

Radial Basis Function (RBF) Neural Network Control for Mechanical Systems

Jinkun Liu

Radial Basis Function (RBF) Neural Network Control for Mechanical Systems

Design, Analysis and Matlab Simulation

With 170 figures



Jinkun Liu
Department of Intelligent System
and Control Engineering
School of Automation Science
and Electrical Engineering
Beihang University
Beijing, China, People's Republic

TUP ISBN 978-7-302-30255-1
ISBN 978-3-642-34815-0 ISBN 978-3-642-34816-7 (eBook)
DOI 10.1007/978-3-642-34816-7
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2012955859

© Tsinghua University Press, Beijing and Springer-Verlag Berlin Heidelberg 2013
This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part
of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations,
recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or
information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar
methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts
in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being
entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication
of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the
Publisher's location, in its current version, and permission for use must always be obtained from
Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center.
Violations are liable to prosecution under the respective Copyright Law.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this
publication does not imply, even in the absence of a specific statement, that such names are exempt
from the relevant protective laws and regulations and therefore free for general use.

While the advice and information in this book are believed to be true and accurate at the date of
publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for
any errors or omissions that may be made. The publisher makes no warranty, express or implied, with
respect to the material contained herein.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Recent years have seen a rapid development of neural network control techniques and their successful applications. Numerous theoretical studies and actual industrial implementations demonstrate that artificial neural network is a good candidate for function approximation and 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 adaptive neural control is still focused on the development of fundamental methodologies.

The advantage of neural networks is that a suitable number of neural network functions can model any (sufficiently smooth) continuous nonlinear function in a compact set, and the modeling error is becoming smaller with an increase of neural network functions. It is even possible to model discontinuous nonlinearities assuming the right choice of discontinuous neural network functions. Thus, an adaptive neural network approach 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 stable adaptive control system design using neural network approximation-based techniques. The main objectives of the book are to introduce the concrete design method and Matlab simulation of stable adaptive RBF (*Radial Basis Function*) neural control strategies.

It is our goal to accomplish these objectives:

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

This book provides the reader with a thorough grounding in the neural network control system design. Typical neural network controllers' designs are verified using Matlab simulation. In this book, concrete case studies, which present the results of neural network 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 adaptive control and neural network control in Chap. 1, RBF neural network algorithm and design remarks are given in Chap. 2, RBF neural network controller design based on gradient descent algorithm is introduced in Chap. 3, since only local optimization can be guaranteed by using the gradient descent method, and several adaptive RBF neural network controller designs are given based on Lyapunov analysis from Chaps. 4, 5, 6, 7 and 8, which include simple adaptive RBF neural network controller, neural network sliding mode controller, adaptive RBF controller based on global approximation, adaptive robust RBF controller based on local approximation, and backstepping adaptive controller with RBF. In Chap. 9, digital RBF neural network controller design is given. Two kinds of discrete neural network controllers are introduced in Chap. 10. At last, a neural network adaptive observer is recommended and a speedless sliding mode controller design is given in Chap. 11.

I would like to thank Prof. S. S. Ge for his fruitful suggestions. I wish to thank my family for their support and encouragement.

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://ljk.buaa.edu.cn/>. If you have questions about algorithms and simulation programs, please E-mail ljk@buaa.edu.cn.

Contents

1	Introduction	1
1.1	Neural Network Control	1
1.1.1	Why Neural Network Control?	1
1.1.2	Review of Neural Network Control	2
1.1.3	Review of RBF Adaptive Control	3
1.2	Review of RBF Neural Network	3
1.3	RBF Adaptive Control for Robot Manipulators	4
1.4	S Function Design for Control System	5
1.4.1	S Function Introduction	5
1.4.2	Basic Parameters in S Function	5
1.4.3	Examples	6
1.5	An Example of a Simple Adaptive Control System	7
1.5.1	System Description	7
1.5.2	Adaptive Control Law Design	7
1.5.3	Simulation Example	9
Appendix		11
References		15
2	RBF Neural Network Design and Simulation	19
2.1	RBF Neural Network Design and Simulation	19
2.1.1	RBF Algorithm	19
2.1.2	RBF Design Example with Matlab Simulation	20
2.2	RBF Neural Network Approximation Based on Gradient Descent Method	22
2.2.1	RBF Neural Network Approximation	22
2.2.2	Simulation Example	24
2.3	Effect of Gaussian Function Parameters on RBF Approximation	25
2.4	Effect of Hidden Nets Number on RBF Approximation	28

2.5 RBF Neural Network Training for System Modeling	33
2.5.1 RBF Neural Network Training	33
2.5.2 Simulation Example	34
2.6 RBF Neural Network Approximation	36
Appendix	37
References	53
3 RBF Neural Network Control Based on Gradient Descent Algorithm	55
3.1 Supervisory Control Based on RBF Neural Network	55
3.1.1 RBF Supervisory Control	55
3.1.2 Simulation Example	56
3.2 RBFNN-Based Model Reference Adaptive Control	58
3.2.1 Controller Design	58
3.2.2 Simulation Example	59
3.3 RBF Self-Adjust Control	61
3.3.1 System Description	61
3.3.2 RBF Controller Design	61
3.3.3 Simulation Example	63
Appendix	63
References	69
4 Adaptive RBF Neural Network Control	71
4.1 Adaptive Control Based on Neural Approximation	71
4.1.1 Problem Description	71
4.1.2 Adaptive RBF Controller Design	72
4.1.3 Simulation Examples	75
4.2 Adaptive Control Based on Neural Approximation with Unknown Parameter	79
4.2.1 Problem Description	79
4.2.2 Adaptive Controller Design	79
4.2.3 Simulation Examples	83
4.3 A Direct Method for Robust Adaptive Control by RBF	83
4.3.1 System Description	83
4.3.2 Desired Feedback Control and Function Approximation	86
4.3.3 Controller Design and Performance Analysis	87
4.3.4 Simulation Example	90
Appendix	92
References	112
5 Neural Network Sliding Mode Control	113
5.1 Typical Sliding Mode Controller Design	114
5.2 Sliding Mode Control Based on RBF for Second-Order SISO Nonlinear System	116
5.2.1 Problem Description	116

5.2.2	Sliding Mode Control Based on RBF for Unknown $f(\cdot)$	117
5.2.3	Simulation Example	118
5.3	Sliding Mode Control Based on RBF for Unknown $f(\cdot)$ and $g(\cdot)$	120
5.3.1	Introduction	120
5.3.2	Simulation Example	122
Appendix		123
References		132
6	Adaptive RBF Control Based on Global Approximation	133
6.1	Adaptive Control with RBF Neural Network Compensation for Robotic Manipulators	134
6.1.1	Problem Description	134
6.1.2	RBF Approximation	135
6.1.3	RBF Controller and Adaptive Law Design and Analysis	136
6.1.4	Simulation Examples	140
6.2	RBF Neural Robot Controller Design with Sliding Mode Robust Term	144
6.2.1	Problem Description	144
6.2.2	RBF Approximation	147
6.2.3	Control Law Design and Stability Analysis	147
6.2.4	Simulation Examples	148
6.3	Robust Control Based on RBF Neural Network with HJI	153
6.3.1	Foundation	153
6.3.2	Controller Design and Analysis	153
6.3.3	Simulation Examples	156
Appendix		159
References		191
7	Adaptive Robust RBF Control Based on Local Approximation	193
7.1	Robust Control Based on Nominal Model for Robotic Manipulators	193
7.1.1	Problem Description	193
7.1.2	Controller Design	194
7.1.3	Stability Analysis	195
7.1.4	Simulation Example	196
7.2	Adaptive RBF Control Based on Local Model Approximation for Robotic Manipulators	197
7.2.1	Problem Description	197
7.2.2	Controller Design	199
7.2.3	Stability Analysis	200
7.2.4	Simulation Examples	203

7.3	Adaptive Neural Network Control of Robot Manipulators in Task Space	205
7.3.1	Coordination Transformation from Task Space to Joint Space	208
7.3.2	Neural Network Modeling of Robot Manipulators	208
7.3.3	Controller Design	210
7.3.4	Simulation Examples	213
Appendix		217
References		249
8	Backstepping Control with RBF	251
8.1	Introduction	251
8.2	Backstepping Control for Inverted Pendulum	252
8.2.1	System Description	253
8.2.2	Controller Design	253
8.2.3	Simulation Example	254
8.3	Backstepping Control Based on RBF for Inverted Pendulum	255
8.3.1	System Description	255
8.3.2	Backstepping Controller Design	256
8.3.3	Adaptive Law Design	257
8.3.4	Simulation Example	259
8.4	Backstepping Control for Single-Link Flexible Joint Robot	260
8.4.1	System Description	260
8.4.2	Backstepping Controller Design	262
8.5	Adaptive Backstepping Control with RBF for Single-Link Flexible Joint Robot	265
8.5.1	Backstepping Controller Design with Function Estimation	265
8.5.2	Backstepping Controller Design with RBF Approximation	269
8.5.3	Simulation Examples	272
Appendix		276
References		291
9	Digital RBF Neural Network Control	293
9.1	Adaptive Runge–Kutta–Merson Method	293
9.1.1	Introduction	293
9.1.2	Simulation Example	295
9.2	Digital Adaptive Control for SISO System	295
9.2.1	Introduction	295
9.2.2	Simulation Example	297
9.3	Digital Adaptive RBF Control for Two-Link Manipulators	298
9.3.1	Introduction	298
9.3.2	Simulation Example	299
Appendix		299
References		309

10 Discrete Neural Network Control	311
10.1 Introduction	311
10.2 Direct RBF Control for a Class of Discrete-Time Nonlinear System	312
10.2.1 System Description	312
10.2.2 Controller Design and Stability Analysis	312
10.2.3 Simulation Examples	316
10.3 Adaptive RBF Control for a Class of Discrete-Time Nonlinear System	319
10.3.1 System Description	319
10.3.2 Traditional Controller Design	320
10.3.3 Adaptive Neural Network Controller Design	320
10.3.4 Stability Analysis	322
10.3.5 Simulation Examples	324
Appendix	329
References	337
11 Adaptive RBF Observer Design and Sliding Mode Control	339
11.1 Adaptive RBF Observer Design	339
11.1.1 System Description	339
11.1.2 Adaptive RBF Observer Design and Analysis	340
11.1.3 Simulation Examples	343
11.2 Sliding Mode Control Based on RBF Adaptive Observer	347
11.2.1 Sliding Mode Controller Design	347
11.2.2 Simulation Example	349
Appendix	351
References	362
Index	363

Table of Notation

Notation	Meaning
\mathbf{R}	The set of real numbers
\mathbf{R}^n	The set of all n -dimensional real vectors
$\mathbf{R}^{n \times m}$	The set of all $n \times m$ -dimensional real matrices
$ a $	The absolute value of scalar a
$\det(\mathbf{A})$	The determinant of matrix \mathbf{A}
$\ \mathbf{x}\ $	The norm of vector \mathbf{x}
\mathbf{A}^T	The transpose of \mathbf{A}
\mathbf{A}^{-1}	The inverse of \mathbf{A}
\mathbf{I}	An identity matrix
\mathbf{I}_n	The identity matrix of dimension $n \times n$
$\lambda_i(\mathbf{A})$	The i th eigenvalue of \mathbf{A}
$\lambda(\mathbf{A})$	The set of eigenvalues of \mathbf{A}
$\lambda_{\min}(\mathbf{A})$	The minimum eigenvalue of \mathbf{A} where $\lambda(\mathbf{A})$ are real
$\lambda_{\max}(\mathbf{A})$	The maximum eigenvalue of \mathbf{A} where $\lambda(\mathbf{A})$ are real
x_i	The i th element of vector \mathbf{A}
a_{ij}	The ij th element of matrix \mathbf{A}
(\bullet)	The estimate of (\bullet)
(\bullet)	(\bullet) - (\bullet)
$\sup \alpha(t)$	The smallest number that is larger than or equal to the maximum value of $\alpha(t)$
$\text{diag}[\cdot \cdot]$	Diagonal matrix with given diagonal elements
\mathbf{h}	Output vector of Gaussian function of RBF
\mathbf{c}_j	Center vector of the j th net of Gaussian function of RBF
b_j	Width of the j th net of Gaussian function of RBF
\mathbf{W}	Weight vector of RBF

List of Acronyms

BP	back-propagation
GL	Ge–Lee matrix
LIP	linear-in-the-parameters
MFN	multilayer feed-forward network
MIMO	multi-input multi-output
MNN	multilayer neural networks
NN	neural networks
RBF	radial basis function
RKM	Runge–Kutta–Merson
SISO	single-input single-output
SMC	sliding mode control

Chapter 1

Introduction

Abstract This chapter gives the review of several kinds of neural network control and introduces the concept of RBF neural network and RBF neural network control. To illustrate the attendant features of robustness and performance specification of RBF adaptive control, a typical RBF adaptive controller design for an example system is given. A concrete analysis, simulation examples, and Matlab programs are given too.

Keywords Neural network control • RBF neural network • Adaptive control

1.1 Neural Network Control

1.1.1 Why 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, non-linear 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, multi-loop, 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 theory 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:

- 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.

1.1.2 Review of Neural Network Control

Conventional methods of designing controllers for a MIMO plant like a multi-joint robot generally require, as a minimum, knowledge of the structure and accurate mathematical model of the plant. In many cases, the values of the parameters of the model also need to be precisely known.

Neural networks, which can learn the forward and inverse dynamic behaviors of complex plants online, offer alternative methods of realizing MIMO controllers capable of adapting to environmental changes. In theory, the design of a neural network-based control system is relatively straightforward as it does not require any prior knowledge about the plant.

The approximation abilities of neural networks have been proven in many research works [3–7], and many adaptive neural network controllers based on the approximation abilities are introduced in some books [8–14]; several research groups have involved in the developments of stable adaptive neural network control techniques.

There have been many papers to be published about neural network control. For example, a unified framework for identification and control of nonlinear dynamic systems was proposed in [15], in which the parametric method of both adaptive nonlinear control and adaptive linear control theory can be applied to

perform the stability analysis. Through introducing the Ge–Lee operator for ease of stability analysis and presentation, systematic and coherent treatments of the common problems in robot neural network control are given in [8]. The typical stable neural network approximation control schemes based on Lyapunov training design are given in [16–18].

The popularization of back-propagation (BP) neural network and RBF neural network have greatly boosted the development of neural control [19]. For example, many neural control approaches have been developed with BP neural network [20–28].

1.1.3 *Review of RBF Adaptive Control*

In recent years, the analytical study of adaptive nonlinear control systems using RBF universal function approximation has received much attention; typically, these methods are given in [29–37].

The RBF network adaptation can effectively improve the control performance against large uncertainty of the system. The adaptation law is derived using the Lyapunov method so that the stability of the entire system and the convergence of the weight adaptation are guaranteed.

Many simulation examples in this book have indicated that by using RBF control, significant improvement has been achieved when the system is subjected to a sudden change with system uncertainty.

1.2 Review of RBF Neural Network

In 1990, artificial neural networks were first proposed for the adaptive control of nonlinear dynamical systems [38]. Since that time, both multilayer neural networks (MNN) and radial basis function (RBF) networks have been used in numerous applications for the identification and control [39].

RBF neural networks were addressed in 1988 [40], 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 has shown that any nonlinear function over a compact set with arbitrary accuracy can be approximated by RBF neural network [41, 42]. There have been significant research efforts on RBF neural control for nonlinear systems [24, 43].

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 c vector that is a parameter vector of the same

dimension as the input vector \mathbf{x} ; the Euclidean distance between the center and the network input vector \mathbf{x} is defined by $\|\mathbf{x}(t) - \mathbf{c}_j(t)\|$.

The output of hidden layer can be produced through a nonlinear activation function $h_j(t)$ as follows:

$$h_j(t) = \exp\left(-\frac{\|\mathbf{x}(t) - \mathbf{c}_j(t)\|^2}{2b_j^2}\right), \quad j = 1, \dots, m \quad (1.1)$$

where b_j notes a positive scalar called a width and m notes the number of hidden nodes. The output layer is a linear weighted combination as follows:

$$y_i(t) = \sum_{j=1}^m w_{ji} h_j(t), \quad i = 1, \dots, n \quad (1.2)$$

where w are the output layer weights, n notes the number of outputs, and y notes the network output.

1.3 RBF Adaptive Control for Robot Manipulators

The control of a multi-input multi-output (MIMO) plant is a difficult problem when the plant is nonlinear and time varying and there are dynamic interactions between the plant variables. A good example of such a plant is a robot manipulator with two or more joints [44].

Robot manipulators have become increasingly important in the field of flexible automation. Through the years, considerable research effort has been made in their controller design. In order to achieve accurate trajectory tracking and good control performance, a number of control schemes have been developed. Computed torque control is one of the most intuitive schemes, which relies on the exact cancellation of the nonlinear dynamics of the manipulator system; however, such a scheme has the disadvantage of requiring the exact dynamic model. In practical engineering, the payload of the robot manipulator may vary during its operation, which is unknown in advance. To overcome these problems, adaptive control strategies for robot manipulators have been developed and have attracted the interest of many researchers, as shown in [45–47]. These adaptive control methods have the advantage, in general, of requiring no a priori knowledge of unknown parameters, such as the mass of the payload.

For rigid robotic manipulators, to relax the requirement for exact knowledge of dynamics, control techniques based on neural networks have been developed. Many books and papers have employed neural network-based schemes for stable tracking control of rigid-link manipulators [8–14, 48–52].

For flexible link manipulators, there are many works about neural network adaptive control [53–55]. For example, a neural controller was proposed for joint-position tracking of flexible link manipulators using singular perturbation techniques [53]; the key feature of the approach is that no exact knowledge of the dynamics of the robot arms is assumed for the controller, and no off-line training is required for the neural network. Neural network-based controllers for tip-position tracking of flexible link manipulators were developed by utilizing the modified output redefinition approach [54]; the a priori knowledge of the payload mass is not needed.

1.4 S Function Design for Control System

1.4.1 *S Function Introduction*

S function provides a powerful mechanism for extending the capabilities of Simulink. An S function is a computer language description of dynamic system. In the control system, S function can be used to describe controller algorithm, adaptive algorithm, and the plant differential dynamic equation.

1.4.2 *Basic Parameters in S Function*

1. S function routine: include initialization routine, `mdlDerivative` routine, and `mdlOutput` routine.
2. `NumContStates`: to express the number of continuous states.
3. `NumDiscStates`: to express the number of discrete states.
4. `NumOutputs` and `NumInputs`: to express the number of input and output of the system.
5. `DirFeedthrough`: means that the output or the variable sample time is controlled directly by the value of an input port.
An example of a system that requires its inputs (i.e., has direct feedthrough) is the operation $y = k \times u$, where u is the input, k is the gain, and y is the output. An example of a system that does not require its inputs (i.e., does not have direct feedthrough) is the equation $y = x, \dot{x} = u$, where x is the state, u is the input, and y is the output.
6. `NumSampleTimes`: Simulink provides the following options for sample times, such as continuous sample time, discrete sample time, and variable sample time. For continuous sample time, the output changes in minor steps.

1.4.3 Examples

In the control system, we can use S function to describe controller, adaptive law, and plant. Consider Sect. 1.5, we give the explanation as follows:

1. Initialization routine for the plant

Consider S function to describe the plant dynamic equation as $m\ddot{x} = u$; we notice the plant is a second-order continuous system. For the S function, if we want to use two inputs and three outputs and initialize the plant as [0,5,0], consider the output is not controlled directly by the value of an input port, we can write initialization routine as:

```
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=1;
sys=simsizes(sizes);
x0=[0.5,0];
str=[];
ts=[0 0];
```

2. mdlDerivative routine for the plant

In the control system, the derivative S function can be used to describe the dynamic plant equation or adaptive law. For example, consider the plant $m\ddot{x} = u$; below is the program.

```
function sys=mdlDerivatives(t,x,u)
m=2;
ut=u(2);
sys(1)=x(2);
sys(2)=1/m*ut;
```

3. mdlDerivative routine for adaptive law

In the control system, the derivative S function can be used to describe the adaptive law. For example, to realize adaptive law $\dot{m} = -\gamma vs$, below is the program.

```
function sys=mdlDerivatives(t,x,u)
xm=u(1);
dxm=u(2);
ddxm=u(3);
x1=u(4);
dx1=u(5);
e=x1-xm;
```

```

de=dx1-dxm;
nmn=6;
s=de+nmn*e;
v=ddxm-2*nmn*de-nmn^2*e;
gama=0.5;
sys(1)=-gama*v*s;

```

4. mdlOutput routine for plant

In the control system, the output routine in S function can be used to describe controller or output of the plant. For example, to express the output of the plant, the program is

```

function sys=mdlOutputs(t,x,u)
m=2;
sys(1)=x(1);
sys(2)=x(2);
sys(3)=m;

```

1.5 An Example of a Simple Adaptive Control System

In this section, based on [56], we give a simulation example of a simple adaptive control system.

1.5.1 System Description

Consider the control of a mass m by a motor force u , with the plant dynamics being

$$m\ddot{x} = u. \quad (1.3)$$

Assume that a human operator provides the positioning command $r(t)$ to the control system. A reference model with command $r(t)$ is

$$\ddot{x}_m + \lambda_1 \dot{x}_m + \lambda_2 x_m = \lambda_2 r(t) \quad (1.4)$$

where λ_1 and λ_2 are positive constant values; $\tilde{x} = x - x_m$ represent the tracking error.

1.5.2 Adaptive Control Law Design

If m is known, an ideal design adaptive control law can be designed easily as

$$u = m(\ddot{x}_m - 2\lambda\dot{\tilde{x}} - \lambda^2\tilde{x}) \quad (1.5)$$

where λ a strictly positive number.

Submitting (1.5) into (1.3), we can get the exponential convergent tracking error dynamics

$$\ddot{\tilde{x}} + 2\lambda\dot{\tilde{x}} + \lambda^2\tilde{x} = 0. \quad (1.6)$$

If m is unknown, an adaptive control law was proposed as [56]

$$u = \hat{m}(\ddot{x}_m - 2\lambda\dot{\tilde{x}} - \lambda^2\tilde{x}) \quad (1.7)$$

where \hat{m} is estimation of m .

Let $v = \ddot{x}_m - 2\lambda\dot{\tilde{x}} - \lambda^2\tilde{x}$; submitting (1.7) into (1.3), the control law leads to

$$m\ddot{x} = \hat{m}(\ddot{x}_m - 2\lambda\dot{\tilde{x}} - \lambda^2\tilde{x}) = \hat{m}v.$$

Let $\tilde{m} = \hat{m} - m$; the above equation leads to

$$m(\ddot{x} - v) = \tilde{m}v. \quad (1.8)$$

Define s as a combined tracking error measure

$$s = \dot{\tilde{x}} + \lambda\tilde{x}. \quad (1.9)$$

Due to the relation (1.9), we know the convergence of s to zero implies the convergence of the position tracking error \tilde{x} and the velocity tracking error $\dot{\tilde{x}}$.

Since $\ddot{x} - v = \ddot{x} - \ddot{x}_m + 2\lambda\dot{\tilde{x}} + \lambda^2\tilde{x} = \ddot{x} + \lambda\dot{\tilde{x}} + \lambda(\dot{\tilde{x}} + \lambda\tilde{x}) = \dot{s} + \lambda s$, then (1.8) leads to

$$m(\dot{s} + \lambda s) = \tilde{m}v. \quad (1.10)$$

Define Lyapunov function as

$$V = \frac{1}{2} \left(ms^2 + \frac{1}{\gamma} \tilde{m}^2 \right).$$

Then

$$\dot{V} = m s \dot{s} + \frac{1}{\gamma} \tilde{m} \dot{\tilde{m}} = m s \dot{s} + \frac{1}{\gamma} \tilde{m} \dot{m}.$$

An adaptive law for parameter \hat{m} was proposed as [56]

$$\dot{\hat{m}} = -\gamma vs. \quad (1.11)$$

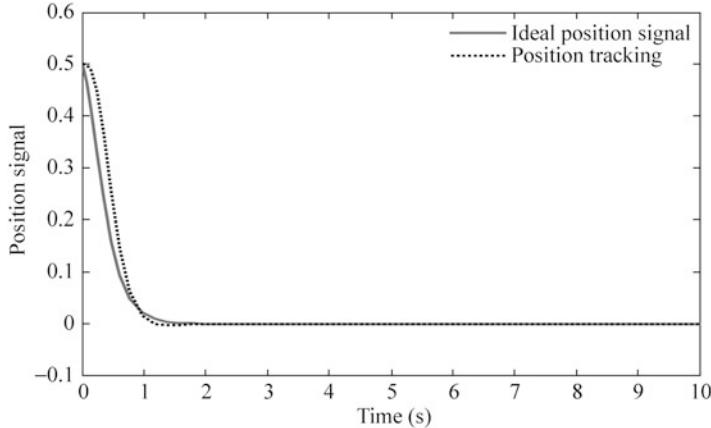


Fig. 1.1 Position tracking with $r(t) = 0$ ($M = 1$)

Consider (1.10), we get

$$\begin{aligned}\dot{V} &= mss + \frac{1}{\gamma} \tilde{m}(-\gamma vs) \\ &= mss - \tilde{m}vs = mss - (\dot{m}s + \lambda ms)s = -\lambda ms^2.\end{aligned}$$

Using Barbalat's lemma, one can easily show that s converges to zero, then the position tracking error \tilde{x} and the velocity tracking error $\dot{\tilde{x}}$ all converge to zero.

1.5.3 Simulation Example

For the system (1.3), the true mass is assumed as $m = 2$. In the simulation, the initial value is set as $\hat{m}(0) = 0$, the control law (1.7) with adaptive law (1.11) is used, the parameters is set as $\gamma = 0.5$, $\lambda_1 = 10$, $\lambda_2 = 25$, $\lambda = 6$, and the commanded position is chosen $r(t) = 0$ and $r(t) = \sin(4t)$, respectively, with initial conditions being $\dot{x}(0) = \dot{x}_m(0) = 0$, $x(0) = x_m(0) = 0.5$.

Figures 1.1 and 1.2 show the results when the desired position is $r(t) = 0$; Figs. 1.3 and 1.4 shows the results when the desired position is $r(t) = \sin(4t)$.

About the convergence analysis of parameter error in adaptive control system, concrete explanation was given in [56]. In this example, it is clear that the position tracking errors in both cases converge to zero, while the parameter error converge to zero only for the latter case. The reason for the non-convergence of parameter error in the first case can be explained by the simplicity of the tracking task: the asymptotic tracking of $x_m(t)$ can be achieved by many possible values of the estimated parameter \hat{m} , besides the true parameter m . Therefore, the parameter adaptation law does not bother to find out the true parameter. On the other hand,

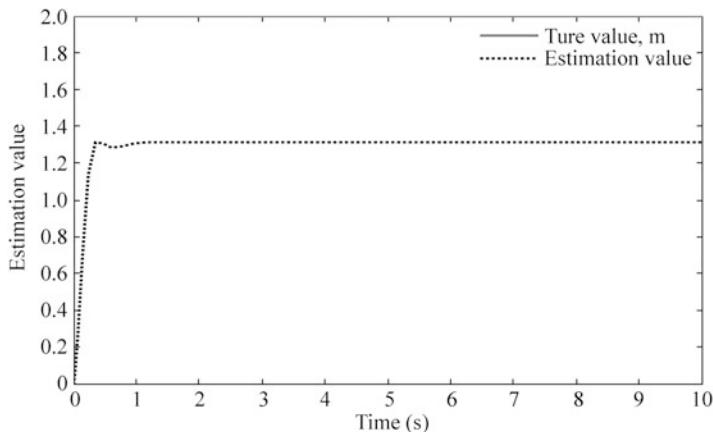


Fig. 1.2 Parameter estimation for an unknown mass with $r(t) = 0$ ($M = 1$)

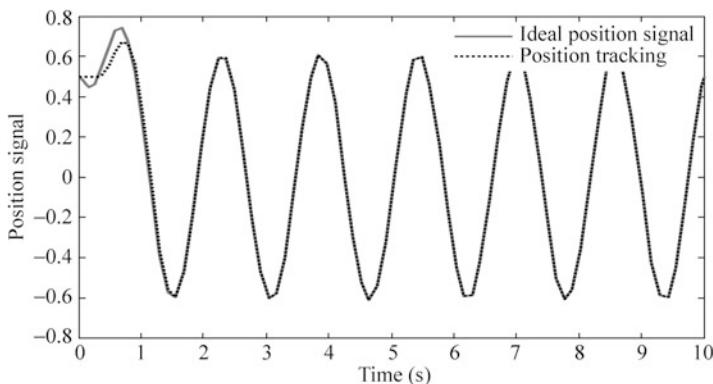


Fig. 1.3 Position tracking with $r(t) = \sin(4t)$ ($M = 2$)

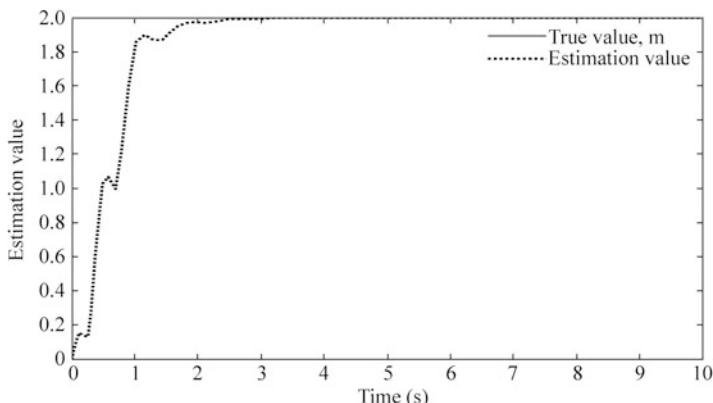


Fig. 1.4 Parameter estimation for an unknown mass with $r(t) = \sin(4t)$ ($M = 2$)

the convergence of the parameter error in Fig. 1.3 is because of the complexity of the tracking task, that is, tracking error convergence can be achieved only when the true mass is used in the control law.

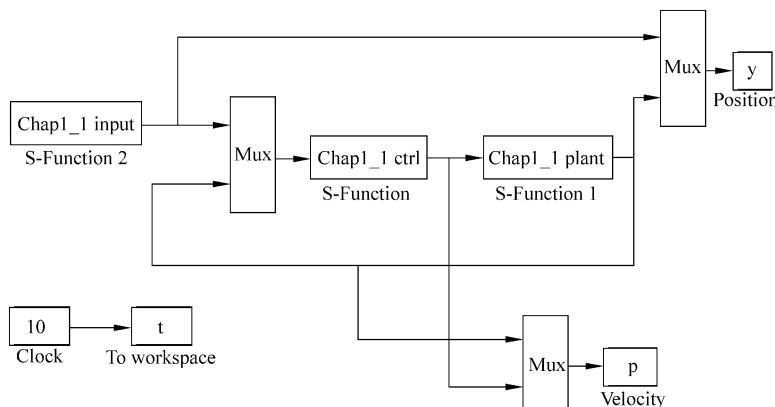
In neural network adaptive control, neural network often be used to approximate to unknown nonlinear system. For the same reason, the convergence of the approximation error often cannot be achieved.

The Simulink program of this example is chap1_1sim.mdl; the Matlab programs of the example are given in the [Appendix](#).

Appendix

Programs for Sect. 1.5.3

1. Simulink main program: chap1_1sim.mdl



2. Controller program: chap1_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
sizes = simsizes;
sizes.NumContStates = 1;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 6;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [0];
str = [];
ts = [];
function sys=mdlDerivatives(t,x,u)
xm=u(1);
dxm=u(2);
ddxm=u(3);
x1=u(4);
dx1=u(5);

e=x1-xm;
de=dx1-dxm;

nmn=6;
s=de+nmn*e;
v=ddxm-2*nmn*de-nmn^2*e;

gama=0.5;
sys(1)=-gama*v*s;
function sys=mdlOutputs(t,x,u)
xm=u(1);
dxm=u(2);
ddxm=u(3);
x1=u(4);
dx1=u(5);

e=x1-xm;
de=dx1-dxm;

nmn=6;
mp=x(1);
```

```
ut=mp*(ddxm-2*nmn*de-nmn^2*e);
```

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

3. Plant program: chap1_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 = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [0.5,0];
str = [];
ts = [0 0];
function sys=mdlDerivatives(t,x,u)
m=2;
ut=u(2);
sys(1)=x(2);
sys(2)=1/m*ut;
function sys=mdlOutputs(t,x,u)
m=2;
sys(1)=x(1);
sys(2)=x(2);
sys(3)=m;
```

4. Input program: chap1_1input.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 M
M=2;
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 0;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [0.5,0];
str = [];
ts = [];
function sys=mdlDerivatives(t,x,u)
global M
if M==1
r=0;
elseif M==2
r=sin(4*t);
end
nmn1=10;
nmn2=25;
sys(1)=x(2);
sys(2)=-nmn1*x(2)-nmn2*x(1)+nmn2*r;
function sys=mdlOutputs(t,x,u)
global M
if M==1
r=0;
elseif M==2
r=sin(4*t);
end
nmn1=10;
nmn2=25;
```

```

xm=x(1);
dxm=x(2);
ddxm=-nmn1*x(2)-nmn2*x(1)+nmn2*r;

sys(1)=xm;
sys(2)=dxm;
sys(3)=ddxm;

```

5. Plot program: chap1_1plot.m

```

close all;

figure(1);
plot(t,y(:,1),'r',t,y(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('position signal');
legend('ideal position signal','position tracking');
figure(2);
plot(t,p(:,3),'r',t,p(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('estimation value');
legend('True value,m','estimation value');

```

References

- McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. *Bull Math Biophys* 5:115–133
- Hunt KJ, Sbarbaro D, Zbikowski R, Gawthrop PJ (1992) Neural networks for control system—a survey. *Automatica* 28(6):1083–1112
- Barron AR (1991) Approximation and estimation bounds for artificial neural networks. In: Proceedings of the 4th annual workshop on computational learning theory. Morgan Kaufmann, San Mateo, pp 243–249
- Barron AR (1993) Universal approximation bounds for superposition for a sigmoidal function. *IEEE Trans Inf Theory* 39(3):930–945
- Chen TP, Chen H (1995) Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks. *IEEE Trans Neural Netw* 6 (4):904–910
- Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximator. *Neural Netw* 2(5):359–366
- Poggio T, Girosi T (1990) Networks for approximation and learning. *Proc IEEE* 78 (9):1481–1497
- Ge SS, Lee TH, Harris CJ (1998) Adaptive neural network control of robotic manipulators. World Scientific, London
- Ge SS, Hang CC, Lee TH, Zhang T (2001) Stable adaptive neural network control. Kluwer, Boston
- Lewis FL, Jagannathan S, Yesildirek A (1999) Neural network control of robot manipulators and nonlinear systems. Taylor & Francis, London
- Lewis FL, Campos J, Selmic R (2002) *Neuro-fuzzy control of industrial systems with actuator nonlinearities*. SIAM, Philadelphia
- Talebi HA, Patel RV, Khorasani K (2000) Control of flexible-link manipulators using neural networks. Springer, London/New York

13. Kim YH, Lewis FL (1998) High-level feedback control with neural networks. World Scientific, Singapore/River Edge
14. Fabri SG, Kadirkamanathan V (2001) Functional adaptive control: an intelligent systems approach. Springer, New York
15. Polycarpou MM (1996) Stable adaptive neural control scheme for nonlinear systems. *IEEE Trans Autom Control* 41(3):447–451
16. Zhang TP, Ge SS (2009) Adaptive neural network tracking control of MIMO nonlinear systems with unknown dead zones and control directions. *IEEE Trans Neural Netw* 20 (3):483–497
17. Yu SH, Annaswamy AM (1997) Adaptive control of nonlinear dynamic systems using θ -adaptive neural networks. *Automatica* 33(11):1975–1995
18. Yu SH, Annaswamy AM (1998) Stable neural controllers for nonlinear dynamic systems. *Automatica* 34(5):641–650
19. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning internal representations by error propagation. *Parallel Distrib Process* 1:318–362
20. Chen FC (1990) Back-propagation neural networks for nonlinear self-tuning adaptive control. *IEEE Control Syst Mag* 10(3):44–48
21. Fierro R, Lewis FL (1998) Control of a nonholonomic mobile robot using neural networks. *IEEE Trans Neural Netw* 9(4):589–600
22. Jagannathan S, Lewis FL (1996) Identification of nonlinear dynamical systems using multi-layered neural networks. *Automatica* 32(12):1707–1712
23. Kwan C, Lewis FL, Dawson DM (1998) Robust neural-network control of rigid-link electrically driven robots. *IEEE Trans Neural Netw* 9(4):581–588
24. Lewis FL, Liu K, Yesildirek A (1995) Neural net robot controller with guaranteed tracking performance. *IEEE Trans Neural Netw* 6(3):703–715
25. Lewis FL, Parisini T (1998) Neural network feedback with guaranteed stability. *Int J Control* 70:337–339
26. Lewis FL, Yesildirek A, Liu K (1996) Multilayer neural-net robot controller with guaranteed tracking performance. *IEEE Trans Neural Netw* 7(2):388–399
27. Yesildirek A, Lewis FL (1995) Feedback linearization using neural networks. *Automatica* 31 (11):1659–1664
28. Jagannathan S, Lewis FL (1996) Discrete-time neural net controller for a class of nonlinear dynamical systems. *IEEE Trans Autom Control* 41(11):1693–1699
29. Sundararajan N, Saratchandran P, Li Y (2002) Fully tuned radial basis function neural networks for flight control. Kluwer, Boston
30. Sanner RM, Slotine JE (1992) Gaussian networks for direct adaptive control. *IEEE Trans Neural Netw* 3(6):837–863
31. Seshagiri S, Khalil HK (2000) Output feedback control of nonlinear systems using RBF neural networks. *IEEE Trans Neural Netw* 11(1):69–79
32. Huang SN, Tan KK, Lee TH (2002) Adaptive motion control using neural network approximations. *Automatica* 38(2):227–233
33. Ge SS, Wang C (2002) Direct adaptive NN control of a class of nonlinear systems. *IEEE Trans Neural Netw* 13(1):214–221
34. Li Y, Qiang S, Zhuang X, Kaynak O (2004) Robust and adaptive backstepping control for nonlinear systems using RBF neural networks. *IEEE Trans Neural Netw* 15(3):693–701
35. Huang S, Tan KK, Lee TH, Putra AS (2007) Adaptive control of mechanical systems using neural networks. *IEEE Trans Syst Man, Cybern Part C* 37(5):897–903
36. Wang SW, Yu DL (2008) Adaptive RBF network for parameter estimation and stable air-fuel ratio control. *Neural Netw* 21(1):102–112
37. Zhu Q, Fei S, Zhang T, Li T (2008) Adaptive RBF neural-networks control for a class of time-delay nonlinear systems. *Neurocomputing* 71(16–18):3617–3624
38. Narendra KS, Parthasarathy K (1990) Identification and control of dynamical systems using neural networks. *IEEE Trans Neural Netw* 1(1):4–27

39. Narendra KS, Mukhopadhyay S (1994) Adaptive control of nonlinear multivariable systems using neural networks. *Neural Netw* 7(5):737–752
40. Broomhead DS, Lowe D (1988) Multivariable functional interpolation and adaptive networks. *Complex Syst* 2:321–355
41. Hartman EJ, Keeler JD, Kowalski JM (1990) Layered neural networks with Gaussian hidden units as universal approximations. *Neural Comput* 2(2):210–215
42. Park J, Sandberg LW (1991) Universal approximation using radial-basis-function networks. *Neural Comput* 3(2):246–257
43. Kobayashi H, Ozawa R (2003) Adaptive neural network control of tendon-driven mechanisms with elastic tendons. *Automatica* 39(9):1509–1519
44. Slotine JJ, Li W (1987) On the adaptive control of robot manipulators. *Int J Robot Res* 6 (3):49–59
45. Craig JJ, Hsu P, Sastry SS (1987) Adaptive control of mechanical manipulators. *Int J Robot Res* 6(2):16–28
46. Astolfi A, Karagiannis D, Ortega R (2008) Nonlinear and adaptive control with applications. Springer, London
47. Alonge F, Ippolito FD, Raimondi FM (2003) An adaptive control law for robotic manipulator without velocity feedback. *Control Eng Pract* 11(9):999–1005
48. Omidvar O, Elliott DL (1997) Neural systems for control. Academic Press, San Diego
49. Lewis FL, Yesildirek A, Liu K (1996) Multilayer neural-net robot controller with guaranteed tracking performance. *IEEE Trans Neural Netw* 7(2):1–11
50. Miyamoto H, Kawato M, Setoyama T, Suzuki R (1988) Feedback error learning neural network for trajectory control of a robotic manipulator. *Neural Netw* 1(3):251–265
51. Ozaki T, Suzuki T, Furuhashi T, Okuma S, Uchikawa Y (1991) Trajectory control of robotic manipulators using neural networks. *IEEE Trans Ind Electron* 38(3):195–202
52. Saad M, Dessaint LA, Bigras P, Al-haddad K (1994) Adaptive versus neural adaptive control: application to robotics. *Int J Adapt Control Signal Process* 8(3):223–236
53. Yesildirek A, Vandegrift MW, Lewis FL (1996) A neural network controller for flexible-link robots. *J Intell Robot Syst* 17(4):327–349
54. Talebi HA, Khorasani K, Patel RV (1998) Neural network based control schemes for flexible-link manipulators: simulation and experiments. *Neural Netw* 11(7–8):1357–1377
55. Cheng XP, Patel RV (2003) Neural network based tracking control of a flexible macro–micro manipulator system. *Neural Netw* 16(2):271–286
56. Slotine JE, Li W (1991) Applied nonlinear control. Prentice Hall, Englewood Cliffs

Chapter 2

RBF Neural Network Design and Simulation

Abstract This chapter introduces RBF neural network design method, gives RBF neural network approximation algorithm based on gradient descent, analyzes the effects of Gaussian function parameters on RBF approximation, and introduces RBF neural network modeling method based on off-line training. Several simulation examples are given.

Keywords Neural network control • Gradient descent rule • Gaussian function • RBF training

2.1 RBF Neural Network Design and Simulation

2.1.1 RBF Algorithm

The structure of a typical three-layer RBF neural network is shown as Fig. 2.1.

In RBF neural network, $\mathbf{x} = [x_i]^T$ is input vector. Assuming there are m th neural nets, and radial-basis function vector in hidden layer of RBF is $\mathbf{h} = [h_j]^T$, h_j is Gaussian function value for neural net j in hidden layer, and

$$h_j = \exp\left(-\frac{\|\mathbf{x} - c_j\|^2}{2b_j^2}\right) \quad (2.1)$$

where $\mathbf{c} = [c_{ij}] = \begin{bmatrix} c_{11} & \cdots & c_{1m} \\ \vdots & \cdots & \vdots \\ c_{n1} & \cdots & c_{nm} \end{bmatrix}$ represents the coordinate value of center point of the

Gaussian function of neural net j for the i th input, $i = 1, 2, \dots, n$, $j = 1, 2, \dots, m$. For the vector $\mathbf{b} = [b_1, \dots, b_m]^T$, b_j represents the width value of Gaussian function for neural net j .

Fig. 2.1 RBF neural network structure

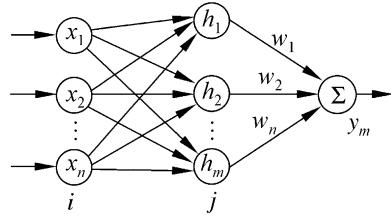
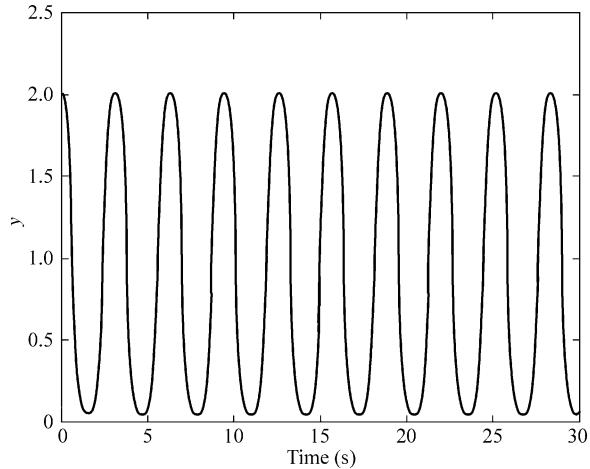


Fig. 2.2 Output of RBF



The weight value of RBF is

$$\mathbf{w} = [w_1, \dots, w_m]^T. \quad (2.2)$$

The output of RBF neural network is

$$y(t) = \mathbf{w}^T \mathbf{h} = w_1 h_1 + w_2 h_2 + \dots + w_m h_m. \quad (2.3)$$

2.1.2 RBF Design Example with Matlab Simulation

2.1.2.1 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 $\mathbf{b} = [b_1 \ b_2 \ b_3 \ b_4 \ b_5]^T$, $\mathbf{c} = [c_{11} \ c_{12} \ c_{13} \ c_{14} \ c_{15}]$, $\mathbf{h} = [h_1 \ h_2 \ h_3 \ h_4 \ h_5]^T$, $\mathbf{w} = [w_1 \ w_2 \ w_3 \ w_4 \ w_5]^T$, and $y(t) = \mathbf{w}^T \mathbf{h} = w_1 h_1 + w_2 h_2 + w_3 h_3 + w_4 h_4 + w_5 h_5$.

Choose the input as $\sin t$; the output of RBF is shown in Fig. 2.2, and the output of hidden neural net is shown in Fig. 2.3.

Fig. 2.3 Output of hidden neural net

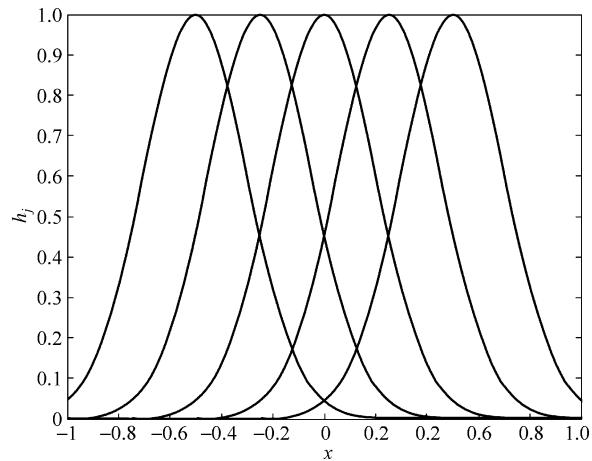
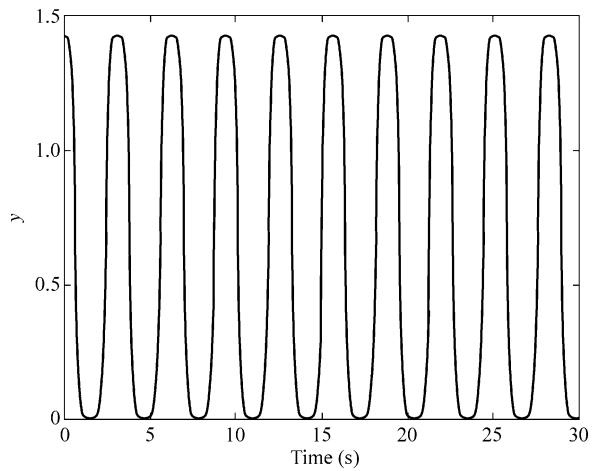


Fig. 2.4 Output of RBF



The Simulink program of this example is chap2_1sim.mdl, and the Matlab programs of the example are given in the [Appendix](#).

2.1.2.2 For Structure 2-5-1 RBF Neural Network

Consider a structure 2-5-1 RBF neural network; we have $\mathbf{x} = [x_1, x_2]^T$, $\mathbf{b} = [b_1 \ b_2 \ b_3 \ b_4 \ b_5]^T$, $\mathbf{c} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \end{bmatrix}$, $\mathbf{h} = [h_1 \ h_2 \ h_3 \ h_4 \ h_5]^T$, $\mathbf{w} = [w_1 \ w_2 \ w_3 \ w_4 \ w_5]^T$, and $y(t) = \mathbf{w}^T \mathbf{h} = w_1 h_1 + w_2 h_2 + w_3 h_3 + w_4 h_4 + w_5 h_5$.

Two inputs are chosen as $\sin t$. The output of RBF is shown in Fig. 2.4, and the output of hidden neural net is shown in Figs. 2.5 and 2.6.

Fig. 2.5 Output of hidden neural net for input 1

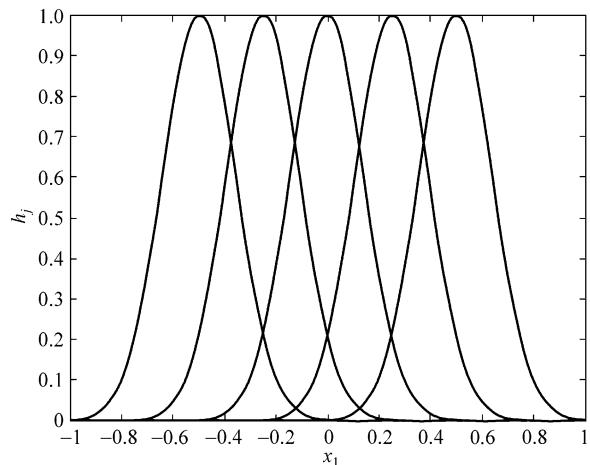
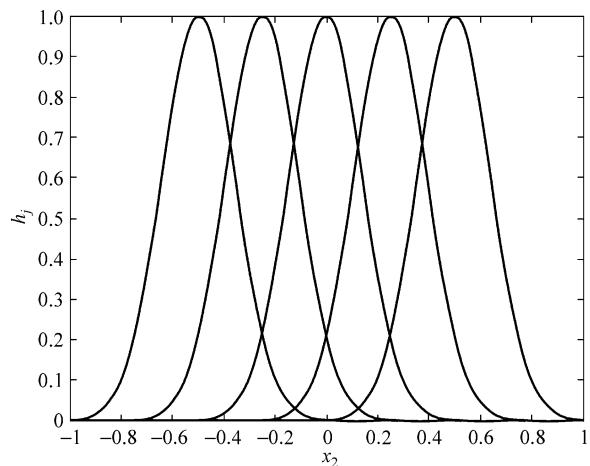


Fig. 2.6 Output of hidden neural net for input 2



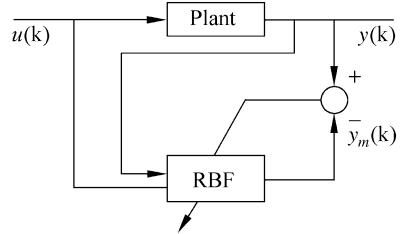
The Simulink program of this example is chap2_2sim.mdl, and the Matlab programs of the example are given in the [Appendix](#).

2.2 RBF Neural Network Approximation Based on Gradient Descent Method

2.2.1 RBF Neural Network Approximation

We use RBF neural network to approximate a plant; the structure is shown in Fig. 2.7.

Fig. 2.7 RBF neural network approximation



In RBF neural network, $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]^T$ is the input vector, and h_j is Gaussian function for neural net j , then

$$h_j = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_j\|^2}{2b_j^2}\right), \quad j = 1, 2, \dots, m. \quad (2.4)$$

where $\mathbf{c}_j = [c_{j1}, \dots, c_{jn}]$ is the center vector of neural net j .

The width vector of Gaussian function is

$$\mathbf{b} = [b_1, \dots, b_m]^T$$

where $b_j > 0$ represents the width value of Gaussian function for neural net j .

The weight value is

$$\mathbf{w} = [w_1, \dots, w_m]^T \quad (2.5)$$

The output of RBF is

$$y_m(t) = w_1 h_1 + w_2 h_2 + \dots + w_m h_m. \quad (2.6)$$

The performance index function of RBF is

$$E(t) = \frac{1}{2}(y(t) - y_m(t))^2. \quad (2.7)$$

According to gradient descent method, the parameters can be updated as follows:

$$\Delta w_j(t) = -\eta \frac{\partial E}{\partial w_j} = \eta(y(t) - y_m(t))h_j$$

$$w_j(t) = w_j(t-1) + \Delta w_j(t) + \alpha(w_j(t-1) - w_j(t-2)) \quad (2.8)$$

$$\Delta b_j = -\eta \frac{\partial E}{\partial b_j} = \eta(y(t) - y_m(t))w_j h_j \frac{\|\mathbf{x} - \mathbf{c}_j\|^2}{b_j^3} \quad (2.9)$$

$$b_j(t) = b_j(t-1) + \Delta b_j + \alpha(b_j(t-1) - b_j(t-2)) \quad (2.10)$$

$$\Delta c_{ji} = -\eta \frac{\partial E}{\partial c_{ji}} = \eta(y(t) - y_m(t))w_j \frac{x_j - c_{ji}}{b_j^2} \quad (2.11)$$

$$c_{ji}(t) = c_{ji}(t-1) + \Delta c_{ji} + \alpha(c_{ji}(t-1) - c_{ji}(t-2)) \quad (2.12)$$

where $\eta \in (0, 1)$ is the learning rate and $\alpha \in (0, 1)$ is momentum factor.

In RBF neural network approximation, the parameters of c_i and b_i must be chosen according to the scope of the input value. If the parameter values 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 c_i and b_i in RBF neural network approximation.

If the initial c_j and b are set in the effective range of input of RBF, we can only update weight value with fixed c_j and b .

2.2.2 Simulation Example

2.2.2.1 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, and we choose $x(1) = u(t)$, $x(2) = y(t)$, and $\alpha = 0.05$, $\eta = 0.5$. The initial weight value is chosen as random value between 0 and 1. Consider the range of the first input is $[0,1]$ and the range of the second input is about $[0,10]$; we choose the initial parameters of Gaussian function as $c_j = \begin{bmatrix} -1 & -0.5 & 0 & 0.5 & 1 \\ -10 & -5 & 0 & 5 & 10 \end{bmatrix}^T$, $b_j = 1.5$, $j = 1, 2, 3, 4, 5$.

Choose the input as $u(t) = \sin t$: in the simulation, we only update w with fixed c_j and b in RBF neural network approximation. The results are shown in Fig. 2.8.

The Simulink program of this example is chap2_3sim.mdl, and the Matlab programs of the example are given in the [Appendix](#).

2.2.2.2 Second Example: Update w , c_j , b by Gradient Descent Method

Using RBF neural network to approximate the following discrete plant

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

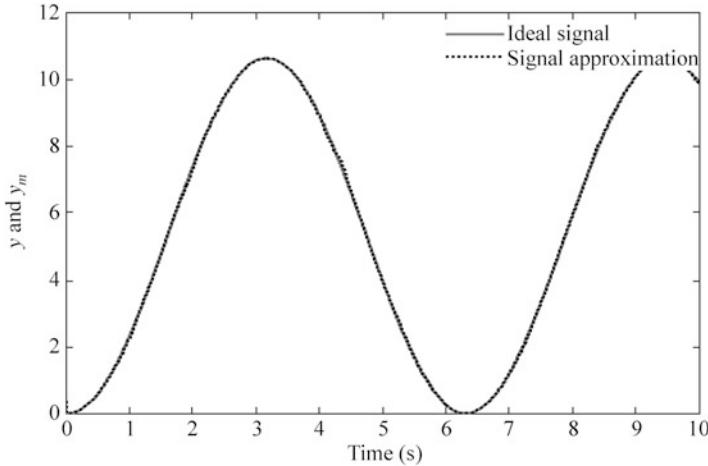


Fig. 2.8 RBF neural network approximation

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, and the initial parameters of Gaussian function are chosen as

$$\mathbf{c}_j = \begin{bmatrix} -1 & -0.5 & 0 & 0.5 & 1 \end{bmatrix}^T, \quad \mathbf{b}_j = 3.0, j = 1, 2, 3, 4, 5.$$

Choose the input as $u(k) = \sin t$, $t = k \times T$, $T = 0.001$: in simulation, $M = 1$ indicates only update \mathbf{w} with fixed \mathbf{c}_j and \mathbf{b} and $M = 2$ indicates update \mathbf{w} , \mathbf{c}_j , \mathbf{b} in RBF neural network approximation; the initial value of the input is set as $[0,1]$, and the results are shown from Figs. 2.9 and 2.10.

From the simulation test, we can see that better results can be gotten by the gradient descent method, especially the initial parameters of Gaussian function \mathbf{c}_j and \mathbf{b} are chosen not suitably.

The program of this example is chap2_4.m, which is given in the [Appendix](#).

2.3 Effect of Gaussian Function Parameters on RBF Approximation

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.

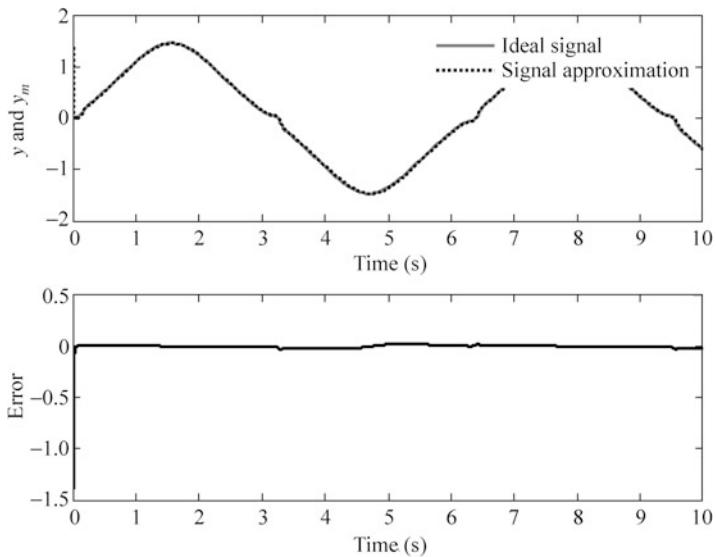


Fig. 2.9 RBF neural network approximation by only updating $w(M = 1)$

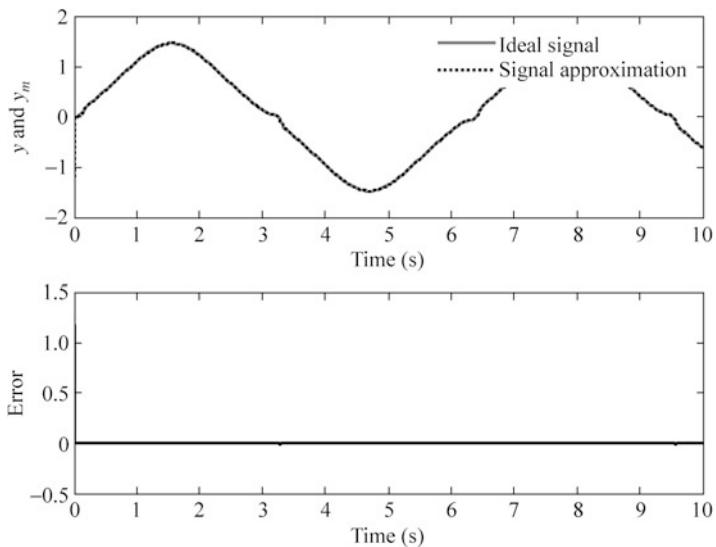


Fig. 2.10 RBF neural network approximation by updating $w, b, c(M = 2)$

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.

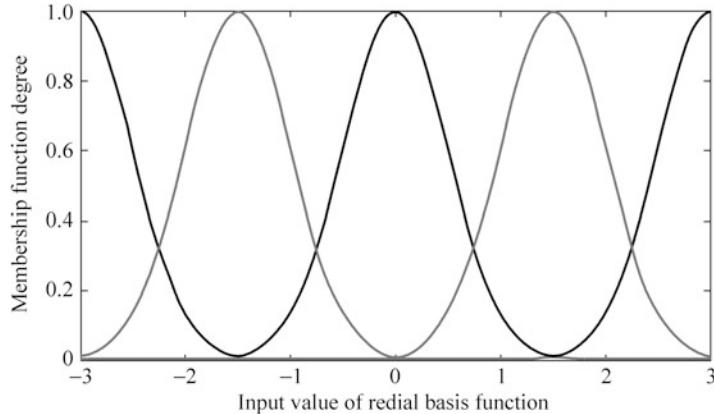


Fig. 2.11 Five Gaussian membership function

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 $[-3, +3]$, 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; thus, input value can be within the effective mapping of Gaussian membership function. Five Gaussian membership functions are shown in Fig. 2.11.

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 value, 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:

1. RBF approximation with moderate b_j and c_j ($M_b = 1, M_c = 1$)
2. RBF approximation with improper b_j and moderate c_j ($M_b = 2, M_c = 1$)
3. RBF approximation with moderate b_j and improper c_j ($M_b = 1, M_c = 2$)
4. RBF approximation with improper b_j and c_j ($M_b = 2, M_c = 2$)

The results are shown from Figs. 2.12, 2.13, 2.14, and 2.15. From the results, we can see if we design improper c_j and b_j , the RBF approximation performance will not be ensured.

The program of this example is chap2_5.m and chap2_6.m, which are given in the [Appendix](#).

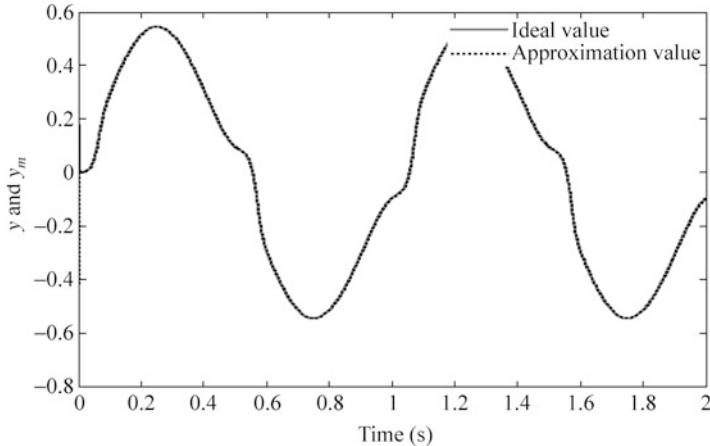


Fig. 2.12 RBF approximation with moderate b_j and c_j ($M_b = 1$, $M_c = 1$)

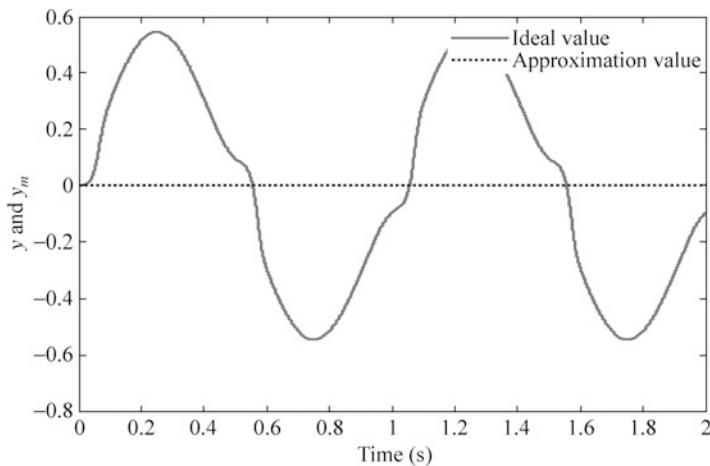


Fig. 2.13 RBF approximation with improper b_j and moderate c_j ($M_b = 2$, $M_c = 1$)

2.4 Effect 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 $b_j = 1.5$. The inputs of RBF are $u(k) = \sin t$ and $y(k)$. Set the structure as 2-m-1: m represents the

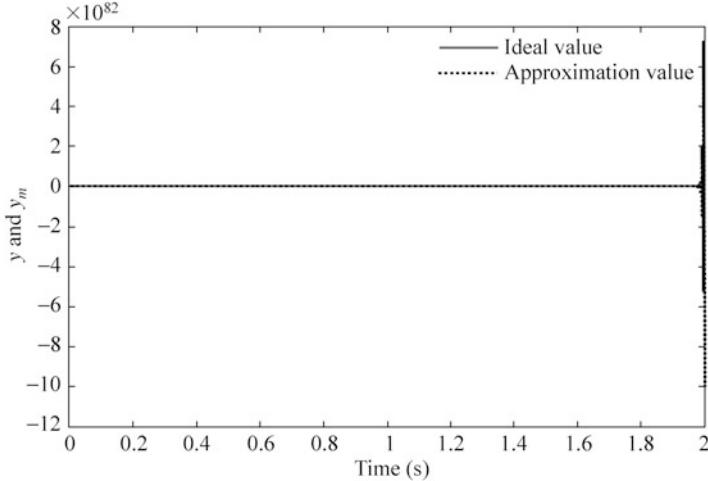


Fig. 2.14 RBF approximation with moderate b_j and improper c_j ($M_b = 1$, $M_c = 2$)

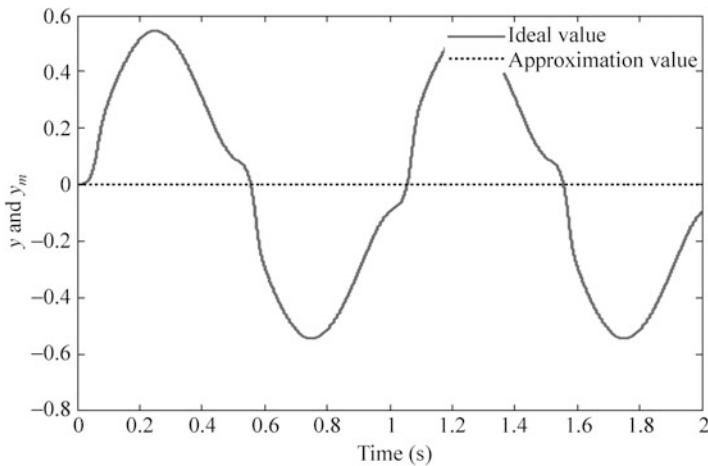


Fig. 2.15 RBF approximation with improper b_j and c_j ($M_b = 2$, $M_c = 2$)

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 $c_j = 0$, $c_j = \frac{1}{3}[-1 \ 0 \ 1]^T$ and $c_j = \frac{1}{9}\begin{bmatrix} -3 & -2 & -1 & 0 & 1 & 2 & 3 \end{bmatrix}^T$, respectively.

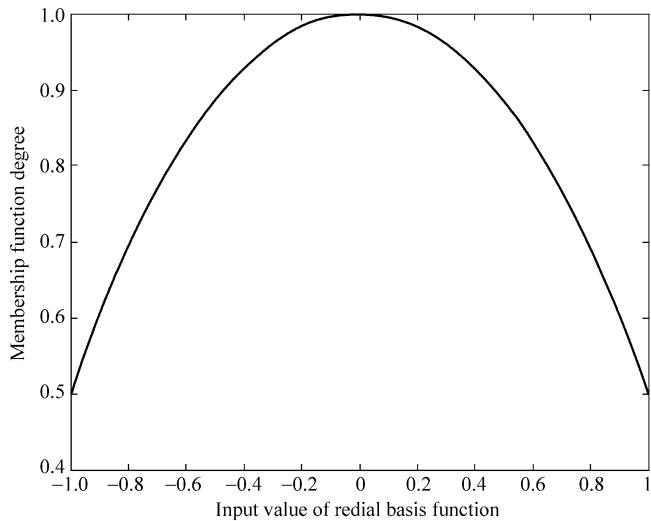


Fig. 2.16 One Gaussian function with only one hidden net ($m = 1$)

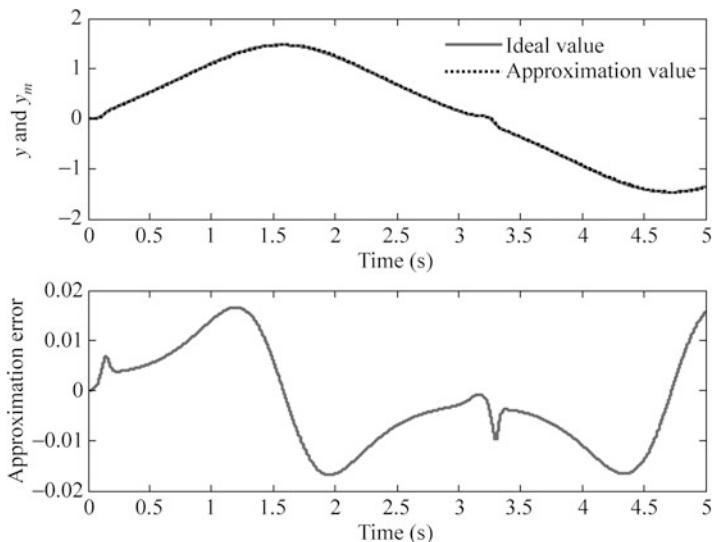


Fig. 2.17 Approximation with only one hidden net ($m = 1$)

The results are shown from Figs. 2.16, 2.17, 2.18, 2.19, 2.20, and 2.21. 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.

The program of this example is chap2_7.m, which is given in the [Appendix](#).

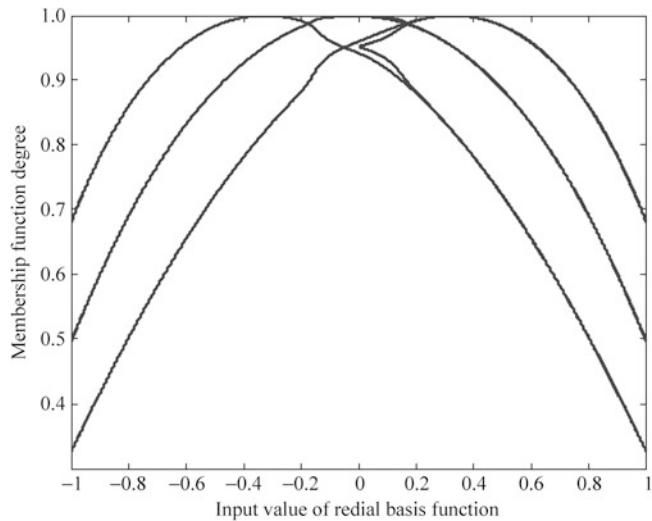


Fig. 2.18 Three Gaussian functions with three hidden nets ($m = 3$)

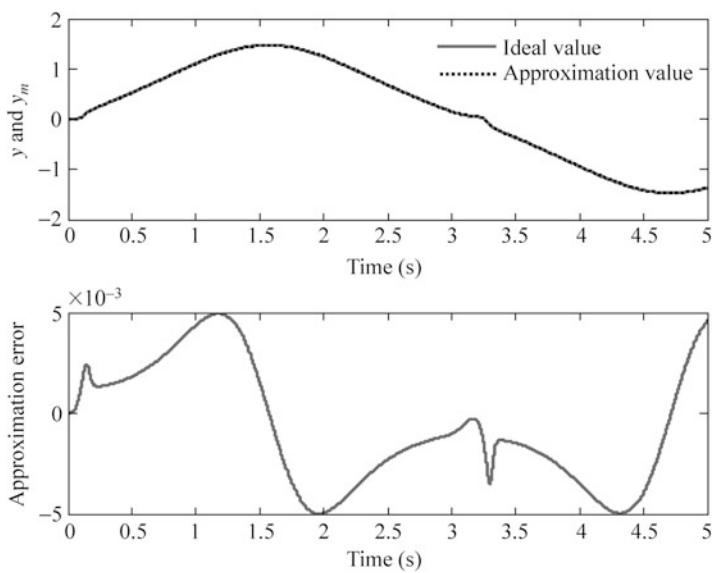


Fig. 2.19 Approximation with three hidden nets ($m = 3$)

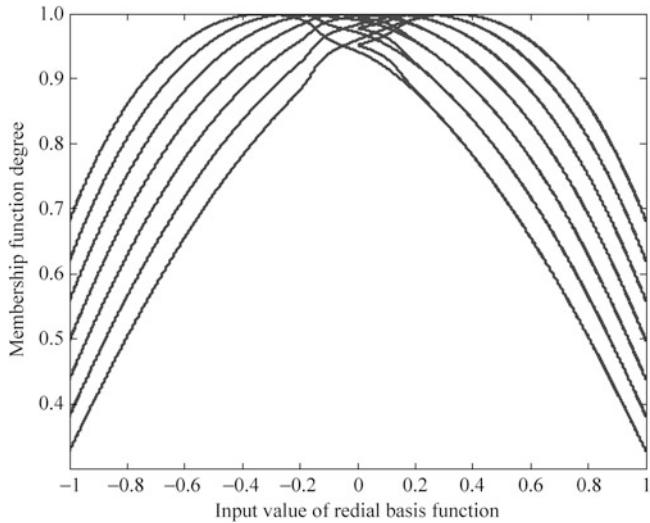


Fig. 2.20 Seven Gaussian functions with seven hidden nets ($m = 7$)

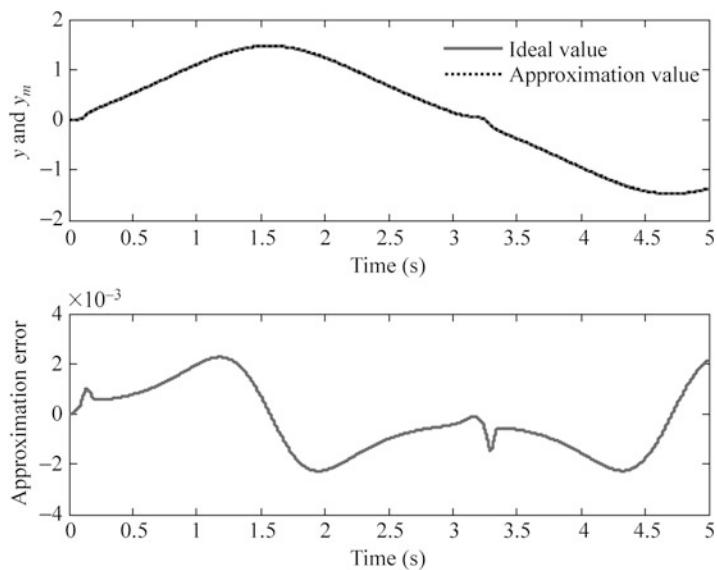


Fig. 2.21 Approximation with seven hidden nets ($m = 7$)

2.5 RBF Neural Network Training for System Modeling

2.5.1 RBF Neural Network Training

We use RBF neural network to train a data vector with multi-input and multi-output or to model a system off-line.

In RBF neural network, $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]^T$ is the input vector, and h_j is Gaussian function for neural net j , then

$$h_j = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_j\|^2}{2b_j^2}\right), \quad j = 1, 2, \dots, m \quad (2.13)$$

where $\mathbf{c}_j = [c_{j1}, \dots, c_{jn}]$ is the center vector of neural net j .

The width vector of Gaussian function is

$$\mathbf{b} = [b_1, \dots, b_m]^T$$

where $b_j > 0$ represents the width value of Gaussian function for neural net j .

The weight value is

$$\mathbf{w} = [w_1, \dots, w_m]^T \quad (2.14)$$

The output of RBF is

$$y_l = w_1 h_1 + w_2 h_2 + \dots + w_m h_m \quad (2.15)$$

where y_l^d denotes the ideal output, $l = 1, 2, \dots, N$.

The error of the l th output is

$$e_l = y_l^d - y_l.$$

The performance index function of the training is

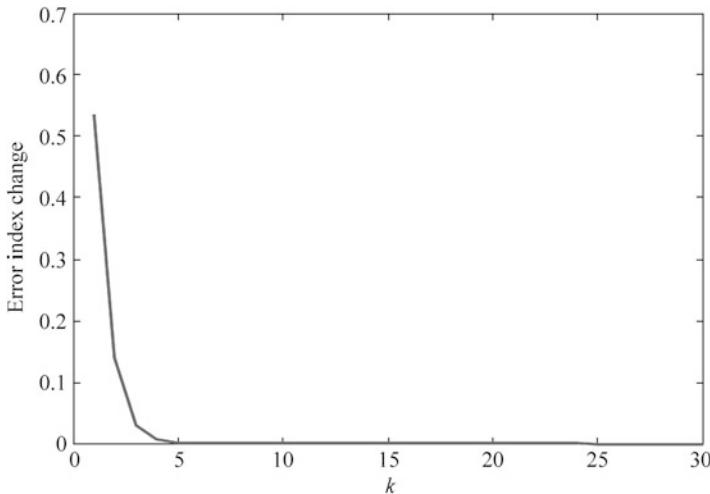
$$E(t) = \sum_{l=1}^N e_l^2. \quad (2.16)$$

According to gradient descent method, the parameters can be updated as follows:

$$\Delta w_j(t) = -\eta \frac{\partial E}{\partial w_j} = \eta \sum_{l=1}^N e_l h_j$$

Table 2.1 One training sample

Input	Output	
1	0	0

**Fig. 2.22** Error index change

$$w_j(t) = w_j(t-1) + \eta w_j(t) + \alpha(w_j(t-1) - w_j(t-2)) \quad (2.17)$$

where $\eta \in (0, 1)$ is the learning rate and $\alpha \in (0, 1)$ is momentum factor.

2.5.2 *Simulation Example*

2.5.2.1 First Example: A MIMO Data Sample Training

Choosing three inputs and two outputs data as a training sample, which are shown in Table 2.1.

RBF network structure is chosen as 3-5-1. The choice of 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 c_i and b_i are

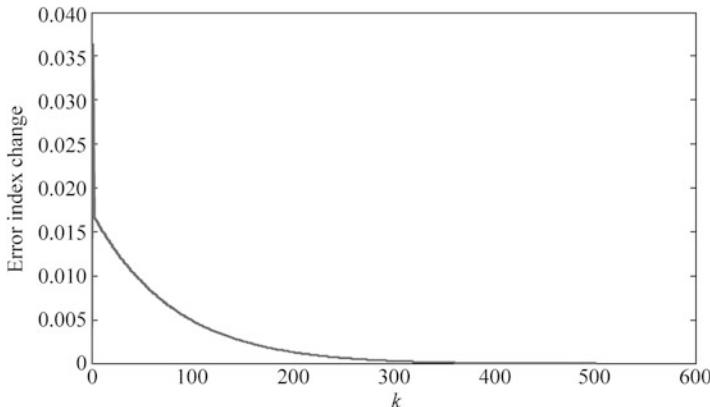
designed as $\begin{bmatrix} -1 & -0.5 & 0 & 0.5 & 1 \\ -1 & -0.5 & 0 & 0.5 & 1 \\ -1 & -0.5 & 0 & 0.5 & 1 \end{bmatrix}$ and 10, the initial weight value is chosen as

random value in the interval of $[-1, 1]$, and $\eta = 0.10$ and $\alpha = 0.05$ are chosen.

Firstly, we run chap2_8a.m, set the error index as $E = 10^{-20}$. Error index change is shown as Fig. 2.22, and the trained weight values are saved as wfile.dat.

Table 2.2 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

**Fig. 2.23** Error index change

Then we run chap2_8b.m, use wfile.dat, the test results with two samples are shown in Table 2.2. From the results, we can see that good modeling performance can be received.

The programs of this example are chap2_8a.m and chap2_8b.m, which are given in the [Appendix](#).

2.5.2.2 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; the parameters of 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; $\eta = 0.50$ and $\alpha = 0.05$ are chosen.

Firstly, we run chap2_9a.m, the input is chosen as $x = [u(k) \ y(k)]$, $u(k) = \sin t$, and $t = k \times ts$, $ts = 0.001$ represents sampling time. The number of samples is chosen as $NS = 3,000$. After 500 steps training off-line, we get the error index change as Fig. 2.23. The trained weight values and Gaussian function parameters are saved as wfile.dat.

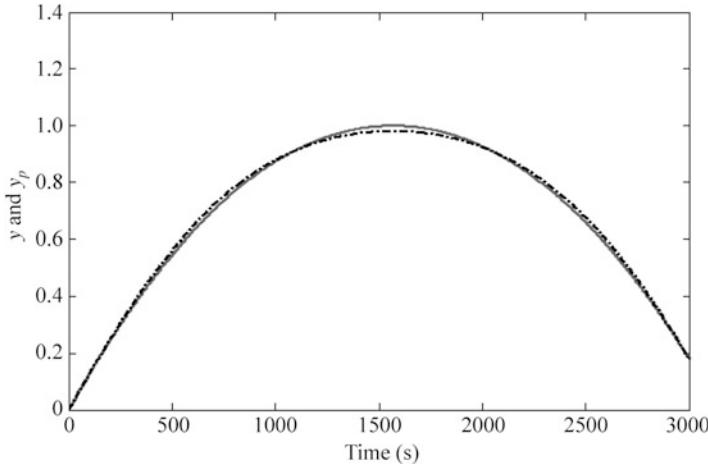


Fig. 2.24 Modeling test

Then we run chap2_9b.m, use wfile.dat, the test results with input $\sin t$ are shown in Fig. 2.24. From the results, we can see that good modeling performance can be received.

The programs of this example are chap2_9a.m and chap2_9b.m, which are given in the Appendix.

2.6 RBF Neural Network Approximation

Since any nonlinear function over a compact set with arbitrary accuracy can be approximated by RBF neural network [1, 2], RBF neural network can be used to approximate uncertainties in the control systems.

For example, to approximate the function $f(\mathbf{x})$, the algorithm of RBF is expressed as

$$h_j = g\left(\|\mathbf{x} - \mathbf{c}_{ij}\|^2 / b_j^2\right)$$

$$f = \mathbf{W}^{*\top} \mathbf{h}(\mathbf{x}) + \varepsilon$$

where \mathbf{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, $\mathbf{h} = [h_1, h_2, \dots, h_n]^T$ denotes the output of hidden layer, \mathbf{W}^* is ideal weight vector, and ε is approximation error, $\varepsilon \leq \varepsilon_N$.

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, that is, $\mathbf{x} = [e \quad \dot{e}]^T$, then the output of RBF is

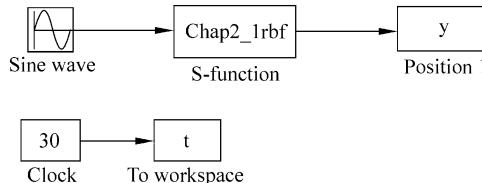
$$\hat{f}(x) = \hat{\mathbf{W}}^T \mathbf{h}(x) \quad (2.18)$$

where $\hat{\mathbf{W}}$ is the estimated weight vector, which can be tuned by the adaptive algorithm in the Lyapunov stability analysis.

Appendix

Programs for Sect. 2.1.2.1

Simulink main program: chap2_1sim.mdl



RBF function: chap2_1rbf.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);

```

Plot program: chap2_1plot.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;

```

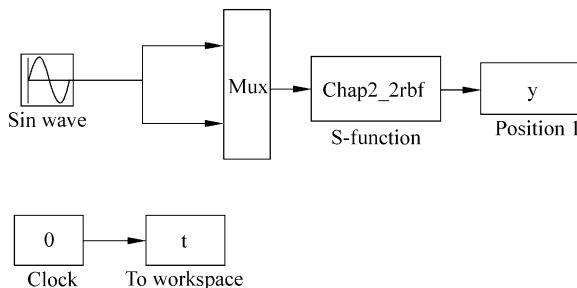
```

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);

```

Programs for Sect. 2.1.2.2

Simulink main program: chap2_2sim.mdl



RBF function: chap2_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 = 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);

```

Plot program: chap2_2plot.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);

```

