivatives(t,x,u)
global c b
gama=1200;
yd=0.1*sin(t);
dyd=0.1*cos(t);
ddyd=-0.1*sin(t);

x1=u(2);x2=u(3);
e=yd-x1;de=dyd-x2;

kp=30;kd=50;
K=[kp kd]';
E=[e de]';

Fai=[0 1;-kp -kd];
A=Fai';

Q=[500 0;0 500];
P=lyap(A,Q);

xi=[x1;x2];
h=zeros(5,1);
for j=1:1:5
  h(j)=exp(-norm(xi-c(:,j))^2/(2*b^2));
end
W=[x(1) x(2) x(3) x(4) x(5)]';

B=[0;1];
S=-gama*E'*P*B*h;

for i=1:1:5
  sys(i)=S(i);
end

function sys=mdlOutputs(t,x,u)
global c b
yd=0.1*sin(t);
dyd=0.1*cos(t);
ddyd=-0.1*sin(t);

x1=u(2);x2=u(3);
e=yd-x1;de=dyd-x2;

kp=30;kd=50;
K=[kp kd]';
E=[e de]';
```

```
Fai=[0 1;-kp -kd];
A=Fai';

W=[x(1) x(2) x(3) x(4) x(5)]';
xi=[e;de];
h=zeros(5,1);
for j=1:1:5
  h(j)=exp(-norm(xi-c(:,j))^2/(2*b^2));
end
fxp=W'*h;

%%%%%%%%%%
g=9.8;mc=1.0;m=0.1;l=0.5;
S=l*(4/3-m*(cos(x(1)))^2/(mc+m));
gx=cos(x(1))/(mc+m);
gx=gx/S;
%%%%%%%%%%%%%
ut=1/gx*(-fxp+ddyd+K'*E);

sys(1)=ut;
sys(2)=fxp;
```

3. S function of Plant: chap8_2plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
  [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
  sys=mdlDerivatives(t,x,u);
case 3,
  sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
  sys = [];
otherwise
  error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 3;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
```

```
sys=simsizes(sizes);
x0=[pi/60 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
g=9.8;mc=1.0;m=0.1;l=0.5;
S=l*(4/3-m*(cos(x(1)))^2/(mc+m));
fx=g*sin(x(1))-m*l*x(2)^2*cos(x(1))*sin(x(1))/(mc+m);
fx=fx/S;
gx=cos(x(1))/(mc+m);
gx=gx/S;

sys(1)=x(2);
sys(2)=fx+gx*u;
function sys=mdlOutputs(t,x,u)
g=9.8;mc=1.0;m=0.1;l=0.5;

S=l*(4/3-m*(cos(x(1)))^2/(mc+m));
fx=g*sin(x(1))-m*l*x(2)^2*cos(x(1))*sin(x(1))/(mc+m);
fx=fx/S;
gx=cos(x(1))/(mc+m);
gx=gx/S;

sys(1)=x(1);
sys(2)=x(2);
sys(3)=fx;
```

4. **Plot program: chap8_2plot.m**

```
close all;

figure(1);
subplot(211);
plot(t,0.1*sin(t),'r',t,y(:,1),'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('ideal angle','Angle tracking');
subplot(212);
plot(t,0.1*cos(t),'r',t,y(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('dyd,dy');
legend('ideal angle speed','Angle speed tracking');

figure(2);
plot(t,ut(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');
```

```
figure(3);
plot(t,fx(:,1),'r',t,fx(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('fx');
legend('Practical fx','fx estimation');
```

8.3 Adaptive Control Based on Neural Approximation with Unknown Parameter

8.3.1 Problem Description

Consider a second-order nonlinear system

$$\ddot{x} = f(x, \dot{x}) + mu \tag{8.18}$$

where f is unknown nonlinear function, m is unknown, the lower bound of m is known, $m \geq \underline{m}$, and $\underline{m} > 0$.

Eq. (8.18) can also be written as

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(\boldsymbol{x}) + mu \\ y &= x_1 \end{aligned} \tag{8.19}$$

We assume the ideal position signal is y_d, let

$$e = y_d - y = y_d - x_1, \boldsymbol{E} = [\, e \quad \dot{e} \,]^T$$

Design the control law as

$$u^* = \frac{1}{m}\left[-f(\boldsymbol{x}) + \ddot{y}_d + \boldsymbol{K}^T\boldsymbol{E}\right] \tag{8.20}$$

Substitute (8.20) into (8.18), we can get the closed control system as

$$\ddot{e} + k_p e + k_d \dot{e} = 0 \tag{8.21}$$

We design $\boldsymbol{K} = [\, k_p \quad k_d \,]^T$ so that all the roots of the polynomial $s^2 + k_d s + k_p = 0$ are in the left part of the complex plane. Then, we have $t \to \infty$, $e(t) \to 0$, and $\dot{e}(t) \to 0$.

From (8.20), we know if the function $f(\boldsymbol{x})$ and parameter m are unknown, the control law will not be realized.

8.3.2 Adaptive Controller Design

8.3.2.1 RBF Neural Network Design

In this section, just like Sect. 8.1, reference to the indirect adaptive fuzzy controller tactics given in [3], we use RBF to replace fuzzy system to design RBF indirect adaptive controller.

The algorithm of RBF to approximate $f(\boldsymbol{x})$ is described as

$$h_j = g\left(\|\boldsymbol{x} - \boldsymbol{c}_i\|^2 / b_j^2\right)$$
$$f = \boldsymbol{W}^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) + \varepsilon$$

where $\boldsymbol{x}$ is the input vector, i denotes input neural nets number in the input layer, j denotes hidden neural nets number in the hidden layer, $\boldsymbol{h} = [h_1, h_2, \cdots, h_n]^{\mathrm{T}}$ denotes the output of hidden layer, $\boldsymbol{W}$ is weight value, ε is approximation error, $|\varepsilon| \leq \varepsilon_{\mathrm{N}}$.

We use RBF to approximate f, the input vector is chosen as $\boldsymbol{x} = [\,e \quad \dot{e}\,]^{\mathrm{T}}$, and the output of RBF is

$$\hat{f}(\boldsymbol{x}) = \hat{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) \tag{8.22}$$

8.3.2.2 Control Law and Adaptive Law Design

If we use RBF neural network to represent the unknown nonlinear function f, the control law becomes

$$u = \frac{1}{\hat{m}}\left[-\hat{f}(\boldsymbol{x}) + \ddot{y}_{\mathrm{d}} + \boldsymbol{K}^{\mathrm{T}}\boldsymbol{E}\right] \tag{8.23}$$

$$\hat{f}(\boldsymbol{x}) = \hat{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) \tag{8.24}$$

where $\boldsymbol{h}(\boldsymbol{x})$ is Gaussian function, and $\hat{\boldsymbol{W}}$ is the estimated parameter for $\boldsymbol{W}$.

8.3.2.3 Stability Analysis

Submitting the control law (8.23) into (8.18), the closed-loop system is expressed as

$$\ddot{e} = -\boldsymbol{K}^{\mathrm{T}}\boldsymbol{E} + \left(\hat{f}(\boldsymbol{x}) - f(\boldsymbol{x})\right) + (m - \hat{m})u \tag{8.25}$$

Let

$$\boldsymbol{\Lambda} = \begin{bmatrix} 0 & 1 \\ -k_{\mathrm{p}} & -k_{\mathrm{d}} \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{8.26}$$

Now, (8.25) can be rewritten as

$$\dot{\boldsymbol{E}} = \boldsymbol{\Lambda E} + B\left[\left(\hat{f}(\boldsymbol{x}) - f(\boldsymbol{x})\right) + (m - \hat{m})u\right] \tag{8.27}$$

The optimal weight values is

$$W^* = \arg\min_{\boldsymbol{W}\in\Omega}\left[\sup\left|\hat{f}(\boldsymbol{x}) - f(\boldsymbol{x})\right|\right] \tag{8.28}$$

Define the modeling error as

$$\omega = \hat{f}(\boldsymbol{x}|\boldsymbol{W}^*) - f(\boldsymbol{x}) \tag{8.29}$$

Then, Eq. (8.27) becomes

$$\dot{\boldsymbol{E}} = \boldsymbol{\Lambda E} + B\left\{\left[\hat{f}(\boldsymbol{x}|) - \hat{f}(\boldsymbol{x}|\boldsymbol{W}^*)\right] + \omega + (m - \hat{m})u\right\} \tag{8.30}$$

Submit Eq. (8.24) into (8.13), we can get closed equation as

$$\dot{\boldsymbol{E}} = \boldsymbol{\Lambda E} + B\left[\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) + \omega + (m - \hat{m})u\right] \tag{8.31}$$

Choose a Lyapunov function as

$$V = \frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{E} + \frac{1}{2\gamma}\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right) + \frac{1}{2}\eta\tilde{m}^2 \tag{8.32}$$

where γ is positive constant. $\hat{\boldsymbol{W}} - \boldsymbol{W}^*$ denotes the parameter estimation error, and the matrix **P** is symmetric and positive definite and satisfies the following Lyapunov equation

$$\boldsymbol{\Lambda}^{\mathrm{T}}\boldsymbol{P} + \boldsymbol{P}\boldsymbol{\Lambda} = -\boldsymbol{Q} \tag{8.33}$$

With $\boldsymbol{Q} \geq 0$, Λ is given by (8.26), $\eta > 0$, $\tilde{m} = m - \hat{m}$.

Choosing $V_1 = \frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{E}$, $V_2 = \frac{1}{2\gamma}\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^T\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)$, $V_3 = \frac{1}{2}\eta\tilde{m}^2$, let $\boldsymbol{M} = B\left[\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) + \omega + \tilde{m}u\right]$, Eq. (8.31) becomes

$$\dot{\boldsymbol{E}} = \boldsymbol{\Lambda E} + \boldsymbol{M}$$

Then,

$$\begin{aligned}\dot{V}_1 &= \frac{1}{2}\dot{\boldsymbol{E}}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{E} + \frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\dot{\boldsymbol{E}} = \frac{1}{2}\left(\boldsymbol{E}^{\mathrm{T}}\boldsymbol{\Lambda}^{\mathrm{T}} + \boldsymbol{M}^{\mathrm{T}}\right)\boldsymbol{P}\boldsymbol{E} + \frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}(\boldsymbol{\Lambda}\boldsymbol{E} + M) \\ &= \frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\left(\boldsymbol{\Lambda}^{\mathrm{T}}\boldsymbol{P} + \boldsymbol{P}\boldsymbol{\Lambda}\right)\boldsymbol{E} + \frac{1}{2}\boldsymbol{M}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{E} + \frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{M} \\ &= -\frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{E} + \frac{1}{2}\left(\boldsymbol{M}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{E} + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{M}\right) = -\frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{E} + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{M}\end{aligned}$$

Submitting $\boldsymbol{M}$ into above, noting that $\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) = \left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\left[\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\boldsymbol{h}(\boldsymbol{x})\right]$, we get

$$\begin{aligned}\dot{V}_1 &= -\frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{E} + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\omega + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\tilde{m}u \\ &= -\frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{E} + \left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\omega + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\tilde{m}u\end{aligned}$$

$$\dot{V}_2 = \frac{1}{\gamma}\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\dot{\hat{\boldsymbol{W}}}$$

$$\dot{V}_3 = -\eta\tilde{m}\dot{\hat{m}}$$

Then, the derivative V becomes

$$\begin{aligned}\dot{V} &= \dot{V}_1 + \dot{V}_2 + \dot{V}_3 \\ &= -\frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{E} + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\omega + \frac{1}{\gamma}\left(\hat{\boldsymbol{W}} - \boldsymbol{W}^*\right)^{\mathrm{T}}\left[\dot{\hat{\boldsymbol{W}}} + \gamma\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\boldsymbol{h}(\boldsymbol{x})\right] + \tilde{m}\left(\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}u - \eta\dot{\hat{m}}\right)\end{aligned}$$

We choose the adaptive law as

$$\dot{\hat{\boldsymbol{W}}} = -\gamma\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{b}\boldsymbol{h}(\boldsymbol{x}) \tag{8.34}$$

To guarantee $\tilde{m}\left(\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}u - \eta\dot{\hat{m}}\right) \le 0$, at the same time to avoid singularity in (8.23) and guarantee $\hat{m} \ge \underline{m}$, we used the adaptive law tactics proposed in [4] as

$$\dot{\hat{m}} = \begin{cases} \frac{1}{\eta}\mathbf{E}^{\mathrm{T}}\mathbf{P}Bu, & \text{if } \mathbf{E}^{\mathrm{T}}\mathbf{P}Bu > 0 \\ \frac{1}{\eta}\mathbf{E}^{\mathrm{T}}\mathbf{P}Bu, & \text{if } \mathbf{E}^{\mathrm{T}}\mathbf{P}Bu \le 0 \text{ and } \hat{m} > \underline{m} \\ \frac{1}{\eta} & \text{if } \mathbf{E}^{\mathrm{T}}\mathbf{P}Bu \le 0 \text{ and } \hat{m} \le \underline{m} \end{cases} \tag{8.35}$$

where $\hat{m}(0) \ge \underline{m}$.

Reference to [4], the adaptive law (8.35) can also be analyzed as

1. if $\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}u > 0$, we get $\tilde{m}\left(\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}u - \eta\dot{\hat{m}}\right) = 0$ and $\dot{\hat{m}} > 0$, thus $\hat{m} > \underline{m}$;

2. if $\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}u \leq 0$ and $\hat{m} > \underline{m}$, we get $\tilde{m}\left(\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}u - \eta\dot{\hat{m}}\right) = 0$;
3. if $\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}u \leq 0$ and $\hat{m} \leq \underline{m}$, we have $\tilde{m} = m - \hat{m} \geq m - \underline{m} > 0$, thus $\tilde{m}\left(\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}u - \eta\dot{\hat{m}}\right) = \tilde{m}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}u - \tilde{m} \leq 0$ and if $\hat{m}$ increases gradually, then $\hat{m} > \underline{m}$ will be guaranteed with $\dot{\hat{m}} > 0$.

Submitting the adaptive law (8.34) and (8.35) into above, we have

$$\dot{V} = -\frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{E} + \boldsymbol{E}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{B}\omega$$

Since $-\frac{1}{2}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{E} \leq 0$, if we can make the approximation error ω very small by using RBF, we can get $\dot{V} \leq 0$. Then we can get $\boldsymbol{E}$, $\tilde{\boldsymbol{W}}$ and $\tilde{m}$ are all limited.

The convergence is

$$\|\boldsymbol{E}\| \leq \frac{2\lambda_{\max}(\boldsymbol{P}\boldsymbol{B})\omega_{\max}}{\lambda_{\min}(\boldsymbol{Q})}$$

where $\lambda(\cdot)$ is characteristic value, λ_{max} and λ_{min} are the maximum and minimum value of matrix.

8.3.3 Simulation Examples

Consider a simple plant as

$$\begin{aligned}\dot{x}_1 &= x_2\\ \dot{x}_2 &= f(\boldsymbol{x}) + mu\end{aligned}$$

where x_1 and x_2 are position and speed, respectively, u is control input, $f(\boldsymbol{x}) = -25x_2 - 10x_1$, $m = 133$.

We use ideal position signal as $y_{\mathrm{d}}(t) = \sin t$ and choose the initial states of the plant as $[0.50, 0]$.

We use RBF to approximate $f(\boldsymbol{x})$ and design adaptive algorithm to estimate parameter m. The structure is used as 2-5-1, and the input vector of RBF is $\boldsymbol{z} = [x_1 \quad x_2]^{\mathrm{T}}$. For each Gaussian function, the parameters of $\boldsymbol{c}_i$ and b_j are designed as $[-1 \quad -0.5 \quad 0 \quad 0.5 \quad 1]$ and 2.0. The initial weight value is chosen as zero.

In simulation, we use control law (8.23) and adaptive law (8.34) and (8.35), the parameters are chosen as $Q = \begin{bmatrix}500 & 0\\ 0 & 500\end{bmatrix}$, $k_{\mathrm{p}} = 30$, $k_{\mathrm{d}} = 50$, $\gamma = 1200$, $\eta = 0.0001$, $\underline{m} = 100$, $\hat{m}(0) = 120$. The results are shown from Figs. 8.7, 8.8, and 8.9.

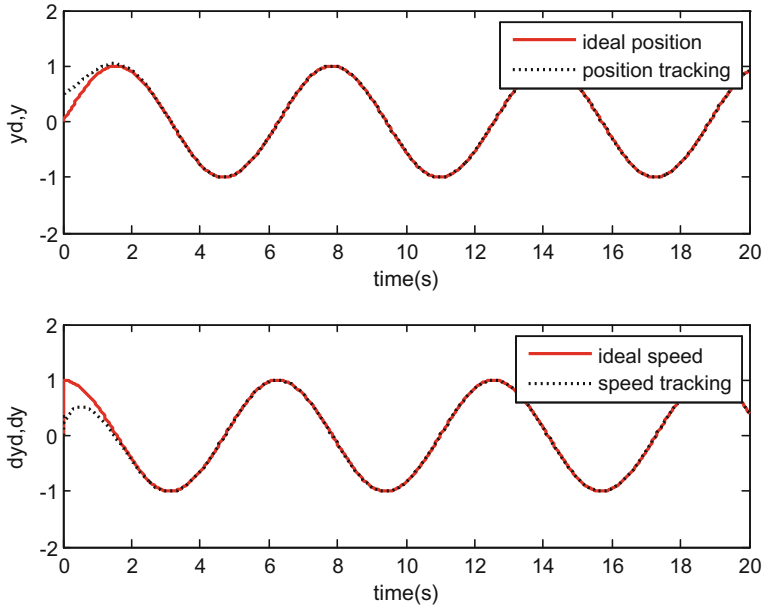


Fig. 8.7 Position and speed tracking

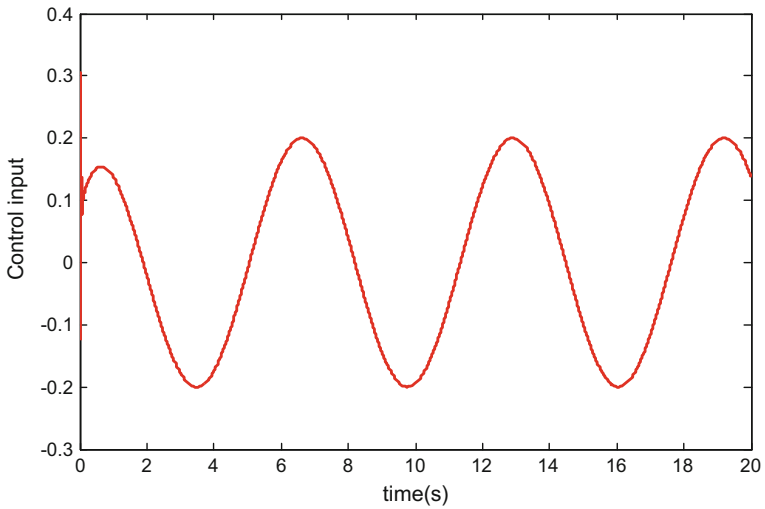


Fig. 8.8 Control input

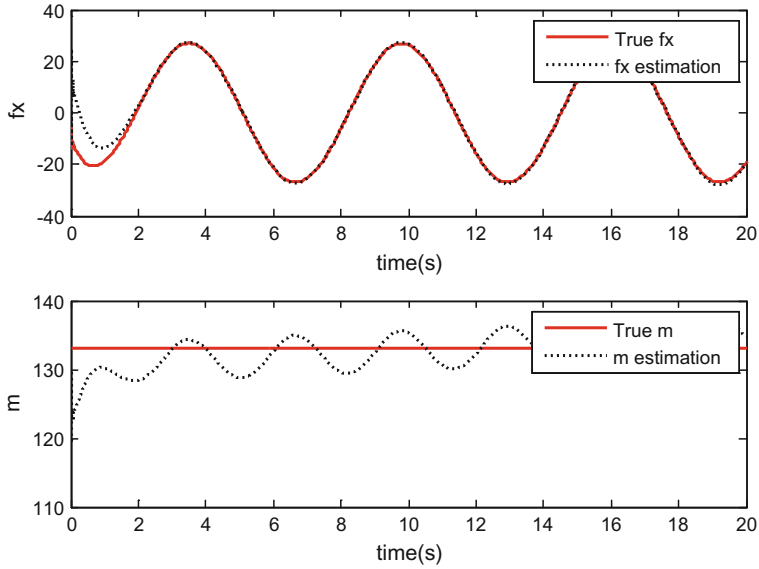
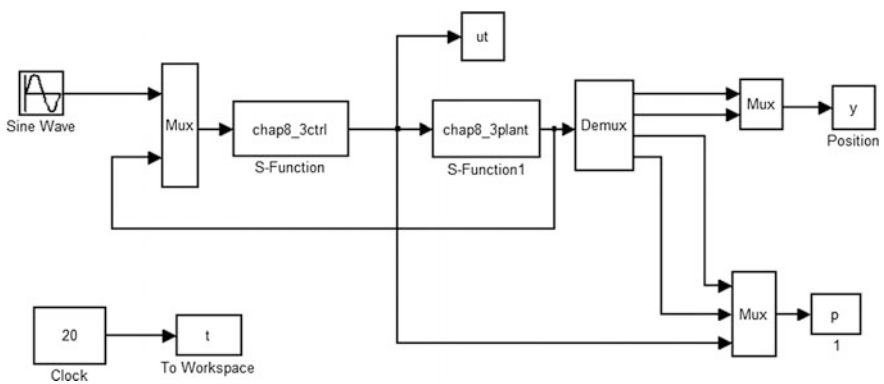


Fig. 8.9 Estimation of $f(x)$ and m

Simulation programs:

1. **Simulink main program: chap8_3sim.mdl**



2. **S function of Control law: chap8_3ctrl.m**

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
```

```
case 1,
  sys=mdlDerivatives(t,x,u);
case 3,
  sys=mdlOutputs(t,x,u);
case {2,4,9}
  sys=[];
otherwise
  error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
global node c b
node=5;
sizes = simsizes;
sizes.NumContStates  = node+1;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 3;
sizes.NumInputs      = 5;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = [zeros(1,5),120];
c= [-1 -0.5 0 0.5 1;
  -1 -0.5 0 0.5 1];
b=2;
str = [];
ts  = [];
function sys=mdlDerivatives(t,x,u)
global node c b
yd=sin(t);
dyd=cos(t);
ddyd=-sin(t);

x1=u(2);x2=u(3);
e=yd-x1;de=dyd-x2;

kp=30;kd=50;
K=[kp kd]';
E=[e de]';

Fai=[0 1;-kp -kd];
A=Fai';
Q=[500 0;0 500];
P=lyap(A,Q);
```

```
W=[x(1) x(2) x(3) x(4) x(5)]';
xi=[x1;x2];
h=zeros(5,1);
for j=1:1:5
  h(j)=exp(-norm(xi-c(:,j))^2/(2*b^2));
end
fxp=W'*h;

mp=x(node+1);

ut=1/mp*(-fxp+ddyd+K'*E);

B=[0;1];
gama=1200;
S=-gama*E'*P*B*h;
for i=1:1:node
  sys(i)=S(i);
end

eta=0.0001;
ml=100;
if (E'*P*B*ut>0)
  dm=(1/eta)*E'*P*B*ut;
end
if (E'*P*B*ut<=0)
  if (mp>ml)
  dm=(1/eta)*E'*P*B*ut;
  else
  dm=1/eta;
  end
end
sys(node+1)=dm;

function sys=mdlOutputs(t,x,u)
global node c b
yd=sin(t);
dyd=cos(t);
ddyd=-sin(t);

x1=u(2);x2=u(3);
e=yd-x1;de=dyd-x2;

kp=30;kd=50;
K=[kp kd]';
E=[e de]';
```

```
W=[x(1) x(2) x(3) x(4) x(5)]';
xi=[x1;x2];
h=zeros(5,1);
for j=1:1:node
  h(j)=exp(-norm(xi-c(:,j))^2/(2*b^2));
end
fxp=W'*h;

mp=x(node+1);

ut=1/mp*(-fxp+ddyd+K'*E);

sys(1)=ut;
sys(2)=fxp;
sys(3)=mp;
```

3. S function of Plant: chap8_3plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
  [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
  sys=mdlDerivatives(t,x,u);
case 3,
  sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
  sys = [];
otherwise
  error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 4;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.5 0];
str=[];
```

```
ts=[];
function sys=mdlDerivatives(t,x,u)
ut=u(1);

fx=-25*x(2)-10*x(1);
m=133;

sys(1)=x(2);
sys(2)=fx+m*ut;
function sys=mdlOutputs(t,x,u)
fx=-25*x(2)-10*x(1);
m=133;

sys(1)=x(1);
sys(2)=x(2);
sys(3)=fx;
sys(4)=m;
```

4. **Plot program: chap8_3plot.m**

```
close all;

figure(1);
subplot(211);
plot(t,sin(t),'r',t,y(:,1),'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('ideal position','position tracking');
subplot(212);
plot(t,cos(t),'r',t,y(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('dyd,dy');
legend('ideal speed','speed tracking');

figure(2);
plot(t,ut(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');

figure(3);
subplot(211);
plot(t,p(:,1),'r',t,p(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('fx');
legend('True fx','fx estimation');
subplot(212);
plot(t,p(:,2),'r',t,p(:,5),'k:','linewidth',2);
xlabel('time(s)');ylabel('m');
legend('True m','m estimation');
```

References

1. W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. **5**, 115–133 (1943)
2. K.J. Hunt, D. Sbarbaro, R. Zbikowski, P.J. Gawthrop, Neural networks for control system-a survey. Automatica **28**(6), 1083–1112 (1992)
3. L.X. Wang, *A Course in Fuzzy Systems and Control* (Prentice-Hall, New York, 1997)
4. A.C. Huang, Y.C. Chen, Adaptive sliding control for single-link flexible joint robot with mismatched uncertainties. IEEE Trans. Control Syst. Technol. **12**(5), 770–775 (2004)

Chapter 9
Adaptive Sliding Mode RBF Neural Network Control

Sliding mode control is an effective approach for the robust control of a class of nonlinear systems with uncertainties defined in compact sets. The direction of the control action at any moment is determined by a switching condition to force the system to evolve on the sliding surface so that the closed-loop system behaves like a lower order linear system. For the method to be applicable, a so-called matching condition should be satisfied, which requires that the uncertainties be in the range space of the control input to ensure an invariance property of the system behavior during the sliding mode.

Sliding mode control is frequently used for the control of nonlinear systems incorporated with neural network. Stability, reaching condition, and chattering phenomena are known important difficulties. For mathematically known models, such a control is used directly to track the reference signals. However, for uncertain systems with disturbance, to eliminate chattering phenomena, there is the need to design neural network compensator and then the sliding mode control law is used to generate the control input.

9.1 Typical Sliding Mode Controller Design

Sliding mode control (SMC) was first proposed and elaborated in the early 1950s in the Soviet Union by Emelyanov and several co-researchers such as Utkins and Itkis. During the last decades, significant interest on SMC has been generated in the control research community.

For linear system

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}u, \boldsymbol{x} \in R^n, u \in R \tag{9.1}$$

A sliding variable can be designed as

$$s(\boldsymbol{x}) = \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} = \sum_{i=1}^{n} c_i x_i = \sum_{i=1}^{n-1} c_i x_i + x_n \tag{9.2}$$

where $\boldsymbol{x}$ is state vector, $\boldsymbol{c} = \begin{bmatrix} c_1 & \cdots & c_{n-1} & 1 \end{bmatrix}^{\mathrm{T}}$.

In sliding mode control, parameters $c_1, c_2, \cdots, c_{n-1}$ should be selected so that the polynomial $p^{n-1} + c_{n-1}p^{n-2} + \cdots c_2 p + c_1$ is Hurwitz, where p is Laplace operator.

For example, $n = 2$, $s(\boldsymbol{x}) = c_1 x_1 + x_2$, to guarantee the polynomial $p + c_1$ Hurwitz, the eigenvalue of $p + c_1 = 0$ should has negative real part, i.e., $c_1 > 0$; e.g., if we set $c_1 = 10$, then $s(\boldsymbol{x}) = 10x_1 + x_2$.

For another example, $n = 3$, $s(\boldsymbol{x}) = c_1 x_1 + c_2 x_2 + x_3$, to guarantee the polynomial $p^2 + c_2 p + c_1$ Hurwitz, the eigenvalue of $p^2 + c_2 p + c_1 = 0$ should has negative real part. For example, we can design $\lambda > 0$ in $(p + \lambda)^2 = 0$, and then, we can get $p^2 + 2\lambda p + \lambda^2 = 0$. Therefore, we have $c_2 = 2\lambda,\ c_1 = \lambda^2$; e.g., if we set $\lambda = 5$, we can get $c_1 = 25,\ c_2 = 10$ and then $s(\boldsymbol{x}) = 25x_1 + 10x_2 + x_3$.

Now, we consider a second-order system and there are two steps in the SMC design. The first step is to design a sliding surface so that the plant restricted to the sliding surface has a desired system response. The second step is to construct a controller to drive the plant's state trajectory to the sliding surface. These constructions are built on the generalized Lyapunov stability theory.

For example, consider a plant as

$$J\ddot{\theta}(t) = u(t) + d(t) \tag{9.3}$$

where J is inertia moment, $\theta(t)$ is angle signal, $u(t)$ is control input, dt is disturbance, and $|d(t)| \leq D$.

Firstly, we design the sliding mode function as

$$s(t) = ce(t) + \dot{e}(t) \tag{9.4}$$

where c must satisfy Hurwitz condition, $c > 0$.

The tracking error and its derivative value is

$$e(t) = \theta(t) - \theta_{\mathrm{d}}(t),\ \dot{e}(t) = \dot{\theta}(t) - \dot{\theta}_{\mathrm{d}}(t)$$

where $\theta_{\mathrm{d}}(t)$ is ideal position signal.

Design Lyapunov function as

$$V = \frac{1}{2}s^2$$

Therefore, we have

$$\dot{s}(t) = c\dot{e}(t) + \ddot{e}(t) = c\dot{e}(t) + \ddot{\theta}(t) - \ddot{\theta}_{\mathrm{d}}(t) = c\dot{e}(t) + \frac{1}{J}(u + d(t)) - \ddot{\theta}_{\mathrm{d}}(t) \quad (9.5)$$

and

$$s\dot{s} = s\left(c\dot{e} + \frac{1}{J}(u + d(t)) - \ddot{\theta}_{\mathrm{d}}\right)$$

Secondly, to guarantee $s\dot{s} < 0$, we design the sliding mode controller as

$$u(t) = J\left(-c\dot{e} + \ddot{\theta}_{\mathrm{d}} - \eta\,\mathrm{sgn}\,(s)\right) - D\,\mathrm{sgn}\,(s) \quad (9.6)$$

Then, we get

$$s\dot{s} = s\left(c\dot{e} + \frac{1}{J}(u + d(t)) - \ddot{\theta}_{\mathrm{d}}\right)$$

and

$$\dot{V} = s\dot{s} = -\eta|s| - \frac{D}{J}|s| \leq 0$$

The closed system is asymptotically stable, $s \to 0$ as $t \to \infty$, then $e \to 0$ and $\dot{e} \to 0$ as $t \to \infty$, and the convergence precision is related to η.

From this example, we can see that the sliding mode control have good robustness performance. However, if we use bigger D value to overcome big disturbance dt, control input chattering phenomenon can be created, which can decorate the control performance.

In addition, in the control law (9.6), modeling information J must be known, which is difficult in practical engineering. In this chapter, we use RBF neural network to approximate unknown part of the plant.

9.2 Sliding Mode Control Based on RBF for Second-Order SISO Nonlinear System

9.2.1 Problem Statement

Consider a second-order nonlinear system as follows:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x) + bu + d(t) \end{aligned} \quad (9.7)$$

where $f(\cdot)$ is unknown nonlinear function, $b > 0$, u and θ are the control input and output, respectively, $d(t)$ is outer disturbance, and $|d(t)| \leq D$.

Let the desired output be x_d and denote

$$e = x_1 - x_\mathrm{d}$$

Design sliding mode function as

$$s = \dot{e} + ce \tag{9.8}$$

where $c > 0$, then

$$\dot{s} = \ddot{e} + c\dot{e} = \ddot{x}_1 - \ddot{x}_\mathrm{d} + c\dot{e} = f + bu + d(t) - \ddot{x}_\mathrm{d} + c\dot{e} \tag{9.9}$$

If f and b are known, we can design control law as

$$u = \frac{1}{b}(-f + \ddot{x}_\mathrm{d} - c\dot{e} - \eta \operatorname{sgn}(s)) \tag{9.10}$$

Using (9.10), (9.9) becomes

$$\dot{s} = -\eta \operatorname{sgn}(s) + d(t)$$

If we choose $\eta \geq D$, then we have

$$s\dot{s} = -\eta|s| - s \cdot d(t) \leq 0$$

Design Lyapunov function $V = \frac{1}{2}s^2$, then we have $\dot{V} \leq 0$.

The closed system is asymptotically stable, $s \to 0$ as $t \to \infty$, then $e \to 0$ and $\dot{e} \to 0$ as $t \to \infty$, and the convergence precision is related to η.

If $f(\cdot)$ is unknown, we should estimate $f(\cdot)$ by some algorithms. In the following, we will simply use RBF neural network to approximate the unknown function $f(\cdot)$.

9.2.2 *Sliding Mode Control Based on RBF for Unknown* $f(\cdot)$

In this control system, we use RBF network to approximate f. The algorithm of RBF network is

$$h_j = \exp\left(\frac{\|\boldsymbol{x} - \boldsymbol{c}_i\|^2}{2b_j^2}\right)$$

$$f = \boldsymbol{W}^{*\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) + \varepsilon$$

where $\boldsymbol{x}$ is input of the network, i is input number of the network, j is the number of hidden layer nodes in the network, $\boldsymbol{h} = [h_j]^{\mathrm{T}}$ is the output of Gaussian function, $\boldsymbol{W}^*$ is the ideal neural network weights, ε is approximation error of the neural network, and $\varepsilon \leq \varepsilon_{\mathrm{N}}$, f is the output value of the network.

The network input is selected as $\boldsymbol{x} = [x_1 \quad x_2]^{\mathrm{T}}$, and the output of RBF is

$$\hat{f}(x) = \widehat{\boldsymbol{W}}^{\mathrm{T}} \boldsymbol{h}(x) \tag{9.11}$$

where $\boldsymbol{h}(\boldsymbol{x})$ is the Gaussian function of RBF neural network.

Then, the control input (9.10) can be written as

$$u = \frac{1}{b}\left(-\hat{f} + \ddot{x}_{\mathrm{d}} - c\dot{e} - \eta \operatorname{sgn}(s)\right) \tag{9.12}$$

Submitting (9.12) to (9.9), we have

$$\begin{aligned} \dot{s} &= f + bu + d(t) - \ddot{x}_{\mathrm{d}} + c\dot{e} = f + \left(-\hat{f} + \ddot{x}_{\mathrm{d}} - c\dot{e} - \eta \operatorname{sgn}(s)\right) - \ddot{x}_{\mathrm{d}} + d(t) + c\dot{e} \\ &= f - \hat{f} - \eta \operatorname{sgn}(s) + d(t) = \tilde{f} - \eta \operatorname{sgn}(s) + d(t) \end{aligned} \tag{9.13}$$

where

$$\tilde{f} = f - \hat{f} = \boldsymbol{W}^{*\mathrm{T}} \boldsymbol{h}(\boldsymbol{x}) + \varepsilon - \widehat{\mathbf{W}}^{\mathrm{T}} \boldsymbol{h}(\boldsymbol{x}) = \widetilde{\mathbf{W}}^{\mathrm{T}} \boldsymbol{h}(\boldsymbol{x}) + \varepsilon \tag{9.14}$$

and $\widetilde{\boldsymbol{W}} = \boldsymbol{W}^* - \widehat{\boldsymbol{W}}$.

Define the Lyapunov function as

$$L = \frac{1}{2}s^2 + \frac{1}{2}\gamma \widetilde{\boldsymbol{W}}^{\mathrm{T}} \widetilde{\boldsymbol{W}}$$

where $\gamma > 0$.

Derivative L, and from (9.12) and (9.13), we have

$$\begin{aligned} \dot{L} &= s\dot{s} + \gamma \widetilde{\boldsymbol{W}}^{\mathrm{T}} \dot{\widetilde{\boldsymbol{W}}} = s\left(\tilde{f} + d(t) - \eta \operatorname{sgn}(s)\right) - \gamma \widetilde{\boldsymbol{W}}^{\mathrm{T}} \dot{\widehat{\boldsymbol{W}}} \\ &= s\left(\widetilde{\boldsymbol{W}}^{\mathrm{T}} \boldsymbol{h}(\boldsymbol{x}) + \varepsilon + d(t) - \eta \operatorname{sgn}(s)\right) - \gamma \widetilde{\boldsymbol{W}}^{\mathrm{T}} \dot{\widehat{\boldsymbol{W}}} \\ &= \widetilde{\boldsymbol{W}}^{\mathrm{T}} \left(s\boldsymbol{h}(\boldsymbol{x}) - \gamma \dot{\widehat{\boldsymbol{W}}}\right) + s(\varepsilon + d(t) - \eta \operatorname{sgn}(s)) \end{aligned}$$

Let adaptive law as

$$\dot{\widehat{\boldsymbol{W}}} = \frac{1}{\gamma} s\boldsymbol{h}(\boldsymbol{x}) \tag{9.15}$$

Then

$$\dot{L} = s(\varepsilon + d(t) - \eta \operatorname{sgn}(s)) = s(\varepsilon + d(t)) - \eta|s|$$

Due to the approximation error ε that is limited and sufficiently small, we can design $\eta \geq \varepsilon_N + D + \eta_0,\ \eta_0 > 0$; then, we can obtain approximately $\dot{L} \leq -\eta_0|s| \leq 0$.

From above analysis, we can see that RBF approximation error can be overcome by the robust term $\eta \operatorname{sgn}(s)$.

From $\dot{L} \leq -\eta_0|s| \leq 0$, we have

$$\int_0^t \dot{L}\,\mathrm{d}t \leq -\eta_0 \int_0^t |s|\mathrm{d}t, \quad \text{i.e. } L(t) - L(0) \leq -\eta_0 \int_0^t |s|\mathrm{d}t$$

Then, V is limited, s and $\widetilde{\boldsymbol{W}}$ are all limited, from $\dot{s}$ expression, $\dot{s}$ is limited, and the $\int_0^\infty \|s\|dt$ is limited. From Barbalat Lemmma [1], when $t \to \infty$, we have $s \to 0$, then $e \to 0,\ \dot{e} \to 0$.

Since V is limited as $t \to \infty$, $\widehat{\boldsymbol{W}}$ is limited. Since when $\dot{V} \equiv 0$, we cannot get $\widetilde{\boldsymbol{W}} \equiv 0$, $\widehat{\boldsymbol{W}}$ cannot converge to $\boldsymbol{W}^*$.

9.2.3 Simulation Example

Consider the following single rank inverted pendulum

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x) + 100u + d(t) \end{aligned}$$

where x_1 and x_2 are, respectively, angle and angle speed, u is the control input, $f(x) = x_1 + x_2,\ d(t) = \cos t$.

Choose desired trajectory as $x_\mathrm{d} = \sin t$, and the initial state of the plant is $[0.20\ 0]$. We adapt control law (9.12) and adaptive law (9.15), and choose $c = 10,\ \eta = 10$ and adaptive parameter $\gamma = 0.01$.

$$\operatorname{sat}(s) = \begin{cases} 1 & s > \Delta \\ ks & |s| \leq \Delta \\ -1 & s < -\Delta \end{cases}, \ k = 1/\Delta \tag{9.16}$$

In the controller, to eliminate chattering, we use saturation function to replace sign function, and choose $\Delta = 0.05$.

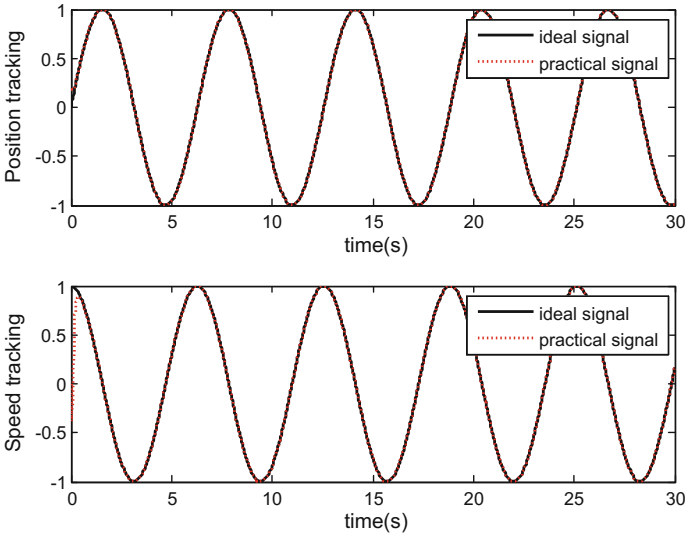
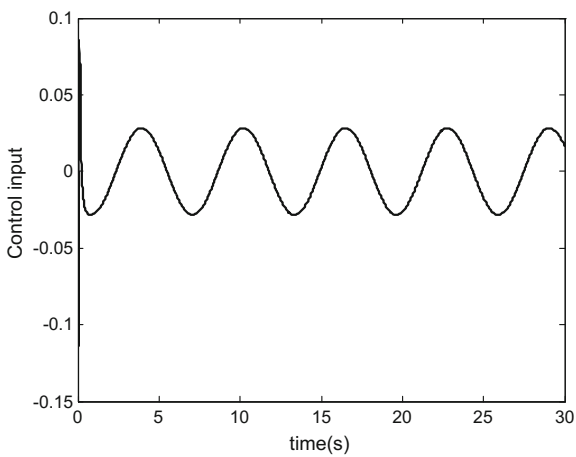


Fig. 9.1 Position and speed tracking

Fig. 9.2 Control input



The structure of RBF is chosen as 2-5-1, $\boldsymbol{c}_{\mathrm{i}}$ and b_i are designed as $[-1.0 \quad -0.5 \quad 0 \quad 0.5 \quad 1.0]$ and $b_j = 5.0$, and the initial value of each element of RBF weight value is set as 0.10.

The curves of position tracking and uncertainty approximation are shown in Figs. 9.1, 9.2 and 9.3.

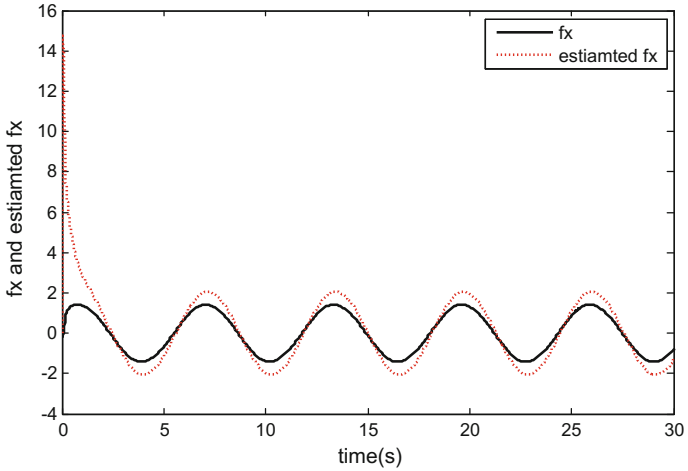
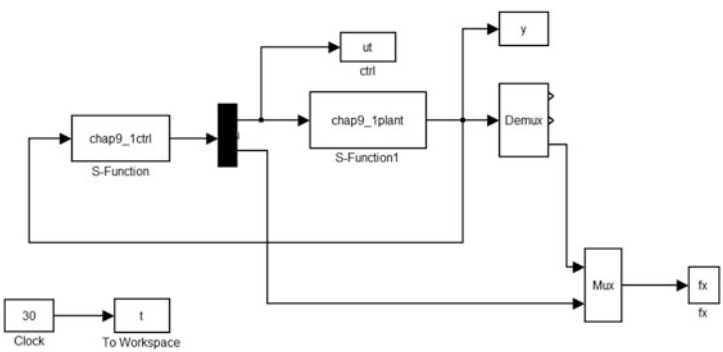


Fig. 9.3 $f(x)$ and $\hat{f}(x)$

Simulation programs:

1. **Simulink main program: chap9_1sim.mdl**



2. **S function of Control law: chap9_1ctrl.m**

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
```

```
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
global cij bj c
sizes = simsizes;
sizes.NumContStates  = 5;
sizes.NumDiscStates  = 0;
sizes.NumOutputs    = 2;
sizes.NumInputs     = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = 0.1*ones(1,5);
str = [];
ts  = [];
cij=0.5*[-2 -1 0 1 2;
       -2 -1 0 1 2];
bj=5;
c=10;
function sys=mdlDerivatives(t,x,u)
global cij bj c
xd=sin(t);dxd=cos(t);
x1=u(1);x2=u(2);
e=x1-xd;
de=x2-dxd;

s=c*e+de;

xi=[x1;x2];
h=zeros(5,1);
for j=1:1:5
  h(j)=exp(-norm(xi-cij(:,j))^2/(2*bj^2));
end
gama=0.01;
W=[x(1) x(2) x(3) x(4) x(5)]';
for i=1:1:5
  sys(i)=1/gama*s*h(i);
end
function sys=mdlOutputs(t,x,u)
global cij bj c
xd=sin(t);dxd=cos(t);ddxd=-sin(t);
x1=u(1);x2=u(2);
```

```
e=x1-xd;
de=x2-dxd;

s=c*e+de;
W=[x(1) x(2) x(3) x(4) x(5)]';
xi=[x1;x2];
h=zeros(5,1);
for j=1:1:5
  h(j)=exp(-norm(xi-cij(:,j))^2/(2*bj^2));
end
fn=W'*h;
xite=10;
b=100;

delta=0.05;
kk=1/delta;
if abs(s)>delta
sats=sign(s);
else
sats=kk*s;
end

ut=1/b*(-fn+ddxd-c*de-xite*sats);

sys(1)=ut;
sys(2)=fn;
```

3. S function of Plant: chap9_1plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
```

```
sizes.NumOutputs   = 3;
sizes.NumInputs    = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.20 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
fx=x(1)+x(2);
b=100;
ut=u(1);
dt=cos(t);

sys(1)=x(2);
sys(2)=fx+b*ut+dt;
function sys=mdlOutputs(t,x,u)
fx=x(1)+x(2);
sys(1)=x(1);
sys(2)=x(2);
sys(3)=fx;
```

4. **Plot program: chap9_1plot.m**

```
close all;

figure(1);
subplot(211);
plot(t,sin(t),'k',t,y(:,1),'r:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal signal','practical signal');
subplot(212);
plot(t,cos(t),'k',t,y(:,2),'r:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('ideal signal','practical signal');

figure(2);
plot(t,ut(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('Control input');

figure(3);
plot(t,fx(:,1),'k',t,fx(:,2),'r:','linewidth',2);
xlabel('time(s)');ylabel('fx and estiamted fx');
legend('fx','estiamted fx');
```

9.3 RBF Neural Robot Controller Design with Sliding Mode Robust Term

9.3.1 Problem Description

Consider dynamic equation of n-link manipulator as

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) + \boldsymbol{F}(\dot{\boldsymbol{q}}) + \tau_{\mathrm{d}} = \boldsymbol{\tau} \tag{9.17}$$

where $\boldsymbol{M}(\boldsymbol{q})$ is an $n \times n$ inertia matrix, $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ is an $n \times n$ matrix containing the centrifugal and Coriolis terms, $\boldsymbol{G}(\boldsymbol{q})$ is an $n \times 1$ vector containing gravitational forces and torques, $\boldsymbol{q}$ is generalized joint coordinates, τ is joint torques, and τ_{d} denotes disturbances.

The tracking error vector is designed as $\boldsymbol{e}(t) = \boldsymbol{q}_{\mathrm{d}}(t) - \boldsymbol{q}(t)$, and define the sliding mode function as

$$\boldsymbol{r} = \dot{\boldsymbol{e}} + \boldsymbol{\Lambda}\boldsymbol{e} \tag{9.18}$$

where $\boldsymbol{\Lambda} = \boldsymbol{\Lambda}^{\mathrm{T}} = [\,\lambda_1 \quad \lambda_2 \quad \cdots \quad \lambda_n\,]^T > 0$ is an appropriately chosen coefficient vector such that $s^{n-1} + \lambda_{n-1}s^{n-2} + \cdots + \lambda_1$ is Hruwitz (i.e., $\boldsymbol{e} \to 0$ exponentially as $\boldsymbol{r} \to 0$).

The sliding mode tracking error $\boldsymbol{r}$ can be viewed as the real-valued utility function of the plant performance. When $\boldsymbol{r}$ is small, system performance is good. For the system (9.16), all modeling information was expressed as $\boldsymbol{f}(\boldsymbol{x})$ by using the sliding mode tracking error $\boldsymbol{r}$ [2].

The item (9.18) gives

$$\dot{\boldsymbol{q}} = -\boldsymbol{r} + \dot{\boldsymbol{q}}_{\mathrm{d}} + \boldsymbol{\Lambda}\boldsymbol{e} \tag{9.19}$$

and

$$\begin{aligned}
\boldsymbol{M}\dot{\boldsymbol{r}} &= \boldsymbol{M}(\ddot{\boldsymbol{q}}_{\mathrm{d}} - \ddot{\boldsymbol{q}} + \boldsymbol{\Lambda}\dot{\boldsymbol{e}}) = \boldsymbol{M}(\ddot{\boldsymbol{q}}_{\mathrm{d}} + \boldsymbol{\Lambda}\dot{\boldsymbol{e}}) - \boldsymbol{M}\ddot{\boldsymbol{q}} \\
&= \boldsymbol{M}(\ddot{\boldsymbol{q}}_{\mathrm{d}} + \boldsymbol{\Lambda}\dot{\boldsymbol{e}}) + \boldsymbol{C}\dot{\boldsymbol{q}} + \boldsymbol{G} + \boldsymbol{F} + \tau_{\mathrm{d}} - \boldsymbol{\tau} \\
&= \boldsymbol{M}(\ddot{\boldsymbol{q}}_{\mathrm{d}} + \boldsymbol{\Lambda}\dot{\boldsymbol{e}}) - \boldsymbol{C}\boldsymbol{r} + \boldsymbol{C}(\dot{\boldsymbol{q}}_{\mathrm{d}} + \boldsymbol{\Lambda}\boldsymbol{e}) + \boldsymbol{G} + \boldsymbol{F} + \tau_{\mathrm{d}} - \boldsymbol{\tau} \\
&= -\boldsymbol{C}\boldsymbol{r} - \boldsymbol{\tau} + \boldsymbol{f} + \tau_{\mathrm{d}}
\end{aligned} \tag{9.20}$$

where $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{M}\ddot{\boldsymbol{q}}_{\mathrm{r}} + \boldsymbol{C}\dot{\boldsymbol{q}}_{\mathrm{r}} + \boldsymbol{G} + \boldsymbol{F}, \quad \dot{\boldsymbol{q}}_{\mathrm{r}} = \dot{\boldsymbol{q}}_{\mathrm{d}} + \boldsymbol{\Lambda}\boldsymbol{e}$.

From $\boldsymbol{f}(\boldsymbol{x})$ expression, we can see that the term $\boldsymbol{f}(\boldsymbol{x})$ includes all the modeling information.

The goal is to design a stable robust controller without any modeling information. In this section, we use RBF to approximate $\boldsymbol{f}(\boldsymbol{x})$.

9.3.2 RBF Approximation

RBF algorithm is described as

$$\begin{aligned} h_j &= \exp\frac{\|\boldsymbol{x}-\boldsymbol{c}_i\|^2}{b_j^2},\ j=1,2,\cdots,m \\ f(x) &= \boldsymbol{W}^{\mathrm{T}}\boldsymbol{h}+\varepsilon \end{aligned} \tag{9.21}$$

where $\boldsymbol{x}$ is input of RBF, $\boldsymbol{W}$ is optimum weight value, $\boldsymbol{h}=[h_1\ h_2\ \cdots\ h_m]^{\mathrm{T}}$, and ε is a very small value.

The output of RBF is used to approximate $\boldsymbol{f}(\boldsymbol{x})$:

$$\hat{\boldsymbol{f}}(\boldsymbol{x})=\widehat{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{h} \tag{9.22}$$

where $\widetilde{\boldsymbol{W}}=\boldsymbol{W}-\widehat{\boldsymbol{W}}$, $\|\boldsymbol{W}\|_{\mathrm{F}}\le W_{\max}$.

From (9.21) and (9.22), we have

$$\boldsymbol{f}-\hat{\boldsymbol{f}}=\boldsymbol{W}^{\mathrm{T}}\boldsymbol{h}+\varepsilon-\widehat{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{h}=\widetilde{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{h}+\varepsilon$$

From $\boldsymbol{f}(\boldsymbol{x})$ expression, the input of RBF should be chosen as $\boldsymbol{x}=\left[\boldsymbol{e}^{\mathrm{T}}\quad\dot{\boldsymbol{e}}^{\mathrm{T}}\quad\boldsymbol{q}_{\mathrm{d}}^{\mathrm{T}}\quad\dot{\boldsymbol{q}}_{\mathrm{d}}^{\mathrm{T}}\quad\ddot{\boldsymbol{q}}_{\mathrm{d}}^{\mathrm{T}}\right]$.

9.3.3 Control Law Design and Stability Analysis

For the system (9.17), refer to [2], the control law is designed as

$$\boldsymbol{\tau}=\hat{\boldsymbol{f}}(\boldsymbol{x})+\boldsymbol{K}_{\mathrm{v}}\boldsymbol{r}-\boldsymbol{v} \tag{9.23}$$

With robust term $\boldsymbol{v}=-(\varepsilon_{\mathrm{N}}+b_{\mathrm{d}})\,\mathrm{sgn}\,(\boldsymbol{r})$, where $\hat{\boldsymbol{f}}(\boldsymbol{x})$ is estimation of $\boldsymbol{f}(\boldsymbol{x})$ and $\boldsymbol{v}$ is robustness term.

The corresponding RBF adaptive law is designed as

$$\dot{\widehat{\boldsymbol{W}}}=\boldsymbol{\Gamma}\boldsymbol{h}\boldsymbol{r}^{\mathrm{T}} \tag{9.24}$$

where $\boldsymbol{\Gamma}=\boldsymbol{\Gamma}^{\mathrm{T}}>0$.

Inserting (9.23) to (9.20) yields

$$\begin{aligned}\boldsymbol{M}\dot{\boldsymbol{r}} &= -\boldsymbol{C}\boldsymbol{r} - \left(\hat{\boldsymbol{f}}(\boldsymbol{x}) + \boldsymbol{K}_{\mathrm{v}}\boldsymbol{r} - \boldsymbol{v}\right) + \boldsymbol{f} + \boldsymbol{\tau}_{\mathrm{d}} \\ &= -(\boldsymbol{K}_{\mathrm{v}} + \boldsymbol{C})\boldsymbol{r} + \widetilde{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{h} + (\boldsymbol{\varepsilon} + \boldsymbol{\tau}_{\mathrm{d}}) + \boldsymbol{v} \\ &= -(\boldsymbol{K}_{\mathrm{v}} + \boldsymbol{C})\boldsymbol{r} + \boldsymbol{\varsigma}_1\end{aligned} \tag{9.25}$$

where $\boldsymbol{\varsigma}_1 = \widetilde{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{\varphi} + (\boldsymbol{\varepsilon} + \boldsymbol{\tau}_{\mathrm{d}}) + \boldsymbol{v}$.