```

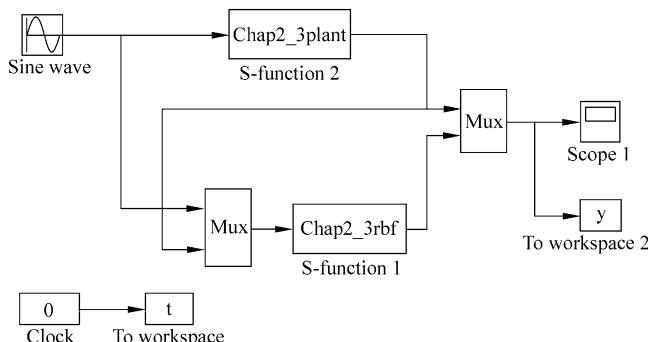
xlabel('x1'); ylabel('hj');
hold on;
plot(y(:,2),y(:,5),'k','LineWidth',2);
hold on;
plot(y(:,2),y(:,6),'k','LineWidth',2);
hold on;
plot(y(:,2),y(:,7),'k','LineWidth',2);
hold on;
plot(y(:,2),y(:,8),'k','LineWidth',2);

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);

```

Programs for Sect. 2.2.2.1

Simulink main program: chap2_3sim.mdl



S function for plant: chap2_3rbf.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;

```

S function for plant: chap2_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 = 1;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0,0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
sys(1)=x(2);
sys(2)=-25*x(2)+133*u;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);

```

Plot program: chap2_3plot.m

```

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');

```

Programs for Sect. 2.2.2.2

Matlab program: chap2_4.m

```

%RBF identification
clear all;
close all;

```

```

alfa=0.05;
xite=0.15;
x=[0,1]';

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=2;
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;
    end
    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
    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');
```

Programs for Sect. 2.3

Program of Gaussian membership function design: chap2_5.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');

```

Program of RBF approximation to test the effect of b and c: chap2_6.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 width 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');

```

Programs for Sect. 2.4

Program of RBF approximation to test the effect of hidden nets number: chap2_7.m

```

%RBF approximation test
clear all;
close all;

alfa=0.05;
xite=0.3;
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

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');
```

Programs for Sect. 2.5.2.1

Program of RBF training: chap2_8a.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
y1=W'*h; %Output Layer

el=0;
y=ys(s,:);
for l=1:1:OUT
    el=el+0.5*(y(l)-y1(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-y1';
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;

```

Program of RBF test: chap2_8b.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
y1(i,:)=W'*h; %Output Layer
end
y1

```

Programs for Sect. 2.5.2.2

Program of RBF training: chap2_9a.m

```

%RBF Training for a Plant
clear all;
close all;
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
y1(s)=W*h; %Output Layer
el=0.5*(y(s)-y1(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)-y1(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;

```

Program of RBF test: chap2_9b.m

```

%Online RBF Estimation 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');

```

References

1. Hartman EJ, Keeler JD, Kowalski JM (1990) Layered neural networks with Gaussian hidden units as universal approximations. *Neural Comput* 2(2):210–215
2. Park J, Sandberg LW (1991) Universal approximation using radial-basis-function networks. *Neural Comput* 3(2):246–257

Chapter 3

RBF Neural Network Control Based on Gradient Descent Algorithm

Abstract This chapter introduces three kinds of RBF neural network control laws based on gradient descent rule, including supervisory control law, model reference adaptive control law, and self-adjust control law; the weight value learning algorithms are presented. Several simulation examples are given.

Keywords Neural network control • Gradient descent rule • Supervisory control • Model reference adaptive control • Self-adjust control

Gradient descent rule is often used in the weight value optimization in neural network discrete control; two typical examples are given in [1, 2].

3.1 Supervisory Control Based on RBF Neural Network

3.1.1 RBF Supervisory Control

The structure of RBF supervisory control is shown in Fig. 3.1.

The radial basis vector is $\mathbf{h} = [h_1, \dots, h_m]^T$, h_j is Gaussian function:

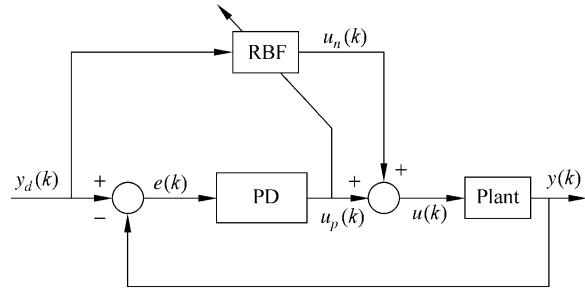
$$h_j = \exp\left(-\frac{\|\mathbf{x}(k) - \mathbf{c}_j\|^2}{2b_j^2}\right) \quad (3.1)$$

where $i = 1; j = 1, \dots, m$; $\mathbf{x}(k)$ is the input of RBF; $\mathbf{c}_j = [c_{11}, \dots, c_{1m}]$, and $\mathbf{b} = [b_1, \dots, b_m]^T$.

The weight vector is

$$\mathbf{w} = [w_1, \dots, w_m]^T. \quad (3.2)$$

Fig. 3.1 RBF supervisory control system



The output of RBF is

$$u_n(k) = h_1 w_1 + \dots + h_j w_j + \dots + h_m w_m \quad (3.3)$$

where m is the number of hidden layer.

The criterion on function used here is $u_p(k)$ as follows:

$$E(k) = \frac{1}{2} (u_n(k) - u(k))^2. \quad (3.4)$$

According to the steepest descent (gradient) method, the learning algorithm is as follows:

$$\begin{aligned} \Delta w_j(k) &= -\eta \frac{\partial E(k)}{\partial w_j(k)} = \eta(u_n(k) - u(k))h_j(k) \\ \mathbf{w}(k) &= \mathbf{w}(k-1) + \Delta \mathbf{w}(k) + \alpha(\mathbf{w}(k-1) - \mathbf{w}(k-2)) \end{aligned} \quad (3.5)$$

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

3.1.2 Simulation Example

The plant is

$$G(s) = \frac{1,000}{s^3 + 87.35s^2 + 10,470s}.$$

Discrete plant with sampling time 1 ms is

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

The structure is 1-4-1, $x(k) = y_d(k)$, the initial value of weight vector is chosen as random value in $[0, 1]$, and the initial Gaussian parameters are

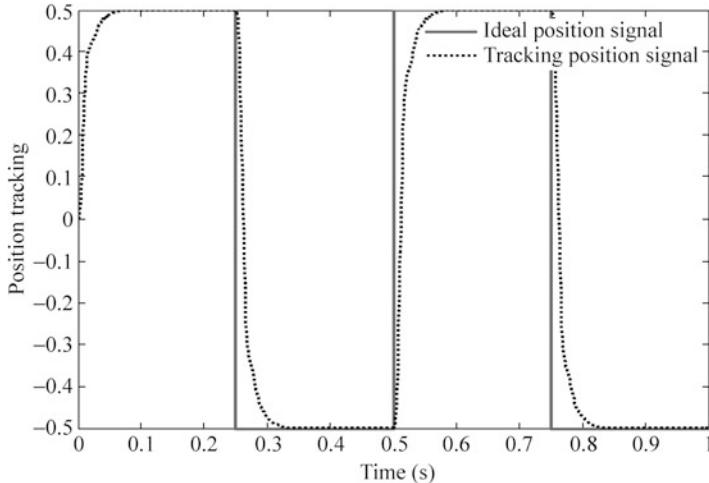


Fig. 3.2 Square position tracking

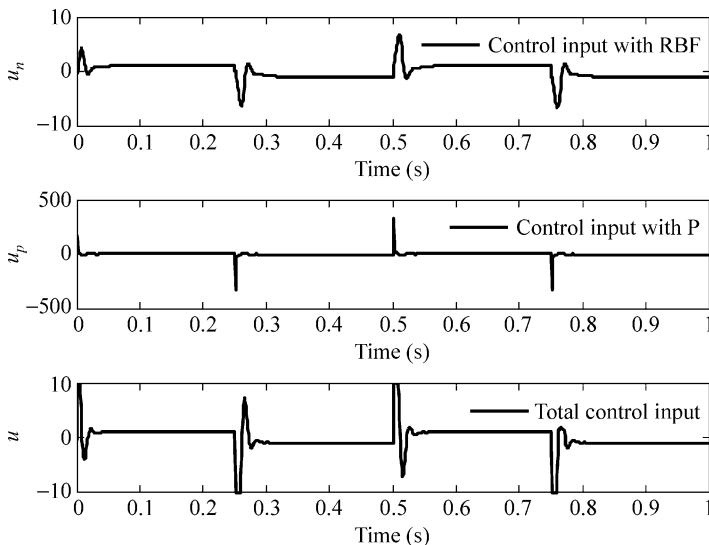


Fig. 3.3 Control input of NN, PD, and overall

$\mathbf{c} = [-5 \quad -3 \quad 0 \quad 3 \quad 5]^T$, $\mathbf{b} = [0.5 \quad 0.5 \quad 0.5 \quad 0.5]^T$. Choosing $\eta = 0.30$, $\alpha = 0.05$.

The results are shown in Figs. 3.2 and 3.3. The program of RBF supervisory control is [chap3_1.m](#).

3.2 RBFNN-Based Model Reference Adaptive Control

3.2.1 Controller Design

The control system is shown in Fig. 3.4.

The reference model is $y_m(k)$; then the tracking error is

$$e(k) = y_m(k) - y(k). \quad (3.6)$$

The criterion on function used here is $u_p(k)$ as follows:

$$E(k) = \frac{1}{2}ec(k)^2. \quad (3.7)$$

The controller is the output of RBF:

$$u(k) = h_1 w_1 + \cdots h_j w_j \cdots + h_m w_m \quad (3.8)$$

where m is the number of neural nets in hidden layer, w_j is weight value of net j , h_j is output of Gaussian function.

In RBF, $\mathbf{x} = [x_1, \dots, x_n]^T$ is input vector, $\mathbf{h} = [h_1, \dots, h_m]^T$, and h_j is Gaussian function:

$$h_j = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_j\|^2}{2b_j^2}\right) \quad (3.9)$$

where $i = 1, \dots, n$ and $j = 1, \dots, m$. $b_j > 0$, $\mathbf{c}_j = [c_{j1}, \dots, c_{ji}, \dots, c_{jn}]$, and $\mathbf{b} = [b_1, \dots, b_m]^T$.

The weight vector is

$$\mathbf{w} = [w_1, \dots, w_m]^T. \quad (3.10)$$

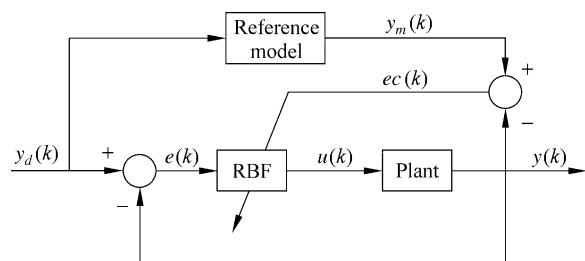


Fig. 3.4 RBF-based model reference adaptive control system

According to the steepest descent (gradient) method, the learning algorithm is as follows:

$$\begin{aligned}\Delta w_j(k) &= -\eta \frac{\partial E(k)}{\partial w} = \eta e c(k) \frac{\partial y(k)}{\partial u(k)} h_j \\ w_j(k) &= w_j(k-1) + \Delta w_j(k) + \alpha \Delta w_j(k)\end{aligned}\quad (3.11)$$

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

For the same reason, we can get

$$\Delta b_j(k) = -\eta \frac{\partial E(k)}{\partial b_j} = \eta e c(k) \frac{\partial y(k)}{\partial u(k)} \frac{\partial u(k)}{\partial b_j} = \eta e c(k) \frac{\partial y(k)}{\partial u(k)} w_j h_j \frac{\|x - c_{ij}\|^2}{b_j^3} \quad (3.12)$$

$$b_j(k) = b_j(k-1) + \eta \Delta b_j(k) + \alpha(b_j(k-1) - b_j(k-2)) \quad (3.13)$$

$$\Delta c_{ij}(k) = -\eta \frac{\partial E(k)}{\partial c_{ij}} = \eta e c(k) \frac{\partial y(k)}{\partial u(k)} \frac{\partial u(k)}{\partial c_{ij}} = \eta e c(k) \frac{\partial y(k)}{\partial u(k)} w_j h_j \frac{x_i - c_{ij}}{b_j^2} \quad (3.14)$$

$$c_{ij}(k) = c_{ij}(k-1) + \eta \Delta c_{ij}(k) + \alpha(c_{ij}(k-1) - c_{ij}(k-2)) \quad (3.15)$$

$\frac{\partial y(k)}{\partial u(k)}$ is *Jacobian* value, which expresses the sensitivity of output and input of the system. *Jacobian* value can be gotten by the sign of $\frac{\partial y(k)}{\partial u(k)}$.

3.2.2 Simulation Example

The plant is

$$y(k) = (-0.10y(k-1) + u(k-1)) / (1 + y(k-1)^2)$$

The sampling time is $ts = 1$ ms; the reference model is $y_m(k) = 0.6y_m(k-1) + y_d(k)$, $y_d(k) = 0.50 \sin(2\pi k \times ts)$.

Choose $y_d(k)$, $e(k)$, and $u(k)$ as input of RBF, and set $\eta = 0.35$, $\alpha = 0.05$.

The initial value of Gaussian function is

$$\mathbf{c} = \begin{bmatrix} -3 & -2 & -1 & 1 & 2 & 3 \\ -3 & -2 & -1 & 1 & 2 & 3 \\ -3 & -2 & -1 & 1 & 2 & 3 \end{bmatrix}^T,$$

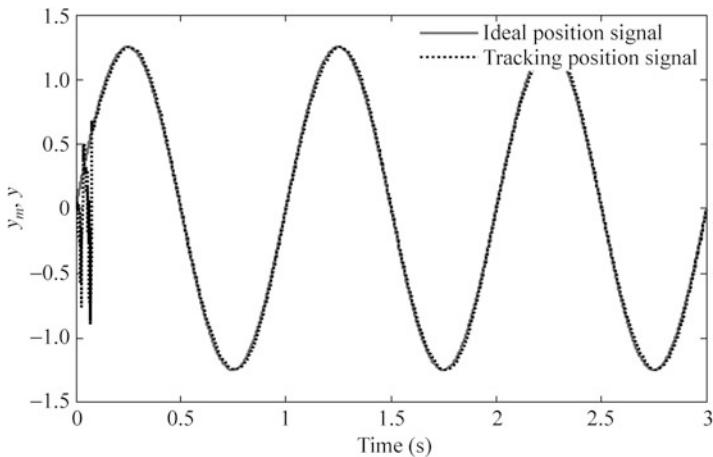


Fig. 3.5 Sine position tracking

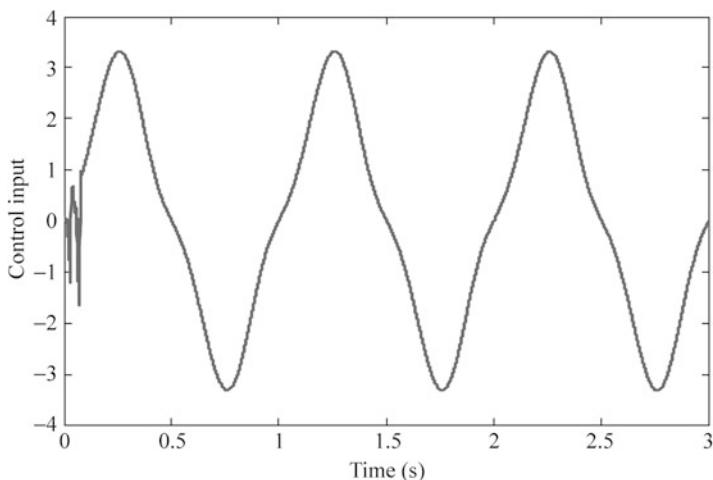


Fig. 3.6 Sine position tracking

and $\mathbf{b} = [2, 2, 2, 2, 2, 2]^T$, the initial weight value is $\mathbf{w} = [-0.0316 \quad -0.0421 \quad -0.0318 \quad 0.0068 \quad 0.0454 \quad -0.0381]$.

The simulation results are shown in Figs. 3.5 and 3.6. The program of model reference adaptive control is chap3_2.m.

3.3 RBF Self-Adjust Control

3.3.1 System Description

Considering plant,

$$y(k+1) = g[y(k)] + \varphi[y(k)]u(k) \quad (3.16)$$

where $y(k)$ is output and $u(k)$ is control input.

Assuming $y_d(k)$ is ideal position signal. If $g[\cdot]$ and $\varphi[\cdot]$ are known, then the self-adjust controller can be designed as

$$u(k) = \frac{-g[\cdot]}{\varphi[\cdot]} + \frac{y_d(k+1)}{\varphi[\cdot]}. \quad (3.17)$$

However, the items $g[\cdot]$ and $\varphi[\cdot]$ are often unknown, so it is difficult to realize the control law (3.17).

3.3.2 RBF Controller Design

If $g[\cdot]$ and $\varphi[\cdot]$ are unknown, we can use two RBF to identify $g[\cdot]$ and $\varphi[\cdot]$, and we can get the estimated value of $g[\cdot]$ and $\varphi[\cdot]$, that is, $Ng[\cdot]$, $N\varphi[\cdot]$. Then the self-adjust controller can be designed as

$$u(k) = \frac{-Ng[\cdot]}{N\varphi[\cdot]} + \frac{y_d(k+1)}{N\varphi[\cdot]} \quad (3.18)$$

where $Ng[\cdot]$ and $N\varphi[\cdot]$ are output of RBF NN identifier.

Using two RBF to approximate $g[\cdot]$ and $\varphi[\cdot]$, respectively, \mathbf{W} and \mathbf{V} are weight vector of RBFNN, respectively.

In RBF neural network, choosing $y(k)$ as input $\mathbf{h} = [h_1 \ \cdots \ h_m]^T$, h_j is Gaussian function:

$$h_j = \exp\left(-\frac{\|y(k) - \mathbf{c}_j\|^2}{2b_j^2}\right) \quad (3.19)$$

where $i = 1; j = 1, \dots, m; b_j > 0; \mathbf{c}_j = [c_{11}, \dots, c_{1m}]$; and $\mathbf{b} = [b_1, \dots, b_m]^T$.

The weight vector of RBF is expressed as

$$\mathbf{W} = [w_1, \dots, w_m]^T \quad (3.20)$$

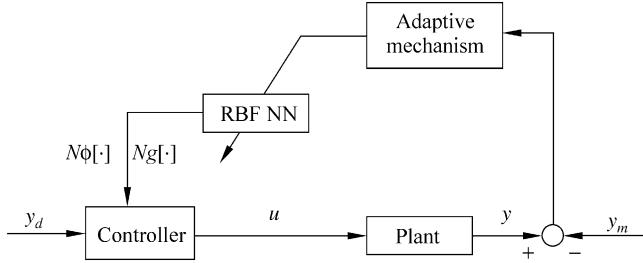


Fig. 3.7 The closed-loop neural-based adaptive control scheme

$$\mathbf{V} = [v_1, \dots, v_m]^T. \quad (3.21)$$

The outputs of two RBF are

$$Ng(k) = h_1 w_1 + \dots + h_j w_j + \dots + h_m w_m \quad (3.22)$$

$$N\varphi(k) = h_1 v_1 + \dots + h_j v_j + \dots + h_m v_m \quad (3.23)$$

where m is the number of hidden layer net.

Using the output of two RBF, the plant can be written as

$$y_m(k) = Ng[y(k-1); W(k)] + N\varphi[y(k-1); V(k)]u(k-1). \quad (3.24)$$

The closed-loop neural-based adaptive control scheme with two RBF neural networks to identify $Ng[\cdot]$ and $N\varphi[\cdot]$ is shown in Fig. 3.7.

The criterion on function used here is an error square function as follows:

$$E(k) = \frac{1}{2}(y(k) - y_m(k))^2. \quad (3.25)$$

According to the steepest descent (gradient) method, the learning algorithm is as follows:

$$\Delta w_j(k) = -\eta_w \frac{\partial E(k)}{\partial w_j(k)} = \eta_w(y(k) - y_m(k))h_j(k)$$

$$\Delta v_j(k) = -\eta_v \frac{\partial E(k)}{\partial v_j(k)} = \eta_v(y(k) - y_m(k))h_j(k)u(k-1)$$

$$W(k) = W(k-1) + \Delta W(k) + \alpha(W(k-1) - W(k-2)) \quad (3.26)$$

$$V(k) = V(k-1) + \Delta V(k) + \alpha(V(k-1) - V(k-2)) \quad (3.27)$$

where η_w and η_v are learning rates and α is a momentum factor.

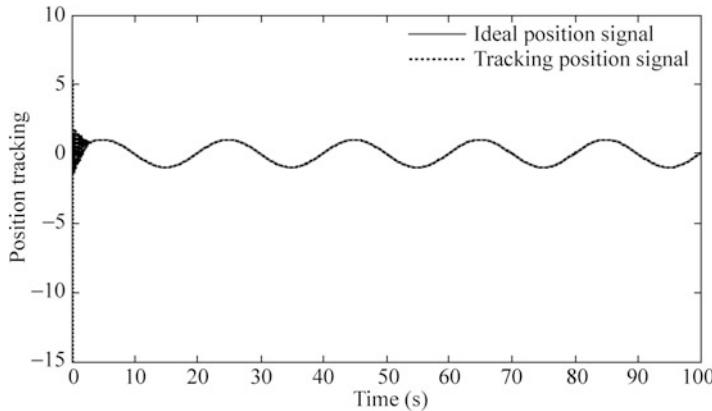


Fig. 3.8 Sine position tracking

3.3.3 Simulation Example

The plant is

$$y(k) = 0.8 \sin(y(k-1)) + 15u(k-1)$$

where $g[y(k)] = 0.8 \sin(y(k-1))$ and $\varphi[y(k)] = 15$.

The ideal signal is $y_d(t) = 2.0 \sin(0.1\pi t)$. RBF neural network structure is 1-6-1. The initial weight value and Gaussian parameters are $W = [0.5, 0.5, 0.5, 0.5, 0.5, 0.5]^T$, $V = [0.5, 0.5, 0.5, 0.5, 0.5, 0.5]^T$, $c_j = [0.5, 0.5, 0.5, 0.5, 0.5, 0.5]^T$, $b = [5, 5, 5, 5, 5, 5]^T$.

Let $\eta_1 = 0.15$ and $\eta_2 = 0.50$, $\alpha = 0.05$. The simulation results are shown from Figs. 3.8, 3.9 and 3.10. The program of RBF self-adjust control is chap3_3.m.

Appendix

Programs for Sect. 3.1.2

The program of RBF supervisory control: chap3_1.m

```
%RBF Supervisory Control
clear all;
close all;

ts=0.001;
sys=tf(1000,[1,50,2000]);
dys=c2d(sys,ts,'z');
[num,den]=tfdata(dys,'v');
```

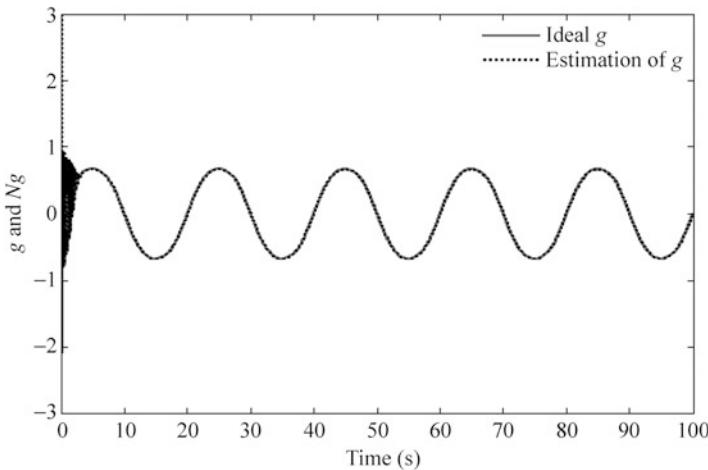


Fig. 3.9 $g(x, t)$ and its estimated $\hat{g}(x, t)$

```

y_1=0;y_2=0;
u_1=0;u_2=0;
e_1=0;

xi=0;
x=[0,0]';

b=0.5*ones(4,1);
c=[-2 -1 1 2];
w=rands(4,1);
w_1=w;
w_2=w_1;

xite=0.30;
alfa=0.05;

kp=25;
kd=0.3;
for k=1:1:1000
    time(k)=k*ts;

S=1;
if S==1
    yd(k)=0.5*sign(sin(2*2*pi*k*ts)); %Square Signal
elseif S==2
    yd(k)=0.5*(sin(3*2*pi*k*ts)); %Square Signal
end

y(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
e(k)=yd(k)-y(k);

xi=yd(k);

```

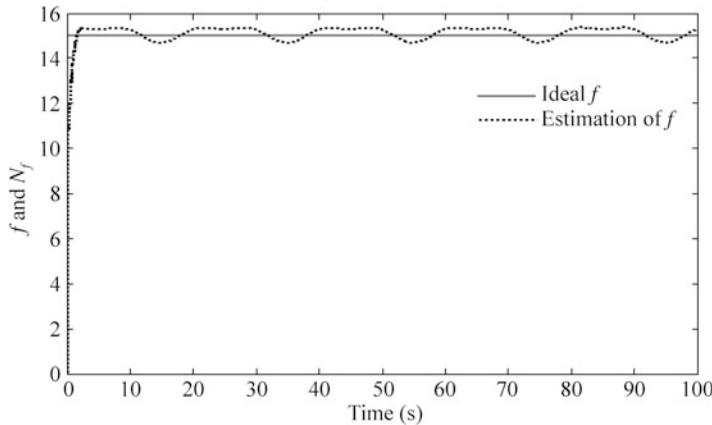


Fig. 3.10 $f(x, t)$ and its estimated $\hat{f}(x, t)$

```

for j=1:1:4
    h(j)=exp(-norm(xi-c(:,j))^2/(2*b(j)*b(j)));
end
un(k)=w'*h';
%PD Controller
up(k)=kp*x(1)+kd*x(2);
M=2;
if M==1      %Only Using PID Control
    u(k)=up(k);
elseif M==2 %Total control output
    u(k)=up(k)+un(k);
end
if u(k)>=10
    u(k)=10;
end
if u(k)<=-10
    u(k)=-10;
end
%Update NN Weight
d_w=-xite*(un(k)-u(k))*h';
w=w_1+d_w+alfa*(w_1-w_2);
w_2=w_1;
w_1=w;
u_2=u_1;
u_1=u(k);

```

```

y_2=y_1;
y_1=y(k);

x(1)=e(k); %Calculating P
x(2)=(e(k)-e_1)/ts; %Calculating D
e_1=e(k);
end
figure(1);
plot(time,yd,'r',time,y,'k:','linewidth',2);
xlabel('Time(second)');ylabel('Position tracking');
legend('Ideal position signal','Tracking position signal');

figure(2);
subplot(311);
plot(time,un,'k','linewidth',2);
xlabel('time(s)');ylabel('un');
legend('Control input with RBF');
subplot(312);
plot(time,up,'k','linewidth',2);
xlabel('time(s)');ylabel('up');
legend('Control input with P');
subplot(313);
plot(time,u,'k','linewidth',2);
xlabel('time(s)');ylabel('u');
legend('Total control input');

```

Programs for Sect. 3.2.2

Programs: chap3_2.m

```

%Model Reference Adaptive RBF Control
clear all;
close all;

u_1=0;
y_1=0;
ym_1=0;

x=[0,0,0]';
c=[-3 -2 -1 0 1 2 3;
   -3 -2 -1 0 1 2 3;
   -3 -2 -1 0 1 2 3];

b=2;
w=rands(1,7);

xite=0.35;

```

```
alfa=0.05;
h=[0,0,0,0,0,0,0]';

c_1=c;c_2=c;
b_1=b;b_2=b;
w_1=w;w_2=w;

ts=0.001;
for k=1:1:3000
time(k)=k*ts;

yd(k)=0.50*sin(2*pi*k*ts);
ym(k)=0.6*ym_1+yd(k);

y(k)=(-0.1*y_1+u_1)/(1+y_1^2); %Nonlinear plant

for j=1:1:7
    h(j)=exp(-norm(x-c(:,j))^2/(2*b^2));
end
u(k)=w*h;

ec(k)=ym(k)-y(k);
dyu(k)=sign((y(k)-y_1)/(u(k)-u_1));

d_w=0*w;
for j=1:1:7
    d_w(j)=xite*ec(k)*h(j)*dyu(k);
end
w=w_1+d_w+alfa*(w_1-w_2);
%Return of parameters
u_1=u(k);
y_1=y(k);
ym_1=ym(k);

x(1)=yd(k);
x(2)=ec(k);
x(3)=y(k);

w_2=w_1;w_1=w;
end
figure(1);
plot(time,ym,'r',time,y,'k:','linewidth',2);
xlabel('time(s)');ylabel('ym,y');
legend('Ideal position signal','Tracking position signal');
figure(2);
plot(time,u,'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');
```

Programs for Sect. 3.3.3

RBF NN Self-adjust Control:chap3_3.m

```
%Self-Correct control based RBF Identification
clear all;
close all;

xite1=0.15;
xite2=0.50;
alfa=0.05;
w=0.5*ones(6,1);
v=0.5*ones(6,1);
cij=0.50*ones(1,6);
bj=5*ones(6,1);
h=zeros(6,1);

w_1=w;w_2=w_1;
v_1=v;v_2=v_1;
u_1=0;y_1=0;

ts=0.02;
for k=1:1:5000
time(k)=k*ts;
yd(k)=1.0*sin(0.1*pi*k*ts);

%Practical Plant;
g(k)=0.8*sin(y_1);
f(k)=15;
y(k)=g(k)+f(k)*u_1;

for j=1:1:6
    h(j)=exp(-norm(y(k)-cij(:,j))^2/(2*bj(j)*bj(j)));
end

Ng(k)=w'*h;
Nf(k)=v'*h;

ym(k)=Ng(k)+Nf(k)*u_1;
e(k)=y(k)-ym(k);

d_w=0*w;
for j=1:1:6
    d_w(j)=xite1*e(k)*h(j);
end
w=w_1+d_w+alfa*(w_1-w_2);

d_v=0*v;
for j=1:1:6
    d_v(j)=xite2*e(k)*h(j)*u_1;
```

```
end
v=v_1+d_v+alfa*(v_1-v_2);
u(k)=-Ng(k)/Nf(k)+yd(k)/Nf(k);
u_1=u(k);
y_1=y(k);
w_2=w_1;
w_1=w;
v_2=v_1;
v_1=v;
end
figure(1);
plot(time,yd,'r',time,y,'k:','linewidth',2);
xlabel('Time(second)');ylabel('Position tracking');
legend('Ideal position signal','Tracking position signal');
figure(2);
plot(time,g,'r',time,Ng,'k:','linewidth',2);
xlabel('Time(second)');ylabel('g and Ng');
legend('Ideal g','Estimation of g');
figure(3);
plot(time,f,'r',time,Nf,'k:','linewidth',2);
xlabel('Time(second)');ylabel('f and Nf');
legend('Ideal f','Estimation of f');
```

References

1. Noriega JR, Wang H (1998) A direct adaptive neural network control for unknown nonlinear systems and its application. IEEE Trans Neural Netw 9(1):27–34
2. Suykens JAK, Vandewalle JPL, Demoor BLR (1996) Artificial neural networks for modeling and control of nonlinear systems. Kluwer Academic, Boston

Chapter 4

Adaptive RBF Neural Network Control

Abstract This chapter introduces several online adaptive RBF neural network control methods, including adaptive control based on neural approximation, adaptive control based on neural approximation with unknown parameter, and a direct robust adaptive control. For above control laws, the adaptive law is designed based on the Lyapunov stability theory, the closed-loop system stability can be achieved.

Keywords RBF neural network• Adaptive control• Neural approximation• Lyapunov stability

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 cannot 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.

4.1 Adaptive Control Based on Neural Approximation

4.1.1 Problem Description

Consider a second-order nonlinear system

$$\ddot{x} = f(x, \dot{x}) + g(x, \dot{x})u \quad (4.1)$$

where f is unknown nonlinear function, g is known nonlinear function, and $u \in R^n$ and $y \in R^n$ are input and output.

The Eq. (4.1) can also be written as

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x_1, x_2) + g(x_1, x_2)u \\ y &= x_1.\end{aligned}\tag{4.2}$$

We assume the ideal position signal is y_d . Let

$$e = y_d - y = y_d - x_1, \quad \mathbf{E} = (e \quad \dot{e})^T.$$

We design the control law as

$$u^* = \frac{1}{g(\mathbf{x})} [-f(\mathbf{x}) + \ddot{y}_d + \mathbf{K}^T \mathbf{E}] \tag{4.3}$$

substituting (4.3) into (4.1), we can get the closed control system as

$$\ddot{e} + k_p e + k_d \dot{e} = 0. \tag{4.4}$$

We design $\mathbf{K} = (k_p, k_d)^T$ 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 $t \rightarrow \infty$, $e(t) \rightarrow 0$ and $\dot{e}(t) \rightarrow 0$.

From (4.3), we know if the function $f(\mathbf{x})$ is unknown, the control law will not be realized.

4.1.2 Adaptive RBF Controller Design

4.1.2.1 RBF Neural Network Design

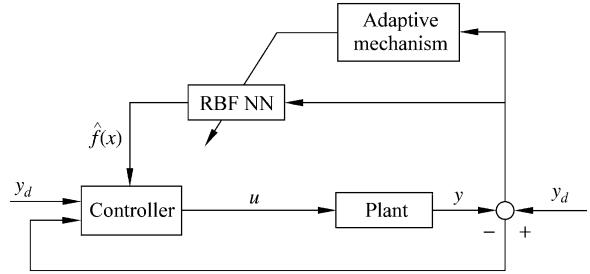
In this section, we use RBF to design $\hat{f}(\mathbf{x})$ to approximate $f(\mathbf{x})$. The algorithm of RBF is described as

$$h_j = g\left(\|\mathbf{x} - \mathbf{c}_{ij}\|^2 / b_j^2\right)$$

$$f = \mathbf{W}^T \mathbf{h}(\mathbf{x}) + \varepsilon$$

where \mathbf{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, $\mathbf{h} = [h_1, h_2, \dots, h_n]^T$ denotes the output of hidden layer, \mathbf{W} is weight value, ε is approximation error, and $\varepsilon \leq \varepsilon_N$.

Fig. 4.1 Block diagram of the control scheme



We use RBF to approximate f , the input vector is chosen as $\mathbf{x} = [e \quad \dot{e}]^T$, the output of RBF is

$$\hat{f}(\mathbf{x}) = \hat{\mathbf{W}}^T \mathbf{h}(\mathbf{x}). \quad (4.5)$$

4.1.2.2 Control Law and Adaptive Law Design

The fuzzy system approximation algorithm was applied to design indirect adaptive fuzzy controller [1]. Now we used RBF to replace fuzzy system to design RBF adaptive controller.

If we use RBF neural network to represent the unknown nonlinear function f , the control law becomes

$$u = \frac{1}{g(\mathbf{x})} [-\hat{f}(\mathbf{x}) + \ddot{y}_d + \mathbf{K}^T \mathbf{E}] \quad (4.6)$$

$$\hat{f}(\mathbf{x}) = \hat{\mathbf{W}}^T \mathbf{h}(\mathbf{x}) \quad (4.7)$$

where $\mathbf{h}(\mathbf{x})$ is Gaussian function and $\hat{\mathbf{W}}$ is the estimated parameter for \mathbf{W} .

Figure 4.1 shows the closed-loop neural-based adaptive control scheme.

We choose the adaptive law as

$$\dot{\hat{\mathbf{W}}} = -\gamma \mathbf{E}^T \mathbf{P} \mathbf{b} \mathbf{h}(\mathbf{x}) \quad (4.8)$$

4.1.2.3 Stability Analysis

Submitting the control law (4.6) into (4.1), the closed-loop system is expressed as

$$\ddot{e} = -\mathbf{K}^T \mathbf{E} + [\hat{f}(\mathbf{x}) - f(\mathbf{x})]. \quad (4.9)$$

Let

$$\boldsymbol{\Lambda} = \begin{bmatrix} 0 & 1 \\ -k_p & -k_d \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \quad (4.10)$$

Now, (4.9) can be rewritten as

$$\dot{\boldsymbol{E}} = \boldsymbol{\Lambda}\boldsymbol{E} + \boldsymbol{B}[\hat{f}(\boldsymbol{x}) - f(\boldsymbol{x})]. \quad (4.11)$$

The optimal weight value is

$$\boldsymbol{W}^* = \arg \min_{\boldsymbol{W} \in \Omega} [\sup |\hat{f}(\boldsymbol{x}) - f(\boldsymbol{x})|]. \quad (4.12)$$

Define the modeling error as

$$\omega = \hat{f}(\boldsymbol{x}|\boldsymbol{W}^*) - f(\boldsymbol{x}). \quad (4.13)$$

Then Eq. (4.11) becomes

$$\dot{\boldsymbol{E}} = \boldsymbol{\Lambda}\boldsymbol{E} + \boldsymbol{B}\{\hat{f}(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}|\boldsymbol{W}^*)\} + \omega. \quad (4.14)$$

Submitting (4.7) into (4.14), we can get closed equation as

$$\dot{\boldsymbol{E}} = \boldsymbol{\Lambda}\boldsymbol{E} + \boldsymbol{B}\left[(\hat{\boldsymbol{W}} - \boldsymbol{W}^*)^T \boldsymbol{h}(\boldsymbol{x}) + \omega\right] \quad (4.15)$$

Choose a Lyapunov function as

$$V = \frac{1}{2}\boldsymbol{E}^T \boldsymbol{P} \boldsymbol{E} + \frac{1}{2\gamma}(\hat{\boldsymbol{W}} - \boldsymbol{W}^*)^T (\hat{\boldsymbol{W}} - \boldsymbol{W}^*) \quad (4.16)$$

where γ is positive constant. $\hat{\boldsymbol{W}} - \boldsymbol{W}^*$ denotes the parameter estimation error, and the matrix \boldsymbol{P} is symmetric and positive definite and satisfies the following Lyapunov equation:

$$\boldsymbol{\Lambda}^T \boldsymbol{P} + \boldsymbol{P} \boldsymbol{\Lambda} = -\boldsymbol{Q}. \quad (4.17)$$

With $\boldsymbol{Q} \geq 0$, $\boldsymbol{\Lambda}$ is given by (4.10).

Choose $V_1 = \frac{1}{2}\boldsymbol{E}^T \boldsymbol{P} \boldsymbol{E}$, $V_2 = \frac{1}{2\gamma}(\hat{\boldsymbol{W}} - \boldsymbol{W}^*)^T (\hat{\boldsymbol{W}} - \boldsymbol{W}^*)$, and let $\boldsymbol{M} = \boldsymbol{B}\left[(\hat{\boldsymbol{W}} - \boldsymbol{W}^*)^T \boldsymbol{h}(\boldsymbol{x}) + \omega\right]$, then (4.15) becomes

$$\dot{\boldsymbol{E}} = \boldsymbol{\Lambda}\boldsymbol{E} + \boldsymbol{M}.$$

Then

$$\begin{aligned}\dot{V}_1 &= \frac{1}{2}\dot{\mathbf{E}}^T \mathbf{P} \mathbf{E} + \frac{1}{2}\mathbf{E}^T \mathbf{P} \dot{\mathbf{E}} = \frac{1}{2}(\mathbf{E}^T \Lambda^T + \mathbf{M}^T) \mathbf{P} \mathbf{E} + \frac{1}{2}\mathbf{E}^T \mathbf{P}(\Lambda \mathbf{E} + \mathbf{M}) \\ &= \frac{1}{2}\mathbf{E}^T(\Lambda^T \mathbf{P} + \mathbf{P} \Lambda) \mathbf{E} + \frac{1}{2}\mathbf{M}^T \mathbf{P} \mathbf{E} + \frac{1}{2}\mathbf{E}^T \mathbf{P} \mathbf{M} \\ &= -\frac{1}{2}\mathbf{E}^T \mathbf{Q} \mathbf{E} + \frac{1}{2}(\mathbf{M}^T \mathbf{P} \mathbf{E} + \mathbf{E}^T \mathbf{P} \mathbf{M}) = -\frac{1}{2}\mathbf{E}^T \mathbf{Q} \mathbf{E} + \mathbf{E}^T \mathbf{P} \mathbf{M}.\end{aligned}$$

Submitting \mathbf{M} into above, noting that $\mathbf{E}^T \mathbf{P} \mathbf{B} (\hat{\mathbf{W}} - \mathbf{W}^*)^T \mathbf{h}(\mathbf{x}) = (\hat{\mathbf{W}} - \mathbf{W}^*)^T [\mathbf{E}^T \mathbf{P} \mathbf{B} \mathbf{h}(\mathbf{x})]$, we get

$$\begin{aligned}\dot{V}_1 &= -\frac{1}{2}\mathbf{E}^T \mathbf{Q} \mathbf{E} + \mathbf{E}^T \mathbf{P} \mathbf{B} (\hat{\mathbf{W}} - \mathbf{W}^*)^T \mathbf{h}(\mathbf{x}) + \mathbf{E}^T \mathbf{P} \mathbf{B} \omega \\ &= -\frac{1}{2}\mathbf{E}^T \mathbf{Q} \mathbf{E} + (\hat{\mathbf{W}} - \mathbf{W}^*)^T \mathbf{E}^T \mathbf{P} \mathbf{B} \mathbf{h}(\mathbf{x}) + \mathbf{E}^T \mathbf{P} \mathbf{B} \omega \\ \dot{V}_2 &= \frac{1}{\gamma}(\hat{\mathbf{W}} - \mathbf{W}^*)^T \dot{\hat{\mathbf{W}}}.\end{aligned}$$

Then the derivative V becomes

$$\dot{V} = \dot{V}_1 + \dot{V}_2 = -\frac{1}{2}\mathbf{E}^T \mathbf{Q} \mathbf{E} + \mathbf{E}^T \mathbf{P} \mathbf{B} \omega + \frac{1}{\gamma}(\hat{\mathbf{W}} - \mathbf{W}^*)^T [\dot{\hat{\mathbf{W}}} + \gamma \mathbf{E}^T \mathbf{P} \mathbf{B} \mathbf{h}(\mathbf{x})].$$

Submitting the adaptive law (4.8) into above, we have

$$\dot{V} = -\frac{1}{2}\mathbf{E}^T \mathbf{Q} \mathbf{E} + \mathbf{E}^T \mathbf{P} \mathbf{B} \omega.$$

Since $-\frac{1}{2}\mathbf{E}^T \mathbf{Q} \mathbf{E} \leq 0$, consider the adaptive fuzzy control system convergence analysis in the book [1]; if we can make the approximation error ω very small by using RBF, we can get $\dot{V} \leq 0$.

4.1.3 Simulation Examples

4.1.3.1 First Simulation Example

Consider a linear plant as follows:

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

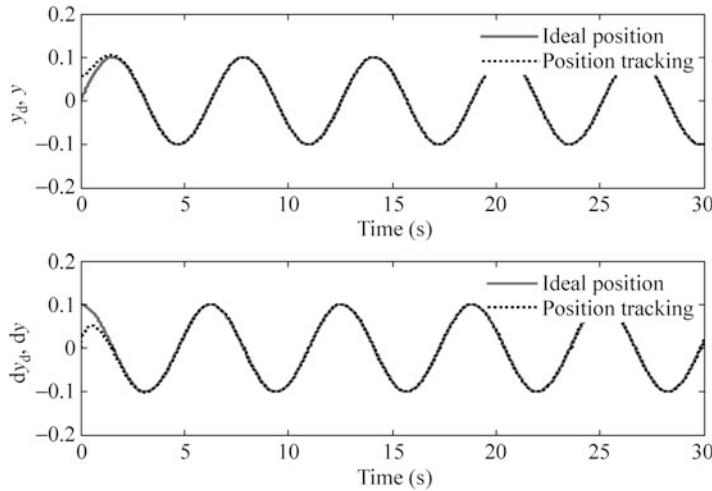


Fig. 4.2 Position and speed tracking

where x_1 and x_2 are position and speed, respectively; u is control input; $f(\mathbf{x}) = -25x_2$, and $g(\mathbf{x}) = 133$.

We use ideal position signal as $y_d(t) = 0.1 \sin t$, 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 parameter values are 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 c_i and b_i are designed as $[-2 \quad -1 \quad 0 \quad 1 \quad 2]$ and 0.20; the initial weight value is chosen as zero.

Adopting control law (4.6) and adaptive law (4.8), choosing $Q = \begin{bmatrix} 500 & 0 \\ 0 & 500 \end{bmatrix}$, $k_d = 50$, $k_p = 30$, and $\gamma = 1, 200$.

The results are shown in Figs. 4.2 and 4.3. The Simulink program of this example is chap4_1sim.mdl; the Matlab programs of the example are given in the [Appendix](#).

4.1.3.2 Second Simulation Example

Consider a single inverted pendulum system as in Fig. 4.4.

The dynamic equation is described as

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

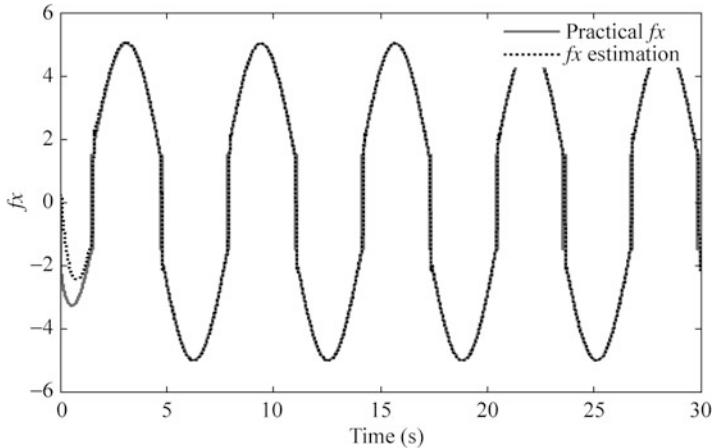
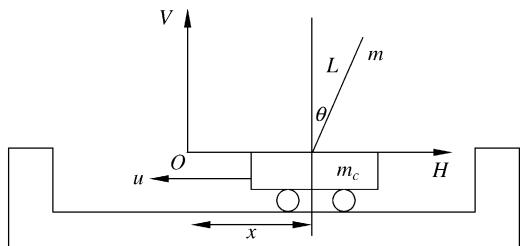


Fig. 4.3 $f(x)$ and $\hat{f}(x)$

Fig. 4.4 Single inverted pendulum system



where $f(\mathbf{x}) = \frac{g \sin x_1 - mlx_2^2 \cos x_1 \sin x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}$; $g(\mathbf{x}) = \frac{\cos x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}$, x_1 and x_2 are angle and angle speed value, respectively; $g = 9.8 \text{ m/s}^2$, $m_c = 1 \text{ kg}$ is mass of cart; $m = 0.1 \text{ kg}$ is mass of the pendulum; $l = 0.5 \text{ m}$ is the half length of the pendulum; and u is control input.

Consider ideal position 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.

According to the practical scope of x_1 and x_2 , the parameters of c_i and b_i are designed as $[-2 \ -1 \ 0 \ 1 \ 2]$ and 0.20; the initial weight value is chosen as zero.

Use control law (4.6) and adaptive law (4.8), and choose $\mathcal{Q} = \begin{bmatrix} 500 & 0 \\ 0 & 500 \end{bmatrix}$, $k_d = 50$, $k_p = 30$, and $\gamma = 1, 200$.

The results are shown in Figs. 4.5 and 4.6. The Simulink program of this example is chap4_2sim.mdl; the Matlab programs of the example are given in the Appendix.

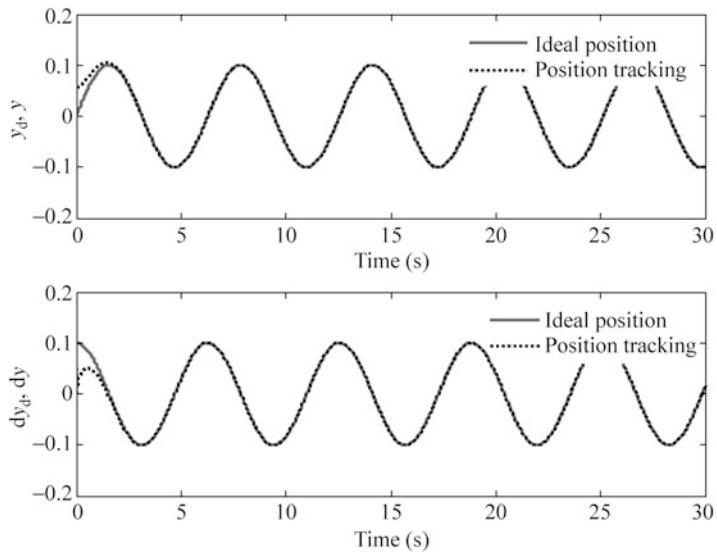


Fig. 4.5 Position and speed tracking

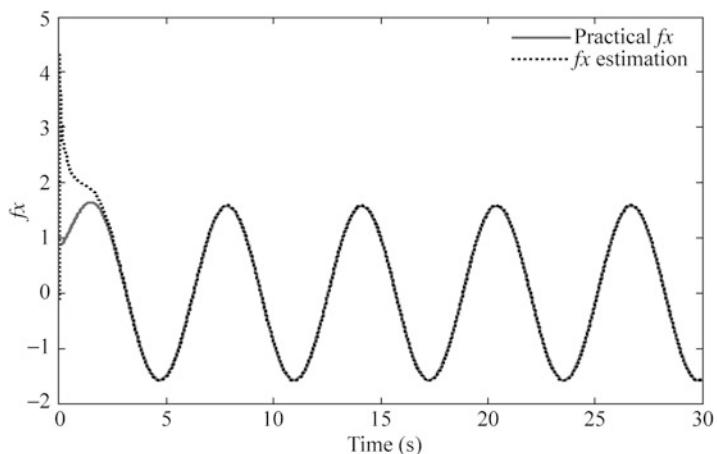


Fig. 4.6 $f(x)$ and $\hat{f}(x)$

4.2 Adaptive Control Based on Neural Approximation with Unknown Parameter

4.2.1 Problem Description

Consider a second-order nonlinear system

$$\ddot{x} = f(x, \dot{x}) + mu \quad (4.18)$$

where f is unknown nonlinear function, m are unknown, the lower bound of m is known, $m \geq -m_0$, and $-m_0 > 0$.

The Eq. (4.18) can also be written as

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(x) + mu \\ y &= x_1.\end{aligned}\quad (4.19)$$

We assume the ideal position signal is y_d , and let

$$e = y_d - y = y_d - x_1, \quad \mathbf{E} = [e \quad \dot{e}]^T.$$

We design the control law as

$$u^* = \frac{1}{m} [-f(\mathbf{x}) + \ddot{y}_d + \mathbf{K}^T \mathbf{E}] \quad (4.20)$$

Substitute (4.20) into (4.18), we can get the closed control system as

$$\ddot{e} + k_p e + k_d \dot{e} = 0 \quad (4.21)$$

We design $\mathbf{K} = [k_p \quad k_d]^T$ 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 $t \rightarrow \infty$, $e(t) \rightarrow 0$, and $\dot{e}(t) \rightarrow 0$.

From (4.20), we know that if the function $f(\mathbf{x})$ and parameter m are unknown, the control law will not be realized.

4.2.2 Adaptive Controller Design

4.2.2.1 RBF Neural Network Design

In this section, just like Sect. 4.1, with reference to the indirect adaptive fuzzy controller tactics given in [1], now we use RBF to replace fuzzy system to design RBF indirect adaptive controller.

The algorithm of RBF to approximate $f(\mathbf{x})$ is described as

$$\begin{aligned} h_j &= g\left(\|\mathbf{x} - \mathbf{c}_{ij}\|^2 / b_j^2\right) \\ f &= \mathbf{W}^T \mathbf{h}(\mathbf{x}) + \varepsilon \end{aligned}$$

where \mathbf{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, $\mathbf{h} = [h_1, h_2, \dots, h_n]^T$ denotes the output of hidden layer, \mathbf{W} is weight value, ε is approximation error, and $\varepsilon \leq \varepsilon_N$.

We use RBF to approximate f , the input vector is chosen as $\mathbf{x} = [e \quad \dot{e}]^T$, and the output of RBF is

$$\hat{f}(\mathbf{x}) = \hat{\mathbf{W}}^T \mathbf{h}(\mathbf{x}). \quad (4.22)$$

4.2.2.2 Control Law and Adaptive Law Design

If we use RBF neural network to represent the unknown nonlinear function f , the control law becomes

$$u = \frac{1}{\hat{m}} [-\hat{f}(\mathbf{x}) + \ddot{y}_d + \mathbf{K}^T \mathbf{E}] \quad (4.23)$$

$$\hat{f}(\mathbf{x}) = \hat{\mathbf{W}}^T \mathbf{h}(\mathbf{x}) \quad (4.24)$$

where $\mathbf{h}(\mathbf{x})$ is Gaussian function and $\hat{\mathbf{W}}$ is the estimated parameter for \mathbf{W} .

4.2.2.3 Stability Analysis

Submitting the control law (4.23) into (4.18), the closed-loop system is expressed as

$$\ddot{e} = -\mathbf{K}^T \mathbf{E} + (\hat{f}(\mathbf{x}) - f(\mathbf{x})) + (m - \hat{m})u. \quad (4.25)$$

Let

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -k_p & -k_d \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \quad (4.26)$$

Now, (4.25) can be rewritten as

$$\dot{\mathbf{E}} = \mathbf{AE} + \mathbf{B}[(\hat{f}(\mathbf{x}) - f(\mathbf{x})) + (m - \hat{m})u]. \quad (4.27)$$

The optimal weight value is

$$W^* = \arg \min_{W \in \Omega} [\sup |\hat{f}(x) - f(x)|]. \quad (4.28)$$

Define the modeling error as

$$\omega = \hat{f}(x|W^*) - f(x) \quad (4.29)$$

Then Eq. (4.27) becomes

$$\dot{E} = \Lambda E + B \{ [\hat{f}(x)] - \hat{f}(x|W^*) \} + \omega + (m - \hat{m})u. \quad (4.30)$$

Submitting Eq. (4.24) into (4.13), we can get closed equation as

$$\dot{E} = \Lambda E + B \left[(\hat{W} - W^*)^T h(x) + \omega + (m - \hat{m})u \right]. \quad (4.31)$$

Define a Lyapunov function as

$$V = \frac{1}{2} E^T P E + \frac{1}{2\gamma} (\hat{W} - W^*)^T (\hat{W} - W^*) + \frac{1}{2} \eta \tilde{m}^2 \quad (4.32)$$

where γ is positive constant. $\hat{W} - W^*$ denotes the parameter estimation error, and the matrix P is symmetric and positive definite and satisfies the following Lyapunov equation

$$\Lambda^T P + P \Lambda = -Q. \quad (4.33)$$

With $Q \geq 0$, Λ is given by (4.26), $\eta > 0$, and $\tilde{m} = m - \hat{m}$.

Choose $V_1 = \frac{1}{2} E^T P E$, $V_2 = \frac{1}{2\gamma} (\hat{W} - W^*)^T (\hat{W} - W^*)$, and $V_3 = \frac{1}{2} \eta \tilde{m}^2$, let $M = B \left[(\hat{W} - W^*)^T h(x) + \omega + \tilde{m}u \right]$, and then Eq. (4.31) becomes

$$\dot{E} = \Lambda E + M.$$

Then

$$\begin{aligned} \dot{V}_1 &= \frac{1}{2} \dot{E}^T P E + \frac{1}{2} E^T P \dot{E} = \frac{1}{2} (E^T \Lambda^T + M^T) P E + \frac{1}{2} E^T P (\Lambda E + M) \\ &= \frac{1}{2} E^T (\Lambda^T P + P \Lambda) E + \frac{1}{2} M^T P E + \frac{1}{2} E^T P M \\ &= -\frac{1}{2} E^T Q E + \frac{1}{2} (M^T P E + E^T P M) = -\frac{1}{2} E^T Q E + E^T P M. \end{aligned}$$

Submitting \mathbf{M} into above, noting that $\mathbf{E}^T \mathbf{P} \mathbf{B} (\hat{\mathbf{W}} - \mathbf{W}^*)^T \mathbf{h}(\mathbf{x}) = (\hat{\mathbf{W}} - \mathbf{W}^*)^T [\mathbf{E}^T \mathbf{P} \mathbf{B} \mathbf{h}(\mathbf{x})]$, we get

$$\begin{aligned}\dot{V}_1 &= -\frac{1}{2} \mathbf{E}^T \mathbf{Q} \mathbf{E} + \mathbf{E}^T \mathbf{P} \mathbf{B} (\hat{\mathbf{W}} - \mathbf{W}^*)^T \mathbf{h}(\mathbf{x}) + \mathbf{E}^T \mathbf{P} \mathbf{B} \omega + \mathbf{E}^T \mathbf{P} \mathbf{B} \tilde{m} u \\ &= -\frac{1}{2} \mathbf{E}^T \mathbf{Q} \mathbf{E} + (\hat{\mathbf{W}} - \mathbf{W}^*)^T \mathbf{E}^T \mathbf{P} \mathbf{B} \mathbf{h}(\mathbf{x}) + \mathbf{E}^T \mathbf{P} \mathbf{B} \omega + \mathbf{E}^T \mathbf{P} \mathbf{B} \tilde{m} u \\ \dot{V}_2 &= \frac{1}{\gamma} (\hat{\mathbf{W}} - \mathbf{W}^*)^T \dot{\hat{\mathbf{W}}} \\ \dot{V}_3 &= -\eta \tilde{m} \dot{\hat{m}}.\end{aligned}$$

Then the derivative V becomes

$$\begin{aligned}\dot{V} &= \dot{V}_1 + \dot{V}_2 + \dot{V}_3 \\ &= -\frac{1}{2} \mathbf{E}^T \mathbf{Q} \mathbf{E} + \mathbf{E}^T \mathbf{P} \mathbf{B} \omega + \frac{1}{\gamma} (\hat{\mathbf{W}} - \mathbf{W}^*)^T [\dot{\hat{\mathbf{W}}} + \gamma \mathbf{E}^T \mathbf{P} \mathbf{B} \mathbf{h}(\mathbf{x})] + \tilde{m} (\mathbf{E}^T \mathbf{P} \mathbf{B} u - \eta \dot{\hat{m}})\end{aligned}$$

We choose the adaptive law as

$$\dot{\hat{\mathbf{W}}} = -\gamma \mathbf{E}^T \mathbf{P} \mathbf{B} \mathbf{h}(\mathbf{x}). \quad (4.34)$$

To guarantee $\tilde{m} (\mathbf{E}^T \mathbf{P} \mathbf{B} u - \eta \dot{\hat{m}}) \leq 0$, at the same time to avoid singularity in (4.23) and guarantee $\hat{m} \geq \underline{m}$, we used the adaptive law tactics proposed in [2] as

$$\dot{\hat{m}} = \begin{cases} \frac{1}{\eta} \mathbf{E}^T \mathbf{P} \mathbf{B} u, & \text{if } \mathbf{E}^T \mathbf{P} \mathbf{B} u > 0 \\ \frac{1}{\eta} \mathbf{E}^T \mathbf{P} \mathbf{B} u, & \text{if } \mathbf{E}^T \mathbf{P} \mathbf{B} u \leq 0 \text{ and } \hat{m} > \underline{m} \\ \frac{1}{\eta}, & \text{if } \mathbf{E}^T \mathbf{P} \mathbf{B} u \leq 0 \text{ and } \hat{m} \leq \underline{m} \end{cases} \quad (4.35)$$

where $\hat{m}(0) \geq \underline{m}$.

Reference to [2], the adaptive law (4.35) can also be analyzed as:

1. If $\mathbf{E}^T \mathbf{P} \mathbf{B} u > 0$, we get $\tilde{m} (\mathbf{E}^T \mathbf{P} \mathbf{B} u - \eta \dot{\hat{m}}) = 0$ and $\dot{\hat{m}} > 0$, thus, $\hat{m} > \underline{m}$;
2. If $\mathbf{E}^T \mathbf{P} \mathbf{B} u \leq 0$ and $\hat{m} > \underline{m}$, we get $\tilde{m} (\mathbf{E}^T \mathbf{P} \mathbf{B} u - \eta \dot{\hat{m}}) = 0$;
3. If $\mathbf{E}^T \mathbf{P} \mathbf{B} u \leq 0$ and $\hat{m} \leq \underline{m}$, we have $\tilde{m} = m - \hat{m} \geq m - \underline{m} > 0$, thus, $\tilde{m} (\mathbf{E}^T \mathbf{P} \mathbf{B} u - \eta \dot{\hat{m}}) = \tilde{m} \mathbf{E}^T \mathbf{P} \mathbf{B} u - \tilde{m} \leq 0$, \hat{m} will increase gradually, and then $\hat{m} > \underline{m}$ will be guaranteed with $\dot{\hat{m}} > 0$.

Submitting the adaptive law (4.34) and (4.35) into above, we have

$$\dot{V} = -\frac{1}{2} \mathbf{E}^T \mathbf{Q} \mathbf{E} + \mathbf{E}^T \mathbf{P} \mathbf{B} \omega.$$

Since $-\frac{1}{2}\mathbf{E}^T \mathbf{Q} \mathbf{E} \leq 0$, consider the adaptive fuzzy control system convergence analysis in [1]; if we can make the approximation error ω very small by using RBF, we can get $\dot{V} \leq 0$.

4.2.3 Simulation Examples

Consider a simple plant as

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

where x_1 and x_2 are position and speed, respectively; u is control input; $f(\mathbf{x}) = -25x_2 - 10x_1$, and $m = 133$.

We use ideal position signal as $y_d(t) = \sin t$, and choose the initial states of the plant as $[0, 50, 0]$.

We use RBF to approximate $f(\mathbf{x})$, and design adaptive algorithm to estimate parameter m . The structure is used as 2-5-1; the input vector of RBF is $\mathbf{z} = [x_1 \ x_2]^T$. For each Gaussian function, the parameters of c_i and b_i are designed as $[-1 \ -0.5 \ 0 \ 0.5 \ 1]$ and 2.0. The initial weight value is chosen as zero.

In simulation, we use control law (4.23) and adaptive law (4.34) and (4.35); the parameters are chosen as $Q = \begin{bmatrix} 500 & 0 \\ 0 & 500 \end{bmatrix}$, $k_p = 30$, $k_d = 50$, $\gamma = 1,200$, $\eta = 0.0001$, $\underline{m} = 100$, and $\hat{m}(0) = 120$.

The results are shown from Figs. 4.7, 4.8, and 4.9. The estimation of $f(x)$ and m do not converge to true value of $f(x)$ and m . This is due to the fact that the desired trajectory is not persistently exciting, and this occurs quite often in real-world application, which has been explained in the Sect. 1.5 in Chap. 1.

The Simulink program of this example is chap4_3sim.mdl; the Matlab programs of the example are given in the [Appendix](#).

4.3 A Direct Method for Robust Adaptive Control by RBF

4.3.1 System Description

Considering the proposed controller in paper [3] and book [4], we analyze design and simulation method of a direct method for robust adaptive control by RBF.

Consider the SISO second-order nonlinear system in the following form:

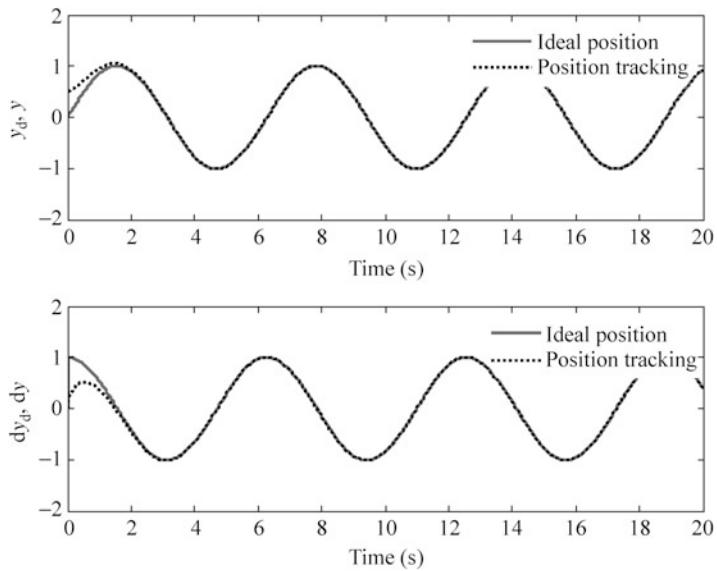


Fig. 4.7 Position and speed tracking

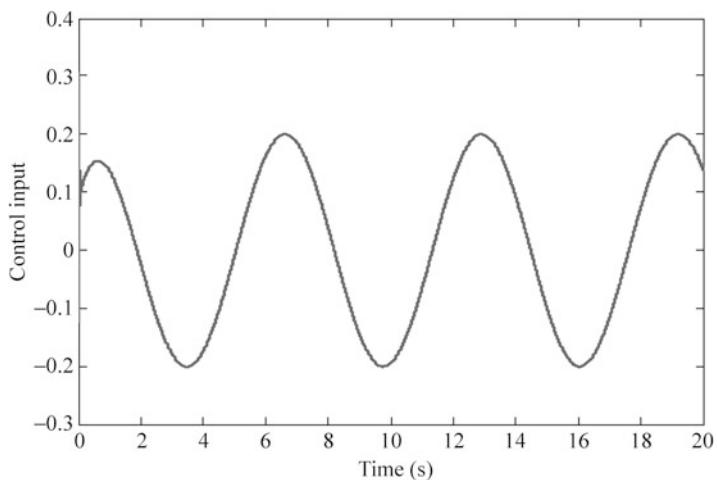


Fig. 4.8 Control input

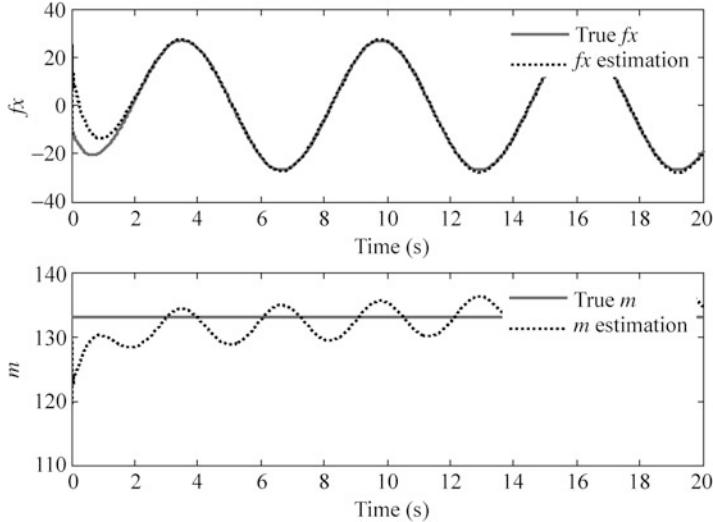


Fig. 4.9 Estimation of $f(x)$ and m

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= \alpha(x) + \beta(x)u + d(t) \\ y &= x_1\end{aligned}\tag{4.36}$$

where $\mathbf{x} = [x_1 \ x_2]^T \in \mathbb{R}$, $u \in \mathbb{R}$, and $y \in \mathbb{R}$ are state variables, system input, and output, respectively; $\alpha(x)$ and $\beta(x)$ are unknown smooth functions; and $d(t)$ denotes the external disturbance bound by a known constant $d_0 > 0$, that is, $|d(t)| \leq d_0$.

Assumption 4.1. The sign of $\beta(x)$ is known and $\beta(x) \neq 0$, $\forall x \in \Omega$.

Since the sign of $\beta(x)$ is known, without losing generality, assume that $\beta(x) > 0$.

Assumption 4.2. There exists a known smooth function $\bar{\beta}$ such that $|\beta(x)| \leq \bar{\beta}$.

Define vector \mathbf{x}_d , \mathbf{e} , and an augmented s item as

$$\begin{aligned}\mathbf{x}_d &= [y_d \ \dot{y}_d]^T \\ \mathbf{e} &= \mathbf{x} - \mathbf{x}_d = [e \ \dot{e}]^T, \quad s = [\lambda \ 1]\mathbf{e} = \lambda e + \dot{e}\end{aligned}\tag{4.37}$$

where λ is chosen as $\lambda > 0$ such that polynomial $s + \lambda$ is Hurwitz.

From (4.37), we have

$$\begin{aligned}\dot{s} &= \lambda \dot{e} + \ddot{e} = \lambda \dot{e} + \ddot{x}_1 - \ddot{y}_d \\ &= \lambda \dot{e} + \alpha(x) + \beta(x)u + d(t) - \ddot{y}_d \\ &= \alpha(x) + v + \beta(x)u + d(t)\end{aligned}\tag{4.38}$$

where $v = -\ddot{y}_d + \lambda \dot{e}$.

4.3.2 Desired Feedback Control and Function Approximation

Lemma 4.1. Consider system (4.36) with Assumptions 4.1 and 4.2 satisfied and $d(t) = 0$. If the desired controller is chosen as

$$u^* = -\frac{1}{\beta(x)}(\alpha(x) + v) - \left(\frac{1}{\epsilon\beta(x)} + \frac{1}{\epsilon\beta^2(x)} - \frac{\dot{\beta}(x)}{2\beta^2(x)} \right) s \quad (4.39)$$

where $\epsilon > 0$ is a design parameter, then $\lim_{t \rightarrow \infty} \|e(t)\| = 0$.

Proof. Substituting $u = u^*$ into (4.38) and noting $d(t) = 0$, we have

$$\begin{aligned} \dot{s} &= \alpha(x) + v + \beta(x) \left(-\frac{1}{\beta(x)}(\alpha(x) + v) - \left(\frac{1}{\epsilon\beta(x)} + \frac{1}{\epsilon\beta^2(x)} - \frac{\dot{\beta}(x)}{2\beta^2(x)} \right) s \right) \\ &= -\left(\frac{1}{\epsilon} + \frac{1}{\epsilon\beta(x)} - \frac{\dot{\beta}(x)}{2\beta(x)} \right) s = -\left(\frac{1}{\epsilon} + \frac{1}{\epsilon\beta(x)} \right) s + \frac{\dot{\beta}(x)}{2\beta(x)} s. \end{aligned}$$

Choosing a Lyapunov function as $V = \frac{1}{2\beta(x)}s^2$, then

$$\begin{aligned} \dot{V} &= \frac{1}{\beta(x)}s\dot{s} - \frac{\dot{\beta}(x)}{2\beta^2(x)}s^2 \\ &= \frac{1}{\beta(x)}s \left[-\left(\frac{1}{\epsilon} + \frac{1}{\epsilon\beta(x)} \right) s + \frac{\dot{\beta}(x)}{2\beta(x)} s \right] - \frac{\dot{\beta}(x)}{2\beta^2(x)}s^2 \\ &= -\left(\frac{1}{\epsilon\beta(x)} + \frac{1}{\epsilon\beta^2(x)} \right) s^2 \leq 0 \end{aligned} \quad (4.40)$$

Since $\beta(x) > 0$, then we have $\dot{V} \leq 0$, which implies that $\lim_{t \rightarrow \infty} |s| = 0$; subsequently, we have $\lim_{t \rightarrow \infty} \|e(t)\| = 0$.

From (4.40), it is shown that the smaller the parameter ϵ is, the more negative \dot{V} is. Hence, the convergence rate of the tracking error can be adjusted by tuning the design parameter ϵ .

From (4.39), the desired controller u^* can be written as a function of x , s , and v as

$$z = \begin{bmatrix} x^T & s & \frac{s}{\epsilon} & v \end{bmatrix}^T \in \Omega_z \subset R^5 \quad (4.41)$$

where compact set Ω_z is defined as

$$\Omega_z = \left\{ \begin{pmatrix} x^T & s & \frac{s}{\epsilon} & v \end{pmatrix} \mid x \in \Omega; |x_d| \in \Omega_d; s = [\lambda \ 1]^T e; v = -\ddot{y}_d + \lambda \dot{e} \right\} \quad (4.42)$$

When nonlinear functions $\alpha(x)$ and $\beta(x)$ are unknown, $u^*(z)$ is not available. In the following, RBF neural network is applied for approximating the unknown function $u^*(z)$.

It should be noted that though the elements s and $\frac{s}{\epsilon}$ in the input vector z are dependent due to constant ϵ , they are in different scales when a very small ϵ is chosen. Thus, $\frac{s}{\epsilon}$ is also fed into neural network as an input for improving the NN approximation accuracy.

There exist an ideal constant weight vector W^* , such that

$$u^*(z) = W^{*T} \mathbf{h}(z) + \mu_l, \quad \forall z \in \Omega_z \quad (4.43)$$

where $\mathbf{h}(z)$ is radial-basis function vector, μ_l is called the NN approximation error satisfying $|\mu_l| \leq \mu_0$, and

$$W^* = \arg \min_{W \in R^l} \left\{ \sup_{z \in \Omega_z} |W^T \mathbf{h}(z) - u^*(z)| \right\}.$$

4.3.3 Controller Design and Performance Analysis

Figure 4.10 shows the closed-loop neural-based adaptive control scheme.

Let \hat{W} be an estimate of the ideal NN weight W^* , the direct adaptive controller is designed as

$$u = \hat{W}^T \mathbf{h}(z). \quad (4.44)$$

The adaptive law is designed as

$$\dot{\hat{W}} = -\Gamma(\mathbf{h}(z)s + \sigma \hat{W}) \quad (4.45)$$

where $\Gamma = \Gamma^T > 0$ is an adaptation gain matrix and $\sigma > 0$ is a constant.

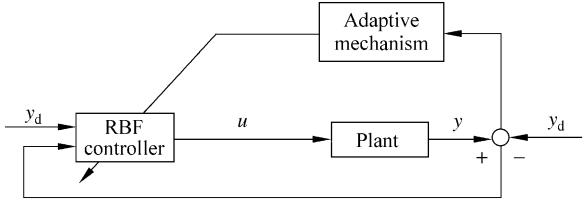
Substituting controller (4.44) into (4.38), error equation (4.38) can be rewritten as

$$\dot{s} = \alpha(x) + v + \beta(x)\hat{W}^T \mathbf{h}(z) + d(t). \quad (4.46)$$

Adding and subtracting on the right-hand side of (4.46) and noting (4.43), we have

$$\dot{s} = \alpha(x) + v + \beta(x)(\hat{W}^T \mathbf{h}(z) - W^{*T} \mathbf{h}(z) - \mu_l) + \beta(x)u^*(z) + d(t). \quad (4.47)$$

Fig. 4.10 Block diagram of direct RBF control scheme



From (4.39) we have

$$u^* = -\frac{1}{\beta(x)}(\alpha(x) + v) - \left(\frac{1}{\varepsilon\beta(x)} + \frac{1}{\varepsilon\beta^2(x)} - \frac{\dot{\beta}(x)}{2\beta^2(x)} \right) s.$$

Substituting above equation into (4.47) leads to

$$\dot{s} = \beta(x) \left(\tilde{W}^T \mathbf{h}(z) - \mu_l \right) - \left(\frac{1}{\varepsilon} + \frac{1}{\varepsilon\beta(x)} - \frac{\dot{\beta}(x)}{2\beta(x)} \right) s + d(t) \quad (4.48)$$

where $\tilde{W} = \hat{W} - W^*$.

The item $\beta(x)$ exists as a coefficient of \tilde{W}^T in (4.48), which can make $\beta(x)$ appear in \dot{V} if we design $\frac{1}{2}s^2$ as a part of Lyapunov function V , and can cause $\beta(x)$ to be included in the adaptive law $\dot{\hat{W}}$.

To avoid $\beta(x)$ being included in the adaptive law $\dot{\hat{W}}$, $\frac{1}{2}\frac{s^2}{\beta(x)}$ should be used instead of $\frac{1}{2}s^2$, then the Lyapunov function is designed as

$$V = \frac{1}{2} \left(\frac{s^2}{\beta(x)} + \tilde{W}^T \Gamma^{-1} \tilde{W} \right). \quad (4.49)$$

Differentiating (4.49) along (4.45) and (4.48) yields

$$\begin{aligned} \dot{V} &= \frac{s\dot{s}}{\beta(x)} - \frac{\dot{\beta}(x)}{2\beta^2(x)} s^2 + \tilde{W}^T \Gamma^{-1} \dot{\tilde{W}} \\ &= \frac{s}{\beta(x)} \left(\beta(x) \left(\tilde{W}^T \mathbf{h}(z) - \mu_l \right) - \left(\frac{1}{\varepsilon} + \frac{1}{\varepsilon\beta(x)} - \frac{\dot{\beta}(x)}{2\beta(x)} \right) s + d(t) \right) \\ &\quad - \frac{\dot{\beta}(x)}{2\beta^2(x)} s^2 + \tilde{W}^T \Gamma^{-1} (-\Gamma(\mathbf{h}(z)s + \sigma\hat{W})) \\ &= - \left(\frac{1}{\varepsilon\beta(x)} + \frac{1}{\varepsilon\beta^2(x)} \right) s^2 + \frac{d(t)}{\beta(x)} s - \mu_l s - \sigma\tilde{W}^T \hat{W} \end{aligned}$$

Use the facts that

$$\begin{aligned}
2\tilde{W}^T\hat{W} &= \tilde{W}^T(\tilde{W} + W^*) + (\hat{W} - W^*)^T\hat{W} \\
&= \tilde{W}^T\tilde{W} + (\hat{W} - W^*)^TW^* + \hat{W}^T\hat{W} - W^{*T}\hat{W} \\
&= \|\tilde{W}\|^2 + \|\hat{W}\|^2 - \|W^*\|^2 \geq \|\tilde{W}\|^2 - \|W^*\|^2 \\
\frac{d(t)}{\beta(x)}s &\leq \frac{s^2}{\epsilon\beta^2(x)} + \frac{\epsilon}{4}d(t)^2 \\
|\mu_l s| &\leq \frac{s^2}{2\epsilon\beta(x)} + \frac{\epsilon}{2}\mu_l^2\beta(x) \leq \frac{s^2}{2\epsilon\beta(x)} + \frac{\epsilon}{2}\mu_l^2\bar{\beta}
\end{aligned}$$

and note that $|\mu_l| \leq \mu_0$, $|d(t)| \leq d_0$, the following inequality holds

$$\dot{V} \leq -\frac{s^2}{2\epsilon\beta(x)} - \frac{\sigma}{2}\|\tilde{W}\|^2 + \frac{\epsilon}{2}\mu_0^2\bar{\beta} + \frac{\epsilon}{4}d_0^2 + \frac{\sigma}{2}\|W^*\|^2.$$

Considering $\tilde{W}^T\Gamma^{-1}\tilde{W} \leq \bar{\gamma}\|\tilde{W}\|^2$ ($\bar{\gamma}$ is the largest eigenvalue of Γ^{-1}), we obtain

$$\dot{V} \leq -\frac{1}{\alpha_0}V + \frac{\epsilon}{2}\mu_0^2\bar{\beta} + \frac{\epsilon}{4}d_0^2 + \frac{\sigma}{2}\|W^*\|^2$$

where $\alpha_0 = \max\{\epsilon, \bar{\gamma}/\sigma\}$. Solving the above inequality using Lemma B.5 in [5], we have

$$\begin{aligned}
V(t) &\leq e^{-t/\alpha_0}V(0) + \left(\frac{\epsilon}{2}\mu_0^2\bar{\beta} + \frac{\epsilon}{4}d_0^2 + \frac{\sigma}{2}\|W^*\|^2\right)\int_0^t e^{-(t-\tau)/\alpha_0}d\tau \\
&\leq e^{-t/\alpha_0}V(0) + \alpha_0\left(\frac{\epsilon}{2}\mu_0^2\bar{\beta} + \frac{\epsilon}{4}d_0^2 + \frac{\sigma}{2}\|W^*\|^2\right), \quad \forall t \geq 0. \tag{4.50}
\end{aligned}$$

Since $V(0)$ is bounded, inequality (4.50) shows that s and $\hat{W}(t)$ are bounded. By (4.49) it follows that $V \geq \frac{1}{2}\frac{s^2}{\beta(x)}$, thus, $s \leq \sqrt{2\beta(x)V} \leq \sqrt{2\bar{\beta}V}$.

Combining with (4.50), noting $\sqrt{ab} \leq \sqrt{a} + \sqrt{b}$ ($a > 0, b > 0$), we obtain

$$|s| \leq e^{-t/2\alpha_0}\sqrt{2\bar{\beta}V(0)} + \sqrt{\alpha_0\bar{\beta}}\left(\epsilon\mu_0^2\bar{\beta} + \frac{\epsilon}{2}d_0^2 + \sigma\|s\|^2\right)^{1/2}, \quad \forall t \geq 0.$$

4.3.4 Simulation Example

4.3.4.1 First Example

The plant dynamics can be expressed in the form of system (4.36) as follows

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= -25x_2 + 133u + d(t) \\ y &= x_1\end{aligned}$$

where $\alpha(x) = -25x_2$, $\beta(x) = 133$, $d(t) = 100 \sin t$, and $\mathbf{x} = [x_1 \ x_2]^T = [\theta \ \dot{\theta}]^T$.

The initial states are $\mathbf{x} = [0.5 \ 0]^T$, and the reference signal is $y_d = \sin t$. The input vector of RBF is $\mathbf{z} = [x_1 \ x_2 \ s \ s/\varepsilon \ v]^T$, the network structure 5-9-1 is used. The parameters of c_i and b_i must be chosen according to the scope of the input value. If the parameter values are chosen inappropriately, Gaussian function will not be effectively mapped, and RBF network will be invalid. In this example, according to the practical scope of x_1 , x_2 , s , s/ε , and v , according to (4.42), for each Gaussian function, the parameters of c_i and b_i are designed as $[-2 \ -1.5 \ -1 \ -0.5 \ 0 \ 0.5 \ 1 \ 1.5 \ 2]$ and 5.0.

The adaptive controllers (4.44) and (4.45) are used; we set $\lambda = 5.0$, $\Gamma_{ii} = 500$ ($i = 9$), $\varepsilon = 0.25$ and $\sigma = 0.005$, from $\beta(x)$ expression we set $\bar{\beta} = 150$; and the initial weight value is chosen as zero.

The results are shown from Figs. 4.11 and 4.12. The Simulink program of this example is chap4_4sim.mdl; the Matlab programs of the example are given in the Appendix.

4.3.4.2 Second Example

A pendulum plant dynamics can be expressed in the form of system (4.36) as follows [3]:

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= \alpha(x) + \beta(x)u + d(t) \\ y &= x_1\end{aligned}$$

where $\alpha(x) = \frac{0.5 \sin x_1(1+0.5 \cos x_1)x_2^2 - 10 \sin x_1(1+\cos x_1)}{0.25(2+\cos x_1)^2}$, $\beta(x) = \frac{1}{0.25(2+\cos x_1)^2}$, $d(t) = d_1(t) \cos x_1$, $\mathbf{x} = [x_1 \ x_2]^T = [\theta \ \dot{\theta}]^T$, $d_1 = \cos(3t)$.

The initial states are $\mathbf{x} = [0 \ 0]^T$, and the reference signal is $y_d = \frac{\pi}{6} \sin t$. The operation range of the system is chosen as $\Omega = \{(x_1, x_2) | |x_1| \leq \frac{\pi}{2}, |x_2| \leq 4\pi\}$.

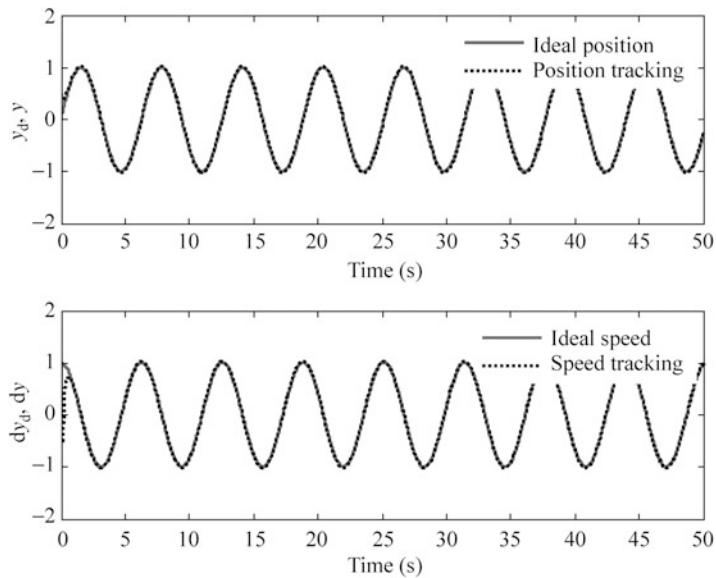


Fig. 4.11 Position and speed tracking

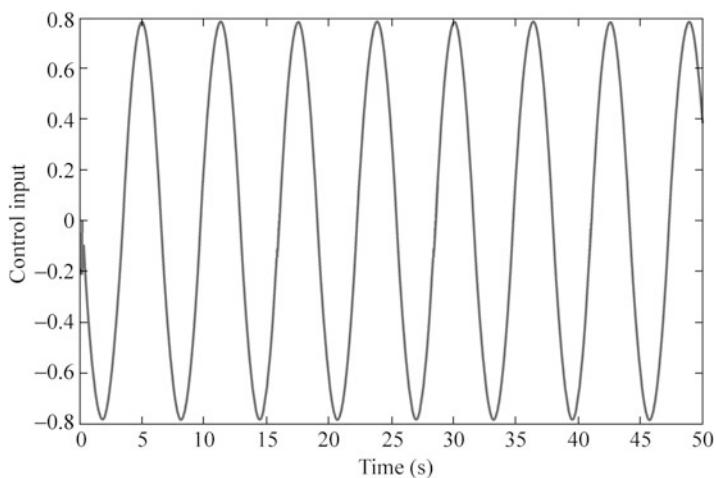


Fig. 4.12 Control input

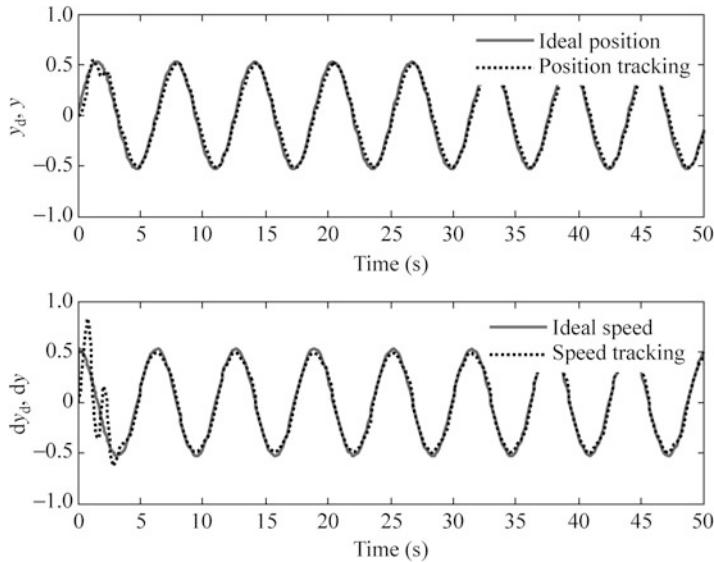


Fig. 4.13 Position and speed tracking

The input vector of RBF is $\mathbf{z} = [x_1 \ x_2 \ s \ \frac{s}{\epsilon} \ v]^T$, the network structure 5-13-1 is used. In this example, according to the practical scope of $x_1, x_2, s, s/\epsilon$, and v , according to (4.42), for each Gaussian function, the parameters of c_i and b_i are designed as $[-6 \ -5 \ -4 \ -3 \ -2 \ -1 \ 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6]$ and 3.0.

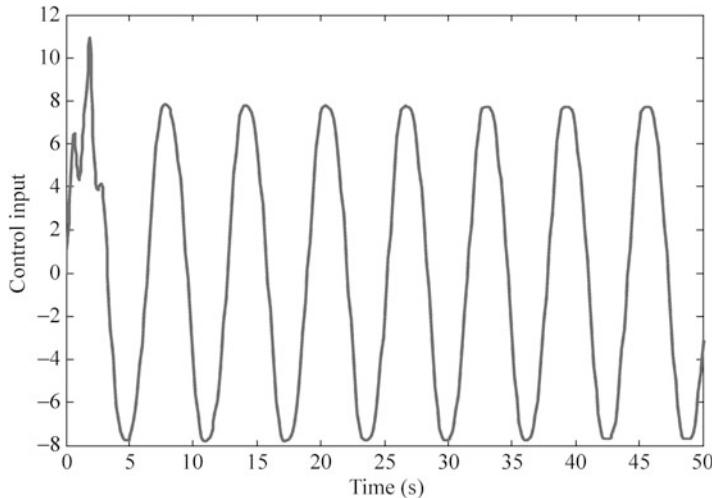
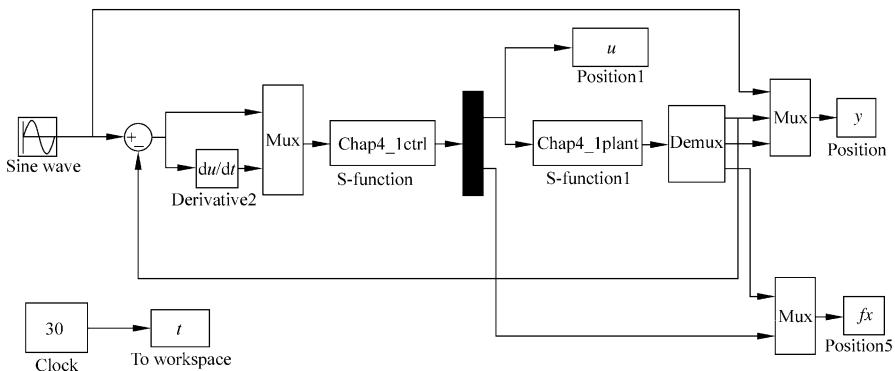
The adaptive controllers (4.44) and (4.45) are used; we set $\lambda = 10$, $\Gamma_{ii} = 15$ ($i = 13$), $\epsilon = 0.25$ and $\sigma = 0.005$, and from $\beta(x)$ expression we set $\bar{\beta} = 1$. The initial weight value is chosen as zero.

The results are shown in Figs. 4.13 and 4.14. The Simulink program of this example is chap4_5sim.mdl; the Matlab programs of the example are given in the Appendix.

Appendix

Programs for Sect. 4.1.3.1

1. Simulink program: chap4_1sim.mdl

**Fig. 4.14** Control input

2. S function for controller: chap4_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=2;
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);

e=u(1);
de=u(2);
x1=yd-e;
x2=dyd-de;

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=[e;de];
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);
e=u(1);
de=u(2);
x1=yd-e;
x2=dyd-de;
kp=30;kd=50;
K=[kp kd]';
E=[e de]';
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;
gx=133;
ut=1/gx*(-fxp+ddyd+K'*E);
sys(1)=ut;
sys(2)=fxp;