Define Lyapunov function as

$$L = \frac{1}{2}\boldsymbol{r}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{r} + \frac{1}{2}\mathrm{tr}\left(\widetilde{\boldsymbol{W}}^{\mathrm{T}}\Gamma^{-1}\widetilde{\boldsymbol{W}}\right)$$

Thus

$$\dot{L} = \boldsymbol{r}^{\mathrm{T}}\boldsymbol{M}\dot{\boldsymbol{r}} + \frac{1}{2}\boldsymbol{r}^{\mathrm{T}}\dot{\boldsymbol{M}}\boldsymbol{r} + \mathrm{tr}\left(\widetilde{\boldsymbol{W}}^{\mathrm{T}}\Gamma^{-1}\dot{\widetilde{\boldsymbol{W}}}\right)$$

Inserting (9.24) into above yields

$$\dot{L} = -\boldsymbol{r}^{\mathrm{T}}\boldsymbol{K}_{\mathrm{v}}\boldsymbol{r} + \frac{1}{2}\boldsymbol{r}^{\mathrm{T}}(\dot{\boldsymbol{M}} - 2\boldsymbol{C})\boldsymbol{r} + \mathrm{tr}\,\widetilde{\boldsymbol{W}}^{\mathrm{T}}\left(\Gamma^{-1}\dot{\widetilde{\boldsymbol{W}}} + \boldsymbol{h}\boldsymbol{r}^{\mathrm{T}}\right) + \boldsymbol{r}^{\mathrm{T}}(\boldsymbol{\varepsilon} + \boldsymbol{\tau}_{\mathrm{d}} + \boldsymbol{v})$$

Since

1. According to the skew-symmetric characteristics of manipulator dynamic equation, $\boldsymbol{r}^{\mathrm{T}}(\dot{\boldsymbol{M}} - 2\boldsymbol{C})\,\boldsymbol{r} = 0$;
2. $\boldsymbol{r}^{\mathrm{T}}\widetilde{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{h} = \mathrm{tr}\left(\widetilde{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{h}\boldsymbol{r}^{\mathrm{T}}\right)$;
3. $\dot{\widetilde{\boldsymbol{W}}} = -\dot{\widehat{\boldsymbol{W}}} = -\boldsymbol{\Gamma}\boldsymbol{h}\boldsymbol{r}^{\mathrm{T}}$.

Then

$$\dot{L} = -\boldsymbol{r}^{\mathrm{T}}\boldsymbol{K}_{\mathrm{v}}\boldsymbol{r} + \boldsymbol{r}^{\mathrm{T}}(\boldsymbol{\varepsilon} + \boldsymbol{\tau}_{\mathrm{d}} + \boldsymbol{v})$$

Consider

$$\begin{aligned}\boldsymbol{r}^{\mathrm{T}}(\boldsymbol{\varepsilon} + \boldsymbol{\tau}_{\mathrm{d}} + \boldsymbol{v}) &= \boldsymbol{r}^{\mathrm{T}}(\boldsymbol{\varepsilon} + \boldsymbol{\tau}_{\mathrm{d}}) + \boldsymbol{r}^{\mathrm{T}}(-(\boldsymbol{\varepsilon}_{\mathrm{N}} + b_{\mathrm{d}})\,\mathrm{sgn}\,(\boldsymbol{r})) \\ &= \boldsymbol{r}^{\mathrm{T}}(\boldsymbol{\varepsilon} + \boldsymbol{\tau}_{\mathrm{d}}) - \|\boldsymbol{r}\|(\boldsymbol{\varepsilon}_{\mathrm{N}} + \boldsymbol{b}_{\mathrm{d}}) \le 0\end{aligned}$$

There results finally

$$\dot{L} \le -\boldsymbol{r}^{\mathrm{T}}\boldsymbol{K}_{\mathrm{v}}\boldsymbol{r} \le 0$$

From above analysis, we can see that RBF approximation error can be overcome by the robust term.

From $\dot{L} \leq -\boldsymbol{r}^{\mathrm{T}}\boldsymbol{K}_{\mathrm{v}}\boldsymbol{r} \leq 0$, we have

$$\int_0^t \dot{L}\,\mathrm{d}t \leq -\lambda_{\min}(\boldsymbol{K}_{\mathrm{v}}) \int_0^t \|\boldsymbol{r}\|\,\mathrm{d}t, \quad \text{i.e. } L(t) - L(0) \leq -\lambda_{\min}(\boldsymbol{K}_{\mathrm{v}}) \int_0^t \|\boldsymbol{r}\|\mathrm{d}t$$

where $\lambda_{\min}(\boldsymbol{K}_{\mathrm{v}})$ is the minimum eigenvalue of $\boldsymbol{K}_{\mathrm{v}}$.

Then, L is limited, $\boldsymbol{r}$ and $\widetilde{\boldsymbol{W}}$ are all limited, from $\dot{\boldsymbol{r}}$ expression, $\dot{\boldsymbol{r}}$ is limited, and the $\int_0^\infty \|\boldsymbol{r}\|dt$ is limited. From Barbalat Lemmma [1], when $t \to \infty$, we have $\boldsymbol{r} \to 0$, then $\boldsymbol{e} \to 0$, $\dot{\boldsymbol{e}} \to 0$, and the convergence precision is related to $\boldsymbol{K}_{\mathrm{v}}$.

Since $L \geq 0$, $\dot{L} \leq 0$, L is limited as $t \to \infty$; thus, $\widehat{\mathbf{W}}$ is limited. Since when $\dot{L} \equiv 0$, we cannot get $\widetilde{\boldsymbol{W}} \equiv 0$, $\widehat{\mathbf{W}}$ cannot converge to $\boldsymbol{W}$.

9.3.4 Simulation Examples

Consider a plant as

$$\boldsymbol{M}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{V}(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) + \boldsymbol{F}(\dot{\boldsymbol{q}}) + \boldsymbol{\tau}_{\mathrm{d}} = \boldsymbol{\tau}$$

where $\boldsymbol{M}(\boldsymbol{q}) = \begin{bmatrix} p_1 + p_2 + 2p_3\cos q_2 & p_2 + p_3\cos q_2 \\ p_2 + p_3\cos q_2 & p_2 \end{bmatrix}$, $\boldsymbol{V}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \begin{bmatrix} -p_3\dot{q}_2\sin q_2 & -p_3(\dot{q}_1 + \dot{q}_2)\sin q_2 \\ p_3\dot{q}_1\sin q_2 & 0 \end{bmatrix}$, $\boldsymbol{G}(\boldsymbol{q}) = \begin{bmatrix} p_4 g\cos q_1 + p_5 g\cos(q_1 + q_2) \\ p_5 g\cos(q_1 + q_2) \end{bmatrix}$, $\boldsymbol{F}(\dot{\boldsymbol{q}}) = 0.02\ \mathrm{sgn}(\dot{\boldsymbol{q}})$, $\boldsymbol{\tau}_{\mathrm{d}} = [0.2\sin(t) \quad 0.2\sin(t)]^{\mathrm{T}}$, $\boldsymbol{p} = [p_1, p_2, p_3, p_4, p_5] = [2.9, 0.76, 0.87, 3.04, 0.87]$.

For RBF neural network, the structure is 2-7-1, the input is chosen as $\boldsymbol{z} = [\boldsymbol{e} \quad \dot{\boldsymbol{e}}]$, the parameters of Gaussian function $\boldsymbol{c}_i$ and b_j are chosen as $[-1.5 \quad -1.0 \quad -0.5 \quad 0 \quad 0.5 \quad 1.0 \quad 1.5]$ and 10, the number of hidden nets are chosen as 7, and the initial weight value is chosen as zero. The desired trajectory is $q_{1\mathrm{d}} = 0.1\sin t$, $q_{2\mathrm{d}} = 0.1\sin t$. The initial value of the plant is $[0.09 \quad 0 \quad -0.09 \quad 0]$.

Use control law (9.23) and adaptive law (9.24), $\boldsymbol{K}_{\mathrm{v}} = \mathrm{diag}\{10, 10\}$, $\boldsymbol{\Gamma} = \mathrm{diag}\{15, 15\}$, $\boldsymbol{\Lambda} = \mathrm{diag}\{5, 5\}$. The simulation results are shown from Figs. 9.4, 9.5, 9.6, and 9.7.

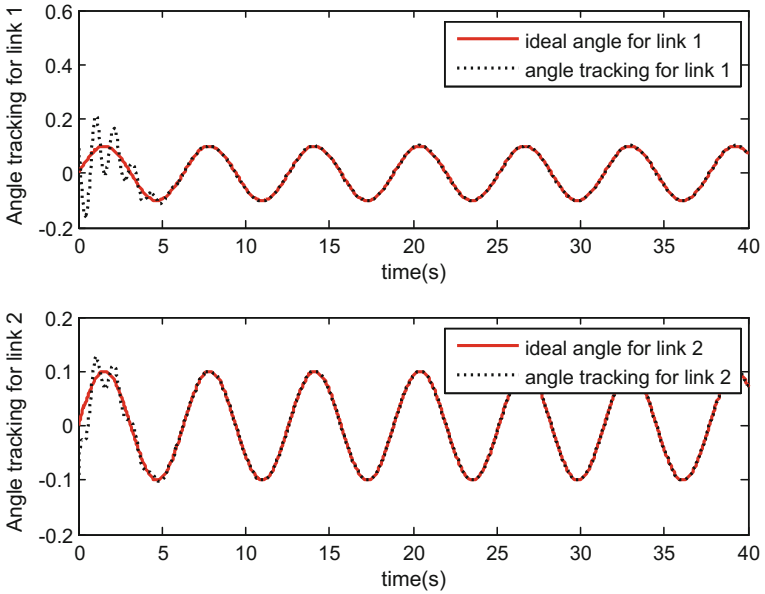


Fig. 9.4 Angle tracking

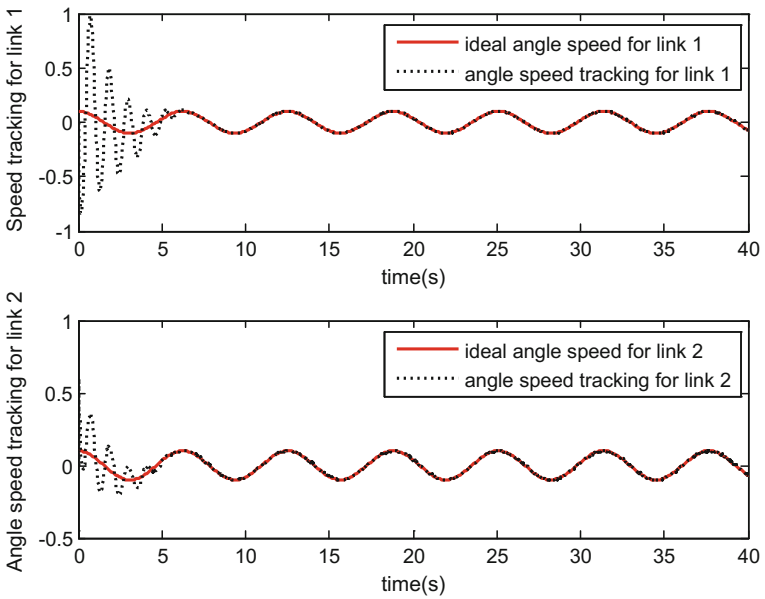


Fig. 9.5 Angle speed tracking

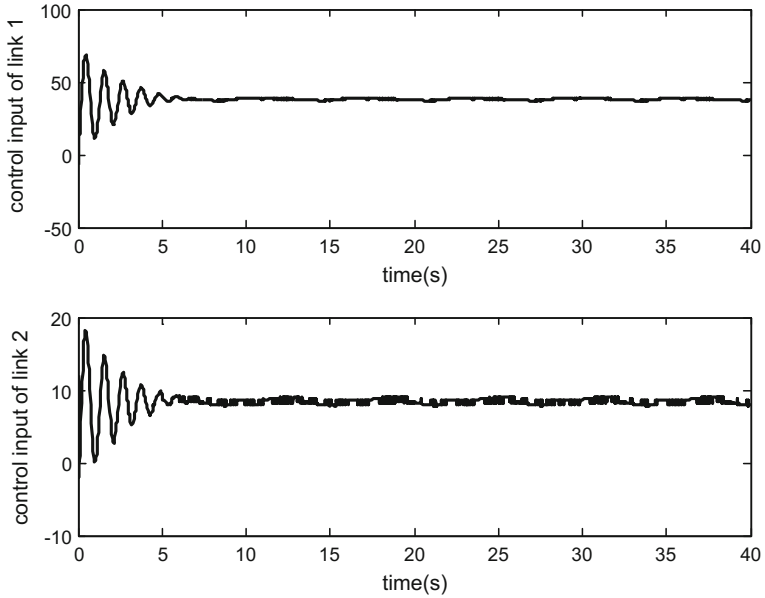


Fig. 9.6 Control input of links 1 and 2

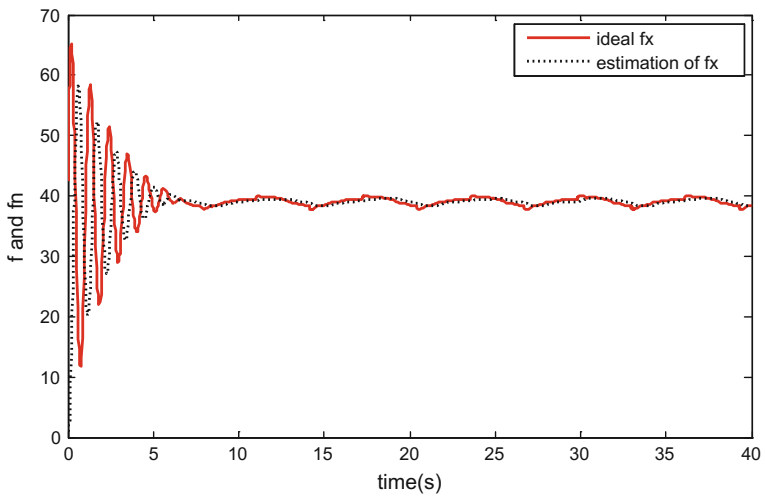
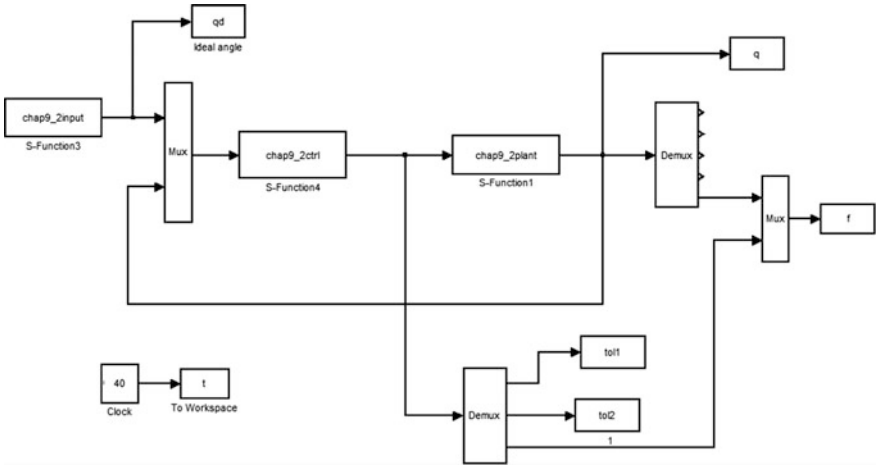


Fig. 9.7 $\|\boldsymbol{f}(\boldsymbol{x})\|$ and $\left\|\hat{\boldsymbol{f}}(\boldsymbol{x})\right\|$

Simulation programs:

1. **Simulink main program: chap9_2sim.mdl**



2. **S function of ideal input: chap9_2input.m**

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 6;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = [];
str = [];
```

```
ts  = [0 0];
function sys=mdlOutputs(t,x,u)
qd1=0.1*sin(t);
d_qd1=0.1*cos(t);
dd_qd1=-0.1*sin(t);
qd2=0.1*sin(t);
d_qd2=0.1*cos(t);
dd_qd2=-0.1*sin(t);

sys(1)=qd1;
sys(2)=d_qd1;
sys(3)=dd_qd1;
sys(4)=qd2;
sys(5)=d_qd2;
sys(6)=dd_qd2;
```

3. S function of control law: chap9_2ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes
global node c b Fai
node=7;
c=[-1.5 -1 -0.5 0 0.5 1 1.5;
   -1.5 -1 -0.5 0 0.5 1 1.5];
b=10;
Fai=5*eye(2);

sizes = simsizes;
sizes.NumContStates  = 2*node;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 3;
sizes.NumInputs      = 11;
```

```
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = zeros(1,2*node);
str = [];
ts  = [];
function sys=mdlDerivatives(t,x,u)
global node c b Fai
qd1=u(1);
d_qd1=u(2);
dd_qd1=u(3);
qd2=u(4);
d_qd2=u(5);
dd_qd2=u(6);

q1=u(7);
d_q1=u(8);
q2=u(9);
d_q2=u(10);

q=[q1;q2];

e1=qd1-q1;
e2=qd2-q2;
de1=d_qd1-d_q1;
de2=d_qd2-d_q2;
e=[e1;e2];
de=[de1;de2];
r=de+Fai*e;

qd=[qd1;qd2];
dqd=[d_qd1;d_qd2];
dqr=dqd+Fai*e;
ddqd=[dd_qd1;dd_qd2];
ddqr=ddqd+Fai*de;

z1=[e(1);de(1)];
z2=[e(2);de(2)];
for j=1:1:node
  h1(j)=exp(-norm(z1-c(:,j))^2/(b*b));
  h2(j)=exp(-norm(z2-c(:,j))^2/(b*b));
end

F=15*eye(node);
for i=1:1:node
  sys(i)=15*h1(i)*r(1);
  sys(i+node)=15*h2(i)*r(2);
```

```
end
function sys=mdlOutputs(t,x,u)
global node c b Fai
qd1=u(1);
d_qd1=u(2);
dd_qd1=u(3);
qd2=u(4);
d_qd2=u(5);
dd_qd2=u(6);

q1=u(7);
d_q1=u(8);
q2=u(9);
d_q2=u(10);

q=[q1;q2];

e1=qd1-q1;
e2=qd2-q2;
de1=d_qd1-d_q1;
de2=d_qd2-d_q2;
e=[e1;e2];
de=[de1;de2];
r=de+Fai*e;

qd=[qd1;qd2];
dqd=[d_qd1;d_qd2];
dqr=dqd+Fai*e;
ddqd=[dd_qd1;dd_qd2];
ddqr=ddqd+Fai*de;

W_f1=[x(1:node)]';
W_f2=[x(node+1:node*2)]';

z1=[e(1);de(1)];
z2=[e(2);de(2)];
for j=1:1:node
  h1(j)=exp(-norm(z1-c(:,j))^2/(b*b));
  h2(j)=exp(-norm(z2-c(:,j))^2/(b*b));
end

fn=[W_f1*h1';
   W_f2*h2'];
Kv=20*eye(2);

epN=0.20;bd=0.1;
v=-(epN+bd)*sign(r);
```

```
tol=fn+Kv*r-v;

fn_norm=norm(fn);
sys(1)=tol(1);
sys(2)=tol(2);
sys(3)=fn_norm;
```

4. S function of plant: chap9_2plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)

switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
global p g
sizes = simsizes;
sizes.NumContStates  = 4;
sizes.NumDiscStates  = 0;
sizes.NumOutputs    = 5;
sizes.NumInputs     =3;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.09 0 -0.09 0];
str=[];
ts=[];

p=[2.9 0.76 0.87 3.04 0.87];
g=9.8;
function sys=mdlDerivatives(t,x,u)
global p g

D=[p(1)+p(2)+2*p(3)*cos(x(3)) p(2)+p(3)*cos(x(3));
    p(2)+p(3)*cos(x(3)) p(2)];
C=[-p(3)*x(4)*sin(x(3)) -p(3)*(x(2)+x(4))*sin(x(3));
    p(3)*x(2)*sin(x(3))  0];
```

```
G=[p(4)*g*cos(x(1))+p(5)*g*cos(x(1)+x(3));
   p(5)*g*cos(x(1)+x(3))];
dq=[x(2);x(4)];
F=0.2*sign(dq);
told=[0.1*sin(t);0.1*sin(t)];

tol=u(1:2);

S=inv(D)*(tol-C*dq-G-F-told);

sys(1)=x(2);
sys(2)=S(1);
sys(3)=x(4);
sys(4)=S(2);
function sys=mdlOutputs(t,x,u)
global p g
D=[p(1)+p(2)+2*p(3)*cos(x(3)) p(2)+p(3)*cos(x(3));
   p(2)+p(3)*cos(x(3)) p(2)];
C=[-p(3)*x(4)*sin(x(3)) -p(3)*(x(2)+x(4))*sin(x(3));
    p(3)*x(2)*sin(x(3)) 0];
G=[p(4)*g*cos(x(1))+p(5)*g*cos(x(1)+x(3));
   p(5)*g*cos(x(1)+x(3))];
dq=[x(2);x(4)];
F=0.2*sign(dq);
told=[0.1*sin(t);0.1*sin(t)];

qd1=sin(t);
d_qd1=cos(t);
dd_qd1=-sin(t);
qd2=sin(t);
d_qd2=cos(t);
dd_qd2=-sin(t);
qd1=0.1*sin(t);
d_qd1=0.1*cos(t);
dd_qd1=-0.1*sin(t);
qd2=0.1*sin(t);
d_qd2=0.1*cos(t);
dd_qd2=-0.1*sin(t);

q1=x(1);
d_q1=dq(1);
q2=x(3);
d_q2=dq(2);
q=[q1;q2];
e1=qd1-q1;
e2=qd2-q2;
```

```
de1=d_qd1-d_q1;
de2=d_qd2-d_q2;
e=[e1;e2];
de=[de1;de2];
Fai=5*eye(2);
dqd=[d_qd1;d_qd2];
dqr=dqd+Fai*e;
ddqd=[dd_qd1;dd_qd2];
ddqr=ddqd+Fai*de;
f=D*ddqr+C*dqr+G+F;
f_norm=norm(f);

sys(1)=x(1);
sys(2)=x(2);
sys(3)=x(3);
sys(4)=x(4);
sys(5)=f_norm;
```

5. Plot program: chap9_2plot.m

```
close all;

figure(1);
subplot(211);
plot(t,qd(:,1),'r',t,q(:,1),'k:','linewidth',2);
xlabel('time(s)');ylabel('Angle tracking for link 1');
legend('ideal angle for link 1','angle tracking for link 1');
subplot(212);
plot(t,qd(:,4),'r',t,q(:,3),'k:','linewidth',2);
xlabel('time(s)');ylabel('Angle tracking for link 2');
legend('ideal angle for link 2','angle tracking for link 2');

figure(2);
subplot(211);
plot(t,qd(:,2),'r',t,q(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking for link 1');
legend('ideal angle speed for link 1','angle speed tracking for link 1');
subplot(212);
plot(t,qd(:,5),'r',t,q(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('Angle speed tracking for link 2');
legend('ideal angle speed for link 2','angle speed tracking for link 2');

figure(3);
subplot(211);
plot(t,tol1(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('control input of link 1');
subplot(212);
```

```
plot(t,tol2(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('control input of link 2');

figure(4);
plot(t,f(:,1),'r',t,f(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('f and fn');
legend('ideal fx','estimation of fx');
```

References

1. P.A. Ioannou, J. Sun, *Robust Adaptive Control*. (PTR Prentice-Hall, 1996), pp. 75–76
2. F.L. Lewis, K. Liu, A. Yesildirek, Neural net robot controller with guaranteed tracking performance. IEEE Trans. Neural Netw. **6**(3), 703–715 (1995)

Chapter 10
Discrete RBF Neural Network Control

The discrete-time implementation of controllers is important. There are two methods for designing the digital controller. One method, called emulation, is to design a controller based on the continuous-time system, then discrete the controller. The other method is to design the discrete controller directly based on the discrete system. In this section, we consider the second approach to design the NN-based nonlinear controller.

Discrete-time adaptive control design is much more complicated than the continuous-time adaptive control design since the discrete-time Lyapunov derivatives tend to have pure and coupling quadratic terms in the states and/or NN weights. There have been many papers to be published about adaptive neural control for discrete-time nonlinear systems [1–4].

10.1 Digital Adaptive RBF Control for a Continuous System

10.1.1 System Description

Consider a simple dynamic system as

$$\ddot{\theta} = f\left(\theta, \dot{\theta}\right) + u \tag{10.1}$$

where θ is angle, u is control input.

Eq. (10.1) can be written as

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x) + u \end{aligned} \tag{10.2}$$

where $f(x)$ is unknown function.

Let the desired output be x_d and denote

$$e = x_1 - x_\mathrm{d},\ \dot{e} = x_2 - \dot{x}_\mathrm{d}$$

Define sliding mode function as

$$s = \lambda e + \dot{e},\ \lambda > 0 \tag{10.3}$$

then

$$\dot{s} = \lambda\dot{e} + \ddot{e} = \lambda\dot{e} + \dot{x}_2 - \ddot{x}_\mathrm{d} = \lambda\dot{e} + f(x) + u - \ddot{x}_\mathrm{d}$$

From (10.3), we can see that if $s \to 0$, then $e \to 0$ and $\dot{e} \to 0$.

10.1.2 RBF Neural Network Approximation

RBF networks are often used to approximate any unknown function [5]. The algorithm of RBF networks is:

$$h_j = \exp\left(\frac{\|\boldsymbol{x} - \boldsymbol{c}_i\|^2}{2b_j^2}\right) \tag{10.4}$$

$$f = \boldsymbol{W}^{*\mathrm{T}}\boldsymbol{h}(\mathbf{x}) + \varepsilon \tag{10.5}$$

where $\boldsymbol{x}$ is the input signal of the network, i is the input number of the network, j is the number of hidden layer nodes in the network, $h = [h_1, h_2, \ldots, h_n]^\mathrm{T}$ is the output of Gaussian function, $\boldsymbol{W}^*$ is the ideal neural network weight value, ε is the approximation error of neural network, and $|\varepsilon| \le \varepsilon_\mathrm{N}$.

If we use RBF network to approximate $f(x)$, the network input is selected as $\boldsymbol{x} = [x_1 \quad x_2]^\mathrm{T}$, and output of RBF neural network is

$$\hat{f}(x) = \hat{\boldsymbol{W}}^\mathrm{T}\boldsymbol{h}(x) \tag{10.6}$$

10.1.3 Adaptive Controller Design

Define Lyapunov function as

$$V = \frac{1}{2}s^2 + \frac{1}{2\gamma}\tilde{\boldsymbol{W}}^{\mathrm{T}}\tilde{\boldsymbol{W}} \tag{10.7}$$

where $\gamma > 0$, $\tilde{\boldsymbol{W}} = \hat{\boldsymbol{W}} - \boldsymbol{W}^*$.

Since $f(x) - \hat{f}(x) = \boldsymbol{W}^{*\mathrm{T}}\boldsymbol{h}(\mathbf{x}) + \varepsilon - \hat{\boldsymbol{W}}^{\mathrm{T}}\mathbf{h}(x) = -\tilde{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) + \varepsilon$, then

$$\dot{V} = s\dot{s} + \frac{1}{\gamma}\tilde{W}^{\mathrm{T}}\dot{\hat{W}} = s(\lambda\dot{e} + f(x) + u - \ddot{x}_{\mathrm{d}}) + \frac{1}{\gamma}\tilde{\boldsymbol{W}}^{\mathrm{T}}\dot{\hat{W}}$$

Design sliding mode controller as

$$u = -\lambda\dot{e} - \hat{f}(x) + \ddot{x}_{\mathrm{d}} - \eta\mathrm{sgn}(s) \tag{10.8}$$

then

$$\begin{aligned}\dot{V} &= s\big(f(x) - \hat{f}(x) - \eta\mathrm{sgn}(s)\big) + \frac{1}{\gamma}\tilde{\boldsymbol{W}}^{\mathrm{T}}\dot{\hat{W}} \\ &= s\big(-\tilde{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{h}(\mathbf{x}) + \varepsilon - \eta\mathrm{sgn}(s)\big) + \frac{1}{\gamma}\tilde{\boldsymbol{W}}^{\mathrm{T}}\dot{\hat{W}} \\ &= \varepsilon s - \eta|s| + \tilde{W}^{\mathrm{T}}\left(\frac{1}{\gamma}\dot{\hat{W}} - s\boldsymbol{h}(\boldsymbol{x})\right)\end{aligned}$$

Design adaptive law as

$$\dot{\hat{\boldsymbol{W}}} = \gamma s\boldsymbol{h}(\boldsymbol{x}) \tag{10.9}$$

If we choose $\eta > |\varepsilon|_{\max}$, then $\dot{V} = \varepsilon s - \eta|s| \le 0$.

From above analysis, we can see that RBF approximation error can be overcome by the robust term $\eta\mathrm{sgn}(s)$.

When $\dot{V} \equiv 0$, we have $s \equiv 0$; according to LaSalle invariance principle, the closed system is asymptotically stable, $s \to 0$ as $t \to \infty$, and the convergence precision is related to η.

Since $V \ge 0$, $\dot{V} \le 0$, V is limited as $t \to \infty$, thus $\hat{\boldsymbol{W}}$ is limited. Since when $\dot{V} \equiv 0$, we cannot get $\tilde{\boldsymbol{W}} \equiv 0$, $\hat{\boldsymbol{W}}$, we cannot converge to $\boldsymbol{W}^*$.

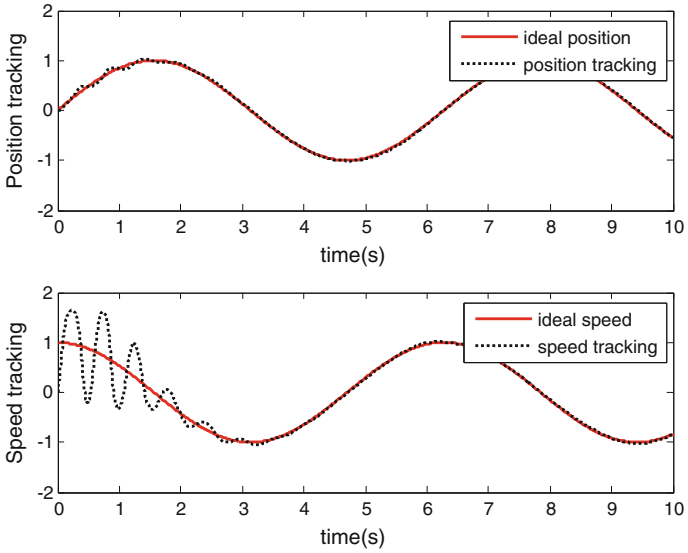


Fig. 10.1 Position and speed tracking

10.1.4 Simulation Example

Consider a plant as

$$\begin{aligned}\dot{x}_1 &= x_2\\ \dot{x}_2 &= f(x)+u\end{aligned}$$

where $f(x) = 10x_1x_2$.

The desired trajectory is chosen as $x_\mathrm{d} = \sin t$. The initial state of the plant is set as $[0.5\,0]$. We adapt control law as (10.8) and adaptive law as (10.9), choose $\lambda = 200$, $\eta = 0.20$, and $\gamma = 100$.

The structure of RBF is chosen as 2-5-1. Consider the range of x_1 and x_2, we choose $\boldsymbol{c}_i = [-1 \quad -0.5 \quad 0 \quad 0.5 \quad 1]$, $b_j = 3.0$, and the initial value of each element of RBF weight matrix is set as zero.

The continuous system simulation is chap11_1sim.mdl. If we discrete the control law (10.8) and adaptive law (10.9), and denote the sampling time as ts = 0.001, simulation results are shown from Figs. 10.1 and 10.2.

The shortcoming of the digital adaptive RBF control simulation for a continuous system is that the stability cannot be guaranteed [6]. To overcome this problem, controller design and stability analysis for discrete system directly is needed.

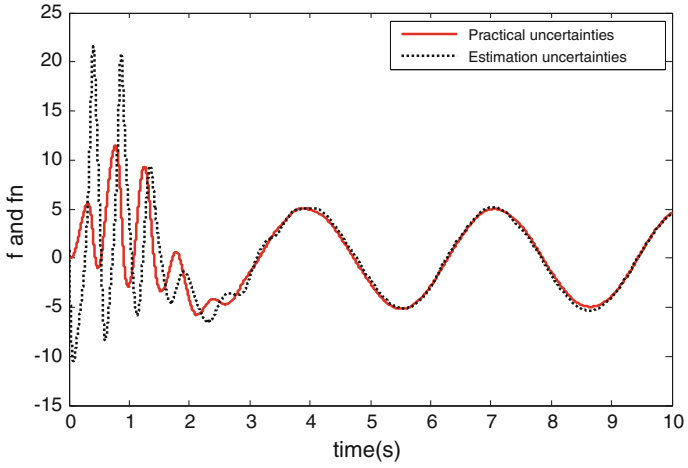
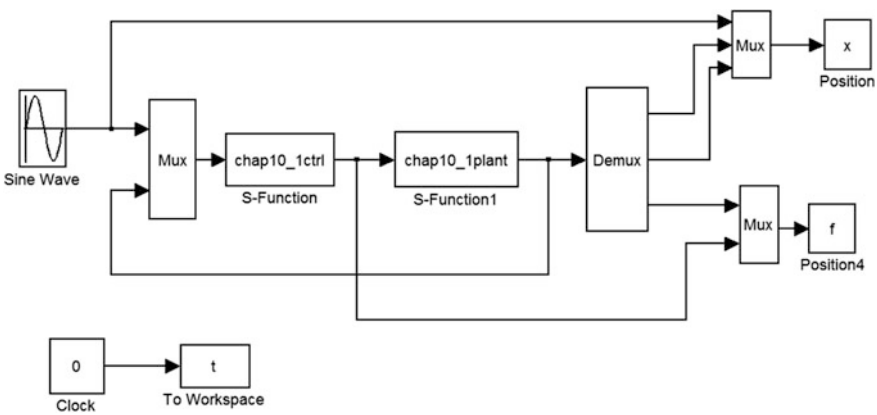


Fig. 10.2 $f(x)$ and $\hat{f}(x)$

Matlab Programs:

1. Continuous simulation programs

(1) Main Simulink program: chap10_1sim.mdl



(2) S function program of controller: chap10_1ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
```

```
   [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
   sys=mdlDerivatives(t,x,u);
case 3,
   sys=mdlOutputs(t,x,u);
case {2,4,9}
   sys=[];
otherwise
   error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
global b c namna
sizes = simsizes;
sizes.NumContStates  = 5;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 4;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = rands(1,5);
str = [];
ts  = [0 0];
c=[-1 -0.5 0 0.5 1;
  -1 -0.5 0 0.5 1];
b=1.2;
namna=10;
function sys=mdlDerivatives(t,x,u)
global b c namna
xd=sin(t);
dxd=cos(t);

x1=u(2);
x2=u(3);
e=x1-xd;
de=x2-dxd;
s=namna*e+de;

W=[x(1) x(2) x(3) x(4) x(5)]';
xi=[x1;x2];

h=zeros(5,1);
for j=1:1:5
   h(j)=exp(-norm(xi-c(:,j))^2/(2*b^2));
end
```

```
gama=100;
for i=1:1:5
  sys(i)=gama*s*h(i);
end

function sys=mdlOutputs(t,x,u)
global b c namna
xd=sin(t);
dxd=cos(t);
ddxd=-sin(t);

x1=u(2);
x2=u(3);
e=x1-xd;
de=x2-dxd;
s=namna*e+de;

W=[x(1) x(2) x(3) x(4) x(5)];
xi=[x1;x2];

h=zeros(5,1);
for j=1:1:5
  h(j)=exp(-norm(xi-c(:,j))^2/(2*b^2));
end
fn=W*h;
xite=0.20;

%fn=10*x1+x2; %Precise f
ut=-namna*de+ddxd-fn-xite*sign(s);

sys(1)=ut;
sys(2)=fn;
```

(3) S function program of plant: chap10_1plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
  [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
  sys=mdlDerivatives(t,x,u);
case 3,
  sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
  sys = [];
otherwise
```

```
  error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 3;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.15;0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
ut=u(1);

f=10*x(1)*x(2);
sys(1)=x(2);
sys(2)=f+ut;
function sys=mdlOutputs(t,x,u)
f=10*x(1)*x(2);

sys(1)=x(1);
sys(2)=x(2);
sys(3)=f;
```

(4) Plot program: chap10_1plot.m

```
close all;

figure(1);
subplot(211);
plot(t,x(:,1),'r',t,x(:,2),'b','linewidth',2);
xlabel('time(s)');ylabel('position tracking');
subplot(212);
plot(t,cos(t),'r',t,x(:,3),'b','linewidth',2);
xlabel('time(s)');ylabel('speed tracking');

figure(2);
plot(t,f(:,1),'r',t,f(:,3),'b','linewidth',2);
xlabel('time(s)');ylabel('f approximation');
```

2. Digital simulation program

(1) Main program: chap10_2.m

```
%Discrete RBF control
clear all;
close all;
ts=0.001; %Sampling time

node=5;   %Number of neural nets in hidden layer
gama=100;

c=[-1 -0.5 0 0.5 1;
  -1 -0.5 0 0.5 1];
bj=1.2;
h=zeros(node,1);
x1_1=0;x2_1=0;u_1=0;

xk=[0.10 0];
w_1=rands(node,1);
namna=10;
xite=0.20;
for k=1:1:10000
time(k)=k*ts;

xd(k)=sin(k*ts);
dxd(k)=cos(k*ts);
ddxd(k)=-sin(k*ts);

tSpan=[0 ts];
para=u_1;         %D/A
[t,xx]=ode45('chap10_2plant',tSpan,xk,[],para);  %Plant
xk=xx(length(xx),:);  %A/D

x1(k)=xk(1);
x2(k)=xk(2);

e(k)=x1(k)-xd(k);
de(k)=x2(k)-dxd(k);
s(k)=namna*e(k)+de(k);

xi=[x1(k);x2(k)];
for i=1:1:node
  w(i,1)=w_1(i,1)+ts*(gama*s(k)*h(i)); %Adaptive law
```

```
end
h=zeros(5,1);
for j=1:1:5
  h(j)=exp(-norm(xi-c(:,j))^2/(2*bj*bj));
end
fn(k)=w'*h;
u(k)=-namna*de(k)-fn(k)+ddxd(k)-xite*sign(s(k));

f(k)=10*x1(k)*x2(k);

x1_1=x1(k);
x2_1=x2(k);
w_1=w;
u_1=u(k);
end
figure(1);
subplot(211);
plot(time,xd,'r',time,x1,'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal position','position tracking');
subplot(212);
plot(time,dxd,'r',time,x2,'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('ideal speed','speed tracking');

figure(2);
plot(time,u,'r','linewidth',2);
xlabel('time(s)'),ylabel('Control input of single link');

figure(3);
plot(time,f,'r',time,fn,'k:','linewidth',2);
xlabel('time(s)'),ylabel('f and fn');
legend('Practical uncertainties','Estimation uncertainties');
```

(2) **Program of plant: chap10_2plant.m**

```
function dx=Plant(t,x,flag,para)
dx=zeros(2,1);

u=para;

f=10*x(1)*x(2);
dx(1)=x(2);
dx(2)=f+u;
```

10.2 Adaptive RBF Control for a Class of Discrete-Time Nonlinear System

10.2.1 System Description

Consider a nonlinear discrete system as follows:

$$y(k+1) = f(\boldsymbol{x}(k)) + u(k) \tag{10.10}$$

where $\boldsymbol{x}(k) = [\, y(k) \quad y(k-1) \quad \cdots \quad y(k-n+1) \,]^{\mathrm{T}}$ is the state vector, $u(k)$ is the control input, and $y(k)$ is the plant output. The nonlinear smooth function $f : R^n \rightarrow R$ is assumed unknown.

10.2.2 Traditional Controller Design

The tracking error $e(k)$ is defined as $e(k) = y(k) - y_{\mathrm{d}}(k)$. If $f(\boldsymbol{x}(k))$ is known, a feedback linearization-type control law can be designed as

$$u(k) = y_{\mathrm{d}}(k+1) - f(\boldsymbol{x}(k)) - c_1 e(k) \tag{10.11}$$

Submitting (10.11) into (10.10), we can get an asymptotical convergence error dynamic system as

$$e(k+1) + c_1 e(k) = 0 \tag{10.12}$$

where $|c_1| < 1$.

10.2.3 Adaptive Neural Network Controller Design

If $f(\boldsymbol{x}(k))$ is unknown, and RBF neural network can be used to approximate $f(\boldsymbol{x}(k))$. The network output is given as

$$\hat{f}(\boldsymbol{x}(k)) = \hat{\boldsymbol{w}}(k)^{\mathrm{T}} \boldsymbol{h}(\boldsymbol{x}(k)) \tag{10.13}$$

where $\hat{\boldsymbol{w}}(k)$ denotes the network output weight vector, and $\boldsymbol{h}(\boldsymbol{x}(k))$ denotes the vector of Gaussian basis functions.

Given any arbitrary nonzero approximation error bound ε_f, there exist some optimal weight vector $\boldsymbol{w}^*$ such that

$$f(\boldsymbol{x}) = \hat{f}(\boldsymbol{x}, \boldsymbol{w}^*) - \Delta_f(\boldsymbol{x}) \tag{10.14}$$

where $\Delta_f(\boldsymbol{x})$ denotes the optimal network approximation error, and $\left|\Delta_f(\boldsymbol{x})\right| < \varepsilon_f$.

Then we can get the general network approximation error as

$$\begin{aligned} \tilde{f}(\boldsymbol{x}(k)) &= f(\boldsymbol{x}(k)) - \hat{f}(\boldsymbol{x}(k)) \\ &= \hat{f}(\boldsymbol{x}, \boldsymbol{w}^*) - \Delta_f(\boldsymbol{x}(k)) - \hat{\boldsymbol{w}}(k)^{\mathrm{T}} \boldsymbol{h}(\boldsymbol{x}(k)) \\ &= -\tilde{\boldsymbol{w}}(k)^{\mathrm{T}} \boldsymbol{h}(\boldsymbol{x}(k)) - \Delta_f(\boldsymbol{x}(k)) \end{aligned} \tag{10.15}$$

where $\tilde{\boldsymbol{w}}(k) = \hat{\boldsymbol{w}}(k) - \boldsymbol{w}^*$.

The control law with RBF approximation can be designed in [7] as follows

$$u(k) = y_{\mathrm{d}}(k+1) - \hat{f}(\boldsymbol{x}(k)) - c_1 e(k) \tag{10.16}$$

Figure 10.3 shows the closed-loop neural-based adaptive control scheme. Substituting (10.16) into (10.10) yields

$$e(k+1) = \tilde{f}(\boldsymbol{x}(k)) - c_1 e(k)$$

Thus

$$e(k) + c_1 e(k-1) = \tilde{f}(\boldsymbol{x}(k-1)) \tag{10.17}$$

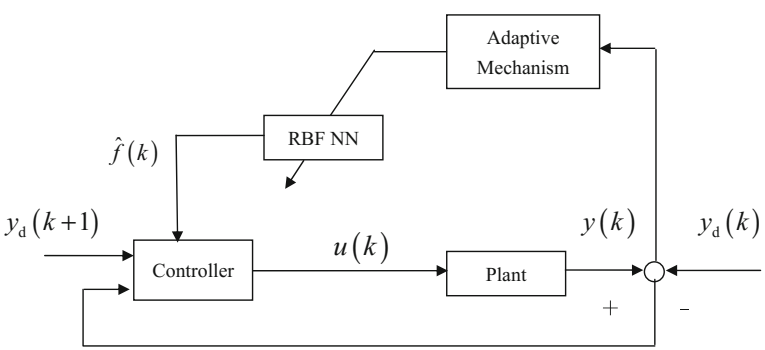


Fig. 10.3 Block diagram of the control scheme

The term (10.17) can also be expressed as

$$e(k) = \Gamma^{-1}\left(z^{-1}\right)\tilde{f}(\boldsymbol{x}(k-1)) \tag{10.18}$$

Refer to [7], we can where $\Gamma(z^{-1}) = 1 + c_1 z^{-1}, z^{-1}$ denotes the discrete-time delay operator.

Define a new augmented error as

$$e_1(k) = \beta\left(e(k) - \Gamma^{-1}\left(z^{-1}\right)v(k)\right) \tag{10.19}$$

where $\beta > 0$.

Substituting (10.18) into (10.19) yields

$$\begin{aligned} e_1(k) &= \beta\Gamma^{-1}\left(z^{-1}\right)\left(\tilde{f}(\boldsymbol{x}(k-1)) - v(k)\right) \\ &= \beta\frac{1}{1 + c_1 z^{-1}}\left(\tilde{f}(\boldsymbol{x}(k-1)) - v(k)\right) \end{aligned}$$

Which leads to the relation as

$$e_1(k-1) = \frac{\beta\left(\tilde{f}(\boldsymbol{x}(k-1)) - v(k)\right) - e_1(k)}{c_1} \tag{10.20}$$

Refer to [7] the adaptive law as can be designed as

$$\Delta\hat{\mathbf{w}}(k) = \begin{cases} \frac{\beta}{\gamma c_1^2}\mathbf{h}(\mathbf{x}(k-1))e_1(k) & \text{if } |e_1(k)| > \varepsilon_f/\mathrm{G} \\ 0 & \text{if } |e_1(k)| \le \varepsilon_f/\mathrm{G} \end{cases} <!endarray > \tag{10.21}$$

where $\Delta\hat{\boldsymbol{w}}(k) = \hat{\boldsymbol{w}}(k) - \hat{\boldsymbol{w}}(k-1)$, γ, and G are strictly positive constants.

10.2.4 Stability Analysis

For the closed system, the discrete-time Lyapunov function can be designed as

$$V(k) = e_1^2(k) + \gamma\tilde{\boldsymbol{w}}^{\mathrm{T}}(k)\tilde{\boldsymbol{w}}(k) \tag{10.22}$$

The first difference is

$$\begin{aligned} \Delta V(k) &= V(k) - V(k-1) \\ &= e_1^2(k) - e_1^2(k-1) + \gamma\left(\tilde{\boldsymbol{w}}^{\mathrm{T}}(k) + \tilde{\boldsymbol{w}}^{\mathrm{T}}(k-1)\right)\left(\tilde{\boldsymbol{w}}(k) - \tilde{\boldsymbol{w}}(k-1)\right) \end{aligned}$$

The stability proof is given with the following three steps. Firstly, using (10.20) for $e_1(k-1)$, it follows that

$$\Delta V(k) = e_1^2(k) - \frac{e_1^2(k) + \beta^2\big(\tilde{f}(\boldsymbol{x}(k-1)) - v(k)\big)^2 - 2\beta\big(\tilde{f}(\boldsymbol{x}(k-1)) - v(k)\big)e_1(k)}{c_1^2}$$

$$= -V_1 + \frac{2\beta\big(\tilde{f}(\boldsymbol{x}(k-1)) - v(k)\big)e_1(k)}{c_1^2} + \gamma\big(\Delta\hat{\boldsymbol{w}}^{\mathrm{T}}(k) + 2\tilde{\boldsymbol{w}}^{\mathrm{T}}(k-1)\big)\Delta\hat{\boldsymbol{w}}(k)$$

where $V_1 = \frac{e_1^2(k)\left(1-c_1^2\right)}{c_1^2} + \frac{\beta^2\left(\tilde{f}(\boldsymbol{x}(k-1)) - v(k)\right)^2}{c_1^2} \geq 0$

Secondly, substituting for $\tilde{f}(\boldsymbol{x}(k-1))$ via (10.15) yields:

$$\begin{aligned}\Delta V(k) &= -V_1 + \frac{2\beta\Big(-\tilde{\boldsymbol{w}}(k-1)^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}(k-1)) - \Delta_f(\boldsymbol{x}(k-1)) - v(k)\Big)e_1(k)}{c_1^2} \\ &\quad + \gamma\Delta\hat{\boldsymbol{w}}^{\mathrm{T}}(k)\Delta\hat{\boldsymbol{w}}(k) + 2\gamma\tilde{\boldsymbol{w}}^{\mathrm{T}}(k-1)\Delta\hat{\boldsymbol{w}}(k) \\ &= -V_1 + 2\tilde{\boldsymbol{w}}^{\mathrm{T}}(k-1)\left(\gamma\Delta\hat{\boldsymbol{w}}(k) - \frac{\beta}{c_1^2}\boldsymbol{h}(\boldsymbol{x}(k-1))e_1(k)\right) \\ &\quad - \frac{2\beta}{c_1^2}\big(\Delta_f(\boldsymbol{x}(k-1)) + v(k)\big)e_1(k) + \gamma\Delta\hat{\boldsymbol{w}}^{\mathrm{T}}(k)\Delta\hat{\boldsymbol{w}}(k)\end{aligned}$$

Thirdly, substituting the adaptive law (10.21) into above, $\Delta V(k)$ is

$$\Delta V(k) = \begin{cases} -V_1 - \frac{2\beta}{c_1^2}\big(\Delta_f(\boldsymbol{x}(k-1)) + v(k)\big)e_1(k) + \\ \left(\frac{\beta}{\sqrt{\gamma}c_1^2}\right)^2 \boldsymbol{h}^{\mathrm{T}}(\boldsymbol{x}(k-1))\boldsymbol{h}(\boldsymbol{x}(k-1))e_1^2(k), \quad \text{if } |e_1(k)| > \varepsilon_f/G \\ -V_1 - \frac{2\beta}{c_1^2}\big[\big(\big(\tilde{\boldsymbol{w}}^{\mathrm{T}}(k-1)\boldsymbol{h}(\boldsymbol{x}(k-1))\big) + \\ v(k) + \Delta_f(\boldsymbol{x}(k-1))e_1(k)\big], \quad \text{if } |e_1(k)| \leq \varepsilon_f/G \end{cases} \tag{10.23}$$

The auxiliary signal $v(k)$ must also be designed so that $e_1(k) \to 0$ could deduce $e(k) \to 0$. The auxiliary term is designed as [7]

$$v(k) = v_1(k) + v_2(k) \tag{10.24}$$

with $v_1(k) = \frac{\beta}{2\gamma c_1^2}\boldsymbol{h}^T(\boldsymbol{x}(k-1))\boldsymbol{h}(\boldsymbol{x}(k-1))e_1(k)$ and $v_2(k) = Ge_1(k)$.

If $|e_1(k)| > \varepsilon_f/G$, substituting for $v(k)$ in (10.19)–(10.17), it follows that

$$\begin{aligned}\Delta V(k) &= -V_1 - \frac{2\beta}{c_1^2}\big(\Delta_f(\boldsymbol{x}(k-1)) + Ge_1(k)\big)e_1(k)\\ &\le -\frac{2\beta}{c_1^2}\big(\Delta_f(\boldsymbol{x}(k-1)) + Ge_1(k)\big)e_1(k)\end{aligned}$$

Since $\big|\Delta_f(\boldsymbol{x})\big| < \varepsilon_f$ and $|e_1(k)| > \varepsilon_f/G$, then $|e_1(k)| > \frac{|\Delta_f(\boldsymbol{x}(k-1))|}{G}$ and $e_1^2(k) > -\frac{\Delta_f(\boldsymbol{x}(k-1))e_1(k)}{G}$, thus $\big(\Delta_f(\boldsymbol{x}(k-1)) + Ge_1(k)\big)e_1(k) > 0$, then $\Delta V(k) < 0$.

If $|e_1(k)| \le \varepsilon_f/G$, tracking performance can be satisfied, and $\Delta V(k)$ can be taken on any value.

In the simulation, we give three remarks as follows:

Remark 1 From (10.19), we have $e_1(k) = \beta\Big(e(k) - \frac{1}{1+c_1z^{-1}}v(k)\Big)$, then $e_1(k)(1+c_1z^{-1}) = \beta(e(k)(1+c_1z^{-1}) - v(k))$, therefore

$$e_1(k) = -c_1e_1(k-1) + \beta(e(k) + c_1e(k-1) - v(k)) \tag{10.25}$$

Remark 2 From Lyapunov analysis, if $k \to \infty$, $e_1(k) \to 0$, from (10.24) we have $v(k) \to 0$, then from (10.25), $e(k) + c_1e(k-1) \to 0$, consider $|c_1| < 1$, and we get $e(k) \to 0$.

Remark 3 Consider $v(k)$ is a virtual variable, for (10.24); let $v_1'(k) = \frac{\beta}{2\gamma c_1^2}\boldsymbol{h}^{\mathrm{T}}(\boldsymbol{x}(k-1))\boldsymbol{h}(\boldsymbol{x}(k-1))$, then we get $v(k) = \big(v_1'(k) + G\big)e_1(k)$, substituting $v(k)$ into (10.25), we have $e_1(k) = -c_1e_1(k-1) + \beta\big(e(k) + c_1e(k-1) - \big(v_1'(k) + G\big) \times e_1(k)\big)$, then

$$e_1(k) = \frac{-c_1e_1(k-1) + \beta(e(k) + c_1e(k-1))}{1 + \beta\big(v_1'(k) + G\big)} \tag{10.26}$$

10.2.5 Simulation Examples

Consider a nonlinear discrete-time system as

$$y(k) = \frac{0.5y(k-1)(1-y(k-1))}{1+\exp(-0.25y(k-1))} + u(k-1)$$

where $f(x(k-1)) = \frac{0.5y(k-1)(1-y(k-1))}{1+\exp(-0.25y(k-1))}$.

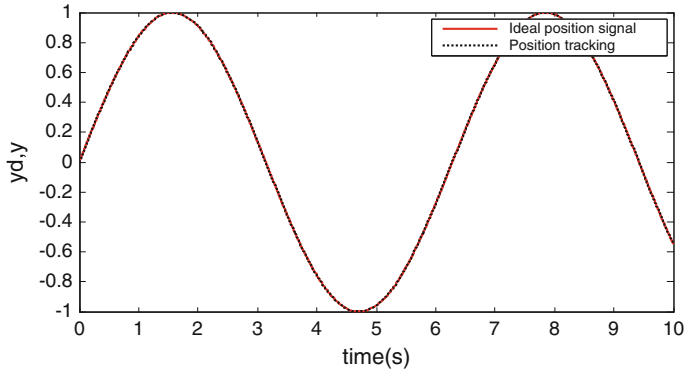


Fig. 10.4 Position tracking

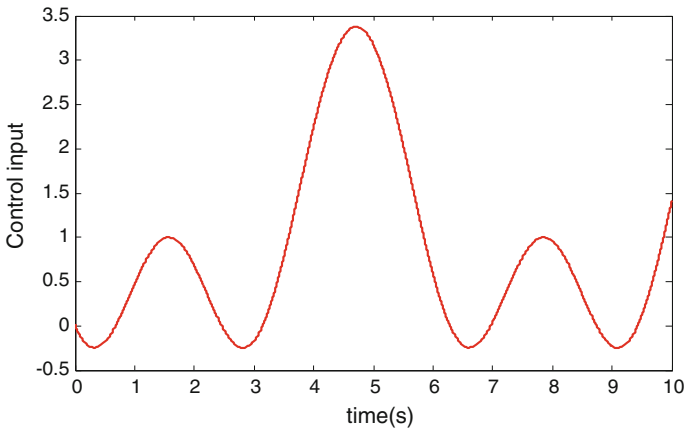


Fig. 10.5 Control input

Firstly, we assume $f(x(k-1))$ is known, use the control law (10.11), and set $c_1 = -0.01$; the results are shown in Figs. 10.4 and 10.5. Then we use RBF to approximate $f(x(k-1))$. For RBF neural network, the structure is 1-9-1, and from $f(x(k-1))$ expression, only one input $y(k-1)$ is chosen; the parameters of Gaussian function $\mathbf{c}_i$ and b_j as chosen as $[-2\ \ -1.5\ \ -1.0\ \ -0.5\ \ 0\ \ 0.5\ \ 1.0\ \ 1.5\ \ 2]$ and $15(i=1, j=1,2,\ldots,9)$, the initial weight value is chosen as random value in the range $(0,1)$. The initial value of the plant is set as zero. The reference signal is $y_d(k) = \sin t$. Using the control law (10.16) with adaptive law (10.21), $e_1(k)$ is calculated by (10.26), and the parameters are chosen as $c_1 = -0.01, \beta = 0.001, \gamma = 0.001, \gamma = 0.001, G = 50000, \varepsilon_f = 0.003$. The results are shown in Figs. 10.6, 10.7, and 10.8. The program of this example is chap10_3.m, which is given in the Appendix.

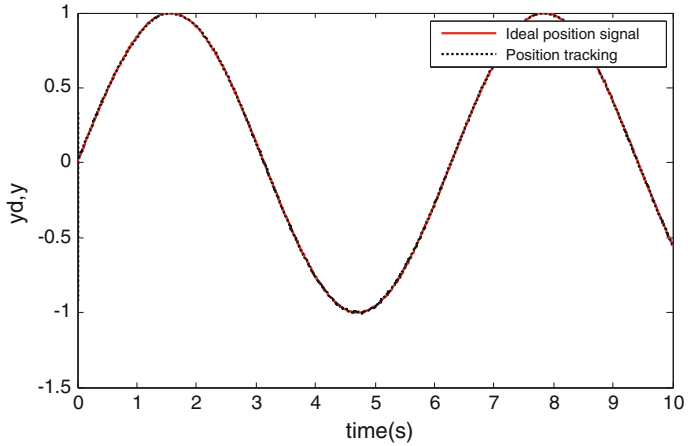


Fig. 10.6 Position tracking

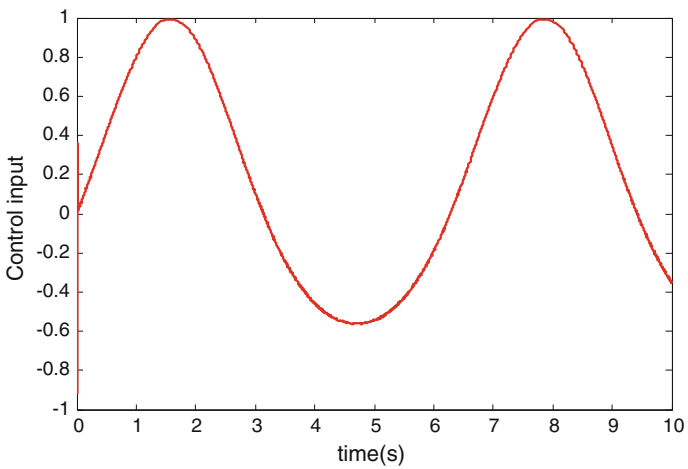


Fig. 10.7 Control input

Simulation program with unknown f(x(k-1)): chap10_3.m