```

3. S function for plant: chap4_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)
F=10*x(2)+1.5*sign(x(2));
fx=-25*x(2)-F;
sys(1)=x(2);
sys(2)=fx+133*u;
function sys=mdlOutputs(t,x,u)
F=10*x(2)+1.5*sign(x(2));
fx=-25*x(2)-F;
sys(1)=x(1);
sys(2)=x(2);
sys(3)=fx;

```

4. Plot program: chap4_1plot.m

```

close all;
figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('ideal position','position tracking');
subplot(212);
plot(t,0.1*cos(t),'r',t,y(:,3),'k:','linewidth',2);
xlabel('time(s)');ylabel('dyd,dy');
legend('ideal speed','speed tracking');

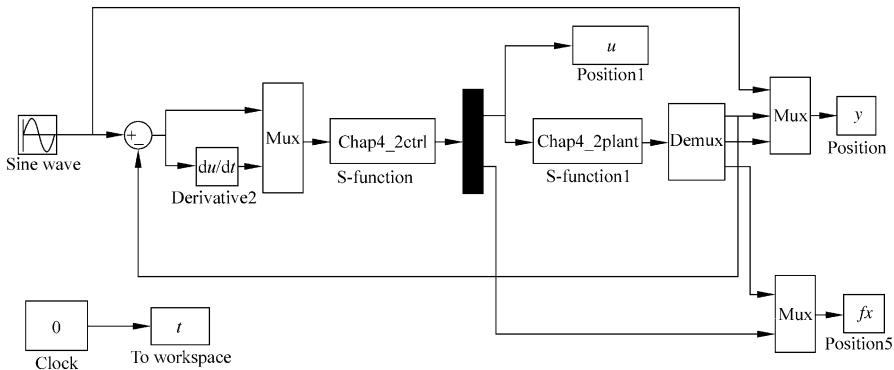
figure(2);
plot(t,u(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');

figure(3);
plot(t,fx(:,1),'r',t,fx(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('fx');
legend('Practical fx','fx estimation');

```

Programs for Sect. 4.1.3.2

1. Simulink program: chap4_2sim.mdl;



2. S function for controller: chap4_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 = 2;
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);

e=u(1);
de=u(2);
x1=yd-e;
x2=dyd-de;

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=[e;de];
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);

```

```

e=u(1);
de=u(2);
x1=yd-e;
x2=ddy-de;

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
fzp=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*(-fzp+ddy+K'*E);

sys(1)=ut;
sys(2)=fzp;

```

3. S function for plant: chap4_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=1*(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=1*(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: chap4_2plot.m

```

close all;

figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('ideal position','position tracking');
subplot(212);
plot(t,0.1*cos(t),'r',t,y(:,3),'k:','linewidth',2);
xlabel('time(s)');ylabel('dyd,dy');
legend('ideal speed','speed tracking');

figure(2);

```

```

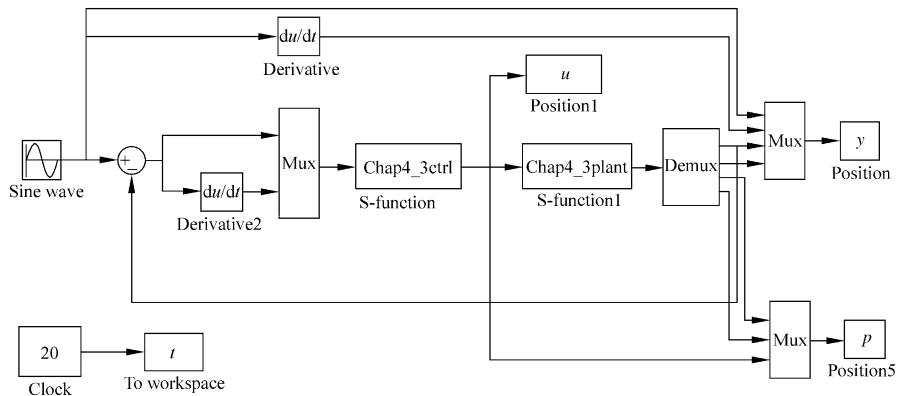
plot(t,u(:,1),'r','LineWidth',2);
xlabel('time(s)');ylabel('Control input');

figure(3);
plot(t,fx(:,1),'r',t,fx(:,2),'k:','LineWidth',2);
xlabel('time(s)');ylabel('fx');
legend('Practical fx','fx estimation');

```

Programs for Sect. 4.2

1. Simulink program: chap4_3sim.mdl;



2. S function for controller: chap4_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 = 2;
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);

e=u(1);
de=u(2);
x1=yd-e;
x2=dyd-de;

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=[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;

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;
else
    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);

e=u(1);
de=u(2);
x1=yd-e;
x2=dyd-de;

kp=30;
kd=50;
K=[kp kd]';
E=[e de]';

W=[x(1) x(2) x(3) x(4) x(5)]';
xi=[e;de];
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 for plant: chap4_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: chap4_3plot.m

```

close all;

figure(1);
subplot(211);
plot(t,y(:,1), 'r', t,y(:,3), 'k:', 'linewidth', 2);
xlabel('time(s)'); ylabel('yd,y');
legend('ideal position','position tracking');
subplot(212);
plot(t,y(:,2), 'r', t,y(:,4), 'k:', 'linewidth', 2);
xlabel('time(s)'); ylabel('dyd,dy');
legend('ideal speed','speed tracking');

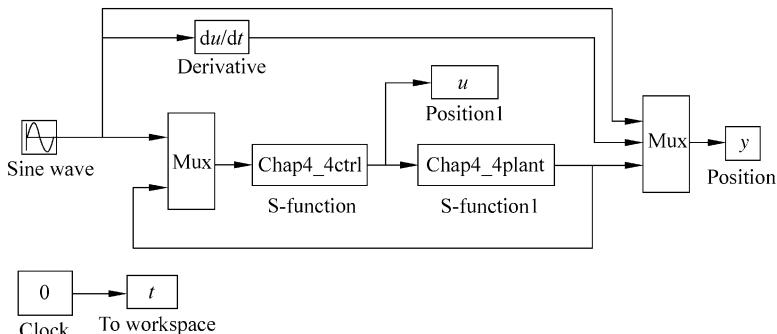
figure(2);
plot(t,u(:,1), 'r', 'linewidth', 2);
xlabel('time(s)'); ylabel('Control input');

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');

```

Programs for Sect. 4.3.4.1

Main Simulink program: chap4_4sim.mdl



Control law program: chap4_4ctrl.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 lambd epc
lambd=5;
epc=0.25;
node=9;
sizes = simsizes;
sizes.NumContStates = node;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = zeros(1,9);
c= [-2 -1.5 -1 -0.5 0 0.5 1 1.5 2;
      -2 -1.5 -1 -0.5 0 0.5 1 1.5 2;
      -2 -1.5 -1 -0.5 0 0.5 1 1.5 2;
      -2 -1.5 -1 -0.5 0 0.5 1 1.5 2;
      -2 -1.5 -1 -0.5 0 0.5 1 1.5 2];
b=5;
str = [];
ts = [];
function sys=mdlDerivatives(t,x,u)
global node c b lambd epc
yd=sin(t);
dyd=cos(t);
ddyd=-sin(t);
x1=u(2);
x2=u(3);
e=x1-yd;
de=x2-dyd;
```

```
s=lambd*e+de;
v=-ddyd+lambd*de;
xi=[x1;x2;s;s/epc;v];

h=zeros(9,1);
for j=1:1:9
    h(j)=exp(-norm(xi-c(:,j))^2/(2*b^2));
end

rou=0.005;
Gama=500*eye(node);
W=[x(1) x(2) x(3) x(4) x(5) x(6) x(7) x(8) x(9)]';
S=-Gama*(h*s+rou*W);

for i=1:1:node
    sys(i)=S(i);
end

function sys=mdlOutputs(t,x,u)
global node c b lambd epc
yd=sin(t);
dyd=cos(t);
ddyd=-sin(t);
x1=u(2);
x2=u(3);
e=x1-yd;
de=x2-dyd;
s=lambd*e+de;
v=-ddyd+lambd*de;
xi=[x1;x2;s;s/epc;v];

W=[x(1) x(2) x(3) x(4) x(5) x(6) x(7) x(8) x(9)]';
h=zeros(9,1);
for j=1:1:9
    h(j)=exp(-norm(xi-c(:,j))^2/(2*b^2));
end
betaU=150;
ut=1/betaU*W'*h;

sys(1)=ut;
Plant program: chap4_4plant.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 = 2;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.5 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
ut=u(1);
dt=100*sin(t);
sys(1)=x(2);
sys(2)=-25*x(2)+133*ut+dt;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
Plot program: chap4_4plot.m
close all;

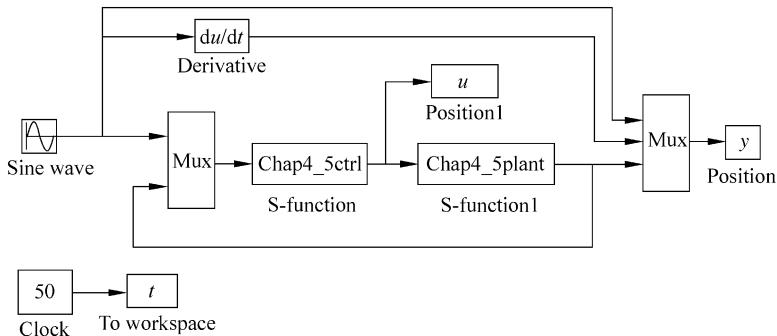
figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,3),'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('ideal position','position tracking');
subplot(212);
plot(t,y(:,2),'r',t,y(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('dyd,dy');
legend('ideal speed','speed tracking');

figure(2);
plot(t,u(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');

```

Programs for Sect. 4.3.4.2

Main Simulink program: chap4_5sim.mdl



Control law program: chap4_5ctrl.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 lambd epc
lambd=5;
epc=0.25;
node=13;
sizes = simsizes;
sizes.NumContStates = node;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = zeros(1,13);
  
```

```

c=2*[-3 -2.5 -2 -1.5 -1 -0.5 0 0.5 1 1.5 2 2.5 3;
      -3 -2.5 -2 -1.5 -1 -0.5 0 0.5 1 1.5 2 2.5 3;
      -3 -2.5 -2 -1.5 -1 -0.5 0 0.5 1 1.5 2 2.5 3;
      -3 -2.5 -2 -1.5 -1 -0.5 0 0.5 1 1.5 2 2.5 3;
      -3 -2.5 -2 -1.5 -1 -0.5 0 0.5 1 1.5 2 2.5 3];
b=3;
str = [];
ts = [];
function sys=mdlDerivatives(t,x,u)
global node c b lambd epc
yd=pi/6*sin(t);
dyd=pi/6*cos(t);
ddyd=-pi/6*sin(t);
x1=u(2);
x2=u(3);
e=x1-yd;
de=x2-dyd;
s=lambd*e+de;
v=-ddyd+lambd*de;
xi=[x1;x2;s;s/epc;v];
h=zeros(13,1);
for j=1:1:13
    h(j)=exp(-norm(xi-c(:,j))^2/(2*b^2));
end
rou=0.005;
Gama=15*eye(13);
W=[x(1) x(2) x(3) x(4) x(5) x(6) x(7) x(8) x(9) x(10) x(11)
x(12) x(13)]';
S=-Gama*(h*s+rou*W);
for i=1:1:node
    sys(i)=S(i);
end
function sys=mdlOutputs(t,x,u)
global node c b lambd epc
yd=pi/6*sin(t);
dyd=pi/6*cos(t);
ddyd=-pi/6*sin(t);
x1=u(2);
x2=u(3);
e=x1-yd;
de=x2-dyd;
s=lambd*e+de;

```

```
v=-ddyd+lambda*de;
xi=[x1;x2;s;s/epc;v];
W=[x(1) x(2) x(3) x(4) x(5) x(6) x(7) x(8) x(9) x(10) x(11)
x(12) x(13)]';
h=zeros(13,1);
for j=1:1:13
    h(j)=exp(-norm(xi-c(:,j))^2/(2*b^2));
end
ut=W'*h;

sys(1)=ut;
Plant program: chap4_5plant.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 = 2;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
ut=u(1);
x1=x(1);
x2=x(2);
a1=0.5*sin(x1)*(1+cos(x1))*x2^2-10*sin(x1)*(1+cos(x1));
a2=0.25*(2+cos(x1))^2;
```

```

alfax=a1/a2;
b=0.25*(2+cos(x1))^2;
betax=1/b;
d1=cos(3*t);
dt=0.1*d1*cos(x1);

sys(1)=x(2);
sys(2)=alfax+betax*ut+dt;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
Plot program: chap4_5plot.m
close all;

figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,3),'k:','linewidth',2);
xlabel('time(s)');ylabel('yd,y');
legend('ideal position','position tracking');
subplot(212);
plot(t,y(:,2),'r',t,y(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('dyd,dy');
legend('ideal speed','speed tracking');

figure(2);
plot(t,u(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');

```

References

1. Wang LX (1997) A course in fuzzy systems and control. Prentice-Hall, New York
2. Huang AC, Chen YC (2004) Adaptive sliding control for single-link flexible joint robot with mismatched uncertainties. IEEE Trans Control Syst Technol 12(5):770–775
3. Ge SS, Hang CC, Zhang T (1999) A direct method for robust adaptive nonlinear control with guaranteed transient performance. Syst Control Lett 37:275–284
4. Ge SS, Hang CC, Lee TH, Zhang T (2001) Stable adaptive neural network control. Kluwer, Boston
5. Krstic M, Kanellakopoulos I, Kokotovic P (1995) Nonlinear and adaptive control design. Wiley, New York

Chapter 5

Neural Network Sliding Mode Control

Abstract This chapter introduces adaptive neural sliding mode control based on RBF neural network approximation, including a simple sliding mode controller and sliding mode control for second-order SISO nonlinear system, the chattering phenomena is eliminated. The closed-loop system stability can be achieved based on the Lyapunov stability.

Keywords Neural network • Sliding mode control • Neural approximation • Lyapunov stability

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 [1, 2]. Stability, reaching condition, and chattering phenomena are known important difficulties [3]. 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.

5.1 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 coresearchers such as Utkins and Itkis. During the last decades, significant interest on SMC has been generated in the control research community.

For linear system,

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{b}u, \mathbf{x} \in R^n, u \in R. \quad (5.1)$$

A sliding variable can be designed as

$$s(\mathbf{x}) = \mathbf{c}^T \mathbf{x} = \sum_{i=1}^n c_i x_i = \sum_{i=1}^{n-1} c_i x_i + x_n \quad (5.2)$$

where \mathbf{x} is state vector, $\mathbf{c} = [c_1 \quad \cdots \quad c_{n-1} \quad 1]^T$.

In sliding mode control, parameters c_1, c_2, \dots, c_{n-1} should be selected so that the polynomial $p^{n-1} + c_{n-1}p^{n-2} + \cdots + c_2p + c_1$ is Hurwitz, where p is Laplace operator.

For example, $n = 2$, $s(\mathbf{x}) = c_1 x_1 + x_2$, to guarantee the polynomial $p + c_1$ Hurwitz, the eigenvalue of $p + c_1 = 0$ should have negative real part, that is, $c_1 > 0$, for example, if we set $c_1 = 10$, then $s(\mathbf{x}) = 10x_1 + x_2$.

For another example, $n = 3$, $s(\mathbf{x}) = c_1 x_1 + c_2 x_2 + x_3$, to guarantee the polynomial $p^2 + c_2 p + c_1$ Hurwitz, the eigenvalue of $p^2 + c_2 p + c_1 = 0$ should have negative real part. For example, we can design $\lambda > 0$ in $(p + \lambda)^2 = 0$, then we can get $p^2 + 2\lambda p + \lambda^2 = 0$. Therefore, we have $c_2 = 2\lambda$, $c_1 = \lambda^2$, for example, if we set $\lambda = 5$, we can get $c_1 = 25$, $c_2 = 10$, then $s(\mathbf{x}) = 25x_1 + 10x_2 + x_3$.

Now we consider a second-order system; 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

$$J\ddot{\theta}(t) = u(t) + dt \quad (5.3)$$

where J is inertia moment, $\theta(t)$ is angle signal, $u(t)$ is control input, dt is disturbance, and $|dt| \leq D$.

Firstly, we design the sliding mode function as

$$s(t) = ce(t) + \dot{e}(t) \quad (5.4)$$

where c must satisfy Hurwitz condition, $c > 0$.

The tracking error and its derivative value are

$$e(t) = \theta(t) - \theta_d(t), \quad \dot{e}(t) = \dot{\theta}(t) - \dot{\theta}_d(t)$$

where $\theta_d(t)$ is ideal position signal.

Design Lyapunov function as

$$V = \frac{1}{2}s^2$$

Therefore, we have

$$\dot{s}(t) = c\dot{e}(t) + \ddot{e}(t) = c\dot{e}(t) + \ddot{\theta}(t) - \ddot{\theta}_d(t) = c\dot{e}(t) + \frac{1}{J}(u + dt) - \ddot{\theta}_d(t) \quad (5.5)$$

and

$$s\dot{s} = s\left(c\dot{e} + \frac{1}{J}(u + dt) - \ddot{\theta}_d\right).$$

Secondly, to guarantee $s\dot{s} < 0$, we design the sliding mode controller as

$$u(t) = J(-c\dot{e} + \ddot{\theta}_d - \eta \operatorname{sgn}(s)) - D \operatorname{sgn}(s). \quad (5.6)$$

Then we get

$$\begin{aligned} s\dot{s} &= s\left(c\dot{e} + \frac{1}{J}(u + dt) - \ddot{\theta}_d\right), \\ s\dot{s} &= -\eta|s| - \frac{D}{J}|s| < 0. \end{aligned}$$

Thus,

$$\dot{V} \leq 0 (\dot{V} = 0 \text{ when } s = 0).$$

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 (5.6), 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.

5.2 Sliding Mode Control Based on RBF for Second-Order SISO Nonlinear System

5.2.1 Problem Description

Consider a second-order nonlinear system as follows:

$$\ddot{\theta} = f(\theta, \dot{\theta}) + g(\theta, \dot{\theta})u + d(t), \quad (5.7)$$

where $f(\cdot)$ and $g(\cdot)$ are all nonlinear functions, $u \in \mathbb{R}$ and $y \in \mathbb{R}$ are the control input and output respectively, and $d(t)$ is outer disturbance, $|d(t)| \leq D$.

Let the desired output be θ_d , and denote

$$e = \theta_d - \theta.$$

Design sliding mode function as

$$s = \dot{e} + ce, \quad (5.8)$$

where $c > 0$, then

$$\dot{s} = \ddot{e} + c\dot{e} = \ddot{\theta}_d - \ddot{\theta} + c\dot{e} = \ddot{\theta}_d - f - gu - d(t) + c\dot{e}. \quad (5.9)$$

If f and g are known, we can design control law as

$$u = \frac{1}{g} [-f + \ddot{\theta}_d + c\dot{e} + \eta \operatorname{sgn}(s)]. \quad (5.10)$$

Using (5.4), (5.3) becomes

$$\dot{s} = \ddot{e} + c\dot{e} = \ddot{\theta}_d - \ddot{\theta} + c\dot{e} = \ddot{\theta}_d - f - gu - d(t) + c\dot{e} = -\eta \operatorname{sgn}(s) - d(t).$$

If we choose $\eta \geq D$, then we have

$$s\dot{s} = -\eta|s| - s \cdot d(t) \leq 0.$$

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 uncertain item $f(\cdot)$.

5.2.2 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

$$h_j = \exp\left(\frac{\|\mathbf{x} - \mathbf{c}_j\|^2}{2b_j^2}\right)$$

$$f = \mathbf{W}^{*\top} \mathbf{h}(\mathbf{x}) + \varepsilon$$

where \mathbf{x} is input of the network, i is input number of the network, j is the number of hidden layer nodes in the network, $\mathbf{h} = [h_j]^T$ is the output of Gaussian function, \mathbf{W}^* is the ideal neural network weights, ε is approximation error of the neural network, and $\varepsilon \leq \varepsilon_N$, f is the output value of the network.

The network input is selected as $\mathbf{x} = [e \quad \dot{e}]^T$, and the output of RBF is

$$\hat{f}(\mathbf{x}) = \hat{\mathbf{W}}^T \mathbf{h}(\mathbf{x}) \quad (5.11)$$

where $\mathbf{h}(\mathbf{x})$ is the Gaussian function of RBF neural network.

Then the control input (5.10) can be written as

$$u = \frac{1}{g} [-\hat{f} + \ddot{\theta}_d + c\dot{e} + \eta \operatorname{sgn}(s)]. \quad (5.12)$$

Submitting (5.9), we have

$$\begin{aligned} \dot{s} &= \ddot{\theta}_d - f - gu - d(t) + c\dot{e} = \ddot{\theta}_d - f - [-\hat{f} + \ddot{\theta}_d + c\dot{e} + \eta \operatorname{sgn}(s)] \\ &\quad - d(t) + c\dot{e} \\ &= -f + \hat{f} - \eta \operatorname{sgn}(s) - d(t) = -\tilde{f} - d(t) - \eta \operatorname{sgn}(s) \end{aligned} \quad (5.13)$$

where

$$\tilde{f} = f - \hat{f} = \mathbf{W}^{*\top} \mathbf{h}(\mathbf{x}) + \varepsilon - \hat{\mathbf{W}}^T \mathbf{h}(\mathbf{x}) = \tilde{\mathbf{W}}^T \mathbf{h}(\mathbf{x}) + \varepsilon \quad (5.14)$$

and $\tilde{\mathbf{W}} = \mathbf{W}^* - \hat{\mathbf{W}}$.

Define the Lyapunov function as

$$L = \frac{1}{2} s^2 + \frac{1}{2} \gamma \tilde{\mathbf{W}}^T \tilde{\mathbf{W}}$$

where $\gamma > 0$.

Derivative L , and from (5.12) and (5.13), we have

$$\begin{aligned}\dot{L} &= s\dot{s} + \gamma\tilde{\mathbf{W}}^T\dot{\tilde{\mathbf{W}}} = s(-\tilde{f} - d(t) - \eta \operatorname{sgn}(s)) - \gamma\tilde{\mathbf{W}}^T\dot{\tilde{\mathbf{W}}} \\ &= s(-\tilde{\mathbf{W}}^T\mathbf{h}(\mathbf{x}) - \varepsilon - d(t) - \eta \operatorname{sgn}(s)) - \gamma\tilde{\mathbf{W}}^T\dot{\tilde{\mathbf{W}}} \\ &= -\tilde{\mathbf{W}}^T(\mathbf{sh}(\mathbf{x}) + \gamma\dot{\tilde{\mathbf{W}}}) - s(\varepsilon + d(t) + \eta \operatorname{sgn}(s))\end{aligned}$$

Let adaptive law as

$$\dot{\tilde{\mathbf{W}}} = -\frac{1}{\gamma}s\mathbf{h}(\mathbf{x}). \quad (5.15)$$

Then

$$\dot{L} = -s(\varepsilon + d(t) + \eta \operatorname{sgn}(s)) = -s(\varepsilon + d(t)) - \eta|s|.$$

Due to the approximation error ε is limited and sufficiently small, we can design $\eta \geq \varepsilon_N + D$, then we can obtain approximately $\dot{L} \leq 0$.

5.2.3 Simulation Example

Consider the following single-rank inverted pendulum:

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{g \sin x_1 - mlx_2^2 \cos x_1 \sin x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))} + \frac{\cos x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))} u\end{aligned}$$

where x_1 and x_2 are, respectively, swing angle and swing rate. $g = 9.8 \text{ m/s}^2$, $m_c = 1 \text{ kg}$ is the vehicle mass, and $m = 0.1 \text{ kg}$ is the mass of pendulum. $l = 0.5 \text{ m}$ is one half of the pendulum length, and u is the control input.

Let $x_1 = \theta$, the desired trajectory is $\theta_d(t) = 0.1 \sin(t)$, and the initial state of the plant is $[\pi/60, 0]$. We adapt control law (5.12) and adaptive law (5.15), and we choose $c = 15$, $\eta = 0.1$ and adaptive parameter $\gamma = 0.05$.

The structure of RBF is chosen as 2-5-1, c_i and b_i are designed as $[-1.0 \ -0.5 \ 0 \ 0.5 \ 1.0]$ and $b_j = 0.50$, and the initial value of RBF weight value is set as 0.10.

The curves of position tracking and uncertainty approximation are shown from Figs. 5.1, 5.2, and 5.3.

The Simulink program of this example is chap5_1sim.mdl; the Matlab programs of the example are given in the Appendix.

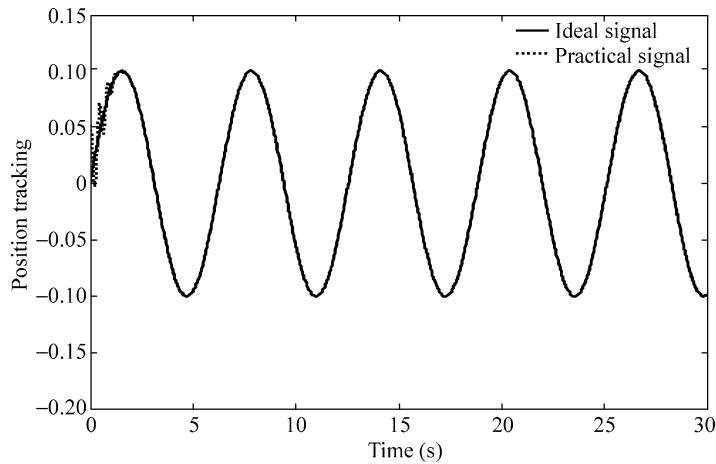


Fig. 5.1 Position tracking

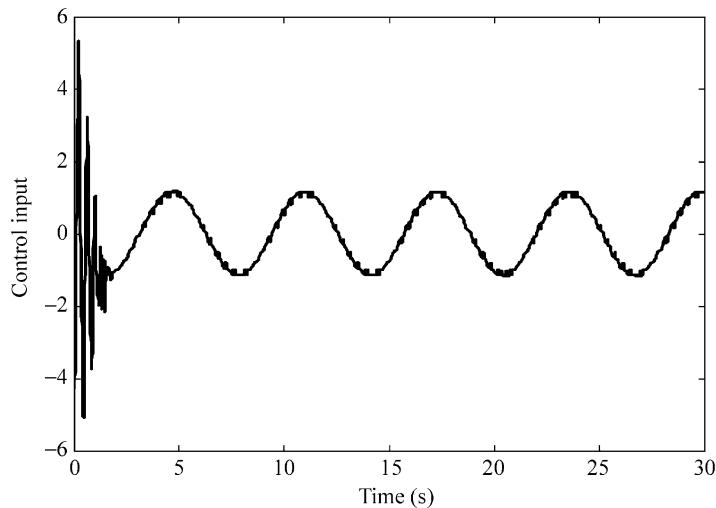


Fig. 5.2 Control input

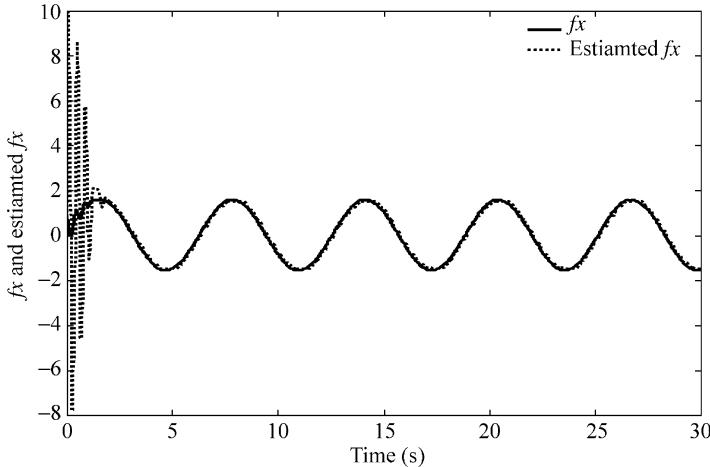


Fig. 5.3 $f(x)$ and $\hat{f}(x)$

5.3 Sliding Mode Control Based on RBF for Unknown $f(\cdot)$ and $g(\cdot)$

5.3.1 Introduction

Consider a second-order nonlinear system as (5.7), and assume $f(\cdot)$ and $g(\cdot)$ are all unknown nonlinear functions, $u \in R$ and $y \in R$ are the control input and output, respectively, and $d(t)$ is outer disturbance, $|d(t)| \leq D$.

Similarly as Sect. 5.2, let the desired output be θ_d , denote $e = \theta_d - \theta$, and design sliding mode function as $s = \dot{e} + ce$, where $c > 0$.

In this control system, we use two RBF networks to approximate $f(\cdot)$ and $g(\cdot)$, respectively. Figure 5.4 shows the closed-loop neural-based adaptive control scheme.

The algorithm of RBF network is

$$h_j = \exp\left(\frac{\|\mathbf{x} - \mathbf{c}_j\|^2}{2b_j^2}\right)$$

$$f(\cdot) = \mathbf{W}^{*\top} \mathbf{h}_f(x) + \varepsilon_f, \quad g(\cdot) = \mathbf{V}^{*\top} \mathbf{h}_g(x) + \varepsilon_g$$

where \mathbf{x} is the input of RBF neural network, i is the input number of the network, j is the number of hidden layer nodes in the network, $\mathbf{h} = [h_j]^T$ is the output of Gaussian function, \mathbf{W}^* and \mathbf{V}^* are the ideal neural network weights, and ε_f and ε_g are the approximation error of the neural network, $|\varepsilon_f| \leq \varepsilon_{Mf}$, $|\varepsilon_g| \leq \varepsilon_{Mg}$. The items $f(\cdot)$ and $g(\cdot)$ are the ideal output value of the network, respectively.

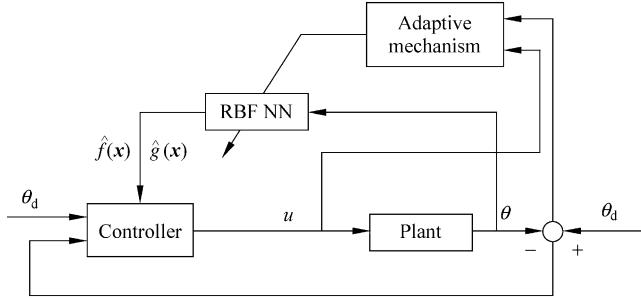


Fig. 5.4 Block diagram of the control scheme

The network input is selected as $\mathbf{x} = [x_1 \quad x_2]^T$, and the output of RBF is

$$\hat{f}(x) = \tilde{\mathbf{W}}^T \mathbf{h}_f(x), \quad \hat{g}(x) = \tilde{\mathbf{V}}^T \mathbf{h}_g(x) \quad (5.16)$$

where $\mathbf{h}_f(x)$ and $\mathbf{h}_g(x)$ are the Gaussian function of RBF neural network.

Then the control input (5.10) can be written as

$$u = \frac{1}{\hat{g}(x)} [-\hat{f}(x) + \ddot{\theta}_d + c\dot{e} + \eta \operatorname{sgn}(s)] \quad (5.17)$$

where $\eta \geq D$.

Submitting (5.9), we have

$$\begin{aligned}
 \dot{s} &= \dot{e} + c\dot{e} = \ddot{\theta}_d - \ddot{\theta} + c\dot{e} = \ddot{\theta}_d - f - gu - d(t) + c\dot{e} \\
 &= \ddot{\theta}_d - f - \hat{g}u + (\hat{g} - g)u - d(t) + c\dot{e} \\
 &= \ddot{\theta}_d - f - \hat{g} \frac{1}{\hat{g}(x)} [-\hat{f}(x) + \ddot{\theta}_d + c\dot{e} + \eta \operatorname{sgn}(s)] + (\hat{g} - g)u - d(t) + c\dot{e} \\
 &= (\hat{f} - f) - \eta \operatorname{sgn}(s) + (\hat{g} - g)u - d(t) = \tilde{f} - \eta \operatorname{sgn}(s) + \tilde{g}u - d(t) \\
 &= \tilde{\mathbf{W}}^T \boldsymbol{\phi}_f(\mathbf{x}) - \varepsilon_f - \eta \operatorname{sgn}(s) + (\tilde{\mathbf{V}}^T \boldsymbol{\phi}_g(\mathbf{x}) - \varepsilon_g)u - d(t)
 \end{aligned} \quad (5.18)$$

where $\tilde{\mathbf{W}} = \mathbf{W}^* - \hat{\mathbf{W}}$, $\tilde{\mathbf{V}} = \mathbf{V}^* - \hat{\mathbf{V}}$, and

$$\begin{aligned}
 \tilde{f} &= \hat{f} - f = \hat{\mathbf{W}}^T \mathbf{h}_f(\mathbf{x}) - \mathbf{W}^{*T} \mathbf{h}_f(\mathbf{x}) - \varepsilon_f = \tilde{\mathbf{W}}^T \mathbf{h}_f(\mathbf{x}) - \varepsilon_f \\
 \tilde{g} &= \hat{g} - g = \hat{\mathbf{V}}^T \mathbf{h}_g(\mathbf{x}) - \mathbf{V}^{*T} \mathbf{h}_g(\mathbf{x}) - \varepsilon_g = \tilde{\mathbf{V}}^T \mathbf{h}_g(\mathbf{x}) - \varepsilon_g
 \end{aligned} \quad (5.19)$$

Define the Lyapunov function as

$$L = \frac{1}{2}s^2 + \frac{1}{2\gamma_1} \tilde{\mathbf{W}}^T \tilde{\mathbf{W}} + \frac{1}{2\gamma_2} \tilde{\mathbf{V}}^T \tilde{\mathbf{V}}$$

where $\gamma_1 > 0$, $\gamma_2 > 0$.

Derivative L , and from (5.18), we have

$$\begin{aligned}\dot{L} &= s\dot{s} + \frac{1}{\gamma_1} \tilde{\mathbf{W}}^T \dot{\tilde{\mathbf{W}}} + \frac{1}{\gamma_2} \tilde{\mathbf{V}}^T \dot{\tilde{\mathbf{V}}} \\ &= s \left(\tilde{\mathbf{W}}^T \mathbf{h}_f(\mathbf{x}) - \varepsilon_f - \eta \operatorname{sgn}(s) + \left(\tilde{\mathbf{V}}^T \mathbf{h}_g(\mathbf{x}) - \varepsilon_g \right) u - d(t) \right) \\ &\quad - \frac{1}{\gamma_1} \tilde{\mathbf{W}}^T \dot{\tilde{\mathbf{W}}} - \frac{1}{\gamma_2} \tilde{\mathbf{V}}^T \dot{\tilde{\mathbf{V}}} \\ &= \tilde{\mathbf{W}}^T \left(s \mathbf{h}_f(\mathbf{x}) - \frac{1}{\gamma_1} \dot{\tilde{\mathbf{W}}} \right) + \tilde{\mathbf{V}}^T \left(s \mathbf{h}_g(\mathbf{x}) u - \frac{1}{\gamma_2} \dot{\tilde{\mathbf{V}}} \right) \\ &\quad + s(-\varepsilon_f - \eta \operatorname{sgn}(s) - \varepsilon_g u - d(t))\end{aligned}$$

Let adaptive law as

$$\dot{\tilde{\mathbf{W}}} = -\gamma_1 s \mathbf{h}_f(\mathbf{x}) \quad (5.20)$$

$$\dot{\tilde{\mathbf{V}}} = -\gamma_2 s \mathbf{h}_g(\mathbf{x}) u \quad (5.21)$$

Then

$$\begin{aligned}\dot{L} &= s(-\varepsilon_f - \eta \operatorname{sgn}(s) - \varepsilon_g u - d(t)) \\ &= (-\varepsilon_f - \varepsilon_g u - d(t))s - \eta |s|\end{aligned}$$

Due to the approximation error ε_f and ε_g is limited and sufficiently small, we can design $\eta \geq |\varepsilon_f + \varepsilon_g u + d(t)|$, then we can obtain approximately $\dot{L} \leq 0$.

5.3.2 Simulation Example

Consider the following single-rank inverted pendulum

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

where $f(\mathbf{x}) = \frac{g \sin x_1 - mlx_2^2 \cos x_1 \sin x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}$, $g(\mathbf{x}) = \frac{\cos x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}$, x_1 and x_2 are, respectively, swing angle and swing rate, $g = 9.8 \text{ m/s}^2$, $m_c = 1 \text{ kg}$ is the vehicle

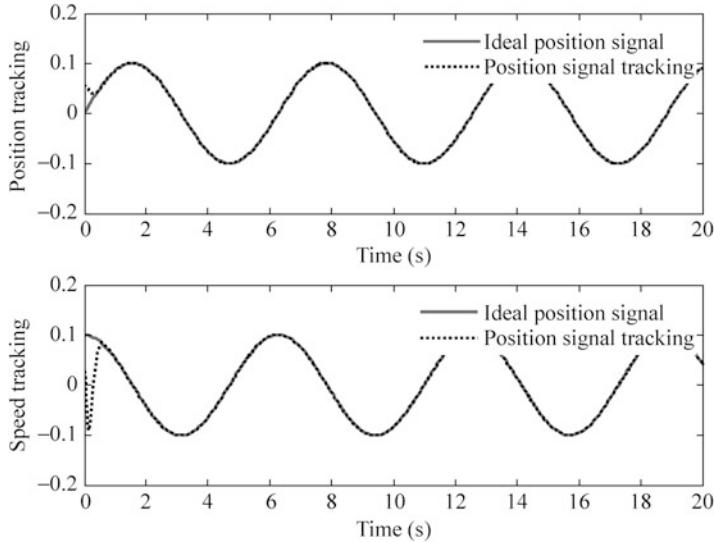


Fig. 5.5 Position and speed tracking

mass, and $m = 0.1 \text{ kg}$ is the mass of pendulum. $l = 0.5 \text{ m}$ is one half of the pendulum length, and u is the control input.

Let $x_1 = \theta$, the desired trajectory is $\theta_d(t) = 0.1 \sin(t)$, and the initial state of the plant is $[\pi/60, 0]$.

For each Gaussian function, the parameters of c_i and b_i are designed as $[-1.0 \ -0.5 \ 0 \ 0.5 \ 1.0]$ and 5.0. The initial weight value is chosen as 0.10. We use control law (5.17) and adaptive laws (5.20) and (5.21); the parameters are chosen as $\gamma_1 = 10$, $\gamma_2 = 1.0$, and $c = 5.0$.

The results are shown from Figs. 5.5, 5.6, and 5.7. The variation of $\hat{f}(\cdot)$ and $\hat{g}(\cdot)$ do not converge to $f(\cdot)$ and $g(\cdot)$. This is due to the fact that the desired trajectory is not persistently exciting and the tracking error performance can be achieved by many possible values of $\hat{f}(\cdot)$ and $\hat{g}(\cdot)$, besides the true $f(\cdot)$ and $g(\cdot)$, which has been explained in Sect. 1.5, and this occurs quite often in real-world application.

The Simulink program of this example is chap5_2sim.mdl; the Matlab programs of the example are given in the [Appendix](#).

Appendix

Programs for Sect. 5.2.3

Main Simulink program: chap5_1sim.mdl

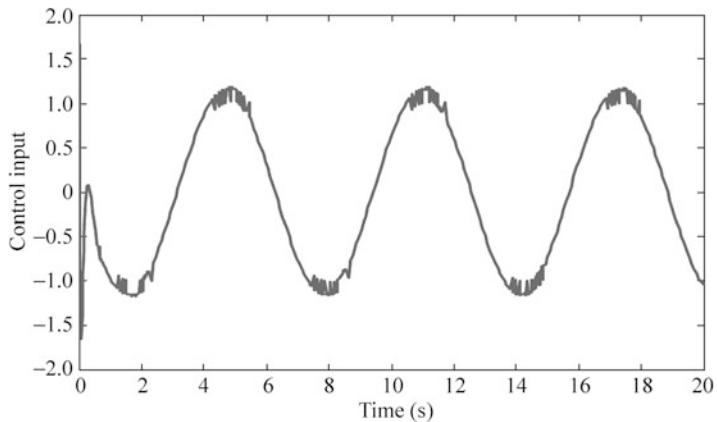


Fig. 5.6 Control input

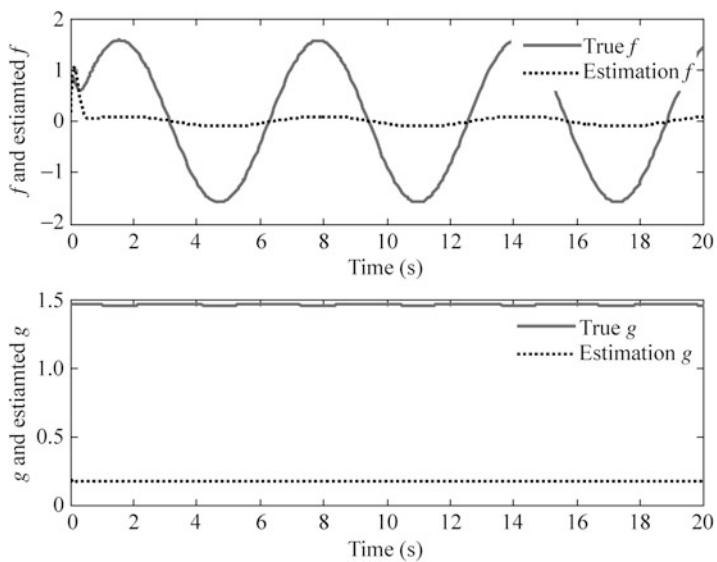
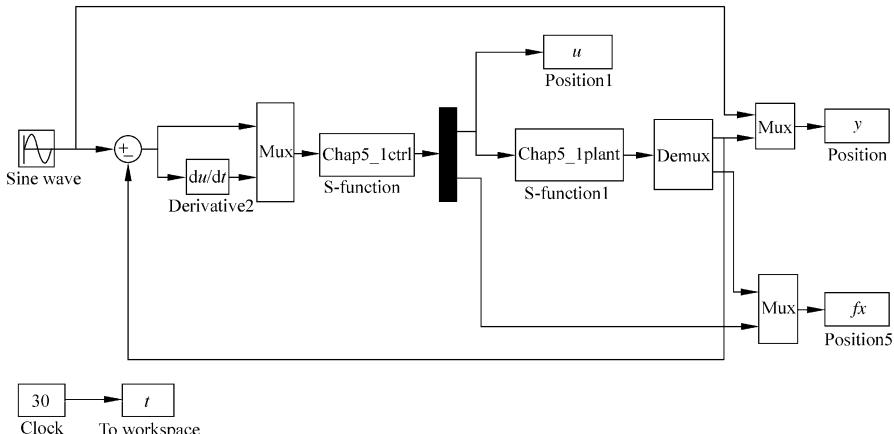


Fig. 5.7 Approximation of $f(\cdot)$ and $g(\cdot)$



Control law program: chap5_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 bjc c
sizes = simsizes;
sizes.NumContStates = 5;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = 0*ones(1,5);
str = [];
ts = [];
cij=0.10*[ -1 -0.5 0 0.5 1;
            -1 -0.5 0 0.5 1];
bj=5.0;

```

```

c=15;
function sys=mdlDerivatives(t,x,u)
global cij bj c
e=u(1);
de=u(2);
s=c*e+de;

xi=[e;de];
h=zeros(5,1);
for j=1:1:5
    h(j)=exp(-norm(xi-cij(:,j))^2/(2*bj^2));
end
gama=0.015;
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
e=u(1);
de=u(2);
thd=0.1*sin(t);
dthd=0.1*cos(t);
ddthd=-0.1*sin(t);
x1=thd-e;

s=c*e+de;
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-cij(:,j))^2/(2*bj^2));
end
fn=W'*h;

g=9.8;mc=1.0;m=0.1;l=0.5;
S=1*(4/3-m*(cos(x1))^2/(mc+m));
gx=cos(x1)/(mc+m);
gx=gx/S;

if t<=1.5
    xite=1.0;
else
    xite=0.10;
end
ut=1/gx*(-fn+ddthd+c*de+xite*sign(s));
sys(1)=ut;

```

```
sys(2)=fn;
Plant program: chap5_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 = 2;
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=1*(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;
%%%%%
dt=0*10*sin(t);
%%%%%
sys(1)=x(2);
sys(2)=fx+gx*u+dt;
function sys=mdlOutputs(t,x,u)
g=9.8;
mc=1.0;
m=0.1;
l=0.5;
```

```

S=1*(4/3-m*(cos(x(1)))^2/(mc+m));
fx=g*sin(x(1))-m*1*x(2)^2*cos(x(1))*sin(x(1))/(mc+m);
fx=fx/S;

sys(1)=x(1);
sys(2)=fx;
Plot program: chap5_1plot.m
close all;

figure(1);
plot(t,y(:,1),'k',t,y(:,2),'r','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal signal','practical signal');

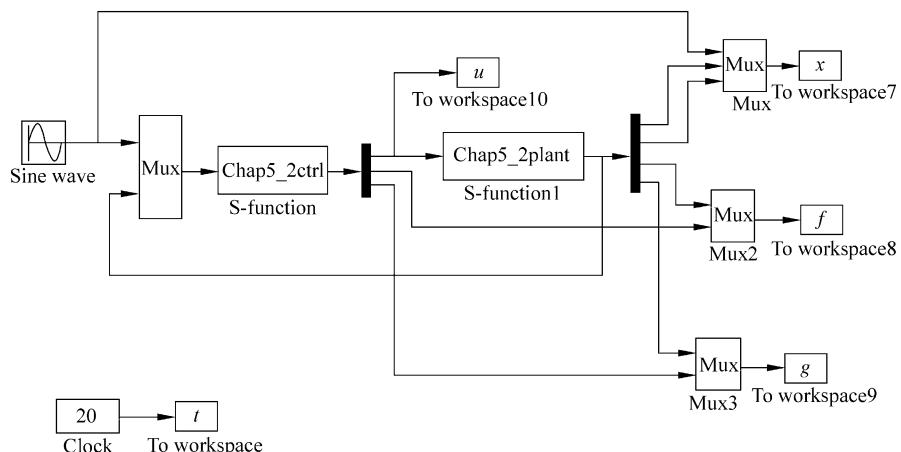
figure(2);
plot(t,u(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('Control input');

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');

```

Programs for Sect. 5.3.2

Main Simulink program: chap5_2sim.mdl



Control law program: chap5_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 xite cij bj h c
sizes = simsizes;
sizes.NumContStates = 10;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 5;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = 0.1*ones(10,1);
str = [];
ts = [];
cij=[-1 -0.5 0 0.5 1;
      -1 -0.5 0 0.5 1];
bj=5;
h=[0,0,0,0,0];
c=5;
xite=0.01;
function sys=mdlDerivatives(t,x,u)
global xite cij bj h c
thd=u(1);
dthd=0.1*cos(t);
ddthd=-0.1*sin(t);

x1=u(2);
x2=u(3);
e=thd-x1;
de=dthd-x2;
s=c*e+de;
```

```
xi=[x1;x2];
for j=1:1:5
    h(j)=exp(-norm(xi-cij(:,j))^2/(2*bj^2));
end
for i=1:1:5
    wf(i,1)=x(i);
end
for i=1:1:5
    wg(i,1)=x(i+5);
end
fxn=wf'*h';
gxn=wg'*h'+0.01;
ut=1/gxn*(-fxn+ddthd+xite*sign(s)+c*de);
gama1=10;gama2=1.0;
S1=-gama1*s*h;
S2=-gama2*s*h*ut;
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 xite cij bj h c
thd=u(1);
dthd=0.1*cos(t);
ddthd=-0.1*sin(t);
x1=u(2);
x2=u(3);
e=thd-x1;
de=dthd-x2;
s=c*e+de;
for i=1:1:5
    wf(i,1)=x(i);
end
for i=1:1:5
    wg(i,1)=x(i+5);
end
xi=[x1;x2];
for j=1:1:5
    h(j)=exp(-norm(xi-cij(:,j))^2/(2*bj^2));
end
```

```
fxn=wf'*h';
gxn=wg'*h'+0.01;
ut=1/gxn*(-fxn+ddthd+xite*sign(s)+c*de);
sys(1)=ut;
sys(2)=fxn;
sys(3)=gxn;
Plant program: chap5_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 = 4;
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=1*(4/3-m*(cos(x(1)))^2/(mc+m));
fx=g*sin(x(1))-m*1*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;
sys(4)=gx;

Plot program: chap5_2plot.m
close all;

figure(1);
subplot(211);
plot(t,x(:,1),'r',t,x(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('Ideal position signal','Position signal tracking');

subplot(212);
plot(t,0.1*cos(t),'r',t,x(:,3),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('Ideal speed signal','Speed signal tracking');

figure(2);
plot(t,u(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');

figure(3);
subplot(211);
plot(t,f(:,1),'r',t,f(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('f and estiamted f');
legend('True f','Estimation f');

subplot(212);
plot(t,g(:,1),'r',t,g(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('g and estimated g');
legend('True g','Estimation g');

```

References

1. Beyhan S, Alc M (2009) A new RBF network modeling based sliding mode control of nonlinear systems. In: Proceedings of the international multi-conference on computer science and information technology, Mragowo, Poland, IEEE, pp 11–16
2. Tsai CH, Chung HY, Yu FM (2004) Neuro-sliding mode control with its applications to seesaw systems. IEEE Trans Neural Netw 15(1):124–134
3. Edwards C, Spurgeon S (1998) Sliding mode control: theory and applications. Taylor & Francis, London

Chapter 6

Adaptive RBF Control Based on Global Approximation

Abstract This chapter introduces three kinds of adaptive neural mode control laws for n-link manipulators based on RBF, including adaptive neural network control law, adaptive neural network control law with sliding mode robust term, and adaptive neural network control law with HJI. The closed-loop system stability can be achieved based on the Lyapunov stability.

Keywords RBF neural network • Adaptive control • Global approximation • Robotic manipulators

Recently, increasing attention has been paid to the use of neural networks in robot control. Previous use of neural networks to improve robot path following performance has concentrated on replacing either the entire control system or the feed-forward controller and/or prefilter with neural networks [1, 2]. Such research work was based on the desire to obtain the benefits of model-based control without a priori knowledge of system dynamics or without the computational burden of classical dynamic equations.

However, in many cases, the approximate dynamic model of the robot manipulators can be found a priori. Actually, the feasibility of model-based control has been demonstrated [3, 4]. Therefore, a priori knowledge of the robot dynamic models should be appropriately used rather than totally discarded. Recently, a direct adaptive control algorithm using neural networks was proposed in [5]. Their method is based on the BP feed-forward networks and the reinforcement learning.

In this chapter, we introduced three typical examples of neural network controller design, analysis, and simulation.

6.1 Adaptive Control with RBF Neural Network Compensation for Robotic Manipulators

In this section, in reference to paper [6], a robot tracking control scheme is introduced. This scheme takes advantage of the computed torque methods and incorporates a compensating controller to achieve high tracking performance. The compensating controller is based on a radial basis function (RBF) neural network, which is trained on-line to identify the robot modeling error. A main feature of the proposed scheme is that the resulting closed-loop control system is guaranteed to be stable. Another important property of the proposed compensating controller is that the neural network-based correcting controller is an add-on device which can be added on many existing robot control systems.

6.1.1 Problem Description

Consider dynamic equation of n -link manipulator as

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} + \mathbf{d} \quad (6.1)$$

where $\mathbf{M}(\mathbf{q})$ is an $n \times n$ inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is an $n \times n$ matrix containing the centrifugal and Coriolis terms, $\mathbf{G}(\mathbf{q})$ is an $n \times 1$ vector containing gravitational forces and torques, \mathbf{q} is generalized joint coordinates, $\boldsymbol{\tau}$ is joint torques, and \mathbf{d} denotes disturbances.

However, in practice, the perfect robot model could be difficult to obtain, and external disturbances are always present in practice. Usually, only a nominal model of the robot could be obtained. It is supposed that the nominal model of the robot is denoted by $\mathbf{M}_0(\mathbf{q})$, $\mathbf{C}_0(\mathbf{q}, \dot{\mathbf{q}})$, $\mathbf{G}_0(\mathbf{q})$, and we assume $\Delta\mathbf{M} = \mathbf{M}_0 - \mathbf{M}$, $\Delta\mathbf{C} = \mathbf{C}_0 - \mathbf{C}$, $\Delta\mathbf{G} = \mathbf{G}_0 - \mathbf{G}$, then from (6.1), we have

$$(\mathbf{M}_0(\mathbf{q}) - \Delta\mathbf{M})\ddot{\mathbf{q}} + (\mathbf{C}_0(\mathbf{q}, \dot{\mathbf{q}}) - \Delta\mathbf{C})\dot{\mathbf{q}} + \mathbf{G}_0(\mathbf{q}) - \Delta\mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} + \mathbf{d}$$

Hence,

$$\mathbf{M}_0(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}_0(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}_0(\mathbf{q}) = \boldsymbol{\tau} + \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$$

where $\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \Delta\mathbf{M}\ddot{\mathbf{q}} + \Delta\mathbf{C}\dot{\mathbf{q}} + \Delta\mathbf{G} + \mathbf{d}$.

Therefore, if the nominal model is used for the design of the computed torque controller, and if $\mathbf{f}(\cdot)$ is known, we can design control law as

$$\boldsymbol{\tau} = \mathbf{M}_0(\mathbf{q})(\ddot{\mathbf{q}}_d - \mathbf{k}_v \dot{\mathbf{e}} - \mathbf{k}_p \mathbf{e}) + \mathbf{C}_0(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}_0(\mathbf{q}) - \mathbf{f}(\cdot) \quad (6.2)$$

where $\mathbf{k}_p = \begin{bmatrix} \alpha^2 & 0 \\ 0 & \alpha^2 \end{bmatrix}$, $\mathbf{k}_v = \begin{bmatrix} 2\alpha & 0 \\ 0 & 2\alpha \end{bmatrix}$, $\alpha > 0$.

Submitting the term (6.2) into (6.1), we can get the closed loop system as

$$\ddot{\mathbf{e}} + \mathbf{k}_v \dot{\mathbf{e}} + \mathbf{k}_p \mathbf{e} = 0 \quad (6.3)$$

where \mathbf{q}_d is ideal angle, $\mathbf{e} = \mathbf{q} - \mathbf{q}_d$, $\dot{\mathbf{e}} = \dot{\mathbf{q}} - \dot{\mathbf{q}}_d$.

The goal is to design a stable robust controller based on nominal modeling information.

In practical engineering, the term $f(\cdot)$ is always unknown. So we must estimate $f(\cdot)$ and compensate it. In this section, we use RBF to approximate $f(\cdot)$ and compensate it.

6.1.2 RBF Approximation

The algorithm of RBF is

$$h_i = g\left(\|\mathbf{x} - \mathbf{c}_i\|^2/b_i^2\right), \quad i = 1, 2, \dots, n \quad (6.4)$$

$$\mathbf{y} = \mathbf{w}^T \mathbf{h}(\mathbf{x}) \quad (6.5)$$

where \mathbf{x} is input vector, \mathbf{y} is output, $\mathbf{h} = [h_1, h_2, \dots, h_n]^T$ is the output of Gaussian function, and \mathbf{w} is neural network weight value.

Given a very small positive constant ε_0 and a continuous function $f(\cdot)$, there exists a weight vector \mathbf{w}^* such that the output $\hat{f}(\cdot)$ of RBF satisfies

$$\max \left\| f(\cdot) - \hat{f}^*(\cdot) \right\| \leq \varepsilon_0 \quad (6.6)$$

where $\mathbf{w}^* = \arg \min_{\theta \in \beta(M_\theta)} \left\{ \sup_{x \in \phi(M_x)} \|f(\cdot) - \hat{f}(\cdot)\| \right\}$, \mathbf{w}^* is $n \times n$ matrix, denote the optimal weight values for $f(\cdot)$ approximation.

Define the approximation error as

$$\eta = f(\cdot) - \hat{f}^*(\cdot) \quad (6.7)$$

Assume the modeling error η is bounded by a finite constant as

$$\eta_0 = \sup \left\| f(\cdot) - \hat{f}^*(\cdot) \right\| \quad (6.8)$$

where $\hat{f}^*(\cdot) = \mathbf{w}^{*T} \mathbf{h}(\mathbf{x})$.

6.1.3 RBF Controller and Adaptive Law Design and Analysis

For the system (6.1), the following controller was proposed as [6]

$$\boldsymbol{\tau} = \mathbf{M}_0(\mathbf{q})(\ddot{\mathbf{q}}_d - \mathbf{k}_v \dot{\mathbf{e}} - \mathbf{k}_p \mathbf{e}) + \mathbf{C}_0(\mathbf{q}, \dot{\mathbf{q}}) \dot{\mathbf{q}} + \mathbf{G}_0(\mathbf{q}) - \hat{\mathbf{f}}(\cdot) \quad (6.9)$$

where $\hat{\mathbf{w}}$ is the estimation value of \mathbf{w}^* , $\|\mathbf{w}^*\|_F \leq w_{\max}$, and $\hat{\mathbf{f}}(\cdot) = \hat{\mathbf{w}}^T \mathbf{h}(\mathbf{x})$.

Submitting (6.9) into (6.1), we have

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \mathbf{M}_0(\mathbf{q})(\ddot{\mathbf{q}}_d - \mathbf{k}_v \dot{\mathbf{e}} - \mathbf{k}_p \mathbf{e}) + \mathbf{C}_0(\mathbf{q}, \dot{\mathbf{q}}) \dot{\mathbf{q}} + \mathbf{G}_0(\mathbf{q}) - \hat{\mathbf{f}}(\cdot) + \mathbf{d}$$

Subtracting the right and left sides of above equation with $\mathbf{M}_0(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}_0(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}_0(\mathbf{q})$, we have

$$\begin{aligned} & \Delta \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \Delta \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \Delta \mathbf{G}(\mathbf{q}) + \mathbf{d} \\ &= \mathbf{M}_0(\mathbf{q})\ddot{\mathbf{q}} - \mathbf{M}_0(\mathbf{q})(\ddot{\mathbf{q}}_d - \mathbf{k}_v \dot{\mathbf{e}} - \mathbf{k}_p \mathbf{e}) + \hat{\mathbf{f}}(\cdot) \\ &= \mathbf{M}_0(\mathbf{q})(\ddot{\mathbf{e}} + \mathbf{k}_v \dot{\mathbf{e}} + \mathbf{k}_p \mathbf{e}) + \hat{\mathbf{f}}(\cdot) \end{aligned}$$

Thus,

$$\ddot{\mathbf{e}} + \mathbf{k}_v \dot{\mathbf{e}} + \mathbf{k}_p \mathbf{e} + \mathbf{M}_0^{-1}(\mathbf{q})\hat{\mathbf{f}}(\cdot) = \mathbf{M}_0^{-1}(\mathbf{q})(\Delta \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \Delta \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \Delta \mathbf{G}(\mathbf{q}) + \mathbf{d})$$

Then, we can get

$$\ddot{\mathbf{e}} + \mathbf{k}_v \dot{\mathbf{e}} + \mathbf{k}_p \mathbf{e} = \mathbf{M}_0^{-1}(\mathbf{q})(\mathbf{f}(\cdot) - \hat{\mathbf{f}}(\cdot))$$

Choose $\mathbf{x} = (\mathbf{e} \quad \dot{\mathbf{e}})^T$, then, above equation becomes

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\{\mathbf{f}(\cdot) - \hat{\mathbf{f}}(\cdot)\}$$

where $\mathbf{A} = \begin{pmatrix} 0 & \mathbf{I} \\ -\mathbf{k}_p & -\mathbf{k}_v \end{pmatrix}$, $\mathbf{B} = \begin{pmatrix} 0 \\ \mathbf{M}_0^{-1}(\mathbf{q}) \end{pmatrix}$.

Since

$$\mathbf{f}(\cdot) - \hat{\mathbf{f}}(\cdot) = \mathbf{f}(\cdot) - \hat{\mathbf{f}}^*(\cdot) + \hat{\mathbf{f}}^*(\cdot) - \hat{\mathbf{f}}(\cdot) = \boldsymbol{\eta} + \mathbf{w}^{*T} \mathbf{h} - \hat{\mathbf{w}}^T \mathbf{h} = \boldsymbol{\eta} - \tilde{\mathbf{w}}^T \mathbf{h}$$

where $\tilde{\mathbf{w}} = \hat{\mathbf{w}} - \mathbf{w}^*$, $\boldsymbol{\eta} = \mathbf{f}(\cdot) - \hat{\mathbf{f}}^*(\cdot)$,
then,

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}(\boldsymbol{\eta} - \tilde{\mathbf{w}}^T \mathbf{h})$$

In paper [6], the stability of the closed control system with the controller (6.9) was given by defining Lyapunov function as

$$V = \frac{1}{2} \mathbf{x}^T \mathbf{P} \mathbf{x} + \frac{1}{2\gamma} \|\tilde{\mathbf{w}}\|^2$$

where $\gamma > 0$.

The matrix \mathbf{P} is symmetric and positive definite and satisfies the following Lyapunov equation

$$\mathbf{P}\mathbf{A} + \mathbf{A}^T \mathbf{P} = -\mathbf{Q} \quad (6.10)$$

where $\mathbf{Q} \geq 0$.

Define

$$\|\mathbf{R}\|^2 = \sum_{i,j} |r_{ij}|^2 = \text{tr}(\mathbf{R}\mathbf{R}^T) = \text{tr}(\mathbf{R}^T \mathbf{R})$$

where $\text{tr}(\cdot)$ is the trace of matrix; then,

$$\|\tilde{\mathbf{w}}\|^2 = \text{tr}(\tilde{\mathbf{w}}^T \tilde{\mathbf{w}})$$

The derivative of V is

$$\begin{aligned} \dot{V} &= \frac{1}{2} [\mathbf{x}^T \mathbf{P} \dot{\mathbf{x}} + \dot{\mathbf{x}}^T \mathbf{P} \mathbf{x}] + \frac{1}{\gamma} \text{tr}(\dot{\tilde{\mathbf{w}}}^T \tilde{\mathbf{w}}) \\ &= \frac{1}{2} [\mathbf{x}^T \mathbf{P} (\mathbf{A}\mathbf{x} + \mathbf{B}(\boldsymbol{\eta} - \tilde{\mathbf{w}}^T \mathbf{h})) + (\mathbf{x}^T \mathbf{A}^T + (\boldsymbol{\eta} - \tilde{\mathbf{w}}^T \mathbf{h})^T \mathbf{B}^T) \mathbf{P} \mathbf{x}] + \frac{1}{\gamma} \text{tr}(\dot{\tilde{\mathbf{w}}}^T \tilde{\mathbf{w}}) \\ &= \frac{1}{2} [\mathbf{x}^T (\mathbf{P}\mathbf{A} + \mathbf{A}^T \mathbf{P}) \mathbf{x} + (\mathbf{x}^T \mathbf{P} \mathbf{B} \boldsymbol{\eta} - \mathbf{x}^T \mathbf{P} \mathbf{B} \tilde{\mathbf{w}}^T \mathbf{h} + \boldsymbol{\eta}^T \mathbf{B}^T \mathbf{P} \mathbf{x} - \mathbf{h}^T \tilde{\mathbf{w}} \mathbf{B}^T \mathbf{P} \mathbf{x})] + \frac{1}{\gamma} \text{tr}(\dot{\tilde{\mathbf{w}}}^T \tilde{\mathbf{w}}) \\ &= -\frac{1}{2} \mathbf{x}^T \mathbf{Q} \mathbf{x} + \boldsymbol{\eta}^T \mathbf{B}^T \mathbf{P} \mathbf{x} - \mathbf{h}^T \tilde{\mathbf{w}} \mathbf{B}^T \mathbf{P} \mathbf{x} + \frac{1}{\gamma} \text{tr}(\dot{\tilde{\mathbf{w}}}^T \tilde{\mathbf{w}}) \end{aligned}$$

where $\mathbf{x}^T \mathbf{P} \mathbf{B} \tilde{\mathbf{w}}^T \mathbf{h} = \mathbf{h}^T \tilde{\mathbf{w}} \mathbf{B}^T \mathbf{P} \mathbf{x}$, $\mathbf{x}^T \mathbf{P} \mathbf{B} \boldsymbol{\eta} = \boldsymbol{\eta}^T \mathbf{B}^T \mathbf{P} \mathbf{x}$.

Since

$$\mathbf{h}^T \tilde{\mathbf{w}} \mathbf{B}^T \mathbf{P} \mathbf{x} = \text{tr}[\mathbf{B}^T \mathbf{P} \mathbf{x} \mathbf{h}^T \tilde{\mathbf{w}}]$$

then,

$$\dot{V} = -\frac{1}{2} \mathbf{x}^T \mathbf{Q} \mathbf{x} + \frac{1}{\gamma} \text{tr}(-\gamma \mathbf{B}^T \mathbf{P} \mathbf{x} \mathbf{h}^T \tilde{\mathbf{w}} + \dot{\tilde{\mathbf{w}}}^T \tilde{\mathbf{w}}) + \boldsymbol{\eta}^T \mathbf{B}^T \mathbf{P} \mathbf{x} \quad (6.11)$$

In reference to the adaptive law proposed in [6], we design two adaptive laws as follows:

1. First adaptive law

Choose adaptive law as

$$\dot{\hat{w}}^T = \gamma B^T P x h^T$$

Then

$$\dot{\hat{w}} = \gamma h x^T P B \quad (6.12)$$

Since $\dot{\tilde{w}} = \dot{\hat{w}}$, then submitting (6.12) into (6.11), we have

$$\dot{V} = -\frac{1}{2} x^T Q x + \eta^T B^T P x$$

From known, we have

$$\begin{aligned} \|\eta^T\| &\leq \|\eta_0\|, \quad \|B\| = \|M_0^{-1}(q)\| \\ \dot{V} &\leq -\frac{1}{2} \lambda_{\min}(Q) \|x\|^2 + \|\eta_0\| \|M_0^{-1}(q)\| \lambda_{\max}(P) \|x\| \\ &= -\frac{1}{2} \|x\| [\lambda_{\min}(Q) \|x\| - 2 \|\eta_0\| \|M_0^{-1}(q)\| \lambda_{\max}(P)] \end{aligned}$$

where $\lambda_{\max}(P)$ and $\lambda_{\min}(Q)$ denote the maximum eigenvalue of matrix P and the minimum eigenvalue of matrix Q respectively.

To satisfy $\dot{V} \leq 0$, $\lambda_{\min}(Q) \geq \frac{2\|M_0^{-1}(q)\|\lambda_{\max}(P)}{\|x\|} \|\eta_0\|$ should be satisfied, that is,

$$\|x\| = \frac{2\|M_0^{-1}(q)\|\lambda_{\max}(P)}{\lambda_{\min}(Q)} \|\eta_0\| \quad (6.13)$$

From (6.12), we can get a conclusion: the bigger value the eigenvalue of Q is, or the smaller value the eigenvalue of P is, or the smaller value of η_0 is, the smaller of x radius convergence is.

The shortcoming of the first adaptive is the boundedness of $\tilde{w} = \hat{w} - w^*$ can be guaranteed.

2. Second adaptive law

Choose adaptive law as

$$\dot{\hat{w}}^T = \gamma B^T P x h^T + k_1 \gamma \|x\| \hat{w}^T$$

then,

$$\dot{\tilde{w}} = \gamma \mathbf{h} \mathbf{x}^T \mathbf{P} \mathbf{B} + k_1 \gamma \|\mathbf{x}\| \tilde{w} \quad (6.14)$$

where $k_1 > 0$.

Submitting (6.14) into (6.11), we have

$$\begin{aligned} \dot{V} &= -\frac{1}{2} \mathbf{x}^T \mathbf{Q} \mathbf{x} + \frac{1}{\gamma} \text{tr}(k_1 \gamma \|\mathbf{x}\| \tilde{w}^T \tilde{w}) + \boldsymbol{\eta}^T \mathbf{B}^T \mathbf{P} \mathbf{x} \\ &= -\frac{1}{2} \mathbf{x}^T \mathbf{Q} \mathbf{x} + k_1 \|\mathbf{x}\| \text{tr}(\tilde{w}^T \tilde{w}) + \boldsymbol{\eta}^T \mathbf{B}^T \mathbf{P} \mathbf{x} \end{aligned}$$

According to the characteristics of norm F, we have $\text{tr}[\tilde{\mathbf{x}}^T (\mathbf{x} - \tilde{\mathbf{x}})] \leq \|\tilde{\mathbf{x}}\|_{\text{F}} \|\mathbf{x}\|_{\text{F}} - \|\tilde{\mathbf{x}}\|_{\text{F}}^2$, then,

$$\text{tr}[\tilde{w}^T \tilde{w}] = \text{tr}[\tilde{w}^T \tilde{w}] = \text{tr}[\tilde{w}^T (\mathbf{w}^* + \tilde{w})] \leq \|\tilde{w}\|_{\text{F}} \|\mathbf{w}^*\|_{\text{F}} - \|\tilde{w}\|_{\text{F}}^2$$

Since

$$-k_1 \|\tilde{w}\|_{\text{F}} w_{\max} + k_1 \|\tilde{w}\|_{\text{F}}^2 = k_1 \left(\|\tilde{w}\|_{\text{F}} - \frac{w_{\max}}{2} \right)^2 - \frac{k_1}{4} w_{\max}^2$$

then

$$\begin{aligned} \dot{V} &\leq -\frac{1}{2} \mathbf{x}^T \mathbf{Q} \mathbf{x} + k_1 \|\mathbf{x}\| \left(\|\tilde{w}\|_{\text{F}} \|\mathbf{w}^*\|_{\text{F}} - \|\tilde{w}\|_{\text{F}}^2 \right) + \boldsymbol{\eta}^T \mathbf{B}^T \mathbf{P} \mathbf{x} \\ &\leq -\frac{1}{2} \lambda_{\min}(\mathbf{Q}) \|\mathbf{x}\|^2 + k_1 \|\mathbf{x}\| \|\tilde{w}\|_{\text{F}} \|\mathbf{w}^*\|_{\text{F}} - k_1 \|\mathbf{x}\| \|\tilde{w}\|_{\text{F}}^2 + \|\boldsymbol{\eta}_0\| \lambda_{\max}(\mathbf{P}) \|\mathbf{x}\| \\ &\leq -\|\mathbf{x}\| \left(\frac{1}{2} \lambda_{\min}(\mathbf{Q}) \|\mathbf{x}\| - k_1 \|\tilde{w}\|_{\text{F}} w_{\max} + k_1 \|\tilde{w}\|_{\text{F}}^2 - \|\boldsymbol{\eta}_0\| \lambda_{\max}(\mathbf{P}) \right) \\ &= -\|\mathbf{x}\| \left(\frac{1}{2} \lambda_{\min}(\mathbf{Q}) \|\mathbf{x}\| + k_1 \left(\|\tilde{w}\|_{\text{F}} - \frac{w_{\max}}{2} \right)^2 - \frac{k_1}{4} w_{\max}^2 - \|\boldsymbol{\eta}_0\| \lambda_{\max}(\mathbf{P}) \right) \end{aligned}$$

To guarantee $\dot{V} \leq 0$, the following conditions must be satisfied:

$$\frac{1}{2} \lambda_{\min}(\mathbf{Q}) \|\mathbf{x}\| \geq \|\boldsymbol{\eta}_0\| \lambda_{\max}(\mathbf{P}) + \frac{k_1}{4} w_{\max}^2$$

or

$$k_1 \left(\|\tilde{w}\|_{\text{F}} - \frac{w_{\max}}{2} \right)^2 \geq \|\boldsymbol{\eta}_0\| \lambda_{\max}(\mathbf{P}) + \frac{k_1}{4} w_{\max}^2$$

Then we get the boundedness of the closed-loop system as

$$\|\mathbf{x}\| \geq \frac{2}{\lambda_{\min}(\mathbf{Q})} \left(\|\boldsymbol{\eta}_0\| \lambda_{\max}(\mathbf{P}) + \frac{k_1}{4} w_{\max}^2 \right) \quad (6.15)$$

or

$$\|\tilde{\mathbf{w}}\|_F \geq \frac{w_{\max}}{2} + \sqrt{\frac{1}{k_1} \left(\|\boldsymbol{\eta}_0\| \lambda_{\max}(\mathbf{P}) + \frac{k_1}{4} w_{\max}^2 \right)}$$

From (6.15), we can get a conclusion: the bigger value the eigenvalue of \mathbf{Q} is, or the smaller value the eigenvalue of \mathbf{P} is, or the smaller value of $\boldsymbol{\eta}_0$ is, or smaller value w_{\max} is, the smaller radius of \mathbf{x} convergence is.

The shortcoming of the controller is that nominal model of the robotic manipulators must be known.

6.1.4 Simulation Examples

6.1.4.1 First Example

Consider a simple servo system as

$$M\ddot{q} = \boldsymbol{\tau} + d(\dot{q})$$

where $M = 10$, $d(\dot{q})$ denotes friction force, and $d(\dot{q}) = -15\dot{q} - 30\text{sgn}(\dot{q})$.

The desired trajectory is $q_d = \sin t$, the initial value of the plant is $[0.6 \quad 0]^T$. In simulation, we use control law (6.9) and first adaptive law (6.12). The parameters are chosen as $\mathbf{Q} = \begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix}$, $\alpha = 3$, $\gamma = 200$, and $k_1 = 0.001$.

For RBF neural network, the parameters of Gaussian function c_i and b_i are chosen as $[-2 \quad -1 \quad 0 \quad 1 \quad 2]$ and 5, and the initial weight value is chosen as zero.

In the simulation, $S1 = 1$ denotes first adaptive law, and $S1 = 2$ denotes second adaptive law. $S = 1$ denotes controller only based on nominal model, $S = 2$ denotes controller based on precise compensation, and $S = 3$ denotes controller based on RBF compensation.

In the simulation, we choose $S1 = 1$ and $S = 3$. To test the effect of hidden nets number on the approximation precision, we use 5 hidden nets and 19 hidden nets, respectively; the simulation results are shown from Figs. 6.1, 6.2, 6.3, and 6.4. From the results, it is shown that the more the hidden nets is chosen, the smaller approximation error can be gotten.

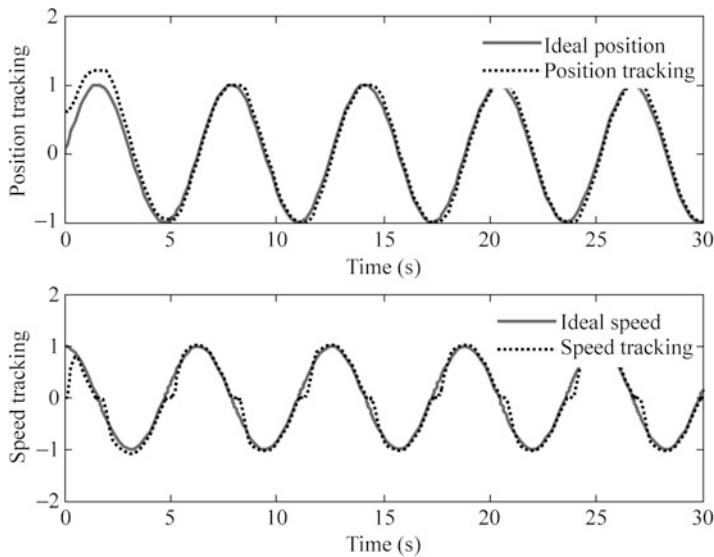


Fig. 6.1 Position and speed tracking with RBF compensation (five hidden nets, $S = 3$)

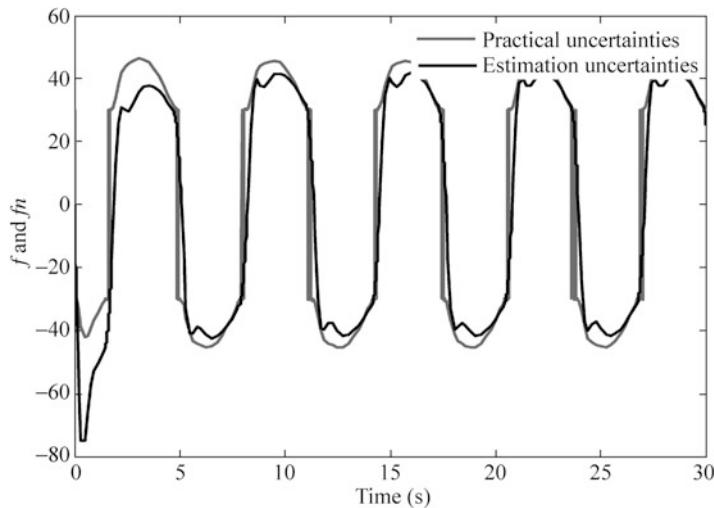


Fig. 6.2 $f(x)$ estimation with RBF (five hidden nets, $S = 3$)

Only choosing $S = 1$, the position and speed tracking without compensation is shown in Fig. 6.5.