```
%Discrete RBF controller
clear all;
close all;
ts=0.001;
```

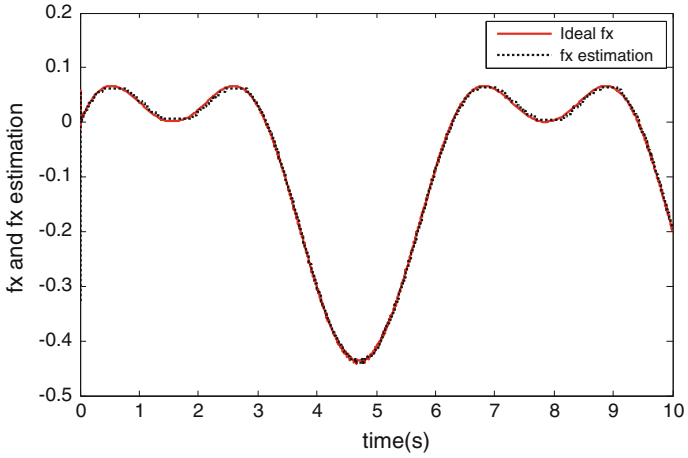


Fig. 10.8 $f(x(k-1))$ and its estimation

```
c1=-0.01;
beta=0.001;
epcf=0.003;
gama=0.001;
G=50000;

b=15;
c=[-2 -1.5 -1 -0.5 0 0.5 1 1.5 2];
w=rands(9,1);
w_1=w;

u_1=0;
y_1=0;
e1_1=0;
e_1=0;
fx_1=0;
for k=1:1:10000
time(k)=k*ts;

yd(k)=sin(k*ts);
yd1(k)=sin((k+1)*ts);
%Nonlinear plant
fx(k)=0.5*y_1*(1-y_1)/(1+exp(-0.25*y_1));
y(k)=fx_1+u_1;

e(k)=y(k)-yd(k);

x(1)=y_1;
for j=1:1:9
```

```
  h(j)=exp(-norm(x-c(:,j))^2/(2*b^2));
end
v1_bar(k)=beta/(2*gama*c1^2)*h*h';

e1(k)=(-c1*e1_1+beta*(e(k)+c1*e_1))/(1+beta*(v1_bar(k)+G));

if abs(e1(k))>epcf/G
  w=w_1+beta/(gama*c1^2)*h'*e1(k);
elseif abs(e1(k))<=epcf/G
  w=w_1;
end
fnn(k)=w'*h';

u(k)=yd1(k)-fnn(k)-c1*e(k);
%u(k)=yd1(k)-fx(k)-c1*e(k); %With precise fx

fx_1=fx(k);
y_1=y(k);

w_1=w;
u_1=u(k);
e1_1=e1(k);
e_1=e(k);
end
figure(1);
plot(time,yd,'r',time,y,'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('Ideal position signal','Position tracking');
figure(2);
plot(time,u,'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');
figure(3);
plot(time,fx,'r',time,fnn,'k:','linewidth',2);
xlabel('time(s)');ylabel('fx and fx estimation');
legend('Ideal fx','fx estimation');
```

References

1. S. Jagannathan, F.L. Lewis. Discrete-time neural net controller with guaranteed performance. *in Proceedings American Control Conference*, (1994) pp. 3334–3339
2. S.S. Ge, C. Yang, S. Dai, Z. Jiao, T.H. Lee, Robust adaptive control of a class of nonlinear strict-feedback discrete-time systems with exact output tracking. Automatica **45**(11), 2537–2545 (2009)
3. C. Yang, S.S. Ge, T.H. Lee, Output feedback adaptive control of a class of nonlinear discrete-time systems with unknown control directions. Automatica **45**(1), 270–276 (2009)

4. C. Yang, S.S. Ge, C. Xiang, T. Chai, T.H. Lee, Output feedback NN control for two classes of discrete-time systems with unknown control directions in a unified approach. IEEE Trans. Neural Networks **19**(11), 1873–1886 (2008)
5. J. Park, I.W. Sandberg, Universal approximation using radial-basis-function networks. Neural Comput. **3**(2), 246–257 (1991)
6. J.K. Liu, *RBF Neural Network Control for Mechanical Systems_Design, Analysis and Matlab Simulation*. (Tsinghua and Springer Press, 2013)
7. S.G. Fabri, V. Kadirkamanathan. *Functional Adaptive Control: An Intelligent Systems Approach* (Springer, New York, 2001)

Chapter 11
Intelligent Search Algorithm Design

With the development of the optimization theory, some new intelligent algorithms have been rapidly developed and widely used, and these algorithms have become new methods to solve the traditional system identification problems, such as genetic algorithm, ant colony algorithm, particle swarm optimization algorithm, differential evolution algorithm. These optimization algorithms simulate natural phenomena and processes.

11.1 GA and Design

11.1.1 Principle of GA

The basic principle of GA(genetic algorithms) were first laid down by Holland in 1962. GA simulate those processes in natural populations that are essential to evolution.

Some common definitions of the technical terms used are described below:

- Chromosome is a vector of parameters which represents the solution of an application task, for example, the dimensions of the beams in a bridge design. These parameters, known as genes, are joined together to form a string of values called chromosomes.
- Gene is a solution which will combine to form a chromosome.
- Selection is the process of choosing parents or offspring chromosome for the next generation.
- Individuals are the solution vectors of chromosome.
- Population is the collection of individuals.
- Population size is the number of chromosome in a population.
- Fitness function is the function which evaluates how each solution is suitable for a given task.

- Phenotype defines the expression type of solution values in the task world, for example, “red,” “blue,” “80 kg”.
- Genotype are the binary (bit) expression type of solution values used in the GA search space, for example, “011”, “000111011”.

Some advantages of GA are the following:

(1) Fast convergence to near global optimum;
(2) Superior global searching capability in a space that has a complex searching surface;
(3) Applicability to a searching space where one cannot use gradient information of the space.

A GA determines the next set of searching points using the fitness values of the current searching points, which are widely distributed throughout the searching space. It uses the mutation operator to escape from a local minimum. A key disadvantage of GA is that their convergence speed near the global optimum can be quite slow.

11.1.2 Steps of GA Design

GA use a direct analogy of natural behavior (see Fig. 11.1). They work with a population of individuals, each reprinting a possible solution to a given problem. Each individual is assigned a fitness score according to how good its solution to the problem is. The highly fit individuals are given opportunities to reproduce, by crossbreeding with other individuals in the population. This produces new individuals as offspring, who share some features taken from each parent. The least fit members of the population are less likely to get selected for reproduction and will eventually die out.

The standard GA algorithm mainly includes the following four operators:

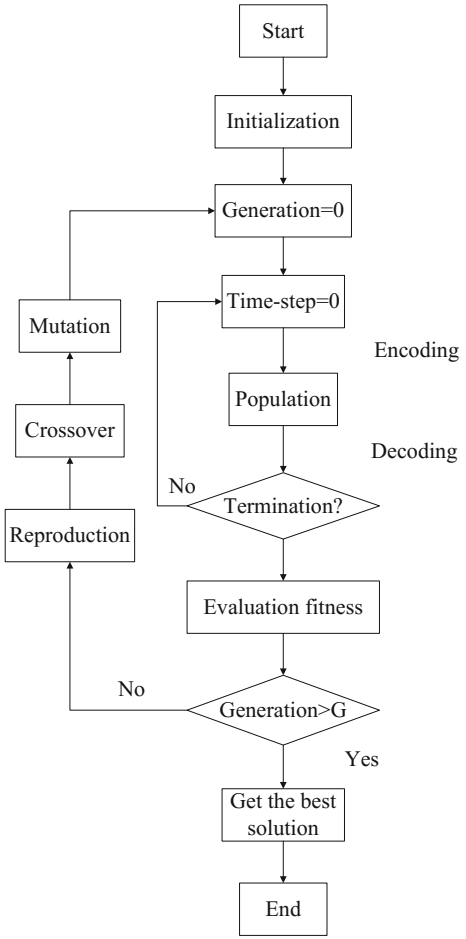
(1) GA Selection

Selection is an operation that will choose parent solutions. New solution vectors in the next generation are calculated from them. Since it is expected that better parent generator generates better offspring, parent solution vectors that have higher fitness values will have a higher probability to be selected. There are several selection methods. The roulette wheel selection is a typical selection method.

(2) GA Reproduction

During the reproductive phase of a GA, individuals are selected from the population and recombined, producing offspring which, in turn, will comprise the next generation. Parents are selected randomly from the population using a scheme that favors the more fit individuals. Having selected two parents, their chromosomes are recombined using the mechanism of crossover and mutation.

Fig. 11.1 Flowchart of GA



(3) Crossover takes two individuals and cuts their chromosome strings at some randomly chosen positions, to produce two "head" segments and two "tail" segments. The tail segments are then swapped over to produce two new full-length chromosomes (see Fig. 11.2). Each of the two offspring will inherit some genes from each parent. This is known as a single-point crossover. Crossover is not usually applied to all pairs of individuals that are chosen for mating. A random choose is made, where the likelihood of crossover being applied is typically between 0.6 and 1.0.

(4) Mutation is applied to each child individually, after crossover. It randomly alters each gene with a small probability (typically 0.001). Figure 11.3 shows the fifth gene of the chromosome being mutated. The traditional view is that crossover is the more important of the two techniques for rapidly exploring a search space.

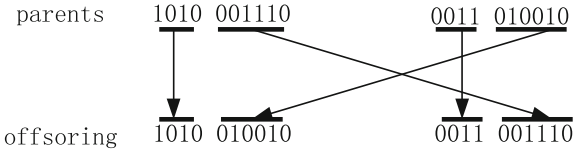


Fig. 11.2 Single-point crossover

Mutation provides a small amount of random search and helps ensures that no point in the search space has zero probability of being examined.

11.1.3 Simulation Example

Using GA to get maximum value of Rosenbrock function,

$$\begin{cases} f_2(x_1, x_2) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 \\ -2.048 \le x_i \le 2.048 \qquad (i = 1, 2) \end{cases} \tag{11.1}$$

From the program function_plot.m, it can be seen that the function has two local maximum values, namely $f(2.048, -2.048) = 3897.7342$ and $f(-2.048, -2.048) = 3905.9262$, and the latter is the global maximum, which can be seen in Fig. 11.4. Therefore, it is necessary to avoid falling into the local optimal solution when the maximum of the optimization algorithm is used.

Firstly, we use 10-bit binary genes to code x_i, -2.048 is coded as 0000000000(0), and 2.048 is coded as 1111111111(1023); then, string x_1, x_2 can be coded to 20-bit binary cluster. For example, we can use x: 0000110111 1101110001 to express a gene,the former 10-bit expresses x_1, the second half expresses x_2.

Secondly, we decode 20-bit binary string to two 10-bit binary strings and change them to decimal system value y_1 and y_2.

The relation of x_i and y_i can be written as

$$x_i = 4.096 \times \frac{y_i}{1023} - 2.048 \quad (i = 1, 2) \tag{11.2}$$

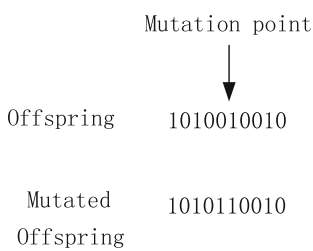


Fig. 11.3 Single mutation

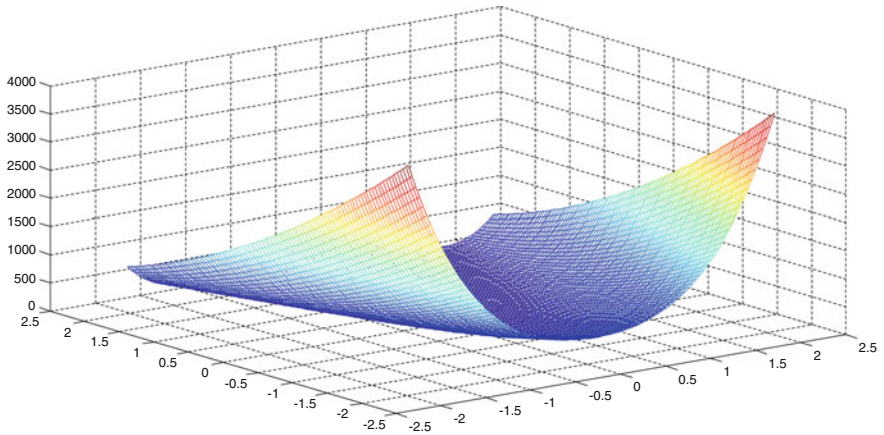


Fig. 11.4 $f(x_1, x_2)$ plot

For example, x : 0000110111 1101110001 can be decoded as

$$y_1 = 55, y_2 = 881$$

By using (11.2), we can get practical value as

$$x_1 = -1.828,\ x_2 = 1.476$$

Thirdly, calculate evaluation fitness function

$$F(x) = f(x_1, x_2)$$

Then, we can get the objective function as

$$J(x) = \frac{1}{F(x)} \tag{11.3}$$

Fourthly, design operators, including proportion selection operator,single-point crossover operator,and basic bit mutation operator, and choose parameters of GA as follows:population Size $= 80$, generation $G = 100$, crossover probability $P_c = 0.60$, mutation probability $P_m = 0.10$.

Adopting the above steps, after 100-step iterations, we get the best individual as

$$\text{Best}S = [00000000000000000000]$$

Using (11.2), we can get $x_1 = -2.0480, x_2 = -2.0480$, and then we can get the maximum value of Rosenbrock function, that is, 3905.9. The simulation results are given in Figs. 11.5 and 11.6.

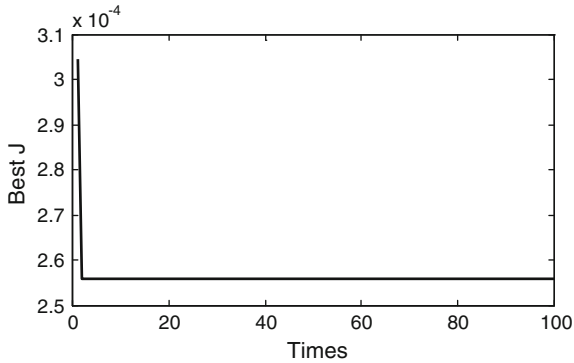


Fig. 11.5 Objective function J

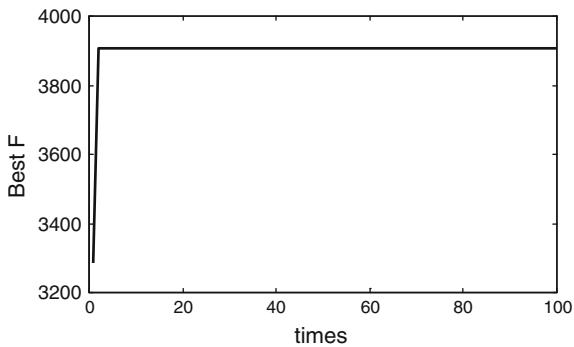


Fig. 11.6 Fitness function F

Simulation program of $f(x_1, x_2)$: function_plot.m

```
clear all;
close all;

x_min=-2.048;
x_max=2.048;

L=x_max-x_min;
N=101;
for i=1:1:N
  for j=1:1:N
    x1(i)=x_min+L/(N-1)*(i-1);  %set 100 points in x1 axis
    x2(j)=x_min+L/(N-1)*(j-1);  %set 100 points in x2 axis
    fx(i,j)=100*(x1(i)^2-x2(j))^2+(1-x1(i))^2;
   end
 end
figure(1);
surf(x1,x2,fx);
title('f(x)');
```

```
display('Maximum value of fx=');
disp(max(max(fx)));
```

Program: chap11_1.m

```
%Generic Algorithm for function f(x1,x2) optimum
clear all;
close all;

%Parameters
Size=80;
G=100;
CodeL=10;

umax=2.048;
umin=-2.048;

E=round(rand(Size,2*CodeL));  %Initial Code

%Main Program
for k=1:1:G
time(k)=k;

for s=1:1:Size
m=E(s,:);
y1=0;y2=0;

%Uncoding
m1=m(1:1:CodeL);
for i=1:1:CodeL
  y1=y1+m1(i)*2^(i-1);
end
x1=(umax-umin)*y1/1023+umin;
m2=m(CodeL+1:1:2*CodeL);
for i=1:1:CodeL
  y2=y2+m2(i)*2^(i-1);
end
x2=(umax-umin)*y2/1023+umin;

F(s)=100*(x1^2-x2)^2+(1-x1)^2;
end

Ji=1./F;
%******Step 1 : Evaluate BestJ******
BestJ(k)=min(Ji);

fi=F;                  %Fitness Function
[Oderfi,Indexfi]=sort(fi);    %Arranging fi small to bigger
```

```
Bestfi=Oderfi(Size);        %Let Bestfi=max(fi)
  BestS=E(Indexfi(Size),:);    %Let BestS=E
(m), m is the Indexfi belong to max(fi)
bfi(k)=Bestfi;

%******Step 2 : Select and Reproduct Operation******
  fi_sum=sum(fi);
  fi_Size=(Oderfi/fi_sum)*Size;

  fi_S=floor(fi_Size);      %Selecting Bigger fi value

  kk=1;
  for i=1:1:Size
    for j=1:1:fi_S(i)      %Select and Reproduce
     TempE(kk,:)=E(Indexfi(i),:);
      kk=kk+1;          %kk is used to reproduce
    end
  end

%************ Step 3 : Crossover Operation ************
pc=0.60;
n=ceil(20*rand);
for i=1:2:(Size-1)
   temp=rand;
   if pc>temp             %Crossover Condition
   for j=n:1:20
     TempE(i,j)=E(i+1,j);
     TempE(i+1,j)=E(i,j);
   end
   end
end
TempE(Size,:)=BestS;
E=TempE;

%************ Step 4: Mutation Operation **************
%pm=0.001;
%pm=0.001-[1:1:Size]*(0.001)/Size; %Bigger fi, smaller Pm
%pm=0.0;   %No mutation
pm=0.1;    %Big mutation

  for i=1:1:Size
    for j=1:1:2*CodeL
      temp=rand;
      if pm>temp           %Mutation Condition
        if TempE(i,j)==0
          TempE(i,j)=1;
        else
```

```
        TempE(i,j)=0;
      end
    end
  end
end

%Guarantee TempPop(30,:) is the code belong to the best individual(max(fi))
TempE(Size,:)=BestS;
E=TempE;
end

Max_Value=Bestfi
BestS
x1
x2
figure(1);
plot(time,BestJ);
xlabel('Times');ylabel('Best J');
figure(2);
plot(time,bfi);
xlabel('times');ylabel('Best F');
```

11.2 PSO Algorithm and Design

11.2.1 Introduction

Kennedy and Eberhart first proposed particle swarm optimization (PSO) algorithm in 1995, which is an optimization algorithm simulating the social behavior of bird flock and their means of information communication [1]. In PSO algorithm, a great number of particles move around in a multidimensional problem space, each individual is characterized by the position vector and represents a potential solution to the optimization problem.

Unlike other swarm intelligence algorithms in which the evolutionary operators are used to manipulate the individuals, each individual in PSO in the problem space has been provided with a velocity which is dynamically adjusted according to the flying experiences of its own and those of its companions. Therefore, every individual is gravitated toward a stochastically weighted average of the previous best point of its own and that of its neighborhood companions.

Initially, a swarm of particles are randomly generated. Each particle has a position vector and a velocity vector. The basic concept of PSO lies in accelerating each particle toward its pbest which is the fittest solution achieved so far by itself,

and the gbest which is the best solution obtained so far by the whole swarm with a random weighted acceleration. At every step, a particle's personal best position pbest and the gbest in the swarm are updated if an improvement in any of better fitness values is captured.

11.2.2 PSO Parameter Setting

There are two important steps in the application of PSO algorithm to solve the optimization problem as follows.

(1) Coding and fitness function:One advantage of PSO is the use of real coding, for example, for the problem of $f(x) = x_1^2 + x_2^2 + x_3^2$ maximization, the particle can be directly decoded as (x_1, x_2, x_3), and fitness function is $f(x)$.
(2) The parameters need to be adjusted in PSO are as follows:

   (a) Number of particles: Generally, the number of particles can be taken to 100 or 200.
   (b) Maximum speed $V_{\max}$: $V_{\max}$ determines the maximum moving distance of a particle in a loop, usually less than the width of the particle. Larger $V_{\max}$ can guarantee the global search ability of particle swarm, and smaller $V_{\max}$ can strengthen local search ability of particle swarm.
   (c) Learning factors c_1 and c_2 : c_1 is the local learning factor, c_2 is the global learning factor. In PSO design, generally we take a larger c_2.
   (d) Weight value: A large weight value is good for global optimization, and a small weight value is good for local optimization. When the maximum velocity $V_{\max}$ is very small, the weight value should be close to 1.0.
   In PSO, linear decreasing weight value in the iterative process is always used to obtain global optimal solution. Generally, the weight value can be set from 0.90 to 0.10.

(e) Stop conditions: Maximum number of cycles or minimum error are often used to judge the stop conditions.

11.2.3 Design Procedure of PSO

The standard PSO algorithm mainly includes the following six steps:

(1) Initialization: The parameters should be set as follows: the range of each parameter, the learning factors c_1 and c_2, the maximum evolution times G, and the particles population Size. Each particle represents a candidate solution in space solution, the position and the velocity of ith $(1 \le i \le \text{Size})$ particle in the whole solution space can be expressed as X_i and V_i.

(2) Individual evaluation (fitness evaluation): The initial position of each particle is taken as the individual extreme value, and the initial fitness value $f(X_i)$ of each particle in the population is calculated. For the i th particle, from the initial to the current iteration, the individual extreme is P_i, the current optimal solution of entire population is BestS. The initial position matrix and velocity matrix are randomly generated.
(3) Update the particle velocity and position, produce new species, check the speed, and position scope. To avoid the algorithm into a local optimal solution, we can use a local adaptive mutation operator as follows:

$$V_i^{\mathrm{kg}+1} = w(t) \times V_i^{\mathrm{kg}} + c_1 r_1 \left(p_i^{\mathrm{kg}} - X_i^{\mathrm{kg}}\right) + c_2 r_2 \left(\mathrm{BestS}_i^{\mathrm{kg}} - X_i^{\mathrm{kg}}\right) \tag{11.4}$$

$$X_i^{\mathrm{kg}+1} = X_i^{\mathrm{kg}} + V_i^{\mathrm{kg}+1} \tag{11.5}$$

where $\mathrm{kg} = 1, 2, \ldots, G$, $\mathrm{i} = 1, 2, \ldots,$ Size, r_1 and r_2 are random number from 0 to 1, c_1 is the local learning factors, and c_2 is the global learning factor; generally, take a larger c_2, $c_2 > c_1 > 0$.

(4) Compare the current fitness $f(X_i)$ value of the particle and its own historical optimal value p_i, if $f(X_i)$ is better than p_i, then p_i can be set as $f(X_i)$, and the particle position can be updated.
(5) Compare the current fitness $f(X_i)$ value of the particle and the optimal BestS value of the population, if $f(X_i)$ is better than BestS, then set BestS as $f(X_i)$, and update the global optimal value.
(6) If the termination conditions are satisfied, end the search, otherwise, go to step (3). The termination conditions can be chosen as maximum evolution times, or the given precision.

PSO algorithm's flowchart is shown in Fig. 11.7.

11.2.4 Simulation Example

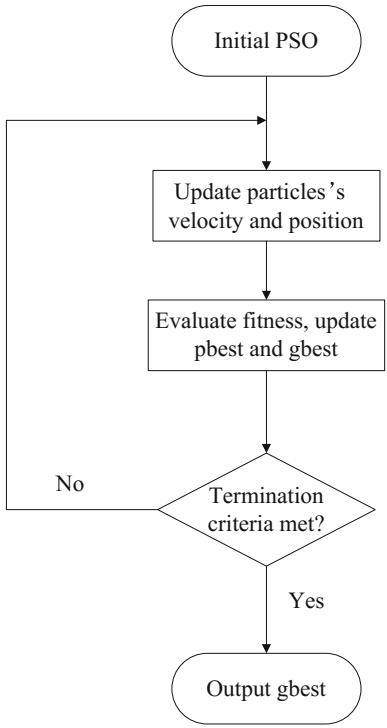
Using PSO to get maximum value of Rosenbrock function,

$$\begin{cases} f(x_1, x_2) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 \\ -2.048 \le x_i \le 2.048 \qquad (i = 1, 2) \end{cases} \tag{11.6}$$

Just like Sect. 11.1.3, the function has two local maximum value, namely $f(2.048, -2.048) = 3897.7342$ and $f(-2.048, -2.048) = 3905.9262$, and the latter is the global maximum.

In global PSO algorithm, the ith particle's neighborhood gradually increases with the increase of iterations. For the first iteration, the number of the ith particle's

Fig. 11.7 Flowchart of PSO



neighborhood is set as 0 and then increases linearly as the number of iterations and finally extended to the entire neighborhood particle swarm. Global PSO algorithm can converge quickly, but it is easy to fall into local optimum. The local PSO algorithm converges slowly, but it can avoid local optimum.

In global PSO, the velocity of each particle is updated according to the optimal value of the particle p_i and the global optimal value p_g. In order to avoid being trapped into local minima, local PSO algorithm can be used to update the velocity of each particle according to the optimal history value p_i of the particle and the optimal value p_{local} of the particle in the neighborhood.

In this section, we use a simplest circular neighborhood method to implement the local PSO algorithm, as shown in Fig. 11.8.

As an example, eight particles are chosen to illustrate local PSO algorithm, as shown in Fig. 11.8. In each update of velocity and position, particle no. 1 tracks the best individuals of particle no. 1, no. 2, and no. 8, and the particle no. 2 tracks the best individuals of the particle no. 1, no. 2, and no. 3. In the simulation, the optimal individual in the neighborhood of a particle is solved by program chap11_2lbest.m.

In local PSO, the speed and position of the particle are updated as follows

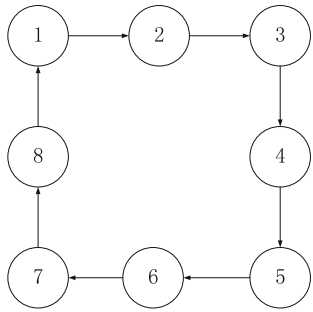


Fig. 11.8 Annular neighborhood method

$$V_i^{\mathrm{kg}+1} = w(t) \times V_i^{\mathrm{kg}} + c_1 r_1 \left(p_i^{\mathrm{kg}} - X_i^{\mathrm{kg}} \right) + c_2 r_2 \left(p_{i\mathrm{local}}^{\mathrm{kg}} - X_i^{\mathrm{kg}} \right) \tag{11.7}$$

$$X_i^{\mathrm{kg}+1} = X_i^{\mathrm{kg}} + V_i^{\mathrm{kg}+1} \tag{11.8}$$

where $p_{i\mathrm{local}}^{\mathrm{kg}}$ is locally optimized particle.

At the same time, the range of velocity and position of the particles should be examined. To prevent the algorithm from falling into the local optimal solution, local adaptive mutation operator is always used.

Real coding is used in PSO design, two real variables are used to represent two decision variables x_1 and x_2, respectively, which are discretized into real value from -2.048 to -2.048. The fitness of individual is taken as the corresponding objective function value, that is, $F(x) = f(x_1, x_2)$.

In the simulation, the number of particles is taken as Size $= 50$, the iterations maximum number is $G = 100$, maximum velocity of particle is $V_{\max} = 1.0$, and the velocity range is set as $[-1, 1]$. The learning factors are chosen as $c_1 = 1.3$ and $c_2 = 1.7$. Using the linear decreasing method, weight value is designed to decrease from 0.90 to 0.10.

In the program, $\mathrm{M} = 1$ and $\mathrm{M} = 2$ indicate local PSO and global PSO, respectively. According to (11.7) and (11.8), the velocity and position of the particles are updated to produce new species. After 100 iterations, the best sample is Best$S = [-2.048 \quad -2.048]$, that is, $x_1 = -2.048, x_2 = -2.048$, and then we can get maximum value, that is, 3905.9.

The change of fitness function F is shown in Fig. 11.9. From the simulation, to find the global optimal solution, the speed and position of particles are updated by tracking the particle swarm and local extremum along with the iterative process, the local search ability is enhanced by using local PSO algorithm, and local optimal solution is avoided. The simulation results have shown that the correct rate is above 95%.

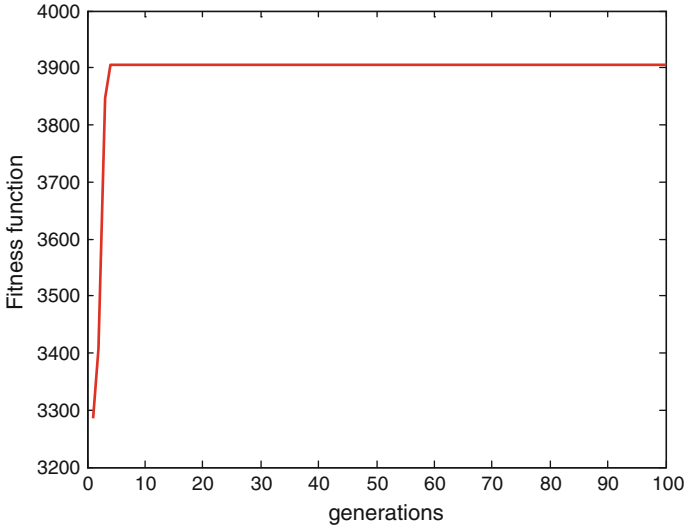


Fig. 11.9 Optimization process of fitness function F

PSO programs are listed as follows:

(1) **Main program: chap11_2.m**