The Simulink program of this example is chap6_1sim.mdl; the Matlab programs of the example are given in the [Appendix](#).

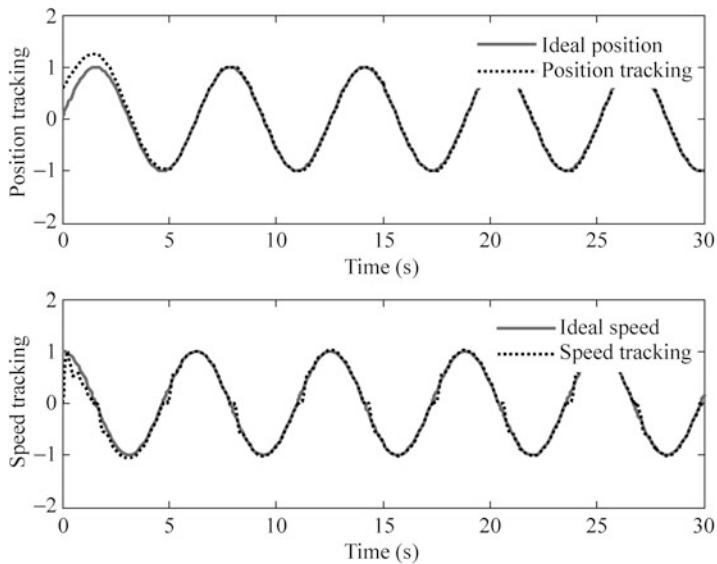


Fig. 6.3 Position and speed tracking with RBF compensation (19 hidden nets, $S = 3$)

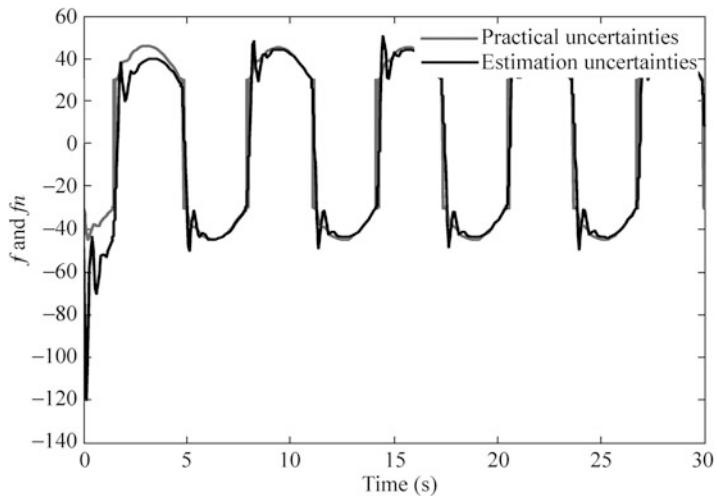


Fig. 6.4 $f(x)$ estimation with RBF (19 hidden nets, $S = 3$)

6.1.4.2 Second Example

Consider a two-link manipulator dynamic equation as

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} + \mathbf{d}$$

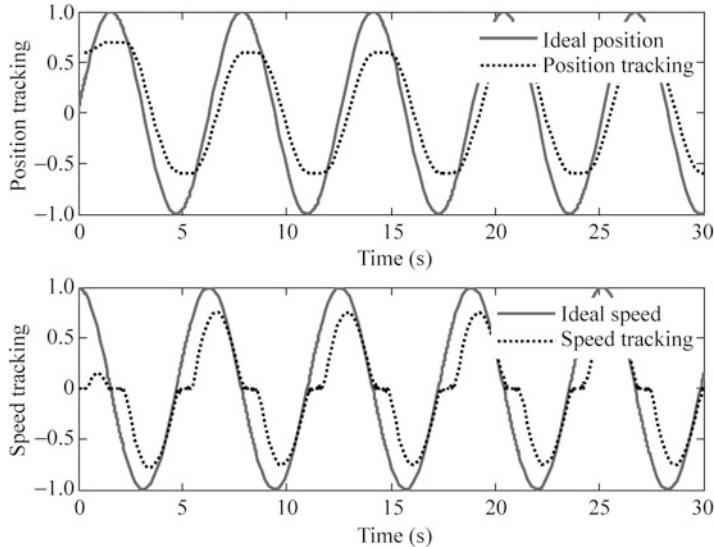


Fig. 6.5 Position and speed tracking without compensation ($S = 1$)

where

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} v + q_{01} + 2\gamma \cos(q_2) & q_{01} + q_{02} \cos(q_2) \\ q_{01} + q_{02} \cos(q_2) & q_{01} \end{bmatrix}$$

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} -q_{02}\dot{q}_2 \sin(q_2) & -q_{02}(\dot{q}_1 + \dot{q}_2) \sin(q_2) \\ q_{02}\dot{q}_1 \sin(q_2) & 0 \end{bmatrix}$$

$$\mathbf{G}(\mathbf{q}) = \begin{bmatrix} 15g \cos q_1 + 8.75g \cos(q_1 + q_2) \\ 8.75g \cos(q_1 + q_2) \end{bmatrix}$$

and $v = 13.33$, $q_{01} = 8.98$, $q_{02} = 8.75$, and $g = 9.8$.

The disturbance is $d = d_1 + d_2\|\mathbf{e}\| + d_3\|\dot{\mathbf{e}}\|$, $d_1 = 2$, $d_2 = 3$, and $d_3 = 6$. The desired joint trajectory is

$$\begin{cases} q_{1d} = 1 + 0.2 \sin(0.5\pi t) \\ q_{2d} = 1 - 0.2 \cos(0.5\pi t) \end{cases}$$

The initial value of the plant is $[q_1 \quad q_2 \quad q_3 \quad q_4]^T = [0.6 \quad 0.3 \quad 0.5 \quad 0.5]^T$, and assume $\Delta\mathbf{M} = 0.2\mathbf{M}$, $\Delta\mathbf{C} = 0.2\mathbf{C}$, and $\Delta\mathbf{G} = 0.2\mathbf{G}$.

In simulation, we use control law (6.9) and first adaptive law (6.12), in the program, we set $S = 3$, and $S_1 = 1$. The parameters are chosen as

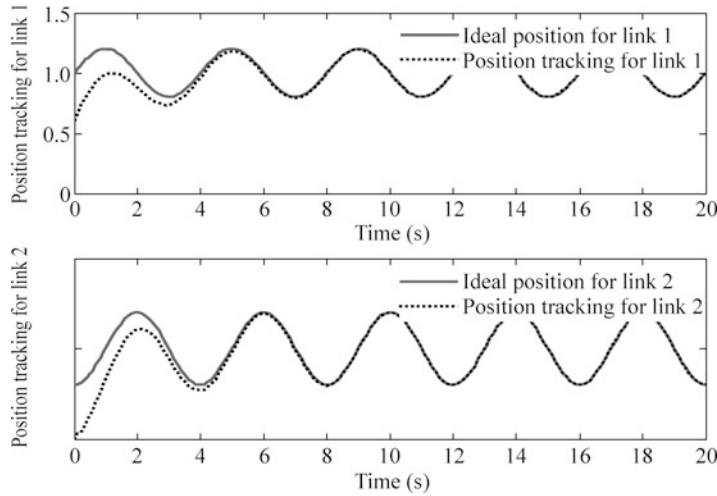


Fig. 6.6 Position tracking of link 1 and link 2

$$\mathbf{Q} = \begin{bmatrix} 50 & 0 & 0 & 0 \\ 0 & 50 & 0 & 0 \\ 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 50 \end{bmatrix},$$

$$\alpha = 3, \gamma = 20, k_1 = 0.001.$$

For RBF neural network, the parameters of Gaussian function c_i and b_i are chosen as $\begin{bmatrix} -2 & -1 & 0 & 1 & 2 \end{bmatrix}$ and 3.0, and the initial weight value is chosen as 0.10. In the simulation, $S_1 = 1$ denotes first adaptive law, and $S_1 = 2$ denotes second adaptive law. $S = 1$ denotes controller only based on nominal model, $S = 2$ denotes controller based on precise compensation, and $S = 3$ denotes controller based on RBF compensation. The simulation results are shown from Figs. 6.6, 6.7, 6.8, and 6.9.

The Simulink program of this example is chap6_2sim.mdl; the Matlab programs of the example are given in the [Appendix](#).

6.2 RBF Neural Robot Controller Design with Sliding Mode Robust Term

6.2.1 Problem Description

Consider dynamic equation of n-link manipulator as

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{F}(\dot{\mathbf{q}}) + \boldsymbol{\tau}_d = \boldsymbol{\tau} \quad (6.16)$$

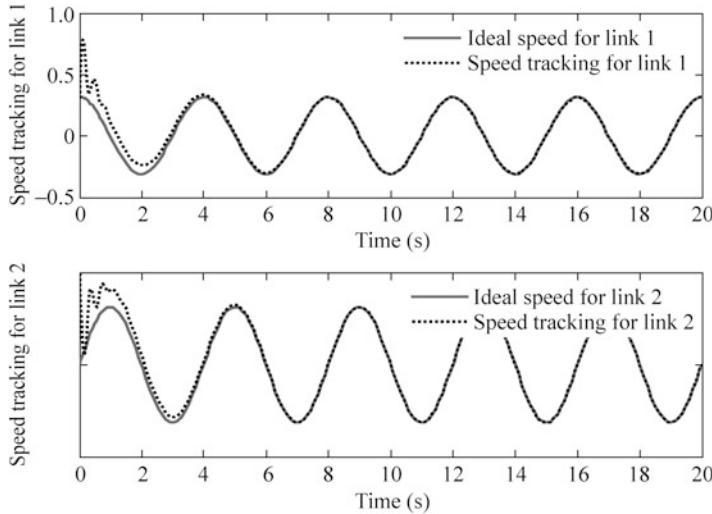


Fig. 6.7 Speed tracking of link 1 and link 2

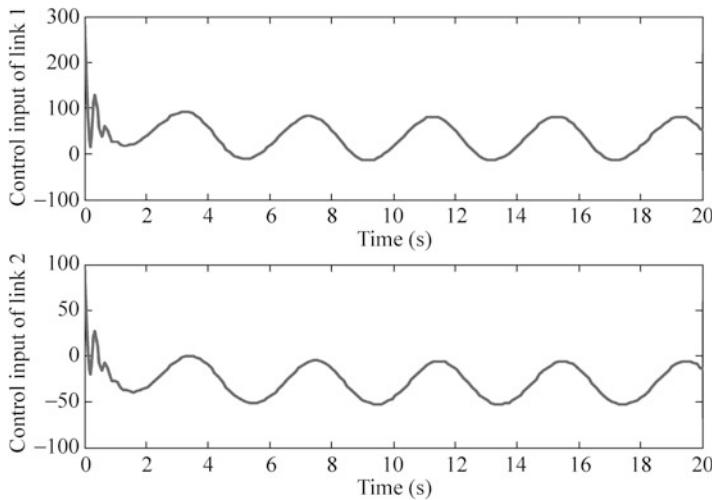


Fig. 6.8 Control input of link 1 and link 2

where $\mathbf{M}(\mathbf{q})$ is an $n \times n$ inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is an $n \times n$ matrix containing the centrifugal and Coriolis terms, $\mathbf{G}(\mathbf{q})$ is an $n \times 1$ vector containing gravitational forces and torques, \mathbf{q} is generalized joint coordinates, τ is joint torques, and τ_d denotes disturbances.

The tracking error vector is designed as $\mathbf{e}(t) = \mathbf{q}_d(t) - \mathbf{q}(t)$, and defines the sliding mode function as

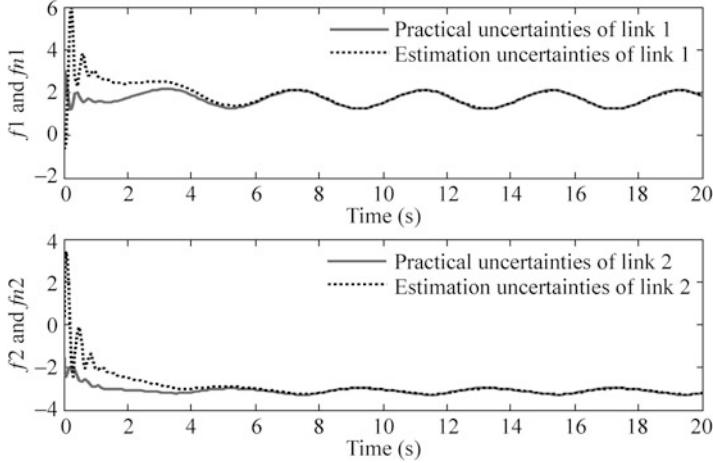


Fig. 6.9 $f(x)$ estimation of link 1 and link 2

$$\mathbf{r} = \dot{\mathbf{e}} + \boldsymbol{\Lambda} \mathbf{e} \quad (6.17)$$

where $\boldsymbol{\Lambda} = \boldsymbol{\Lambda}^T = [\lambda_1 \quad \lambda_2 \quad \cdots \quad \lambda_n]^T > 0$ is an appropriately chosen coefficient vector such that $s^{n-1} + \lambda_{n-1}s^{n-2} + \cdots + \lambda_1$ is Hurwitz (i.e., $\mathbf{e} \rightarrow 0$ exponentially as $\mathbf{r} \rightarrow 0$).

The sliding mode tracking error \mathbf{r} can be viewed as the real-valued utility function of the plant performance. When \mathbf{r} is small, system performance is good. For the system (6.16), all modeling information was expressed as $f(\mathbf{x})$ by using the sliding mode tracking error \mathbf{r} [7].

The item (6.17) gives

$$\dot{\mathbf{q}} = -\mathbf{r} + \dot{\mathbf{q}}_d + \boldsymbol{\Lambda} \mathbf{e} \quad (6.18)$$

and

$$\begin{aligned} M\dot{\mathbf{r}} &= M(\ddot{\mathbf{q}}_d - \ddot{\mathbf{q}} + \boldsymbol{\Lambda}\dot{\mathbf{e}}) = M(\ddot{\mathbf{q}}_d + \boldsymbol{\Lambda}\dot{\mathbf{e}}) - M\ddot{\mathbf{q}} \\ &= M(\ddot{\mathbf{q}}_d + \boldsymbol{\Lambda}\dot{\mathbf{e}}) + C\dot{\mathbf{q}} + \mathbf{G} + \mathbf{F} + \boldsymbol{\tau}_d - \boldsymbol{\tau} \\ &= M(\ddot{\mathbf{q}}_d + \boldsymbol{\Lambda}\dot{\mathbf{e}}) - \mathbf{C}\mathbf{r} + C(\dot{\mathbf{q}}_d + \boldsymbol{\Lambda}\mathbf{e}) + \mathbf{G} + \mathbf{F} + \boldsymbol{\tau}_d - \boldsymbol{\tau} \\ &= -\mathbf{C}\mathbf{r} - \boldsymbol{\tau} + \mathbf{f} + \boldsymbol{\tau}_d \end{aligned} \quad (6.19)$$

where $\mathbf{f}(\mathbf{x}) = M\ddot{\mathbf{q}}_r + C\dot{\mathbf{q}}_r + \mathbf{G} + \mathbf{F}$, $\dot{\mathbf{q}}_r = \dot{\mathbf{q}}_d + \boldsymbol{\Lambda}\mathbf{e}$.

From $\mathbf{f}(\mathbf{x})$ expression, we can see that the term $\mathbf{f}(\mathbf{x})$ 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 $\mathbf{f}(\mathbf{x})$.

6.2.2 RBF Approximation

RBF algorithm is described as

$$h_j = \exp \frac{\|\mathbf{x} - \mathbf{c}_j\|^2}{b_j^2}, \quad j = 1, 2, \dots, m$$

$$f(\mathbf{x}) = \mathbf{W}^T \mathbf{h} + \varepsilon \quad (6.20)$$

where \mathbf{x} is input of RBF, \mathbf{W} is optimum weight value, $\mathbf{h} = [h_1 \ h_2 \ \dots \ h_m]^T$, and ε is a very small value.

The output of RBF is used to approximate $f(\mathbf{x})$,

$$\hat{f}(\mathbf{x}) = \tilde{\mathbf{W}}^T \mathbf{h} \quad (6.21)$$

where $\tilde{\mathbf{W}} = \mathbf{W} - \hat{\mathbf{W}}$, $\|\mathbf{W}\|_F \leq W_{\max}$.

From (6.20) and (6.21), we have

$$f - \hat{f} = \mathbf{W}^T \mathbf{h} + \varepsilon - \hat{\mathbf{W}}^T \mathbf{h} = \tilde{\mathbf{W}}^T \mathbf{h} + \varepsilon$$

From $\frac{f(\mathbf{x})}{x} = [\mathbf{e}^T \ \dot{\mathbf{e}}^T \ \mathbf{q}_d^T \ \dot{\mathbf{q}}_d^T]$ expression, the input of RBF should be chosen as

6.2.3 Control Law Design and Stability Analysis

For the system (6.16), the control law as proposed as [7],

$$\boldsymbol{\tau} = \hat{f}(\mathbf{x}) + \mathbf{K}_v \mathbf{r} - \nu \quad (6.22)$$

with robust term $\nu = -(\varepsilon_N + b_d) \text{sgn}(\mathbf{r})$, where $\hat{f}(\mathbf{x})$ is estimation of $f(\mathbf{x})$, and ν is robustness term.

The corresponding RBF adaptive law is designed as

$$\dot{\hat{\mathbf{W}}} = \boldsymbol{\Gamma} \mathbf{h} \mathbf{r}^T \quad (6.23)$$

where $\boldsymbol{\Gamma} = \boldsymbol{\Gamma}^T > 0$.

Inserting (6.19) yields

$$\begin{aligned} \mathbf{M} \dot{\mathbf{r}} &= -\mathbf{C} \mathbf{r} - (\hat{f}(\mathbf{x}) + \mathbf{K}_v \mathbf{r} - \nu) + \mathbf{f} + \boldsymbol{\tau}_d \\ &= -(\mathbf{K}_v + \mathbf{C}) \mathbf{r} + \tilde{\mathbf{W}}^T \mathbf{h} + (\boldsymbol{\epsilon} + \boldsymbol{\tau}_d) + \nu \\ &= -(\mathbf{K}_v + \mathbf{C}) \mathbf{r} + \boldsymbol{\varsigma}_1 \end{aligned} \quad (6.24)$$

where $\zeta_1 = \tilde{W}^T \varphi + (\boldsymbol{\epsilon} + \boldsymbol{\tau}_d) + \boldsymbol{v}$.

The stability proof of the closed system was given with two steps as follows [7]. Firstly, define Lyapunov function as

$$L = \frac{1}{2} \mathbf{r}^T \mathbf{M} \mathbf{r} + \frac{1}{2} \text{tr}(\tilde{W}^T \Gamma^{-1} \tilde{W})$$

Thus,

$$\dot{L} = \mathbf{r}^T \mathbf{M} \dot{\mathbf{r}} + \frac{1}{2} \mathbf{r}^T \dot{\mathbf{M}} \mathbf{r} + \text{tr}(\tilde{W}^T \Gamma^{-1} \dot{\tilde{W}})$$

Secondly, inserting (6.24) into above yields

$$\dot{L} = -\mathbf{r}^T \mathbf{K}_v \mathbf{r} + \frac{1}{2} \mathbf{r}^T (\dot{\mathbf{M}} - 2\mathbf{C}) \mathbf{r} + \text{tr} \tilde{W}^T (\Gamma^{-1} \dot{\tilde{W}} + \mathbf{h} \mathbf{r}^T) + \mathbf{r}^T (\boldsymbol{\epsilon} + \boldsymbol{\tau}_d + \boldsymbol{v})$$

Since:

1. According to the skew-symmetric characteristics of manipulator dynamic equation, $\mathbf{r}^T (\dot{\mathbf{M}} - 2\mathbf{C}) \mathbf{r} = 0$;
2. $\mathbf{r}^T \tilde{W}^T \mathbf{h} = \text{tr}(\tilde{W}^T \mathbf{h} \mathbf{r}^T)$;
3. $\dot{\tilde{W}} = -\dot{\tilde{W}} = -\Gamma \mathbf{h} \mathbf{r}^T$.
then,

$$\dot{L} = -\mathbf{r}^T \mathbf{K}_v \mathbf{r} + \mathbf{r}^T (\boldsymbol{\epsilon} + \boldsymbol{\tau}_d + \boldsymbol{v})$$

Consider

$$\begin{aligned} \mathbf{r}^T (\boldsymbol{\epsilon} + \boldsymbol{\tau}_d + \boldsymbol{v}) &= \mathbf{r}^T (\boldsymbol{\epsilon} + \boldsymbol{\tau}_d) + \mathbf{r}^T ((-\epsilon_N + b_d) \text{sgn}(\mathbf{r})) \\ &= \mathbf{r}^T (\boldsymbol{\epsilon} + \boldsymbol{\tau}_d) - \|\mathbf{r}\| (\epsilon_N + b_d) \leq 0 \end{aligned}$$

There results finally

$$\dot{L} \leq -\mathbf{r}^T \mathbf{K}_v \mathbf{r} \leq 0$$

6.2.4 Simulation Examples

6.2.4.1 First Example

Consider a simple servo system as

$$M\ddot{q} + F(\dot{q}) = \tau$$

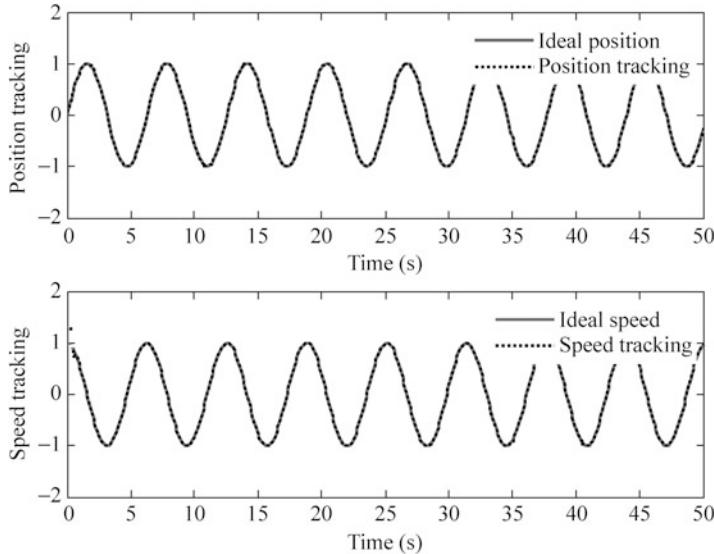


Fig. 6.10 Position and speed tracking with RBF compensation

where $M = 10$, $F(\dot{q})$ denotes friction force, and $F(\dot{q}) = 15\dot{q} + 30\text{sgn}(\dot{q})$.

From (6.16), (6.17), (6.18), and (6.19), we have $f(x) = M\ddot{q}_r + F = M(\ddot{q}_d + \Lambda\dot{e}) + F$.

For RBF neural network, the structure is 2-7-1, the input is chosen as $\mathbf{z} = [e \quad \dot{e}]$, the parameters of Gaussian function c_i and b_i are chosen as $[-1.5 \quad -1.0 \quad -0.5 \quad 0 \quad 0.5 \quad 1.0 \quad 1.5]$ and 10, and the initial weight value is chosen as zero. The desired trajectory is $q_d = \sin t$, and the initial value of the plant is $[0.10 \quad 0]^T$.

In simulation, we use control law (6.22) and adaptive law (6.23), the parameters are chosen as $\Lambda = 15$, $\Gamma = 100$, and $K_v = 110$. The simulation results are shown from Figs. 6.10 and 6.11.

The Simulink program of this example is chap6_3sim.mdl; the Matlab programs of the example are given in the [Appendix](#).

6.2.4.2 Second Example

Consider a plant as

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{F}(\dot{\mathbf{q}}) + \boldsymbol{\tau}_d = \boldsymbol{\tau}$$

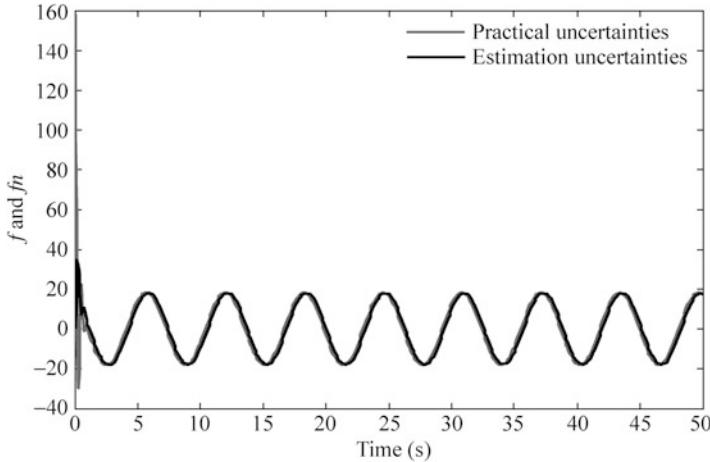


Fig. 6.11 $f(\mathbf{x})$ estimation with RBF

where

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} p_1 + p_2 + 2p_3 \cos q_2 & p_2 + p_3 \cos q_2 \\ p_2 + p_3 \cos q_2 & p_2 \end{bmatrix},$$

$$\mathbf{V}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} -p_3 \dot{q}_2 \sin q_2 & -p_3 (\dot{q}_1 + \dot{q}_2) \sin q_2 \\ p_3 \dot{q}_1 \sin q_2 & 0 \end{bmatrix},$$

$$\mathbf{G}(\mathbf{q}) = \begin{bmatrix} p_4 g \cos q_1 + p_5 g \cos(q_1 + q_2) \\ p_5 g \cos(q_1 + q_2) \end{bmatrix},$$

$$\mathbf{F}(\dot{\mathbf{q}}) = 0.02 \operatorname{sgn}(\dot{\mathbf{q}}), \quad \boldsymbol{\tau}_d = [0.2 \sin(t) \quad 0.2 \sin(t)]^T, \quad \mathbf{p} = [p_1, p_2, p_3, p_4, p_5] = [2.9, 0.76, 0.87, 3.04, 0.87].$$

For RBF neural network, the structure is 2-7-1, the input is chosen as $\mathbf{z} = [\mathbf{e} \quad \dot{\mathbf{e}}]$, the parameters of Gaussian function c_i and b_i are chosen as $[-1.5 \quad -1.0 \quad -0.5 \quad 0 \quad 0.5 \quad 1.0 \quad 1.5]$ and 10, and the initial weight value is chosen as zero. The desired trajectory is $q_{1d} = 0.1 \sin t$, $q_{2d} = 0.1 \sin t$. The initial value of the plant is $[0.09 \quad 0 \quad -0.09 \quad 0]$.

Using control law (6.22) and adaptive law (6.23), $\mathbf{K}_v = \operatorname{diag}\{10, 10\}$, $\boldsymbol{\Gamma} = \operatorname{diag}\{15, 15\}$, and $\Lambda = \operatorname{diag}\{5, 5\}$. The simulation results are shown from Figs. 6.12, 6.13, 6.14, and 6.15.

The Simulink program of this example is chap6_4sim.mdl; the Matlab programs of the example are given in the Appendix.

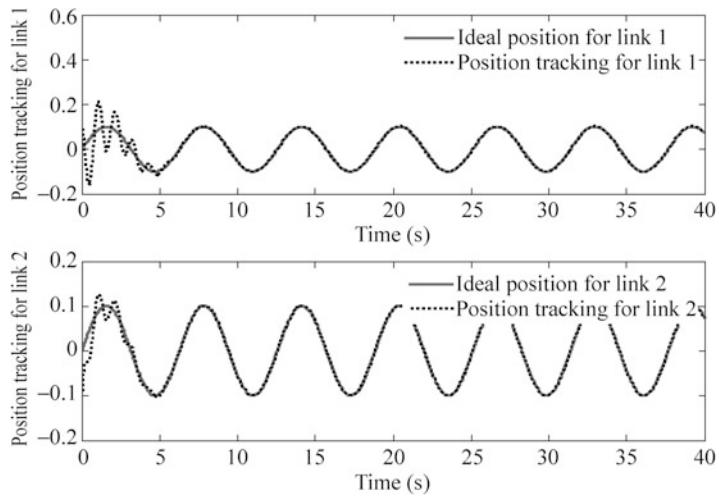


Fig. 6.12 Position tracking

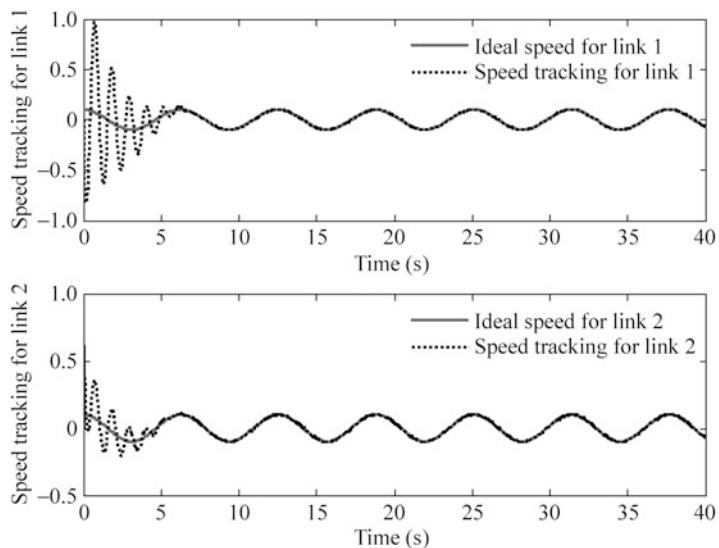


Fig. 6.13 Speed tracking

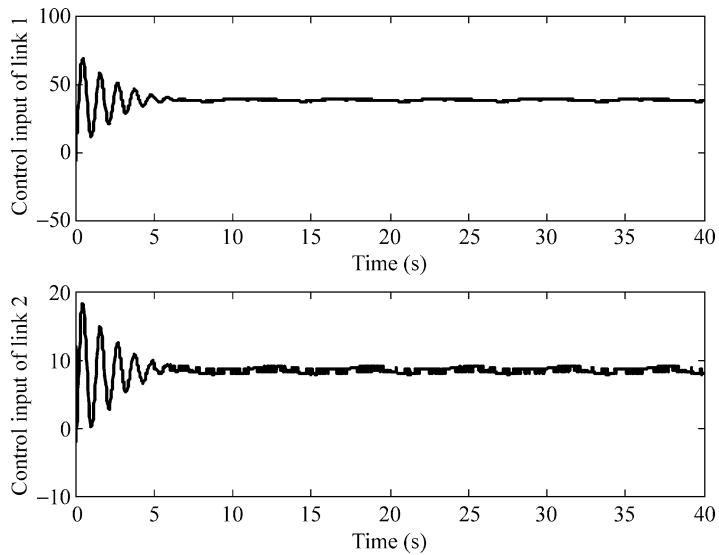


Fig. 6.14 Control input of link 1 and 2

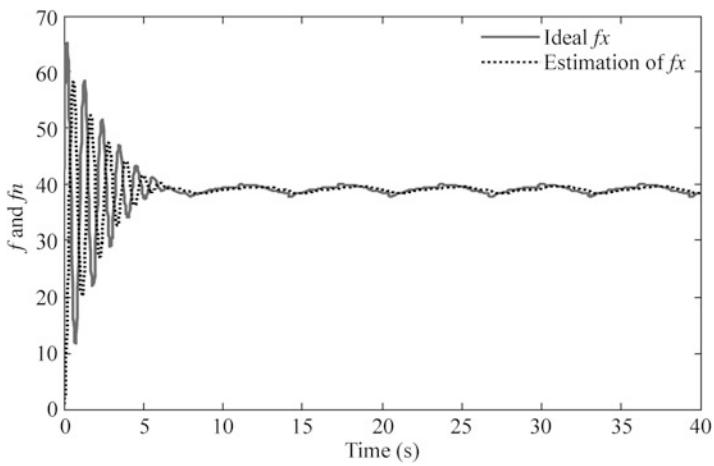


Fig. 6.15 $\|f(x)\|$ and $\|\hat{f}(x)\|$

6.3 Robust Control Based on RBF Neural Network with HJI

6.3.1 Foundation

Consider a system as

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{d} \\ \mathbf{z} = \mathbf{h}(\mathbf{x}) \end{cases} \quad (6.25)$$

where \mathbf{d} is disturbance and \mathbf{z} is critic signal for the system.

Definition. For signal $\mathbf{d}(t)$, its L_2 is $\|\mathbf{d}(t)\|_2 = \{\int_0^\infty \mathbf{d}^T(t)\mathbf{d}(t)dt\}^{\frac{1}{2}}$, which denotes the energy of $\mathbf{d}(t)$.

To express suppression ability, the following performance index is defined as

$$J = \sup_{\|\mathbf{d}\| \neq 0} \frac{\|\mathbf{z}\|_2}{\|\mathbf{d}\|_2} \quad (6.26)$$

The term J is called L_2 gain of the system, which denotes the robust performance index of the system. The smaller J is, the better robust performance is.

With theorem 2 given in [8] and the system (6.25), HJI (Hamilton–Jacobi inequality) can be described as follows: for a positive number γ , if there exists positive definite quasi-differentiable function $L(\mathbf{x}) \geq 0$ and

$$\dot{L} \leq \frac{1}{2} \left\{ \gamma^2 \|\mathbf{d}\|^2 - \|\mathbf{z}\|^2 \right\} \quad (\forall \mathbf{d}) \quad (6.27)$$

then, $J \leq \gamma$.

6.3.2 Controller Design and Analysis

HJI inequation was applied to design robust neural network controller for robots in [9]. In reference to theorem 3 in [9], a neural network adaptive controller is designed with HJI for n – link manipulators as follows.

Consider dynamic equation of n – link manipulators as

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \Delta(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{d} = \mathbf{T} \quad (6.28)$$

where $\mathbf{M}(\mathbf{q})$ is an $n \times n$ inertia matrix, $\mathbf{V}(\mathbf{q}, \dot{\mathbf{q}})$ is a $n \times n$ matrix containing the centrifugal and Coriolis terms, $\mathbf{G}(\mathbf{q})$ is an $n \times 1$ vector containing gravitational

forces and torques, \mathbf{q} is the angle vector, τ is joint torques, $\Delta(\mathbf{q}, \dot{\mathbf{q}})$ denotes uncertainties, and \mathbf{d} denotes outer disturbances.

Consider desired angle signal \mathbf{q}_d , the tracking error vector is $\mathbf{e} = \mathbf{q} - \mathbf{q}_d$, and design a feed-forward control law as

$$\mathbf{T} = \mathbf{u} + \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_d + \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}_d + \mathbf{G}(\mathbf{q}) \quad (6.29)$$

where \mathbf{u} is a feedback control law and to be designed in the following.

Submitting (6.29) into (6.28), we get closed system as

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{e}} + \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{e}} + \Delta(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{d} = \mathbf{u} \quad (6.30)$$

Letting $\Delta f(\mathbf{q}, \dot{\mathbf{q}}) = \Delta(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{d}$, we have

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{e}} + \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{e}} + \Delta f = \mathbf{u} \quad (6.31)$$

Use RBF to approximate Δf , then,

$$\Delta f = \mathbf{W}_f^* \boldsymbol{\sigma}_f + \boldsymbol{\epsilon}_f \quad (6.32)$$

where $\boldsymbol{\epsilon}_f$ denotes the approximation error, $\boldsymbol{\sigma}_f$ denotes the Gaussian function of RBF, and \mathbf{W}_f is weight value.

From (6.31) and (6.32), we have

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{e}} + \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{e}} + \mathbf{W}_f^* \boldsymbol{\sigma}_f + \boldsymbol{\epsilon}_f = \mathbf{u}$$

Define two variables as

$$\begin{cases} x_1 = \mathbf{e} \\ x_2 = \dot{\mathbf{e}} + \alpha \mathbf{e} \end{cases} \quad (6.33)$$

where $\alpha > 0$.

Then,

$$\begin{cases} \dot{x}_1 = x_2 - \alpha x_1 \\ M\dot{x}_2 = -Vx_2 + \omega - W_f^* \boldsymbol{\sigma}_f - \boldsymbol{\epsilon}_f + u \end{cases} \quad (6.34)$$

where $\omega = Ma\dot{\mathbf{e}} + V\alpha\mathbf{e}$.

To utilize the HJI inequation, the Eq. (6.34) should be written as (6.25),

$$\begin{cases} \dot{x} = f(x) + g(x)d \\ z = h(x) \end{cases} \quad (6.35)$$

where $f(x) = \begin{bmatrix} x_2 - \alpha x_1 \\ \frac{1}{M}(-Vx_2 + \omega - W_f^* \boldsymbol{\sigma}_f + u) \end{bmatrix}$, $g(x) = \begin{bmatrix} 0 \\ -\frac{1}{M} \end{bmatrix}$, and $d = \boldsymbol{\epsilon}_f$.

Since $d = \epsilon_f$, we can consider the approximation error ϵ_f as the disturbance d , and define $z_R = \dot{e} + \alpha e$, then, we can write the L_2 gain as $J_R = \sup_{\|e_f\| \neq 0} \frac{\|z\|_2}{\|e_f\|_2}$.

For the system (6.34), we design adaptive law as

$$\dot{\hat{W}}_f = -\eta x_2 \sigma_f^T \quad (6.36)$$

Design feedback control law as

$$u = -\omega - \frac{1}{2\gamma^2} x_2 + \hat{W}_f \sigma_f - \frac{1}{2} x_2 \quad (6.37)$$

where \hat{W}_f and σ_f are weight value and Gaussian function of RBF.

Then for the closed system (6.30), $J \leq \gamma$,

For the closed control system, in reference to the analysis in paper [9], the stability can be analyzed as follows:

Firstly, define the Lyapunov function as

$$L = \frac{1}{2} x_2^T M x_2 + \frac{1}{2\eta} \text{tr}(\tilde{W}_f^T \tilde{W}_f)$$

where $\tilde{W}_f = \hat{W}_f - W_f^*$.

Then, using (6.34) and (6.37), and considering the skew-symmetric characteristics of manipulator dynamic equation, we have

$$\begin{aligned} \dot{L} &= x_2^T M \dot{x}_2 + \frac{1}{2} x_2^T \dot{M} x_2 + \frac{1}{\eta} \text{tr}(\dot{\tilde{W}}_f^T \tilde{W}_f) \\ &= x_2^T (-Vx_2 + \omega - W_f^* \sigma_f - \epsilon_f + u) + \frac{1}{2} x_2^T \dot{M} x_2 + \frac{1}{\eta} \text{tr}(\dot{\tilde{W}}_f^T \tilde{W}_f) \\ &= x_2^T \left(-Vx_2 - W_f^* \sigma_f - \epsilon_f - \frac{1}{2\gamma^2} x_2 + \hat{W}_f \sigma_f - \frac{1}{2} x_2 \right) + \frac{1}{2} x_2^T \dot{M} x_2 + \frac{1}{\eta} \text{tr}(\dot{\tilde{W}}_f^T \tilde{W}_f) \\ &= x_2^T \left(-\epsilon_f - \frac{1}{2\gamma^2} x_2 + \tilde{W}_f \sigma_f - \frac{1}{2} x_2 \right) + \frac{1}{2} x_2^T (\dot{M} - 2V) x_2 + \frac{1}{\eta} \text{tr}(\dot{\tilde{W}}_f^T \tilde{W}_f) \\ &= -x_2^T \epsilon_f - \frac{1}{2\gamma^2} x_2^T x_2 + x_2^T \tilde{W}_f \sigma_f - \frac{1}{2} x_2^T x_2 + \frac{1}{\eta} \text{tr}(\dot{\tilde{W}}_f^T \tilde{W}_f) \end{aligned}$$

Define

$$H = \dot{L} - \frac{1}{2} \gamma^2 \|e_f\|^2 + \frac{1}{2} \|z_R\|^2 \quad (6.38)$$

Then,

$$H = -\mathbf{x}_2^T \boldsymbol{\epsilon}_f - \frac{1}{2\gamma^2} \mathbf{x}_2^T \mathbf{x}_2 + \mathbf{x}_2^T \tilde{\mathbf{W}}_f \boldsymbol{\sigma}_f - \frac{1}{2} \mathbf{x}_2^T \mathbf{x}_2 + \frac{1}{\eta} \text{tr} \left(\dot{\tilde{\mathbf{W}}}_f^T \tilde{\mathbf{W}}_f \right) - \frac{1}{2} \gamma^2 \|\boldsymbol{\epsilon}_f\|^2 + \frac{1}{2} \|z_R\|^2$$

Consider

1. $-\mathbf{x}_2^T \boldsymbol{\epsilon}_f - \frac{1}{2\gamma^2} \mathbf{x}_2^T \mathbf{x}_2 - \frac{1}{2} \gamma^2 \|\boldsymbol{\epsilon}_f\|^2 = -\frac{1}{2} \left\| \frac{1}{\gamma} \mathbf{x}_2 + \gamma \boldsymbol{\epsilon}_f \right\|^2 \leq 0;$
2. $\mathbf{x}_2^T \tilde{\mathbf{W}}_f \boldsymbol{\sigma}_f + \frac{1}{\eta} \text{tr} \left(\dot{\tilde{\mathbf{W}}}_f^T \tilde{\mathbf{W}}_f \right) = 0;$
3. $-\frac{1}{2} \mathbf{x}_2^T \mathbf{x}_2 + \frac{1}{2} \|z_R\|^2 = 0.$

Therefore we have $H \leq 0$, according to H definition in (6.38), we have

$$\dot{L} \leq \frac{1}{2} \gamma^2 \|\boldsymbol{\epsilon}_f\|^2 - \frac{1}{2} \|z_R\|^2$$

Thus from theorem 1, we can get $J \leq \gamma$.

6.3.3 Simulation Examples

6.3.3.1 First Example

Consider a servo system as

$$M\ddot{q} = T - d$$

where $M = 1.0$ is the inertia moment.

Suppose the ideal position signal as $q_d = \sin t$, the disturbance signal is $d = 150 \operatorname{sgn} \dot{q} + 10\dot{q}$, and the initial states of the plant are zero. Choose $\eta = 1,000$, $\alpha = 200$, and $\gamma = 0.10$, and the parameters of Gaussian function c_i and b_i are chosen as $[-1 \ -0.5 \ 0 \ 0.5 \ 1]$ and 50. Use adaptive law (6.36) and control laws (6.29) and (6.37); the results are shown from Figs. 6.16 and 6.17.

The Simulink program of this example is chap6_5sim.mdl; the Matlab programs of the example are given in the [Appendix](#).

6.3.3.2 Second Example

Consider two link manipulators dynamic equation as

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{D} = \mathbf{T}$$

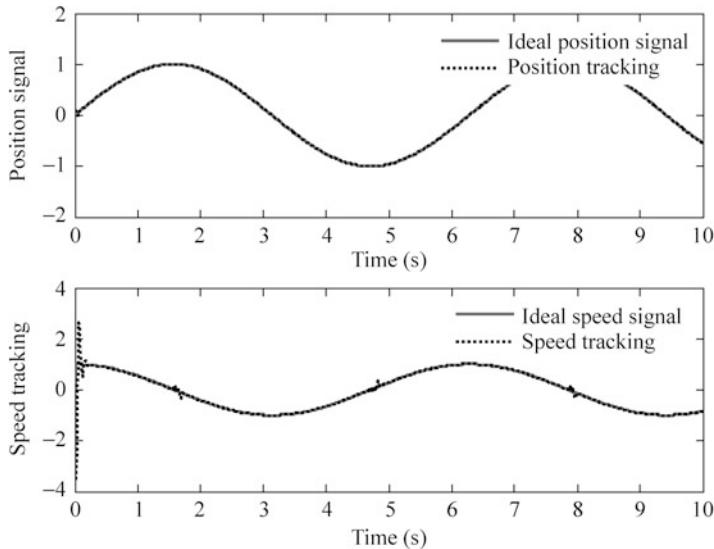


Fig. 6.16 Position and speed tracking with RBF compensation ($S = 2$)

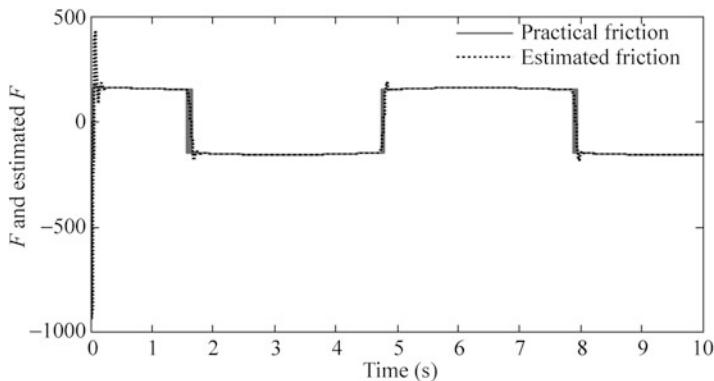


Fig. 6.17 Disturbance and its estimation ($S = 2$)

where $\mathbf{D} = \Delta(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{d}$, $M_{11} = (m_1 + m_2)r_1^2 + m_2r_2^2 + 2m_2r_1r_2 \cos q_2$, $M_{12} = M_{21} = m_2r_2^2 + m_2r_1r_2 \cos q_2$, $M_{22} = m_2r_2^2$, $\mathbf{V} = \begin{bmatrix} -V_{12}\dot{q}_2 & -V_{12}(\dot{q}_1 + \dot{q}_2) \\ V_{12}q_1 & 0 \end{bmatrix}$, $V_{12} = m_2r_1 \sin q_2$, $G_1 = (m_1 + m_2)r_1 \cos q_2 + m_2r_2 \cos(q_1 + q_2)$, $G_2 = m_2r_2 \cos(q_1 + q_2)$, $\mathbf{D} = \begin{bmatrix} 30 \operatorname{sgn} q_2 \\ 30 \operatorname{sgn} q_4 \end{bmatrix}$, $r_1 = 1$, $r_2 = 0.8$, $m_1 = 1$, $m_2 = 1.5$.

Suppose the ideal position signals as $q_{1d} = \sin t$, and $q_{2d} = \sin t$, and the initial states of the plant is zero. Choose 4-7-1 RBF structure, and set $\eta = 1,500$, $\alpha = 20$, $\gamma = 0.05$, the parameters of Gaussian function c_i and b_i are chosen as

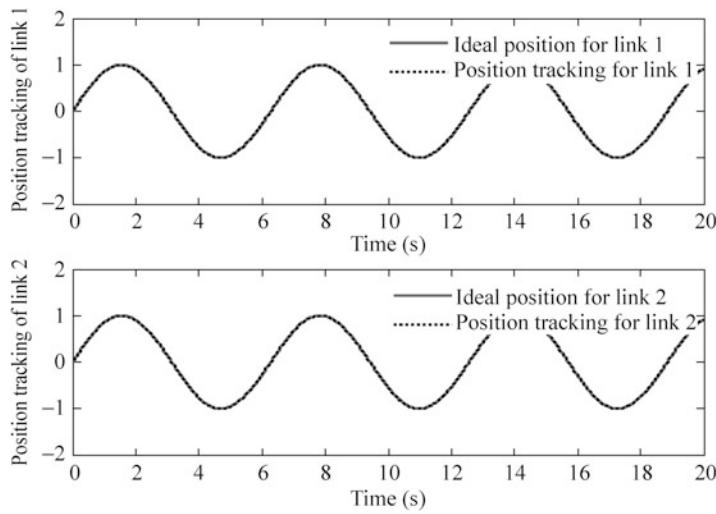


Fig. 6.18 Position tracking of with RBF compensation ($S = 2$)

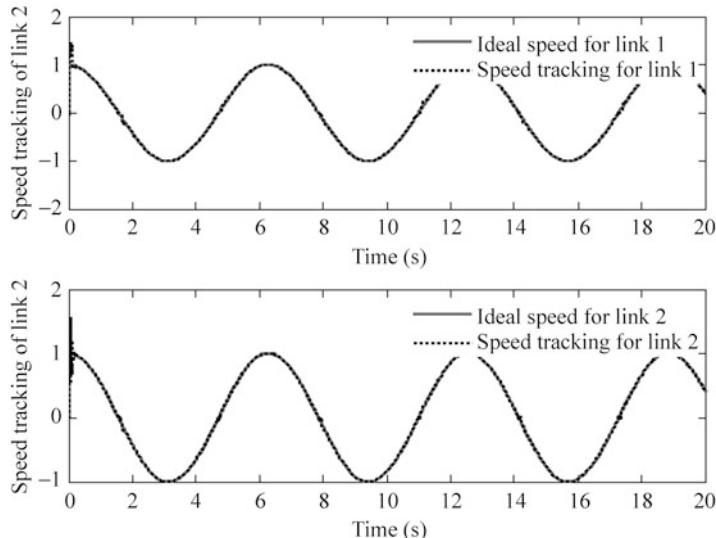


Fig. 6.19 Speed tracking of with RBF compensation ($S = 2$)

$[-1.5 \quad -1.0 \quad -0.5 \quad 0 \quad 0.5 \quad 1.0 \quad 1.5]$ and 10. Use adaptive law (6.36)) and control laws (6.29) and (6.37); the results are shown from Figs. 6.18, 6.19, and 6.20.

The Simulink program of this example is chap6_6sim.mdl; the Matlab programs of the example are given in the [Appendix](#).

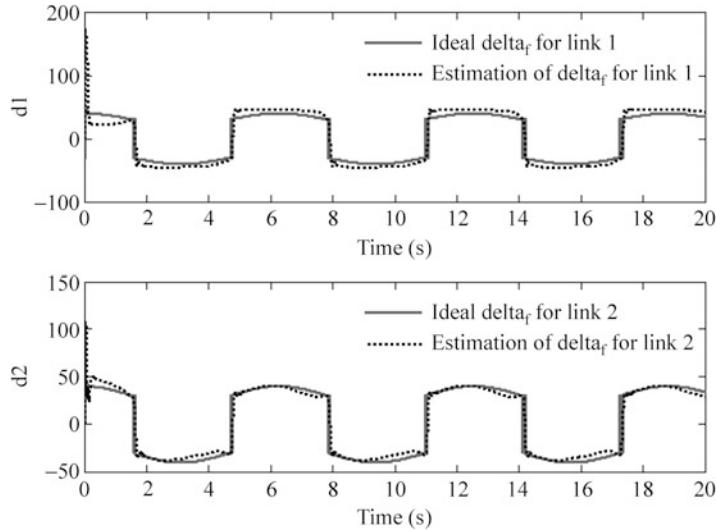
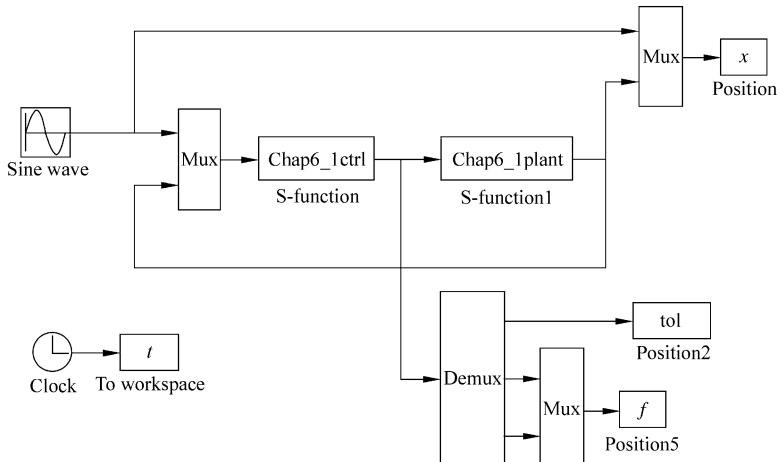


Fig. 6.20 Disturbance and its estimation ($S = 2$)

Appendix

Programs for Sect. 6.1.4.1

Simulink main program: chap6_1sim.mdl



S function for control law and adaptive law: chap6_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 kv kp
sizes = simsizes;
sizes.NumContStates = 5;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = 0.1*ones(1,5);
str = [];
ts = [0 0];
c=0.5*[-2 -1 0 1 2;
         -2 -1 0 1 2];
b=1.5*ones(5,1);
alfa=3;
kp=alfa^2;
kv=2*alfa;
function sys=mdlDerivatives(t,x,u)
global c b kv kp
qd=u(1);
dqdd=cos(t);
ddqdd=-sin(t);
q=u(2);
```

```
dq=u(3);  
e=q-qd;  
de=dq-dqd;  
  
A=[0 1;-kp -kv];  
D=10;  
B=[0 1/D]';  
  
Q=50*eye(2);  
P=lyap(A',Q);  
eig(P);  
  
th=[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(j)*b(j)));  
end  
gama=1200;  
  
S1=1;  
if S1==1          % First adaptive Law  
    S=gama*h*xi'*P*B;  
elseif S1==2      % Secod adaptive Law with UUB  
    k1=0.001;  
    S=gama*h*xi'*P*B+k1*gama*norm(xi)*th;  
end  
S=S';  
for i=1:1:5  
    sys(i)=S(i);  
end  
  
function sys=mdlOutputs(t,x,u)  
global c b kv kp  
qd=u(1);  
dqd=cos(t);  
ddqd=-sin(t);  
  
q=u(2);  
dq=u(3);  
e=q-qd;  
de=dq-dqd;
```

```

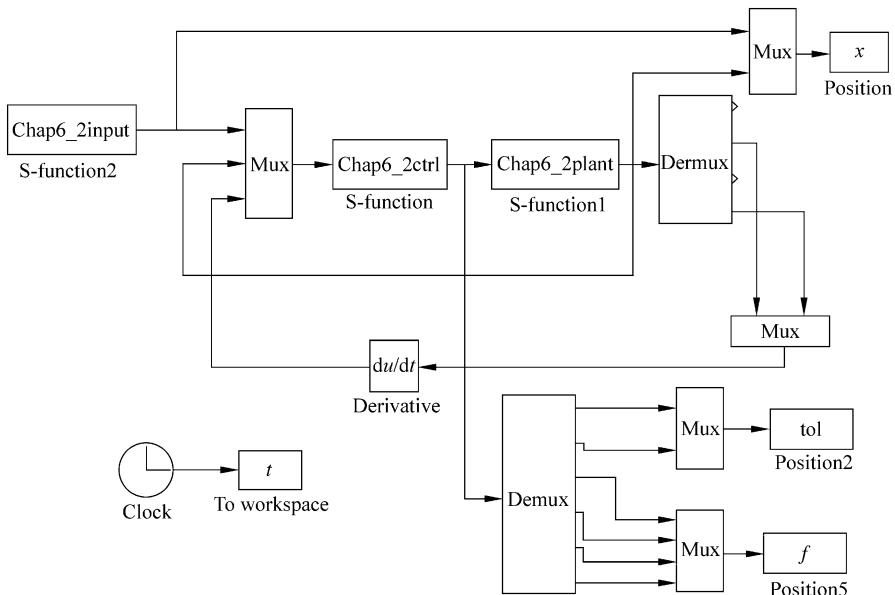
M=10;
tol1=M*(ddqd-kv*de-kp*e);
xi=[e;de];
h=zeros(5,1);
for j=1:1:5
    h(j)=exp(-norm(xi-c(:,j))^2/(2*b(j)*b(j)));
end
d=-15*dq-30*sign(dq);
f=d;
S=3;
if S==1           %Nominal model based controller
    fn=0;
    tol=tol1;
elseif S==2       %Modified computed torque controller
    fn=0;
    tol2=-f;
    tol=tol1+tol2;
elseif S==3       %RBF compensated controller
    th=[x(1) x(2) x(3) x(4) x(5)]';
    fn=th'*h;
    tol2=-fn;
    tol=tol1+1*tol2;
end
sys(1)=tol;
sys(2)=f;
sys(3)=fn;
S function for plant:chap6_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 = 2;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.6;0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
M=10;
d=-15*x(2)-30*sign(x(2));
tol=u(1);
sys(1)=x(2);
sys(2)=1/M*(tol+d);
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
Plot program:chap6_1plot.m
close all;
figure(1);
subplot(211);
plot(t,x(:,1),'r',t,x(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal position','position tracking');
subplot(212);
plot(t,cos(t),'r',t,x(:,3),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('ideal speed','speed tracking');
figure(2);
plot(t,f(:,1),'r',t,f(:,2),'b','linewidth',2);
xlabel('time(s)');ylabel('f and fn');
legend('Practical uncertainties','Estimation
uncertainties');
```

Programs for Sect. 6.1.4.2

Simulink main program: chap6_2sim.mdl



Tracking command program: chap6_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 = 4;
sizes.NumInputs = 0;
  
```

```
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [0 0];

function sys=mdlOutputs(t,x,u)
qd1=1+0.2*sin(0.5*pi*t);
qd2=1-0.2*cos(0.5*pi*t);
dqd1=0.2*0.5*pi*cos(0.5*pi*t);
dqd2=0.2*0.5*pi*sin(0.5*pi*t);

sys(1)=qd1;
sys(2)=qd2;
sys(3)=dqd1;
sys(4)=dqd2;
S function for control law and adaptive law:chap6_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 kv kp
sizes = simsizes;
sizes.NumContStates = 10;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 6;
sizes.NumInputs = 10;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = 0.1*ones(1,10);
str = [];
ts = [0 0];
```

```

c= [-2 -1 0 1 2;
      -2 -1 0 1 2;
      -2 -1 0 1 2;
      -2 -1 0 1 2];
b=3.0;
alfa=3;
kp=[alfa^2 0;
      0 alfa^2];
kv=[2*alfa 0;
      0 2*alfa];
function sys=mdlDerivatives(t,x,u)
global c b kv kp
A=[zeros(2) eye(2);
      -kp -kv];
B=[0 0;0 0;1 0;0 1];
Q=[50 0 0 0;
      0 50 0 0;
      0 0 50 0;
      0 0 0 50];
P=lyap(A',Q);
eig(P);
qd1=u(1);
qd2=u(2);
d_qd1=u(3);
d_qd2=u(4);
q1=u(5);dq1=u(6);q2=u(7);dq2=u(8);
e1=q1-qd1;
e2=q2-qd2;
de1=dq1-d_qd1;
de2=dq2-d_qd2;
w=[x(1) x(2) x(3) x(4) x(5);x(6) x(7) x(8) x(9) x(10)]';
xi=[e1;e2;de1;de2];
h=zeros(5,1);
for j=1:1:5
    h(j)=exp(-norm(xi-c(:,j))^2/(2*b^2));
end
gama=20;
S1=1;
if S1==1          % Adaptive Law
    dw=gama*h*xi'*P*B;
elseif S1==2      % Adaptive Law with UUB
    k1=0.001;
    dw=gama*h*xi'*P*B+k1*gama*norm(x)*w;
end

```

```
dw=dw' ;
for i=1:1:5
    sys(i)=dw(1,i) ;
    sys(i+5)=dw(2,i) ;
end

function sys=mdlOutputs(t,x,u)
global c b kv kp
qd1=u(1) ;
qd2=u(2) ;
d_qd1=u(3) ;
d_qd2=u(4) ;

dd_qd1=-0.2*(0.5*pi)^2*sin(0.5*pi*t) ;
dd_qd2=0.2*(0.5*pi)^2*cos(0.5*pi*t) ;
dd_qd=[dd_qd1;dd_qd2] ;

q1=u(5) ;dq1=u(6) ;q2=u(7) ;dq2=u(8) ;

ddq1=u(9) ;ddq2=u(10) ;
ddq=[ddq1;ddq2] ;

e1=q1-qd1;
e2=q2-qd2;
de1=dq1-d_qd1;
de2=dq2-d_qd2;
e=[e1;e2];
de=[de1;de2];

v=13.33;
q01=8.98;
q02=8.75;
g=9.8;

M0=[v+q01+2*q02*cos(q2) q01+q02*cos(q2) ;
     q01+q02*cos(q2) q01];
C0=[-q02*dq2*sin(q2) -q02*(dq1+dq2)*sin(q2) ;
     q02*dq1*sin(q2) 0];
G0=[15*g*cos(q1)+8.75*g*cos(q1+q2) ;
     8.75*g*cos(q1+q2) ];

dq=[dq1;dq2];
t011=M0*(dd_qd-kv*de-kp*e)+C0*dq+G0;

d_M=0.2*M0;
d_C=0.2*C0;
d_G=0.2*G0;
d1=2;d2=3;d3=6;
```

```

d=[d1+d2*norm([e1,e2])+d3*norm([de1,de2])];
%d=[20*sin(2*t);20*sin(2*t)];
f=inv(M0)*(d_M*ddq+d_C*dq+d_G+d);
xi=[e1;e2;de1;de2];
h=zeros(5,1);
for j=1:1:5
    h(j)=exp(-norm(xi-c(:,j))^2/(2*b^2));
end

S=3;
if S==1           %Nominal model based controller
    tol=tol1;
elseif S==2       %Modified computed torque controller
    tol2=-M0*f;
    tol=tol1+tol2;
elseif S==3       %RBF compensated controller
    w=[x(1) x(2) x(3) x(4) x(5);x(6) x(7) x(8) x(9)
        x(10)]';
    fn=w'*h;
    tol2=-M0*fn;
    tol=tol1+1*tol2;
end

sys(1)=tol(1);
sys(2)=tol(2);
sys(3)=f(1);
sys(4)=fn(1);
sys(5)=f(2);
sys(6)=fn(2);

S function for plant:chap6_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 = 4;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 4;
sizes.NumInputs = 6;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.6;0.3;0.5;0.5];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
persistent ddx1 ddx2
if t==0
    ddx1=0;
    ddx2=0;
end
qd1=1+0.2*sin(0.5*pi*t);
dqd1=0.2*0.5*pi*cos(0.5*pi*t);
qd2=1-0.2*cos(0.5*pi*t);
dqd2=0.2*0.5*pi*sin(0.5*pi*t);

e1=x(1)-qd1;
e2=x(3)-qd2;
de1=x(2)-dqd1;
de2=x(4)-dqd2;

v=13.33;
q1=8.98;
q2=8.75;
g=9.8;

M0=[v+q1+2*q2*cos(x(3)) q1+q2*cos(x(3));
     q1+q2*cos(x(3)) q1];
C0=[-q2*x(4)*sin(x(3))-q2*(x(2)+x(4))*sin(x(3));
     q2*x(2)*sin(x(3)) 0];
G0=[15*g*cos(x(1))+8.75*g*cos(x(1)+x(3));
     8.75*g*cos(x(1)+x(3))];
d_M=0.2*M0;
d_C=0.2*C0;
d_G=0.2*G0;

d1=2;d2=3;d3=6;
d=[d1+d2*norm([e1,e2])+d3*norm([de1,de2])];
%d=20*sin(2*t);
tol(1)=u(1);
tol(2)=u(2);
dq=[x(2);x(4)];
```

```
ddq=[ddx1;ddx2];
f=inv(M0)*(d_M*ddq+d_C*dq+d_G+d);
ddx=inv(M0)*(tol'-C0*dq-G0)+1*f;
sys(1)=x(2);
sys(2)=ddx(1);
sys(3)=x(4);
sys(4)=ddx(2);
ddx1=ddx(1);
ddx2=ddx(2);
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
sys(3)=x(3);
sys(4)=x(4);
Plot program:chap6_2plot.m
close all;

figure(1);
subplot(211);
plot(t,x(:,1),'r',t,x(:,5),'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking for link 1');
legend('ideal position for link 1','position tracking for link 1');
subplot(212);
plot(t,x(:,2),'r',t,x(:,7),'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking for link 2');
legend('ideal position for link 2','position tracking for link 2');

figure(2);
subplot(211);
plot(t,x(:,3),'r',t,x(:,6),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking for link 1');
legend('ideal speed for link 1','speed tracking for link 1');
subplot(212);
plot(t,x(:,4),'r',t,x(:,8),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking for link 2');
legend('ideal speed for link 2','speed tracking for link 2');

figure(3);
subplot(211);
plot(t,tol(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input of link 1');
subplot(212);
```

```

plot(t,tol(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input of link 2');

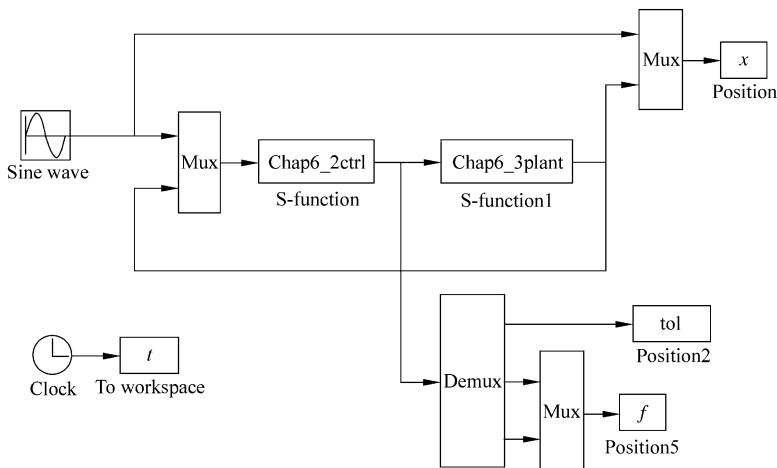
figure(4);
subplot(211);
plot(t,f(:,1),'r',t,f(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('f1 and fn1');
legend('Practical uncertainties of link 1','Estimation
uncertainties of link 1');

subplot(212);
plot(t,f(:,3),'r',t,f(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('f2 and fn2');
legend('Practical uncertainties of link 2','Estimation
uncertainties of link 2');

```

Programs for Sect. 6.2.4.1

Simulink main program: chap6_3sim.mdl



S function for control law and adaptive 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
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=15;

sizes = simsizes;
sizes.NumContStates = node;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = zeros(1,node);
str = [];
ts = [];
function sys=mdlDerivatives(t,x,u)
global node c b Fai
qd=u(1);
dqd=cos(t);
ddqd=-sin(t);

q=u(2);
dq=u(3);

e=qd-q;
de=dqd-dq;
r=de+Fai*e;

z=[e;de];
for j=1:1:node
    h(j)=exp(-norm(z-c(:,j))^2/(b*b));
end

Gama=100;
for i=1:1:node
    sys(i)=Gama*h(i)*r;
end

```

```
function sys=mdlOutputs(t,x,u)
global node c b Fai
qd=u(1);
dqd=cos(t);
ddqd=-sin(t);

q=u(2);
dq=u(3);

e=qd-q;
de=dqd-dq;
r=de+Fai*e;

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

z=[e;de];
w=[x(1:node)]';

for j=1:1:node
    h(j)=exp(-norm(z-c(:,j))^2/(b*b));
end

fn=w*h';
Kv=110;

epN=0.20;bd=0.1;
v=-(epN+bd)*sign(r);
tol=fn+Kv*r-v;

F=15*dq+0.3*sign(dq);
M=10;
f=M*ddqr+F;

fn_norm=norm(fn);
sys(1)=tol;
sys(2)=f;
sys(3)=fn;
S function for 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 = [];
end
```

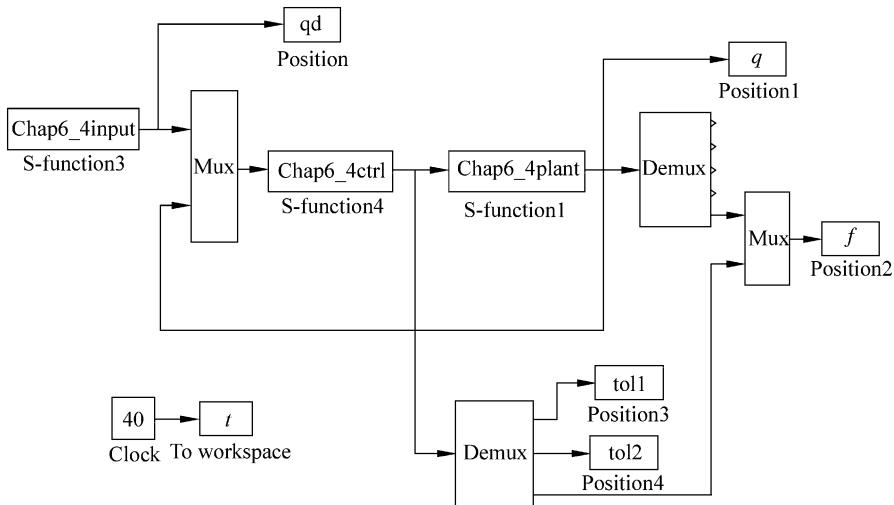
```
otherwise
    error(['Unhandled flag = ', num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.1;0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
M=10;
F=15*x(2)+0.30*sign(x(2));
tol=u(1);
sys(1)=x(2);
sys(2)=1/M*(tol-F);
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
Plot program:chap6_3plot.m
close all;

figure(1);
subplot(211);
plot(t,x(:,1),'r',t,x(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal position','position tracking');
subplot(212);
plot(t,cos(t),'r',t,x(:,3),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('ideal speed','speed tracking');

figure(2);
plot(t,f(:,1),'r',t,f(:,2),'b','linewidth',2);
xlabel('time(s)');ylabel('f and fn');
legend('Practical uncertainties','Estimation
uncertainties');
```

Programs for Sect. 6.2.4.2

Simulink main program: chap6_4sim.mdl



Tracking command program: chap6_4input.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;
S function for control law and adaptive law:chap6_4ctrl.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];
dqdr=[d_qd1;d_qd2];
dqr=dqdr+Fai*e;
ddqdr=[dd_qd1;dd_qd2];
ddqr=ddqdr+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];
dqrd=[d_qd1;d_qd2];
dqr=dqrd+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;
S function for plant:chap6_4plant.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);
dqdr=[d_qd1;d_qd2];
dqr=dqdr+Fai*e;
ddqdr=[dd_qd1;dd_qd2];
ddqr=ddqdr+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;
Plot program:chap6_4plot.m
close all;

figure(1);
subplot(211);
plot(t,qd(:,1),'r',t,q(:,1),'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking for link 1');
legend('ideal position for link 1','position tracking for link 1');
subplot(212);
plot(t,qd(:,4),'r',t,q(:,3),'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking for link 2');
legend('ideal position for link 2','position tracking for link 2');

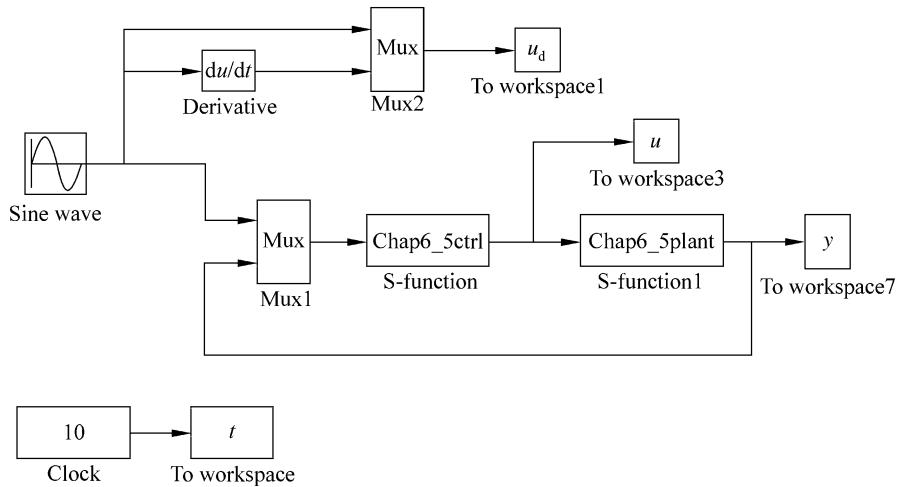
figure(2);
subplot(211);
plot(t,qd(:,2),'r',t,q(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking for link 1');
legend('ideal speed for link 1','speed tracking for link 1');
subplot(212);
plot(t,qd(:,5),'r',t,q(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking for link 2');
legend('ideal speed for link 2','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');
```

Programs for Sect. 6.3.3.1

Simulink main program: chap6_5sim.mdl



S function for control law and adaptive law: chap6_5ctrl.m

```

function [sys,x0,str,ts] = Robust_RBF(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 alfa
c=[-1 -0.5 0 0.5 1;
 -1 -0.5 0 0.5 1];
b=50*ones(5,1);
alfa=200;

sizes = simsizes;
sizes.NumContStates = 5;
sizes.NumDiscStates = 0;
  
```

```
sizes.NumOutputs = 2;
sizes.NumInputs = 4;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [zeros(5,1)];
str = [];
ts = [];
function sys=mdlDerivatives(t,x,u)
global c b alfa
qd=u(1);
dqd=cos(t);
ddqd=-sin(t);

q=u(2);dq=u(3);

e=q-qd;
de=dq-dqd;
x2=de+alfa*e;

xi=[e;de];
h=zeros(5,1);
for j=1:1:5
    h(j)=exp(-norm(xi-c(:,j))^2/(2*b(j)*b(j)));
end
% Adaptive Law
xite=1000;
S=-xite*x2*h';
for i=1:1:5
    sys(i)=S(i);
end
function sys=mdlOutputs(t,x,u)
global c b alfa

qd=u(1);
dqd=cos(t);
ddqd=-sin(t);

q=u(2);dq=u(3);

e=q-qd;
de=dq-dqd;
M=1.0;
w=M*alfa*de;
Gama=0.10;

xi=[e;de];
h=zeros(5,1);
```

```

for j=1:1:5
    h(j)=exp(-norm(xi-c(:,j))^2/(2*b(j)*b(j)));
end
Wf=[x(1) x(2) x(3) x(4) x(5)]';
x2=de+alfa*e;
S=2;
if S==1 %Without RBF compensation
    ut=-w-0.5*1/Gama^2*x2-0.5*x2;
elseif S==2 %With RBF compensation
    ut=-w+Wf'*h-0.5*1/Gama^2*x2-0.5*x2;
end
T=ut+M*ddqd;
NN=Wf'*h;
sys(1)=T;
sys(2)=NN;
S function for plant:chap6_5plant.m
function [sys,x0,str,ts]=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 = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.10 0];
str=[];

```

```
ts=[];
function sys=mdlDerivatives(t,x,u)
J=1.0;
d=150*sign(x(2))+10*x(2);
T=u(1);

sys(1)=x(2);
sys(2)=1/J*(T-d);
function sys=mdlOutputs(t,x,u)
d=150*sign(x(2))+10*x(2);
sys(1)=x(1);
sys(2)=x(2);
sys(3)=d;
Plot program:chap6_5plot.m
close all;

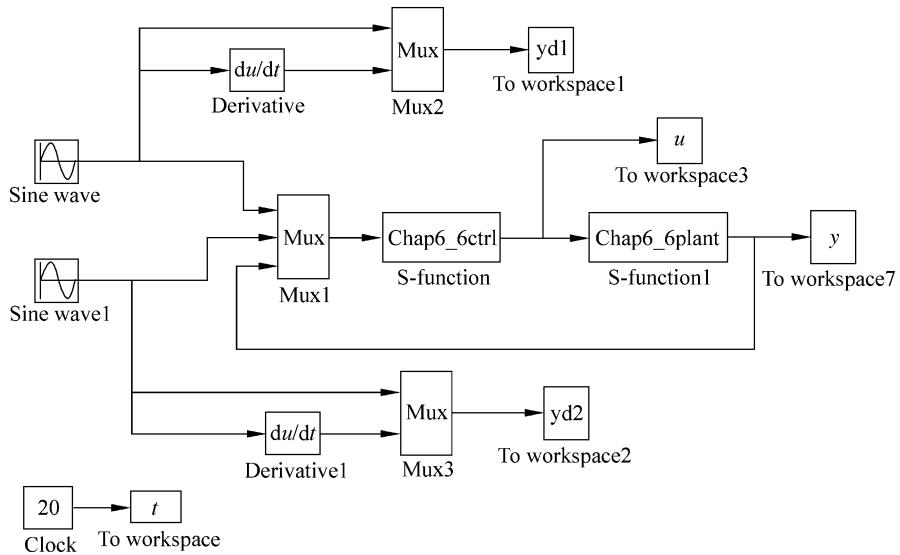
figure(1);
subplot(211);
plot(t,yd(:,1),'r',t,y(:,1),'k:','linewidth',2);
xlabel('time(s)');ylabel('position signal');
legend('ideal position signal','position tracking');
subplot(212);
plot(t,yd(:,2),'r',t,y(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('ideal speed signal','speed tracking');

figure(2);
subplot(211);
plot(t,yd(:,1)-y(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('position signal error');
legend('position tracking error');
subplot(212);
plot(t,yd(:,2)-y(:,2),'k','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking error');
legend('speed tracking error');

figure(3);
plot(t,y(:,3),'r',t,u(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('F and Estimated F');
legend('Practical Friction','Estimated Friction');
```

Programs for Sect. 6.3.3.2

Simulink main program: chap6_6sim.mdl



S function for control law and adaptive law: chap6_6ctrl.m

```

function [sys,x0,str,ts] = Robust_RBF(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 alfa
c=0.5*[-3 -2 -1 0 1 2 3;
          -3 -2 -1 0 1 2 3;
          -3 -2 -1 0 1 2 3;
          -3 -2 -1 0 1 2 3];
b=10*ones(7,1);
alfa=20;

```

```
sizes = simsizes;
sizes.NumContStates = 14;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 4;
sizes.NumInputs = 8;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [zeros(14,1)];
str = [];
ts = [];
function sys=mdlDerivatives(t,x,u)
global c b alfa

qd1=u(1);qd2=u(2);
dqd1=cos(t);dqd2=cos(t);
dqd=[dqd1 dqd2]';
ddqd1=-sin(t);ddqd2=-sin(t);
ddqd=[ddqd1 ddqd2]';

q1=u(3);dq1=u(4);
q2=u(5);dq2=u(6);

e1=q1-qd1;
e2=q2-qd2;
e=[e1 e2]';
de1=dq1-dqd1;
de2=dq2-dqd2;
de=[de1 de2]';
x2=de+alfa*e;

xi=[e1;e2;de1;de2];
%xi=[q1;dq1;q2;dq2];
h=zeros(7,1);
for j=1:1:7
    h(j)=exp(-norm(xi-c(:,j))^2/(2*b(j)*b(j)));
end

% Adaptive Law
xite=1500;
S=-xite*x2*h';
for i=1:1:7
    sys(i)=S(1,i);
    sys(i+7)=S(2,i);
end
function sys=mdlOutputs(t,x,u)
global c b alfa
```

```

qd1=u(1);qd2=u(2);
dqqd1=cos(t);dqqd2=cos(t);
dqd=[dqd1 dqd2]';
ddqd1=-sin(t);ddqd2=-sin(t);
ddqd=[ddqd1 ddqd2]';

q1=u(3);dq1=u(4);
q2=u(5);dq2=u(6);

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

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

M11=(m1+m2)*r1^2+m2*r2^2+2*m2*r1*r2*cos(x(3));
M22=m2*r2^2;
M21=m2*r2^2+m2*r1*r2*cos(x(3));
M12=M21;
M=[M11 M12;M21 M22];

V12=m2*r1*sin(x(3));
V=[-V12*x(4) -V12*(x(2)+x(4));V12*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];

w=M*alfa*de+V*alfa*e;
Gama=0.050;

xi=[e1;e2;de1;de2];
%xi=[q1;dq1;q2;dq2];
h=zeros(7,1);
for j=1:1:5
    h(j)=exp(-norm(xi-c(:,j))^2/(2*b(j)*b(j)));
end
Wf=[x(1) x(2) x(3) x(4) x(5) x(6) x(7);
     x(8) x(9) x(10) x(11) x(12) x(13) x(14)]';

S=2;
if S==1 %Without RBF compensation
    ut=-e-w-0.5*1/Gama^2*(de+alfa*e);
elseif S==2 %Without RBF compensation
    x2=de+alfa*e;
    ut=-w+Wf'*h-0.5*1/Gama^2*x2-0.5*x2;

```

```
end
T=ut+M*ddqd+V*dqd+G;

NN=Wf'*h;
sys(1)=T(1);
sys(2)=T(2);
sys(3)=NN(1);
sys(4)=NN(2);
S function for plant:chap6_6plant.m
function [sys,x0,str,ts]=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;

M11=(m1+m2)*r1^2+m2*r2^2+2*m2*r1*r2*cos(x(3));
M22=m2*r2^2;
M21=m2*r2^2+m2*r1*r2*cos(x(3));
M12=M21;
M=[M11 M12;M21 M22];

V12=m2*r1*sin(x(3));
V=[-V12*x(4) -V12*(x(2)+x(4));V12*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];
D=[10*x(2)+30*sign(x(2)) 10*x(4)+30*sign(x(4))]';
T=[u(1) u(2)]';
S=inv(M)*(T-V*[x(2);x(4)]-G-D);
sys(1)=x(2);
sys(2)=S(1);
sys(3)=x(4);
sys(4)=S(2);
function sys=mdlOutputs(t,x,u)
D=[10*x(2)+30*sign(x(2)) 10*x(4)+30*sign(x(4))]';
sys(1)=x(1);
sys(2)=x(2);
sys(3)=x(3);
sys(4)=x(4);
sys(5)=D(1);
sys(6)=D(2);
Plot program:chap6_6plot.m
close all;
figure(1);
subplot(211);
plot(t,yd1(:,1),'r',t,y(:,1),'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking of link 1');
legend('ideal position for link 1','position tracking for link 1');
subplot(212);
plot(t,yd2(:,1),'r',t,y(:,3),'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking of link 2');
legend('ideal position for link 2','position tracking for link 2');
figure(2);
subplot(211);
plot(t,yd1(:,2),'r',t,y(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking of link 2');
legend('ideal speed for link 1','speed tracking for link 1');
subplot(212);
plot(t,yd2(:,2),'r',t,y(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking of link 2');

```

```
legend('ideal speed for link 2','speed tracking for  
link 2');  
  
figure(3);  
subplot(211);  
plot(t,y(:,5),'r',t,u(:,3),'k:','linewidth',2);  
xlabel('time(s)');ylabel('d1');  
legend('ideal delta_f for link 1','estimation of delta_f  
for link1');  
subplot(212);  
plot(t,y(:,6),'r',t,u(:,4),'k:','linewidth',2);  
xlabel('time(s)');ylabel('d2');  
legend('ideal delta_f for link 2','estimation of delta_f  
for link2');
```

References

1. Miller TW, Hewes RP, Galnz FH, Kraft LG (1990) Real time dynamic control of an industrial manipulator using a neural network based learning controller. *IEEE Trans Robot Autom* 6 (1):1–9
2. Kuperstein M, Wang J (1990) Neural controller for adaptive movements with unforeseen payloads. *IEEE Trans Neural Netw* 1(1):137–142
3. Khosla PK, Kanade T (1989) Real-time implementation and evaluation of computed-torque scheme. *IEEE Trans Robot Autom* 5(2):245–253
4. Leahy MB Jr (1990) Model-based control of industrial manipulators: an experimental analysis. *J Robot Syst* 7(5):741–758
5. Zomaya AY, Nabhan TM (1993) Centralized and decentralized neuro-adaptive robot controllers. *Neural Netw* 6(2):223–244
6. Feng G (1995) A compensating scheme for robot tracking based on neural networks. *Robot Auton Syst* 15(3):199–206
7. Lewis FL, Liu K, Yesildirek A (1995) Neural net robot controller with guaranteed tracking performance. *IEEE Trans Neural Netw* 6(3):703–715
8. Schaft AJV (1992) L_2 gain analysis of nonlinear systems and nonlinear state feedback H_∞ control. *IEEE Trans Autom Control* 37(6):770–784
9. Wang Y, Sun W, Xiang Y, Miao S (2009) Neural network-based robust tracking control for robots. *Int J Intel Autom Soft Comput* 15(2):211–222

Chapter 7

Adaptive Robust RBF Control Based on Local Approximation

Abstract This chapter introduces three kinds of adaptive robust RBF controllers for robotic manipulators based on local approximation, including robust adaptive controller based on nominal model, adaptive controller based on local model approximation, and adaptive controller based on task space.