```
clear all;
close all;
%(1)Initialize PSO
min=-2.048;max=2.048;
Vmax=1;Vmin=-1;
c1=1.3;c2=1.7;
wmin=0.10;wmax=0.90;
G=100;
Size=50;

for i=1:G
   w(i)=wmax-((wmax-wmin)/G)*i;
end

for i=1:Size
   for j=1:2
      x(i,j)=min+(max-min)*rand(1);
      v(i,j)=Vmin +(Vmax-Vmin)*rand(1);
   end
end
```

```
%(2) Calculte fitness
for i=1:Size
  p(i)=chap11_2func(x(i,:));
  y(i,:)=x(i,:);

  if i==1
    plocal(i,:)=chap11_2lbest(x(Size,:),x(i,:),x(i+1,:));
  elseif i==Size
    plocal(i,:)=chap11_2lbest(x(i-1,:),x(i,:),x(1,:));
    else
    plocal(i,:)=chap11_2lbest(x(i-1,:),x(i,:),x(i+1,:));
  end
end

BestS=x(1,:);
for i=2:Size
  if chap11_2func(x(i,:))>chap11_2func(BestS)
    BestS=x(i,:);
  end
end

%(3) Main loop
for kg=1:G
  for i=1:Size

   M=1;
   if M==1

v(i,:)=w(kg)*v(i,:)+c1*rand*(y(i,:)-x(i,:))+c2*rand*(plocal(i,:)-x
(i,:));%Local optimization
elseif M==2
v(i,:)=w(kg)*v(i,:)+c1*rand*(y(i,:)-x(i,:))+c2*rand*(BestS-x
(i,:));    %Global optimization
   end
     for j=1:2  %Judge the limit of velocity
      if v(i,j)<Vmin
        v(i,j)=Vmin;
      elseif  x(i,j)>Vmax
        v(i,j)=Vmax;
      end
     end
    x(i,:)=x(i,:)+v(i,:)*1; %Update position
    for j=1:2  %Check the limit
      if x(i,j)<min
        x(i,j)=min;
      elseif  x(i,j)>max
```

```
          x(i,j)=max;
        end
      end
%Adaptive mutation
     if rand>0.60
        k=ceil(2*rand);
        x(i,k)=min+(max-min)*rand(1);
     end
%(4)Judge and update
     if i==1
        plocal(i,:)=chap11_2lbest(x(Size,:),x(i,:),x(i+1,:));
      elseif i==Size
        plocal(i,:)=chap11_2lbest(x(i-1,:),x(i,:),x(1,:));
      else
        plocal(i,:)=chap11_2lbest(x(i-1,:),x(i,:),x(i+1,:));
      end

     if chap11_2func(x(i,:))>p(i) %Judge and update
       p(i)=chap11_2func(x(i,:));
       y(i,:)=x(i,:);
      end
      if p(i)>chap11_2func(BestS)
        BestS=y(i,:);
      end
   end
Best_value(kg)=chap11_2func(BestS);
end
figure(1);
kg=1:G;
plot(kg,Best_value,'r','linewidth',2);
xlabel('generations');ylabel('Fitness function');
display('Best Sample=');disp(BestS);
display('Biggest value=');disp(Best_value(G));
```

(2) **Program for local best evaluation: chap11_2lbest.m**

```
function f =evaluate_localbest(x1,x2,x3)
K0=[x1;x2;x3];
K1=[chap11_2func(x1),chap11_2func(x2),chap11_2func(x3)];
[maxvalue index]=max(K1);
plocalbest=K0(index,:);
f=plocalbest;
```

(3) **Object function program: chap11_2func.m**

```
function f = func(x)
f=100*(x(1)^2-x(2))^2+(1-x(1))^2;
```

11.3 DE Algorithm and Design

In evolutionary computation, differential evolution (DE) is a method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. Such methods are commonly known as metaheuristics as they make few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions.

DE is used for multidimensional real-valued functions but does not use the gradient of the problem being optimized, which means DE is not required for the optimization problem to be differentiable. DE can therefore also be used on optimization problems that are not even continuous, are noisy, change over time, etc. [2].

DE optimizes a problem by maintaining a population of candidate solutions and creating new candidate solutions by combining existing ones according to its simple formulae, and then keeping whichever candidate solution has the best score or fitness on the optimization problem at hand. In this way, the optimization problem is treated as a black box that merely provides a measure of quality given a candidate solution and the gradient is therefore not needed.

DE is originally due to Storn and Price [2]. Many books have been published on theoretical and practical aspects of using DE in parallel computing, multiobjective optimization, constrained optimization, and some books also contain surveys of application areas.

11.3.1 Standard DE Algorithm

DE algorithm is an optimization algorithm based on swarm intelligence theory and is guided by swarm intelligence which is generated by the cooperation and competition among individuals. DE preserves the global searching strategy based on population. By using real encoding, simple mutation differential, and one-on-one competition strategies, the complexity of DE can be reduced. DE algorithm has strong global convergence ability and robustness. The main advantages of the DE algorithm can be summarized as the following three points: few parameters, not easy to fall into local optimum, and faster convergence rate.

DE algorithm can do mutation, crossover, and selection operations based on parent individual difference; the basic idea is to generate a random initial population from the beginning, and then any two individuals are weighted and a third

individual is added according to certain rules to produce new individual. Comparing a predetermined individual with the contemporary new individual in a population, if the new individual's fitness is better than the predetermined individual fitness value, then in the next generation the new individual will replace the predetermined individual, otherwise we must preserve the predetermined individual. Through iterations, we can keep good individuals, eliminate inferior individuals, and guide the search process approach to the optimal solution.

Compared with the traditional optimization method, DE algorithm has the following main characteristics:

(1) DE algorithm starts from a group instead of one point, which is the main reason that DE can find global optimal solution with a higher probability.
(2) The evolution rule of DE algorithm is based on the adaptive information, which can greatly extend its application range without aid of other auxiliary information, such as function differentiability or continuity.
(3) DE algorithm has inherent parallelism, which makes it very suitable for massively parallel distributed processing and reduces the time cost overhead.
(4) DE algorithm uses the probability transition rule and does not need deterministic rules.

11.3.2 Basic Flow of DE

DE algorithm is an evolutionary algorithm based on real coding, which is similar to other evolutionary algorithms on the whole structure. It is composed of three basic operations: mutation, crossover, and selection. Standard DE algorithm mainly includes the following four steps:

(1) Initial population generation

In the dimensional n space, M individuals are randomly generated as

$$x_{ij}(0) = \mathrm{rand}_{ij}(0,1)\left(x_{ij}^{\mathrm{U}} - x_{ij}^{\mathrm{L}}\right) + x_{ij}^{\mathrm{L}} \tag{11.9}$$

where x_{ij}^{U} and x_{ij}^{L} are the upper and lower bounds of the jth chromosome, $\mathrm{rand}_{ij}(0,1)$ is a real value in the range $[0,1]$.

(2) Mutation operator

Three individuals x_{p1}, x_{p2}, and x_{p3} are randomly selected from the population, let $i \neq p_1 \neq p_2 \neq p_3$, and then basic mutation operator is

$$h_{ij}(t+1) = x_{\mathrm{p1j}}(t) + F\left(x_{\mathrm{p2j}}(t) - x_{\mathrm{p3j}}(t)\right) \tag{11.10}$$

If there is no local optimization problem, the mutation operator can be written as

$$h_{ij}(t+1) = x_{\mathrm{bj}}(t) + F\left(x_{\mathrm{p2j}}(t) - x_{\mathrm{p3j}}(t)\right) \tag{11.11}$$

where $x_{\mathrm{p2j}}(t) - x_{\mathrm{p3j}}(t)$ is the difference vector, the difference operation is the key of DE algorithm, and F is a scaling factor. p_1, p_2, p_3 are random integer, which indicates the number of individuals in a population, and $x_{bj}(t)$ indicates the best individual in the current generation. Since (11.11) draws on the best individual information in the current population, the convergence speed can be accelerated.

(3) Cross operator

Cross operator is to increase the diversity of the group, and the operator is as follows:

$$v_{ij}(t+1) = \begin{cases} h_{ij}(t+1), \text{rand } l_{ij} \leq \mathrm{CR} \\ x_{ij}(t), \text{rand } l_{ij} > \mathrm{CR} \end{cases} \tag{11.12}$$

where rand l_{ij} is a random value, CR is the crossover probability, $\mathrm{CR} \in [0, 1]$.

(4) Selection operator

In order to determine whether $x_i(t)$ become a member of the next generation, vector $v_{ij}(t+1)$ and vector $x_{ij}(t)$ are used to compare the evaluation functions:

$$x_i(t+1) = \begin{cases} v_i(t+1), f(v_{i1}(t+1), \cdots, v_{in}(t+1)) < f(x_{i1}(t), \cdots, x_{in}(t)) \\ x_{ij}(t), f(v_{i1}(t+1), \cdots, v_{in}(t+1)) \geq f(x_{i1}(t), \cdots, x_{in}(t)) \end{cases} \tag{11.13}$$

where $j = 1, 2, \cdots, n$.

Repeat the steps (2) to step (4) until the maximum evolutionary iteration G is reached. The basic flow of DE is shown in Fig. 11.10.

11.3.3 Parameter Setting of DE

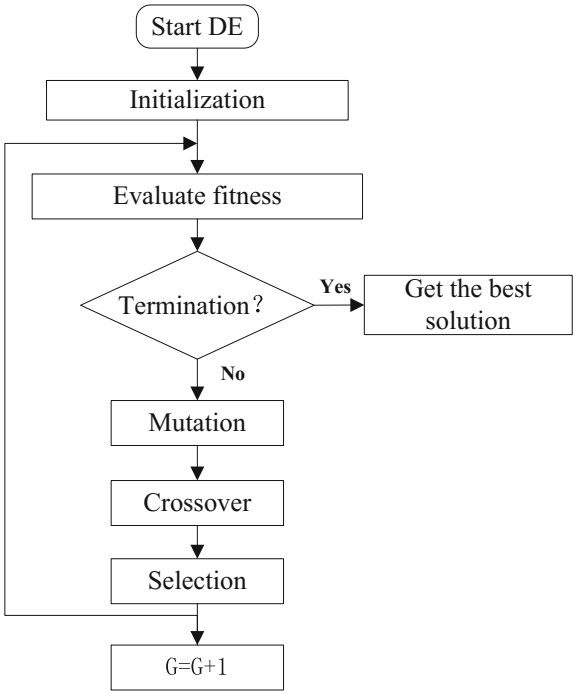
In order to improve the convergence speed of DE algorithm, we need to set reasonable parameters. For different optimization problems, parameter settings are often different.

The main parameters of DE algorithm are given as follows.

(1) Mutation factor F

The mutation factor F is an important parameter for the diversity and convergence of the population; generally, F is set in [0, 2]. When the mutation factor is small, the difference degree of the population will decrease, and the evolutionary process may not jump out of the local extremum. When the mutation factor F is

Fig. 11.10 Flowchart of DE



large, it is easy to jump out of the local extremum, but the convergence rate will slow down. Generally, we can set F = 0.3–0.6.

(2) Crossover factor CR

Crossover factor CR can affect the balance between global and local search ability. The smaller the crossover factor CR is, the less diversity of the population is, and the more easily DE algorithm will be deceived. The larger the crossover factor CR is, the larger convergence rate is, but too large CR may lead to slow convergence. Generally, we can set CR as the range $[0.6, 0.9]$.

(3) Group size

The group Size contains individual number between 5 D and 10 D(D is generally the space dimension), and D must be not less than 4, otherwise the mutation operation cannot be effective. The larger value Size is chosen, the greater the probability of obtaining the optimal solution, but the computing time is longer, usually Size can be designed from 20 to 50.

(4) Maximum iterations G

Maximum iteration G is generally used as the termination condition of evolutionary process. The greater the number of iterations, the more accurate the optimal solution, but the time will be longer.

The above four parameters have great influence on the performance of DE algorithm and efficiency of the solution.

11.3.4 Simulation Example

Solve the maximum value of Rosenbrock function by DE

$$\begin{cases} f(x_1, x_2) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 \\ -2.048 \le x_i \le 2.048 \qquad (i = 1, 2) \end{cases} \tag{11.14}$$

Just like Sect. 11.1.3, the function has two local maximum value, namely $f(2.048, -2.048) = 3897.7342$ and $f(-2.048, -2.048) = 3905.9262$, and the latter is the global maximum.

Real coding is used to find the maximum value of the function CodeL = 2, two real variables are used to represent two decision variables x_1 and x_2, respectively. x_1 and x_2 are discretized into Size real numbers from the discrete point −2.048 to −2.048. The fitness of individual is taken as the corresponding objective function value, that is, $F(x) = f(x_1, x_2)$.

In the simulation, the number of particles is taken as Size = 30, the iterations' maximum number is $G = 50$, DE algorithm is designed according to (11.9)–(11.13), F = 1.2, CR=1.9, after a total of 30 iterations, the best sample is Best$S$ = [−2.048 −2.048], that is, $x_1 = -2.048$, $x_2 = -2.048$, and at this point, Rosenbrock function has a maximum value, the maximum value is 3905.9.

The change process of fitness function $F(x)$ is shown in Fig. 11.11. By appropriately increasing F value, the local optimal solution can be avoided. The results show that the correct rate is close to 100%.

DE programs are listed as follows:

(1) **Main program: chap11_3.m**

```
%To Get maximum value of function f(x1,x2) by Differential Evolution
clear all;
close all;

Size=30;
CodeL=2;

MinX(1)=-2.048;
MaxX(1)=2.048;
MinX(2)=-2.048;
MaxX(2)=2.048;
```

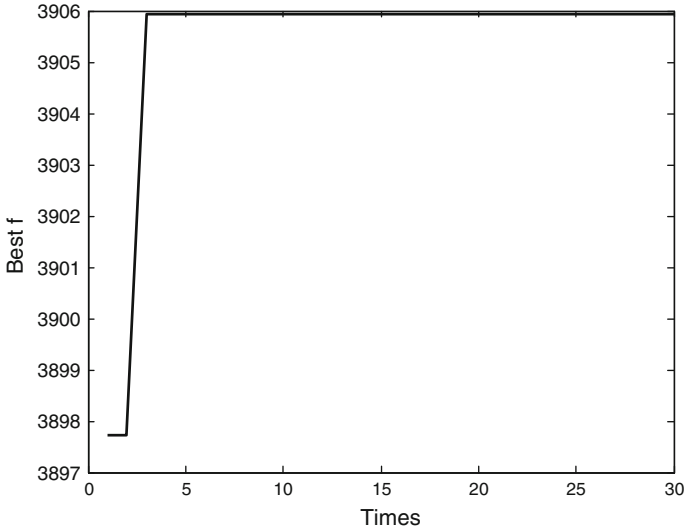


Fig. 11.11 Optimization process of fitness function $F(x)$

```
G=50;

F=1.2;          %[0,2]
cr=0.9;         %[0.6,0.9]
% Initialization
for i=1:1:CodeL
   P(:,i)=MinX(i)+(MaxX(i)-MinX(i))*rand(Size,1);
end

BestS=P(1,:); % Best individual
for i=2:Size
     if(chap11_3obj( P(i,1),P(i,2))>chap11_3obj( BestS(1),BestS(2)))
     BestS=P(i,:);
   end
end
fi=chap11_3obj( BestS(1),BestS(2));

for kg=1:1:G
    time(kg)=kg;
%mutation
   for i=1:Size
     r1 = 1;r2=1;r3=1;
     while(r1 == r2|| r1 == r3 || r2 == r3 || r1 == i || r2 ==i || r3 == i )
        r1 = ceil(Size * rand(1));
        r2 = ceil(Size * rand(1));
        r3 = ceil(Size * rand(1));
```

```
    end
    h(i,:) = P(r1,:)+F*(P(r2,:)-P(r3,:));

    for j=1:CodeL   %Check limit
      if h(i,j)<MinX(j)
        h(i,j)=MinX(j);
      elseif h(i,j)>MaxX(j)
        h(i,j)=MaxX(j);
      end
    end

%crossover
    for j = 1:1:CodeL
       tempr = rand(1);
       if(tempr<cr)
         v(i,j) = h(i,j);
       else
         v(i,j) = P(i,j);
       end
    end

%selection
    if(chap11_3obj( v(i,1),v(i,2))>chap11_3obj( P(i,1),P(i,2)))
      P(i,:)=v(i,:);
    end
%Judge and update
   if(chap11_3obj( P(i,1),P(i,2))>fi)
     fi=chap11_3obj( P(i,1),P(i,2));
     BestS=P(i,:);
    end
  end
Best_f(kg)=chap11_3obj( BestS(1),BestS(2));
end
BestS    % Best individual
Best_f(kg)  %Biggest value

figure(1);
plot(time,Best_f(time),'k','linewidth',2);
xlabel('Times');ylabel('Best f');
```

(2) Object function program: chap11_3obj.m

```
function J=evaluate_objective(x1,x2)
 J=100*( x1^2- x2)^2+(1- x1)^2;
end
```

11.4 TSP Optimization Based on Hopfield Neural Network

11.4.1 Traveling Salesman Problem

The traveling salesman problem (TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?" It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science.

TSP is a special case of the traveling purchaser problem and the vehicle routing problem.

In the theory of computational complexity, the decision version of the TSP (where, given a length L, the task is to decide whether the graph has any tour shorter than L) belongs to the class of NP-complete problems. Thus, it is possible that the worst-case running time for any algorithm for the TSP increases super-polynomially (but no more than exponentially) with the number of cities.

The problem was first formulated in 1930 and is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods. Even though the problem is computationally difficult, a large number of heuristics and exact algorithms are known, so that some instances with tens of thousands of cities can be solved completely and even problems with millions of cities can be approximated within a small fraction of 1%.

The TSP has several applications even in its purest formulation, such as planning, logistics, and the manufacture of microchips. Slightly modified, it appears as a subproblem in many areas, such as DNA sequencing. In these applications, the concept city represents, for example, customers, soldering points, or DNA fragments, and the concept distance represents traveling times or cost, or a similarity measure between DNA fragments. The TSP also appears in astronomy, as astronomers observing many sources will want to minimize the time spent moving the telescope between the sources. In many applications, additional constraints such as limited resources or time windows may be imposed.

11.4.2 Hopfield Network Design for Solving TSP Problem

The TSP problem is to find the shortest path in a set of cities $\{A_c, B_c, C_c \ldots\}$. In order to map the TSP problem into a dynamic process of a neural network, Hopfield uses $N \times N$ matrix to express the visit of N cities.

For example, there are four cities $\{A_c, B_c, C_c, D_c\}$, the route is $D_c \rightarrow A_c \rightarrow C_c \rightarrow B_c \rightarrow D_c$, and then Hopfield network output can be expressed

Table 11.1 Visit routes for four cities

City	Visit order			
	1	2	3	4
A_c	0	1	0	0
B_c	0	0	0	1
C_c	0	0	1	0
D_c	1	0	0	0

by the effective solution with the following two-dimensional matrix (see Table 11.1).

Table 11.1 consists of a matrix 4×4; in the matrix, each column has only one element whose value is 1; the remaining elements are 0, otherwise the path is an invalid path. V_{xi} indicates output of the neuron $(x, i), U_{xi}$, which is the corresponding input. If the city x is accessed at location i, we set $V_{xi} = 1$, otherwise we set $V_{xi} = 0$.

For the TSP problem, Hopfield defines the energy function as follows [3]

$$E = \frac{A}{2}\sum_{x=1}^{N}\sum_{i=1}^{N}\sum_{j=1}^{N} V_{xi}V_{xj} + \frac{B}{2}\sum_{i=1}^{N}\sum_{x=1}^{N}\sum_{y=x}^{N} V_{xi}V_{yj} + \frac{C}{2}\left(\sum_{x=1}^{N}\sum_{i=1}^{N} V_{xi} - N\right)^2 + \frac{D}{2}\sum_{x=1}^{N}\sum_{y=1}^{N}\sum_{i=1}^{N} d_{xy}V_{xi}\left(V_{y,i+1} + V_{y,i-1}\right) \tag{11.15}$$

where A, B, C, D, are weight value, d_{xy} is the distance between city x and city y.

In (11.15), the first three items of E are the constraints, and the last one is the optimization item. The first term expresses that E is minimum when each row of $\mathbf{V}$ matrix is no more than 1 (i.e., each city only once), the second guarantees that E is minimum when each column of $\boldsymbol{V}$ matrix is no more than 1 (i.e., visit only one city at a time), and the third term expresses that E is minimum when the number of $\boldsymbol{V}$ is N.

Hopfield introduces the concept of energy function to the neural network and creates a new method to solve the optimization problem. However, this method has some problems such as local minima and instability, and to solve this problem, in paper [4], the authors proposed an improved energy function of TSP as follows

$$E = \frac{A}{2}\sum_{x=1}^{N}\left(\sum_{i=1}^{N} V_{xi} - 1\right)^2 + \frac{A}{2}\sum_{i=1}^{N}\left(\sum_{x=1}^{N} V_{xi} - 1\right)^2 + \frac{D}{2}\sum_{x=1}^{N}\sum_{y=1}^{N}\sum_{i=1}^{N} V_{xi}d_{xy}V_{y,i+1} \tag{11.16}$$

From (11.16), the dynamic equation of Hopfield network is as follows:

$$\begin{aligned}\frac{dU_{xi}}{\mathrm{d}t} &= -\frac{\partial E}{\partial V_{xi}}(x, i = 1, 2, \ldots, N-1) \\ &= -A\left(\sum_{i=1}^{N} V_{xi} - 1\right) - A\left(\sum_{y=1}^{N} V_{yi} - 1\right) - D\sum_{y=1}^{N} d_{xy} V_{y,i+1}\end{aligned} \tag{11.17}$$

To solve the problem, Hopfield network algorithm is described as follows:

1. Initialization: set $t = 0$, $A = B$;
2. Calculate the distance $d_{xy}(x, y = 1, 2, \ldots, N)$ between any two cities;
3. Initialize neural network input $U_{xi}(t)$;
4. Use dynamic Eq. (11.17) to calculate $\frac{\mathrm{d}U_{xi}}{\mathrm{d}t}$;
5. Calculate $U_{xi}(t+1)$ based on first-order Euler method;

$$U_{xi}(t+1) = U_{xi}(t) + \frac{\mathrm{d}U_{xi}}{\mathrm{d}t}\Delta T \tag{11.18}$$

6. In order to ensure the convergence to the correct solution, that is, for every row and every column in the matrix $\boldsymbol{V}$, only one element is 1, and the remaining are 0, use adapt Sigmoid function to calculate $V_{xi}(t)$

$$V_{xi}(t) = \frac{1}{1 + e^{-\mu U_{xi}(t)}} \tag{11.19}$$

   where $\mu > 0$.
7. Calculate energy function E according to (11.16);
8. Check the legitimacy of the path, if the number of iterations are arrived, then terminate the iterative algorithm, or else return to step (4).
9. Give the number of iterations, the optimal path, the optimal energy function, the length of the path, and plot the curve of the energy function with time.

11.4.3 Simulation Example

In (11.16), we choose A = 1.5, D = 1.0 and we set sampling time as $\Delta T = 0.01$; the initial value of network input $U_{xi}(t)$ is chosen as random values in the range $[-1, +1]$. In (11.19), we choose larger μ value as $\mu = 50$, so that the Sigmoid function can be relatively steep, and thus in the steady states, $V_{xi}(t)$ can tend to 1 or 0.

Taking the path optimization of eight cities as an example, the path coordinates are stored in the program city8.txt. If the optimization path is effective, after 2000

iterations, the optimal energy function is Final_E = 1.4468, the initial distance is Initial_Length = 4.1419, and the shortest distance is Final_Length = 2.8937.

As the initial value of input $U_{xi}(t)$ is random, which may lead to invalid path matrix V, that is, for matrix V, some row or some column does not meet "only one element is 1, the remains are 0," the optimization program should be re-run. The simulation results have shown that in the 20 times simulation experiments, about 16 times can converge to the optimal solution.

The simulation results are shown in Figs. 11.12 and 11.13. Figure 11.12 shows the comparison between the initial path and the optimized path, and Fig. 11.13 shows energy function E change with time. The simulation results show that the energy function E tends to decrease monotonically, and the minimum point of E is the optimal solution.

The key commands used in the simulation are explained as follows:

(1) Sumsqr($\boldsymbol{X}$) can be used to Summarize the squares of all elements in matrix $\boldsymbol{X}$;
(2) Sum($\boldsymbol{X}$) or Sum($\boldsymbol{X}$,1) can be used to get the sum of each row in $\boldsymbol{X}$ matrix, and Sum ($\mathbf{X}$, 2) is the sum of each column in $\boldsymbol{X}$ matrix;

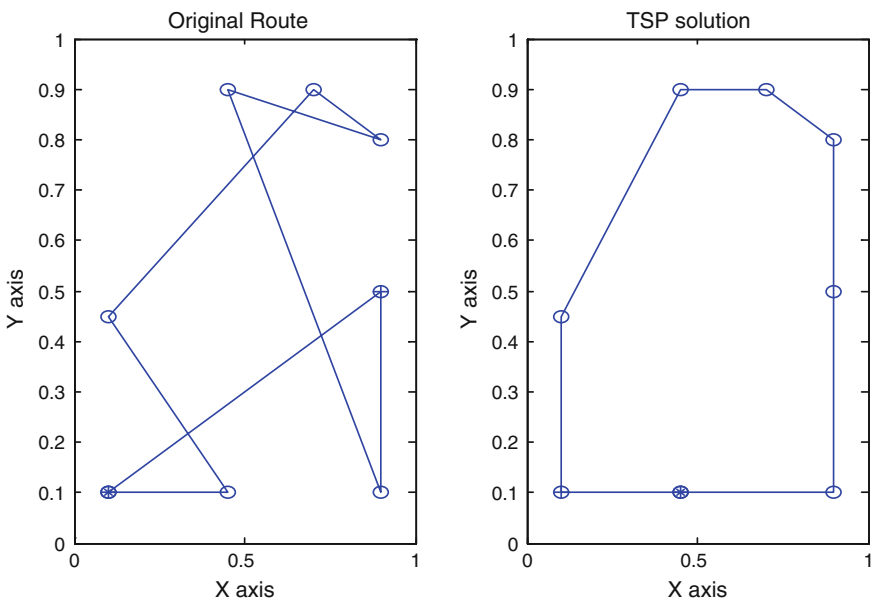


Fig. 11.12 Initial path and optimized path for eight cities

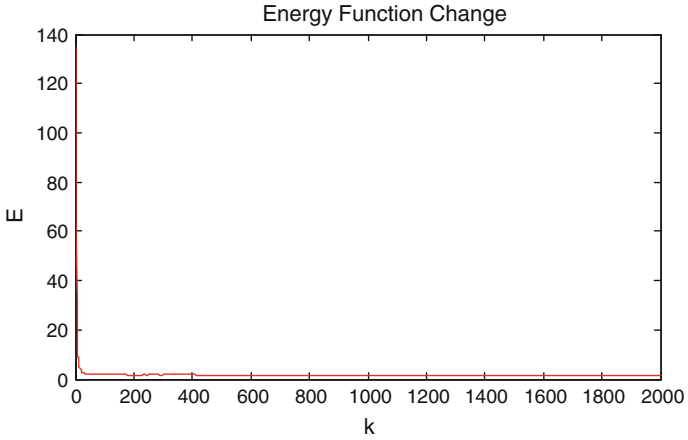


Fig. 11.13 Energy function changes with iterations

(3) Repmat can be used for matrix replication, for example, $\boldsymbol{X} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, then $\text{repmat}(\boldsymbol{X}, 1, 1) = \boldsymbol{X}$,

$$\text{repmat}(\boldsymbol{X}, 1, 2) = \begin{bmatrix} 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \end{bmatrix}, \text{repmat}(\boldsymbol{X}, 2, 1) = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 1 & 2 \\ 3 & 4; \end{bmatrix};$$

(4) $\text{dist}(\boldsymbol{x}, \boldsymbol{y})$ can be used to Calculate the distance between two points, for example, $\boldsymbol{x} = [1 \quad 1], \mathbf{y} = [2 \quad 2]'$, then $\text{dist}(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(2-1)^2 + (2-1)^2} = \sqrt{2}$.

Simulation programs

(1) **Main program: chap11_4.m**