Keywords Neural network • Adaptive control • Local approximation • Robotic manipulators

GL matrix was proposed and applied to analyze adaptive neural network control system stability [1–4]. In this chapter, we use GL matrix to design adaptive controller for n-link manipulators based on RBF local approximation.

7.1 Robust Control Based on Nominal Model for Robotic Manipulators

7.1.1 Problem Description

Consider dynamic equation of n – link manipulator as

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} - \boldsymbol{\tau}_d \quad (7.1)$$

where $\mathbf{M}(\mathbf{q})$ is an $n \times n$ inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is an $n \times n$ matrix containing the centrifugal and Coriolis terms, $\mathbf{G}(\mathbf{q})$ is an $n \times 1$ vector containing gravitational forces and torques, \mathbf{q} is generalized joint coordinates, $\boldsymbol{\tau}$ is joint torques, and $\boldsymbol{\tau}_d$ denotes disturbances.

In practical engineering, $\mathbf{M}(\mathbf{q})$, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ and $\mathbf{G}(\mathbf{q})$ are often unknown, which can be expressed as

$$\begin{aligned}\mathbf{M}(\mathbf{q}) &= \mathbf{M}_0(\mathbf{q}) + \mathbf{E}_M \\ \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) &= \mathbf{C}_0(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{E}_C \\ \mathbf{G}(\mathbf{q}) &= \mathbf{G}_0(\mathbf{q}) + \mathbf{E}_G\end{aligned}$$

where \mathbf{E}_M , \mathbf{E}_C , and \mathbf{E}_G are modeling error of $\mathbf{M}(\mathbf{q})$, $\mathbf{C}_0(\mathbf{q}, \dot{\mathbf{q}})$, and $\mathbf{G}(\mathbf{q})$ respectively.

7.1.2 Controller Design

Define position tracking error as

$$\mathbf{e}(t) = \mathbf{q}_d(t) - \mathbf{q}(t)$$

where $\mathbf{q}_d(t)$ is ideal position signal and $\mathbf{q}(t)$ is practical position signal.

Define sliding mode function as

$$\mathbf{r} = \dot{\mathbf{e}} + \Lambda \mathbf{e} \quad (7.2)$$

where $\Lambda > 0$.

Define $\dot{\mathbf{q}}_r = \mathbf{r}(t) + \dot{\mathbf{q}}(t)$, then $\ddot{\mathbf{q}}_r = \dot{\mathbf{r}}(t) + \ddot{\mathbf{q}}(t)$, $\dot{\mathbf{q}}_r = \dot{\mathbf{q}}_d + \Lambda \mathbf{e}$, and $\ddot{\mathbf{q}}_r = \ddot{\mathbf{q}}_d + \Lambda \dot{\mathbf{e}}$. From (7.1), we have

$$\begin{aligned}\boldsymbol{\tau} &= \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \boldsymbol{\tau}_d \\ &= \mathbf{M}(\mathbf{q})(\ddot{\mathbf{q}}_r - \dot{\mathbf{r}}) + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})(\dot{\mathbf{q}}_r - \mathbf{r}) + \mathbf{G}(\mathbf{q}) + \boldsymbol{\tau}_d \\ &= \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_r + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}_r + \mathbf{G}(\mathbf{q}) - \mathbf{M}(\mathbf{q})\dot{\mathbf{r}} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{r} + \boldsymbol{\tau}_d \\ &= \mathbf{M}_0(\mathbf{q})\ddot{\mathbf{q}}_r + \mathbf{C}_0(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}_r + \mathbf{G}_0(\mathbf{q}) + \mathbf{E}' - \mathbf{M}(\mathbf{q})\dot{\mathbf{r}} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{r} + \boldsymbol{\tau}_d \quad (7.3)\end{aligned}$$

where $\mathbf{E}' = \mathbf{E}_M\ddot{\mathbf{q}}_r + \mathbf{E}_C\dot{\mathbf{q}}_r + \mathbf{E}_G$.

For the system, a controller was proposed as [1]

$$\boldsymbol{\tau} = \boldsymbol{\tau}_m + \mathbf{K}_p \mathbf{r} + \mathbf{K}_i \int \mathbf{r} dt + \boldsymbol{\tau}_r \quad (7.4)$$

where $\mathbf{K}_p > 0$, $\mathbf{K}_i > 0$, $\boldsymbol{\tau}_m$ denotes control term based on nominal model, $\boldsymbol{\tau}_r$ denotes robust term, and

$$\boldsymbol{\tau}_m = \mathbf{M}_0(\mathbf{q})\ddot{\mathbf{q}}_r + \mathbf{C}_0(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}_r + \mathbf{G}_0(\mathbf{q}) \quad (7.5)$$

$$\boldsymbol{\tau}_r = \mathbf{K}_r \text{sgn}(\mathbf{r}) \quad (7.6)$$

where $\mathbf{K}_r = \text{diag}[k_{r_{ii}}]$, $k_{r_{ii}} \geq |\mathbf{E}_i|$, $i = 1, \dots, n$, and $\mathbf{E} = \mathbf{E}' + \boldsymbol{\tau}_d$.

From (7.3)–(7.6), we have

$$\begin{aligned} \mathbf{M}_0(\mathbf{q})\ddot{\mathbf{q}}_r + \mathbf{C}_0(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}_r + \mathbf{G}_0(\mathbf{q}) - \mathbf{M}(\mathbf{q})\dot{\mathbf{r}} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{r} + \mathbf{E}' + \boldsymbol{\tau}_r \\ = \mathbf{M}_0(\mathbf{q})\ddot{\mathbf{q}}_r + \mathbf{C}_0(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}_r + \mathbf{G}_0(\mathbf{q}) + \mathbf{K}_p\mathbf{r} + \mathbf{K}_i \int_0^t \mathbf{r} dt + \mathbf{K}_r \text{sgn}(\mathbf{r}). \end{aligned}$$

Thus,

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{r}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{r} + \mathbf{K}_i \int_0^t \mathbf{r} dt = -\mathbf{K}_p\mathbf{r} - \mathbf{K}_r \text{sgn}(\mathbf{r}) + \mathbf{E}. \quad (7.7)$$

7.1.3 Stability Analysis

An integration-type Lyapunov function is designed as [1]

$$V = \frac{1}{2}\mathbf{r}^T \mathbf{M} \mathbf{r} + \frac{1}{2} \left(\int_0^t \mathbf{r} d\tau \right)^T \mathbf{K}_i \left(\int_0^t \mathbf{r} d\tau \right), \quad (7.8)$$

Then

$$\dot{V} = \mathbf{r}^T \left[\mathbf{M} \dot{\mathbf{r}} + \frac{1}{2} \dot{\mathbf{M}} \mathbf{r} + \mathbf{K}_i \int_0^t \mathbf{r} d\tau \right].$$

Considering the skew-symmetric characteristics of manipulator dynamic equation, $\mathbf{r}^T (\dot{\mathbf{M}} - 2\mathbf{C})\mathbf{r} = 0$, we have

$$\dot{V} = \mathbf{r}^T \left[\mathbf{M} \dot{\mathbf{r}} + \mathbf{C} \mathbf{r} + \mathbf{K}_i \int_0^t \mathbf{r} d\tau \right].$$

Substituting (7.7) into above, we have

$$\begin{aligned} \dot{V} &= -\mathbf{r}^T \mathbf{K}_p \mathbf{r} - \mathbf{r}^T \mathbf{K}_r \text{sgn}(\mathbf{r}) + \mathbf{r}^T \mathbf{E} \\ &= -\mathbf{r}^T \mathbf{K}_p \mathbf{r} - \sum_{i=1}^n K_{rii} |r|_i + \mathbf{r}^T \mathbf{E}. \end{aligned}$$

Considering $k_{rii} \geq |E_i|$, then

$$\dot{V} \leq -\mathbf{r}^T \mathbf{K}_p \mathbf{r} \leq 0.$$

Hence,

$$\lambda_{\min}(\mathbf{K}_p) \int_0^t \mathbf{r}^T \mathbf{r} \leq \int_0^t \mathbf{r}^T \mathbf{K}_p \mathbf{r} \leq V(0)$$

where $\lambda_{\min}(\mathbf{K}_p)$ is the minimum eigenvalue of \mathbf{K}_p .

Since $V(0)$ and $\lambda_{\min}(\mathbf{K}_p)$ are positive constants, it follows that $\mathbf{r} \in L_2^n$. Consequently, $\mathbf{e} \in L_2^n \cap L_\infty^n$, \mathbf{e} is continuous, $\mathbf{e} \rightarrow 0$ as $t \rightarrow \infty$, and $\dot{\mathbf{e}} \in L_2^n$.

Furthermore, since $\dot{V} \leq -\mathbf{r}^T \mathbf{K}_p \mathbf{r} \leq 0$, it follows that $0 \leq V \leq V(0)$, $\forall t \geq 0$. Hence, $V(t) \in L_\infty$ implies that $\int_0^t \mathbf{r} d\tau$ is bounded.

From $\mathbf{e} \in L_2^n \cap L_\infty^n$, $\dot{\mathbf{e}} \in L_2^n$ and $\dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d \in L_\infty^n$, we can conclude that $\dot{\mathbf{q}}_r \in L_\infty^n$ and $\ddot{\mathbf{q}}_r \in L_\infty^n$, by observing that $\mathbf{r} \in L_2^n$, and from $\mathbf{q}_d, \boldsymbol{\tau}_r \in L_\infty^n$, we can conclude that $\dot{\mathbf{r}} \in L_\infty^n$ from (7.7) and $\boldsymbol{\tau} \in L_\infty^n$ from (7.4).

Using the fact that $\mathbf{r} \in L_2^n$ and $\dot{\mathbf{r}} \in L_\infty^n$, thus $\mathbf{r} \rightarrow 0$ as $t \rightarrow \infty$. Hence, $\dot{\mathbf{e}} \rightarrow 0$ as $t \rightarrow \infty$. This completes the proof.

The shortcoming of the controller is that nominal model of the robotic manipulators must be known, and big uncertainties will need big k_{rii} , which will deduce big control chattering. To overcome this problem, we can use RBF neural network.

7.1.4 Simulation Example

The dynamic equation of two-link manipulator is

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} - \boldsymbol{\tau}_d$$

where

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} p_1 + p_2 + 2p_3 \cos q_2 & p_2 + p_3 \cos q_2 \\ p_2 + p_3 \cos q_2 & p_2 \end{bmatrix}$$

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} -p_3 \dot{q}_2 \sin q_2 & -p_3 (\dot{q}_1 + \dot{q}_2) \sin q_2 \\ p_3 \dot{q}_1 \sin q_2 & 0 \end{bmatrix}$$

$$\mathbf{G}(\mathbf{q}) = \begin{bmatrix} p_4 g \cos q_1 + p_5 g \cos(q_1 + q_2) \\ p_5 g \cos(q_1 + q_2) \end{bmatrix}, \quad \boldsymbol{\tau}_d = 20 \operatorname{sgn}(\dot{\mathbf{q}}).$$

Choose $\mathbf{p} = [2.90 \ 0.76 \ 0.87 \ 3.04 \ 0.87]^T$, the initial states of the plant are $\mathbf{q}_0 = [0.09 \ -0.09]^T$ and $\dot{\mathbf{q}}_0 = [0.0 \ 0.0]^T$, and consider $\mathbf{M}_0 = 0.8\mathbf{M}$, $\mathbf{C}_0 = 0.8\mathbf{C}$, $\mathbf{G}_0 = 0.8\mathbf{G}$.

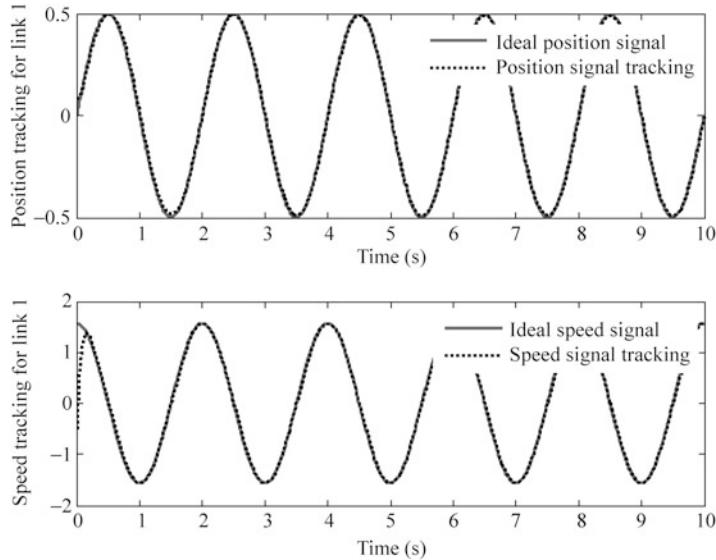


Fig. 7.1 Position and speed tracking for link 1

The desired trajectory is $q_{d1} = 0.5 \sin(\pi t)$ and $q_{d2} = \sin(\pi t)$. In simulation, we use control law (7.4), (7.5), and (7.6), and the parameters are chosen as $\mathbf{K}_p = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$, $\mathbf{K}_i = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$, $\mathbf{K}_r = \begin{bmatrix} 15 & 0 \\ 0 & 15 \end{bmatrix}$, and $\Lambda = \begin{bmatrix} 5.0 & 0 \\ 0 & 5.0 \end{bmatrix}$. The simulation results are shown in Figs. 7.1 and 7.2.

The Simulink program of this example is chap7_1sim.mdl; the Matlab programs of the example are given in the [Appendix](#).

7.2 Adaptive RBF Control Based on Local Model Approximation for Robotic Manipulators

7.2.1 Problem Description

Consider dynamic equation of n – link manipulator as

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} \quad (7.9)$$

where $\mathbf{M}(\mathbf{q})$ is an $n \times n$ inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is an $n \times n$ matrix containing the centrifugal and Coriolis terms, $\mathbf{G}(\mathbf{q})$ is an $n \times 1$ vector containing gravitational forces and torques, \mathbf{q} denotes angle signal, and $\boldsymbol{\tau}$ is joint torques.

In practical engineering, $\mathbf{M}(\mathbf{q})$, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$, and $\mathbf{G}(\mathbf{q})$ are always unknown. In this section, we use three kinds of RBF to model $\mathbf{M}(\mathbf{q})$, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$, and $\mathbf{G}(\mathbf{q})$, respectively.

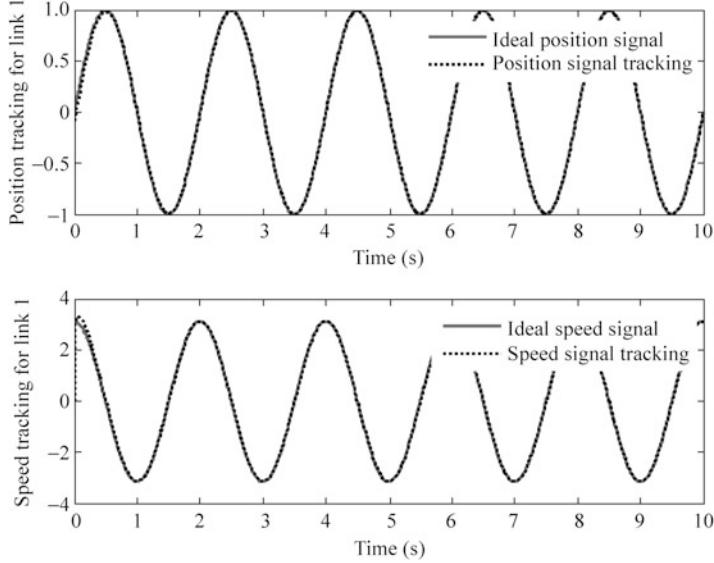


Fig. 7.2 Position and speed tracking for link 2

We assume the outputs of ideal RBF neural network are $\mathbf{M}_{\text{SNN}}(\mathbf{q})$, $\mathbf{C}_{\text{DNN}}(\mathbf{q}, \dot{\mathbf{q}})$, and $\mathbf{G}_{\text{SNN}}(\mathbf{q})$, respectively, that is,

$$\mathbf{M}(\mathbf{q}) = \mathbf{M}_{\text{SNN}}(\mathbf{q}) + \mathbf{E}_M \quad (7.10)$$

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{C}_{\text{DNN}}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{E}_C \quad (7.11)$$

$$\mathbf{G}(\mathbf{q}) = \mathbf{G}_{\text{SNN}}(\mathbf{q}) + \mathbf{E}_G \quad (7.12)$$

where \mathbf{E}_M , \mathbf{E}_C , and \mathbf{E}_G are modeling error of $\mathbf{M}(\mathbf{q})$, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$, and $\mathbf{G}(\mathbf{q})$, respectively. Then we have

$$\begin{aligned} \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_r + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}_r + \mathbf{G}(\mathbf{q}) &= \mathbf{M}_{\text{SNN}}(\mathbf{q})\ddot{\mathbf{q}}_r + \mathbf{C}_{\text{DNN}}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}_r + \mathbf{G}(\mathbf{q}) + \mathbf{E} \\ &= \left[\{\mathbf{W}_M\}^T \cdot \{\Xi_M(\mathbf{q})\} \right] \ddot{\mathbf{q}}_r + \left[\{\mathbf{W}_C\}^T \cdot \{\Xi_C(\mathbf{z})\} \right] \dot{\mathbf{q}}_r \\ &\quad + \left[\{\mathbf{W}_G\}^T \cdot \{\Xi_G(\mathbf{q})\} \right] + \mathbf{E} \end{aligned} \quad (7.13)$$

where \mathbf{W}_M , \mathbf{W}_C , and \mathbf{W}_G are ideal weight value of RBF; Ξ_M , Ξ_C , and Ξ_G are output of hidden layer; and $\mathbf{E} = \mathbf{E}_M\ddot{\mathbf{q}}_r + \mathbf{E}_C\dot{\mathbf{q}}_r + \mathbf{E}_G$.

The estimates of $\mathbf{M}_{\text{SNN}}(\mathbf{q})$, $\mathbf{C}_{\text{DNN}}(\mathbf{q}, \dot{\mathbf{q}})$, and $\mathbf{G}_{\text{SNN}}(\mathbf{q})$ can be expressed by RBF as

$$\hat{\mathbf{M}}_{\text{SNN}}(\mathbf{q}) = \left[\{\hat{\mathbf{W}}_M\}^T \cdot \{\Xi_M(\mathbf{q})\} \right] \quad (7.14)$$

$$\hat{\mathbf{C}}_{\text{DNN}}(\mathbf{q}, \dot{\mathbf{q}}) = \left[\{\hat{\mathbf{W}}_C\}^T \cdot \{\Xi_C(z)\} \right] \quad (7.15)$$

$$\hat{\mathbf{G}}_{\text{SNN}}(\mathbf{q}) = \left[\{\hat{\mathbf{W}}_G\}^T \cdot \{\Xi_G(\mathbf{q})\} \right] \quad (7.16)$$

where $\{\hat{\mathbf{W}}_M\}$, $\{\hat{\mathbf{W}}_C\}$, and $\{\hat{\mathbf{W}}_G\}$ are estimates of $\{\mathbf{W}_M\}$, $\{\mathbf{W}_C\}$, and $\{\mathbf{W}_G\}$, respectively, and $z = [\mathbf{q}^T \quad \dot{\mathbf{q}}^T]^T$.

7.2.2 Controller Design

Define

$$\mathbf{e}(t) = \mathbf{q}_d(t) - \mathbf{q}(t) \quad (7.17)$$

$$\dot{\mathbf{q}}_r = \mathbf{r}(t) + \dot{\mathbf{q}}(t) \quad (7.18)$$

$$\ddot{\mathbf{q}}_r = \dot{\mathbf{r}}(t) + \ddot{\mathbf{q}}(t) \quad (7.19)$$

where $\mathbf{q}_d(t)$ is ideal position signal and $\mathbf{q}(t)$ is practical position signal.

Define

$$\mathbf{r} = \dot{\mathbf{e}} + \Lambda \mathbf{e}. \quad (7.20)$$

Then we have $\dot{\mathbf{q}}_r = \dot{\mathbf{q}}_d + \Lambda \mathbf{e}$ and $\ddot{\mathbf{q}}_r = \ddot{\mathbf{q}}_d + \Lambda \dot{\mathbf{e}}$, $\Lambda > 0$.

Submitting (7.18) and (7.19) into (7.9), we have

$$\begin{aligned} \boldsymbol{\tau} &= \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \\ &= \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_r + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}_r + \mathbf{G}(\mathbf{q}) - \mathbf{M}(\mathbf{q})\dot{\mathbf{r}} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{r} \\ &= \left[\{\mathbf{W}_M\}^T \cdot \{\Xi_M(\mathbf{q})\} \right] \ddot{\mathbf{q}}_r + \left[\{\mathbf{W}_C\}^T \cdot \{\Xi_C(z)\} \right] \dot{\mathbf{q}}_r + \left[\{\mathbf{W}_G\}^T \cdot \{\Xi_G(\mathbf{q})\} \right] \\ &\quad - \mathbf{M}(\mathbf{q})\dot{\mathbf{r}} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{r} + \mathbf{E}. \end{aligned} \quad (7.21)$$

For the system, the controller is proposed as [1]

$$\begin{aligned} \boldsymbol{\tau} &= \boldsymbol{\tau}_m + \mathbf{K}_p \mathbf{r} + \mathbf{K}_i \int \mathbf{r} dt + \boldsymbol{\tau}_r \\ &= \hat{\mathbf{M}}_{\text{SNN}}(\mathbf{q})\ddot{\mathbf{q}}_r + \hat{\mathbf{C}}_{\text{DNN}}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}_r + \hat{\mathbf{G}}_{\text{SNN}}(\mathbf{q}) + \mathbf{K}_p \mathbf{r} + \mathbf{K}_i \int \mathbf{r} dt + \boldsymbol{\tau}_r \\ &= \left[\{\hat{\mathbf{W}}_M\}^T \cdot \{\Xi_M(\mathbf{q})\} \right] \ddot{\mathbf{q}}_r + \left[\{\hat{\mathbf{W}}_C\}^T \cdot \{\Xi_C(z)\} \right] \dot{\mathbf{q}}_r + \left[\{\hat{\mathbf{W}}_G\}^T \cdot \{\Xi_G(\mathbf{q})\} \right] \\ &\quad + \mathbf{K}_p \mathbf{r} + \mathbf{K}_i \int \mathbf{r} dt + \boldsymbol{\tau}_r \end{aligned} \quad (7.22)$$

where $\mathbf{K}_p > 0$ and $\mathbf{K}_i > 0$.

The model-estimated control law is designed as

$$\boldsymbol{\tau}_m = \hat{\mathbf{M}}_{SNN}(\mathbf{q})\ddot{\mathbf{q}}_r + \hat{\mathbf{C}}_{DNN}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}_r + \hat{\mathbf{G}}_{SNN}(\mathbf{q}). \quad (7.23)$$

The robust term is designed as

$$\boldsymbol{\tau}_r = \mathbf{K}_r \text{sgn}(\mathbf{r}) \quad (7.24)$$

where $\mathbf{K}_r = \text{diag}[k_{rii}]$, and $k_{rii} \geq |E_i|$.

From (7.21) and (7.22), we have

$$\begin{aligned} \mathbf{M}(\mathbf{q})\dot{\mathbf{r}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{r} + \mathbf{K}_p\mathbf{r} + \mathbf{K}_I \int_0^t \mathbf{r} dt + \boldsymbol{\tau}_r &= \left[\{\tilde{\mathbf{W}}_M\}^T \cdot \{\Xi_M(\mathbf{q})\} \right] \ddot{\mathbf{q}}_r \\ &\quad + \left[\{\tilde{\mathbf{W}}_C\}^T \cdot \{\Xi_C(z)\} \right] \dot{\mathbf{q}}_r \\ &\quad + \left[\{\tilde{\mathbf{W}}_G\}^T \cdot \{\Xi_G(\mathbf{q})\} \right] + \mathbf{E} \end{aligned} \quad (7.25)$$

where $\tilde{\mathbf{W}}_M = \mathbf{W}_M - \hat{\mathbf{W}}_M$, $\tilde{\mathbf{W}}_C = \mathbf{W}_C - \hat{\mathbf{W}}_C$, and $\tilde{\mathbf{W}}_G = \mathbf{W}_G - \hat{\mathbf{W}}_G$.

The Eq. (7.25) can be rewritten as

$$\begin{aligned} \mathbf{M}(\mathbf{q})\dot{\mathbf{r}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{r} + \mathbf{K}_I \int_0^t \mathbf{r} dt &= -\mathbf{K}_p\mathbf{r} - \mathbf{K}_r \text{sgn}(\mathbf{r}) \\ &\quad + \left[\{\tilde{\mathbf{W}}_M\}^T \cdot \{\Xi_M(\mathbf{q})\} \right] \ddot{\mathbf{q}}_r + \left[\{\tilde{\mathbf{W}}_C\}^T \cdot \{\Xi_C(z)\} \right] \dot{\mathbf{q}}_r \\ &\quad + \left[\{\tilde{\mathbf{W}}_G\}^T \cdot \{\Xi_G(\mathbf{q})\} \right] + \mathbf{E}. \end{aligned} \quad (7.26)$$

The adaptive law is designed as [1]

$$\dot{\tilde{\mathbf{W}}}_{Mk} = \boldsymbol{\Gamma}_{Mk} \cdot \{\xi_{Mk}(\mathbf{q})\} \ddot{\mathbf{q}}_r \mathbf{r}_k \quad (7.27)$$

$$\dot{\tilde{\mathbf{W}}}_{Ck} = \boldsymbol{\Gamma}_{Ck} \cdot \{\xi_{Ck}(z)\} \dot{\mathbf{q}}_r \mathbf{r}_k \quad (7.28)$$

$$\dot{\tilde{\mathbf{W}}}_{Gk} = \boldsymbol{\Gamma}_{Gk} \cdot \{\xi_{Gk}(\mathbf{q})\} \mathbf{r}_k \quad (7.29)$$

where $k = 1, 2, \dots, n$.

7.2.3 Stability Analysis

An integration-type Lyapunov function is designed as [1]

$$\begin{aligned} V = & \frac{1}{2} \mathbf{r}^T \mathbf{M} \mathbf{r} + \frac{1}{2} \left(\int_0^t \mathbf{r} d\tau \right)^T \mathbf{K}_I \left(\int_0^t \mathbf{r} d\tau \right) + \frac{1}{2} \sum_{k=1}^n \tilde{\mathbf{W}}_{Mk}^T \boldsymbol{\Gamma}_{Mk}^{-1} \tilde{\mathbf{W}}_{Mk} \\ & + \frac{1}{2} \sum_{k=1}^n \tilde{\mathbf{W}}_{Ck}^T \boldsymbol{\Gamma}_{Ck}^{-1} \tilde{\mathbf{W}}_{Ck} + \frac{1}{2} \sum_{k=1}^n \tilde{\mathbf{W}}_{Gk}^T \boldsymbol{\Gamma}_{Gk}^{-1} \tilde{\mathbf{W}}_{Gk} \end{aligned} \quad (7.30)$$

where $\boldsymbol{\Gamma}_{Mk}$, $\boldsymbol{\Gamma}_{Ck}$, and $\boldsymbol{\Gamma}_{Gk}$ are symmetric positive-definite constant matrices.

The derivative is given by

$$\begin{aligned} \dot{V} = & \mathbf{r}^T \left[\mathbf{M} \dot{\mathbf{r}} + \frac{1}{2} \dot{\mathbf{M}} \mathbf{r} + \mathbf{K}_I \int_0^t \mathbf{r} d\tau \right] + \sum_{k=1}^n \tilde{\mathbf{W}}_{Mk}^T \boldsymbol{\Gamma}_{Mk}^{-1} \dot{\tilde{\mathbf{W}}}_{Mk} + \sum_{k=1}^n \tilde{\mathbf{W}}_{Ck}^T \boldsymbol{\Gamma}_{Ck}^{-1} \dot{\tilde{\mathbf{W}}}_{Ck} \\ & + \sum_{k=1}^n \tilde{\mathbf{W}}_{Gk}^T \boldsymbol{\Gamma}_{Gk}^{-1} \dot{\tilde{\mathbf{W}}}_{Gk}. \end{aligned}$$

Considering the skew-symmetric characteristics of manipulator dynamic equation, $\mathbf{r}^T (\dot{\mathbf{M}} - 2\mathbf{C}) \mathbf{r} = 0$, we have

$$\begin{aligned} \dot{V} = & \mathbf{r}^T \left[\mathbf{M} \dot{\mathbf{r}} + \mathbf{C} \mathbf{r} + \mathbf{K}_I \int_0^t \mathbf{r} d\tau \right] + \sum_{k=1}^n \tilde{\mathbf{W}}_{Mk}^T \boldsymbol{\Gamma}_{Mk}^{-1} \dot{\tilde{\mathbf{W}}}_{Mk} + \sum_{k=1}^n \tilde{\mathbf{W}}_{Ck}^T \boldsymbol{\Gamma}_{Ck}^{-1} \dot{\tilde{\mathbf{W}}}_{Ck} \\ & + \sum_{k=1}^n \tilde{\mathbf{W}}_{Gk}^T \boldsymbol{\Gamma}_{Gk}^{-1} \dot{\tilde{\mathbf{W}}}_{Gk}. \end{aligned}$$

Submitting (7.26) into above, we have

$$\begin{aligned} \dot{V} = & -\mathbf{r}^T \mathbf{K}_p \mathbf{r} - \mathbf{r}^T \mathbf{K}_r \text{sgn}(\mathbf{r}) + \mathbf{r}^T \left[\{\tilde{\mathbf{W}}_M\}^T \cdot \{\boldsymbol{\Xi}_M\} \right] \ddot{\mathbf{q}}_r \\ & + \mathbf{r}^T \left[\{\tilde{\mathbf{W}}_C\}^T \cdot \{\boldsymbol{\Xi}_C\} \right] \dot{\mathbf{q}}_r + \mathbf{r}^T \left[\{\tilde{\mathbf{W}}_G\}^T \cdot \{\boldsymbol{\Xi}_G\} \right] + \mathbf{r}^T \mathbf{E} \\ & + \sum_{k=1}^n \tilde{\mathbf{W}}_{Mk}^T \boldsymbol{\Gamma}_{Mk}^{-1} \dot{\tilde{\mathbf{W}}}_{Mk} + \sum_{k=1}^n \tilde{\mathbf{W}}_{Ck}^T \boldsymbol{\Gamma}_{Ck}^{-1} \dot{\tilde{\mathbf{W}}}_{Ck} + \sum_{k=1}^n \tilde{\mathbf{W}}_{Gk}^T \boldsymbol{\Gamma}_{Gk}^{-1} \dot{\tilde{\mathbf{W}}}_{Gk} \end{aligned}$$

since

$$\begin{aligned} \mathbf{r}^T [\{\tilde{\mathbf{W}}_M\}^T \cdot \{\boldsymbol{\Xi}_M\}] \ddot{\mathbf{q}}_r &= [\mathbf{r}_1 \quad \mathbf{r}_2 \quad \dots \quad \mathbf{r}_n] \begin{bmatrix} \{\tilde{\mathbf{W}}_{M1}\}^T \cdot \{\boldsymbol{\xi}_{M1}\} \ddot{\mathbf{q}}_r \\ \{\tilde{\mathbf{W}}_{M2}\}^T \cdot \{\boldsymbol{\xi}_{M2}\} \ddot{\mathbf{q}}_r \\ \vdots \\ \{\tilde{\mathbf{W}}_{Mn}\}^T \cdot \{\boldsymbol{\xi}_{Mn}\} \ddot{\mathbf{q}}_r \end{bmatrix} \\ &= \sum_{k=1}^n \{\tilde{\mathbf{W}}_{Mk}\}^T \cdot \{\boldsymbol{\xi}_{Mk}\} \ddot{\mathbf{q}}_r \mathbf{r}_k. \end{aligned}$$

For the same reason, we have

$$\begin{aligned} \mathbf{r}^T \left[\{\tilde{\mathbf{W}}_C\}^T \cdot \{\boldsymbol{\Xi}_C\} \right] \ddot{\mathbf{q}}_r &= \sum_{k=1}^n \{\tilde{\mathbf{W}}_{Ck}\}^T \cdot \{\boldsymbol{\xi}_{Ck}\} \ddot{\mathbf{q}}_r \mathbf{r}_k \\ \mathbf{r}^T \left[\{\tilde{\mathbf{W}}_G\}^T \cdot \{\boldsymbol{\Xi}_G\} \right] &= \sum_{k=1}^n \tilde{\mathbf{W}}_{Gk}^T \cdot \boldsymbol{\xi}_{Gk} \mathbf{r}_k. \end{aligned}$$

Thus,

$$\begin{aligned} \dot{V} &= -\mathbf{r}^T \mathbf{K}_p \mathbf{r} + \mathbf{r}^T \mathbf{E} - \mathbf{r}^T \mathbf{K}_r \text{sgn}(\mathbf{r}) + \sum_{k=1}^n \{\tilde{\mathbf{W}}_{Dk}\}^T \cdot \{\boldsymbol{\xi}_{Dk}\} \ddot{\mathbf{q}}_r \mathbf{r}_k \\ &\quad + \sum_{k=1}^n \{\tilde{\mathbf{W}}_{Ck}\} \cdot \{\boldsymbol{\xi}_{Ck}\} \dot{\mathbf{q}}_r \mathbf{r}_k + \sum_{k=1}^n \tilde{\mathbf{W}}_{Gk} \cdot \boldsymbol{\xi}_{Gk} \mathbf{r}_k \\ &\quad + \sum_{k=1}^n \tilde{\mathbf{W}}_{Dk}^T \Gamma_{Dk}^{-1} \dot{\tilde{\mathbf{W}}}_{Dk} + \sum_{k=1}^n \tilde{\mathbf{W}}_{Ck}^T \Gamma_{Ck}^{-1} \dot{\tilde{\mathbf{W}}}_{Ck} + \sum_{k=1}^n \tilde{\mathbf{W}}_{Gk}^T \Gamma_{Gk}^{-1} \dot{\tilde{\mathbf{W}}}_{Gk} \end{aligned}$$

since

$$\dot{\tilde{\mathbf{W}}}_{Mk} = -\dot{\tilde{\mathbf{W}}}_{Mk}, \quad \dot{\tilde{\mathbf{W}}}_{Ck} = -\dot{\tilde{\mathbf{W}}}_{Ck}, \quad \text{and} \quad \dot{\tilde{\mathbf{W}}}_{Gk} = -\dot{\tilde{\mathbf{W}}}_{Gk}$$

Inserting adaptive law (7.27), (7.28), and (7.29) into above and considering $k_{rii} \geq |E_i|$, we have

$$\dot{V} = -\mathbf{r}^T \mathbf{K}_p \mathbf{r} + \mathbf{r}^T \mathbf{E} - \mathbf{r}^T \mathbf{K}_r \text{sgn}(\mathbf{r}) \leq -\mathbf{r}^T \mathbf{K}_p \mathbf{r} \leq 0.$$

Hence,

$$\lambda_{\min}(\mathbf{K}_p) \int_0^t \mathbf{r}^T \mathbf{r} dt \leq \int_0^t \mathbf{r}^T \mathbf{K}_p \mathbf{r} dt \leq V(0)$$

where $\lambda_{\min}(\mathbf{K}_p)$ is the minimum eigenvalue of \mathbf{K}_p .

Since $V(0)$ and $\lambda_{\min}(\mathbf{K}_p)$ are positive constants, it follows that $\mathbf{r} \in L_2^n$. Consequently, $\mathbf{e} \in L_2^n \cap L_\infty^n$, \mathbf{e} is continuous, $\mathbf{e} \rightarrow 0$ as $t \rightarrow \infty$, and $\dot{\mathbf{e}} \in L_2^n$.

Furthermore, since $\dot{V} \leq -\mathbf{r}^T \mathbf{K}_p \mathbf{r} \leq 0$, it follows that $0 \leq V \leq V(0)$, $\forall t \geq 0$. Hence, $V(t) \in L_\infty$ implies that $\int_0^t \mathbf{r} d\tau$, $\tilde{\mathbf{W}}_{Mk}$, $\tilde{\mathbf{W}}_{Ck}$, and $\tilde{\mathbf{W}}_{Gk}$ are bounded, that is, $\dot{\tilde{\mathbf{W}}}_{Mk}$, $\dot{\tilde{\mathbf{W}}}_{Ck}$ and $\dot{\tilde{\mathbf{W}}}_{Gk}$ are bounded.

From $\mathbf{e} \in L_2^n \cap L_\infty^n$, $\dot{\mathbf{e}} \in L_2^n$, and $\dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d \in L_\infty^n$, we can conclude that $\dot{\mathbf{q}}_r \in L_\infty^n$ and $\ddot{\mathbf{q}}_r \in L_\infty^n$, by observing that $\mathbf{r} \in L_2^n$, and from $\mathbf{q}_d, \boldsymbol{\tau}_r \in L_\infty^n$, we can conclude that $\dot{\mathbf{r}} \in L_\infty^n$ from (7.26) and $\boldsymbol{\tau} \in L_\infty^n$ from (7.22).

Using the fact that $\mathbf{r} \in L_2^n$ and $\dot{\mathbf{r}} \in L_\infty^n$, thus $\mathbf{r} \rightarrow 0$ as $t \rightarrow \infty$. Hence $\dot{\mathbf{e}} \rightarrow 0$ as $t \rightarrow \infty$. This completes the proof.

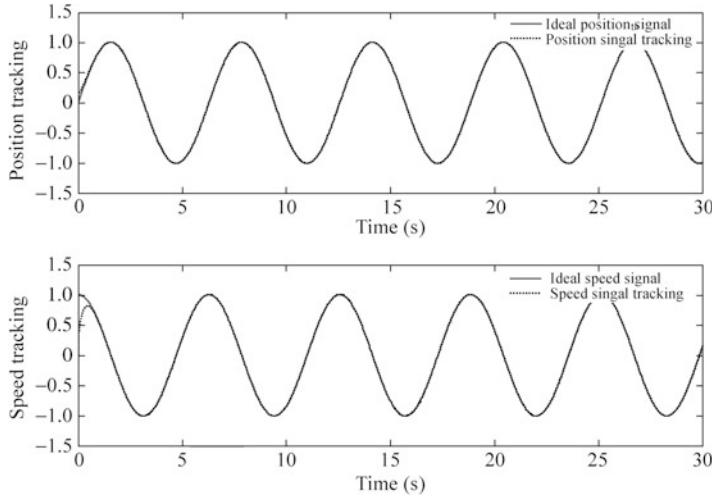


Fig. 7.3 Position and speed tracking

7.2.4 Simulation Examples

7.2.4.1 First Example

Consider a simple plant as

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau$$

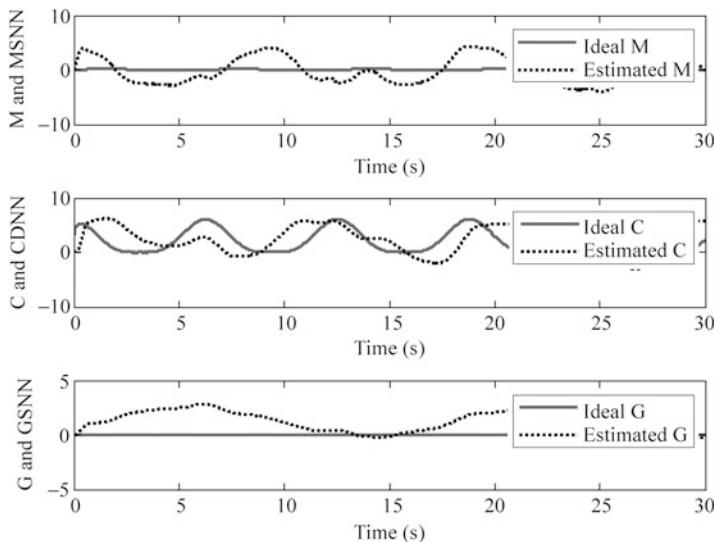
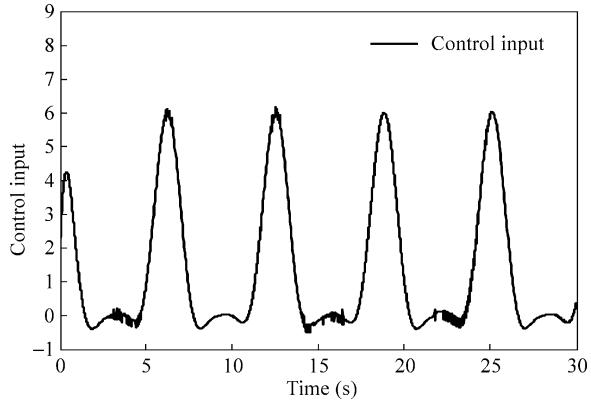
where $M = 0.1 + 0.06 \sin q$, $C = 3\dot{q} + 3 \cos \dot{q}$, $G = mgl \cos q$, $m = 0.02$, $l = 0.05$, and $g = 9.8$.

The initial states of the plant are $q(0) = 0.15$ and $\dot{q}(0) = 0$. The desired trajectory is $q_d = \sin t$.

For RBF neural network, the structure is 2-7-1, the input is $z = [q \quad \dot{q}]$, and the parameters of Gaussian functions c_i and b_i are designed as $[-1.5 \quad -1.0 \quad -0.5 \quad 0 \quad 0.5 \quad 1.0 \quad 1.5]$ and 20. The initial weight value is chosen as zero. We use control law (7.22), (7.23), and (7.24) and adaptive law (7.27), (7.28), and (7.29), the parameters are chosen as $K_r = 0.10$, $K_p = 15$, $K_i = 15$, and $\Lambda = 5.0$, the gain in the adaptive law (7.27), (7.28), and (7.29) is $\Gamma_M = 100$, $\Gamma_C = 100$ and $\Gamma_G = 100$, respectively. The results are shown from Figs. 7.3, 7.4, and 7.5.

The Simulink program of this example is chap7_2sim.mdl, and the Matlab programs of the example are given in the Appendix.

Just like the explanation in section ‘simulation of a simple adaptive control system,’ the variation of the elements of $\hat{M}(q)$, $\hat{C}(q,\dot{q})$, and $\hat{G}(q)$ do not converge to $M(q)$, $C(q,\dot{q})$, and $G(q)$. This is due to the fact that the desired trajectory is not persistently exciting, and this occurs quite often in real-world application.

Fig. 7.4 Control input**Fig. 7.5** Estimation of $M(\boldsymbol{q})$, $C(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and $G(\boldsymbol{q})$.

7.2.4.2 Second Example

Not considering friction and disturbance, the dynamic equation of two-link manipulator is

$$M(\boldsymbol{q})\ddot{\boldsymbol{q}} + C(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + G(\boldsymbol{q}) = \boldsymbol{\tau}$$

where

$$\begin{aligned}\mathbf{M}(\mathbf{q}) &= \begin{bmatrix} p_1 + p_2 + 2p_3 \cos q_2 & p_2 + p_3 \cos q_2 \\ p_2 + p_3 \cos q_2 & p_2 \end{bmatrix} \\ \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) &= \begin{bmatrix} -p_3 \dot{q}_2 \sin q_2 & -p_3 (\dot{q}_1 + \dot{q}_2) \sin q_2 \\ p_3 \dot{q}_1 \sin q_2 & 0 \end{bmatrix} \\ \mathbf{G}(\mathbf{q}) &= \begin{bmatrix} p_4 g \cos q_1 + p_5 g \cos (q_1 + q_2) \\ p_5 g \cos (q_1 + q_2) \end{bmatrix}\end{aligned}$$

Choose $\mathbf{p} = [2.90 \ 0.76 \ 0.87 \ 3.04 \ 0.87]^T$, and the initial states of the plant are $\mathbf{q}_0 = [0.09 \ -0.09]^T$ and $\dot{\mathbf{q}}_0 = [0.0 \ 0.0]^T$.

The desired trajectory is $q_{d1} = 0.5 \sin(\pi t)$ and $q_{d2} = \sin(\pi t)$. In simulation, we use control law (7.22), (7.23), and (7.24) with adaptive law (7.27), (7.28), and (7.29), and the parameters are chosen as $\mathbf{K}_p = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$, $\mathbf{K}_i = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$, $\mathbf{K}_r = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}$, and $\Lambda = \begin{bmatrix} 5.0 & 0 \\ 0 & 5.0 \end{bmatrix}$. The element of the gain matrix in the adaptive law (7.27), (7.28), and (7.29) is $\Gamma_{Mk} = 5$, $\Gamma_{Ck} = 10$ and $\Gamma_{Gk} = 5$, respectively.

For RBF neural network, the structure is 2-5-1. For $\mathbf{M}(\mathbf{q})$ and $\mathbf{G}(\mathbf{q})$, the input is $\mathbf{z} = [q_1 \ q_2]$. For $\mathbf{M}(\mathbf{q})$ and $\mathbf{G}(\mathbf{q})$, the input is $\mathbf{z} = [q_1 \ q_2 \ \dot{q}_1 \ \dot{q}_2]$.

The parameters of Gaussian functions c_i and b_i are designed as $[-1.5 \ -1.0 \ -0.5 \ 0 \ 0.5 \ 1.0 \ 1.5]$ and 10. The initial weight value is chosen as zero. The simulation results are shown from Figs. 7.6, 7.7, 7.8, and 7.9.

For the same reason, the variation of the norm of $\hat{\mathbf{M}}(\mathbf{q})$, $\hat{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}})$, and $\hat{\mathbf{G}}(\mathbf{q})$ do not converge to the norm of $\mathbf{M}(\mathbf{q})$, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ and $\mathbf{G}(\mathbf{q})$.

The Simulink program of this example is chap7_3sim.mdl, and the Matlab programs of the example are given in the [Appendix](#).

7.3 Adaptive Neural Network Control of Robot Manipulators in Task Space

The adaptive neural network control can be further extended to the task space or the so-called Cartesian space [3, 4]. To apply robot manipulators to a wide class of tasks, it will be necessary to control not only the position of the end effector but also the force exerted by the end effector on the object. By designing the control law in task space, force control can be easily formulated. Most controllers proposed thus far for adaptive manipulator tracking in the task space require some sort of inverse of the Jacobian matrix [5]. However, it is time-consuming and quite difficult to obtain the inverse of the Jacobian matrix. By directly parameterizing the control law, the inverse of the Jacobian matrix is not needed in this section.

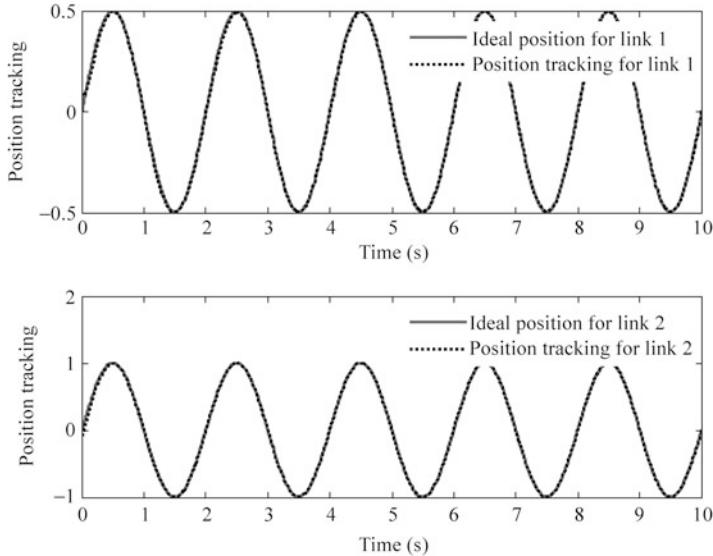


Fig. 7.6 Position tracking of link 1 and link2

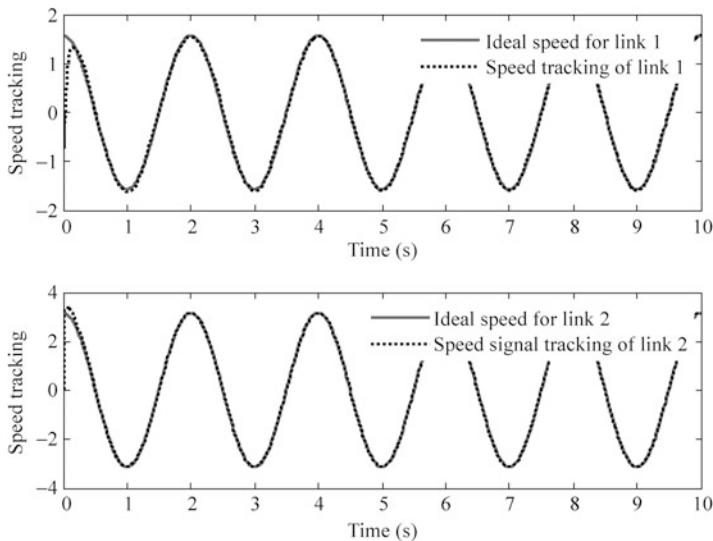


Fig. 7.7 Speed tracking of link 1 and link 2

In this section, reference to the paper [4], an adaptive neural network controller design of robot manipulators in task space is introduced. RBF neural networks are used, uniformly stable adaptation is assured, and asymptotically tracking is achieved.

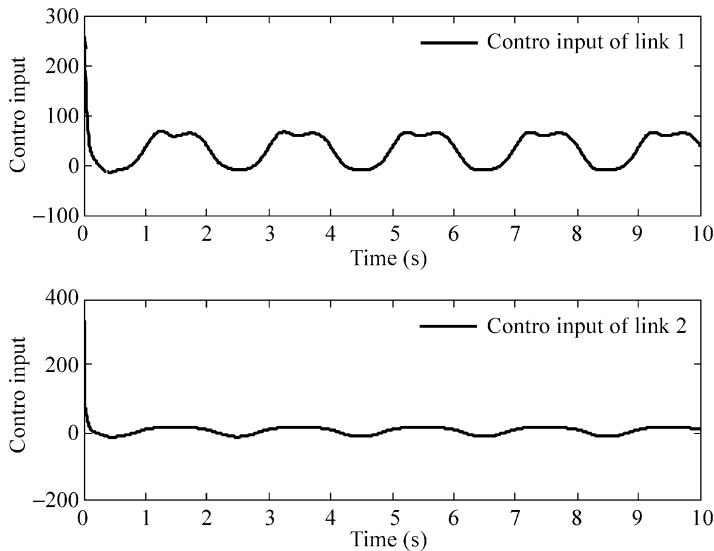


Fig. 7.8 Control inputs of link1 and link 2

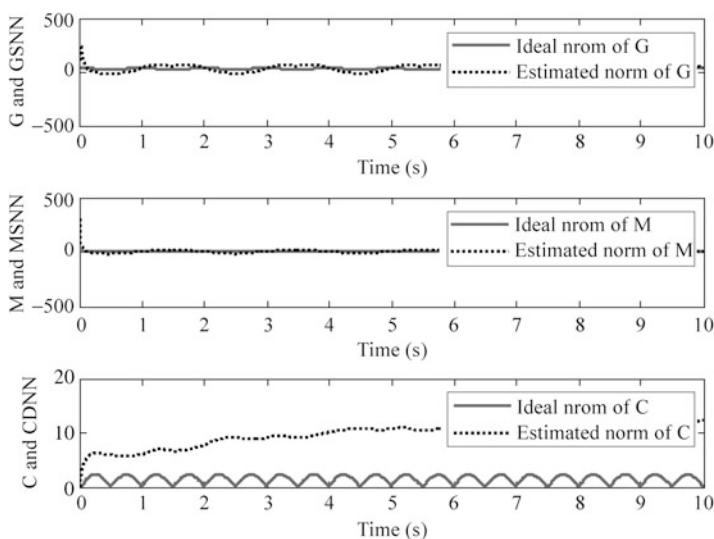
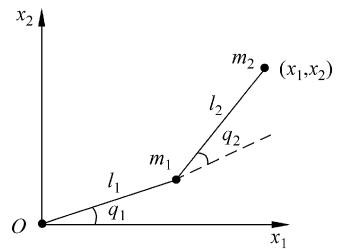


Fig. 7.9 Norm estimation of $M(\boldsymbol{q})$, $C(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and $G(\boldsymbol{q})$

Fig. 7.10 Two-degree-of-freedom robot manipulator



7.3.1 Coordination Transformation from Task Space to Joint Space

To express the system in the form of task space, we must express the end-effector joint position coordinates (q_1, q_2) in the form of Cartesian coordinates (x_1, x_2) .

From Fig. 7.10, we have

$$x_1 = l_1 \cos q_1 + l_2 \cos(q_1 + q_2) \quad (7.31)$$

$$x_2 = l_1 \sin q_1 + l_2 \sin(q_1 + q_2) \quad (7.32)$$

From (7.31) and (7.32), we have

$$x_1^2 + x_2^2 = l_1^2 + l_2^2 + 2l_1 l_2 \cos q_2$$

Thus

$$q_2 = \cos^{-1} \left(\frac{x_1^2 + x_2^2 - l_1^2 - l_2^2}{2l_1 l_2} \right) \quad (7.33)$$

From the book [6], if we let $p_1 = \arctan \frac{x_2}{x_1}$ and $p_2 = \arccos \frac{x_1^2 + x_2^2 + l_1^2 - l_2^2}{2l_1 \sqrt{x_1^2 + x_2^2}}$,

then

$$q_1 = \begin{cases} p_1 - p_2, & q_2 > 0 \\ p_1 + p_2, & q_2 \leq 0. \end{cases} \quad (7.34)$$

7.3.2 Neural Network Modeling of Robot Manipulators

Consider dynamic equation of n – link manipulator as

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} \quad (7.35)$$

where $\mathbf{M}(\mathbf{q})$ is an $n \times n$ inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is an $n \times n$ matrix containing the centrifugal and Coriolis terms, $\mathbf{G}(\mathbf{q})$ is an $n \times 1$ vector containing gravitational forces and torques, \mathbf{q} denotes the vector of joint variables, and τ is joint torques.

Usually, the manipulator task specification is given relative to the end effector. Hence, it is natural to attempt to derive the control algorithm directly in the task space rather than in the joint space. Denote the end-effector position and orientation in the task space by $\mathbf{x} \in \mathbf{R}^n$. The task space dynamics can then be written as [7]

$$\mathbf{M}_x(\mathbf{q})\ddot{\mathbf{x}} + \mathbf{C}_x(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{x}} + \mathbf{G}_x(\mathbf{q}) = \mathbf{F}_x \quad (7.36)$$

where $\mathbf{M}_x(\mathbf{q}) = \mathbf{J}^{-T}(\mathbf{q})\mathbf{M}(\mathbf{q})\mathbf{J}^{-1}(\mathbf{q})$, $\mathbf{C}_x(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{J}^{-T}(\mathbf{q})(\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{D}(\mathbf{q})\mathbf{J}^{-1}(\mathbf{q})\dot{\mathbf{J}}(\mathbf{q}))\mathbf{J}^{-1}(\mathbf{q})$, $\mathbf{G}_x(\mathbf{q}) = \mathbf{J}^{-T}(\mathbf{q})\mathbf{G}(\mathbf{q})$, $\mathbf{F}_x = \mathbf{J}^{-T}(\mathbf{q})\tau$. $\mathbf{J}(\mathbf{q}) \in \mathbf{R}^{n \times n}$ is the configuration-dependent Jacobian matrix, which is assumed to be nonsingular in the finite work space Ω .

The above dynamic equation has the following useful structure properties, which can be used in the controller design.

Property 7.1. The inertia $\mathbf{M}_x(\mathbf{q})$ is symmetric and positive definite.

Property 7.2. Matrix $\dot{\mathbf{M}}_x(\mathbf{q}) - 2\mathbf{C}_x(\mathbf{q}, \dot{\mathbf{q}})$ is skew symmetric if $\mathbf{C}_x(\mathbf{q}, \dot{\mathbf{q}})$ is defined by the Christoffel symbols.

It is observed that both $\mathbf{M}_x(\mathbf{q})$ and $\mathbf{G}_x(\mathbf{q})$ are functions of \mathbf{q} only; hence, static neural networks are sufficient to model them. Assume that $\mathbf{M}_x(\mathbf{q})$ and $\mathbf{G}_x(\mathbf{q})$ can be modeled as

$$m_{xkj}(\mathbf{q}) = \sum_l \theta_{kjl}\xi_{kjl}(\mathbf{q}) + \epsilon_{mkj}(\mathbf{q}) = \boldsymbol{\theta}_{kj}^T \boldsymbol{\xi}_{kj}(\mathbf{q}) + \epsilon_{mkj}(\mathbf{q})$$

$$g_{xk}(\mathbf{q}) = \sum_l \beta_{kl}\eta_{kl}(\mathbf{q}) + \epsilon_{gk}(\mathbf{q}) = \boldsymbol{\beta}_k^T \boldsymbol{\eta}_k(\mathbf{q}) + \epsilon_{gk}(\mathbf{q})$$

where $\theta_{kjl}, \beta_{kl} \in \mathbf{R}$ are the weights of the neural networks, $\xi_{kjl}(\mathbf{q}), \eta_{kl}(\mathbf{q}) \in \mathbf{R}$ are the corresponding Gaussian basis functions with input vector \mathbf{q} , and $\epsilon_{mkj}(\mathbf{q})$, and $\epsilon_{gk}(\mathbf{q}) \in \mathbf{R}$ are the modeling errors of $m_{xkj}(\mathbf{q})$ and $g_{xk}(\mathbf{q})$, respectively, and are assumed to be bounded.

For $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$, a dynamic neural network of \mathbf{q} and $\dot{\mathbf{q}}$ is used to model it. Assume that $c_{xkj}(\mathbf{q}, \dot{\mathbf{q}})$ can be modeled as

$$c_{xkj}(\mathbf{q}, \dot{\mathbf{q}}) = \sum_l \alpha_{kjl}\xi_{kjl}(z) + \epsilon_{ckj}(z) = \boldsymbol{\alpha}_{kj}^T \boldsymbol{\xi}_{kj}(z) + \epsilon_{ckj}(z)$$

where $z = [\mathbf{q}^T \dot{\mathbf{q}}^T]^T \in \mathbf{R}^{2n}$, and $\alpha_{kjl} \in \mathbf{R}$ are the weights, $\xi_{kjl}(z) \in \mathbf{R}$ is the corresponding Gaussian basis function with input vector z , and $\epsilon_{ckj}(z)$ is the modeling error of $c_{xkj}(\mathbf{q}, \dot{\mathbf{q}})$, which is also assumed to be bounded.

Therefore, the task space dynamics of the manipulators can be described as

$$\mathbf{M}_x(\mathbf{q})\ddot{\mathbf{x}} + \mathbf{C}_x(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{x}} + \mathbf{G}_x(\mathbf{q}) = \mathbf{F}_x \quad (7.37)$$

where

$$\begin{aligned} m_{xkj}(\mathbf{q}) &= \boldsymbol{\theta}_{kj}^T \boldsymbol{\xi}_{kj}(\mathbf{q}) + \epsilon_{dkj}(\mathbf{q}) \\ c_{xkj}(\mathbf{q}, \dot{\mathbf{q}}) &= \boldsymbol{\alpha}_{kj}^T \boldsymbol{\xi}_{kj}(z) + \epsilon_{ckj}(z) \\ g_{xk}(\mathbf{q}) &= \boldsymbol{\beta}_k^T \boldsymbol{\eta}_k(\mathbf{q}) + \epsilon_{gk}(\mathbf{q}) \end{aligned}$$

Using the GL matrix and its product operator [3], we can write $\mathbf{M}_x(\mathbf{q})$ as

$$\mathbf{M}_x(\mathbf{q}) = \left[\{\boldsymbol{\Theta}\}^T \cdot \{\Xi(\mathbf{q})\} \right] + \mathbf{E}_M(\mathbf{q}) \quad (7.38)$$

where $\{\boldsymbol{\Theta}\}$ and $\{\Xi(\mathbf{q})\}$ are the GL matrices with their elements being θ_{kj} and $\xi_{kj}(\mathbf{q})$, respectively. $\mathbf{E}_M(\mathbf{q}) \in \mathbf{R}^{n \times n}$ is a matrix with its elements being the modeling errors $\epsilon_{mkj}(\mathbf{q})$.

Similarly, for $\mathbf{C}_x(\mathbf{q}, \dot{\mathbf{q}})$ and $\mathbf{G}_x(\mathbf{q})$, we have

$$\mathbf{C}_x(\mathbf{q}, \dot{\mathbf{q}}) = \left[\{\mathbf{A}\}^T \cdot \{\mathbf{Z}(z)\} \right] + \mathbf{E}_C(z) \quad (7.39)$$

$$\mathbf{G}_x(\mathbf{q}) = \left[\{\mathbf{B}\}^T \cdot \{\mathbf{H}(\mathbf{q})\} \right] + \mathbf{E}_G(\mathbf{q}) \quad (7.40)$$

where $\{\mathbf{A}\}$, $\{\mathbf{Z}(z)\}$, $\{\mathbf{B}\}$, and $\{\mathbf{H}(\mathbf{q})\}$ are the GL matrices and vectors with their elements being α_{kj} , $\xi_{kj}(z)$, β_k , and $\eta_k(\mathbf{q})$, respectively. $\mathbf{E}_C(z) \in \mathbf{R}^{n \times n}$ and $\mathbf{E}_G(\mathbf{q}) \in \mathbf{R}^n$ are the matrix and vector with their elements being the modeling errors $\epsilon_{ckj}(z)$ and $\epsilon_{gk}(\mathbf{q})$, respectively.

7.3.3 Controller Design

Let $\mathbf{x}_d(t)$ be the desired trajectory in the task space and $\dot{\mathbf{x}}_d(t)$ and $\ddot{\mathbf{x}}_d(t)$ be the desired velocity and acceleration. Define

$$\begin{aligned} \mathbf{e}(t) &= \mathbf{x}_d(t) - \mathbf{x}(t) \\ \dot{\mathbf{x}}_r(t) &= \dot{\mathbf{x}}_d(t) + \Lambda \mathbf{e}(t) \\ \mathbf{r}(t) &= \dot{\mathbf{x}}_r(t) - \dot{\mathbf{x}}(t) = \dot{\mathbf{e}}(t) + \Lambda \mathbf{e}(t) \end{aligned}$$

where Λ is a positive-definite matrix.

Lemma 7.1 (Barbalat's Lemma) [8]. Let $h: \mathbb{R} \rightarrow \mathbb{R}$ be a uniformly continuous function defined on $[0, +\infty)$. Suppose that $\lim_{t \rightarrow \infty} \int_0^t h(\delta) d\delta$ exists and is finite. Then we have $\lim_{t \rightarrow \infty} h(t) = 0$.

Lemma 7.2 [9]. Let $e(t) = h(t) * r(t)$. “*” denotes convolution product; $h(t) = L^{-1}H(s)$, where $H(s)$ is an $n \times n$ strictly proper, exponentially stable transfer function. If $r \in L_2^n$, then $e \in L_2^n \cap L_\infty^n$, $\dot{e} \in L_2^n$, e is continuous, and $e \rightarrow 0$ as $t \rightarrow \infty$. In addition, if $r \rightarrow 0$ as $t \rightarrow \infty$, then $\dot{e} \rightarrow 0$.

Explanation of Lemma 7.1: Consider a second SISO system. We select $r(t) = c e(t) + \dot{e}(t)$, then we have $r(s) = e(s)(c + s)$ and $e(s) = \frac{1}{s+c} r(s)$, and thus $H(s) = \frac{1}{s+c}$. To guarantee $H(s)$ a strictly proper, exponentially stable transfer function, we can design $c > 0$. Under these conditions, if $r(t) = 0$, we have $c e(t) + \dot{e}(t) = 0$, and the exponentially convergence can be guaranteed.

Denote the estimate of (\cdot) by $(\hat{\cdot})$, and define $(\tilde{\cdot}) = (\cdot) - (\hat{\cdot})$; hence, $\{\hat{\Theta}\}$, $\{\hat{A}\}$ and $\{\hat{B}\}$ represent the estimates of the true parameter matrices $\{\Theta\}$, $\{A\}$, and $\{B\}$, respectively.

Design a general controller in the form [3]

$$\begin{aligned} F_x = & \left[\{\hat{\Theta}\}^T \cdot \{\Xi(q)\} \right] \ddot{x}_r + \left[\{\hat{A}\}^T \cdot \{Z(z)\} \right] \dot{x}_r + \left[\{\hat{B}\}^T \cdot \{H(q)\} \right] + Kr \\ & + k_s \text{sgn}(r) \end{aligned} \quad (7.41)$$

where $K \in R^{n \times n} > 0$, $k_s > \|E\|$, and $E = E_M(q)\ddot{x}_r + E_C(z)\dot{x}_r + E_G(q)$.

The first three terms of the control law are the model-based control, whereas the K_r term gives PD-type control, and the last term in the control law is added to suppress the modeling error of the neural networks.

From the controller expression, it is clear that the controller does not require the inverse of the Jacobian matrix. In real-time implementation, the practical control input can be computed as $\tau = J^T(q)F_x$.

Substituting (7.38), (7.39), and (7.40) into (7.37), we have

$$\begin{aligned} & \left\{ \left[\{\Theta\}^T \cdot \{\Xi(q)\} \right] + E_M(q) \right\} \ddot{x} + \left\{ \left[\{A\}^T \cdot \{Z(z)\} \right] + E_C(z) \right\} \dot{x} \\ & + \left[\{B\}^T \cdot \{H(q)\} \right] + E_G(q) = F_x. \end{aligned}$$

Applying the control law (7.41) into above equation yields

$$\begin{aligned} & \left\{ \left[\{\Theta\}^T \cdot \{\Xi(q)\} \right] + E_M(q) \right\} \ddot{x} + \left\{ \left[\{A\}^T \cdot \{Z(z)\} \right] + E_C(z) \right\} \dot{x} + \left[\{B\}^T \cdot \{H(q)\} \right] + E_G(q) \\ & = \left[\{\hat{\Theta}\}^T \cdot \{\Xi(q)\} \right] \ddot{x}_r + \left[\{\hat{A}\}^T \cdot \{Z(z)\} \right] \dot{x}_r + \left[\{\hat{B}\}^T \cdot \{H(q)\} \right] + Kr + k_s \text{sgn}(r). \end{aligned}$$

Substituting $\dot{\mathbf{x}} = \dot{\mathbf{x}}_r - \mathbf{r}$ and $\ddot{\mathbf{x}} = \ddot{\mathbf{x}}_r - \dot{\mathbf{r}}$ into above yields

$$\begin{aligned} & \left\{ \left[\{\boldsymbol{\theta}\}^T \cdot \{\Xi(\mathbf{q})\} \right] + \mathbf{E}_M(\mathbf{q}) \right\} (\ddot{\mathbf{x}}_r - \dot{\mathbf{r}}) + \left\{ \left[\{\mathbf{A}\}^T \cdot \{\mathbf{Z}(z)\} \right] + \mathbf{E}_C(z) \right\} (\dot{\mathbf{x}}_r - \mathbf{r}) + \left[\{\mathbf{B}\}^T \cdot \{\mathbf{H}(\mathbf{q})\} \right] + \mathbf{E}_G(\mathbf{q}) \\ &= \left[\{\tilde{\boldsymbol{\theta}}\}^T \cdot \{\Xi(\mathbf{q})\} \right] \ddot{\mathbf{x}}_r + \left[\{\tilde{\mathbf{A}}\}^T \cdot \{\mathbf{Z}(z)\} \right] \dot{\mathbf{x}}_r + \left[\{\tilde{\mathbf{B}}\}^T \cdot \{\mathbf{H}(\mathbf{q})\} \right] + \mathbf{K}\mathbf{r} + k_s \operatorname{sgn}(\mathbf{r}) \end{aligned}$$

Simplifying the above equation yields

$$\begin{aligned} & \left\{ \left[\{\boldsymbol{\theta}\}^T \cdot \{\Xi(\mathbf{q})\} \right] + \mathbf{E}_M(\mathbf{q}) \right\} \dot{\mathbf{r}} + \left\{ \left[\{\mathbf{A}\}^T \cdot \{\mathbf{Z}(z)\} \right] + \mathbf{E}_C(z) \right\} \mathbf{r} + \mathbf{K}\mathbf{r} + k_s \operatorname{sgn}(\mathbf{r}) \\ &= \left[\{\tilde{\boldsymbol{\theta}}\}^T \cdot \{\Xi(\mathbf{q})\} \right] \ddot{\mathbf{x}}_r + \left[\{\tilde{\mathbf{A}}\}^T \cdot \{\mathbf{Z}(z)\} \right] \dot{\mathbf{x}}_r + \left[\{\tilde{\mathbf{B}}\}^T \cdot \{\mathbf{H}(\mathbf{q})\} \right] + \mathbf{E}. \end{aligned}$$

Applying (7.38) and (7.39) into above equation yields

$$\begin{aligned} & \mathbf{M}_x(\mathbf{q})\dot{\mathbf{r}} + \mathbf{C}_x(\mathbf{q}, \dot{\mathbf{q}})\mathbf{r} + \mathbf{K}\mathbf{r} + k_s \operatorname{sgn}(\mathbf{r}) \\ &= \left[\{\tilde{\boldsymbol{\theta}}\}^T \cdot \{\Xi(\mathbf{q})\} \right] \ddot{\mathbf{x}}_r + \left[\{\tilde{\mathbf{A}}\}^T \cdot \{\mathbf{Z}(z)\} \right] \dot{\mathbf{x}}_r + \left[\{\tilde{\mathbf{B}}\}^T \cdot \{\mathbf{H}(\mathbf{q})\} \right] + \mathbf{E}. \quad (7.42) \end{aligned}$$

For the closed system (7.42), an adaptive law theorem is given as follows:

Theorem 7.1 [3]. For the closed-loop system (7.42), if $\mathbf{K} > 0$, $k_s > \|\mathbf{E}\|$, and the adaptive weight value are designed as

$$\begin{aligned} \dot{\hat{\boldsymbol{\theta}}}_k &= \boldsymbol{\Gamma}_k \cdot \{\boldsymbol{\xi}_k(\mathbf{q})\} \ddot{\mathbf{x}}_r r_k \\ \dot{\hat{\boldsymbol{\alpha}}}_k &= \boldsymbol{Q}_k \cdot \{\boldsymbol{\xi}_k(z)\} \dot{\mathbf{x}}_r r_k \\ \dot{\hat{\boldsymbol{\beta}}}_k &= \mathbf{N}_k \boldsymbol{\eta}_k(\mathbf{q}) r_k \end{aligned} \quad (7.43)$$

where $\boldsymbol{\Gamma}_k = \boldsymbol{\Gamma}_k^T > 0$, $\boldsymbol{Q}_k = \boldsymbol{Q}_k^T > 0$, and $\mathbf{N}_k = \mathbf{N}_k^T > 0$, then $\hat{\boldsymbol{\theta}}_k$, $\hat{\boldsymbol{\alpha}}_k$, $\hat{\boldsymbol{\beta}}_k \in L_\infty$ and $\mathbf{e} \in L_2^n \cap L_\infty^n$, \mathbf{e} are continuous, $\mathbf{e} \rightarrow 0$, and $\dot{\mathbf{e}} \rightarrow 0$ as $t \rightarrow \infty$.

Proof:

Consider the nonnegative Lyapunov function as

$$V = \frac{1}{2} \mathbf{r}^T \mathbf{M}_x(\mathbf{q}) \mathbf{r} + \frac{1}{2} \sum_{k=1}^n \tilde{\boldsymbol{\theta}}_k^T \boldsymbol{\Gamma}_k^{-1} \tilde{\boldsymbol{\theta}}_k + \frac{1}{2} \sum_{k=1}^n \tilde{\boldsymbol{\alpha}}_k^T \boldsymbol{Q}_k^{-1} \tilde{\boldsymbol{\alpha}}_k + \frac{1}{2} \sum_{k=1}^n \tilde{\boldsymbol{\beta}}_k^T \mathbf{N}_k^{-1} \tilde{\boldsymbol{\beta}}_k.$$

Where $\boldsymbol{\Gamma}_k$, \boldsymbol{Q}_k , \mathbf{N}_k are dimensional compatible symmetric positive-definite matrices, computing the derivative of V yields

$$\dot{V} = \mathbf{r}^T \mathbf{M}_x \dot{\mathbf{r}} + \frac{1}{2} \mathbf{r}^T \dot{\mathbf{M}}_x \mathbf{r} + \sum_{k=1}^n \tilde{\boldsymbol{\theta}}_k^T \boldsymbol{\Gamma}_k^{-1} \dot{\tilde{\boldsymbol{\theta}}}_k + \sum_{k=1}^n \tilde{\boldsymbol{\alpha}}_k^T \boldsymbol{Q}_k^{-1} \dot{\tilde{\boldsymbol{\alpha}}}_k + \sum_{k=1}^n \tilde{\boldsymbol{\beta}}_k^T \mathbf{N}_k^{-1} \dot{\tilde{\boldsymbol{\beta}}}_k.$$

Consider the property of skew symmetric of $\dot{\mathbf{M}}_x(\mathbf{q}) - 2\mathbf{C}_x(\mathbf{q}, \dot{\mathbf{q}})$, then $\mathbf{r}^T(\dot{\mathbf{M}}_x - 2\mathbf{C}_x)\mathbf{r} = 0$, and

$$\dot{V} = \mathbf{r}^T(\mathbf{M}_x \dot{\mathbf{r}} + \mathbf{C}_x \mathbf{r}) - \sum_{k=1}^n \tilde{\boldsymbol{\theta}}_k^T \boldsymbol{\Gamma}_k^{-1} \dot{\boldsymbol{\theta}}_k - \sum_{k=1}^n \tilde{\boldsymbol{\alpha}}_k^T \boldsymbol{Q}_k^{-1} \dot{\boldsymbol{\alpha}}_k - \sum_{k=1}^n \tilde{\boldsymbol{\beta}}_k^T \boldsymbol{N}_k^{-1} \dot{\boldsymbol{\beta}}_k.$$

Substituting (7.42) into above yields

$$\begin{aligned} \dot{V} = & -\mathbf{r}^T \mathbf{K} \mathbf{r} - k_s \mathbf{r}^T \text{sgn}(\mathbf{r}) + \sum_{k=1}^n \{\tilde{\boldsymbol{\theta}}_k\}^T \cdot \{\xi_k(\mathbf{q})\} \ddot{x}_r r_k + \sum_{k=1}^n \tilde{\boldsymbol{\alpha}}_k^T \cdot \{\xi_k(z)\} \dot{x}_r \mathbf{r}_k \\ & + \sum_{k=1}^n \tilde{\boldsymbol{\beta}}_k^T \eta_k(\mathbf{q}) \mathbf{r}_k + \mathbf{r}^T \mathbf{E} - \sum_{k=1}^n \tilde{\boldsymbol{\theta}}_k^T \boldsymbol{\Gamma}_k^{-1} \dot{\boldsymbol{\theta}}_k - \sum_{k=1}^n \tilde{\boldsymbol{\alpha}}_k^T \boldsymbol{Q}_k^{-1} \dot{\boldsymbol{\alpha}}_k - \sum_{k=1}^n \tilde{\boldsymbol{\beta}}_k^T \boldsymbol{N}_k^{-1} \dot{\boldsymbol{\beta}}_k. \end{aligned} \quad (7.44)$$

Substituting the adaptive law (7.43) into (7.44), with $k_s > \|\mathbf{E}\|$, yields

$$\dot{V} = -\mathbf{r}^T \mathbf{K} \mathbf{r} - k_s \mathbf{r}^T \text{sgn}(\mathbf{r}) + \mathbf{r}^T \mathbf{E} \leq 0.$$

- From $\dot{V} \leq -\mathbf{r}^T \mathbf{K} \mathbf{r} \leq 0$ and $\mathbf{K} > 0$, it follows that $\mathbf{r} \in \mathbf{L}_2^n$. From Lemma 7.1, $\mathbf{e} \in \mathbf{L}_2^n \cap \mathbf{L}_\infty^n$, $\dot{\mathbf{e}} \in \mathbf{L}_2^n$, \mathbf{e} is continuous, and $\mathbf{e} \rightarrow 0$ as $t \rightarrow \infty$.
- Since $\dot{V} \leq -\mathbf{r}^T \mathbf{K} \mathbf{r} \leq 0$, it follows that $0 \leq V(t) \leq V(0)$ and $\forall t \geq 0$. Hence, as $V(t) \in \mathbf{L}_\infty$, this implies that $\mathbf{r}, \tilde{\boldsymbol{\theta}}_k, \tilde{\boldsymbol{\alpha}}_k, \tilde{\boldsymbol{\beta}}_k \in \mathbf{L}_\infty$, that is, $\hat{\boldsymbol{\theta}}_k, \hat{\boldsymbol{\alpha}}_k, \hat{\boldsymbol{\beta}}_k \in \mathbf{L}_\infty$.

By observing that $\mathbf{r} \in \mathbf{L}_2^n$, $x_d, \dot{x}_d, \ddot{x}_d \in \mathbf{L}_\infty^n$, $\{\Xi(\mathbf{q})\}$, $\{\mathbf{Z}(z)\}$, and $\{\mathbf{H}(\mathbf{q})\}$ are bounded basis functions, we can conclude that $\dot{\mathbf{r}} \in \mathbf{L}_2^n$ from (7.42). Therefore, \mathbf{r} is uniformly continuous. The proof is completed using the Barbalat's Lemma: if \mathbf{r} is uniformly continuous and $\mathbf{r} \in \mathbf{L}_2^n$, then $\mathbf{r} \rightarrow 0$ as $t \rightarrow \infty$; hence, $\dot{\mathbf{e}} \rightarrow 0$ as $t \rightarrow \infty$.

7.3.4 Simulation Examples

Consider dynamic equation of two-link manipulator as

$$\mathbf{M}(\mathbf{q}) \ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau}$$

$$\text{where } \mathbf{M}(\mathbf{q}) = \begin{bmatrix} m_1 + m_2 + 2m_3 \cos q_2 & m_2 + m_3 \cos q_2 \\ m_2 + m_3 \cos q_2 & m_2 \end{bmatrix},$$

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} -m_3 \dot{q}_2 \sin q_2 & -m_3 (\dot{q}_1 + \dot{q}_2) \sin q_2 \\ m_3 \dot{q}_1 \sin q_2 & 0.0 \end{bmatrix}, \quad \mathbf{G}(\mathbf{q}) = \begin{bmatrix} m_4 g \cos q_1 + m_5 g \cos(q_1 + q_2) \\ m_5 g \cos(q_1 + q_2) \end{bmatrix}, \quad m_i$$

are the parameters given by $\mathbf{M}_m = \mathbf{P} + p_l \mathbf{L}$ with $\mathbf{M}_m = [m_1 \ m_2 \ m_3 \ m_4 \ m_5]^T$,

$\mathbf{P} = [p_1 \ p_2 \ p_3 \ p_4 \ p_5]^T$, and $\mathbf{L} = [l_1^2 \ l_2^2 \ l_1 l_2 \ l_1 \ l_2]^T$, where p_i is the payload, l_1 and l_2 are the lengths of link 1 and link 2, respectively, and \mathbf{p} is the parameter vector of the robot itself.

The true parameters of the robot used for the simulation are $\mathbf{P} = [1.66 \ 0.42 \ 0.63 \ 3.75 \ 1.25]^T \text{kg} \cdot \text{m}^2$ and $l_1 = l_2 = 1 \text{ m}$.

In order to test load disturbance rejection of the controller, a payload $p_l = 0.5$ was put on at time $t = 4.0$.

The desired trajectory in the Cartesian space is chosen as $x_{d1}(t) = 1.0 + 0.2 \cos(\pi t)$, $x_{d2}(t) = 1.0 + 0.2 \sin(\pi t)$, which represents a circle of radius 0.2 m, and its center is located at $(x_1, x_2) = (1.0, 1.0) \text{m}$. The robot is initially rested with its end effector positioned at the center of the circle, that is, $\mathbf{x}(0) = [1.0 \ 1.0]^T \text{m}$ and $\dot{\mathbf{x}}(0) = [0.0 \ 0.0]^T \text{m/s}$.

To express the system in the form of task space, we must use (7.33) and (7.34) to express the end-effector position in the form of Cartesian coordinates.

The Jacobian matrix $\mathbf{J}(\mathbf{q})$ is known as

$$\mathbf{J}(\mathbf{q}) = \begin{bmatrix} -l_1 \sin(q_1) - l_2 \sin(q_1 + q_2) & -l_2 \sin(q_1 + q_2) \\ l_1 \cos(q_1) + l_2 \cos(q_1 + q_2) & l_2 \cos(q_1 + q_2) \end{bmatrix}.$$

Thus,

$$\begin{aligned} \dot{\mathbf{J}}(\mathbf{q}) &= \begin{bmatrix} -l_1 \cos(q_1) - l_2 \cos(q_1 + q_2) & -l_2 \cos(q_1 + q_2) \\ -l_1 \sin(q_1) - l_2 \sin(q_1 + q_2) & -l_2 \sin(q_1 + q_2) \end{bmatrix} \dot{q}_1 \\ &\quad + \begin{bmatrix} -l_2 \cos(q_1 + q_2) & -l_2 \cos(q_1 + q_2) \\ -l_2 \sin(q_1 + q_2) & -l_2 \sin(q_1 + q_2) \end{bmatrix} \dot{q}_2. \end{aligned}$$

For each element of $\mathbf{M}_x(\mathbf{q})$ and $\mathbf{G}_x(\mathbf{q})$, the input of RBF is \mathbf{q} , and seven neural nets in hidden layer are designed. For each element of $\mathbf{C}_x(\mathbf{q}, \dot{\mathbf{q}})$, the input of RBF is $(\mathbf{q}, \dot{\mathbf{q}})$, and also seven neural nets in hidden layer are designed.

For each Gaussian function, the parameters of c_i and b_i are designed as $[-1.5 \ -1.0 \ -0.5 \ 0 \ 0.5 \ 1.0 \ 1.5]$ and 10. The initial weight value is chosen as zero. We use the control law (7.41) and adaptive law (7.43), and the parameters are chosen as $\mathbf{K} = \begin{bmatrix} 30 & 0 \\ 0 & 30 \end{bmatrix}$, $k_s = 0.5$. According to Lemma 7.2, we

choose $\Lambda = \begin{bmatrix} 15 & 0 \\ 0 & 15 \end{bmatrix}$. The gains in the adaptive law (7.43) were given as $\Gamma_k = \text{diag}\{2.0\}$, $\mathbf{Q}_k = \text{diag}\{0.10\}$, $\mathbf{N}_k = \text{diag}\{5.0\}$, respectively. The results are shown from Figs. 7.11, 7.12, 7.13, 7.14, and 7.15.