```
% TSP Solving by Hopfield Neural Network
function TSP_hopfield()
clear all;
close all;

%Step 1: Initialization
A=1.5;
D=1;
Mu=50;
Step=0.01;

%Step 2: %Calculate initial route length
N=8;
```

```
cityfile = fopen('city8.txt','rt' );
cities = fscanf(cityfile, '%f %f',[ 2,inf] )
fclose(cityfile);
Initial_Length=Initial_RouteLength(cities);

DistanceCity=dist(cities',cities);
%Step 3: Initialization NN
U=rands(N,N);
V=1./(1+exp(-Mu*U)); % S function

for k=1:1:2000
times(k)=k;
%Step 4: Calculate du/dt
  dU=DeltaU(V,DistanceCity,A,D);
%Step 5: Calculate u(t)
  U=U+dU*Step;
%Step 6: Calculate output of NN
  V=1./(1+exp(-Mu*U)); % S function
%Step 7: Calculate energy function
  E=Energy(V,DistanceCity,A,D);
  Ep(k)=E;
%Step 8: Check validity of the route
  [V1,CheckR]=RouteCheck(V);
End

%Step 9: Results
if(CheckR==0)
  Final_E=Energy(V1,DistanceCity,A,D);
  Final_Length=Final_RouteLength(V1,cities); %Give final length
   disp('Iteration times');k
   disp(' the optimization route is');V1
   disp('Final optimization engergy function:');Final_E
   disp('Initial length');Initial_Length
   disp('Final optimization length');Final_Length

   PlotR(V1,cities);
else
   disp('the optimization route is');V1
   disp('the route is invalid');
end

figure(2);
plot(times,Ep,'r');
title('Energy Function Change');
xlabel('k');ylabel('E');
```

```
% Calculate energy function
function E=Energy(V,d,A,D)
[n,n]=size(V);
t1=sumsqr(sum(V,2)-1);
t2=sumsqr(sum(V,1)-1);
PermitV=V(:,2:n);
PermitV=[PermitV,V(:,1)];
temp=d*PermitV;
t3=sum(sum(V.*temp));
E=0.5*(A*t1+A*t2+D*t3);

%%%%%%% Calculate du/dt
function du=DeltaU(V,d,A,D)
[n,n]=size(V);
t1=repmat(sum(V,2)-1,1,n);
t2=repmat(sum(V,1)-1,n,1);
PermitV=V(:,2:n);
PermitV=[PermitV, V(:,1)];
t3=d*PermitV;
du=-1*(A*t1+A*t2+D*t3);

%Check the validity of route
function [V1,CheckR]=RouteCheck(V)
[rows,cols]=size(V);
V1=zeros(rows,cols);
[XC,Order]=max(V);
for j=1:cols
  V1(Order(j),j)=1;
end
C=sum(V1);
R=sum(V1');
CheckR=sumsqr(C-R);

% Calculate Initial Route Length
function L0=Initial_RouteLength(cities)
[r,c]=size(cities);
L0=0;
for i=2:c
 L0=L0+dist(cities(:,i-1)',cities(:,i));
end
% Calculate Final Route Length
function L=Final_RouteLength(V,cities)
[xxx,order]=max(V);
New=cities(:,order);
New=[New New(:,1)];
[rows,cs]=size(New);
```

```
L=0;
for i=2:cs
  L=L+dist(New(:,i-1)',New(:,i));
end

% Give Path optimization plot
function PlotR(V,cities)
figure;

cities=[cities cities(:,1)];

[xxx,order]=max(V);
New=cities(:,order);
New=[New New(:,1)];

subplot(1,2,1);
plot( cities(1,1), cities(2,1),'r*' );  %First city
hold on;
plot( cities(1,2), cities(2,2),'+' );   %Second city
hold on;
plot( cities(1,:), cities(2,:),'o-' ), xlabel('X axis'), ylabel
('Y axis'), title('Original Route');
axis([0,1,0,1]);

subplot(1,2,2);
plot( New(1,1), New(2,1),'r*' );  %First city
hold on;
plot( New(1,2), New(2,2),'+' );   %Second city
hold on;
plot(New(1,:),New(2,:),'o-');
title('TSP solution');
xlabel('X axis');ylabel('Y axis');
axis([0,1,0,1]);
axis on
```

(2) **Program for coordinates of eight cities: city8.txt**

```
0.1 0.1
0.9 0.5
0.9 0.1
0.45 0.9
0.9 0.8
0.7 0.9
0.1 0.45
0.45 0.1
```

References

1. J. Kennedy, R. Eberhart, Particle swarm optimization. IEEE Int. Conf. Neural Netw. **4**, 1942–1948 (1995)
2. R. Storn, K. Price, Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**, 341–359 (1997)
3. J.J. Hopfield, D.W. Tank, Neural computation of decision in optimization problems. Biol. Cybernrtics **52**, 141–152 (1985)
4. S.Y. Sun, J.L. Zheng, A modified algorithm and theoretical analysis for hopfield neural solving TSP. Acta Electronica Sinca **23**(1), 73–78 (1995). (in Chinese)

Chapter 12
Iterative Learning Control and Applications

There is a kind of trajectory tracking problem in practical control. The control task is to find the control law, which makes the output of the controlled object to achieve the zero error of trajectory tracking along the desired trajectory. This tracking problem is a challenging control problem.

When dealing with the repetitive tasks in the actual practical engineering, we often adjust the decision according to the difference between the dynamic behavior and the expected behavior. Through repeated operations, the object behavior and the expected behavior can meet the requirements.

The idea of iterative learning control (ILC) was first proposed by Uchiyama, a Japanese scholar in 1978, Arimoto, etc. [1] made a pioneering study in 1984.

Iterative learning control method has strong engineering background, and these backgrounds include the following: industrial robot such as welding, spraying, assembly, handling, and other repetitive tasks, disk drive system used in mechanical manufacturing, and coordinate measuring machine [2–4].

Iterative learning control is a typical intelligent control method, which simulates the function of human brain learning and self-regulation. After more than thirty years of development, iterative learning control has become a branch of intelligent control with strict mathematical description. At present, iterative learning control has made great progress in learning algorithm, convergence, robustness, learning speed, and engineering applications.

12.1 Basic Principle

Consider a dynamic model as

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), t), \boldsymbol{y}(t) = g(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \tag{12.1}$$

where $\boldsymbol{x} \in \boldsymbol{R}^n, \boldsymbol{y} \in \boldsymbol{R}^m, \boldsymbol{u} \in \boldsymbol{R}^r$ are system state, output and input variables, respectively, $f(\cdot)$ and $g(\cdot)$ are unknown vector functions.

For the expected control $\boldsymbol{u}_{\mathrm{d}}(t)$, if the initial states $\boldsymbol{x}_k(0)$ and expected output $\boldsymbol{y}_{\mathrm{d}}(t)$ are given, for the given period of time $t \in [0, T]$, according to the learning algorithm by repeated operation, we can realize $\boldsymbol{u}_k(t) \to \boldsymbol{u}_{\mathrm{d}}(t)$ and $\boldsymbol{y}_k(t) \to \boldsymbol{y}_{\mathrm{d}}(t)$, in the k times running, (12.1) can be represented as

$$\dot{\boldsymbol{x}}_k(t) = f(\boldsymbol{x}_k(t), \boldsymbol{u}_k(t), t),\ \boldsymbol{y}_k(t) = g(\boldsymbol{x}_k(t), \boldsymbol{u}_k(t), t) \tag{12.2}$$

The tracking error is

$$\boldsymbol{e}_k(t) = \boldsymbol{y}_{\mathrm{d}}(t) - \boldsymbol{y}_k(t) \tag{12.3}$$

The iterative learning control can be divided into open-loop learning control and closed-loop learning control.

For the open-loop learning control, the $k+1$ times control is equal to the correction of the k times control combine with the k times output error.

$$\boldsymbol{u}_{k+1}(t) = L(\boldsymbol{u}_k(t), \boldsymbol{e}_k(t)) \tag{12.4}$$

The closed-loop learning strategy is to take the error in $k+1$ times as the correction of learning

$$\boldsymbol{u}_{k+1}(t) = L(\boldsymbol{u}_k(t), \boldsymbol{e}_{k+1}(t)) \tag{12.5}$$

where L is linear or nonlinear operator.

12.2 Basic Iterative Learning Control Algorithm

The D-type iterative learning control law for linear time-varying continuous systems is given by Arimoto et al. [1]

$$\boldsymbol{u}_{k+1}(t) = \boldsymbol{u}_k(t) + \boldsymbol{\Gamma}\dot{\boldsymbol{e}}_k(t) \tag{12.6}$$

where $\boldsymbol{\Gamma}$ is constant gain matrix.

PID-type iterative learning control law is expressed as

$$\boldsymbol{u}_{k+1}(t) = \boldsymbol{u}_k(t) + \boldsymbol{\Gamma}\dot{\boldsymbol{e}}_k(t) + \boldsymbol{\Phi}\boldsymbol{e}_k(t) + \boldsymbol{\Psi}\int_0^t \boldsymbol{e}_k(\tau)\mathrm{d}\tau \tag{12.7}$$

where $\boldsymbol{\Gamma}, \boldsymbol{\Phi}$, and $\boldsymbol{\Psi}$ are learning gain matrices.

In iterative learning control law, if $\boldsymbol{e}_k(t)$ is used, the control law is called as open-loop LTC, if $\boldsymbol{e}_{k+1}(t)$ is used, the control law is called as closed-loop LTC, and

if $\boldsymbol{e}_k(t)$ and $\boldsymbol{e}_{k+1}(t)$ are used at the same time, the control law is called as open-loop and closed-loop LTC.

In addition, there also have other LTC algorithm, such as higher order iterative learning control algorithm, optimal iterative learning control algorithm, forgetting factor iterative learning control algorithm and feedback feed-forward iterative learning control algorithm, etc.

12.3 Key Techniques of Iterative Learning Control

12.3.1 Stability and Convergence

For learning control system, only stability is not enough, only convergence can guarantee that the practical value converges to ideal value.

12.3.2 Initial Value Problem

Most of the iterative learning control algorithms require that the initial states' value of the system is equal to the initial states' value of the desired trajectory, i.e.,

$$x_k(0) = x_{\mathrm{d}}(0), k = 0, 1, 2, \ldots \tag{12.8}$$

12.3.3 Learning Speed Problem

In iterative learning algorithm, the convergence condition is given by $k \to \infty$, which is obviously of no practical significance. Therefore, how to make the iterative learning process converge faster to the expected value is another important problem in the research of iterative learning control.

12.3.4 Robustness

In addition to the initial offset, a practical iterative learning control system has more or less disturbances such as state disturbance, measurement noise, and input disturbance. Robustness problems should be discussed for iterative learning control systems with various disturbances.

12.4 ILC Simulation for Manipulator Trajectory Tracking

12.4.1 Controller Design

Consider dynamic equation of N link manipulator as

$$\boldsymbol{D}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{\tau} - \boldsymbol{\tau}_{\mathrm{d}} \tag{12.9}$$

where $\boldsymbol{q} \in \boldsymbol{R}^n$ is joint angular vector, $\boldsymbol{D}(\boldsymbol{q}) \in \boldsymbol{R}^{n\times n}$ is inertia matrix, $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \boldsymbol{R}^n$ represents centrifugal force and Coriolis force, $\boldsymbol{G}(\boldsymbol{q}) \in \boldsymbol{R}^n$ is gravity term, $\boldsymbol{\tau} \in \boldsymbol{R}^n$ is control input vector, and $\boldsymbol{\tau}_{\mathrm{d}} \in \boldsymbol{R}^n$ is disturbance.

The desired trajectory to be tracked by the system is set as $\mathbf{y}_{\mathrm{d}}(t), t \in [0, T]$. The system output at i times at time t is $\mathbf{y}_i(t)$, and let $\mathbf{e}_i(t) = \mathbf{y}_{\mathrm{d}}(t) - \mathbf{y}_i(t)$.

Based on feedback, three kinds of iterative learning control laws are as follows:

(1) D-type closed-loop ILC

$$\boldsymbol{u}_{k+1}(t) = \boldsymbol{u}_k(t) + \boldsymbol{K}_{\mathrm{d}}\left(\dot{\boldsymbol{q}}_{\mathrm{d}}(t) - \dot{\boldsymbol{q}}_{k+1}(t)\right) \tag{12.10}$$

(2) PD-type closed-loop ILC

$$\boldsymbol{u}_{k+1}(t) = \boldsymbol{u}_k(t) + \boldsymbol{K}_{\mathrm{p}}\left(\boldsymbol{q}_{\mathrm{d}}(t) - \boldsymbol{q}_{k+1}(t)\right) + \boldsymbol{K}_{\mathrm{d}}\left(\dot{\boldsymbol{q}}_{\mathrm{d}}(t) - \dot{\boldsymbol{q}}_{k+1}(t)\right) \tag{12.11}$$

(3) Exponential variable gain D-type closed-loop ILC

$$\boldsymbol{u}_{k+1}(t) = \boldsymbol{u}_k(t) + \boldsymbol{K}_{\mathrm{d}}\left(\dot{\boldsymbol{q}}_{\mathrm{d}}(t) - \dot{\boldsymbol{q}}_{k+1}(t)\right) \tag{12.12}$$

The convergence analysis of above controller is given in paper [1].

12.4.2 Simulation Example

Consider the plant as (12.9), we assume

$$\mathbf{D} = \left[d_{ij}\right]_{2\times 2},$$
$$d_{11} = d_1 l_{c1}^2 + d_2\left(l_1^2 + l_{c2}^2 + 2l_1 l_{c2}\cos q_2\right) + I_1 + I_2$$
$$d_{12} = d_{21} = d_2\left(l_{c2}^2 + l_1 l_{c2}\cos q_2\right) + l_2$$
$$d_{22} = d_2 l_{c2}^2 + I_2$$
$$\mathbf{C} = \left[c_{ij}\right]_{2\times 2}$$
$$c_{11} = h\dot{q}_2, c_{12} = h\dot{q}_1 + h\dot{q}_2, c_{21} = -h\dot{q}_1, c_{22} = 0, h = -m_2 l_1 l_{c2}\sin q_2$$
$$\mathbf{G} = \left[G_1 \quad G_2\right]^{\mathrm{T}}$$
$$G_1 = (d_1 l_{c1} + d_2 l_1)\mathrm{g}\cos q_1 + d_2 l_{c2}\mathrm{g}\cos(q_1 + q_2), G_2 = d_2 l_{c2}\mathrm{g}\cos(q_1 + q_2)$$
$$\tau_{\mathrm{d}} = \left[0.3\sin t \quad 0.1(1 - \mathrm{e}^{-t})\right]^{\mathrm{T}}.$$

The physical parameters are set as $d_1 = d_2 = 1$, $l_1 = l_2 = 0.5$, $l_{c1} = l_{c2} = 0.25$, $I_1 = I_2 = 0.1$, $g = 9.81$.

Three kinds of closed-loop iterative learning control laws are used; set M = 1 as D-type ILC, set M = 2 as PD-type ILC, and set M = 3 as exponential variable gain D-type ILC.

The ideal position signal of the two joints is set as $\sin(3t)$ and $\cos(3t)$, respectively. To ensure $\boldsymbol{q}_{\mathrm{d}}(0) = \boldsymbol{q}(0)$, choose $\boldsymbol{x}(0) = \left[0 \quad 3 \quad 1 \quad 0\right]^{\mathrm{T}}$. Choosing $M = 2$, $K_{\mathrm{p}} = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$, $K_{\mathrm{d}} = \begin{bmatrix} 500 & 0 \\ 0 & 500 \end{bmatrix}$ the simulation results are shown in Figs. 12.1, 12.2, and 12.3.

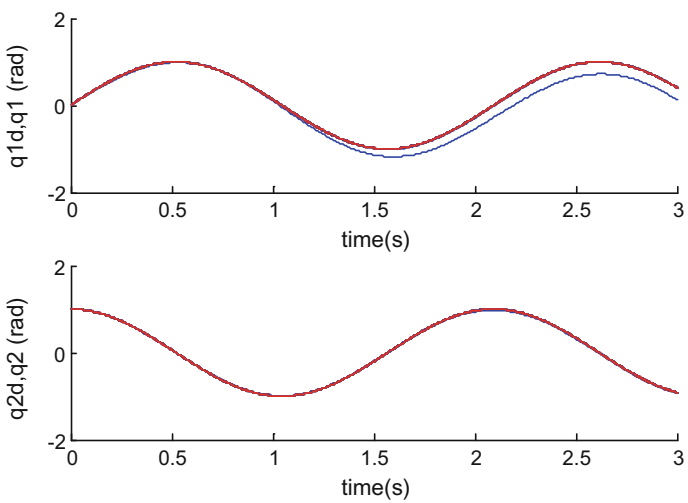


Fig. 12.1 Tracking process during the twentieth times

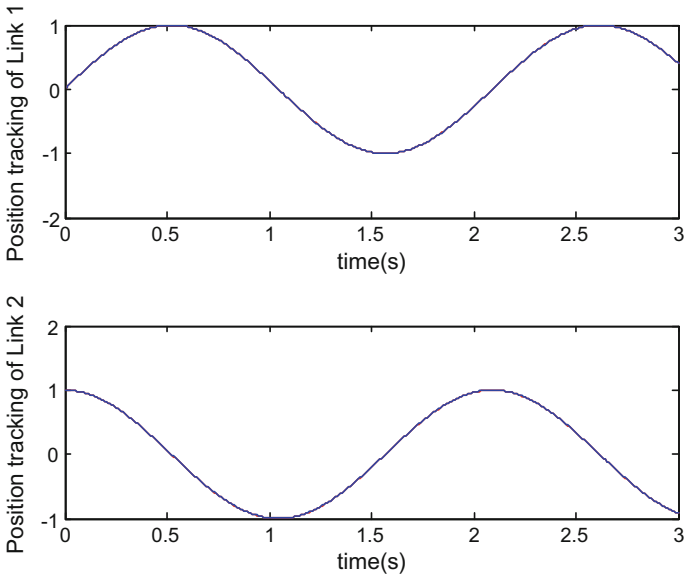


Fig. 12.2 Angle tracking process for the twentieth times

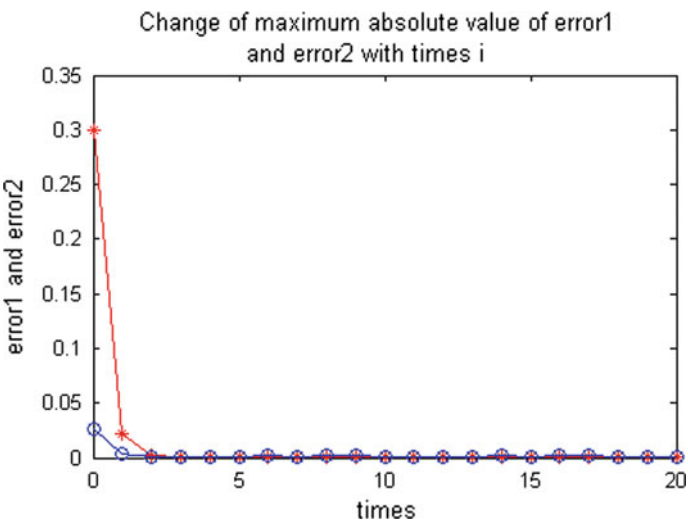


Fig. 12.3 Convergence of error norm process during the twentieth times

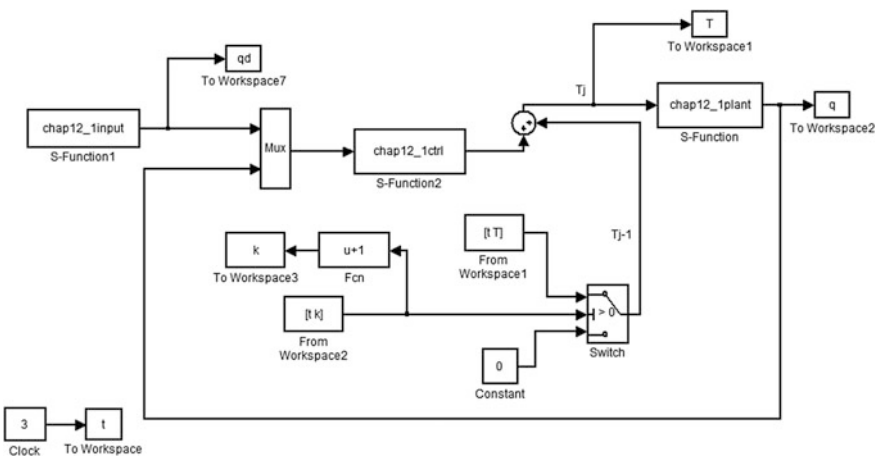
Simulation programs:

(1) Main program: chap12_1main.m

```
%Adaptive switching Learning Control for 2DOF robot manipulators
clear all;
close all;

t=[0:0.01:3]';
k(1:301)=0;   %Total initial points
k=k';
T1(1:301)=0;
T1=T1';
T2=T1;
T=[T1 T2];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
M=20;
for i=0:1:M   % Start Learning Control
i
pause(0.01);

sim('chap12_1sim',[0,3]);

q1=q(:,1);
dq1=q(:,2);
q2=q(:,3);
dq2=q(:,4);

q1d=qd(:,1);
dq1d=qd(:,2);
q2d=qd(:,3);
dq2d=qd(:,4);

e1=q1d-q1;
e2=q2d-q2;
de1=dq1d-dq1;
de2=dq2d-dq2;

figure(1);
subplot(211);
hold on;
plot(t,q1,'b',t,q1d,'r');
xlabel('time(s)');ylabel('q1d,q1 (rad)');

subplot(212);
hold on;
plot(t,q2,'b',t,q2d,'r');
xlabel('time(s)');ylabel('q2d,q2 (rad)');
```

```
j=i+1;
times(j)=i;
e1i(j)=max(abs(e1));
e2i(j)=max(abs(e2));
de1i(j)=max(abs(de1));
de2i(j)=max(abs(de2));
end      %End of i
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(2);
subplot(211);
plot(t,q1d,'r',t,q1,'b');
xlabel('time(s)');ylabel('Position tracking of Link 1');
subplot(212);
plot(t,q2d,'r',t,q2,'b');
xlabel('time(s)');ylabel('Position tracking of Link 2');

figure(3);
plot(times,e1i,'*-r',times,e2i,'o-b');
title
('Change of maximum absolute value of error1 and error2 with times i');
xlabel('times');ylabel('error 1 and error 2');
```

(2) **Simulink program: chap12_1sim.mdl**



(3) **Ideal signal program: chap12_1input.m**

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
```

```
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 4;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [0 0];
function sys=mdlOutputs(t,x,u)
q1d=sin(3*t);
dq1d=3*cos(3*t);
q2d=cos(3*t);
dq2d=-3*sin(3*t);

sys(1)=q1d;
sys(2)=dq1d;
sys(3)=q2d;
sys(4)=dq2d;
```

(4) S function for plant: chap12_1plant.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
```

```
sizes = simsizes;
sizes.NumContStates  = 4;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 4;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = [0;3;1;0];  %Must be equal to x(0) of ideal input
str = [];
ts  = [0 0];
function sys=mdlDerivatives(t,x,u)
Tol=[u(1) u(2)]';

g=9.81;
d1=10;d2=5;
l1=1;l2=0.5;
lc1=0.5;lc2=0.25;
I1=0.83;I2=0.3;

D11=d1*lc1^2+d2*(l1^2+lc2^2+2*l1*lc2*cos(x(3)))+I1+I2;
D12=d2*(lc2^2+l1*lc2*cos(x(3)))+I2;
D21=D12;
D22=d2*lc2^2+I2;
D=[D11 D12;D21 D22];
h=-d2*l1*lc2*sin(x(3));
C11=h*x(4);
C12=h*x(4)+h*x(2);
C21=-h*x(2);
C22=0;
C=[C11 C12;C21 C22];
g1=(d1*lc1+d2*l1)*g*cos(x(1))+d2*lc2*g*cos(x(1)+x(3));
g2=d2*lc2*g*cos(x(1)+x(3));
G=[g1;g2];

a=1.0;
d1=a*0.3*sin(t);
d2=a*0.1*(1-exp(-t));
Td=[d1;d2];

S=-inv(D)*C*[x(2);x(4)]-inv(D)*G+inv(D)*(Tol-Td);

sys(1)=x(2);
sys(2)=S(1);
sys(3)=x(4);
sys(4)=S(2);
function sys=mdlOutputs(t,x,u)
```

```
sys(2)=x(2);  %Angle1 speed:dq1
sys(3)=x(3);  %Angle2:q2
sys(4)=x(4);  %Angle2 speed:dq2
```

(5) S function of controller: chap12_1ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs    = 2;
sizes.NumInputs     = 8;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [0 0];
function sys=mdlOutputs(t,x,u)
q1d=u(1);dq1d=u(2);
q2d=u(3);dq2d=u(4);

q1=u(5);dq1=u(6);
q2=u(7);dq2=u(8);

e1=q1d-q1;
e2=q2d-q2;
e=[e1 e2]';
de1=dq1d-dq1;
de2=dq2d-dq2;
de=[de1 de2]';

Kp=[100 0;0 100];
Kd=[500 0;0 500];
```

```
M=2;
if M==1
    Tol=Kd*de;     %D Type
elseif M==2
    Tol=Kp*e+Kd*de; %PD Type
elseif M==3
    Tol=Kd*exp(0.8*t)*de; %Exponential Gain D Type
end
sys(1)=Tol(1);
sys(2)=Tol(2);
```

12.5 Iterative Learning Control for Time-Varying Linear System

12.5.1 System Description

Consider a time-varying linear system as

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \boldsymbol{A}(t)\boldsymbol{x}(t) + \boldsymbol{B}(t)\boldsymbol{u}(t) \\ \boldsymbol{y}(t) &= \boldsymbol{C}(t)\boldsymbol{x}(t) \end{aligned} \tag{12.13}$$

The open-loop PID-type LTC law is

$$\boldsymbol{u}_{k+1}(t) = \boldsymbol{u}_k(t) + \left(\boldsymbol{\Gamma}\frac{\mathrm{d}}{\mathrm{d}t} + \mathbf{L} + \boldsymbol{\Psi}\int \mathrm{d}t\right)\boldsymbol{e}_k(t) \tag{12.14}$$

where $\boldsymbol{\Gamma}, \mathbf{L}, \boldsymbol{\Psi}$ are gain matrices.