The variation of the norm of $\|\hat{\mathbf{M}}_x(\mathbf{q})\|$, $\|\hat{\mathbf{C}}_x(\mathbf{q}, \dot{\mathbf{q}})\|$, and $\|\hat{\mathbf{G}}_x(\mathbf{q})\|$ do not converge to $\|\mathbf{M}_x(\mathbf{q})\|$, $\|\mathbf{C}_x(\mathbf{q}, \dot{\mathbf{q}})\|$, and $\|\mathbf{G}_x(\mathbf{q})\|$. This is due to the fact that the desired trajectory is not persistently exciting, and this occurs quite often in real-world application.

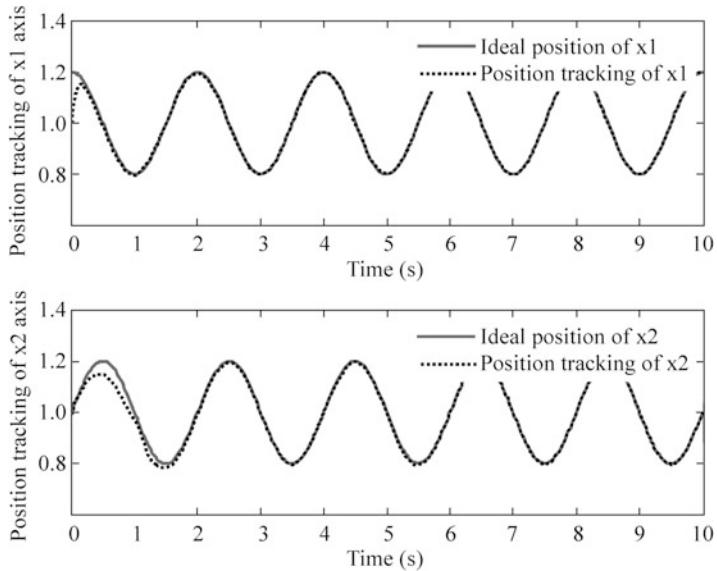


Fig. 7.11 Position tracking in x_1 and x_2 axis of end effector

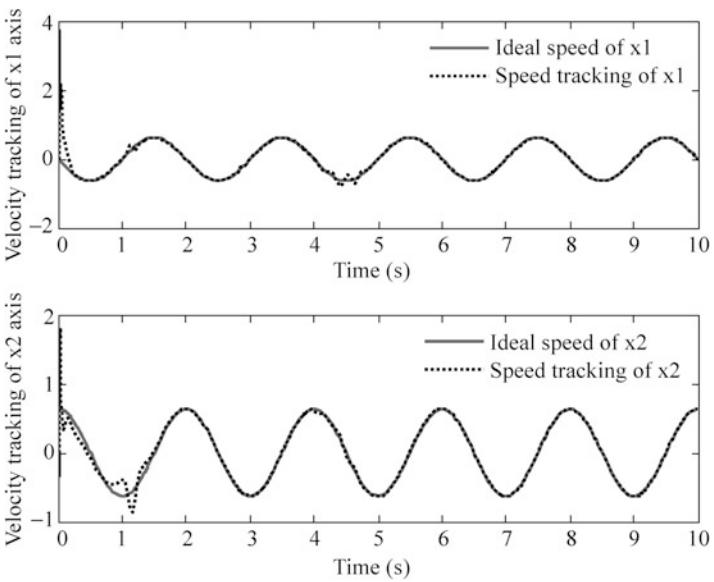


Fig. 7.12 Speed tracking in x_1 and x_2 axis of end effector

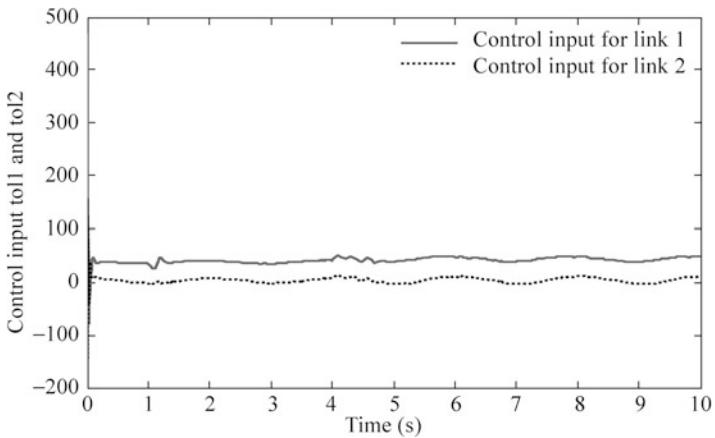


Fig. 7.13 Control input for link 1 and link 2

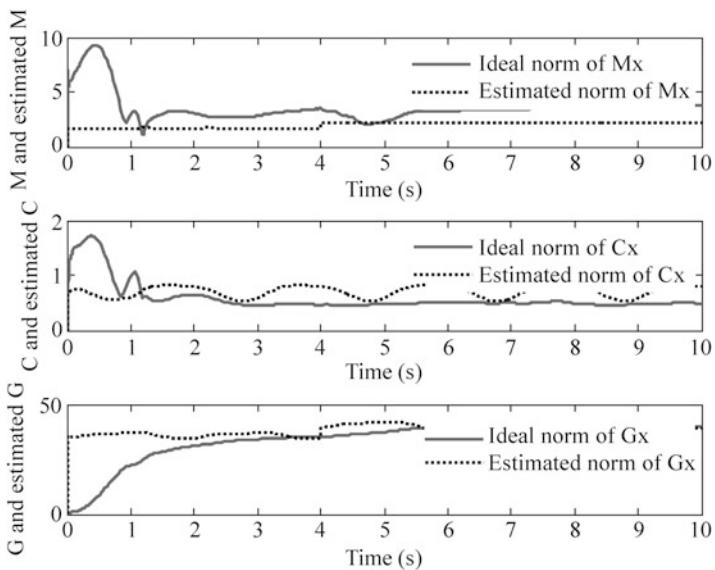


Fig. 7.14 Norm estimation of $\|M_x(\mathbf{q})\|$, $\|C_x(\mathbf{q}, \dot{\mathbf{q}})\|$, and $\|G_x(\mathbf{q})\|$

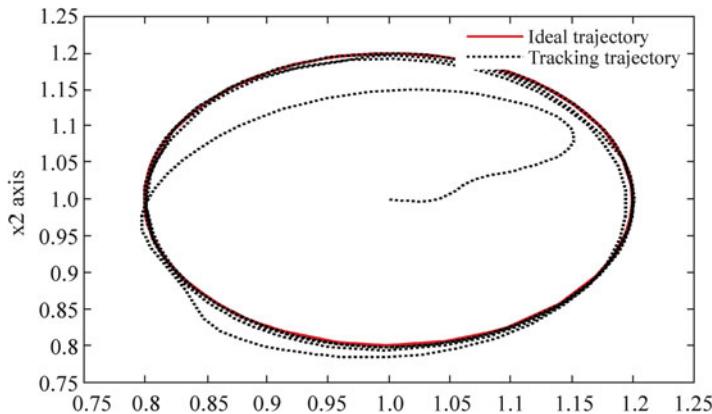


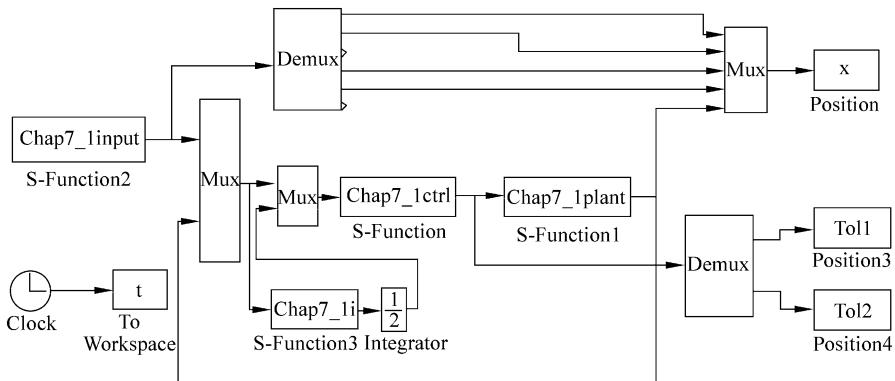
Fig. 7.15 Trajectory tracking of the end-effector

The Simulink program of this example is chap7_4sim.mdl, and the Matlab programs of the example are given in the [Appendix](#).

Appendix

Programs for Sect. 7.1.4

Simulink main program:chap7_1sim.mdl



Tracking command program:chap7_1input.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=0.5*sin(pi*t);
d_qd1=0.5*pi*cos(pi*t);
dd_qd1=-0.5*pi*pi*sin(pi*t);
qd2=sin(pi*t);
d_qd2=pi*cos(pi*t);
dd_qd2=-pi*pi*sin(pi*t);

sys(1)=qd1;
sys(2)=d_qd1;
sys(3)=dd_qd1;
sys(4)=qd2;
sys(5)=d_qd2;
sys(6)=dd_qd2;

```

S function for control law:chap7_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 {1,2,4,9}
    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 = 12;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [];
function sys=mdlOutputs(t,x,u)
qd1=u(1);
dqd1=u(2);
ddq1=u(3);
qd2=u(4);
dqd2=u(5);
ddq2=u(6);

q1=u(7);
dq1=u(8);
q2=u(9);
dq2=u(10);
dq=[dq1;dq2];

e1=qd1-q1;
e2=qd2-q2;
de1=dqd1-dq1;
de2=dqd2-dq2;
e=[e1;e2];
de=[de1;de2];
Fai=5*eye(2);
r=de+Fai*e;

dqd=[dq1;dq2];
dqr=dqd+Fai*e;
ddqd=[ddq1;ddq2];
ddqr=ddqd+Fai*de;

p=[2.9 0.76 0.87 3.04 0.87];
g=9.8;
M=[p(1)+p(2)+2*p(3)*cos(q2) p(2)+p(3)*cos(q2);
    p(2)+p(3)*cos(q2) p(2)];
```

```

C=[-p(3)*dq2*sin(q2) -p(3)*(dq1+dq2)*sin(q2);
   p(3)*dq1*sin(q2) 0];
G=[p(4)*g*cos(q1)+p(5)*g*cos(q1+q2);
   p(5)*g*cos(q1+q2)];
M0=0.8*M;
C0=0.8*C;
G0=0.8*G;
tolm=M0*ddqr+C0*dqr+G0;

EM=0.2*M; EC=0.2*C; EG=0.2*G;
E1=EM*ddqr+EC*dqr+EG;
told=20*sign(dq);
E=E1+told;

Kr=15*eye(2); %Krii>=Ei
tolr=Kr*sign(r);

Kp=100*eye(2);
Ki=100*eye(2);

I=[u(11);u(12)];
tol=tolm+Kp*r+Ki*I+tolr;

sys(1)=tol(1);
sys(2)=tol(2);

```

S function for control integrator: chap7_1i.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 = 10;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);

```

```
x0 = [] ;
str = [] ;
ts = [] ;
function sys=mdlOutputs(t,x,u)
qd1=u(1) ;
dqd1=u(2) ;
ddqd1=u(3) ;
qd2=u(4) ;
dqd2=u(5) ;
ddqd2=u(6) ;

q1=u(7) ;
dq1=u(8) ;
q2=u(9) ;
dq2=u(10) ;
q=[q1;q2] ;

e1=qd1-q1;
e2=qd2-q2;
de1=dqd1-dq1;
de2=dqd2-dq2;
e=[e1;e2];
de=[de1;de2];
Fai=5*eye(2);
r=de+Fai*e;

sys(1:2)=r;
```

S function for plant: chap7_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 = 4;
sizes.NumDiscStates = 0;
```

```

sizes.NumOutputs = 4;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.09 0 -0.09 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
p=[2.9 0.76 0.87 3.04 0.87];
g=9.8;

M=[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))];

M0=0.8*M;
C0=0.8*C;
G0=0.8*G;

tol=u(1:2);
q=[x(1);x(3)];
dq=[x(2);x(4)];
told=20*sign(dq);

ddq=inv(M0)*(tol-C0*dq-G0-told);

sys(1)=x(2);
sys(2)=ddq(1);
sys(3)=x(4);
sys(4)=ddq(2);
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
sys(3)=x(3);
sys(4)=x(4);

```

Plot program:chap7_1plot.m

```

close all;

figure(1);
subplot(211);
plot(t,x(:,1),'r',t,x(:,5),'b:','linewidth',2);
xlabel('time(s)');ylabel('position tracking for link 1');
legend('Ideal position signal','Position signal tracking');

```

```

subplot(212);
plot(t,x(:,2),'r',t,x(:,6),'b','linewidth',2);
xlabel('time(s)');ylabel('speed tracking for link 1');
legend('Ideal speed signal','Speed signal tracking');

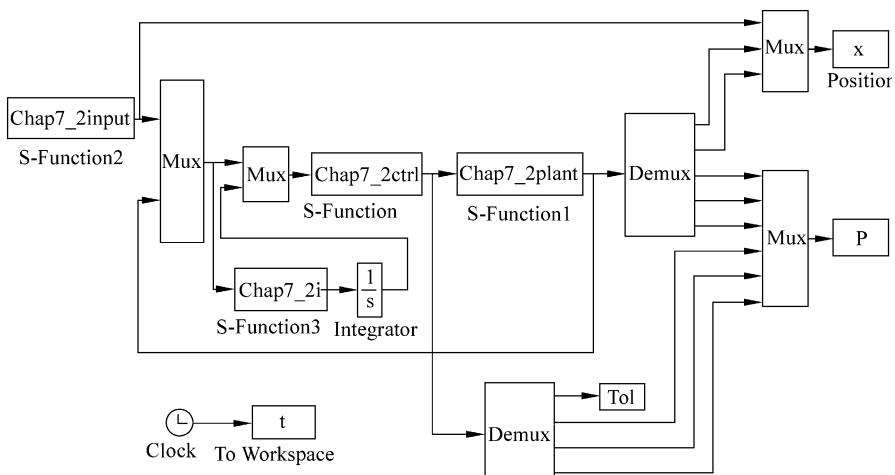
figure(2);
subplot(211);
plot(t,x(:,3),'r',t,x(:,7),'b','linewidth',2);
xlabel('time(s)');ylabel('position tracking for link 1');
legend('Ideal position signal','Position signal
tracking');
subplot(212);
plot(t,x(:,4),'r',t,x(:,8),'b','linewidth',2);
xlabel('time(s)');ylabel('speed tracking for link 1');
legend('Ideal speed signal','Speed signal tracking');

figure(3);
subplot(211);
plot(t,tol1(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('control input of link 1');
subplot(212);
plot(t,tol2(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('control input of link 2');

```

Programs for Sect. 7.2.4.1

Simulink main program: chap7_2sim.mdl



Tracking command program:chap7_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 = 3;
sizes.NumInputs = 0;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [0 0];
function sys=mdlOutputs(t,x,u)
qd=sin(t);
dqd=cos(t);
ddqd=-sin(t);

sys(1)=qd;
sys(2)=dqd;
sys(3)=ddqd;

```

S function for control law:chap7_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_M c_C c_G b Fai
node=7;
c_M=[-1.5 -1 -0.5 0 0.5 1 1.5];
c_G=[-1.5 -1 -0.5 0 0.5 1 1.5];
c_C=[-1.5 -1 -0.5 0 0.5 1 1.5;
      -1.5 -1 -0.5 0 0.5 1 1.5];
b=20;
Fai=5;

sizes = simsizes;
sizes.NumContStates = 3*node;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 4;
sizes.NumInputs = 9;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = zeros(1,3*node);
str = [];
ts = [];
function sys=mdlDerivatives(t,x,u)
global node c_M c_C c_G b Fai
qd=u(1);
dqd=u(2);
ddqd=u(3);

q=u(4);
dq=u(5);

for j=1:1:node
    h_M(j)=exp(-norm(q-c_M(:,j))^2/(b*b));
end

for j=1:1:node
    h_G(j)=exp(-norm(q-c_G(:,j))^2/(b*b));
end

z=[q;dq];
for j=1:1:node
    h_C(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
end
e=qd-q;
de=dqd-dq;
r=de+Fai*e;
```

```

dqr=dqd+Fai*e;
ddqr=ddqd+Fai*de;
T_M=100;
for i=1:1:node
    sys(i)=T_M*h_M(i)*ddqr*r;
end
T_C=100;
for i=1:1:node
    sys(2*node+i)=T_C*h_C(i)*dqr*r;
end
T_G=100;
for i=1:1:node
    sys(node+1)=T_G*h_G(i)*r;
end

function sys=mdlOutputs(t,x,u)
global node c_M c_C c_G b Fai
qd=u(1);
dqd=u(2);
ddqd=u(3);

q=u(4);
dq=u(5);

for j=1:1:node
    h_M(j)=exp(-norm(q-c_M(:,j))^2/(b*b));
end
for j=1:1:node
    h_G(j)=exp(-norm(q-c_G(:,j))^2/(b*b));
end

z=[q;dq];
for j=1:1:node
    h_C(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
end

W_M=x(1:node)';
MSNN=W_M*h_M';
W_C=x(2*node+1:3*node)';
CDNN=W_C*h_C';
W_G=x(node+1:2*node)';
GSNN=W_G*h_G';

e=qd-q;
de=dqd-dq;
r=de+Fai*e;
dqr=dqd+Fai*e;

```

```
ddqr=ddqdr+Fai*de;
tolm=MSNN*ddqr+CDNN*dqr+GSNN;
Kr=0.10;
tolr=Kr*sign(r);
Kp=15;
Ki=15;
I=u(9);
tol=tolm+Kp*r+Ki*I+tolr;
sys(1)=tol(1);
sys(2)=MSNN;
sys(3)=CDNN;
sys(4)=GSNN;
```

S function for control integrator: chap7_2i.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 = 8;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [];
function sys=mdlOutputs(t,x,u)
qd=u(1);
dqdr=u(2);
ddqr=u(3);
```

```

q=u(4);
dq=u(5);
e=qd-q;
de=dqd-dq;
Fai=5;
r=de+Fai*e;
sys(1)=r;

```

S function for plant: chap7_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 = 5;
sizes.NumInputs = 4;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [0.15;0];
str = [];
ts = [0 0];
function sys=mdlDerivatives(t,x,u)
tol=u(1);
M=0.1+0.06*sin(x(1));
C=3*x(2)+3*cos(x(1));
m=0.020;g=9.8;l=0.05;
G=m*g*l*cos(x(1));
sys(1)=x(2);
sys(2)=1/M*(-C*x(2)-G+tol);

```

```
function sys=mdlOutputs(t,x,u) %PID book: page 416
M=0.1+0.06*sin(x(1));
C=3*x(2)+3*cos(x(1));
m=0.020;g=9.8;l=0.05;
G=m*g*l*cos(x(1));
sys(1)=x(1);
sys(2)=x(2);
sys(3)=M;
sys(4)=C;
sys(5)=G;
```

Plot program: chap7_2plot.m

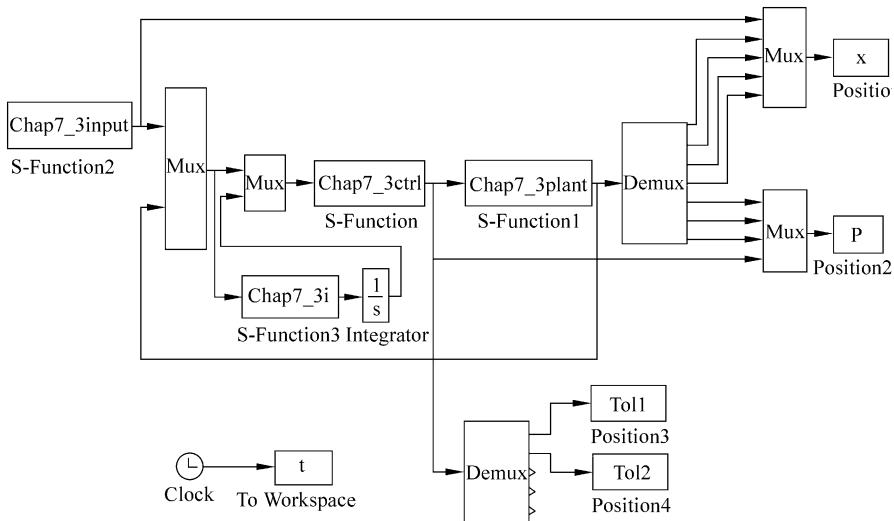
```
close all;
figure(1);
subplot(211);
plot(t,x(:,1),'r',t,x(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('position tracking');
legend('Ideal position signal','Position signal tracking');
subplot(212);
plot(t,x(:,2),'r',t,x(:,5),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('Ideal speed signal','Speed signal tracking');

figure(2);
plot(t,tol(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('Control input');
legend('Control input');

figure(3);
subplot(311);
plot(t,P(:,1),'r',t,P(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('M and MSNN');
legend('Ideal M','Estimated M');
subplot(312);
plot(t,P(:,2),'r',t,P(:,5),'k:','linewidth',2);
xlabel('time(s)');ylabel('C and CDNN');
legend('Ideal C','Estimated C');
subplot(313);
plot(t,P(:,3),'r',t,P(:,6),'k:','linewidth',2);
xlabel('time(s)');ylabel('G and GSNN');
legend('Ideal G','Estimated G');
```

Programs for Sect. 7.2.4.2

Simulink main program:chap7_3sim.mdl



Tracking position command program:chap7_3input.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)
S=2;
if S==1
    qd0=[0;0];
    qdtf=[1;2];
    td=1;
    if t<1
        qd1=qd0(1)+(-2*t.^3/td^3+3*t.^2/td^2)*(qdtf
            (1)-qd0(1));
        qd2=qd0(2)+(-2*t.^3/td^3+3*t.^2/td^2)*(qdtf
            (2)-qd0(2));
        d_qd1=(-6*t.^2/td^3+6*t./td^2)*(qdtf(1)-qd0
            (1));
        d_qd2=(-6*t.^2/td^3+6*t./td^2)*(qdtf(2)-qd0
            (2));
        dd_qd1=(-12*t/td^3+6/td^2)*(qdtf(1)-qd0(1));
        dd_qd2=(-12*t/td^3+6/td^2)*(qdtf(2)-qd0
            (2));
    else
        qd1=qdtf(1);
        qd2=qdtf(2);
        d_qd1=0;
        d_qd2=0;
        dd_qd1=0;
        dd_qd2=0;
    end
elseif S==2
    qd1=0.5*sin(pi*t);
    d_qd1=0.5*pi*cos(pi*t);
    dd_qd1=-0.5*pi*pi*sin(pi*t);

    qd2=sin(pi*t);
    d_qd2=pi*cos(pi*t);
    dd_qd2=-pi*pi*sin(pi*t);
end
sys(1)=qd1;
sys(2)=d_qd1;
sys(3)=dd_qd1;
sys(4)=qd2;
sys(5)=d_qd2;
sys(6)=dd_qd2;
```

S function for control law: chap7_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_M c_C c_G b
node=5;
c_M=[-1 -0.5 0 0.5 1;
      -1 -0.5 0 0.5 1];
c_G=[-1 -0.5 0 0.5 1;
      -1 -0.5 0 0.5 1];
c_C=[-1 -0.5 0 0.5 1;
      -1 -0.5 0 0.5 1;
      -1 -0.5 0 0.5 1;
      -1 -0.5 0 0.5 1];
b=10;
sizes = simsizes;
sizes.NumContStates = 10*node;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 5;
sizes.NumInputs = 15;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = zeros(1,10*node);
str = [];
ts = [];
function sys=mdlDerivatives(t,x,u)
global node c_M c_C c_G b
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];
for j=1:1:node
    h_M11(j)=exp(-norm(q-c_M(:,j))^2/(b*b));
    h_M12(j)=exp(-norm(q-c_M(:,j))^2/(b*b));
    h_M21(j)=exp(-norm(q-c_M(:,j))^2/(b*b));
    h_M22(j)=exp(-norm(q-c_M(:,j))^2/(b*b));
end

for j=1:1:node
    h_G1(j)=exp(-norm(q-c_G(:,j))^2/(b*b));
    h_G2(j)=exp(-norm(q-c_G(:,j))^2/(b*b));
end

z=[q1;q2;d_q1;d_q2];
for j=1:1:node
    h_C11(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
    h_C12(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
    h_C21(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
    h_C22(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
end

W_M11=[x(1:node)]';
W_M12=[x(node+1:node*2)]';
W_M21=[x(node*2+1:node*3)]';
W_M22=[x(node*3+1:node*4)]';

T_M11=5*eye(node);
T_M12=5*eye(node);
T_M21=5*eye(node);
T_M22=5*eye(node);

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);
r=de+Fai*e;

dqdr=[d_qd1;d_qd2];
dqqr=dqdr+Fai*e;
ddqd=[dd_qd1;dd_qd2];
ddqr=ddqd+Fai*de;

```

```

for i=1:1:node
    sys(i)=T_M11(i,i)*h_M11(i)*ddqr(1)*r(1);
    sys(i+node)=T_M12(i,i)*h_M12(i)*ddqr(2)*r(1);
    sys(i+node*2)=T_M21(i,i)*h_M21(i)*ddqr(1)*r(2);
    sys(i+node*3)=T_M22(i,i)*h_M22(i)*ddqr(2)*r(2);
end

W_G1=[x(node*4+1:node*5)]';
W_G2=[x(node*5+1:node*6)]';
T_G1=10*eye(node);
T_G2=10*eye(node);
for i=1:1:node
    sys(i+node*4)=T_G1(i,i)*h_G1(i)*r(1);
    sys(i+node*5)=T_G2(i,i)*h_G2(i)*r(2);
end

W_C11=[x(node*6+1:node*7)]';
W_C12=[x(node*7+1:node*8)]';
W_C21=[x(node*8+1:node*9)]';
W_C22=[x(node*9+1:node*10)]';

T_C11=10*eye(node);
T_C12=10*eye(node);
T_C21=10*eye(node);
T_C22=10*eye(node);

for i=1:1:node
    sys(i+node*6)=T_C11(i,i)*h_C11(i)*dqr(1)*r(1);
    sys(i+node*7)=T_C12(i,i)*h_C12(i)*ddqr(2)*r(1);
    sys(i+node*8)=T_C21(i,i)*h_C21(i)*dqr(1)*r(2);
    sys(i+node*9)=T_C22(i,i)*h_C22(i)*ddqr(2)*r(2);
end

function sys=mdlOutputs(t,x,u)
global node c_M c_C c_G b
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];
for j=1:1:node

```

```

h_M11(j)=exp(-norm(q-c_M(:,j))^2/(b*b));
h_M12(j)=exp(-norm(q-c_M(:,j))^2/(b*b));
h_M21(j)=exp(-norm(q-c_M(:,j))^2/(b*b));
h_M22(j)=exp(-norm(q-c_M(:,j))^2/(b*b));
end
for j=1:1:node
    h_G1(j)=exp(-norm(q-c_G(:,j))^2/(b*b));
    h_G2(j)=exp(-norm(q-c_G(:,j))^2/(b*b));
end
z=[q1;q2;d_q1;d_q2];
for j=1:1:node
    h_C11(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
    h_C12(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
    h_C21(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
    h_C22(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
end
W_M11=[x(1:node)]';
W_M12=[x(node+1:node*2)]';
W_M21=[x(node*2+1:node*3)]';
W_M22=[x(node*3+1:node*4)]';
MSNN=[W_M11*h_M11' W_M12*h_M12';
       W_M21*h_M21' W_M22*h_M22'];
Mm=norm(MSNN);
W_G1=[x(node*4+1:node*5)]';
W_G2=[x(node*5+1:node*6)]';
GSNN=[W_G1*h_G1';
       W_G2*h_G2'];
Gm=norm(GSNN);
W_C11=[x(node*6+1:node*7)]';
W_C12=[x(node*7+1:node*8)]';
W_C21=[x(node*8+1:node*9)]';
W_C22=[x(node*9+1:node*10)]';
CDNN=[W_C11*h_C11' W_C12*h_C12';
       W_C21*h_C21' W_C22*h_C22'];
Cm=norm(CDNN);
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);
r=de+Fai*e;

```

```

dqd=[d_qd1;d_qd2];
dqr=dqd+Fai*e;
ddqd=[dd_qd1;dd_qd2];
ddqr=ddqd+Fai*de;

tolm=MSNN*ddqr+CDNN*dqr+GSNN;

Kr=0.10;
tolr=Kr*sign(r);

Kp=100*eye(2);
Ki=100*eye(2);

I=[u(14);u(15)];
tol=tolm+Kp*r+Ki*I+tolr;

sys(1)=tol(1);
sys(2)=tol(2);
sys(3)=Gm;
sys(4)=Mm;
sys(5)=Cm;

```

S function for control integrator: chap7_3i.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 = 13;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [];
function sys=mdlOutputs(t,x,u)
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];
Hur=5*eye(2);
r=de+Hur*e;

sys(1:2)=r;
```

S function for plant:chap7_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
global p g
sizes = simsizes;
sizes.NumContStates = 4;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 7;
sizes.NumInputs = 5;
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
M=[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))];
tol=u(1:2);
dq=[x(2);x(4)];
S=inv(M)*(tol-C*dq-G);
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
M=[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))];
Gm=norm(G);
Cm=norm(C);
Mm=norm(M);
sys(1)=x(1);
sys(2)=x(2);
sys(3)=x(3);
sys(4)=x(4);
sys(5)=Gm;
sys(6)=Mm;
sys(7)=Cm;

```

Plot program:chap7_3plot.m

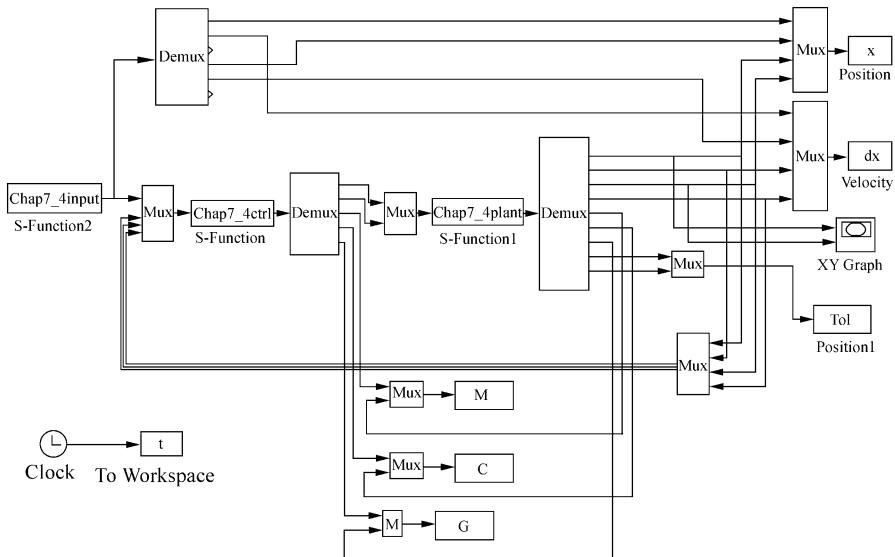
```
close all;
```

```
figure(1);
subplot(211);
plot(t,x(:,1),'r',t,x(:,7),'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('Ideal position for link 1','Position tracking for
link 1');
subplot(212);
plot(t,x(:,4),'r',t,x(:,9),'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('Ideal position for link 2','Position tracking for
link 2');
figure(2);
subplot(211);
plot(t,x(:,2),'r',t,x(:,8),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('Ideal speed for link 1','Speed tracking of link
1');
subplot(212);
plot(t,x(:,5),'r',t,x(:,10),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('Ideal speed for link 2','Speed signal tracking of
link2');
figure(3);
subplot(211);
plot(t,tol1(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('Contro input');
legend('Contro input of link 1');
subplot(212);
plot(t,tol2(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('Contro input');
legend('Contro input of link 2');

figure(4);
subplot(311);
plot(t,P(:,1),'r',t,P(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('G and GSNN');
legend('Ideal nrom of G','Estimated norm of G');
subplot(312);
plot(t,P(:,2),'r',t,P(:,5),'k:','linewidth',2);
xlabel('time(s)');ylabel('M and MSNN');
legend('Ideal norm of M','Estimated norm of M');
subplot(313);
plot(t,P(:,3),'r',t,P(:,6),'k:','linewidth',2);
xlabel('time(s)');ylabel('C and CDNN');
legend('Ideal norm of C','Estimated norm of C');
```

Programs for Sect. 7.3.4

Simulink main program: chap7_4sim.mdl



Ideal tracking input program:chap7_4input.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 = 1;
sizes.NumSampleTimes = 1;

```

```

sys = simsizes(sizes);
x0 = [];
str = [];
ts = [0 0];
function sys=mdlOutputs(t,x,u)
xd1=1+0.2*cos(pi*t);
d_xd1=-0.2*pi*sin(pi*t);
dd_xd1=-0.2*pi*pi*cos(pi*t);
xd2=1+0.2*sin(pi*t);
d_xd2=0.2*pi*cos(pi*t);
dd_xd2=-0.2*pi*pi*sin(pi*t);
sys(1)=xd1;
sys(2)=d_xd1;
sys(3)=dd_xd1;
sys(4)=xd2;
sys(5)=d_xd2;
sys(6)=dd_xd2;

```

S function for control law:chap7_4ctrl.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_M c_C c_G b ce
node=7;
c_M=zeros(2,node);
c_G=zeros(2,node);
c_C=zeros(4,node);
c_M=[0.25 0.5 0.75 1 1.25 1.5 1.75;
      0.25 0.5 0.75 1 1.25 1.5 1.75];
c_G=[0.25 0.5 0.75 1 1.25 1.5 1.75;
      0.25 0.5 0.75 1 1.25 1.5 1.75];
c_C=[0.25 0.5 0.75 1 1.25 1.5 1.75;
      0.25 0.5 0.75 1 1.25 1.5 1.75;
      -1.5 -1 -0.5 0 0.5 1.0 1.50;
      -1.5 -1 -0.5 0 0.5 1.0 1.50];

```

```

b=10;
ce=15.0;
sizes=simsizes;
sizes.NumContStates = 10*node;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 5;
sizes.NumInputs = 10;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=zeros(1,10*node);
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
global node c_M c_C c_G b ce
xd1=u(1);
d_xd1=u(2);
dd_xd1=u(3);
xd2=u(4);
d_xd2=u(5);
dd_xd2=u(6);

x1=u(7);
d_x1=u(8);
x2=u(9);
d_x2=u(10);

xx=[x1;x2];
for j=1:1:node
    h_M11(j)=exp(-norm(xx-c_M(:,j))^2/(b*b));
    h_M12(j)=exp(-norm(xx-c_M(:,j))^2/(b*b));
    h_M21(j)=exp(-norm(xx-c_M(:,j))^2/(b*b));
    h_M22(j)=exp(-norm(xx-c_M(:,j))^2/(b*b));
end

for j=1:1:node
    h_G1(j)=exp(-norm(xx-c_G(:,j))^2/(b*b));
    h_G2(j)=exp(-norm(xx-c_G(:,j))^2/(b*b));
end

z=[x1;x2;d_x1;d_x2];
for j=1:1:node
    h_C11(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
    h_C12(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
    h_C21(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
    h_C22(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
end

```

```

W_M11=[x(1:node)]';
W_M12=[x(node+1:node*2)]';
W_M21=[x(node*2+1:node*3)]';
W_M22=[x(node*3+1:node*4)]';

e1=xd1-x1;
e2=xd2-x2;
de1=d_xd1-d_x1;
de2=d_xd2-d_x2;
e=[e1;e2];
de=[de1;de2];
Hur=ce*eye(2);
r=de+Hur*e;

dxd=[d_xd1;d_xd2];
dxr=dxr+Hur*e;
ddxd=[dd_xd1;dd_xd2];
ddxr=ddxr+Hur*de;

T_M11=2*eye(node);
T_M12=2*eye(node);
T_M21=2*eye(node);
T_M22=2*eye(node);
for i=1:1:node
    sys(i)=T_M11(i,i)*h_M11(i)*ddxr(1)*r(1);
    sys(i+node)=T_M12(i,i)*h_M12(i)*ddxr(2)*r(1);
    sys(i+node*2)=T_M21(i,i)*h_M21(i)*ddxr(1)*r(2);
    sys(i+node*3)=T_M22(i,i)*h_M22(i)*ddxr(2)*r(2);
end

W_G1=[x(node*4+1:node*5)]';
W_G2=[x(node*5+1:node*6)]';
T_G1=5*eye(node);
T_G2=5*eye(node);
for i=1:1:node
    sys(i+node*4)=T_G1(i,i)*h_G1(i)*r(1);
    sys(i+node*5)=T_G2(i,i)*h_G2(i)*r(2);
end

W_C11=[x(node*6+1:node*7)]';
W_C12=[x(node*7+1:node*8)]';
W_C21=[x(node*8+1:node*9)]';
W_C22=[x(node*9+1:node*10)]';

T_C11=0.5*eye(node);
T_C12=0.5*eye(node);
T_C21=0.5*eye(node);
T_C22=0.5*eye(node);

```

```

for i=1:1:node
    sys(i+node*6)=T_C11(i,i)*h_C11(i)*dxr(1)*r(1);
    sys(i+node*7)=T_C12(i,i)*h_C12(i)*ddxr(2)*r(1);
    sys(i+node*8)=T_C21(i,i)*h_C21(i)*dxr(1)*r(2);
    sys(i+node*9)=T_C22(i,i)*h_C22(i)*ddxr(2)*r(2);
end

function sys=mdlOutputs(t,x,u)
global node c_M c_C c_G b ce
xd1=u(1);
d_xd1=u(2);
dd_xd1=u(3);
xd2=u(4);
d_xd2=u(5);
dd_xd2=u(6);

x1=u(7);
d_x1=u(8);
x2=u(9);
d_x2=u(10);

xx=[x1;x2];
for j=1:1:node
    h_M11(j)=exp(-norm(xx-c_M(:,j))^2/(b*b));
    h_M12(j)=exp(-norm(xx-c_M(:,j))^2/(b*b));
    h_M21(j)=exp(-norm(xx-c_M(:,j))^2/(b*b));
    h_M22(j)=exp(-norm(xx-c_M(:,j))^2/(b*b));
end

for j=1:1:node
    h_G1(j)=exp(-norm(xx-c_G(:,j))^2/(b*b));
    h_G2(j)=exp(-norm(xx-c_G(:,j))^2/(b*b));
end

z=[x1;x2;d_x1;d_x2];
for j=1:1:node
    h_C11(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
    h_C12(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
    h_C21(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
    h_C22(j)=exp(-norm(z-c_C(:,j))^2/(b*b));
end

W_M11=[x(1:node)]';
W_M12=[x(node+1:node*2)]';
W_M21=[x(node*2+1:node*3)]';
W_M22=[x(node*3+1:node*4)]';

```

```

MSNN_g=[W_M11*h_M11' W_M12*h_M12';
         W_M21*h_M21' W_M22*h_M22'];
norm_Mp=norm(MSNN_g);
W_G1=[x(node*4+1:node*5)]';
W_G2=[x(node*5+1:node*6)]';
GSNN_g=[W_G1*h_G1';
         W_G2*h_G2'];
norm_Gp=norm(GSNN_g);
W_C11=[x(node*6+1:node*7)]';
W_C12=[x(node*7+1:node*8)]';
W_C21=[x(node*8+1:node*9)]';
W_C22=[x(node*9+1:node*10)]';
CDNN_g=[W_C11*h_C11' W_C12*h_C12';
         W_C21*h_C21' W_C22*h_C22'];
norm_Cp=norm(CDNN_g);

e1=xd1-x1;
e2=xd2-x2;
de1=d_xd1-d_x1;
de2=d_xd2-d_x2;
e=[e1;e2];
de=[de1;de2];
Hur=ce*eye(2);
r=de+Hur*e;

dxd=[d_xd1;d_xd2];
dxr=dxd+Hur*e;
ddxd=[dd_xd1;dd_xd2];
ddxr=ddxd+Hur*de;

Ks=0.5;
K=30*eye(2);
Fx=MSNN_g*ddxr+CDNN_g*dxr+GSNN_g+K*r+Ks*sign(r);

sys(1)=Fx(1);
sys(2)=Fx(2);
sys(3)=norm_Mp;
sys(4)=norm_Cp;
sys(5)=norm_Gp;

```

S function for plant:chap7_4plant.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 J Mx Cx Gx l1 l2
l1=1;l2=1;
sizes = simsizes;
sizes.NumContStates = 4;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 9;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[1 0 1 0];
str=[];
ts=[];
J=0;Mx=0;Cx=0;Gx=0;
function sys=mdlDerivatives(t,x,u)
global J Mx Cx Gx l1 l2
P=[1.66 0.42 0.63 3.75 1.25];
g=9.8;
L=[l1^2 l2^2 l1*l2 l1 l2];
if t<4.0 %Simulate time-varying load
    pl=0;
else
    pl=0.5;
end
Mm=P+pl*L;
Q=(x(1)^2+x(3)^2-l1^2-l2^2)/(2*l1*l2);
q2=acos(Q);
dq2=-1/sqrt(1-Q^2);
A=x(3)/x(1);
p1=atan(A);
d_p1=1/(1+A^2);
B=sqrt(x(1)^2+x(3)^2+l1^2-l2^2)/(2*l1*sqrt(x(1)^2+x(3)^2));

```

```

p2=acos(B);
d_p2=-1/sqrt(1-B^2);

if q2>0
    q1=p1-p2;
    dq1=d_p1-d_p2;

else
    q1=p1+p2;
    dq1=d_p1+d_p2;
end

J=[-sin(q1)-sin(q1+q2) -sin(q1+q2);
   cos(q1)+cos(q1+q2) cos(q1+q2)];
d_J=[-dq1*cos(q1)-(dq1+dq2)*cos(q1+q2) -(dq1+dq2)*cos
      (q1+q2);
      -dq1*sin(q1)-(dq1+dq2)*sin(q1+q2) -(dq1+dq2)*sin
      (q1+q2)];

m1=Mm(1);m2=Mm(2);m3=Mm(3);m4=Mm(4);m5=Mm(5);

M=[m1+m2+2*m3*cos(q2) m2+m3*cos(q2);
    m2+m3*cos(q2) m2];
C=[-m3*dq2*sin(q2) -m3*(dq1+dq2)*sin(q2);
   m3*dq1*sin(q2) 0];
G=[m4*g*cos(q1)+m5*g*cos(q1+q2);
   m5*g*cos(q1+q2)];

Mx=(inv(J))'*M*inv(J);
Cx=(inv(J))'*(C-M*inv(J)*d_J)*inv(J);
Gx=(inv(J))'*G;

Fx=u(1:2);
dx=[x(2);x(4)];

ddx=inv(Mx)*(Fx-Cx*dx-Gx); %ddx

sys(1)=x(2);
sys(2)=ddx(1);
sys(3)=x(4);
sys(4)=ddx(2);
function sys=mdlOutputs(t,x,u)
global J Mx Cx Gx l1 l2
norm_M=norm(Mx);
norm_C=norm(Cx);
norm_G=norm(Gx);
Fx=u(1:2);

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

```

```

sys(4)=x(4);
sys(5)=norm_M;
sys(6)=norm_C;
sys(7)=norm_G;
sys(8:9)=J'*Fx; %Practical control input

```

Plot program: chap7_4plot.m

```

close all;

figure(1);
subplot(211);
plot(t,x(:,1),'r',t,x(:,3),'k:','linewidth',2);
xlabel('time(s)');ylabel('position tracking of x1 axis');
legend('Ideal position of x1','Position tracking of x1');
subplot(212);
plot(t,x(:,2),'r',t,x(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('position tracking of x2 axis');
legend('Ideal position of x2','Position tracking of x2');

figure(2);
subplot(211);
plot(t,dx(:,1),'r',t,dx(:,3),'k:','linewidth',2);
xlabel('time(s)');ylabel('velocity tracking of x1 axis');
legend('Ideal speed of x1','Speed tracking of x1');
subplot(212);
plot(t,dx(:,2),'r',t,dx(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('velocity tracking of x2 axis');
legend('Ideal speed of x2','Speed tracking of x2');

figure(3);
plot(t,tol(:,1),'r',t,tol(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('Control input tol1 and tol2');
legend('Control input for link 1','Control input for link 2');

figure(4);
subplot(311);
plot(t,M(:,1),'r',t,M(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('M and estimated M');
legend('Ideal norm of Msx','Estimated norm of Mx');
subplot(312);
plot(t,C(:,1),'r',t,C(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('C and estimated C');

```

```
legend('Ideal norm of Cx','Estimated norm of Cx');
subplot(313);
plot(t,G(:,1),'r',t,G(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('G and estimated G');
legend('Ideal norm of Gx','Estimated norm of Gx');
figure(5);

plot(x(:,1),x(:,2),'r','linewidth',2);
xlabel('x1 axis');ylabel('x2 axis');
hold on;
plot(x(:,3),x(:,4),'k:','linewidth',2);
xlabel('x1 axis');ylabel('x2 axis');
legend('Ideal trajectory','Tracking trajectory');
```

References

1. Ge SS, Lee TH, Harris CJ (1998) Adaptive neural network control of robotic manipulators. World Scientific, London
2. Ge SS, Hang CC, Lee TH, Zhang T (2001) Stable adaptive neural network control. Kluwer, Boston
3. Ge SS, Hang CC (1996) Direct adaptive neural network control of robots. Int J Syst Sci 27 (6):533–542
4. Ge SS, Hang CC, Woon LC (1997) Adaptive neural network control of robot manipulators in task space. IEEE Trans Ind Electron 44(6):746–752
5. Slotine JJ, Li W (1987) On the adaptive control of robot manipulators. Int J Robot Res 6 (3):49–59
6. Shen TL (2004) Robot robust control foundation. Tsinghua University Press, Beijing (in Chinese)
7. Lewis FL, Abdallah CT, Dawson DM (1993) Control of robot manipulators. Maxwell Macmillan, New York
8. Khalil HK (2002) Nonlinear systems, 3rd edn. Pearson Education, Inc., London
9. Desoer C, Vidyasagar M (1975) Feedback systems: input-output properties. Academic, New York

Chapter 8

Backstepping Control with RBF

Abstract This chapter introduces backstepping controller design with RBF neural network approximation. Several controller design examples for mechanical systems are given, including backstepping controller for inverted pendulum, backstepping controller for single-link flexible joint robot, and adaptive backstepping controller for single-link flexible joint robot.

Keywords RBF neural network • Backstepping control • Single-link flexible joint robot

8.1 Introduction

Many physical systems do not satisfy the matching condition, which includes some uncertain flexible joint robots in particular.

The backstepping control approach has shown itself very effective in dealing with systems with multiple dynamics and with mismatched uncertainties, such as mechanical systems driven by electrical systems or multiple coupled mechanical systems [1, 2].

The idea of backstepping design is that some appropriate functions of state variables are selected recursively as pseudocontrol inputs for lower dimension subsystems of the overall system. Each backstepping stage results in a new pseudocontrol design, expressed in terms of the pseudocontrol designs from the preceding design stages. When the procedure terminates, a feedback design for the true control input results, which achieves the original design objective by virtue of a final Lyapunov function, formed by summing the Lyapunov functions associated with each individual design stage [3]. The backstepping design provides a systematic framework for the design of tracking and regulation strategies, suitable for a large class of state feedback linearizable nonlinear systems.

A major problem with the backstepping approach is that certain functions must be “linear in the unknown parameters” and some very tedious analysis is needed to determine “regression matrices.”

The possible solution of the problem is to use neural networks (NNs) to estimate certain nonlinear functions. A stable neural controller design using backstepping can be found in [4], where rigorous stability proofs are also provided.

A unified and general approach to backstepping control of nonlinear systems using neural networks is presented in [5]. By using the NNs in each stage of the backstepping procedure to estimate certain nonlinear functions, one can design control law using the backstepping approach, but the linear-in-the-parameters (LIP) assumption is not needed, and no regression matrices need to be found. The NNs weights are also tuned online, with no learning phase required. The boundedness of the tracking error and weight updates is guaranteed.

The very rapid developments described in adaptive and robust control techniques are accompanied by an increase in the use of neural networks for system identification-based control [6–9]. With the help of neural networks, the linear-in-the-parameters assumption of nonlinear function and the determination of regression matrices can be avoided.

A large number of backstepping design schemes are reported that combine the backstepping technique with adaptive neural network [10–13]. For example, an adaptive neural network control via backstepping design was presented for a class of minimum-phase nonlinear systems with known relative degree [4], and the combination of NN with backstepping has been proposed for multiple-input–multiple-output nonlinear systems in block-triangular form [14]. Based on implicit function theory, adaptive neural network control using backstepping was constructed for two special classes of non-affine pure-feedback systems [15].

In this chapter, we take single-rank inverted pendulum and single-link flexible joint robot as two typical examples to explain backstepping controller design with RBF.

8.2 Backstepping Control for Inverted Pendulum

The basic idea of backstepping design is that a complex nonlinear system is decomposed into the subsystems and the degree of each subsystem doesn’t exceed that of the whole system. Accordingly, the Lyapunov function and medial fictitious control are designed respectively, and the whole system is obtained through “backstepping.” Thus, the control rule is designed thoroughly. The backstepping method is called as the back-deduce method, and the desired dynamic indexes are satisfied.

8.2.1 System Description

Suppose the plant is a nonlinear system as follows:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = f(x, t) + g(x, t)u \end{cases} \quad (8.1)$$

where $f(x, t)$ and $g(x, t)$ are the nonlinear functions and $g(x, t) \neq 0$

Define $e_1 = x_1 - x_{1d}$, where x_{1d} is the ideal position signal, the goal is $e_1 \rightarrow 0$ and $\dot{e}_1 \rightarrow 0$

8.2.2 Controller Design

Basic backstepping control is designed as follows:

Step 1

$$\dot{e}_1 = \dot{x}_1 - \dot{x}_{1d} = x_2 - \dot{x}_{1d} \quad (8.2)$$

To realize $e_1 \rightarrow 0$, design Lyapunov function as

$$V_1 = \frac{1}{2} e_1^2 \quad (8.3)$$

Then

$$\dot{V}_1 = e_1 \dot{e}_1 = e_1 (x_2 - \dot{x}_{1d})$$

To realize $\dot{V}_1 < 0$, if we choose $x_2 - \dot{x}_{1d} = -k_1 e_1$, $k_1 > 0$, then $\dot{V}_1 = -k_1 e_1^2$.

Step 2 To realize $x_2 - \dot{x}_{1d} = -k_1 e_1$, that is, $x_2 = \dot{x}_{1d} - k_1 e_1$, we choose virtual control as

$$x_{2d} = \dot{x}_{1d} - k_1 e_1 \quad (8.4)$$

To realize $x_2 \rightarrow x_{2d}$, we get a new error

$$e_2 = x_2 - x_{2d} \quad (8.5)$$

Then, $\dot{e}_2 = \dot{x}_2 - \dot{x}_{2d} = f(x, t) + g(x, t)u - \dot{x}_{2d}$

To realize $e_2 \rightarrow 0$ and $e_1 \rightarrow 0$, design Lyapunov function as

$$V_2 = V_1 + \frac{1}{2} e_2^2 = \frac{1}{2} (e_1^2 + e_2^2)$$

Then

$$\begin{aligned}\dot{V}_2 &= e_1(x_2 - \dot{x}_{1d}) + e_2\dot{e}_2 \\ &= e_1(x_{2d} + e_2 - \dot{x}_{1d}) + e_2\dot{e}_2 \\ &= -k_1e_1^2 + e_1e_2 + e_2(f(x, t) + g(x, t)u - \dot{x}_{2d})\end{aligned}$$

To realize $\dot{V}_2 < 0$, we choose

$$e_1 + f(x, t) + g(x, t)u - \dot{x}_{2d} = -k_2e_2, \quad k_2 > 0 \quad (8.6)$$

Then

$$\dot{V}_2 = -k_1e_1^2 - k_2e_2^2$$

From (8.6), consider $\dot{x}_{2d} = \ddot{x}_{1d} - k_1\dot{e}_1$, and we can get the control law

$$u = \frac{1}{g(x, t)}(-k_2e_2 + \dot{x}_{2d} - e_1 - f(x, t)) \quad (8.7)$$

In addition, if $e_1 \rightarrow 0$ and $e_2 \rightarrow 0$, then we can get $\dot{e}_1 = x_2 - \dot{x}_{1d} = e_2 + x_{2d} - \dot{x}_{1d} = e_2 - k_1e_1 \rightarrow 0$.

To realize the control law (8.7), exact values of modeling information $g(x)$, $f(x)$ are needed, which are difficult in practical engineering. We can use RBF to approximate them.

8.2.3 Simulation Example

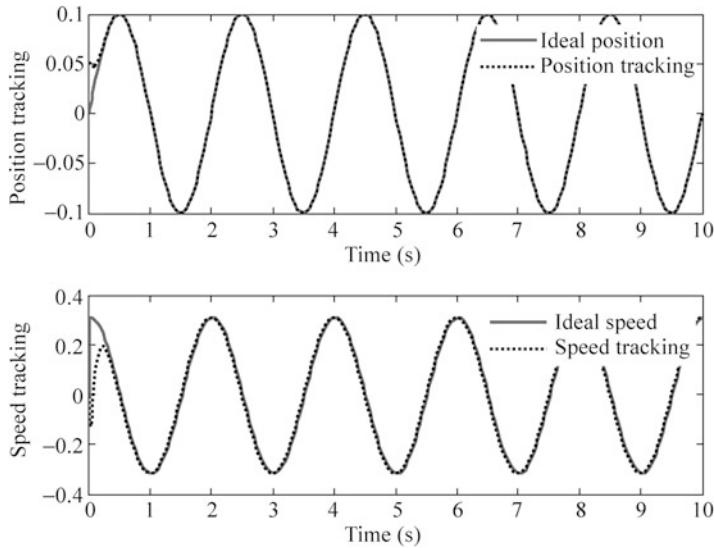
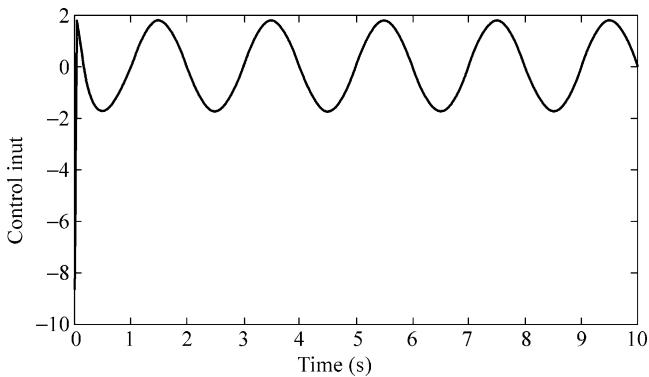
Consider one link inverted pendulum as follows:

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

where $f(\mathbf{x}) = \frac{g \sin x_1 - mx_2^2 \cos x_1 \sin x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}$, $g(\mathbf{x}) = \frac{\cos x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}$, and x_1 and x_2 are oscillation angle and oscillation rate, respectively. $g = 9.8 \text{ m/s}^2$, $m_c = 1 \text{ kg}$ is the vehicle mass, $m_c = 1 \text{ kg}$, m is the mass of pendulum bar, $m = 0.1 \text{ kg}$, l is one half of pendulum length, $l = 0.5 \text{ m}$, and u is the control input.

Consider the desired trajectory as $x_d(t) = 0.1 \sin(\pi t)$, adopt the control law as (8.7), and select $k_1 = 35$ and $k_2 = 15$. The initial state of the inverted pendulum is $[\pi/60, 0]$. Simulation results are shown in Figs. 8.1 and 8.2.

The Simulink program of this example is chap8_1sim.mdl, and the Matlab programs of the example are given in the [Appendix](#).

**Fig. 8.1** Position and speed tracking**Fig. 8.2** Control input

8.3 Backstepping Control Based on RBF for Inverted Pendulum

8.3.1 System Description

Suppose the plant is a nonlinear system as follows:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = f(x, t) + g(x, t)u \end{cases} \quad (8.8)$$

where $f(x, t)$ and $g(x, t)$ are the nonlinear functions and $g(x, t) \neq 0$

Define $e_1 = x_1 - x_{1d}$, where x_{1d} is the ideal position signal, the goal is $e_1 \rightarrow 0$ and $\dot{e}_1 \rightarrow 0$

8.3.2 Backstepping Controller Design

Basic backstepping control is designed as follows.

Step 1

$$\dot{e}_1 = \dot{x}_1 - \dot{x}_{1d} = x_2 - \dot{x}_{1d} \quad (8.9)$$

To realize $e_1 \rightarrow 0$, design Lyapunov function as

$$V_1 = \frac{1}{2} e_1^2 \quad (8.10)$$

Then

$$\dot{V}_1 = e_1 \dot{e}_1 = e_1 (x_2 - \dot{x}_{1d})$$

To realize $\dot{V}_1 < 0$, if we choose $x_2 - \dot{x}_{1d} = -k_1 e_1$, $k_1 > 0$, then we would have $\dot{V}_1 = -k_1 e_1^2$.

Step 2 To realize $x_2 - \dot{x}_{1d} = -k_1 e_1$, that is $x_2 = \dot{x}_{1d} - k_1 e_1$, we choose virtual control as

$$x_{2d} = \dot{x}_{1d} - k_1 e_1 \quad (8.11)$$

To realize $x_2 \rightarrow x_{2d}$, we get a new error

$$e_2 = x_2 - x_{2d} \quad (8.12)$$

Then, $\dot{e}_2 = \dot{x}_2 - \dot{x}_{2d} = f(x, t) + g(x, t)u - \dot{x}_{2d}$

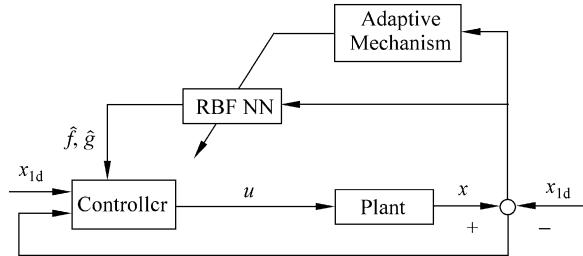
To realize $e_2 \rightarrow 0$ and $e_1 \rightarrow 0$, design Lyapunov function as

$$V_2 = V_1 + \frac{1}{2} e_2^2 = \frac{1}{2} (e_1^2 + e_2^2)$$

Then

$$\begin{aligned} \dot{V}_2 &= e_1 (x_2 - \dot{x}_{1d}) + e_2 \dot{e}_2 \\ &= e_1 (x_{2d} + e_2 - \dot{x}_{1d}) + e_2 \dot{e}_2 \\ &= -k_1 e_1^2 + e_1 e_2 + e_2 (f + gu - \dot{x}_{2d}) \\ &= -k_1 e_1^2 + e_1 e_2 + e_2 (f + \hat{g}u + (g - \hat{g})u - \dot{x}_{2d}) \end{aligned}$$

Fig. 8.3 Block diagram of the control scheme



where \hat{g} is estimation of g .

To realize $\dot{V}_2 < 0$, we design control law as

$$u = \frac{-e_1 - \hat{f} + \dot{x}_{2d} - k_2 e_2}{\hat{g}}, \quad k_2 > 0 \quad (8.13)$$

where \hat{f} is estimation of f .

Then, we can get

$$\dot{V}_2 = -k_1 e_1^2 - k_2 e_2^2 + e_2(f - \hat{f}) + e_2(g - \hat{g})u \quad (8.14)$$

8.3.3 Adaptive Law Design

The unknown functions of \hat{f} and \hat{g} can be approximated by neural network. Figure 8.3 shows the closed-loop neural-based adaptive control scheme.

We use two RBFs to approximate f and g respectively as follows:

$$\begin{cases} f = \mathbf{W}_1^T \mathbf{h}_1 + \boldsymbol{\varepsilon}_1 \\ g = \mathbf{W}_2^T \mathbf{h}_2 + \boldsymbol{\varepsilon}_2 \end{cases} \quad (8.15)$$

where W_i is the ideal neural network weight value, h_i is the Gaussian function, $i = 1, 2$, $\boldsymbol{\varepsilon}_i$ is the approximation error, and $\|\boldsymbol{\varepsilon}\| = \|[\boldsymbol{\varepsilon}_1 \quad \boldsymbol{\varepsilon}_2]^T\| < \boldsymbol{\varepsilon}_N$, $\|\mathbf{W}_i\|_F \leq W_M$. Define

$$\begin{cases} \hat{f} = \hat{\mathbf{W}}_1^T \mathbf{h}_1 \\ \hat{g} = \hat{\mathbf{W}}_2^T \mathbf{h}_2 \end{cases} \quad (8.16)$$

where $\hat{\mathbf{W}}_i$ is the weight value estimation.

Define

$$\mathbf{Z} = \begin{bmatrix} \mathbf{W}_1 & \\ & \mathbf{W}_2 \end{bmatrix}, \quad \|\mathbf{Z}\|_F \leq \mathbf{Z}_M, \quad \hat{\mathbf{Z}} = \begin{bmatrix} \hat{\mathbf{W}}_1 & \\ & \hat{\mathbf{W}}_2 \end{bmatrix}, \quad \tilde{\mathbf{Z}} = \mathbf{Z} - \hat{\mathbf{Z}}$$

Design Lyapunov function as

$$V = \frac{1}{2} \boldsymbol{\xi}^T \boldsymbol{\xi} + \frac{1}{2} \text{tr}(\tilde{\mathbf{Z}}^T \boldsymbol{\Gamma}^{-1} \tilde{\mathbf{Z}}) \quad (8.17)$$

where $V_2 = \frac{1}{2} \boldsymbol{\xi}^T \boldsymbol{\xi}$, $\eta > 0$, $\boldsymbol{\Gamma}$ is a positive-definite matrix with proper dimension, $\boldsymbol{\Gamma} = \begin{bmatrix} \Gamma_1 & \\ & \Gamma_2 \end{bmatrix}$, and $\boldsymbol{\xi} = [e_1 \ e_2]^T$

Let adaptive law as

$$\dot{\tilde{\mathbf{Z}}} = \boldsymbol{\Gamma} \mathbf{h} \boldsymbol{\xi}^T - n \boldsymbol{\Gamma} \|\boldsymbol{\xi}\| \hat{\mathbf{Z}} \quad (8.18)$$

where $\mathbf{h} = [h_1 \ h_2]^T$ and n is a positive number.

From (8.14), (8.15), and (8.16), we have

$$\begin{aligned} \dot{V} &= \boldsymbol{\xi}^T \dot{\boldsymbol{\xi}} + \text{tr}(\tilde{\mathbf{Z}}^T \boldsymbol{\Gamma}^{-1} \dot{\tilde{\mathbf{Z}}}) \\ &= -k_1 e_1^2 - k_2 e_2^2 + (\tilde{\mathbf{W}}_1^T h_1 + \varepsilon_1) e_2 + (\tilde{\mathbf{W}}_2^T h_2 + \varepsilon_2) e_3 + \text{tr}(\tilde{\mathbf{Z}}^T \boldsymbol{\Gamma}^{-1} \dot{\tilde{\mathbf{Z}}}) \end{aligned}$$

where $\tilde{\mathbf{W}}_i^T = \mathbf{W}_i^T - \hat{\mathbf{W}}_i^T$, $i = 1, 2$

Then

$$\begin{aligned} \dot{V} &= -\boldsymbol{\xi}^T \mathbf{K}_e \boldsymbol{\xi} + \boldsymbol{\xi}^T \boldsymbol{\epsilon} + \boldsymbol{\xi}^T \tilde{\mathbf{Z}} \mathbf{h} + \text{tr}(\tilde{\mathbf{Z}}^T \boldsymbol{\Gamma}^{-1} \dot{\tilde{\mathbf{Z}}}) + \tilde{m} e_4 u \\ &= -\boldsymbol{\xi}^T \mathbf{K}_e \boldsymbol{\xi} + \boldsymbol{\xi}^T \boldsymbol{\epsilon} + \text{tr}(\tilde{\mathbf{Z}}^T \boldsymbol{\Gamma}^{-1} \dot{\tilde{\mathbf{Z}}} + \tilde{\mathbf{Z}}^T \mathbf{h} \boldsymbol{\xi}^T) + \tilde{m} e_4 u \end{aligned}$$

where $\mathbf{K}_e = [k_1 \ k_2]^T$ and $\boldsymbol{\epsilon} = [e_1 \ e_2]^T$

Since $\dot{\tilde{\mathbf{Z}}} = -\dot{\tilde{\mathbf{Z}}}$, submitting the adaptive law (8.18), we have

$$\dot{V} = -\boldsymbol{\xi}^T \mathbf{K}_e \boldsymbol{\xi} + \boldsymbol{\xi}^T \boldsymbol{\epsilon} + n \|\boldsymbol{\xi}\| \text{tr}(\tilde{\mathbf{Z}}^T (\mathbf{Z} - \tilde{\mathbf{Z}})) \quad (8.19)$$

According to Schwarz inequality, we have $\text{tr}(\tilde{\mathbf{Z}}^T (\mathbf{Z} - \tilde{\mathbf{Z}})) \leq \|\tilde{\mathbf{Z}}\|_F \|\mathbf{Z}\|_F - \|\tilde{\mathbf{Z}}\|_F^2$, since $K_{\min} \|\boldsymbol{\xi}\|^2 \leq \boldsymbol{\xi}^T \mathbf{K} \boldsymbol{\xi}$, K_{\min} is the minimum eigenvalue of \mathbf{K} , (8.19) becomes

$$\begin{aligned}\dot{V} &\leq -K_{\min} \|\xi\|^2 + \varepsilon_N \|\xi\| + n \|\xi\| \left(\|\tilde{\mathbf{Z}}\|_{\text{F}} \|\mathbf{Z}\|_{\text{F}} - \|\tilde{\mathbf{Z}}\|_{\text{F}}^2 \right) \\ &\leq -\|\xi\| \left(K_{\min} \|\xi\| - \varepsilon_N + n \|\tilde{\mathbf{Z}}\|_{\text{F}} (\|\tilde{\mathbf{Z}}\|_{\text{F}} - Z_M) \right)\end{aligned}$$

Since

$$\begin{aligned}K_{\min} \|\xi\| - \varepsilon_N + n \left(\|\tilde{\mathbf{Z}}\|_{\text{F}}^2 - \|\tilde{\mathbf{Z}}\|_{\text{F}} Z_M \right) \\ = K_{\min} \|\xi\| - \varepsilon_N + n \left(\|\tilde{\mathbf{Z}}\|_{\text{F}} - \frac{1}{2} Z_M \right)^2 - \frac{n}{4} Z_M^2\end{aligned}$$

this implies that $\dot{V} < 0$ as long as

$$\|\xi\| > \frac{\varepsilon_N + \frac{n}{4} Z_M^2}{K_{\min}} \quad \text{or} \quad \|\tilde{\mathbf{Z}}\|_{\text{F}} > \frac{1}{2} Z_M + \sqrt{\frac{Z_M^2}{4} + \frac{\varepsilon_N}{n}} \quad (8.20)$$

From the above expression, we can see that the tracking performance is related to the value of ε_N , n and K_{\min} .

In addition, if $e_1 \rightarrow 0$ and $e_2 \rightarrow 0$, then we can get $\dot{e}_1 = \mathbf{x}_2 - \dot{\mathbf{x}}_{1d} = \mathbf{e}_2 + \mathbf{x}_{2d} - \dot{\mathbf{x}}_{1d} = \mathbf{e}_2 - \mathbf{k}_1 \mathbf{e}_1 \rightarrow 0$

8.3.4 Simulation Example

Consider one link inverted pendulum as follows:

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{g \sin x_1 - mlx_2^2 \cos x_1 \sin x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))} + \frac{\cos x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))} u\end{aligned}$$

where x_1 and x_2 are the oscillation angle and oscillation rate, respectively. $g = 9.8 \text{ m/s}^2$, $m_c = 1 \text{ kg}$ is the vehicle mass, $m_c = 1 \text{ kg}$, m is the mass of pendulum bar, $m = 0.1 \text{ kg}$, l is one half of pendulum length, $l = 0.5 \text{ m}$, and u is the control input.

Consider the desired trajectory as $\mathbf{x}_d(t) = 0.1 \sin t$, adopt the control law as (8.13) and adaptive law (8.18), and select $\Gamma_1 = 500$ and $\Gamma_2 = 0.50$, $n = 0.10$, and $\mathbf{k}_1 = \mathbf{k}_2 = 35$

Two RBF neural networks are designed, for each Gaussian function, the parameters of \mathbf{c}_i and b_i are designed as $[-0.5 \quad -0.25 \quad 0 \quad 0.25 \quad 0.5]$ and 15. The initial weight value of each neural net in hidden layer is chosen as 0.10.

In the control law (8.13), to prevent from singularity, we should prevent the item \hat{g} value from changing frequently; thus, we should choose small Γ_2 in adaptive law (8.18).

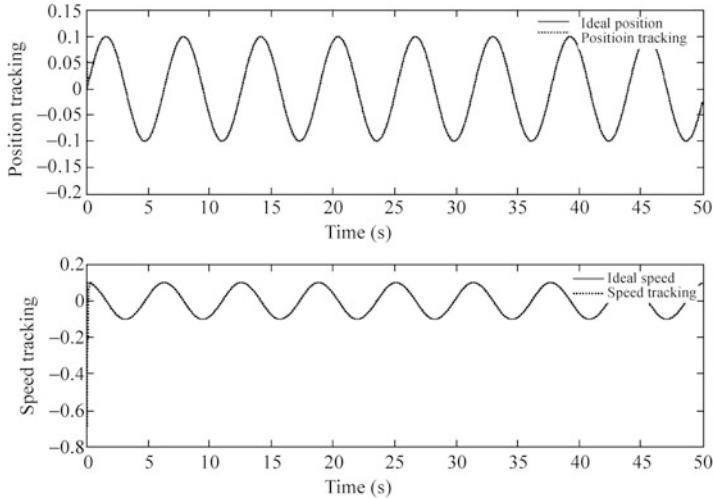


Fig. 8.4 Position and speed tracking

The initial state of the inverted pendulum is $[\pi/60, 0]$. Simulation results are shown from Figs. 8.4, 8.5, and 8.6. The variation of $\hat{f}(\cdot)$ and $\hat{g}(\cdot)$ do not converge to $f(\cdot)$ and $g(\cdot)$. This is due to the fact that the desired trajectory is not persistently exciting and the tracking error performance can be achieved by many possible values of $\hat{f}(\cdot)$ and $\hat{g}(\cdot)$, besides the true $f(\cdot)$ and $g(\cdot)$, which has been explained in Sect. 1.5, and this occurs quite often in real-world application.

The Simulink program of this example is chap8_2sim.mdl, and the Matlab programs of the example are given in the [Appendix](#).

8.4 Backstepping Control for Single-Link Flexible Joint Robot

8.4.1 System Description

The dynamic equation of single-link flexible joint robot is

$$\begin{cases} I\ddot{q}_1 + MgL \sin q_1 + K(q_1 - q_2) = 0 \\ J\ddot{q}_2 + K(q_2 - q_1) = u \end{cases} \quad (8.21)$$

where $q_1 \in R^n$ and $q_2 \in R^n$ are the link angular and motor angular respectively, K is the stiffness of the link, $u \in R^n$ is control input, J is motor inertia, I is link inertia, M is the link mass, and L is the length from the joint to the center of the link.

Choose $x_1 = q_1$, $x_2 = \dot{q}_1$, $x_3 = q_2$, $x_4 = \dot{q}_2$, and then the system (8.21) can be written as

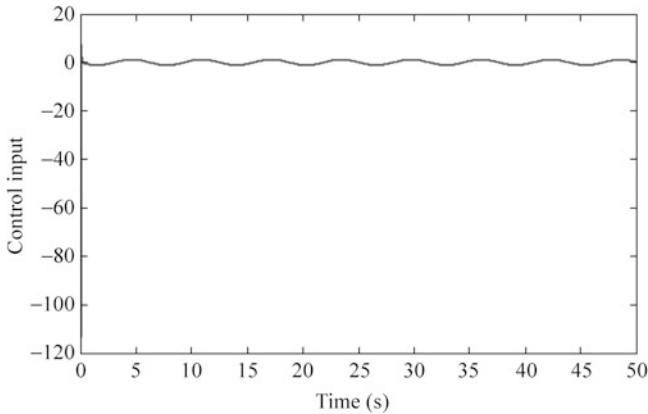


Fig. 8.5 Control input

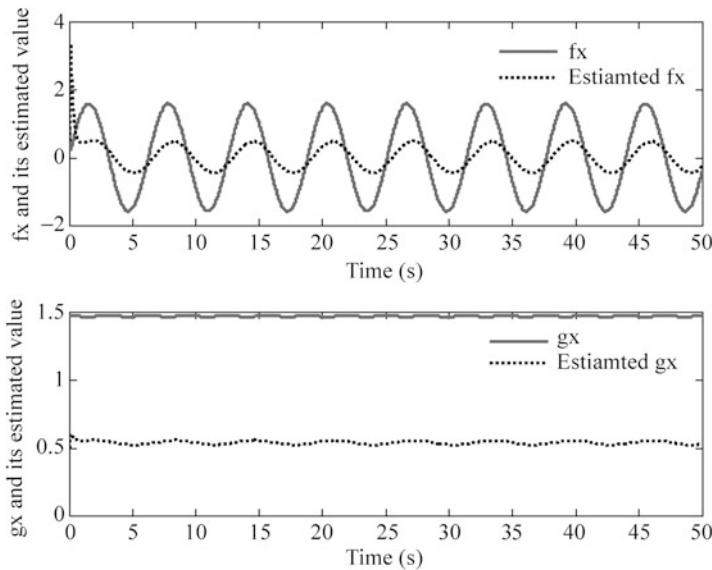


Fig. 8.6 $f(x)$, $g(x)$ and their estimation

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -\frac{1}{I}(MgL \sin x_1 + K(x_1 - x_3)) \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = \frac{1}{J}(u - K(x_3 - x_1)) \end{cases} \quad (8.22)$$

Also, the simplified system equation can be written as

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_3 + g(x) \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = f(x) + mu \end{cases} \quad (8.23)$$

where $x = [x_1 \ x_2 \ x_3 \ x_4]^T$ is the state vector, $g(x) = -x_3 - MgL \sin(x_1)/I - K(x_1 - x_3)/I$, $f(x) = K(x_1 - x_3)/J$, and $m = 1/J$.

Let $e_1 = x_1 - x_{1d}$, and $e_2 = \dot{x}_1 - \dot{x}_{1d}$. The control goal is x_1 track x_{1d} , \dot{x}_1 track \dot{x}_{1d} , that is, $e_1 \rightarrow 0$, and $e_2 \rightarrow 0$. If we design Lyapunov function as $V = \frac{1}{2}e_1^2 + \frac{1}{2}e_2^2$, then we have $\dot{V} = e_1\dot{e}_1 + e_2\dot{e}_2$, and since the control input u does not appear in \dot{V} expression, the control input cannot be designed.

8.4.2 Backstepping Controller Design

Reference to the main idea of adaptive backstepping sliding controller design for single-link flexible joint robot proposed in [16], we discuss backstepping controller design for single-link flexible joint robot in several steps as follows:

Step 1 Define $e_1 = x_1 - x_{1d}$, and x_{1d} is the ideal position signal, and then

$$\dot{e}_1 = \dot{x}_1 - \dot{x}_{1d} = x_2 - \dot{x}_{1d}$$

To realize $e_1 \rightarrow 0$, define a Lyapunov function as

$$V_1 = \frac{1}{2}e_1^2 \quad (8.24)$$

Then

$$\dot{V}_1 = e_1\dot{e}_1 = e_1(x_2 - \dot{x}_{1d})$$

To realize $\dot{V}_1 < 0$, if we choose $x_2 - \dot{x}_{1d} = -k_1e_1$, $k_1 > 0$, then we would have $\dot{V}_1 = -k_1e_1^2$

Step 2 To realize $x_2 - \dot{x}_{1d} = -k_1e_1$, that is, $x_2 = \dot{x}_{1d} - k_1e_1$, we choose virtual control as

$$x_{2d} = \dot{x}_{1d} - k_1e_1 \quad (8.25)$$

To realize $x_2 \rightarrow x_{2d}$, we get a new error

$$e_2 = x_2 - x_{2d}$$

Then $\dot{e}_2 = \dot{x}_2 - \dot{x}_{2d} = x_3 + g(x) - \dot{x}_{2d}$, and

$$\begin{aligned}\dot{V}_1 &= e_1(x_2 - \dot{x}_{1d}) = e_1(x_{2d} + e_2 - \dot{x}_{1d}) = e_1(\dot{x}_{1d} - k_1 e_1 + e_2 - \dot{x}_{1d}) \\ &= -k_1 e_1^2 + e_1 e_2\end{aligned}$$

To realize $e_2 \rightarrow 0$ and $e_1 \rightarrow 0$, design Lyapunov function as

$$V_2 = V_1 + \frac{1}{2} e_2^2 = \frac{1}{2} (e_1^2 + e_2^2) \quad (8.26)$$

Then

$$\dot{V}_2 = -k_1 e_1^2 + e_1 e_2 + e_2(x_3 + g(x) - \dot{x}_{2d})$$

To realize $\dot{V}_2 < 0$, if we choose

$$e_1 + x_3 + g(x) - \dot{x}_{2d} = -k_2 e_2, \quad k_2 > 0$$

then we would have $\dot{V}_2 = -k_1 e_1^2 - k_2 e_2^2$

Step 3 To realize $e_1 + x_3 + g(x) - \dot{x}_{2d} = -k_2 e_2$, that is, $x_3 = \dot{x}_{2d} - g(x) - k_2 e_2 - e_1$, we choose virtual control as

$$x_{3d} = \dot{x}_{2d} - g(x) - k_2 e_2 - e_1 \quad (8.27)$$

To realize $x_3 \rightarrow x_{3d}$, we get a new error

$$e_3 = x_3 - x_{3d}$$

Then $\dot{e}_3 = \dot{x}_3 - \dot{x}_{3d} = x_4 - \dot{x}_{3d}$

To realize $e_3 \rightarrow 0$ and $e_1 \rightarrow 0$ and $e_2 \rightarrow 0$, design Lyapunov function as

$$V_3 = V_2 + \frac{1}{2} e_3^2 = \frac{1}{2} (e_1^2 + e_2^2 + e_3^2) \quad (8.28)$$

Then

$$\begin{aligned}\dot{V}_3 &= -k_1 e_1^2 + e_1 e_2 + e_2(x_3 + g(x) - \dot{x}_{2d}) + e_3 \dot{e}_3 \\ &= -k_1 e_1^2 + e_1 e_2 + e_2(e_3 + x_{3d} + g(x) - \dot{x}_{2d}) + e_3 \dot{e}_3 \\ &= -k_1 e_1^2 - k_2 e_2^2 + e_2 e_3 + e_3(x_4 - \dot{x}_{3d}) \\ &= -k_1 e_1^2 - k_2 e_2^2 + e_3(e_2 + x_4 - \dot{x}_{3d})\end{aligned}$$

To realize $\dot{V}_3 < 0$, we choose

$$e_2 + x_4 - \dot{x}_{3d} = -k_3 e_3, \quad k_3 > 0$$

Then we would have

$$\dot{V}_3 = -k_1 e_1^2 - k_2 e_2^2 - k_3 e_3^2$$

Step 4 To realize $e_2 + x_4 - \dot{x}_{3d} = -k_3 e_3$, that is, $x_4 = \dot{x}_{3d} - k_3 e_3 - e_2$, we choose virtual control as

$$x_{4d} = \dot{x}_{3d} - k_3 e_3 - e_2 \quad (8.29)$$

To realize $x_4 \rightarrow x_{4d}$, we get a new error

$$e_4 = x_4 - x_{4d}$$

Then, $\dot{e}_4 = \dot{x}_4 - \dot{x}_{4d} = f(x) + mu - \dot{x}_{4d}$

To realize $e_4 \rightarrow 0$ and $e_1 \rightarrow 0$, $e_2 \rightarrow 0$, and $e_3 \rightarrow 0$, design Lyapunov function as

$$V_4 = V_3 + \frac{1}{2} e_4^2 = \frac{1}{2} (e_1^2 + e_2^2 + e_3^2 + e_4^2) \quad (8.30)$$

Then

$$\dot{V}_4 = -k_1 e_1^2 - k_2 e_2^2 + e_3 (e_2 + x_4 - \dot{x}_{3d}) + e_4 (f(x) + mu - \dot{x}_{4d})$$

Since

$$\begin{aligned} e_3 (e_2 + x_4 - \dot{x}_{3d}) &= e_3 (e_2 + x_{4d} + e_4 - \dot{x}_{3d}) \\ &= e_3 (e_2 + \dot{x}_{3d} - k_3 e_3 - e_2 + e_4 - \dot{x}_{3d}) \\ &= e_3 e_4 - k_3 e_3^2 \end{aligned}$$

then

$$\begin{aligned} \dot{V}_4 &= -k_1 e_1^2 - k_2 e_2^2 + e_3 e_4 - k_3 e_3^2 + e_4 (f(x) + mu - \dot{x}_{4d}) \\ &= -k_1 e_1^2 - k_2 e_2^2 - k_3 e_3^2 + e_4 (e_3 + f(x) + mu - \dot{x}_{4d}) \end{aligned}$$

To realize $\dot{V}_4 < 0$, we choose

$$e_3 + f(x) + mu - \dot{x}_{4d} = -k_4 e_4, \quad k_4 > 0 \quad (8.31)$$

Then

$$\dot{V}_4 = -k_1 e_1^2 - k_2 e_2^2 - k_3 e_3^2 - k_4 e_4^2$$

From (8.31), we can get the control law

$$u = \frac{1}{m}(-f(x) + \dot{x}_{4d} - k_4 e_4 - e_3) \quad (8.32)$$

To realize the control law (8.32), modeling information $g(x)$, $f(x)$, and m are needed, which are difficult in practical engineering. We can use RBF to approximate them.

8.5 Adaptive Backstepping Control with RBF for Single-Link Flexible Joint Robot

For the system (8.23) in Sect. 8.4, we consider $g(x)$, $f(x)$, and m are unknown, the lower bound of m is known, $m \geq \bar{m}$, and $\bar{m} > 0$

The unknown functions of \hat{f} , \hat{g} , and \hat{d} can be approximated by neural network, and \hat{m} can be estimated by adaptive law. Figure 8.7 shows the closed-loop neural-based adaptive control scheme.

8.5.1 Backstepping Controller Design with Function Estimation

Reference to the main idea of adaptive neural network backstepping sliding controller design for single-link flexible joint robot proposed in [16], we discuss RBF-based backstepping controller design for single-link flexible joint robot in several steps as follows:

Step 1 Define $e_1 = x_1 - x_{1d}$, and x_{1d} is the ideal position signal, and then

$$\dot{e}_1 = \dot{x}_1 - \dot{x}_{1d} = x_2 - \dot{x}_{1d} \quad (8.33)$$

To realize $e_1 \rightarrow 0$, define a Lyapunov function as

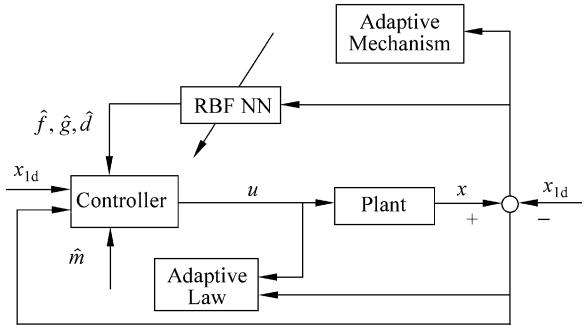
$$V_1 = \frac{1}{2} e_1^2 \quad (8.34)$$

Then

$$\dot{V}_1 = e_1 \dot{e}_1 = e_1 (x_2 - \dot{x}_{1d})$$

To realize $\dot{V}_1 < 0$, if we choose $x_2 - \dot{x}_{1d} = -k_1 e_1$, $k_1 > 0$, then we would have $\dot{V}_1 = -k_1 e_1^2$.

Fig. 8.7 Block diagram of the control scheme



Step 2 To realize $x_2 - \dot{x}_{1d} = -k_1 e_1$, that is, $x_2 = \dot{x}_{1d} - k_1 e_1$, we choose virtual control as

$$x_{2d} = \dot{x}_{1d} - k_1 e_1 \quad (8.35)$$

To realize $x_2 \rightarrow x_{2d}$, we get a new error

$$e_2 = x_2 - x_{2d}$$

Then $\dot{e}_2 = \dot{x}_2 - \dot{x}_{2d} = x_3 + g(x) - \dot{x}_{2d}$, and

$$\begin{aligned} \dot{V}_1 &= e_1(x_2 - \dot{x}_{1d}) = e_1(x_{2d} + e_2 - \dot{x}_{1d}) = e_1(\dot{x}_{1d} - k_1 e_1 + e_2 - \dot{x}_{1d}) \\ &= -k_1 e_1^2 + e_1 e_2 \end{aligned}$$

To realize $e_2 \rightarrow 0$ and $e_1 \rightarrow 0$, design Lyapunov function as

$$V_2 = V_1 + \frac{1}{2} e_2^2 = \frac{1}{2} (e_1^2 + e_2^2) \quad (8.36)$$

Then

$$\dot{V}_2 = -k_1 e_1^2 + e_1 e_2 + e_2(x_3 + g(x) - \dot{x}_{2d})$$

To realize $\dot{V}_2 < 0$, if we choose

$$e_1 + x_3 + g(x) - \dot{x}_{2d} = -k_2 e_2, \quad k_2 > 0$$

then we would have $\dot{V}_2 = -k_1 e_1^2 - k_2 e_2^2$

Step 3 To realize $e_1 + x_3 + g(x) - \dot{x}_{2d} = -k_2 e_2$, that is, $x_3 = \dot{x}_{2d} - g(x) - k_2 e_2 - e_1$, we choose virtual control as

$$x_{3d} = \dot{x}_{2d} - \hat{g}(x) - k_2 e_2 - e_1 \quad (8.37)$$

To realize $x_3 \rightarrow x_{3d}$, we get a new error

$$e_3 = x_3 - x_{3d}$$

Then $\dot{e}_3 = \dot{x}_3 - \dot{x}_{3d} = x_4 - \dot{x}_{3d}$.

To realize $e_3 \rightarrow 0$ and $e_1 \rightarrow 0$ and $e_2 \rightarrow 0$, design Lyapunov function as

$$V_3 = V_2 + \frac{1}{2}e_3^2 = \frac{1}{2}(e_1^2 + e_2^2 + e_3^2) \quad (8.38)$$

Then

$$\begin{aligned} \dot{V}_3 &= -k_1e_1^2 + e_1e_2 + e_2(x_3 + g(x) - \dot{x}_{2d}) + e_3\dot{e}_3 \\ &= -k_1e_1^2 + e_1e_2 + e_2(e_3 + x_{3d} + g(x) - \dot{x}_{2d}) + e_3\dot{e}_3 \\ &= -k_1e_1^2 - k_2e_2^2 + e_2e_3 + e_3(x_4 - \dot{x}_{3d}) + e_2(g(x) - \hat{g}(x)) \\ &= -k_1e_1^2 - k_2e_2^2 + e_3(e_2 + x_4 - \dot{x}_{3d}) + e_2(g(x) - \hat{g}(x)) \end{aligned}$$

To realize $\dot{V}_3 < 0$, we choose

$$e_2 + x_4 - \dot{x}_{3d} = -k_3e_3, \quad k_3 > 0$$

Then, we would have $\dot{V}_3 = -k_1e_1^2 - k_2e_2^2 - k_3e_3^2 + e_2(g(x) - \hat{g}(x))$.

Step 4 Design control law

To realize $e_2 + x_4 - \dot{x}_{3d} = -k_3e_3$, that is, $x_4 = \dot{x}_{3d} - k_3e_3 - e_2$, we choose virtual control as

$$x_{4d} = \hat{\dot{x}}_{3d} - k_3e_3 - e_2 \quad (8.39)$$

where $\dot{x}_{3d} = \dot{x}_{3d1} - d$, $\hat{\dot{x}}_{3d} = \dot{x}_{3d1} - \hat{d}$, and

$$\begin{aligned} \dot{x}_{3d} &= -\dot{\hat{g}} + \ddot{x}_{2d} - k_2\dot{e}_2 - \dot{e}_1 \\ &= -\dot{\hat{g}} + \ddot{x}_{1d} - k_1\dot{e}_1 - k_2(x_3 + g - \dot{x}_{2d}) - (x_2 - \dot{x}_{1d}) \\ &= -\dot{\hat{g}} + \ddot{x}_{1d} - k_1(x_3 + g - \dot{x}_{1d}) - k_2(x_3 + g - \dot{x}_{2d}) - x_2 + \dot{x}_{1d} \\ &= \dot{x}_{3d1} - d \end{aligned}$$

where $\dot{x}_{3d} = \dot{x}_{3d1} - \dot{x}_{3d2}$, $\dot{x}_{3d1} = \ddot{x}_{1d} - k_1(x_3 - \dot{x}_{1d}) - k_2(x_3 - \dot{x}_{2d}) + \dot{x}_{1d} - x_2$ and $\dot{x}_{3d2} = \dot{\hat{g}} + k_1g + k_2g$, \dot{x}_{3d1} is composed of known values, and \dot{x}_{3d2} is the unknown part of \dot{x}_{3d} , and let $\dot{x}_{3d2} = d$.

To realize $x_4 \rightarrow x_{4d}$, we get a new error

$$e_4 = x_4 - x_{4d}$$

Then

$$\dot{e}_4 = \dot{x}_4 - \dot{x}_{4d} = \bar{f} + mu - \dot{x}_{4d1} - \dot{x}_{4d2}$$

where

$$\begin{aligned}\dot{x}_{4d} &= \ddot{x}_{3d1} - \dot{\hat{d}} - k_3\dot{e}_3 - \dot{e}_2 \\ &= \ddot{x}_{1d} - k_1(\dot{x}_3 - \ddot{x}_{1d}) - k_2(\dot{x}_3 - \ddot{x}_{2d}) + \ddot{x}_{1d} - \dot{x}_2 - \dot{\hat{d}} - k_3(x_4 - \dot{x}_{3d1} + \dot{x}_{3d2}) - (\dot{x}_2 - \dot{x}_{2d}) \\ &= \ddot{x}_{1d} - k_1(x_4 - \ddot{x}_{1d}) - k_2(x_4 - \ddot{x}_{1d} + k_1(\dot{x}_2 - \ddot{x}_{1d})) + \ddot{x}_{1d} - x_3 - g - \dot{\hat{d}} \\ &\quad - k_3(x_4 - \dot{x}_{3d1} + \dot{x}_{3d2}) - (x_3 + g - \dot{x}_{2d}) \\ &= \ddot{x}_{1d} - k_1(x_4 - \ddot{x}_{1d}) - k_2(x_4 - \ddot{x}_{1d} + k_1(x_3 - \ddot{x}_{1d})) + \ddot{x}_{1d} - x_3 - k_3(x_4 - \dot{x}_{3d1}) \\ &\quad - (x_3 - \dot{x}_{2d}) - k_1k_2g - g - \dot{\hat{d}} - k_3\dot{x}_{3d2} - g \\ &= \dot{x}_{4d1} + \dot{x}_{4d2}\end{aligned}$$

where \dot{x}_{4d} is composed of known values \dot{x}_{4d1} and the unknown part of \dot{x}_{4d2}

$$\begin{cases} \dot{x}_{4d1} = \ddot{x}_{1d} - k_1(x_4 - \ddot{x}_{1d}) - k_2(x_4 - \ddot{x}_{1d} + k_1(x_3 - \ddot{x}_{1d})) \\ \quad + \ddot{x}_{1d} - x_3 - k_3(x_4 - \dot{x}_{3d1}) - (x_3 - \dot{x}_{2d}) \\ \dot{x}_{4d2} = -k_1k_2g - g - \dot{\hat{d}} - k_3\dot{x}_{3d2} - g \end{cases} \quad (8.40)$$

Define $\bar{f} = f - \dot{x}_{4d2}$, and then

$$\dot{e}_4 = \bar{f} - \dot{x}_{4d1} + mu = \bar{f} - \dot{x}_{4d1} + (m - \hat{m})u + \hat{m}u$$

where \hat{m} is estimation value of m .

To realize $e_4 \rightarrow 0$ and $e_1 \rightarrow 0$, $e_2 \rightarrow 0$, and $e_3 \rightarrow 0$, design Lyapunov function as

$$V_4 = V_3 + \frac{1}{2}e_4^2 = \frac{1}{2}(e_1^2 + e_2^2 + e_3^2 + e_4^2) \quad (8.41)$$

Then

$$\begin{aligned}\dot{V}_4 &= -k_1e_1^2 - k_2e_2^2 + e_3(e_2 + x_4 - \dot{x}_{3d}) + e_2(g(x) - \hat{g}(x)) \\ &\quad + e_4(\bar{f} - \dot{x}_{4d1} + (m - \hat{m})u + \hat{m}u)\end{aligned}$$

Since

$$\begin{aligned}e_3(e_2 + x_4 - \dot{x}_{3d}) &= e_3(e_2 + x_{4d} + e_4 - \dot{x}_{3d}) \\ &= e_3(e_2 + \dot{x}_{3d1} - \dot{\hat{d}} - k_3e_3 - e_2 + e_4 - \dot{x}_{3d1} + d) \\ &= e_3e_4 - k_3e_3^2 + e_3(d - \dot{\hat{d}})\end{aligned}$$

then

$$\begin{aligned}\dot{V}_4 &= -k_1e_1^2 - k_2e_2^2 + e_3e_4 - k_3e_3^2 + e_3(d - \hat{d}) + e_2(g - \hat{g}) + e_4(\bar{f} - \dot{x}_{4d1} + (m - \hat{m})u + \hat{m}u) \\ &= -k_1e_1^2 - k_2e_2^2 - k_3e_3^2 + e_2(g - \hat{g}) + e_3(d - \hat{d}) + e_4(m - \hat{m})u + e_4(e_3 + \bar{f} - \dot{x}_{4d1} + \hat{m}u)\end{aligned}$$

To realize $\dot{V}_4 < 0$, we choose the control law as

$$u = \frac{1}{\hat{m}}(-\hat{\bar{f}} + \dot{x}_{4d1} - k_4e_4 - e_3) \quad (8.42)$$

then

$$\begin{aligned}\dot{V}_4 &= -k_1e_1^2 - k_2e_2^2 - k_3e_3^2 - k_4e_4^2 + (m - \hat{m})ue_4 + (g - \hat{g})e_2 \\ &\quad + (d - \hat{d})e_3 + (\bar{f} - \hat{\bar{f}})e_4\end{aligned} \quad (8.43)$$

If $\hat{m} = m$, $\hat{g} = g$, $\hat{d} = d$, and $\hat{\bar{f}} = \bar{f}$, we can get $\dot{V}_4 < 0$

8.5.2 Backstepping Controller Design with RBF Approximation

We use three kind of RBF neural network to approximate g , d and \bar{f} respectively as follows

$$\begin{cases} g = \mathbf{W}_1^T \mathbf{h}_1 + \varepsilon_1 \\ d = \mathbf{W}_2^T \mathbf{h}_2 + \varepsilon_2 \\ \bar{f} = \mathbf{W}_3^T \mathbf{h}_3 + \varepsilon_3 \end{cases}$$

where W_i is the ideal neural network weight value, h_i is the Gaussian function, $i = 1, 2, 3$, ε_i is the approximation error and $\|\boldsymbol{\varepsilon}\| = \|[\varepsilon_1 \quad \varepsilon_2 \quad \varepsilon_3]^T\| < \varepsilon_N$, $\|\mathbf{W}\|_F \leq W_M$

Define

$$\begin{cases} \hat{g} = \hat{\mathbf{W}}_1^T \mathbf{h}_1 \\ \hat{d} = \hat{\mathbf{W}}_2^T \mathbf{h}_2 \\ \hat{\bar{f}} = \hat{\mathbf{W}}_3^T \mathbf{h}_3 \end{cases} \quad (8.44)$$

where $\hat{\mathbf{W}}_i^T$ is weight value estimation.

Define

$$\mathbf{Z} = \begin{bmatrix} 0 & \mathbf{W}_1 & & \\ & & \mathbf{W}_2 & \\ & & & \mathbf{W}_3 \end{bmatrix}, \quad \|\mathbf{Z}\|_F \leq Z_M$$

$$\hat{\mathbf{Z}} = \begin{bmatrix} 0 & \hat{\mathbf{W}}_1 & & \\ & & \hat{\mathbf{W}}_2 & \\ & & & \hat{\mathbf{W}}_3 \end{bmatrix}, \quad \tilde{\mathbf{Z}} = \mathbf{Z} - \hat{\mathbf{Z}}$$

Design Lyapunov function as

$$V = \frac{1}{2}\xi^T \xi + \frac{1}{2}\text{tr}(\tilde{\mathbf{Z}}^T \Gamma^{-1} \tilde{\mathbf{Z}}) + \frac{1}{2}\eta\tilde{m}^2 \quad (8.45)$$

where $V_4 = \frac{1}{2}\xi^T \xi$, $\eta > 0$, Γ is a positive-definite matrix with proper dimension,

$$\Gamma = \begin{bmatrix} 0 & & & \\ & \Gamma_2 & & \\ & & \Gamma_3 & \\ & & & \Gamma_4 \end{bmatrix}, \quad \xi = [e_1 \ e_2 \ e_3 \ e_4]^T, \text{ and } \tilde{m} = m - \hat{m}.$$

Let adaptive law as

$$\dot{\tilde{\mathbf{Z}}} = \Gamma \mathbf{h} \xi^T - n\Gamma \|\xi\| \hat{\mathbf{Z}} \quad (8.46)$$

where $\mathbf{h} = [0 \ h_1 \ h_2 \ h_3]^T$, n is a positive number, and $\hat{m}(0) \geq m$

From (8.43), (8.44), and (8.45), we have

$$\begin{aligned} \dot{V} &= \xi^T \dot{\xi} + \text{tr}(\tilde{\mathbf{Z}}^T \Gamma^{-1} \dot{\tilde{\mathbf{Z}}}) + \eta\tilde{m}\dot{\tilde{m}} \\ &= -k_1 e_1^2 - k_2 e_2^2 - k_3 e_3^2 - k_4 e_4^2 + (\tilde{\mathbf{W}}_1^T h_1 + \varepsilon_1) e_2 + (\tilde{\mathbf{W}}_2^T h_2 + \varepsilon_2) e_3 \\ &\quad + (\tilde{\mathbf{W}}_3^T h_3 + \varepsilon_3) e_4 + \text{tr}(\tilde{\mathbf{Z}}^T \Gamma^{-1} \dot{\tilde{\mathbf{Z}}}) + \tilde{m} e_4 u + \eta\tilde{m}\dot{\tilde{m}} \end{aligned}$$

where $\tilde{\mathbf{W}}_i^T = \mathbf{W}_i^T - \hat{\mathbf{W}}_i^T$, $i = 1, 2, 3$

Then

$$\begin{aligned} \dot{V} &= -\xi^T \mathbf{K}_e \xi + \xi^T \boldsymbol{\varepsilon} + \xi^T \tilde{\mathbf{Z}} \mathbf{h} + \text{tr}(\tilde{\mathbf{Z}}^T \Gamma^{-1} \dot{\tilde{\mathbf{Z}}}) + \tilde{m} e_4 u + \eta\tilde{m}\dot{\tilde{m}} \\ &= -\xi^T \mathbf{K}_e \xi + \xi^T \boldsymbol{\varepsilon} + \text{tr}(\tilde{\mathbf{Z}}^T \Gamma^{-1} \dot{\tilde{\mathbf{Z}}} + \tilde{\mathbf{Z}}^T \mathbf{h} \xi^T) + \tilde{m} e_4 u + \eta\tilde{m}\dot{\tilde{m}} \end{aligned}$$

where $\mathbf{K}_e = [k_1 \ k_2 \ k_3 \ k_4]^T$, and $\boldsymbol{\varepsilon} = [0 \ \varepsilon_1 \ \varepsilon_2 \ \varepsilon_3]^T$.

Since $\dot{\tilde{\mathbf{Z}}} = -\dot{\tilde{\mathbf{Z}}}$, and $\dot{\tilde{m}} = -\dot{m}$, submitting the adaptive law (8.46), we have

$$\dot{V} = -\xi^T \mathbf{K}_e \xi + \xi^T \mathbf{e} + n \|\xi\| \text{tr}(\tilde{\mathbf{Z}}^T (\mathbf{Z} - \tilde{\mathbf{Z}})) + \tilde{m}(e_4 u - \eta \dot{m}) \quad (8.47)$$

To guarantee $\tilde{m}(e_4 u - \eta \dot{m}) \leq 0$, at the same time to avoid singularity in (8.42) and guarantee $\hat{m} \geq \underline{m}$, we use an adaptive law for \hat{m} given in [16] as follows:

$$\dot{\hat{m}} = \begin{cases} \eta^{-1} e_4 u, & \text{if } e_4 u > 0 \\ \eta^{-1} e_4 u, & \text{if } e_4 u \leq 0 \text{ and } \hat{m} > m \\ \eta^{-1}, & \text{if } e_4 u \leq 0 \text{ and } \hat{m} \leq m \end{cases} \quad (8.48)$$

where $\hat{m}(0) \geq \underline{m}$.

The adaptive law (8.48) can be analyzed as:

1. If $e_4 u > 0$, we get $\tilde{m}(e_4 u - \eta \dot{m}) = 0$ and $\dot{\hat{m}} > 0$; thus, $\hat{m} > \underline{m}$
2. If $e_4 u \leq 0$ and $\hat{m} > m$, we get $\tilde{m}(e_4 u - \eta \dot{m}) = 0$
3. If $e_4 u \leq 0$ and $\hat{m} \leq \underline{m}$, we have $\tilde{m} = m - \hat{m} \geq m - \underline{m} > 0$; thus, $\tilde{m}(e_4 u - \eta \dot{m}) = \tilde{m}e_4 u - \tilde{m} \leq 0$, and \hat{m} will increase gradually, and then $\hat{m} > \underline{m}$ will be guaranteed with $\dot{\hat{m}} > 0$

According to Schwarz inequality, we have $\text{tr}(\tilde{\mathbf{Z}}^T (\mathbf{Z} - \tilde{\mathbf{Z}})) \leq \|\tilde{\mathbf{Z}}\|_{\text{F}} \|\mathbf{Z}\|_{\text{F}} - \|\tilde{\mathbf{Z}}\|_{\text{F}}^2$, since $K_{\min} \|\xi\|^2 \leq \xi^T \mathbf{K} \xi$, K_{\min} is the minimum eigenvalue of \mathbf{K} , (8.47) becomes

$$\begin{aligned} \dot{V} &\leq -K_{\min} \|\xi\|^2 + \varepsilon_N \|\xi\| + n \|\xi\| \left(\|\tilde{\mathbf{Z}}\|_{\text{F}} \|\mathbf{Z}\|_{\text{F}} - \|\tilde{\mathbf{Z}}\|_{\text{F}}^2 \right) + M \\ &\leq -\|\xi\| (K_{\min} \|\xi\| - \varepsilon_N + n \|\tilde{\mathbf{Z}}\|_{\text{F}} (\|\tilde{\mathbf{Z}}\|_{\text{F}} - Z_M)) \end{aligned}$$

Since

$$\begin{aligned} K_{\min} \|\xi\| - \varepsilon_N + n \left(\|\tilde{\mathbf{Z}}\|_{\text{F}}^2 - \|\tilde{\mathbf{Z}}\|_{\text{F}} Z_M \right) \\ = K_{\min} \|\xi\| - \varepsilon_N + n \left(\|\tilde{\mathbf{Z}}\|_{\text{F}} - \frac{1}{2} Z_M \right)^2 - \frac{n}{4} Z_M^2 \end{aligned}$$

this implies that $\dot{V} < 0$ as long as

$$\|\xi\| > \frac{\varepsilon_N + \frac{n}{4} Z_M^2}{K_{\min}} \quad \text{or} \quad \|\tilde{\mathbf{Z}}\|_{\text{F}} > \frac{1}{2} Z_M + \sqrt{\frac{Z_M^2}{4} + \frac{\varepsilon_N}{n}} \quad (8.49)$$

From the above expression, we can see that the tracking performance is related to the value of ε_N , n , and K_{\min}

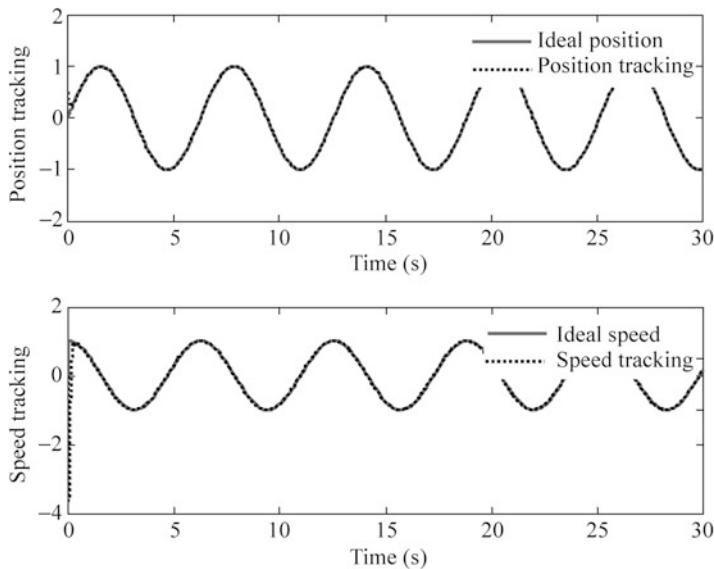


Fig. 8.8 Position and speed tracking

8.5.3 Simulation Examples

8.5.3.1 First Example

For the system (8.23), we choose $f(x) = 0$, $g(x) = 0$, and $m = 3.0$, and then (8.23) can be written as

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_3 \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = mu \end{cases}$$

We assume $f(x) = 0$ and $g(x) = 0$ are known and only m is unknown, and then Lyapunov function becomes $V = \frac{1}{2}\xi^T\xi + \frac{1}{2}\eta\hat{m}^2$; thus, only adaptive law (8.48) can be used.

The ideal position signal is $x_{1d} = \sin t$, and the initial value is $\mathbf{x}(0) = [0.5 \ 0 \ 0 \ 0]$. We use the control law (8.42) with adaptive law (8.48). The parameters are chosen as $k_1 = k_2 = k_3 = k_4 = 35$. In the adaptive law, we choose $\underline{m} = 1.0$, $\hat{m}(0) = 500$, and $\eta = 150$. The results are shown from Figs. 8.8, 8.9, and 8.10.

The Simulink program of this example is chap8_3sim.mdl and the Matlab programs of the example are given in the [Appendix](#).

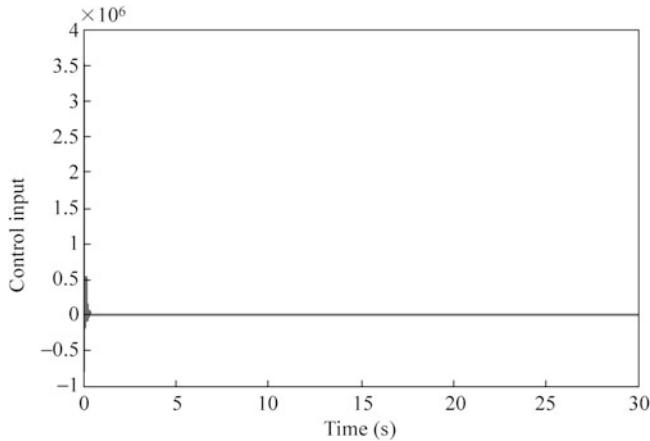


Fig. 8.9 Control input

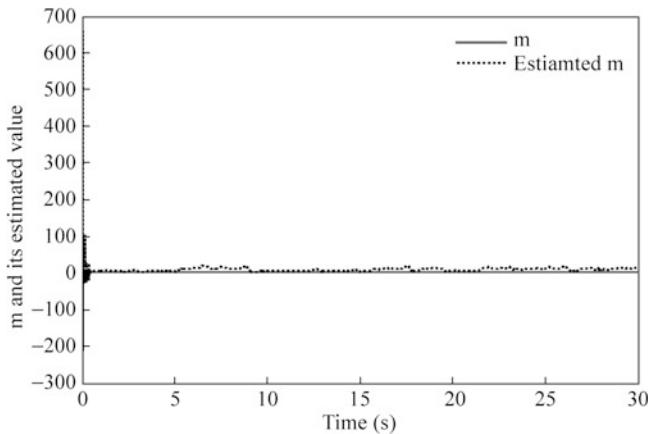


Fig. 8.10 Estimation of m

8.5.3.2 Second Example

Consider a dynamic equation of single-link flexible joint robot as

$$\begin{cases} I\ddot{q}_1 + MgL \sin q_1 + K(q_1 - q_2) = 0 \\ J\ddot{q}_2 + K(q_2 - q_1) = u \end{cases}$$

where $M = 0.2$ kg, $L = 0.02$ m, $I = 1.35 \times 10^{-3}$ kg · m 2 ; $K = 7.47$ Nm/rad, and $J = 2.16 \times 10^{-1}$ kg · m 2 .

The ideal position signal is $x_{1d} = \sin t$, and the initial value is $\mathbf{x}(0) = [0 \ 0 \ 0 \ 0]$. Control law (8.42) is used with adaptive law (8.46) and (8.48). The parameters are chosen as $k_1 = k_2 = k_3 = k_4 = 3.5$, $n = 0.01$, $\Gamma_2 = \Gamma_3 = \Gamma_4 = 250$, and $\boldsymbol{\Gamma} = \text{diag}\{0, \Gamma_2, \Gamma_3, \Gamma_4\}$. Since $J = 2.16 \times 10^{-1} \text{kg} \cdot \text{m}^2$, we choose $-m = 1.0$.

We use three RBFs to approximate g , d , and f , respectively. The structure is used as 4-5-1, and the input vector of RBF is $\mathbf{z} = [x_1 \ x_2 \ x_3 \ x_4]^T$. For each Gaussian function, the parameters of \mathbf{c}_i and b_i are designed as $[-1 \ -0.5 \ 0 \ 0.5 \ 1]$ and 1.5. The initial weight value is chosen as zero.

In the adaptive law (8.48), since there exists strong coupling between \hat{m} and u , the choice of initial value of \hat{m} is important. If $\hat{m}(0)$ is chosen as very small, u become very big, and then $\dot{\hat{m}}$ will become very big, which will cause big chattering of \hat{m} , and may cause \hat{m} to become zero, and control input will be singular. For the same reason, if $\hat{m}(0)$ is chosen as very big, u will become very small, and then $\dot{\hat{m}}$ will become very small, which will cause small change of \hat{m} , and may cause u failure. Therefore, in simulation, we should design initial value of \hat{m} as big as possible. In this simulation, we set $\hat{m}(0) = 500$. In addition, to guarantee not big of $\dot{\hat{m}}$ value, we choose big η , that is, $\eta = 150$.

The results are shown from Figs. 8.11, 8.12, and 8.13. The estimation of $\hat{g}(x)$ and \hat{m} do not converge to true value of $g(x)$ and m . This is due to the fact that the desired trajectory is not persistently exciting and the tracking error performance can be achieved by many possible values of $\hat{g}(x)$ and \hat{m} , besides the true $g(x)$ and m , which

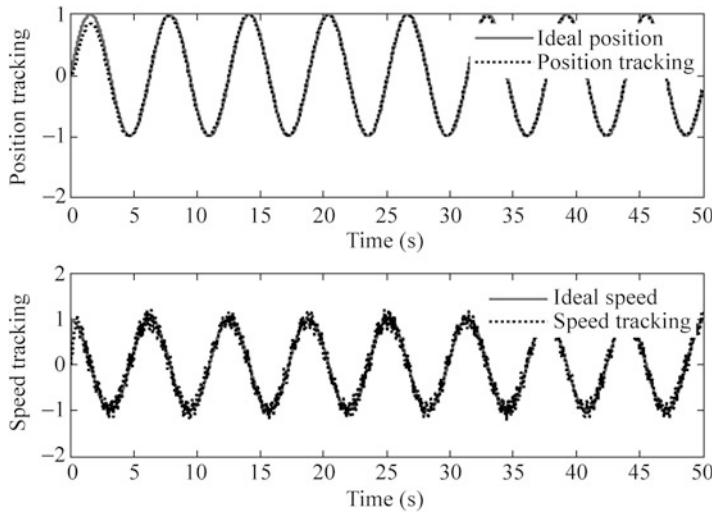


Fig. 8.11 Position and speed tracking

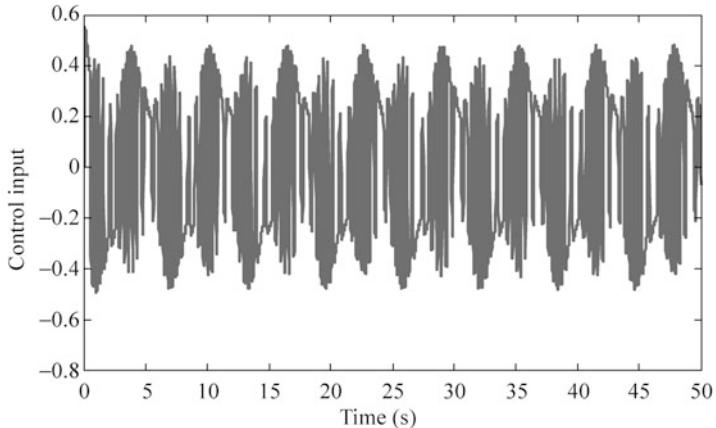


Fig. 8.12 Control input

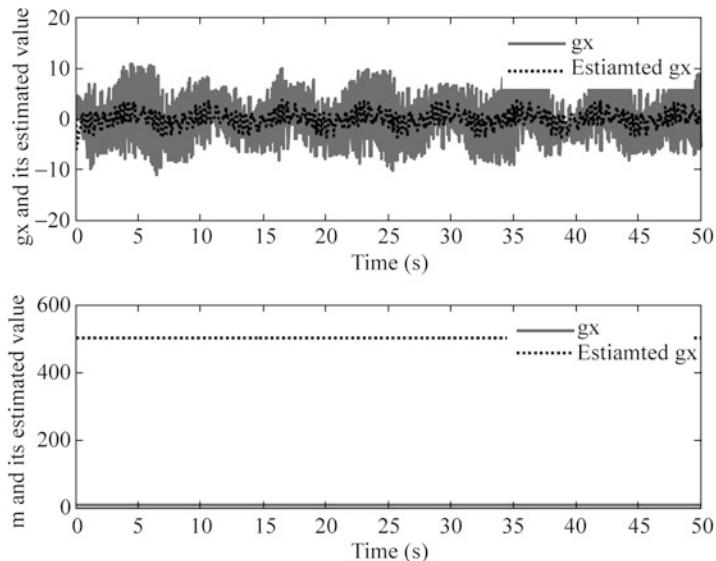


Fig. 8.13 Estimation of $g(x)$ and m

has been explained in Sect. 1.5, and this occurs quite often in real-world application.

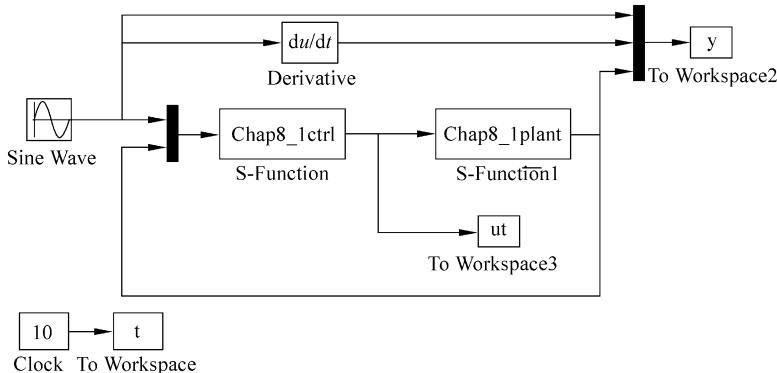
The Simulink program of this example is `chap8_4sim.mdl`, and the Matlab programs of the example are given in the [Appendix](#).

Appendix

Programs for Sect. 8.2.3

Simulation programs:

Simulink program: chap8_1sim.mdl



S function for 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 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 M V x0 fai

sizes = simsizes;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0=[];
str = [];