12.5.2 Design and Convergence Analysis

Theorem 12.1 For the control system (12.13) and (12.14), if the following conditions are satisfied [1, 5]:

(1) $\|\boldsymbol{I} - \boldsymbol{C}(t)\boldsymbol{B}(t)\boldsymbol{\Gamma}(t)\| \le \bar{\rho} < 1$;
(2) For each iteration, $\boldsymbol{x}_k(0) = \boldsymbol{x}_0(k = 1, 2, 3, \ldots), \boldsymbol{y}_0(0) = \boldsymbol{y}_\mathrm{d}(0)$.

Refer to [5], the concrete analysis is given below.

Then, $k \to \infty, \boldsymbol{y}_k(t) \to \boldsymbol{y}_\mathrm{d}(t), \quad \forall t \in [0, T]$.

Proof From (12.13) and above condition (2), we have $\boldsymbol{y}_{k+1}(0) = \boldsymbol{C}\boldsymbol{x}_{k+1}(0) = \boldsymbol{C}\boldsymbol{x}_k(0) = \boldsymbol{y}_k(0)$, and then $\boldsymbol{e}_k(0) = 0(k = 0, 1, 2, \ldots)$.

The solution of $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}(t)\boldsymbol{x}(t) + \boldsymbol{B}(t)\boldsymbol{u}(t)$ is

$$\begin{aligned}\boldsymbol{x}(t) &= \boldsymbol{C}\exp\left(\int_0^t \boldsymbol{A}\mathrm{d}\tau\right) + \exp\left(\int_0^t \boldsymbol{A}\mathrm{d}\tau\right)\int_0^t \boldsymbol{B}(\tau)\boldsymbol{u}(\tau)\exp\left(\int_0^\tau -\boldsymbol{A}\mathrm{d}\delta\mathrm{d}\tau\right) \\ &= \boldsymbol{C}\exp(\boldsymbol{A}t) + \exp(\boldsymbol{A}t)\int_0^t \boldsymbol{B}(\tau)\boldsymbol{u}(\tau)\exp(-\boldsymbol{A}\tau)\mathrm{d}\tau\int_0^t = \boldsymbol{C}\exp(\boldsymbol{A}t) + \int_0^t \exp(\boldsymbol{A}(t-\tau))\boldsymbol{B}(\tau)\boldsymbol{u}(\tau)\mathrm{d}\tau\end{aligned}$$

Let $\boldsymbol{\Phi}(t,\tau) = \exp(\boldsymbol{A}(t-\tau))$, then

$$\boldsymbol{x}_k(t) - \boldsymbol{x}_{k+1}(t) = \int_0^t \boldsymbol{\Phi}(t,\tau)\boldsymbol{B}(\tau)(\boldsymbol{u}_k(\tau) - \boldsymbol{u}_{k+1}(\tau))\mathrm{d}\tau$$

Let $\boldsymbol{e}_k(t) = \boldsymbol{y}_\mathrm{d}(t) - \boldsymbol{y}_k(t), \boldsymbol{e}_{k+1}(t) = \boldsymbol{y}_\mathrm{d}(t) - \boldsymbol{y}_{k+1}(t)$, then

$$\begin{aligned}\boldsymbol{e}_{k+1}(t) - \boldsymbol{e}_k(t) &= \boldsymbol{y}_k(t) - \boldsymbol{y}_{k+1}(t) = \boldsymbol{C}(t)(\boldsymbol{x}_k(t) - \boldsymbol{x}_{k+1}(t)) \\ &= \int_0^t \boldsymbol{C}(t)\boldsymbol{\Phi}(t,\tau)\boldsymbol{B}(\tau)(\boldsymbol{u}_k(\tau) - \boldsymbol{u}_{k+1}(\tau))\mathrm{d}\tau\end{aligned}$$

i.e.,

$$\boldsymbol{e}_{k+1}(t) = \boldsymbol{e}_k(t) - \int_0^t \boldsymbol{C}(t)\boldsymbol{\Phi}(t,\tau)\boldsymbol{B}(\tau)(\boldsymbol{u}_{k+1}(\tau) - \boldsymbol{u}_k(\tau))\mathrm{d}\tau$$

Inserting (12.14) into above, we have

$$\begin{aligned}\boldsymbol{e}_{k+1}(t) = \boldsymbol{e}_k(t) \\ - \int_0^t \boldsymbol{C}(t)\boldsymbol{\Phi}(t,\tau)\boldsymbol{B}(\tau)\left[\boldsymbol{\Gamma}(\tau)\dot{\boldsymbol{e}}_k(\tau) + \boldsymbol{L}(\tau)\boldsymbol{e}_k(\tau) + \boldsymbol{\Psi}(\tau)\int_0^\tau \boldsymbol{e}_k(\delta)\mathrm{d}\delta\right]\mathrm{d}\tau\end{aligned} \tag{12.15}$$

Using integration by parts, let $\boldsymbol{G}(t,\tau) = \boldsymbol{C}(t)\boldsymbol{B}(\tau)\boldsymbol{\Gamma}(\tau)$, then

$$\begin{aligned}\int_0^t \boldsymbol{C}(t)\boldsymbol{B}(\tau)\boldsymbol{\Gamma}(\tau)\dot{\boldsymbol{e}}_k(\tau)\mathrm{d}\tau &= \boldsymbol{G}(t,\tau)\boldsymbol{e}_k(\tau)\big|_0^t - \int_0^t \frac{\partial}{\partial\tau}\boldsymbol{G}(t,\tau)\boldsymbol{e}_k(\tau)\mathrm{d}\tau \\ &= \boldsymbol{C}(t)\boldsymbol{B}(\tau)\boldsymbol{\Gamma}(\tau)\boldsymbol{e}_k(\tau) - \int_0^t \frac{\partial}{\partial\tau}\boldsymbol{G}(t,\tau)\boldsymbol{e}_k(\tau)\mathrm{d}\tau\end{aligned} \tag{12.16}$$

Inserting (12.16) into (12.15), we have

$$\boldsymbol{e}_{k+1}(t) = [\boldsymbol{I} - \boldsymbol{C}(t)\boldsymbol{B}(t)\boldsymbol{\Gamma}(t)]\boldsymbol{e}_k(t) + \int_0^t \frac{\partial}{\partial\tau}\boldsymbol{G}(t,\tau)\boldsymbol{e}_k(\tau)\mathrm{d}\tau$$
$$- \int_0^t \boldsymbol{C}(t)\boldsymbol{\Phi}(t,\tau)\boldsymbol{B}(\tau)\mathbf{L}(\tau)\boldsymbol{e}_k(\tau)\mathrm{d}\tau - \int_0^t\int_0^\tau \boldsymbol{C}(t)\boldsymbol{\Phi}(t,\tau)\boldsymbol{B}(\tau)\boldsymbol{\psi}(\tau)\boldsymbol{e}_k(\sigma)\mathrm{d}\sigma\mathrm{d}\tau \tag{12.17}$$

For (12.17), we have

$$\begin{aligned}\|\boldsymbol{e}_{k+1}(t)\| &\le \|\boldsymbol{I} - \boldsymbol{C}(t)\boldsymbol{B}(t)\boldsymbol{\Gamma}(t)\|\|\boldsymbol{e}_k(t)\| + \int_0^t \left\|\frac{\partial}{\partial\tau}\boldsymbol{G}(t,\tau)\right\|\|\boldsymbol{e}_k(\tau)\|\mathrm{d}\tau \\ &+ \int_0^t \|\boldsymbol{C}(t)\boldsymbol{\Phi}(t,\tau)\boldsymbol{B}(\tau)\mathbf{L}(\tau)\|\|\boldsymbol{e}_k(\tau)\|\mathrm{d}\tau + \int_0^t\int_0^\tau \|\boldsymbol{C}(t)\boldsymbol{\Phi}(t,\tau)\boldsymbol{B}(\tau)\boldsymbol{\psi}(\tau)\|\|\boldsymbol{e}_k(\sigma)\|\mathrm{d}\sigma\mathrm{d}\tau \\ &\le \|\boldsymbol{I} - \boldsymbol{C}(t)\boldsymbol{B}(t)\boldsymbol{\Gamma}(t)\|\|\boldsymbol{e}_k(t)\| + \int_0^t b_1\|\boldsymbol{e}_k(\tau)\|\mathrm{d}\tau + \int_0^t\int_0^\tau b_2\|\boldsymbol{e}_k(\sigma)\|\mathrm{d}\sigma\mathrm{d}\tau\end{aligned} \tag{12.18}$$

where

$$b_1 = \max\left\{\sup_{t,\tau\in[0,T]}\left\|\frac{\partial}{\partial\tau}\boldsymbol{G}(t,\tau)\right\|, \sup_{t,\tau\in[0,T]}\left\|\boldsymbol{C}(t)\int_0^\tau \|\mathbf{C}(t)\boldsymbol{\Phi}(t,\tau)\mathbf{B}(\tau)\mathbf{L}(\tau)\|\right\|\right\}$$
$$b_2 = \sup_{t,\tau\in[0,T]}\|\boldsymbol{C}(t)\boldsymbol{\Phi}(t,\tau)\boldsymbol{B}(\tau)\boldsymbol{\psi}(\tau)\|$$

According to the definition of λ-norm, $\|f\|_\lambda = \sup_{0\le t\le T}\{\|f(t)\|\mathrm{e}^{-\lambda t}\}$.

Multiply by $\exp(-\lambda t)$ on both sides in (12.18), $\lambda > 0$, consider $\int_0^t \exp(\lambda\tau)\mathrm{d}\tau = \frac{\exp(\lambda t)-1}{\lambda}$, we have

$$\begin{aligned}\exp(-\lambda t)\int_0^t b_1\|\boldsymbol{e}_k(\tau)\|\mathrm{d}\tau &= \exp(-\lambda t)\int_0^t b_1\|\boldsymbol{e}_k(\tau)\|\exp(-\lambda\tau)\exp(\lambda\tau)\mathrm{d}\tau \le b_1\exp(-\lambda t)\|\boldsymbol{e}_k(\tau)\|_\lambda\int_0^t \exp(\lambda\tau)\mathrm{d}\tau \\ &= b_1\exp(-\lambda t)\|\boldsymbol{e}_k(\tau)\|_\lambda\frac{\exp(\lambda t)-1}{\lambda} = \frac{b_1}{\lambda}\|\boldsymbol{e}_k(\tau)\|_\lambda\exp(-\lambda t)(\exp(\lambda t)-1) \\ &= b_1\frac{(1-\exp(-\lambda t))}{\lambda}\|\boldsymbol{e}_k(\tau)\|_\lambda \le b_1\frac{(1-\exp(-\lambda T))}{\lambda}\|\boldsymbol{e}_k(\tau)\|_\lambda\end{aligned} \tag{12.19}$$

For $\forall t\in[0,T]$, $\forall\tau\in[0,t]$, $\forall\sigma\in[0,\tau]$, we have $\|\boldsymbol{e}_k(\sigma)\|_\lambda \le \|\boldsymbol{e}_k(\tau)\|_\lambda$.

From (12.19), we have

$$\begin{aligned}
\exp(-\lambda t)\int_0^t\int_0^\tau b_2\|\boldsymbol{e}_k(\sigma)\|\mathrm{d}\sigma\mathrm{d}\tau &= \exp(-\lambda t)\int_0^t \exp(\lambda\tau)\exp(-\lambda\tau)\int_0^\tau b_2\|\boldsymbol{e}_k(\sigma)\|\mathrm{d}\sigma\mathrm{d}\tau \\
&\le \exp(-\lambda t)\int_0^t \exp(\lambda\tau)b_2\frac{1-\exp(-\lambda T)}{\lambda}\|\boldsymbol{e}_k(\sigma)\|_\lambda\mathrm{d}\tau \\
&\le b_2\frac{1-\exp(-\lambda T)}{\lambda}\exp(-\lambda t)\int_0^t \exp(\lambda\tau)\|\boldsymbol{e}_k(\tau)\|_\lambda\mathrm{d}\tau \\
&= b_2\frac{1-\exp(-\lambda T)}{\lambda}\exp(-\lambda t)\|\boldsymbol{e}_k(\tau)\|_\lambda\int_0^t \exp(\lambda\tau)\mathrm{d}\tau \\
&= b_2\frac{1-\exp(-\lambda T)}{\lambda}\exp(-\lambda t)\|\boldsymbol{e}_k(\tau)\|_\lambda\frac{\exp(\lambda t)-1}{\lambda} \\
&= b_2\frac{1-\exp(-\lambda T)}{\lambda}\|\boldsymbol{e}_k(\tau)\|_\lambda\frac{1-\exp(-\lambda t)}{\lambda} \le b_2\left(\frac{1-\exp(-\lambda T)}{\lambda}\right)^2\|\boldsymbol{e}_k(\tau)\|_\lambda
\end{aligned}$$

where $0<\frac{1-\exp(-\lambda t)}{\lambda}\le\frac{1-\exp(-\lambda T)}{\lambda}$.

i.e.,

$$\exp(-\lambda t)\int_0^t\int_0^\tau b_2\|\boldsymbol{e}_k(\sigma)\|\mathrm{d}\sigma\mathrm{d}\tau \le b_2\left(\frac{1-\exp(-\lambda T)}{\lambda}\right)^2\|\boldsymbol{e}_k(\tau)\|_\lambda \tag{12.20}$$

Then, inserting (12.19) and (12.20) into (12.18), we have

$$\|\boldsymbol{e}_{k+1}\|_\lambda \le \tilde{\rho}\|\boldsymbol{e}_k\|_\lambda \tag{12.21}$$

where $\tilde{\rho}=\bar{\rho}+b_1\frac{1-\exp(-\lambda T)}{\lambda}+b_2\left(\frac{1-\exp(-\lambda T)}{\lambda}\right)^2$.

Since $\bar{\rho}<1$, when we choose λ larger value, we can guarantee $\tilde{\rho}<1$, and then $\lim_{k\to\infty}\|\boldsymbol{e}_k\|_\lambda=0$.

In (12.14), if we replace $\boldsymbol{e}(k)$ as $\boldsymbol{e}(k+1)$, then the controller becomes closed-loop PID-type ILC, and the convergence analysis is the same as Theorem 12.1.

12.5.3 Simulation Example

Consider two-input two-output linear system

$$\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix}=\begin{bmatrix}-2 & 3\\ 1 & 1\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix}+\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}\begin{bmatrix}u_1(t)\\ u_2(t)\end{bmatrix}$$

$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$

The ideal trajectory is

$$\begin{bmatrix} y_{1\mathrm{d}}(t) \\ y_{2\mathrm{d}}(t) \end{bmatrix} = \begin{bmatrix} \sin(3t) \\ \cos(3t) \end{bmatrix}, t \in [0, 1]$$

To guarantee the conditions in Theorem 12.1, from $\mathrm{CB} = \begin{bmatrix} 2 & 2 \\ 0 & 1 \end{bmatrix}$, let $\boldsymbol{\Gamma} = \begin{bmatrix} 0.95 & 0 \\ 0 & 0.95 \end{bmatrix}$, choosing $\mathbf{L} = \begin{bmatrix} 2.0 & 0 \\ 0 & 2.0 \end{bmatrix}$, $\boldsymbol{\Psi} = 0$ in (12.14), the initial states are set as $\begin{bmatrix} x_{1(0)}(0) \\ x_{2(0)}(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

Firstly, using PD open-loop control, the simulation results are shown in Figs. 12.4, 12.5, and 12.6, and then, using PD closed-loop control, the simulation results are shown in Figs. 12.7, 12.8, and 12.9.

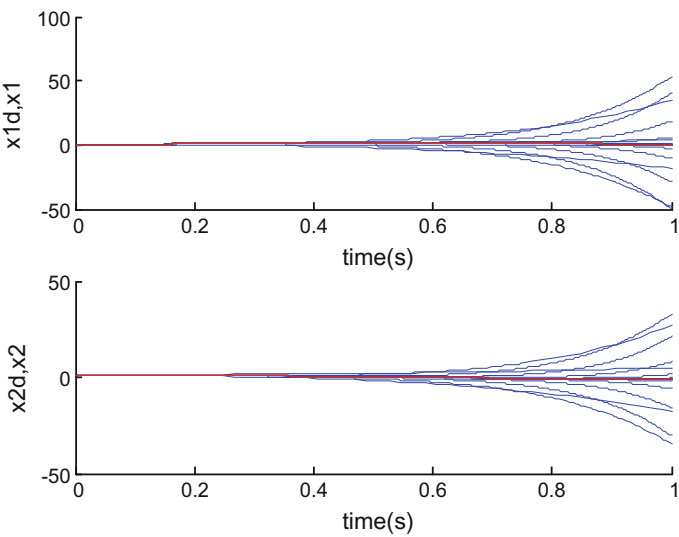


Fig. 12.4 x_i tracking during thirty times (open-loop PD control)

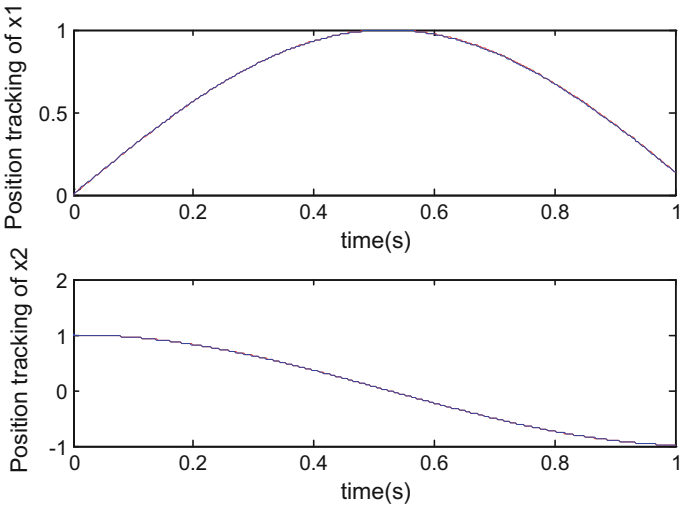


Fig. 12.5 Position tracking for thirty times (open-loop PD control)

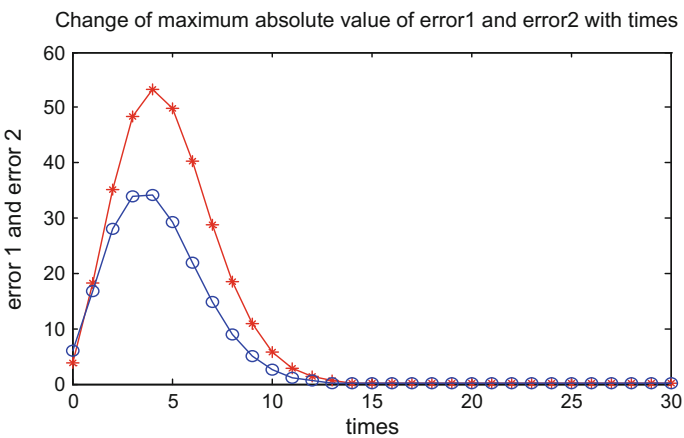


Fig. 12.6 Absolute maximum value of error during thirty times (open-loop PD control)

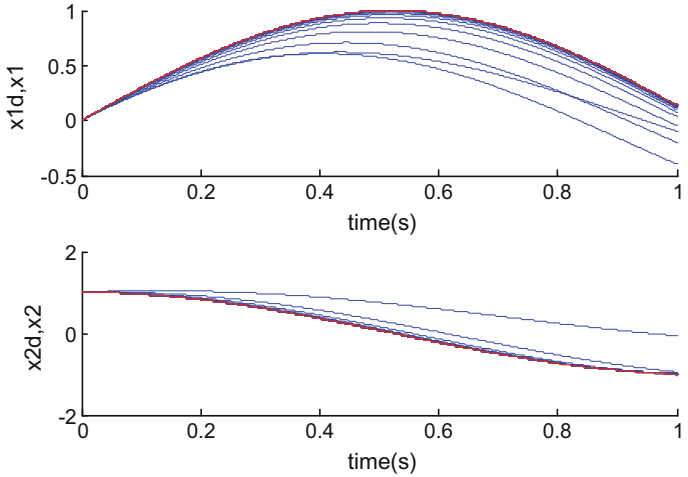


Fig. 12.7 x_i tracking during thirty times (closed PD control)

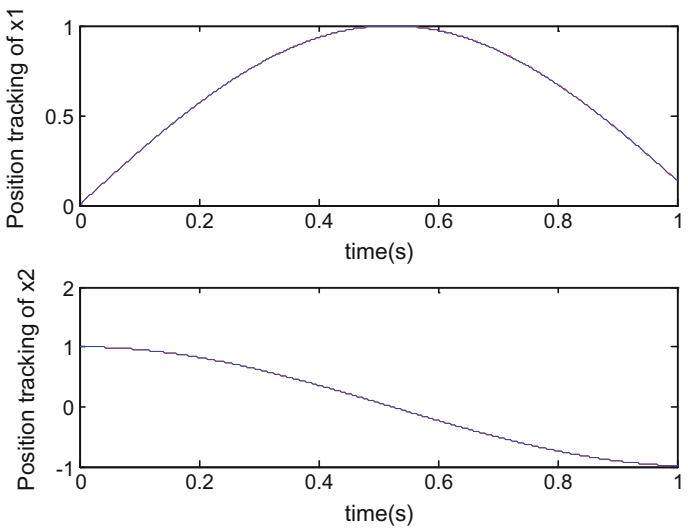


Fig. 12.8 Position tracking for thirty times (closed PD control)

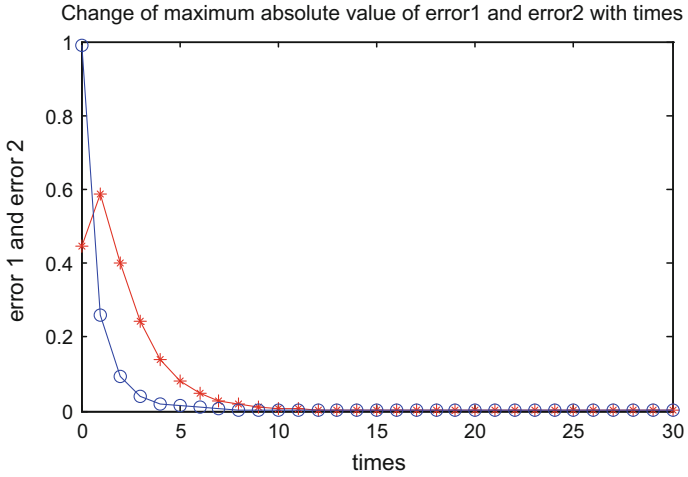


Fig. 12.9 Absolute maximum value of error during thirty times (closed PD control)

Simulink programs:

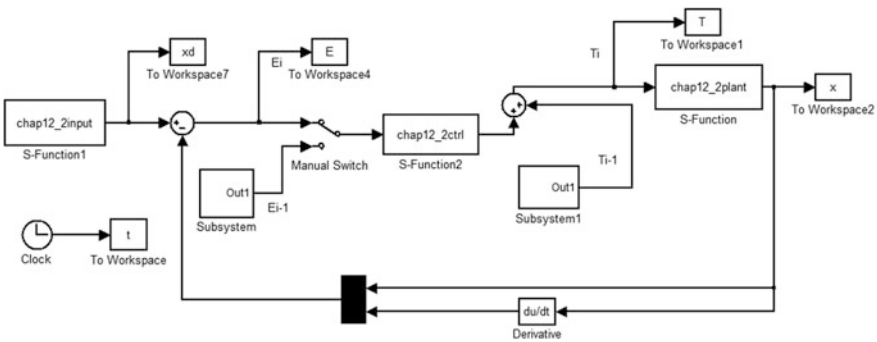
(1) Main program: chap12_2main.m

```
%Iterative D-Type Learning Control
clear all;
close all;

t=[0:0.01:1]';
k(1:101)=0;   %Total initial points
k=k';
T1(1:101)=0;
T1=T1';
T2=T1;
T=[T1 T2];

k1(1:101)=0;   %Total initial points
k1=k1';
E1(1:101)=0;
E1=E1';
E2=E1;
E3=E1;
E4=E1;
E=[E1 E2 E3 E4];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
M=30;
for i=0:1:M   % Start Learning Control
i
pause(0.01);

sim('chap12_2sim',[0,1]);

x1=x(:,1);
x2=x(:,2);

x1d=xd(:,1);
x2d=xd(:,2);
dx1d=xd(:,3);
dx2d=xd(:,4);

e1=E(:,1);
e2=E(:,2);
de1=E(:,3);
de2=E(:,4);
e=[e1 e2]';
de=[de1 de2]';

figure(1);
subplot(211);
hold on;
plot(t,x1,'b'',t,x1d,'r');
xlabel('time(s)'');ylabel('x1d,x1');

subplot(212);
hold on;
plot(t,x2,'b',t,x2d,'r');
xlabel('time(s)');ylabel('x2d,x2');

j=i+1;
times(j)=i;
e1i(j)=max(abs(e1));
e2i(j)=max(abs(e2));
de1i(j)=max(abs(de1));
de2i(j)=max(abs(de2));
end      %End of i
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(2);
subplot(211);
plot(t,x1d,'r',t,x1,'b');
xlabel('time(s)');ylabel('Position tracking of x1');
subplot(212);
plot(t,x2d,'r',t,x2,'b');
xlabel('time(s)');ylabel('Position tracking of x2');
```

```
figure(3);
subplot(211);
plot(t,T(:,1),'r');
xlabel('time(s)');ylabel('Control input 1');
subplot(212);
plot(t,T(:,2),'r');
xlabel('time(s)');ylabel('Control input 2');

figure(4);
plot(times,e1i,'*-r',times,e2i,'o-b');
title('Change of maximum absolute value of error1 and error2 with times');
xlabel('times');ylabel('error 1 and error 2');
```

(2) **Simulink program: chap12_2sim.mdl**



(3) **S function for plant: chap12_2plant.m**

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
```

```
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs    = 2;
sizes.NumInputs     = 2;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = [0;1];
str = [];
ts  = [0 0];
function sys=mdlDerivatives(t,x,u)
A=[-2 3;1 1];
C=[1 0;0 1];
B=[1 1;0 1];
Gama=0.95;
norm(eye(2)-C*B*Gama); % Must be smaller than 1.0

U=[u(1);u(2)];
dx=A*x+B*U;
sys(1)=dx(1);
sys(2)=dx(2);
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
```

(4) S function for controller: chap12_2ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs    = 2;
sizes.NumInputs     = 4;
```

```
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [0 0];
function sys=mdlOutputs(t,x,u)
e1=u(1);e2=u(2);
de1=u(3);de2=u(4);

e=[e1 e2]';
de=[de1 de2]';

Kp=2.0;
Gama=0.95;
Kd=Gama*eye(2);

Tol=Kp*e+Kd*de;      %PD Type

sys(1)=Tol(1);
sys(2)=Tol(2);
```

(5) S function for ideal trajectory: chap12_2input.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 4;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = [];
str = [];
```

```
ts = [0 0];
function sys=mdlOutputs(t,x,u)
x1d=sin(3*t);
dx1d=3*cos(3*t);
x2d=cos(3*t);
dx2d=-3*sin(3*t);

sys(1)=x1d;
sys(2)=x2d;
sys(3)=dx1d;
sys(4)=dx2d;
```

References

1. S. Arimoto, S. Kawamura, F. Miyazaki, Bettering operation of robotics by leaning. J. Rob. Syst. **1**(2), 123–140 (1984)
2. P.R. Ouyang, W.J. Zhang, M.M. Gupta, An adaptive switching learning control method for trajectory tracking of robot manipulators. Mechatronics **16**, 51–61 (2006)
3. A. Tayebi, Adaptive iterative learning control for robot manipulators. Automatica **40**, 1195–1203 (2004)
4. A. Mohammadi, M. Tavakoli, H.J. Marquez, F. Hashemzadeh, Nonlinear disturbance observer design for robotic manipulators. Eng. Practice **21**, 253–267 (2013)
5. S.L. Xie, S.P. Tian, Theory and application of iterative learning control. Science Press, China, (2005)