```

```

ts = [0 0];
function sys=mdlOutputs(t,x,u)
k1=35;
k2=15;

x1d=u(1);
dx1d=0.1*pi*cos(pi*t);
ddx1d=-0.1*pi^2*sin(pi*t);
x1=u(2);
x2=u(3);

g=9.8;mc=1.0;m=0.1;l=0.5;
S=1*(4/3-m*(cos(x1))^2/(mc+m));
fx=g*sin(x1)-m*l*x2^2*cos(x1)*sin(x1)/(mc+m);
fx=fx/S;
gx=cos(x1)/(mc+m);
gx=gx/S;

e1=x1-x1d;
de1=x2-dx1d;

x2d=dx1d-k1*e1;
dx2d=ddx1d-k1*de1;
e2=x2-x2d;
ut=(1/gx)*(-k2*e2+dx2d-e1-fx);

sys(1)=ut;

```

S function for 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 = 2;
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=1*(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)
sys(1)=x(1);
sys(2)=x(2);

```

Plot program: chap8_1plot.m

```

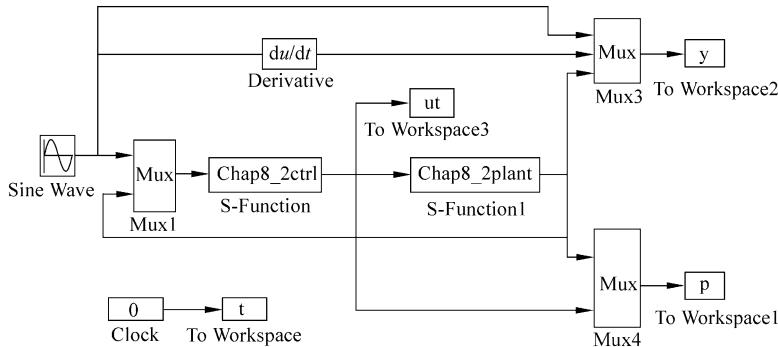
close all;
figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,3),'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('Ideal position','Position tracking');
subplot(212);
plot(t,y(:,2),'r',t,y(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('Ideal speed','Speed tracking');

figure(2);
plot(t,ut(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('Control input');

```

Programs for Sect. 8.3.4

Simulink program: chap8_2sim.mdl



S function for control law: chap8_2ctrl.m

```

function [sys,x0,str,ts] = MIMO_Tong_s(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 k1 k2 node
node=5;
sizes = simsizes;
sizes.NumContStates = 2*node;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 5;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [0.1*ones(2*node,1)];      %m(0)>ml
str = [];
ts = [];
c=0.5*[ -1 -0.5 0 0.5 1;
         -1 -0.5 0 0.5 1];
b=15;
k1=35;k2=35;
function sys=mdlDerivatives(t,x,u)

```

```

global c b k1 k2 node
x1d=u(1);
dx1d=0.1*cos(t);
ddx1d=-0.1*sin(t);

x1=u(2);x2=u(3);      %Plant states
z=[x1,x2]';
for j=1:1:node
    h(j)=exp(-norm(z-c(:,j))^2/(2*b^2));
end

e1=x1-x1d;
de1=x2-dx1d;
x2d=dx1d-k1*e1;
e2=x2-x2d;
dx2d=ddx1d-k1*de1;

Kexi=[e1 e2]';

n=0.1;
Gama1=500;Gama2=0.50;
Gama=[Gama1 0;
      0 Gama2];
%dZ=Gama*h*Kexi'-n*Gama*norm(Kexi)*Z;
w_fp=[x(1:node)]';      %fp weight
w_gp=[x(node+1:node*2)]';      %gp weight
for i=1:1:node
    sys(i)=Gama(1,1)*h(i)*Kexi(1)-n*Gama(1,1)*norm(Kexi)
        *w_fp(i); %f estimation
    sys(i+node)=Gama(2,2)*h(i)*Kexi(2)-n*Gama(2,2)*norm
        (Kexi)*w_gp(i); %g estimation
end
function sys=mdlOutputs(t,x,u)
global c b k1 k2 node
x1d=u(1);
dx1d=0.1*cos(t);
ddx1d=-0.1*sin(t);

x1=u(2);x2=u(3);
z=[x1,x2]';

for j=1:1:node
    h(j)=exp(-norm(z-c(:,j))^2/(2*b^2));
end

w_fp=[x(1:node)]'; %fp weight
w_gp=[x(node+1:node*2)]'; %gp weight
fp=w_fp*h';
gp=w_gp*h';

```

```
e1=x1-x1d;
de1=x2-dx1d;
x2d=dx1d-k1*e1;
e2=x2-x2d;
dx2d=ddx1d-k1*de1;

ut=1/(gp+0.01)*(-e1-fp+dx2d-k2*e2);

sys(1)=ut;
sys(2)=fp;
sys(3)=gp;
```

S function for 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 = 4;
sizes.NumInputs = 3;
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(1);
function sys=mdlOutputs(t,x,u)
g=9.8;mc=1.0;m=0.1;l=0.5;
S=1*(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;
sys(4)=gx;

```

Plot program: chap8_2plot.m

```

close all;

figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,3),'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('Ideal position','Position tracking');
subplot(212);
plot(t,y(:,2),'r',t,y(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('Ideal speed','Speed tracking');

figure(2);
plot(t,ut(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');

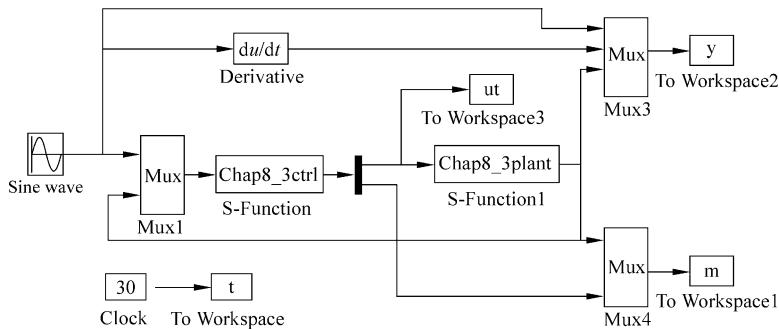
figure(3);
subplot(211);
plot(t,p(:,3),'r',t,p(:,6),'k:','linewidth',2);
xlabel('time(s)');ylabel('fx and its estimated value');
legend('fx','estiamted fx');
subplot(212);
plot(t,p(:,4),'r',t,p(:,7),'k:','linewidth',2);
xlabel('time(s)');ylabel('gx and its estimated value');
legend('gx','estiamted gx');

```

Programs for Sect. 8.5.3.1

Programs:

Simulink program: chap8_3sim.mdl



S function for control law and adaptive law: chap8_3ctrl.m

```

function [sys,x0,str,ts] = MIMO_Tong_s(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 k1 k2 k3 k4
sizes = simsizes;
sizes.NumContStates = 1;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 6;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [10]; %Let m(0)>>ml
str = [];
ts = [];
k1=35;k2=35;k3=35;k4=35;
function sys=mdlDerivatives(t,x,u)
global k1 k2 k3 k4
x1d=u(1);
dx1d=cos(t);
ddx1d=-sin(t);

```

```

dddx1d=-cos(t);
dddx1d=sin(t);
x1=u(2);x2=u(3);x3=u(4);x4=u(5); %Plant states
mp=x(1);

e1=x1-x1d;
x2d=dx1d-k1*e1;
e2=x2-x2d;
dx2d=ddx1d-k1*(x2-dx1d);

x3d=dx2d-k2*e2-e1;
dx3d1=dddx1d-k1*(x3-ddx1d)-k2*(x3-dx2d)+dx1d-x2;

e3=x3-x3d;
x4d=dx3d1-k3*e3-e2;
e4=x4-x4d;

dx4d1=dddx1d-k1*(x4-dddx1d)-k2*(x4-ddx1d+k1*(x3-
ddx1d))+ddx1d-x3-k3*(x4-dx3d1)-(x3-dx2d);

ut=(1/mp)*(dx4d1-k4*e4-e3);

eta=150;
m1=1;
if (e4*ut>0)
dm=(1/eta)*e4*ut;
end
if (e4*ut<=0)
if (mp>m1)
dm=(1/eta)*e4*ut;
else
dm=1/eta;
end
end
sys(1)=dm;
function sys=mdlOutputs(t,x,u)
global k1 k2 k3 k4
x1d=u(1);
dx1d=cos(t);
ddx1d=-sin(t);
dddx1d=-cos(t);
ddddx1d=sin(t);
x1=u(2);x2=u(3);x3=u(4);x4=u(5); %Plant states
mp=x(1);

e1=x1-x1d;
x2d=dx1d-k1*e1;
e2=x2-x2d;

```

```
dx2d=ddx1d-k1*(x2-dx1d);
x3d=dx2d-k2*e2-e1;
dx3d1=dddx1d-(k1)*(x3-ddx1d)-(k2)*(x3-dx2d)+dx1d-x2;
e3=x3-x3d;
x4d=dx3d1-k3*e3-e2;
e4=x4-x4d;

dx4d1=dddx1d-(k1)*(x4-ddx1d)-(k2)*(x4-ddx1d+(k1)*
(x3-ddx1d))-(k3)*(x4-dx3d1)-(x3-dx2d)+ddx1d-x3;
ut=(1/mp)*(dx4d1-k4*e4-e3);
sys(1)=ut;
sys(2)=mp;
S function for plant:chap8_3plant.m
function [sys,x0,str,ts]=MIMO_Tong_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 = 5;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.5 0 0 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
m=3.0;
ut=u(1);
sys(1)=x(2);
sys(2)=x(3);
sys(3)=x(4);
sys(4)=m*ut;
```

```

function sys=mdlOutputs(t,x,u)
m=3.0;

sys(1)=x(1);
sys(2)=x(2);
sys(3)=x(3);
sys(4)=x(4);
sys(5)=m;
Plot program:chap8_3plot.m
close all;

figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,3),'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('Ideal position','Position tracking');
subplot(212);
plot(t,y(:,2),'r',t,y(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('Ideal speed','Speed tracking');

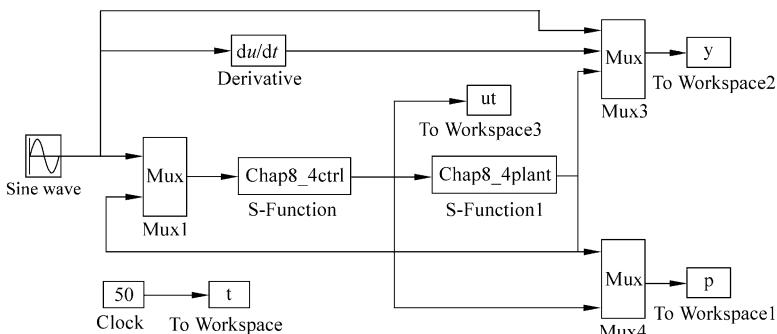
figure(2);
plot(t,ut(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');

figure(3);
plot(t,m(:,5),'r',t,m(:,6),'k:','linewidth',2);
xlabel('time(s)');ylabel('m and its estiamted value');
legend('m','estiamted m');

```

Programs for Sect. 8.5.3.2

Simulink program: chap8_4sim.mdl



S function for control law and adaptive law: chap8_4ctrl.m

```
function [sys,x0,str,ts] = MIMO_Tong_s(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 k1 k2 k3 k4 node
node=5;
sizes = simsizes;
sizes.NumContStates = 3*node+1;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 3;
sizes.NumInputs = 7;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [zeros(3*node,1);500];      %m(0)>ml
str = [];
ts = [];
c=[-1 -0.5 0 0.5 1;
   -1 -0.5 0 0.5 1;
   -1 -0.5 0 0.5 1;
   -1 -0.5 0 0.5 1];
b=1.5;
k1=0.35;k2=0.35;k3=0.35;k4=0.35;
function sys=mdlDerivatives(t,x,u)
global c b k1 k2 k3 k4 node
x1d=u(1);
dx1d=cos(t);
ddx1d=-sin(t);
dddx1d=-cos(t);
ddddx1d=sin(t);
x1=u(2);x2=u(3);x3=u(4);x4=u(5); %Plant states
z=[x1,x2,x3,x4]';
for j=1:1:node
```

```

h(j)=exp(-norm(z-c(:,j))^2/(2*b^2));
end
th_gp=[x(1:node)]';
%gp weight value
th_dp=[x(node+1:node*2)]';
%dp weight value
th_fp=[x(node*2+1:node*3)]';
%fp weight value
gp=th_gp*h';
dp=th_dp*h';
fp=th_fp*h';
mp=x(3*node+1);

e1=x1-x1d;
x2d=dx1d-k1*e1;
e2=x2-x2d;
dx2d=ddx1d-k1*(x2-dx1d);
x3d=-gp+dx2d-k2*e2-e1;

%D3=dddx1d-k1*(x3-ddx1d)-k2*(x3-dx2d)+dx1d-x2;
dx3d1=dddx1d-k1*(x3-ddx1d)-k2*(x3-dx2d)+dx1d-x2;

e3=x3-x3d;
x4d=dx3d1-dp-k3*e3-e2;
e4=x4-x4d;

%D4=ddddx1d-k1*(x4-dddx1d)-k2*(x4-dddx1d+k1*(x3-
ddx1d))+ddx1d-x3-k3*(x4-D3)-(x3-dx2d);
dx4d1=ddddx1d-k1*(x4-dddx1d)-k2*(x4-dddx1d+k1*(x3-
ddx1d))+ddx1d-x3-k3*(x4-dx3d1)-(x3-dx2d);

ut=(1/mp)*(-fp+dx4d1-k4*e4-e3);
Kexi=[e1 e2 e3 e4]';

n=0.01;
Gama2=250; Gama3=250; Gama4=250;
Gama=[ 0 0 0;
        0 Gama2 0;
        0 0 Gama3 0;
        0 0 0 Gama4];
eta=150;
m1=1;
if (e4*ut>0)
    dm=(1/eta)*e4*ut;
end
if (e4*ut<=0)
    if (mp>m1)
        dm=(1/eta)*e4*ut;
    else
        dm=1/eta;
    end
end

```

```
    end
end
for i=1:1:node
    sys(i)=Gama(2,2)*h(i)*Kexi(2)-n*Gama(2,2)*norm
        (Kexi)*th_gp(i); %g estimation
    sys(i+node)=Gama(3,3)*h(i)*Kexi(3)-n*Gama(3,3)
        *norm(Kexi)*th_dp(i); %d estimation
    sys(i+node*2)=Gama(4,4)*h(i)*Kexi(4)-n*Gama(4,4)
        *norm(Kexi)*th_fp(i); %f estimation
end
sys(3*node+1)=dm;
function sys=mdlOutputs(t,x,u)
global c b k1 k2 k3 k4 node
x1d=u(1);
dx1d=cos(t);
ddx1d=-sin(t);
dddx1d=-cos(t);
ddddx1d=sin(t);

x1=u(2);x2=u(3);x3=u(4);x4=u(5);
z=[x1,x2,x3,x4]';
for j=1:1:node
    h(j)=exp(-norm(z-c(:,j))^2/(2*b^2));
end

th_gp=[x(1:node) ]';           %gp weight
th_dp=[x(node+1:node*2) ]';   %dp weight
th_fp=[x(node*2+1:node*3) ]'; %fp weight
gp=th_gp*h';
dp=th_dp*h';
fp=th_fp*h';
mp=x(node*3+1);

e1=x1-x1d;
x2d=dx1d-k1*e1;
e2=x2-x2d;
dx2d=ddx1d-k1*(x2-dx1d);
x3d=-gp+dx2d-k2*e2-e1;
%D3=dddx1d-(k1)*(x3-ddx1d)-(k2)*(x3-dx2d)+dx1d-x2;
dx3d1=dddx1d-(k1)*(x3-ddx1d)-(k2)*(x3-dx2d)+dx1d-x2;

e3=x3-x3d;
x4d=dx3d1-dp-k3*e3-e2;
e4=x4-x4d;
```

```
%D4=ddddx1d-(k1)*(x4-dddx1d)-(k2)*(x4-dddx1d+(k1)*
    (x3-ddx1d))-(k3)*(x4-D3)-(x3-dx2d)+ddx1d-x3;
dx4d1=ddddx1d-(k1)*(x4-dddx1d)-(k2)*(x4-dddx1d+(k1)*
    (x3-ddx1d))-(k3)*(x4-dx3d1)-(x3-dx2d)+ddx1d-x3;
ut=(1/mp)*(-fp+dx4d1-k4*e4-e3);
sys(1)=ut;
sys(2)=gp;
sys(3)=mp;
```

S function for plant: chap8_4plant.m

```
function [sys,x0,str,ts]=MIMO_Tong_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 = 3;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0 0 0 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
M=0.2;L=0.02;I=1.35*0.001;K=7.47;J=2.16*0.1;
g=9.8;
gx=-x(3)-M*g*L*sin(x(1))/I-(K/I)*(x(1)-x(3));
fx=(K/J)*(x(1)-x(3));
m=1/J;
ut=u(1);
sys(1)=x(2);
sys(2)=x(3)+gx;
```

```
sys(3)=x(4);
sys(4)=fx+m*ut;
function sys=mdlOutputs(t,x,u)
M=0.2;L=0.02;I=1.35*0.001;K=7.47;J=2.16*0.1;
g=9.8;
gx=-x(3)-M*g*L*sin(x(1))/I-(K/I)*(x(1)-x(3));
m=1/J;

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

Plot program: chap8_4plot.m

```
close all;

figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,3),'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('Ideal position','Position tracking');
subplot(212);
plot(t,y(:,2),'r',t,y(:,4),'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
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(:,5),'r',t,p(:,8),'k:','linewidth',2);
xlabel('time(s)');ylabel('gx and its estimated value');
legend('gx','estiamted gx');
subplot(212);
plot(t,p(:,6),'r',t,p(:,9),'k:','linewidth',2);
xlabel('time(s)');ylabel('m and its estimated value');
legend('m','estiamted m');
```

References

1. Kanellakopoulos I, Kokotovic PV, Morse AS (1991) Systematic design of adaptive controllers for feedback linearizable systems. IEEE Trans Autom Control 36(11):1241–1253

2. Kokotovic PV (1992) The joy of feedback: nonlinear and adaptive. *IEEE Control Syst Mag* 12 (3):7–17
3. Krstic M, Kanellakopoulos I, Kokotovic P (1995) Nonlinear and adaptive control design. Wiley, New York
4. Zhang Y, Peng PY, Jiang ZP (2000) Stable neural controller design for unknown nonlinear systems using backstepping. *IEEE Trans Neural Netw* 11(6):1347–1360
5. Kwan CM, Lewis FL (2000) Robust backstepping control of nonlinear systems using neural networks. *IEEE Trans Syst Man Cybern (Part A)* 30(6):753–766
6. Ognjen K, Nitin S, Lewis FL, Chiman MK (2003) Design and implementation of industrial neural network controller using backstepping. *IEEE Trans Ind Electron* 50(1):193–201
7. Chen FC, Khalil HK (1992) Adaptive control of nonlinear systems using neural networks. *Int J Control* 55(6):1299–1317
8. Narendra KS (1991) Adaptive control using neural networks. In: Neural networks for control. MIT Press, Cambridge, MA, pp 115–142
9. Ozaki T, Suzuki T, Furuhashi T, Okuma S, Ushikawa Y (1991) Trajectory control of robotic manipulators. *IEEE Trans Ind Electron* 38(3):195–202
10. Ge SS, Wang C (2002) Direct adaptive NN control of a class of nonlinear systems. *IEEE Trans Neural Netw* 13(1):214–221
11. Kwan CM, Lewis FL (2000) Robust backstepping control of induction motors using neural networks. *IEEE Trans Neural Netw* 11(5):1178–1187
12. Choi JY, Farrell JA (2001) Adaptive observer backstepping control using neural networks. *IEEE Trans Neural Netw* 12(5):1103–1112
13. Zhang T, Ge SS, Hang CC (2000) Adaptive neural network control for strict-feedback nonlinear systems using backstepping design. *Automatica* 36(12):1835–1846
14. Ge SS, Wang C (2004) Adaptive neural control of uncertain MIMO nonlinear systems. *IEEE Trans Neural Netw* 15(3):674–692
15. Ge SS, Wang C (2002) Adaptive NN control of uncertain nonlinear pure-feedback systems. *Automatica* 38(4):671–682
16. Huang AC, Chen YC (2004) Adaptive sliding control for single-link flexible joint robot with mismatched uncertainties. *IEEE Trans Control Syst Technol* 12(5):770–775

Chapter 9

Digital RBF Neural Network Control

Abstract This chapter introduces adaptive Runge–Kutta–Merson method for digital RBF neural network controller design. Two examples for mechanical controls are given, including digital adaptive control for a servo system and digital adaptive control for two-link manipulators.

Keywords Neural network control • Digital control • Adaptive Runge–Kutta–Merson

9.1 Adaptive Runge–Kutta–Merson Method

9.1.1 Introduction

By simulation, we mean to simulate the actual situation as close as possible. In actual implementation of feedback control using digital computer, zero-order holders are usually used which keep the input constant during the sampling interval T_s , that is, $u(t) = u(nT_s)$ for $nT_s \leq t \leq (n+1)T_s$. It is clear that, in solving the dynamic response of the system numerically, the control inputs should be kept constant during the sampling interval and only be updated at the fixed intervals of T_s .

Thus, we shall discuss the numerical solution of

$$\dot{x}(t) = f(x(t)), \quad \forall x(0) = x_0 \quad (9.1)$$

at the discrete time instants

$$0, T_s, 2T_s, \dots, nT_s.$$

The algorithm will then generate the values as x_1, x_2, x_3, \dots , to approximate $x_1(t_1), x_2(t_2), x_3(t_3), \dots$

To solve the differential equations numerically, Runge–Kutta method is commonly used. One of the most popular is the fourth-order method given as follows: given an initial value problem (9.1), the values of x at time $t = t_0 + (n + 2)T$ can be computed from the values of x at time $t = t_0 + nT$ using the following formulae:

$$x_{n+1} = x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \quad (9.2)$$

where

$$\begin{aligned} k_1 &= Tf(t_n, x_n) \\ k_2 &= Tf\left(t_n + \frac{1}{2}T, x_n + \frac{1}{2}k_1\right) \\ k_3 &= Tf\left(t_n + \frac{1}{2}T, x_n + \frac{1}{2}k_2\right) \\ k_4 &= Tf(t_n + T, x_n + k_3) \end{aligned}$$

and T is the size of each step.

The fourth-order Runge–Kutta method is explicit and requires four derivative function evaluations per step.

To robustly solve the differential equations, one variant of the Runge–Kutta method, Runge–Kutta–Merson (RKM) method, is used over the step length T applying the technique of adaptive step size [1, 2]. This means that the step size for the numerical method changes according to a certain criterion for robust numerical analysis.

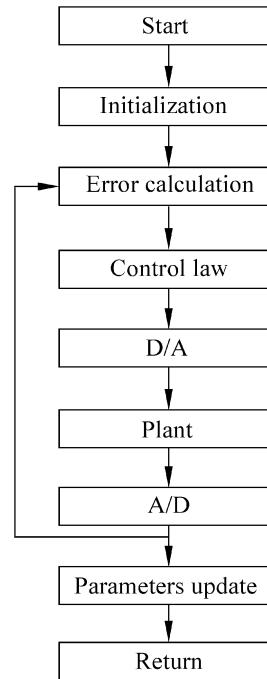
The Runge–Kutta–Merson (RKM) method is presented below as

$$x_{n+1} = x_n + \frac{1}{6}(k_1 + 4k_4 + k_5) \quad (9.3)$$

where

$$\begin{aligned} k_1 &= Tf(t_n, x_n) \\ k_2 &= Tf\left(t_n + \frac{1}{3}T, x_n + \frac{1}{3}k_1\right) \\ k_3 &= Tf\left(t_n + \frac{1}{3}T, x_n + \frac{1}{6}k_1 + \frac{1}{6}k_2\right) \\ k_4 &= Tf\left(t_n + \frac{1}{2}T, x_n + \frac{1}{8}k_1 + \frac{3}{8}k_3\right) \\ k_5 &= Tf\left(t_n + T, x_n + \frac{1}{2}k_1 - \frac{3}{2}k_3 + 2k_4\right). \end{aligned}$$

Fig. 9.1 Digital control algorithm block diagram



The digital control algorithm block diagram is shown in Fig. 9.1.

9.1.2 Simulation Example

We choose $f(x(t)) = 10 \sin t \cdot x$ and discrete $\dot{x} = 10 \sin t \cdot x$ with the sampling time $T_s = 0.001$; by using the Runge–Kutta–Merson (RKM) method, the result is shown in Fig. 9.2.

The program of this example is chap9_1.m, which is given in the [Appendix](#).

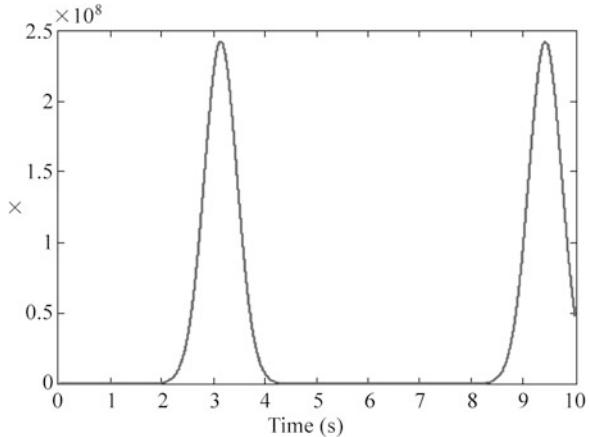
9.2 Digital Adaptive Control for SISO System

9.2.1 Introduction

Consider the Sect. 6.1, that is, “Adaptive Control with RBF Neural Network Compensation for Robotic Manipulators.”

For the first example, we assume the plant as a simple servo system as

Fig. 9.2 Discrete value of x with RKM



$$M\ddot{q} = t + d(\dot{q}) \quad (9.4)$$

where $M = 10$, d denotes friction force, and $d = -15\dot{q} - 30\text{sgn}(\dot{q})$.

Choose $x_1 = q$, and $x_2 = \dot{q}$, then the above equation becomes

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{1}{M}(\tau + d), \quad \text{and} \quad d = -15x_2 - 30\text{sgn}(x_2). \end{aligned} \quad (9.5)$$

To realize digital control, we use the Runge–Kutta–Merson (RKM) algorithm (9.3) to discrete the neural adaptive law (5.14), that is, $\hat{w} = \gamma h x^T P B + k_1 \gamma ||x|| \hat{w}$. Consider without the time variable “ t_n ” in the adaptive law, the discrete expression in Matlab is as follows:

```
w(i,1)=w_-1(i,1)+1/6*(k1+4*k4+k5);
k1=ts*(gama*h(i)*xi'*P*B+k1*gama*norm(xi)*w_-1(i,1));
k2=ts*(gama*h(i)*xi'*P*B+k1*gama*norm(xi)*(w_-1(i,1)
+1/3*k1));
k3=ts*(gama*h(i)*xi'*P*B+k1*gama*norm(xi)*(w_-1(i,1)
+1/6*k1+1/6*k2));
k4=ts*(gama*h(i)*xi'*P*B+k1*gama*norm(xi)*(w_-1(i,1)
+1/8*k1+3/8*k3));
k5=ts*(gama*h(i)*xi'*P*B+k1*gama*norm(xi)*(w_-1(i,1)
+1/2*k1-3/2*k3+2*k4));
```

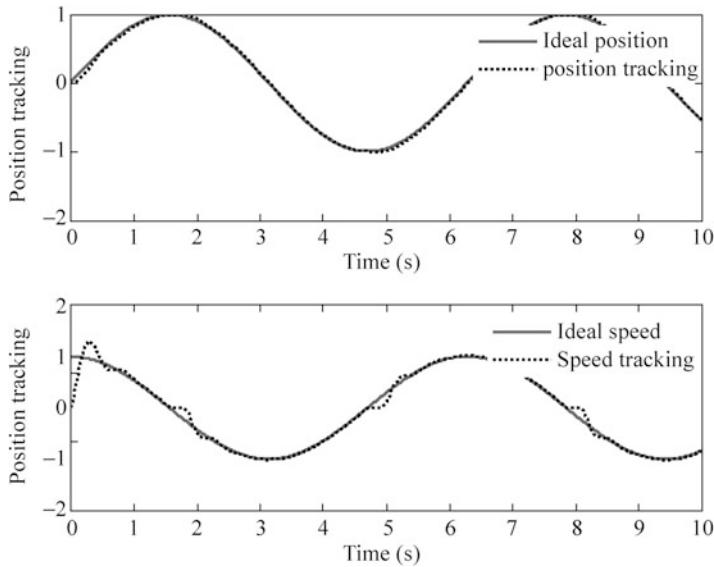


Fig. 9.3 Position and speed tracking with RBF compensation ($S = 2, F = 3$)

9.2.2 Simulation Example

For the continuous plant (9.4), we use Runge–Kutta (Matlab function “ode45”) to get discrete value with sampling time $T = 0.001$.

The desired trajectory is $q_d = \sin t$, and the initial value of the plant is $[0 \ 0]^T$. In simulation, we use control law (6.9) and second adaptive law (6.14). The parameters are chosen as $\mathbf{Q} = \begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix}$, $\alpha = 3$, $\gamma = 500$, $k_1 = 0.001$.

For RBF neural network, the parameters of Gaussian function c_i and b_i are chosen as $[-1 \ -0.5 \ 0 \ 0.5 \ 1]$ and 1.5; the initial weight value is chosen as zero.

In the simulation, we use two kinds of discrete methods to discrete the adaptive law and three kinds of controllers. The first discrete method is difference method ($S = 1$), the other discrete method is RKM method ($S = 2$). The first controller is the model-based controller without RBF compensation ($F = 1$), the second controller is the controller with precise compensation ($F = 2$), and the third controller is the model-based controller with RBF compensation ($F = 3$). If we choose $S = 2$, $F = 3$, the simulation results are shown from Figs. 9.3 to 9.4. If we only choose $F = 1$, the simulation result is shown in Fig. 9.5.

The program of this example is chap9_2.m, which is given in the [Appendix](#).

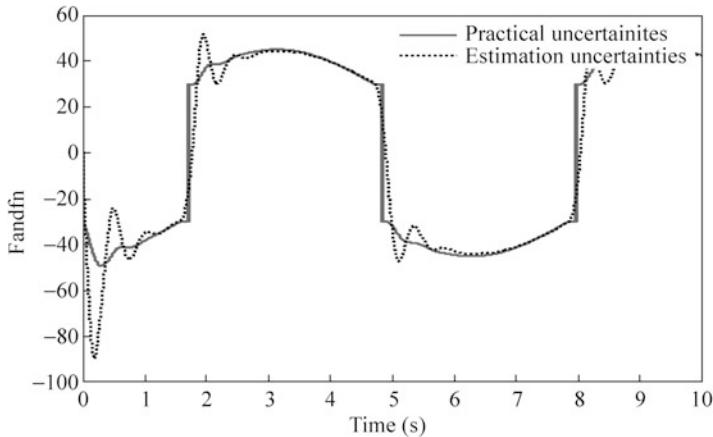


Fig. 9.4 $f(x)$ estimation with RBF ($S = 2, F = 3$)

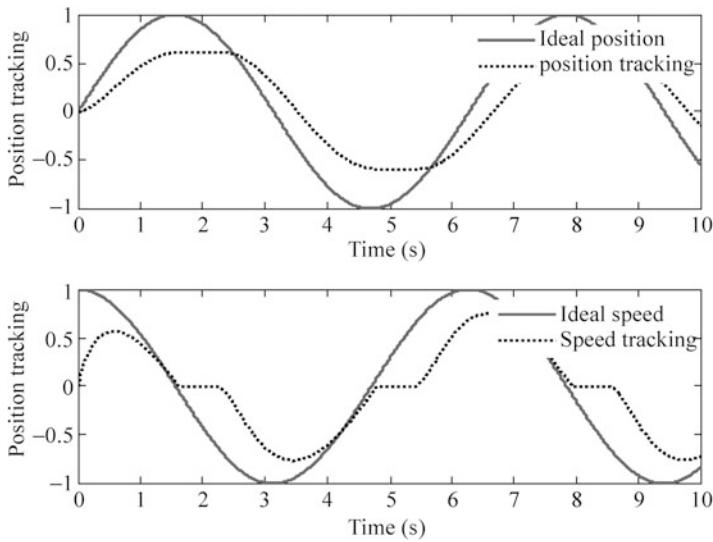


Fig. 9.5 Position and speed tracking without compensation ($F = 1$)

9.3 Digital Adaptive RBF Control for Two-Link Manipulators

9.3.1 Introduction

Consider to realize digital adaptive control for the Sect. 7.2, that is, “Adaptive RBF Control Based on Local Model Approximation for Robotic Manipulators.”

We assume the plant as a two-link manipulators. The dynamic equation is

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau}. \quad (9.6)$$

Choose $x_1 = \mathbf{q}$, $x_2 = \dot{\mathbf{q}}$, and then the above equation becomes

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= \mathbf{M}^{-1}(x_1)(\boldsymbol{\tau} - \mathbf{C}(x_1, x_2)x_2 - \mathbf{G}(x_1)).\end{aligned} \quad (9.7)$$

To simplify, we only use difference method to discrete the adaptive law (7.27), (7.28), and (7.29).

9.3.2 Simulation Example

For the continuous plant (9.7), we use Runge–Kutta (Matlab function “ode45”) to get discrete value with sampling time $T = 0.001$.

The desired trajectory is $\mathbf{q}_{d1} = \mathbf{q}_{d2} = \sin(2\pi kT)$, and the initial value of the plant is $\mathbf{q}_0 = [0 \ 0]^T$, $\dot{\mathbf{q}}_0 = [0 \ 0]^T$. In simulation, we use control law (7.22) and adaptive law (7.27), (7.28), and (7.29). The parameters are chosen as $\mathbf{K}_p = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$, $\mathbf{K}_i = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$, $\mathbf{K}_r = \begin{bmatrix} 0.10 & 0 \\ 0 & 0.10 \end{bmatrix}$, $\mathbf{A} = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}$.

For RBF neural network, we choose the structure as 2-5-1; the number of the nodes in the hidden layer is chosen as 5. For Gaussian function, the parameters c_i and b_i are chosen as $[-1 \ -0.5 \ 0 \ 0.5 \ 1]$ and 10; the initial weight vector is chosen as zero. In the adaptive law (7.27), (7.28), and (7.29), we set $\Gamma_{Mk}(i, i) = 5.0$, $\Gamma_{Ck}(i, i) = 10$, $\Gamma_{Gk}(i, i) = 5.0$, $i = 1, 2, 3, 4, 5$. The simulation results are shown from Figs. 9.6, 9.7, and 9.8.

The program of this example is chap9_3.m, which is given in the [Appendix](#).

Appendix

Programs for Sect. 9.1.2

Program: chap9_1.m

```
clear all;
close all;
ts=0.001;      %Sampling time
x_1=0.5;
```

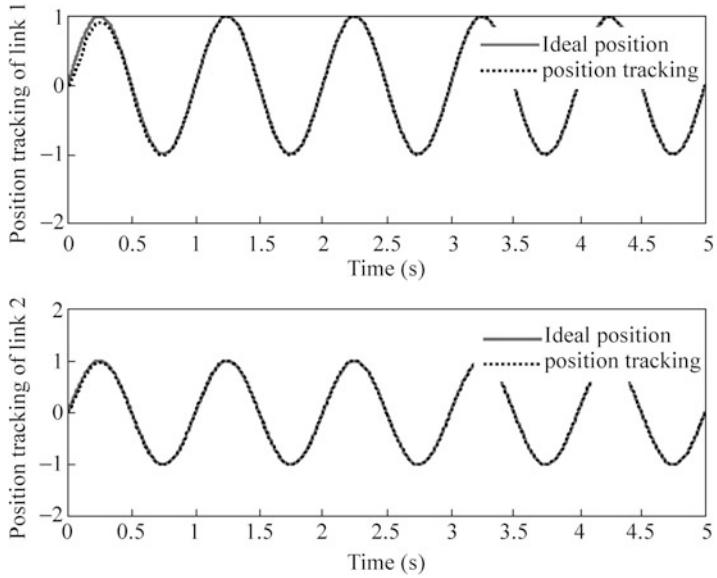


Fig. 9.6 Position tracking with RBF compensation

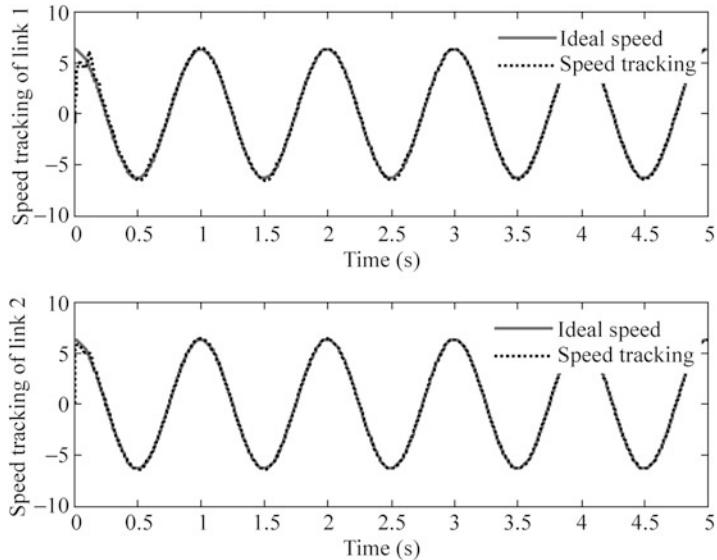


Fig. 9.7 Speed tracking with RBF compensation

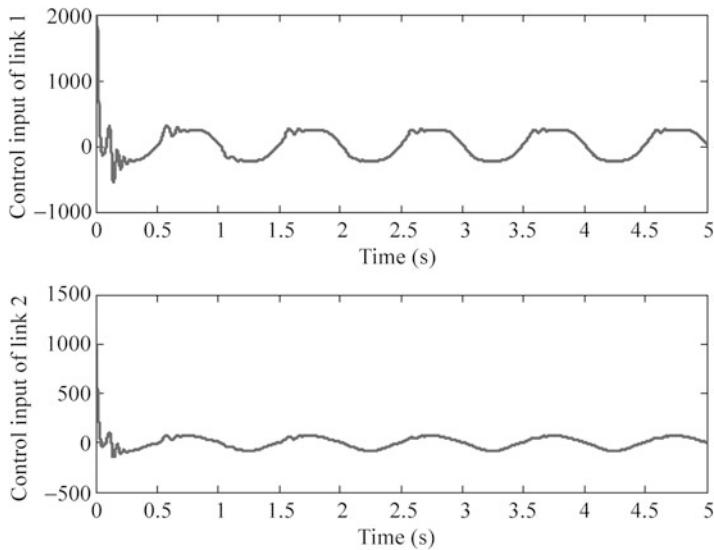


Fig. 9.8 Control input

```

for k=1:1:10000      %dx=10*sint*x
    t(k)=k*ts;
    k1=ts*10*sin(t(k))*x_1;
    k2=ts*10*sin(t(k))*(x_1+1/3*k1);
    k3=ts*10*sin(t(k))*(x_1+1/6*k1+1/6*k2);
    k4=ts*10*sin(t(k))*(x_1+1/8*k1+3/8*k3);
    k5=ts*10*sin(t(k))*(x_1+1/2*k1-3/2*k3+2*k4);
    x(k)=x_1+1/6*(k1+4*k4+k5);
    x_1=x(k);

end
figure(1);
plot(t,x,'r','linewidth',2);
xlabel('time(s)');ylabel('x');

```

Programs for Sect. 9.2.2

Main program: chap9_2.m

```

%Discrete RBF control for Motor
clear all;
close all;
ts=0.001; %Sampling time

```

```

node=5; %Number of neural nets in hidden layer
gama=100;
c=0.5* [-2 -1 0 1 2;
          -2 -1 0 1 2];
b=1.5*ones(5,1);
h=zeros(node,1);

alfa=3;
kp=alfa^2;
kv=2*alfa;
q_1=0;dq_1=0;tol_1=0;
xk=[0 0];
w_1=0.1*ones(node,1);

A=[0   1;
   -kp -kv];
B=[0;1];
Q=[50 0;
   0 50];
P=lyap(A',Q);
eig(P);
k1=0.001;
for k=1:1:10000
time(k)=k*ts;

qd(k)=sin(k*ts);
dqd(k)=cos(k*ts);
ddqd(k)=-sin(k*ts);

tSpan=[0 ts];
para=tol_1;           %D/A
[t,xx]=ode45('chap9_2plant',tSpan,xk,[],para); %Plant
xk=xx(length(xx),:); %A/D

q(k)=xk(1);
%dq(k)=xk(2);
dq(k)=(q(k)-q_1)/ts;
ddq(k)=(dq(k)-dq_1)/ts;

e(k)=q(k)-qd(k);
de(k)=dq(k)-dqd(k);

xi=[e(k);de(k)];

```

```

for i=1:1:node
    S=2;
    if S==1
        w(i,1)=w_1(i,1)+ts*(gama*h(i)*xi'*P*B+k1*gama*
            norm(xi)*w_1(i,1)); %Adaptive law
    elseif S==2
        k1=ts*(gama*h(i)*xi'*P*B+k1*gama*norm(xi)*
            w_1(i,1));
        k2=ts*(gama*h(i)*xi'*P*B+k1*gama*norm(xi)*
            (w_1(i,1)+1/3*k1));
        k3=ts*(gama*h(i)*xi'*P*B+k1*gama*norm(xi)*
            (w_1(i,1)+1/6*k1+1/6*k2));
        k4=ts*(gama*h(i)*xi'*P*B+k1*gama*norm(xi)*
            (w_1(i,1)+1/8*k1+3/8*k3));
        k5=ts*(gama*h(i)*xi'*P*B+k1*gama*norm(xi)*
            (w_1(i,1)+1/2*k1-3/2*k3+2*k4));
        w(i,1)=w_1(i,1)+1/6*(k1+4*k4+k5);
    end
end
h=zeros(5,1);
for j=1:1:5
    h(j)=exp(-norm(xi-c(:,j))^2/(2*b(j)*b(j)));
end
fn(k)=w'*h;
M=10;

tol1(k)=M*(ddqd(k)-kv*de(k)-kp*e(k));
dq(k)=-15*dq(k)-30*sign(dq(k));
f(k)=d(k);

F=3;
if F==1           %No compensation
    fn(k)=0;
    tol(k)=tol1(k);
elseif F==2 %Modified computed torque controller
    fn(k)=0;
    tol2(k)=-f(k);
    tol(k)=tol1(k)+tol2(k);
elseif F==3 %RBF compensated controller
    tol2(k)=-fn(k);
    tol(k)=tol1(k)+1*tol2(k);
end
q_1=q(k);
dq_1=dq(k);
w_1=w;

```

```

tol_1=tol(k);
end
figure(1);
subplot(211);
plot(time,qd,'r',time,q,'k:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal position','position tracking');
subplot(212);
plot(time,dqd,'r',time,dq,'k:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking');
legend('ideal speed','speed tracking');

figure(2);
plot(time,tol,'r','linewidth',2);
xlabel('time(s)'),ylabel('Control input of single link');
if F==2|F==3
    figure(3);
    plot(time,f,'r',time,fn,'k:','linewidth',2);
    xlabel('time(s)'),ylabel('f and fn');
    legend('Practical uncertainties','Estimation
        uncertainties');
end

```

Program for plant: chap9_2plant.m

```

function dx=Plant(t,x,flag,para)
dx=zeros(2,1);
tol=para;
M=10;
d=-15*x(2)-30*sign(x(2));
dx(1)=x(2);
dx(2)=1/M*(tol+d);

```

Programs for Sect. 9.3.2

Main program: chap9_3.m

```

%Discrete RBF control for two-link manipulators
clear all;
close all;
T=0.001; %Sampling time
xk=[0 0 0 0];

```

```

tol1_1=0;
tol2_1=0;
ei=0;

node=5;
c_M=[-1 -0.5 0 0.5 1;
      -1 -0.5 0 0.5 1];
c_C=[-1 -0.5 0 0.5 1;
      -1 -0.5 0 0.5 1;
      -1 -0.5 0 0.5 1;
      -1 -0.5 0 0.5 1];
c_G=[-1 -0.5 0 0.5 1;
      -1 -0.5 0 0.5 1];
b=10;

W_M11_1=zeros(node,1);W_M12_1=zeros(node,1);
W_M21_1=zeros(node,1);W_M22_1=zeros(node,1);

W_C11_1=zeros(node,1);W_C12_1=zeros(node,1);
W_C21_1=zeros(node,1);W_C22_1=zeros(node,1);

W_G1_1=zeros(node,1);W_G2_1=zeros(node,1);

Hur=[5 0;0 5];
for k=1:1:5000
if mod(k,200)==1
    k
end
time(k) = k*T;

qd1(k)=sin(2*pi*k*T);
qd2(k)=sin(2*pi*k*T);
dqd1(k)=2*pi*cos(2*pi*k*T);
dqd2(k)=2*pi*cos(2*pi*k*T);
ddqd1(k)=-(2*pi)^2*sin(2*pi*k*T);
ddqd2(k)=-(2*pi)^2*sin(2*pi*k*T);

para=[tol1_1 tol2_1];          %D/A
tSpan=[0 T];

[t,xx]=ode45('chap9_3plant',tSpan,xk,[],para);
%A/D speed
xk = xx(length(xx),:); %A/D position
q1(k)=xk(1);
dq1(k)=xk(2);
q2(k)=xk(3);
dq2(k)=xk(4);

q=[q1(k);q2(k)];
z=[q1(k);q2(k);dq1(k);dq2(k)];

```

```

e1(k)=qd1(k)-q1(k);
de1(k)=dqd1(k)-dq1(k);
e2(k)=qd2(k)-q2(k);
de2(k)=dqd2(k)-dq2(k);

e=[e1(k);e2(k)];
de=[de1(k);de2(k)];

r=de+Hur*e;
dqdr=[dqdr1(k);dqdr2(k)];
dqr=dqdr+Hur*e;
ddqdr=[ddqdr1(k);ddqdr2(k)];
ddqr=ddqdr+Hur*de;
for j=1:1:node
    h_M11(j)=exp(-norm(q-c_M(:,j))^2/(b^2));
    h_M21(j)=exp(-norm(q-c_M(:,j))^2/(b^2));
    h_M12(j)=exp(-norm(q-c_M(:,j))^2/(b^2));
    h_M22(j)=exp(-norm(q-c_M(:,j))^2/(b^2));
end
for j=1:1:node
    h_C11(j)=exp(-norm(z-c_C(:,j))^2/(b^2));
    h_C21(j)=exp(-norm(z-c_C(:,j))^2/(b^2));
    h_C12(j)=exp(-norm(z-c_C(:,j))^2/(b^2));
    h_C22(j)=exp(-norm(z-c_C(:,j))^2/(b^2));
end
for j=1:1:node
    h_G1(j)=exp(-norm(q-c_G(:,j))^2/(b^2));
    h_G2(j)=exp(-norm(q-c_G(:,j))^2/(b^2));
end

T_M11=5*eye(node);
T_M21=5*eye(node);
T_M12=5*eye(node);
T_M22=5*eye(node);
T_C11=10*eye(node);
T_C21=10*eye(node);
T_C12=10*eye(node);
T_C22=10*eye(node);
T_G1=5*eye(node);
T_G2=5*eye(node);

for i=1:1:node
    W_M11(i,1)=W_M11_1(i,1)+T*T_M11(i,i)*h_M11(i)
                *ddqr(1)*r(1);
    W_M21(i,1)=W_M21_1(i,1)+T*T_M21(i,i)*h_M21(i)
                *ddqr(1)*r(2);

```

```

W_M12(i,1)=W_M12_1(i,1)+T*T_M12(i,i)*h_M12(i)
           *ddqr(2)*r(1);
W_M22(i,1)=W_M22_1(i,1)+T*T_M22(i,i)*h_M22(i)
           *ddqr(2)*r(2);
W_C11(i,1)=W_C11_1(i,1)+T*T_C11(i,i)*h_C11(i)
           *dqr(1)*r(1);
W_C21(i,1)=W_C21_1(i,1)+T*T_C21(i,i)*h_C21(i)
           *dqr(1)*r(2);
W_C12(i,1)=W_C12_1(i,1)+T*T_C12(i,i)*h_C12(i)*
           ddqr(2)*r(1);
W_C22(i,1)=W_C22_1(i,1)+T*T_C22(i,i)*h_C22(i)*
           ddqr(2)*r(2);

W_G1=W_G1_1+T*T_G1(i,i)*h_G1(i)*r(1);
W_G2=W_G2_1+T*T_G2(i,i)*h_G2(i)*r(2);
end
MSNN_g=[W_M11'*h_M11' W_M12'*h_M12';
         W_M21'*h_M21' W_M22'*h_M22'];
GSNN_g=[W_G1'*h_G1';
         W_G2'*h_G2'];
CDNN_g=[W_C11'*h_C11' W_C12'*h_C12';
         W_C21'*h_C21' W_C22'*h_C22'];

tol_m=MSNN_g*ddqr+CDNN_g*dqr+GSNN_g;

Kp=[20 0;0 20];
Ki=[20 0;0 20];
Kr=[1.5 0;0 1.5];
Kp=[100 0;0 100];
Ki=[100 0;0 100];
Kr=[0.1 0;0 0.1];
ei=ei+e*T;
tol=tol_m+Kp*r+Kr*sign(r)+Ki*ei;
tol1(k)=tol(1);
tol2(k)=tol(2);

W_M11_1=W_M11;
W_M21_1=W_M21;
W_M12_1=W_M12;
W_M22_1=W_M22;
W_C11_1=W_C11;
W_C21_1=W_C21;
W_C12_1=W_C12;
W_C22_1=W_C22;
W_G1_1=W_G1;
W_G2_1=W_G2;

```

```

tol1_1=tol1(k);
tol2_1=tol2(k);
end
figure(1);
subplot(211);
plot(time,qd1,'r',time,q1,'k:','linewidth',2);
xlabel('time(s)'),ylabel('Position tracking of link 1');
legend('ideal position','position tracking');
subplot(212);
plot(time,qd2,'r',time,q2,'k:','linewidth',2);
xlabel('time(s)'),ylabel('Position tracking of link 2');
legend('ideal position','position tracking');
figure(2);
subplot(211);
plot(time,dqd1,'r',time,dq1,'k:','linewidth',2);
xlabel('time(s)'),ylabel('Speed tracking of link 1');
legend('ideal speed','speed tracking');
subplot(212);
plot(time,dqd2,'r',time,dq2,'k:','linewidth',2);
xlabel('time(s)'),ylabel('Speed tracking of link 2');
legend('ideal speed','speed tracking');
figure(3);
subplot(211);
plot(time,tol1,'r','linewidth',2);
xlabel('time(s)'),ylabel('Control input of link 1');
subplot(212);
plot(time,tol2,'r','linewidth',2);
xlabel('time(s)'),ylabel('Control input of link 2');

```

Program for plant: chap9_3plant.m

```

function dx=Plant(t,x,flag,para)
%%%%%%%
x(1)=q1; x(2)=dq1; x(3)=q2; x(4)=dq2;
%%%%%
dx=zeros(4,1);

p=[2.9 0.76 0.87 3.04 0.87];
g=9.8;

M0=[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)];
C0=[-p(3)*x(4)*sin(x(3)) -p(3)*(x(2)+x(4))*sin(x(3));
     p(3)*x(2)*sin(x(3)) 0];
G0=[p(4)*g*cos(x(1))+p(5)*g*cos(x(1)+x(3));
     p(5)*g*cos(x(1)+x(3))];

```

```
tol=para(1:2);  
dq=[x(2);x(4)];  
ddq=inv(M0)*(tol'-c0*dq-G0);  
%%%%%%%%% dx(1)=dq1; dx(2)=ddq1; dx(3)=dq2; dx(4)=ddq2;  
%%%%%% dx(1)=x(2);  
dx(2)=ddq(1);  
dx(3)=x(4);  
dx(4)=ddq(2);
```

References

1. Hoffman JD (1992) Numerical methods for engineers and scientists. McGraw-Hill, New York
2. Quinney D (1985) An introduction to the numerical solution of differential equations. Wiley, New York

Chapter 10

Discrete Neural Network Control

Abstract This chapter introduces two kinds of adaptive discrete neural network controllers for discrete nonlinear system, including a direct RBF controller and an indirect RBF controller. For the two control laws, the adaptive laws are designed based on the Lyapunov stability theory; the closed-loop system stability can be achieved.

Keywords Discrete RBF neural network • Discrete system • Adaptive control

10.1 Introduction

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 [1–15], where a direct adaptive neural controller for second-order discrete-time nonlinear systems was proposed in [14], and a NN-based adaptive controller without off-line training is presented for a class of discrete-time multi-input multi-output (MIMO) nonlinear systems [15].

In this chapter, we introduce two typical examples for discrete neural network controller design: analysis and simulation.

10.2 Direct RBF Control for a Class of Discrete-Time Nonlinear System

10.2.1 System Description

A nonlinear model is

$$y(k+1) = f(y(k), u(k)). \quad (10.1)$$

Assumption.

1. The unknown nonlinear function $f(\cdot)$ is continuous and differentiable.
2. The neural number of hidden neurons is l .
3. Assume that the partial derivative $g_1 \geq \left| \frac{\partial f}{\partial u} \right| > \epsilon > 0$, where both ϵ and g_1 are positive constants.

Assume that $y_m(k+1)$ is the system's desired output at time $k+1$. In the ideal case, there is no disturbance; we can show that if the control input $u^*(k)$ satisfying

$$f(y(k), u^*(k)) - y_m(k+1) = 0. \quad (10.2)$$

Then the system's output tracking error will converge to zero.

Define the tracking error as $e(k) = y(k) - y_m(k)$, and then the tracking error dynamic equation is given as follows:

$$e(k+1) = f(y(k), u(k)) - y_m(k+1)$$

10.2.2 Controller Design and Stability Analysis

For the system (10.1), a RBF neural network was designed to realize the controller directly [12, 16]. Figure 10.1 shows the closed-loop neural-based adaptive control scheme.

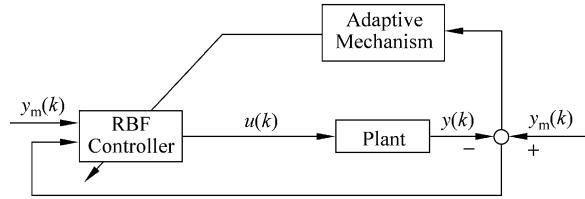
The ideal control input is u^* , and

$$u^*(k) = u^*(z) = w^{*T} h(z) + \epsilon_u(z). \quad (10.3)$$

On the compact set Ω_z , the ideal neural network weights w^* and the approximation error are bounded by

$$\|w^*\| \leq w_m, |\epsilon_u(z)| \leq \epsilon_l \quad (10.4)$$

Fig. 10.1 Block diagram of direct RBF control scheme



where w_m and ϵ_l are positive constants.

Define $\hat{w}(k)$ as the actual neural network weight value, the control law is designed as by using RBF neural network directly

$$u(k) = \hat{w}^T(k)h(z) \quad (10.5)$$

where $h(z)$ is Gaussian function output and $z(k)$ is input of RBF.

By noticing (10.3), we have

$$\begin{aligned} u(k) - u^*(k) &= \hat{w}^T(k)h(z) - (w^{*T}h(z) + \epsilon_u(z)) \\ &= \tilde{w}^T(k)h(z) - \epsilon_u(z) \end{aligned} \quad (10.6)$$

where $\tilde{w}(k) = \hat{w}(k) - w^*$ is the weight approximation error.

The weight update law was designed as follows [12, 16]:

$$\hat{w}(k+1) = \hat{w}(k) - \gamma(h(z)e(k+1) + \sigma\hat{w}(k)) \quad (10.7)$$

where $\gamma > 0$ is a positive number and $\sigma > 0$.

Subtracting w^* to both sides of Eq. (10.6), we have

$$\tilde{w}(k+1) = \tilde{w}(k) - \gamma(h(z)e(k+1) + \sigma\hat{w}(k)) \quad (10.8)$$

Mean Value Theorem. If a function $f(x)$ is continuous on the closed interval $[a, b]$ and differentiable on the open interval (a, b) , then there exists a point c in (a, b) such that

$$f(b) = f(a) + (b-a)f'(c)|_{c \in (a,b)}. \quad (10.9)$$

Using mean value theorem, let $a = u^*(k)$, $b = u(k) = u^*(k) + \tilde{w}^T(k)h(z) - \epsilon_u(z)$, $c = \zeta$, noticing Eqs. (10.2) and (10.6), then

$$\begin{aligned} f(y(k), u(k)) &= f(y(k), u^*(k) + \tilde{w}^T(k)h(z) - \epsilon_u(z)) \\ &= f(y(k), u^*(k)) + (\tilde{w}^T(k)h(z) - \epsilon_u(z))f_u \\ &= y_m(k+1) + (\tilde{w}^T(k)h(z) - \epsilon_u(z))f_u \end{aligned}$$

where $f_u = \frac{\partial f}{\partial u}|_{u=\zeta}$, $\zeta \in (u^*(k), u(k))$.

Then we get

$$\begin{aligned} e(k+1) &= f(y(k), u(k)) - y_m(k+1) \\ &= (\tilde{w}^T(k)h(z) - \varepsilon_u(z))f_u \end{aligned}$$

and

$$\tilde{w}^T(k)h(z) = \frac{e(k+1)}{f_u} + \varepsilon_u(z). \quad (10.10)$$

The stability analysis of the closed system is given by Zhang et al. [16] as follows.

Firstly the Lyapunov function is designed as

$$J(k) = \frac{1}{g_1}e^2(k) + \frac{1}{\gamma}\tilde{w}(k)^T\tilde{w}(k). \quad (10.11)$$

Then we have

$$\begin{aligned} \Delta J(k) &= J(k+1) - J(k) \\ &= \frac{1}{g_1}(e^2(k+1) - e^2(k)) + \frac{1}{\gamma}\tilde{w}(k+1)^T\tilde{w}(k+1) - \frac{1}{\gamma}\tilde{w}(k)^T\tilde{w}(k) \\ &= \frac{1}{g_1}(e^2(k+1) - e^2(k)) + \frac{1}{\gamma}(\tilde{w}(k) - \gamma(h(z)e(k+1) + \sigma\hat{w}(k)))^T \\ &\quad (\tilde{w}(k) - \gamma(h(z)e(k+1) + \sigma\hat{w}(k))) - \frac{1}{\gamma}\tilde{w}(k)^T\tilde{w}(k) \\ &= \frac{1}{g_1}(e^2(k+1) - e^2(k)) - 2\tilde{w}(k)^Th(z)e(k+1) - 2\sigma\tilde{w}(k)^T\hat{w}(k) \\ &\quad + \gamma h^T(z)h(z)e^2(k+1) + 2\gamma\sigma\tilde{w}^T(k)h(z)e(k+1) + \gamma\sigma^2\hat{w}(k)^T\hat{w}(k). \end{aligned}$$

Since

$$|h_i(z)| \leq 1, \quad \|h(z)\| \leq \sqrt{l} \leq l, \quad h^T(z)h(z) = \|h(z)\|^2 \leq l, \quad i = 1, 2, \dots, l$$

$$\begin{aligned} 2\sigma\tilde{w}(k)^T\hat{w}(k) &= \sigma\tilde{w}(k)^T(\tilde{w}(k) + w^*) + \sigma(\hat{w}(k) - w^*)^T\hat{w}(k) \\ &= \sigma\left(\|\tilde{w}(k)\|^2 + \|\hat{w}(k)\|^2 + \tilde{w}(k)^Tw^* - w^*\hat{w}(k)\right) = \sigma\left(\|\tilde{w}(k)\|^2 + \|\hat{w}(k)\|^2 - \|w^*\|^2\right) \end{aligned}$$

$$\gamma h^T(z)h(z)e^2(k+1) \leq \gamma le^2(k+1)$$

$$2\gamma\sigma\hat{w}^T(k)h(z)e(k+1) \leq \gamma\sigma l \left[\|\hat{w}(k)\|^2 + e^2(k+1) \right]$$

$$\gamma\sigma^2\hat{w}^T(k)\hat{w}(k) = \gamma\sigma^2\|\hat{w}(k)\|^2,$$

we obtain

$$\begin{aligned} \Delta J(k) &\leq \frac{1}{g_1} (e^2(k+1) - e^2(k)) - 2 \left(\frac{e(k+1)}{f_u} + \varepsilon_u(z) \right) e(k+1) \\ &\quad - \sigma \left(\|\tilde{w}(k)\|^2 + \|\hat{w}(k)\|^2 - \|w^*\|^2 \right) + \gamma l e^2(k+1) \\ &\quad + \gamma\sigma l \left[\|\hat{w}(k)\|^2 + e^2(k+1) \right] + \gamma\sigma^2\|\hat{w}(k)\|^2 \\ &= \left(\frac{1}{g_1} - \frac{2}{f_u} + \gamma(1+\sigma)l \right) e^2(k+1) - \frac{1}{g_1} e^2(k) - 2\varepsilon_u(z)e(k+1) \\ &\quad - \sigma\|\tilde{w}(k)\|^2 + \sigma\|w^*\|^2 + \sigma(-1 + \gamma l + \gamma\sigma)\|\hat{w}(k)\|^2. \end{aligned}$$

Noticing Assumption (10.3), from $0 < \varepsilon < f_u \leq g_1$, we can derive that

$$\begin{aligned} \frac{1}{g_1} - \frac{2}{f_u} &\leq \frac{1}{g_1} - \frac{2}{g_1} = -\frac{1}{g_1} < 0 \\ -2\varepsilon_u(z)e(k+1) &\leq k_0\varepsilon_1^2 + \frac{1}{k_0}e^2(k+1) \end{aligned}$$

where k_0 is a positive number. Thus, we have

$$\begin{aligned} \Delta J(k) &\leq \left(-\frac{1}{g_1} + \gamma(1+\sigma)l + \frac{1}{k_0} \right) e^2(k+1) + \sigma(\gamma l + \gamma\sigma - 1)\|\hat{w}(k)\|^2 \\ &\quad - \frac{1}{g_1} e^2(k) - \sigma\|\tilde{w}(k)\|^2 + \sigma w_m^2 + k_0\varepsilon_1^2 \\ &= - \left(\frac{1}{g_1} - (1+\sigma)l\gamma - \frac{1}{k_0} \right) e^2(k+1) + \sigma((l+\sigma)\gamma - 1)\|\hat{w}(k)\|^2 \\ &\quad - \frac{1}{g_1} (e^2(k) - \beta) - \sigma\|\hat{w}(k)\|^2 \end{aligned}$$

where β is a positive number as

$$\beta = g_1(\sigma w_m^2 + k_0\varepsilon_1^2) \tag{10.12}$$

Choosing the positive constants as k_0 , λ and σ , these constants must be satisfied the following inequalities: $\frac{1}{g_1} - \frac{1}{k_0} \geq 0$, $\frac{1}{g_1} - (1+\sigma)l\gamma - \frac{1}{k_0} \geq 0$, $(l+\sigma)\gamma - 1 \leq 0$, that is,

$$0 < g_1 \leq k_0 \quad (10.13)$$

$$0 < (1 + \sigma)l\gamma \leq \frac{1}{g_1} - \frac{1}{k_0} \quad (10.14)$$

$$0 < (l + \sigma)\gamma \leq 1 \quad (10.15)$$

Then we have $\Delta J(k) \leq 0$ once $e^2(k) \geq \beta$. This states that for all $k \geq 0$, $J(k)$ is bounded because

$$J(k) = J(0) + \sum_{j=0}^k \Delta J(j) < \infty$$

Define compact set $\Omega_e = \{e | e^2 \leq \beta\}$, then we can see that the tracking error $e(k)$ will converge to Ω_e if $e(k)$ is out of compact Ω_e .

10.2.3 Simulation Examples

10.2.3.1 First Example: For Linear Plant

Consider a linear discrete-time system as

$$\begin{aligned} x_1(k+1) &= x_2(k) \\ x_2(k+1) &= u(k) \\ y(k) &= x_1(k). \end{aligned}$$

Its model is described as the following form

$$y(k+1) = f(y(k), u(k)).$$

In the program, we set $M = 1$ for linear plant. Since $g_1 \geq \frac{\partial f}{\partial u} = 1$, from simulation test, we can set $g_1 = 5$, and from (10.13), we can choose $k_0 = 10$.

From (10.14), we have $0 < (1 + \sigma)l\gamma \leq \frac{1}{5} - \frac{1}{10} = \frac{1}{10} = 0.10$; from (10.15), we have $0 < (l + \sigma)\gamma \leq 1$; if we choose $l = 9$, we can get the conditions as: $0 < 9(1 + \sigma)\gamma \leq 0.10$ and $0 < (9 + \sigma)\gamma \leq 1$.

To satisfy (10.14) and (10.15), we can choose $\gamma = 0.01$, $\sigma = 0.001$.

For RBF neural network, the structure is 2-9-1, the input is chosen as $z(k) = [x_1(k) \ x_2(k)]^T$, the parameters of Gaussian function c_i and b_i are chosen as $[-2 \ -1.5 \ -1.0 \ -0.5 \ 0 \ 0.5 \ 0.5 \ 1.0 \ 1.5 \ 2]$ and 2.0, and the initial weight value is chosen as random value in the range (0,1). The initial value of the plant is

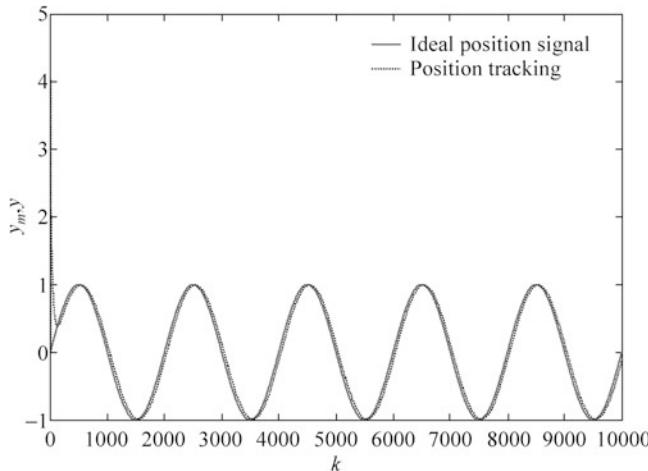


Fig. 10.2 Position tracking ($M = 1$)

$[0 \ 0]$. The reference signal is $y_m(k) = \sin\left(\frac{\pi}{1000}k\right)$. The simulation results are shown from Figs. 10.2, 10.3, and 10.4.

The program of this example is chap10_1.m ($M = 1$), which is given in the Appendix.

10.2.3.2 Second Example: For Nonlinear Plant

Consider a nonlinear discrete-time system as

$$\begin{aligned}x_1(k+1) &= x_2(k) \\x_2(k+1) &= \frac{x_1(k)x_2(k)(x_1(k) + 2.5)}{1 + x_1^2(k) + x_2^2(k)} + u(k) + 0.1u^3(k) \\y(k) &= x_1(k)\end{aligned}$$

Its model is described as the following form

$$y(k+1) = f(y(k), u(k)).$$

In the program, we set $M = 2$ for nonlinear plant. Since $g_1 \geq \frac{\partial f}{\partial u} = 1 + 0.3u^2(k)$, from simulation test, we can set $g_1 = 5$, and from (10.13), we can choose $k_0 = 10$.

From (10.14), we have $0 < (1 + \sigma)l\gamma \leq \frac{1}{5} - \frac{1}{10} = \frac{1}{10} = 0.10$, from (10.15), we have $0 < (l + \sigma)\gamma \leq 1$, if we choose $l = 9$, we can get the conditions as: $0 < 9(1 + \sigma)\gamma \leq 0.10$ and $0 < (9 + \sigma)\gamma \leq 1$.

To satisfy (10.14) and (10.15), we can choose $\gamma = 0.01$, $\sigma = 0.001$.

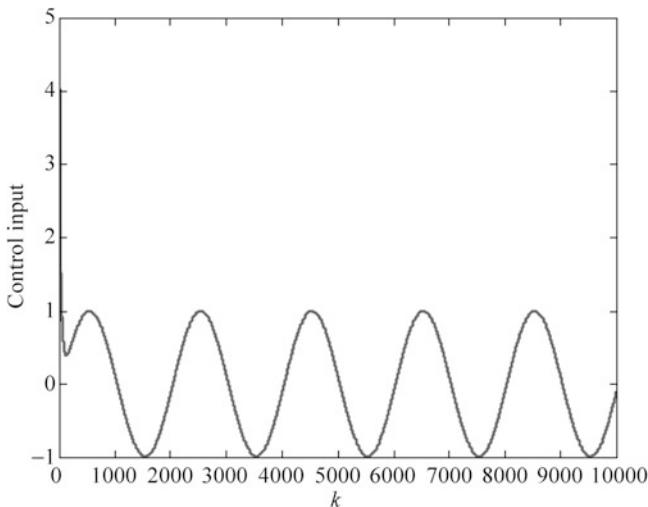


Fig. 10.3 Control input ($M = 1$)

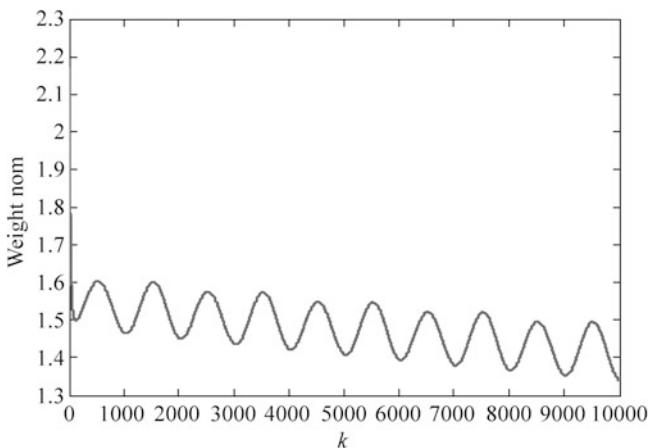


Fig. 10.4 Weight norm ($M = 1$)

For RBF neural network, the structure is 2-9-1, the input is chosen as $\mathbf{z}(k) = [x_1(k) \quad x_2(k)]^T$, the parameters of Gaussian function c_i and b_i are chosen as $[-2 \quad -1.5 \quad -1.0 \quad -0.5 \quad 0 \quad 0.5 \quad 1.0 \quad 1.5 \quad 2]$ and 2.0, and the initial weight value is chosen as random value in the range (0,1). The initial value of the plant is $[0 \quad 0]$. The reference signal is $y_m(k) = \sin(\frac{\pi}{1000}k)$. The simulation results are shown from Figs. 10.5, 10.6, and 10.7.

The program of this example is chap10_1.m ($M = 2$), which is given in the Appendix.

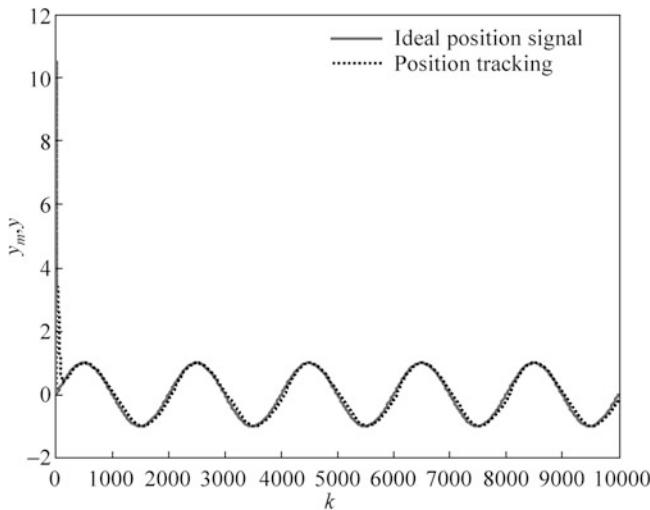


Fig. 10.5 Position tracking ($M = 2$)

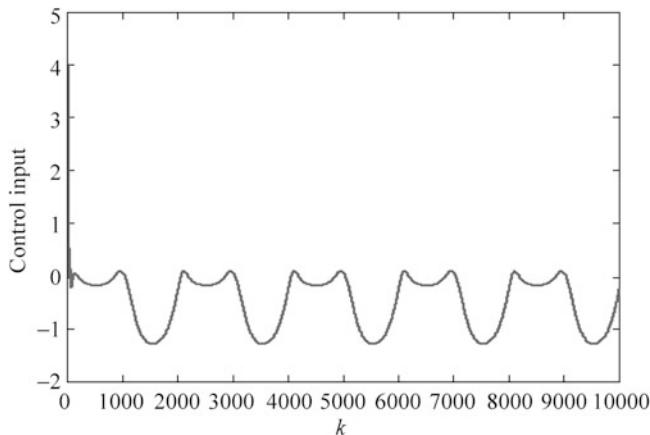


Fig. 10.6 Control input ($M = 2$)

10.3 Adaptive RBF Control for a Class of Discrete-Time Nonlinear System

10.3.1 System Description

Consider a nonlinear discrete system as follows:

$$y(k+1) = f(\mathbf{x}(k)) + u(k) \quad (10.16)$$

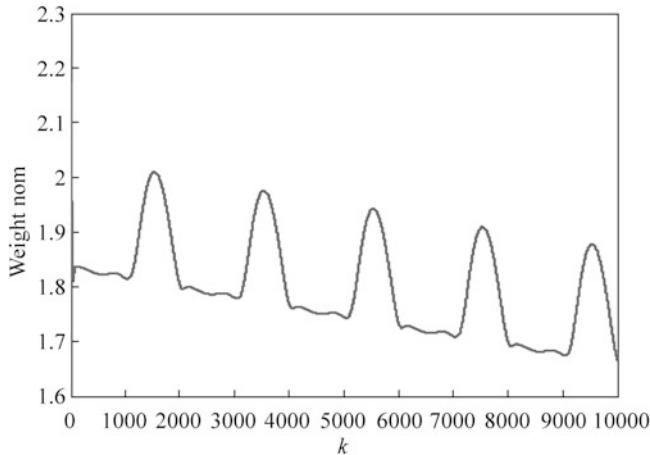


Fig. 10.7 Weight norm ($M = 2$)

where $\mathbf{x}(k) = [y(k) \ y(k-1) \dots \ y(k-n+1)]^T$ 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.

10.3.2 Traditional Controller Design

The tracking error $e(k)$ is defined as $e(k) = y(k) - y_d(k)$. If $f(\mathbf{x}(k))$ were known, a feedback linearization-type control law can be designed as

$$u(k) = y_d(k+1) - f(\mathbf{x}(k)) - c_1 e(k). \quad (10.17)$$

Submitting (10.17) into (10.16), we can get an asymptotical convergence error dynamic system as

$$e(k+1) + c_1 e(k) = 0 \quad (10.18)$$

where $|c_1| < 1$.

10.3.3 Adaptive Neural Network Controller Design

If $f(\mathbf{x}(k))$ is unknown, RBF neural network can be used to approximate $f(\mathbf{x}(k))$. The network output is given as

$$\hat{f}(\mathbf{x}(k)) = \hat{\mathbf{w}}(k)^T \mathbf{h}(\mathbf{x}(k)) \quad (10.19)$$

where $\hat{\mathbf{w}}(k)$ denotes the network output weight vector and $\mathbf{h}(\mathbf{x}(k))$ denotes the vector of Gaussian basis functions.

Given any arbitrary nonzero approximation error bound ε_f , there exist some optimal weight vector \mathbf{w}^* such that

$$f(\mathbf{x}) = \hat{f}(\mathbf{x}, \mathbf{w}^*) - \Delta_f(\mathbf{x}) \quad (10.20)$$

where $\Delta_f(\mathbf{x})$ denotes the optimal network approximation error, and $|\Delta_f(\mathbf{x})| < \varepsilon_f$.

Then we can get the general network approximation error as

$$\begin{aligned} \tilde{f}(\mathbf{x}(k)) &= f(\mathbf{x}(k)) - \hat{f}(\mathbf{x}(k)) \\ &= \hat{f}(\mathbf{x}, \mathbf{w}^*) - \Delta_f(\mathbf{x}(k)) - \hat{\mathbf{w}}(k)^T \mathbf{h}(\mathbf{x}(k)) \\ &= -\tilde{\mathbf{w}}(k)^T \mathbf{h}(\mathbf{x}(k)) - \Delta_f(\mathbf{x}(k)) \end{aligned} \quad (10.21)$$

where $\tilde{\mathbf{w}}(k) = \hat{\mathbf{w}}(k) - \mathbf{w}^*$.

The control law with RBF approximation was proposed in [17] as follows

$$u(k) = y_d(k+1) - \hat{f}(\mathbf{x}(k)) - c_1 e(k). \quad (10.22)$$

Figure 10.8 shows the closed-loop neural-based adaptive control scheme.

Substituting (10.22) into (10.16), yields

$$e(k+1) = \tilde{f}(\mathbf{x}(k)) - c_1 e(k).$$

Thus,

$$e(k) + c_1 e(k-1) = \tilde{f}(\mathbf{x}(k-1)). \quad (10.23)$$

The term (10.23) can also be expressed as

$$e(k) = \Gamma^{-1}(z^{-1}) \tilde{f}(\mathbf{x}(k-1)) \quad (10.24)$$

where $\Gamma(z^{-1}) = 1 + c_1 z^{-1}$, z^{-1} denotes the discrete-time delay operator.

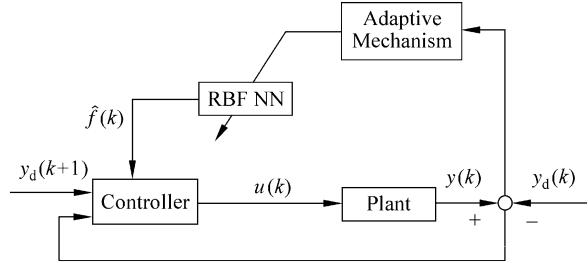
Define a new augmented error as [17]

$$e_1(k) = \beta(e(k) - \Gamma^{-1}(z^{-1}) v(k)) \quad (10.25)$$

where $\beta > 0$.

Substituting (10.24) into (10.25) yields

Fig. 10.8 Block diagram of the control scheme



$$\begin{aligned} e_1(k) &= \beta \Gamma^{-1}(z^{-1})(\tilde{f}(\mathbf{x}(k-1)) - v(k)) \\ &= \beta \frac{1}{1 + c_1 z^{-1}} (\tilde{f}(\mathbf{x}(k-1)) - v(k)) \end{aligned}$$

which leads to the relation as

$$e_1(k-1) = \frac{\beta(\tilde{f}(\mathbf{x}(k-1)) - v(k)) - e_1(k)}{c_1}. \quad (10.26)$$

The adaptive law as was proposed in [17] as

$$\Delta \hat{w}(k) = \begin{cases} \frac{\beta}{\gamma c_1} \mathbf{h}(\mathbf{x}(k-1)) e_1(k) & \text{if } |e_1(k)| > \epsilon_f/G \\ 0 & \text{if } |e_1(k)| \leq \epsilon_f/G \end{cases} \quad (10.27)$$

where $\Delta \hat{w}(k) = \hat{w}(k) - \hat{w}(k-1)$, γ and G are strictly positive constants.

10.3.4 Stability Analysis

The discrete-time Lyapunov function is designed by [17] as

$$V(k) = e_1^2(k) + \gamma \tilde{w}^T(k) \tilde{w}(k) \quad (10.28)$$

The first difference is

$$\begin{aligned} \Delta V(k) &= V(k) - V(k-1) \\ &= e_1^2(k) - e_1^2(k-1) + \gamma (\tilde{w}^T(k) + \tilde{w}^T(k-1)) (\tilde{w}(k) - \tilde{w}(k-1)). \end{aligned}$$

The stability proof was given with the following three steps [17].

Firstly, using (10.26) for $e_1(k-1)$, it follows that

$$\begin{aligned}\Delta V(k) &= e_1^2(k) - \frac{e_1^2(k) + \beta^2(\tilde{f}(\mathbf{x}(k-1)) - v(k))^2 - 2\beta(\tilde{f}(\mathbf{x}(k-1)) - v(k))e_1(k)}{c_1^2} \\ &\quad + \gamma \left((\hat{\mathbf{w}}(k) - \mathbf{w}^*)^\top + (\hat{\mathbf{w}}(k-1) - \mathbf{w}^*)^\top \right) ((\hat{\mathbf{w}}(k) - \mathbf{w}^*) - (\hat{\mathbf{w}}(k-1) - \mathbf{w}^*)) \\ &= -V_1 + \frac{2\beta(\tilde{f}(\mathbf{x}(k-1)) - v(k))e_1(k)}{c_1^2} + \gamma(\Delta\hat{\mathbf{w}}^\top(k) + 2\tilde{\mathbf{w}}^\top(k-1))\Delta\hat{\mathbf{w}}(k)\end{aligned}$$

$$\text{where } V_1 = \frac{e_1^2(k)(1 - c_1^2)}{c_1^2} + \frac{\beta^2(\tilde{f}(\mathbf{x}(k-1)) - v(k))^2}{c_1^2} \geq 0.$$

Secondly, substituting for $\tilde{f}(\mathbf{x}(k-1))$ via (10.21) yields

$$\begin{aligned}\Delta V(k) &= -V_1 + \frac{2\beta(-\tilde{\mathbf{w}}(k-1)^\top \mathbf{h}(\mathbf{x}(k-1)) - \Delta_f(\mathbf{x}(k-1)) - v(k))e_1(k)}{c_1^2} \\ &\quad + \gamma\Delta\hat{\mathbf{w}}^\top(k)\Delta\hat{\mathbf{w}}(k) + 2\gamma\tilde{\mathbf{w}}^\top(k-1)\Delta\hat{\mathbf{w}}(k) \\ &= -V_1 + 2\tilde{\mathbf{w}}^\top(k-1) \left(\gamma\Delta\hat{\mathbf{w}}(k) - \frac{\beta}{c_1^2} \mathbf{h}(\mathbf{x}(k-1))e_1(k) \right) \\ &\quad - \frac{2\beta}{c_1^2} (\Delta_f(\mathbf{x}(k-1)) + v(k))e_1(k) + \gamma\Delta\hat{\mathbf{w}}^\top(k)\Delta\hat{\mathbf{w}}(k).\end{aligned}$$

Thirdly, substituting the adaptive law (10.27) into above, $\Delta V(k)$ is

$$\Delta V(k) = \begin{cases} -V_1 - \frac{2\beta}{c_1^2} (\Delta_f(\mathbf{x}(k-1)) + v(k))e_1(k) + \\ \left(\frac{\beta}{\sqrt{\gamma}c_1^2} \right)^2 \mathbf{h}^\top(\mathbf{x}(k-1))\mathbf{h}(\mathbf{x}(k-1))e_1^2(k), & \text{if } |e_1(k)| > \varepsilon_f/G \\ -V_1 - \frac{2\beta}{c_1^2} [(\tilde{\mathbf{w}}^\top(k-1)\mathbf{h}(\mathbf{x}(k-1)) + \\ v(k) + \Delta_f(\mathbf{x}(k-1))e_1(k)], & \text{if } |e_1(k)| \leq \varepsilon_f/G \end{cases} \quad (10.29)$$

The auxiliary signal $v(k)$ must also be designed so that $e_1(k) \rightarrow 0$ could deduce $e(k) \rightarrow 0$. The auxiliary signal is designed as [17]

$$v(k) = v_1(k) + v_2(k) \quad (10.30)$$

with $v_1(k) = \frac{\beta}{2\gamma c_1^2} \mathbf{h}^\top(\mathbf{x}(k-1))\mathbf{h}(\mathbf{x}(k-1))e_1(k)$ and $v_2(k) = Ge_1(k)$.

If $|e_1(k)| > \varepsilon_f/G$, substituting for $v(k)$ in (10.30) to (10.29), it follows that

$$\begin{aligned}\Delta V(k) &= -V_1 - \frac{2\beta}{c_1^2} (\Delta_f(\mathbf{x}(k-1)) + Ge_1(k))e_1(k) \\ &\leq -\frac{2\beta}{c_1^2} (\Delta_f(\mathbf{x}(k-1)) + Ge_1(k))e_1(k).\end{aligned}$$

Since $|\Delta_f(\mathbf{x})| < \varepsilon_f$ and $|e_1(k)| > \varepsilon_f/G$, then $|e_1(k)| > \frac{|\Delta_f(\mathbf{x}(k-1))|}{G}$ and $e_1^2(k) > -\frac{\Delta_f(\mathbf{x}(k-1))e_1(k)}{G}$, thus $(\Delta_f(\mathbf{x}(k-1)) + Ge_1(k))e_1(k) > 0$, then $\Delta V(k) < 0$.

If $|e_1(k)| \leq \varepsilon_f/G$, 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 (10.25), we have $e_1(k) = \beta(e(k) - \frac{1}{1+c_1z^{-1}}v(k))$, then $e_1(k) \times (1 + c_1z^{-1}) = \beta(e(k)(1 + c_1z^{-1}) - v(k))$; therefore,

$$e_1(k) = -c_1e_1(k-1) + \beta(e(k) + c_1e(k-1) - v(k)) \quad (10.31)$$

Remark 2. If $k \rightarrow \infty$, $e_1(k) \rightarrow 0$, from (10.30), we have $v(k) \rightarrow 0$, and then from (10.31), $e(k) + c_1e(k-1) \rightarrow 0$, **considering** $|c_1| < 1$, we get $e(k) \rightarrow 0$.

Remark 3. Consider $v(k)$ as a virtual variable, for (10.30), let $v'_1(k) = \frac{\beta}{2\gamma c_1^2} \mathbf{h}^T(\mathbf{x}(k-1))\mathbf{h}(\mathbf{x}(k-1))$, then we get $v(k) = (v'_1(k) + G)e_1(k)$; substituting $v(k)$ into (10.31), we have $e_1(k) = -c_1e_1(k-1) + \beta(e(k) + c_1e(k-1) - (v'_1(k) + G)e_1(k))$, then

$$e_1(k) = \frac{-c_1e_1(k-1) + \beta(e(k) + c_1e(k-1))}{1 + \beta(v'_1(k) + G)}. \quad (10.32)$$

10.3.5 Simulation Examples

10.3.5.1 First Example: Linear Discrete-Time System

Consider a linear discrete-time system as

$$y(k) = 0.5y(k-1) + u(k-1)$$

where $f(x(k-1)) = 0.5y(k-1)$.

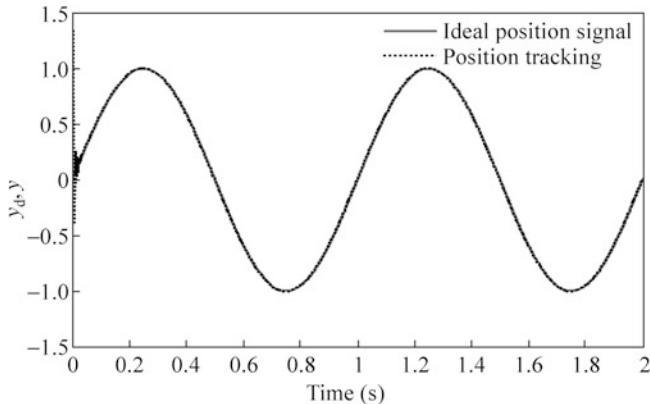


Fig. 10.9 Position tracking

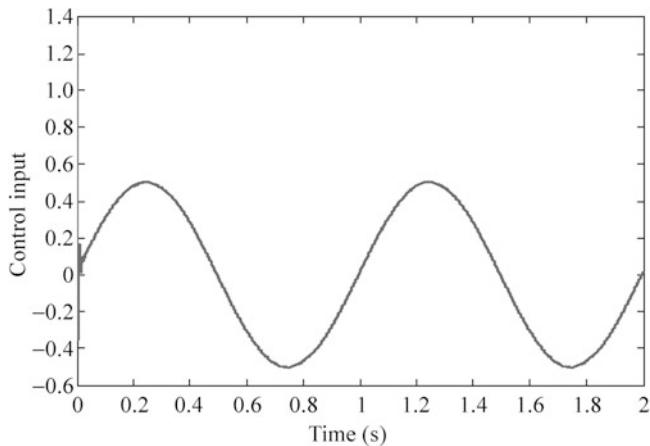


Fig. 10.10 Control input

We use RBF to approximate $f(x(k-1))$. For RBF neural network, the structure is set as 1-9-1; from $f(x(k-1))$ expression, only one input is chosen as $y(k-1)$; the parameters of Gaussian function c_i and b_i are chosen as $\begin{bmatrix} -1 & -0.5 & 0 & 0.5 & 1 \end{bmatrix}$ and $15(i=1, j=1, 2, \dots, 5)$; and 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 $y_d(k) = \sin 2\pi t$. Using the control law (10.22) with adaptive law (10.27), $e_1(k)$ is calculated by (10.32); the parameters are chosen as $c_1 = -0.01$, $\beta = 0.001$, $\gamma = 0.001$, $\gamma = 0.001$, $G = 50000$, $\epsilon_f = 0.003$. The results are shown in Figs. 10.9, 10.10, and 10.11.

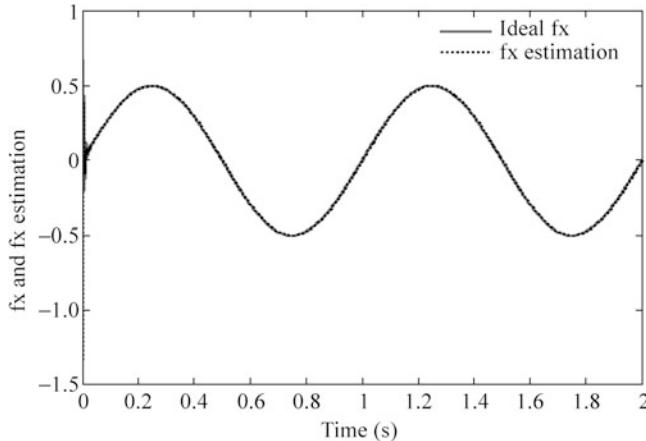


Fig. 10.11 $f(x(k-1))$ and its estimation

The program of this example is `chap10_2.m`, which is given in the [Appendix](#).

10.3.5.2 Second Example: Nonlinear Discrete-Time System

Consider a nonlinear discrete-time system as

$$y(k) = \frac{0.5y(k-1)(1-y(k-1))}{1 + \exp(-0.25y(k-1))} + u(k-1)$$

$$\text{where } f(x(k-1)) = \frac{0.5y(k-1)(1-y(k-1))}{1 + \exp(-0.25y(k-1))}.$$

Firstly, we assume $f(x(k-1))$ is known, use the control law (10.17), and set $c_1 = -0.01$, the results are shown in Figs. 10.12 and 10.13.

Then we use RBF to approximate $f(x(k-1))$. For RBF neural network, the structure is set as 1-9-1; from $f(x(k-1))$ expression, only one input $y(k-1)$ is chosen; the parameters of Gaussian function c_i and b_i are chosen as $[-2 -1.5 -1.0 -0.5 0 0.5 1.0 1.5 2]$ and $15(i=1, j=1, 2, \dots, 9)$, and 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 (10.22) with adaptive law (10.27), $e_1(k)$ is calculated by (10.32); the parameters are chosen as $c_1 = -0.01, \beta = 0.001, \gamma = 0.001, \gamma_f = 0.001, G = 50000, \varepsilon_f = 0.003$. The results are shown in Figs. 10.14, 10.15, and 10.16.

The program of this example is `chap10_3.m` and `chap10_4.m`, which are given in the [Appendix](#).

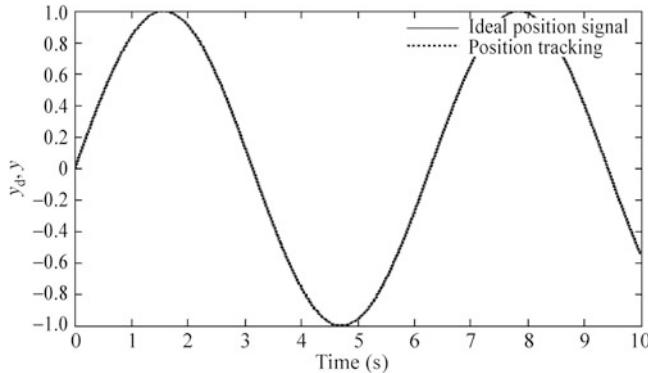


Fig. 10.12 Position tracking

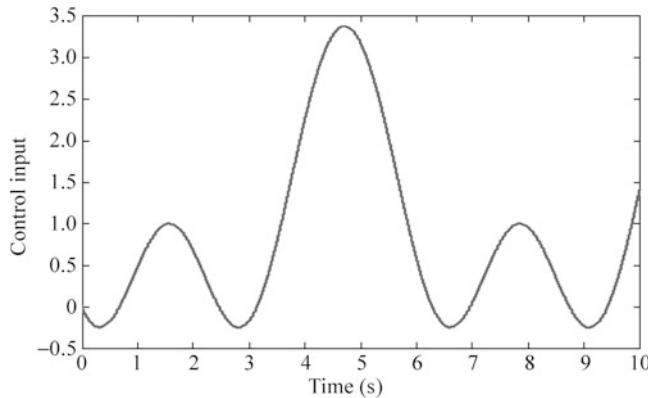


Fig. 10.13 Control input

10.3.5.3 Third Example: Nonlinear Discrete-Time System

Consider a nonlinear discrete-time system as

$$y(k) = f(x(k-1)) + u(k-1)$$

where $f(x(k-1)) = \frac{1.5y(k-1)y(k-2)}{1+y^2(k-1)+y^2(k-2)} + 0.35 \sin(y(k-1) + y(k-2))$.

Then we use RBF to approximate $f(x(k-1))$. For RBF neural network, the structure is 2-9-1; from $f(x(k-1))$ expression, two inputs are chosen as $y(k-1)$ and $y(k-2)$, the parameters of Gaussian function c_{ij} and b_j are chosen as $\begin{bmatrix} -2 & -1.5 & -1.0 & -0.5 & 0 & 0.5 & 1.0 & 1.5 & 2 \\ -2 & -1.5 & -1.0 & -0.5 & 0 & 0.5 & 1.0 & 1.5 & 2 \end{bmatrix}$ and $15(i=1,2, \dots, 9); j=1,2,\dots,9$; and

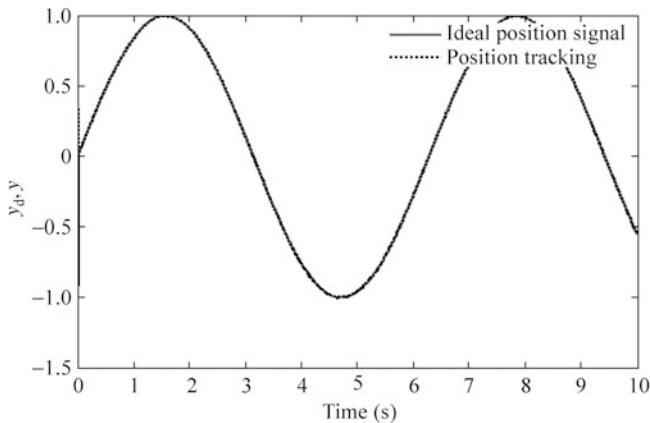


Fig. 10.14 Position tracking

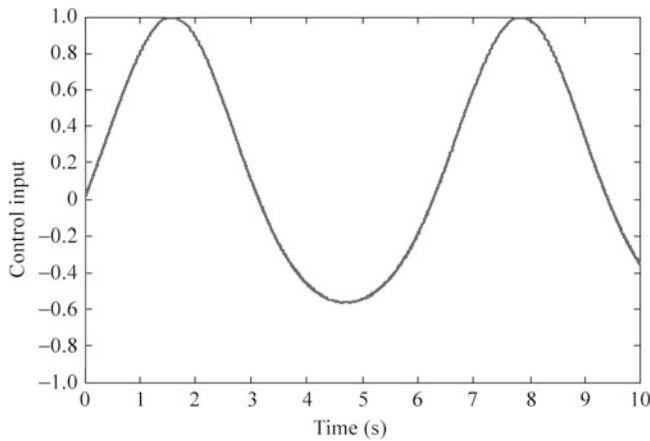


Fig. 10.15 Control input

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 (10.22) with adaptive law (10.27), $e_1(k)$ is calculated by (10.32); the parameters are chosen as $c_1 = -0.01, \beta = 0.001, \gamma = 0.001, \gamma = 0.001, G = 50000, \epsilon_f = 0.003$. The results are shown in Figs. 10.17, 10.18, and 10.19.

The program of this example is chap10_5.m, which is given in the Appendix.

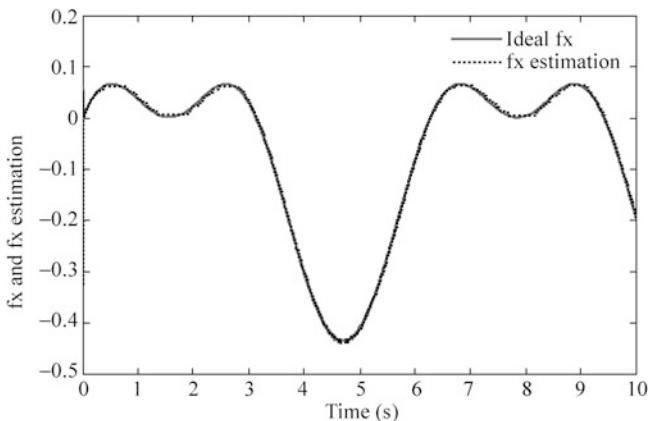


Fig. 10.16 $f(x(k-1))$ and its estimation

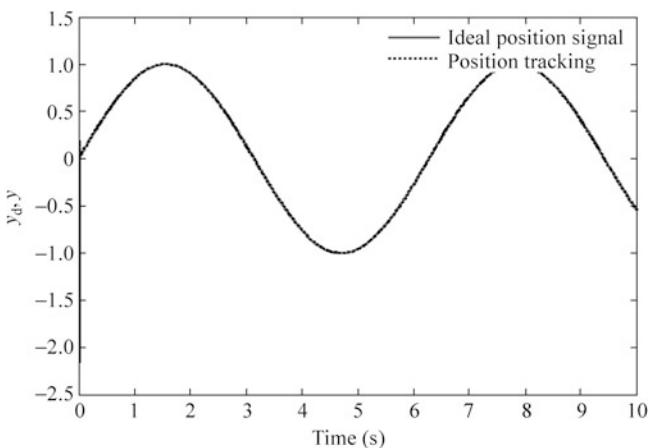


Fig. 10.17 Position tracking

Appendix

Programs for Sect. 10.2.3

Simulation program: chap10_1.m

```
%Discrete neural controller
clear all;
close all;
L=9; %Hidden neural nets
c=[-2 -1.5 -1 -0.5 0 0.5 1 1.5 2;
-2 -1.5 -1 -0.5 0 0.5 1 1.5 2];
```

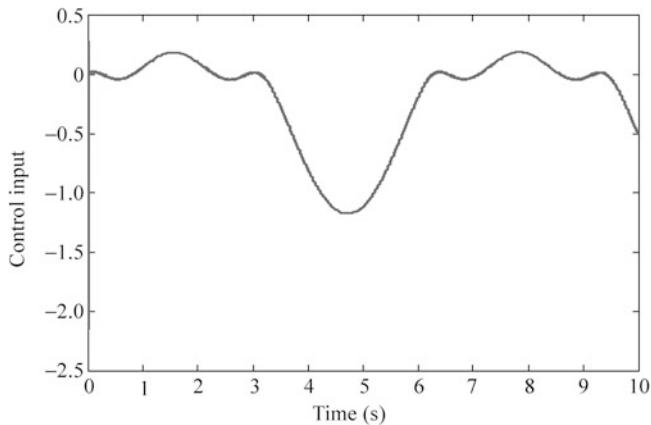


Fig. 10.18 Control input

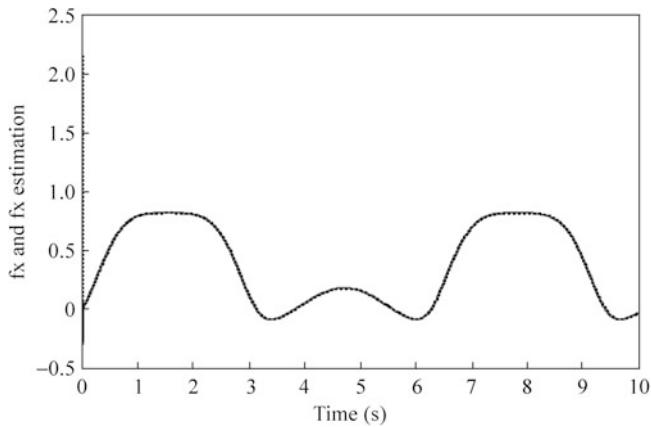


Fig. 10.19 $f(x(k-1))$ and its estimation

```

b=2;
w=rand(L,1);
w_1=w;
u_1=0;
x1_1=0;x1_2=0;
x2_1=0;
z=[0,0]';
Gama=0.01;rou=0.001;
L*(1+rou)*Gama %<=1/g1-1/k0
(L+rou)*Gama %<=1.0
for k=1:1:10000
time(k)=k;

```

```

ym(k)=sin(pi/1000*k);
y(k)=x1_1; %tol=1
e(k)=y(k)-ym(k);

M=2;
if M==1 %Linear model
x1(k)=x2_1;
x2(k)=u_1;
elseif M==2 %Nonlinear model
x1(k)=x2_1;
x2(k)=(x1(k)*x2_1*(x1(k)+2.5))/(1+x1(k)^2+x2_1^2)
+u_1+0.1*u_1^3;
end

z(1)=x1(k);z(2)=x2(k);
for j=1:1:L
    h(j)=exp(-norm(z-c(:,j))^2/(2*b^2));
end
w=w_1-Gama*(h'*e(k)+rou*w_1);
wn(k)=norm(w);

u(k)=w'*h';
%u(k)=0.20*(ym(k)-x1(k)); %P control
x1_2=x1_1;
x1_1=x1(k);

x2_1=x2(k);
w_1=w;
u_1=u(k);
end
figure(1);
plot(time,ym,'r',time,x1,'k:','linewidth',2);
xlabel('k');ylabel('ym,y');
legend('Ideal position signal','Position tracking');
figure(2);
plot(time,u,'r','linewidth',2);
xlabel('k');ylabel('Control input');
figure(3);
plot(time,wn,'r','linewidth',2);
xlabel('k');ylabel('Weight Norm');

```

Programs for Sect. 10.3.5.1

Simulation program: chap10_2.m

```
%Discrete RBF controller
clear all;
close all;
ts=0.001;

c1=-0.01;
beta=0.001;
epcf=0.003;
gama=0.001;
G=50000;

b=15;
c=[-1 -0.5 0 0.5 1];
w=rands(5,1);
w_1=w;
u_1=0;
y_1=0;
e1_1=0;
e_1=0;
fx_1=0;
for k=1:1:2000
time(k)=k*ts;

yd(k)=sin(2*pi*k*ts);
yd1(k)=sin(2*pi*(k+1)*ts);
%Nonlinear plant
fx(k)=0.5*y_1;
y(k)=fx_1+u_1;

e(k)=y(k)-yd(k);
x(1)=y_1;
for j=1:1:5
    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');

```

Programs for Sect. 10.3.5.2

Simulation program with known $f(x(k - 1))$: chap10_3.m

```

%Discrete controller
clear all;
close all;
ts=0.001;

c1=-0.01;
u_1=0;y_1=0;
fx_1=0;
for k=1:1:20000
    time(k)=k*ts;

    yd(k)=sin(k*ts);
    yd1=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);
    u(k)=yd1-fx(k)-c1*e(k);

    y_1=y(k);

```

```

u_1=u(k);
fx_1=fx(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');

```

Simulation program with unknown $f(x(k - 1))$: chap10_4.m

```

%Discrete RBF controller
clear all;
close all;
ts=0.001;

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');

```

Programs for Sect. 10.3.5.3

Simulation program: chap10_5.m

```

%Discrete RBF controller
clear all;
close all;
ts=0.001;

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;
   -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;y_2=0;
e1_1=0;e_1=0;
x=[0 0]';
fx_1=0;
for k=1:1:10000
time(k)=k*ts;

yd(k)=sin(k*ts);
yd1(k)=sin((k+1)*ts);
%Linear model
fx(k)=1.5*y_1*y_2/(1+y_1^2+y_2^2)+0.35*sin(y_1+y_2);
y(k)=fx_1+u_1;

e(k)=y(k)-yd(k);

x(1)=y_1;x(2)=y_2;
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
%u(k)=0.10*(x1d(k)-x1(k)); %P control

fx_1=fx(k);
y_2=y_1;
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');
```

References

1. Jagannathan S, Lewis FL (1994) Discrete-time neural net controller with guaranteed performance. In: Proceedings of the American control conference, Baltimore, Maryland, USA, pp 3334–3339
2. Ge SS, Li GY, Lee TH (2003) Adaptive NN control for a class of strict-feedback discrete-time nonlinear systems. *Automatica* 39(5):807–819
3. Yang C, Li Y, Ge SS, Lee TH (2010) Adaptive control of a class of discrete-time MIMO nonlinear systems with uncertain couplings. *Int J Control* 83(10):2120–2133
4. Ge SS, Yang C, Dai S, Jiao Z, Lee TH (2009) Robust adaptive control of a class of nonlinear strict-feedback discrete-time systems with exact output tracking. *Automatica* 45(11):2537–2545
5. Yang C, Ge SS, Lee TH (2009) Output feedback adaptive control of a class of nonlinear discrete-time systems with unknown control directions. *Automatica* 45(1):270–276
6. Yang C, Ge SS, Xiang C, Chai T, Lee TH (2008) Output feedback NN Control for two classes of discrete-time systems with unknown control directions in a unified approach. *IEEE Trans Neural Netw* 19(11):1873–1886
7. Ge SS, Yang C, Lee TH (2008) Adaptive robust control of a class of nonlinear strict-feedback discrete-time systems with unknown control directions. *Syst Control Lett* 57:888–895
8. Ge SS, Yang C, Lee TH (2008) Adaptive predictive control using neural network for a class of pure-feedback systems in discrete time. *IEEE Trans Neural Netw* 19(9):1599–1614
9. Zhang J, Ge SS, Lee TH (2005) Output feedback control of a class of discrete MIMO nonlinear systems with triangular form inputs. *IEEE Trans Neural Netw* 16(6):1491–1503
10. Ge SS, Zhang J, Lee TH (2004) State feedback NN control of a class of discrete MIMO nonlinear systems with disturbances. *IEEE Trans Syst, Man Cybern (Part B) Cybern* 34 (4):1630–1645
11. Ge SS, Li Y, Zhang J, Lee TH (2004) Direct adaptive control for a class of MIMO nonlinear systems using neural networks. *IEEE Trans Autom Control* 49(11):2001–2006
12. Ge SS, Zhang J, Lee TH (2004) Adaptive MNN control for a class of non-affine NARMAX systems with disturbances. *Syst Control Lett* 53:1–12
13. Ge SS, Lee TH, Li GY, Zhang J (2003) Adaptive NN control for a class of discrete-time nonlinear systems. *Int J Control* 76(4):334–354
14. Lee S (2001) Neural network based adaptive control and its applications to aerial vehicles. Ph. D. dissertation, School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA
15. Shin DH, Kim Y (2006) Nonlinear discrete-time reconfigurable flight control law using neural networks. *IEEE Trans Control Syst Technol* 14(3):408–422
16. Zhang J, Ge SS, Lee TH (2002) Direct RBF neural network control of a class of discrete-time non-affine nonlinear systems. In: Proceedings of the American control conference, Anchorage, Alaska, USA, pp 424–429
17. Fabri SG, Kadirkamanathan V (2001) Functional adaptive control: an intelligent systems approach. Springer, New York

Chapter 11

Adaptive RBF Observer Design and Sliding Mode Control

Abstract This chapter introduces a kind of adaptive observer with RBF neural network approximation. Using this observer, a speedless sliding mode controller is designed. Stability analysis of the observer and the closed control system are presented. Simulation examples for single-link manipulator are given.

Keywords RBF neural network • Observer • Sliding mode control • Neural network approximation

11.1 Adaptive RBF Observer Design

With neural network observer, speedless adaptive controller can be realized without modeling information. Several works have been published about neural network observer and control [1–4].

An adaptive observer for a class of single-input single-output (SISO) nonlinear systems is proposed by [1] with a dynamic recurrent neural network. The neural network weights are tuned online. No exact knowledge of nonlinearities in the observed system is required.

In this section, we use RBF neural network to realize the adaptive observer Matlab simulation.

11.1.1 System Description

Consider a second order SISO nonlinear system as

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{b}[f(\mathbf{x}) + g(\mathbf{x})u + d(t)] \\ y &= \mathbf{C}^T\mathbf{x}\end{aligned}\tag{11.1}$$

where $\mathbf{x} = [x_1 \ x_2]^T$, $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$, $\mathbf{b} = [0 \ 1]^T$, $\mathbf{C} = [1 \ 0]^T$, $y \in R$, $u \in R$, $d(t)$ is disturbance, and $|d(t)| \leq b_d$, $f(\mathbf{x})$, and $g(\mathbf{x})$ are unknown nonlinear functions.

11.1.2 Adaptive RBF Observer Design and Analysis

The adaptive RBF observer for the system (11.1) was given by [1] as

$$\begin{aligned}\dot{\hat{\mathbf{x}}} &= \mathbf{A}\hat{\mathbf{x}} + \mathbf{b}[\hat{f}(\hat{\mathbf{x}}) + \hat{g}(\hat{\mathbf{x}})u - v(t)] + \mathbf{K}(y - \mathbf{C}^T\hat{\mathbf{x}}) \\ \hat{y} &= \mathbf{C}^T\hat{\mathbf{x}}\end{aligned}\quad (11.2)$$

Where $\hat{\mathbf{x}}$ is observer value of \mathbf{x} , \mathbf{K} is the gain vector, $\mathbf{K} = [k_1 \ k_2]^T$, $\hat{f}(\hat{\mathbf{x}})$ and $\hat{g}(\hat{\mathbf{x}})$ are estimation of $f(\mathbf{x})$ and $g(\mathbf{x})$, and $v(t)$ is robust term.

In the observer (11.2), $f(\mathbf{x})$ and $g(\mathbf{x})$ were estimated by using neural network. The continuous unknown nonlinear functions in the system can be represented by RBF with constant “ideal” weights \mathbf{W}^* and a sufficient number of basis functions $\mathbf{h}(\mathbf{x})$, that is,

$$\begin{aligned}f(\mathbf{x}) &= \mathbf{W}_1^{*T}\mathbf{h}_1(\mathbf{x}) + \epsilon_1(\mathbf{x}), \quad \epsilon_1(\mathbf{x}) \leq \epsilon_{1,N} \\ g(\mathbf{x}) &= \mathbf{W}_2^{*T}\mathbf{h}_2(\mathbf{x}) + \epsilon_2(\mathbf{x}), \quad \epsilon_2(\mathbf{x}) \leq \epsilon_{2,N}\end{aligned}\quad (11.3)$$

where $\epsilon_1(\mathbf{x})$ and $\epsilon_2(\mathbf{x})$ are the neural-net reconstruction errors.

It is assumed that the ideal weights \mathbf{W}_1^* and \mathbf{W}_2^* are bounded by known values as

$$\|\mathbf{W}^*\|_{i,F} \leq W_{i,M}, \quad i = 1, 2. \quad (11.4)$$

Let the NN functional estimates for $f(\mathbf{x})$ and $g(\mathbf{x})$ be given by

$$\hat{f}(\hat{\mathbf{x}}) = \hat{\mathbf{W}}_1^T\mathbf{h}_1(\hat{\mathbf{x}}), \quad \hat{g}(\hat{\mathbf{x}}) = \hat{\mathbf{W}}_2^T\mathbf{h}_2(\hat{\mathbf{x}}) \quad (11.5)$$

where $\hat{\mathbf{W}}_1$ and $\hat{\mathbf{W}}_2$ are estimated weight value, $\tilde{\mathbf{W}}_i = \mathbf{W}_i^* - \hat{\mathbf{W}}_i (i = 1, 2)$.

The following theorem was given in [1] for the adaptive RBF observer stability proof.

Theorem 11.1. Suppose the control input $u(t)$ is bounded by a positive constant as $|u(t)| \leq u_d$. Design the observer as

$$\begin{aligned}\dot{\hat{\mathbf{x}}} &= \mathbf{A}\hat{\mathbf{x}} + \mathbf{b}[\hat{\mathbf{W}}_1^T\hat{\mathbf{h}}_1 + \hat{\mathbf{W}}_2^T\hat{\mathbf{h}}_2u - v_1 - v_2] + \mathbf{K}(y - \mathbf{C}\hat{\mathbf{x}}) \\ \hat{y} &= \mathbf{C}^T\hat{\mathbf{x}}\end{aligned}\quad (11.6)$$

where the robust terms are given by

$$v_i(t) = -D_i \frac{\tilde{y}}{|\tilde{y}|}, \quad i = 1, 2 \quad (11.7)$$

where $D_1 \geq \beta_1 \sigma_M$, $D_2 \geq \beta_2 \sigma_M u_d$, $\sigma_M = \sigma_{\max}[L^{-1}(s)]$, $\sigma_{\max}[\cdot]$ is the maximum singular value. $L^{-1}(s)$ is a proper transfer function with stable poles, and $L(s)$ is chosen so that $H(s)L(s)$ is SPR.

The adaptive laws of RBF are designed as

$$\begin{aligned} \dot{\hat{W}}_1 &= \mathbf{F}_1 \hat{\mathbf{h}}_1 \tilde{y} - \kappa_1 \mathbf{F}_1 |\tilde{y}| \hat{W}_1 \\ \dot{\hat{W}}_2 &= \mathbf{F}_2 \hat{\mathbf{h}}_2 \tilde{y} u - \kappa_2 \mathbf{F}_2 |\tilde{y}| \hat{W}_2 \end{aligned} \quad (11.8)$$

where $\mathbf{F}_i = \mathbf{F}_i^T > 0$, $\kappa_i > 0$, $i = 1, 2$.

Then the state estimation error $\tilde{x}(t)$ and the NN weight estimation errors $\tilde{W}_1(t)$ and $\tilde{W}_2(t)$ are UUB.

The stability proof of the theorem has been given in the paper [1]. Now we complement the convergence analysis of \tilde{x} as follows.

Remark 1.

The analysis is divided into two steps.

Step 1 The solution of \tilde{x}

The Eq. (11.6) can be written as

$$\dot{\tilde{x}} = (\mathbf{A} - \mathbf{K}\mathbf{C}^T)\tilde{x} + \mathbf{b}\tilde{u}. \quad (11.9)$$

The solution of $\dot{\tilde{x}} = (\mathbf{A} - \mathbf{K}\mathbf{C}^T)\tilde{x}$ is

$$\tilde{x}(t) = \tilde{x}(0) e^{\int_0^t (\mathbf{A} - \mathbf{K}\mathbf{C}^T) dt}.$$

Then the solution of (11.9) is

$$\tilde{x}(t) = \tilde{x}(0) e^{\int_0^t (\mathbf{A} - \mathbf{K}\mathbf{C}^T) dt} + e^{\int_0^t (\mathbf{A} - \mathbf{K}\mathbf{C}^T) dt} \int_0^t b\tilde{u}(\tau) e^{-\int_0^\tau (\mathbf{A} - \mathbf{K}\mathbf{C}^T) d\tau} d\tau$$

Choose $\Phi(t, 0) = e^{\int_0^t (\mathbf{A} - \mathbf{K}\mathbf{C}^T) dt}$, $\Phi(t, \tau) = e^{\int_0^\tau (\mathbf{A} - \mathbf{K}\mathbf{C}^T) d\tau - \int_0^t (\mathbf{A} - \mathbf{K}\mathbf{C}^T) d\tau}$, then above equation becomes

$$\tilde{x}(t) = \Phi(t, 0) \tilde{x}(0) + \int_0^t \Phi(t, \tau) b\tilde{u}(\tau) d\tau. \quad (11.10)$$

Since

$$\begin{aligned} e^{\int_0^t (A - KC^T) dt - \int_0^\tau (A - KC^T) d\tau} &= e^{(A - KC^T)} e^{(t-\tau)} = e^{A(t-\tau)} \times e^{-KC^T(t-\tau)} \\ &= m_0 e^{-\alpha(t-\tau)} \end{aligned}$$

where $m_0 = e^{A(t-\tau)}$, $\alpha = KC^T$, then $\Phi(t, \tau)$ is bounded by $m_0 e^{-\alpha(t-\tau)}$, with m_0 and α positive constants.

Step 2 Convergence analysis of \tilde{x}

From Lemma 2 [1] and (11.9) yields

$$\|\tilde{x}(t)\| \leq k_1 + k_2 \|\tilde{u}\|_2^\alpha, \quad \forall t \geq 0 \quad (11.11)$$

where $\|\tilde{u}\|_2^\alpha = \|\tilde{W}_1^T \hat{h}_1 + w_1 + \epsilon_1 + [\tilde{W}_2^T \hat{h}_2 + w_2 + \epsilon_2]u + d + v_1 + v_2\|_2^\alpha$.

Define $c = w_1 + \epsilon_1 + [w_2 + \epsilon_2]u + d + v_1 + v_2$, then

$$\|\tilde{u}\|_2^\alpha = \|\tilde{W}_1^T \hat{h}_1 + \tilde{W}_2^T \hat{h}_2 u + c\|_2^\alpha \leq \|\tilde{W}_1^T \hat{h}_1\|_2^\alpha + \|\tilde{W}_2^T \hat{h}_2 u\|_2^\alpha + c'{}_4$$

where $\|c\|_2^\alpha \leq c'{}_4$.

Since $\|Ax\|_2 \leq \|A\|_{\text{F}} \|x\|_2$ and $\|x\|_2^\alpha = \sqrt{\int_0^t e^{-\alpha(t-\tau)} x^T(\tau) x(\tau) d\tau}$, then

$$\begin{aligned} \|\tilde{W}_1^T \hat{h}_1\|_2^\alpha &\leq \|\tilde{W}_1^T\|_{\text{F}}^\alpha \|\hat{h}_1\|_2^\alpha = \|\tilde{W}_1^T\|_{\text{F}}^\alpha \sqrt{\int_0^t e^{-\alpha(t-\tau)} \hat{h}_1 \hat{h}_1^T d\tau} \\ &= \|\tilde{W}_1^T\|_{\text{F}}^\alpha \|\hat{h}_1\| \sqrt{\int_0^t e^{-\alpha(t-\tau)} d\tau} \\ &= \|\tilde{W}_1^T\|_{\text{F}}^\alpha \|\hat{h}_1\| \frac{1}{\sqrt{\alpha}} \sqrt{\int_0^t e^{-\alpha(t-\tau)} d(-\alpha(t-\tau))} \\ &= \|\tilde{W}_1^T\|_{\text{F}}^\alpha \|\hat{h}_1\| \frac{1}{\sqrt{\alpha}} \sqrt{1 - e^{-\alpha t}} \leq \|\tilde{W}_1^T\|_{\text{F}}^\alpha \frac{1}{\sqrt{\alpha}} c'{}_5 \end{aligned}$$

Similarly,

$$\|\tilde{W}_2^T \hat{h}_2 u\|_2^\alpha \leq \|\tilde{W}_2^T\|_{\text{F}}^\alpha \frac{1}{\sqrt{\alpha}} c'{}_6$$

where $c'{}_5 = \|\hat{h}_1\| \sqrt{1 - e^{-\alpha t}}$, $c'{}_6 = \|\hat{h}_2\| \sqrt{1 - e^{-\alpha t}} u_d$.

Then

$$\|\tilde{u}\|_2^\alpha \leq \|\tilde{W}_1^T\|_{\text{F}}^\alpha \frac{1}{\sqrt{\alpha}} c'{}_5 + \|\tilde{W}_2^T\|_{\text{F}}^\alpha \frac{1}{\sqrt{\alpha}} c'{}_6 + c'{}_4. \quad (11.12)$$

Substituting (11.12) into (11.11) yields

$$\begin{aligned}\|\tilde{\mathbf{x}}(t)\| &\leq k_1 + k_2 \left(\|\tilde{\mathbf{W}}_1^T\|_{\text{F}}^{\alpha} \frac{1}{\sqrt{\alpha}} c'_5 + \|\tilde{\mathbf{W}}_2^T\|_{\text{F}}^{\alpha} \frac{1}{\sqrt{\alpha}} c'_6 + c'_4 \right) \\ &= c_3 + \left(c_4 + c_5 \|\tilde{\mathbf{W}}_1\|_{\text{F}}^{\alpha} + c_6 \|\tilde{\mathbf{W}}_2\|_{\text{F}}^{\alpha} \right) \frac{1}{\sqrt{\alpha}}\end{aligned}$$

where $c_3 = k_1$, $c_4 = k_2 c'_4$, $c_5 = k_2 c'_5$, $c_6 = k_2 c'_6$, c_4, c_5 , and c_6 are positive constants.

From Lemma 2 in the paper [1], c_3 is a term decaying exponentially to zero owing to the initial conditions.

11.1.3 Simulation Examples

Two examples are given for the same nonlinear SISO system as

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{b}[f(\mathbf{x}) + g(\mathbf{x})u + d(t)] \\ y &= \mathbf{C}^T \mathbf{x}\end{aligned}\tag{11.13}$$

where $\mathbf{x} = [x_1 \quad x_2]^T$, $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$, $\mathbf{b} = [0 \quad 1]^T$, $\mathbf{C} = [1 \quad 0]^T$, $d(t) = 0$.

11.1.3.1 First Example

Using the model in paper [1], consider a single-link robot equation as

$$Mq + \frac{1}{2}mglsin q = u, \quad y = q$$

where q is the angle, u is the control input, M is the moment of inertia, g is the acceleration, and m and l are the mass and the length of the link.

Letting $x_1 = q$, $x_2 = \dot{q}$ yields the state-space equation as

$$\begin{aligned}[\dot{x}_1 \dot{x}_2] &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} [x_1 x_2] + [0 1] \left(-\frac{1}{2} \frac{mglsin x_1}{M} + \frac{1}{M} u \right) \\ y &= x_1\end{aligned}$$

From (11.13), we can get $f(\mathbf{x}) = -\frac{1}{2} \frac{mglsin x_1}{M}$ and $g(\mathbf{x}) = \frac{1}{M}$.

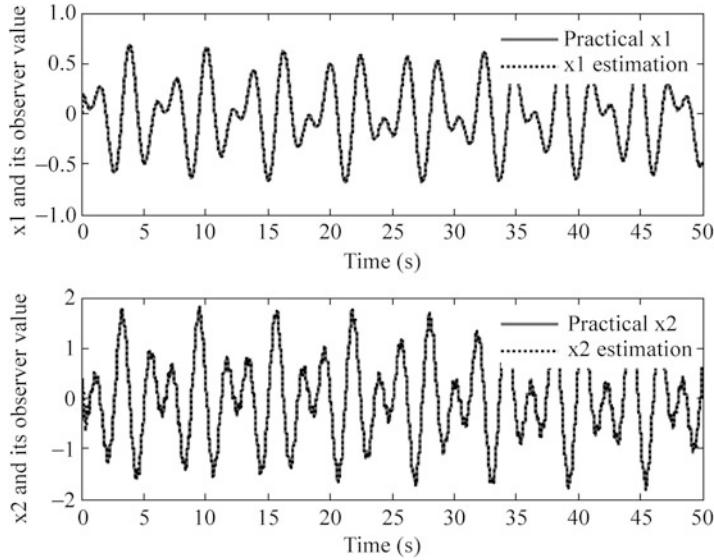


Fig. 11.1 States estimation of x_1 and x_2

The input vector of RBF is $[\hat{x}_1 \quad \hat{x}_2]^T$, and the network structure 2-7-1 is used. In this example, according to the practical scope of x_1 and x_2 , for each Gaussian function, the parameters are designed as $\mathbf{c}_1 = \frac{1}{3}[-3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3]$, $\mathbf{c}_2 = \frac{2}{3}[-3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3]$, and $b_j = 5.0$, $j = 1, \dots, 7$.

In the simulation, the robot parameters are $m = 1$, $l = 1$, $M = 0.5$, $g = 9.8$; choose $L^{-1}(s) = \frac{1}{s+3}$, $\mathbf{K} = [400 \quad 800]$, $\mathbf{F}_1 = \text{diag}[5 \times 10^5]$, $\mathbf{F}_2 = \text{diag}[5 \times 10^4]$, $\kappa_1 = \kappa_2 = 0.001$, $D = 0.8$, $\mathbf{x}(0) = [0 \quad 0.5]^T$, $\dot{\mathbf{x}}(0) = [0.1 \quad 0]^T$, $u(t) = \sin(2t) + \cos(20t)$. The adaptive observer (11.6) and (11.7) are used; the initial weight value is chosen as zero. The results are given in Figs. 11.1 and 11.2.

In the simulation, we give two remarks as follows:

Remark 2. $H(s)$ is strictly positive real (SPR), $s = \sigma + j\omega$, if three conditions as follows are satisfied:

1. When s is real, $H(s)$ is real.
2. The poles of $H(s)$ is not in the right half plane.
3. For any real ω , the real part of $H(j\omega)$ is positive, that is, $\text{Re}[H(j\omega)] \geq 0$.

In the simulation, from $H(s)$ expression in (11.7), we have $H(s) = \mathbf{C}^T(s\mathbf{I} - (\mathbf{A} - \mathbf{KC}^T))^{-1}\mathbf{b} = \frac{1}{s^2 + 400s + 800}$. Obviously, when s is real, $H(s)$ is real; the poles of $H(s)$ are -397.99 and -2.01 , which are in the left part of; but $\text{Re}[H(j\omega)] = \frac{2400 - 403\omega^2}{(2400 - 403\omega^2)^2 + (2000\omega - \omega^3)^2}$ may be negative. Therefore, $H(s)$ is not SPR.

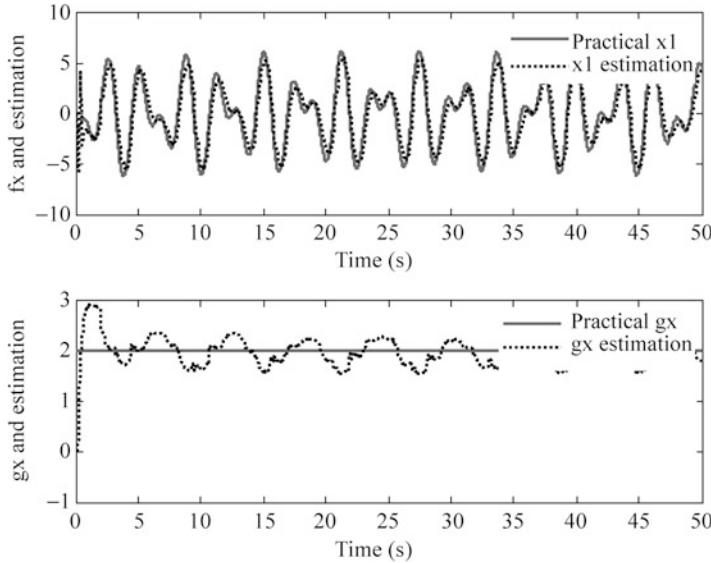


Fig. 11.2 Estimation of $f(\mathbf{x})$ and $g(\mathbf{x})$

In order to use Lemma 1 [1] in the stability analysis, $L(s)$ is chosen so that $H(s)L(s)$ is SPR. Choose $L^{-1}(s) = \frac{1}{s+3}$, then $\text{sys}(s) = H(s)L(s) = \frac{s+3}{s^2 + 400s + 800}$; the poles of $\text{sys}(s)$ are negative. The real part of $\text{sys}(j\omega)$ is $\text{Re}[\text{sys}(j\omega)] = \frac{397\omega^2 + 2400}{(800 - \omega^2)^2 + 400^2}$, which is positive. Then $H(s)L(s)$ is SPR.

Remark 3. To solve in the adaptive law (11.8), from $\hat{\mathbf{h}} = L^{-1}(s)\hat{\mathbf{h}}$, then $\hat{\mathbf{h}} = \frac{1}{s+3}\hat{\mathbf{h}}$, $\hat{\mathbf{h}} = \hat{\mathbf{h}} - 3\hat{\mathbf{h}}$, which can be regarded as an adaptive law and realized by combining with (11.8) in the simulation.

The programs of this example are chap11_1.m and chap11_2sim.mdl; the Matlab programs of the example are given in the [Appendix](#).

11.1.3.2 Second Example

Consider a single-link inverted pendulum equation as

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{g \sin x_1 - mlx_2^2 \cos x_1 \sin x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))} + \frac{\cos x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))} u\end{aligned}$$

where x_1 and x_2 are angle and speed signal, respectively; $g = 9.8 \text{ m/s}^2$; the mass of cart is $m_c = 1 \text{ kg}$; the mass of the pendulum is $m = 0.1 \text{ kg}$; the length of the pendulum is $l = 0.5 \text{ m}$; and u is the control input.

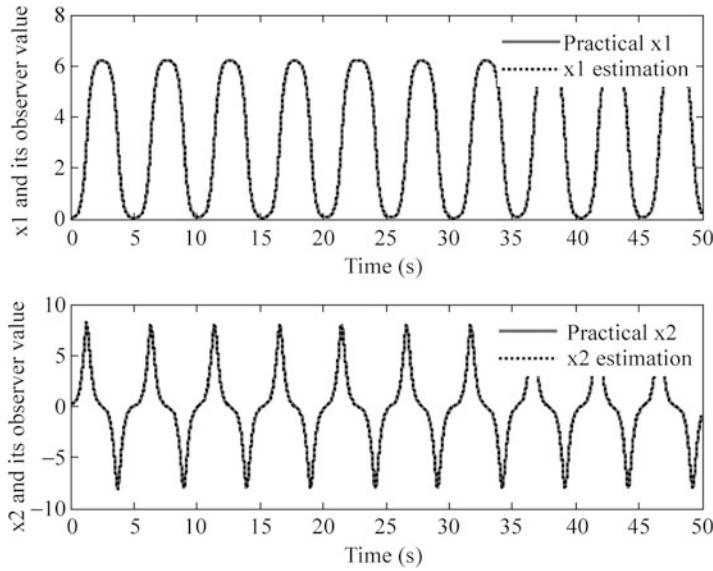


Fig. 11.3 States estimation of x_1 and x_2

The above equation yields the state-space equation as

$$\begin{bmatrix} \dot{x}_1 & \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} [x_1 x_2] + [01](f(\mathbf{x}) + g(\mathbf{x})u). \\ y = x_1$$

From (11.13), we can get $f(\mathbf{x}) = \frac{g \sin x_1 - mlx_2^2 \cos x_1 \sin x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}$ and $g(x) = \frac{\cos x_1 / (m_c + m)}{l(4/3 - m \cos^2 x_1 / (m_c + m))}$.

The input vector of RBF is $[\hat{x}_1 \quad \hat{x}_2]^T$, and the network structure 2-7-1 is used. In this example, according to the practical scope of x_1 and x_2 , for each Gaussian function, the parameters are designed as $\mathbf{c}_1 = \frac{6}{3}[-3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3]$, $\mathbf{c}_2 = \frac{8}{3}[-3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3]$ and $b_j = 5.0$, $j = 1, \dots, 7$.

In the simulation, choose $L^{-1}(s) = \frac{1}{s + 0.5}$, $\mathbf{K} = [1000 \quad 1000]$, $\mathbf{F}_1 = \text{diag}[15 \times 10^4]$, $\mathbf{F}_2 = \text{diag}[15 \times 10^4]$, $\kappa_1 = \kappa_2 = 0.001$, $D = 1.5$, $\mathbf{x}(0) = [-\pi/60, 0]^T$, $\dot{\mathbf{x}}(0) = [0 \ 0]^T$, $u(t) = 0.01 \sin(2t) + 0.01 \cos(20t)$. Using remark 2, $H(s)L(s)$ is SPR. The adaptive observer (11.6) and (11.7) are used; the initial weight value is chosen as zero. The results are given in Figs. 11.3 and 11.4.

In the two above examples, the variation of $\hat{g}(\cdot)$ do not converge to $g(\cdot)$. This is due to the fact that the input signal is not persistently exciting, and the observer error convergence can be achieved by many possible values of $\hat{g}(\cdot)$, besides the true $g(\cdot)$ value, which has been explained in Sect. 1.5.

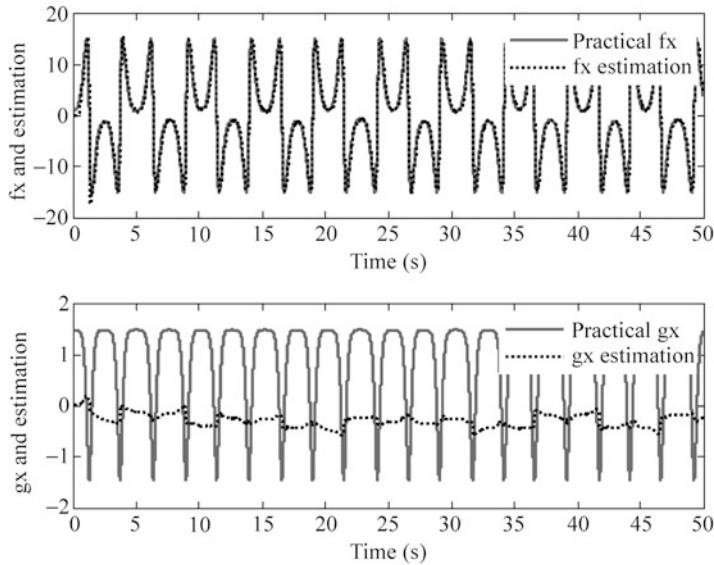


Fig. 11.4 Estimation of $f(\mathbf{x})$ and $g(\mathbf{x})$

The Simulink program of this example is `chap11_3sim.mdl`; the Matlab programs of the example are given in the [Appendix](#).

11.2 Sliding Mode Control Based on RBF Adaptive Observer

With neural network observer designed in Sect. 11.1, we can design speedless adaptive controller without modeling information. Figure 11.5 shows the closed-loop neural-based sliding mode control scheme with RBF observer.

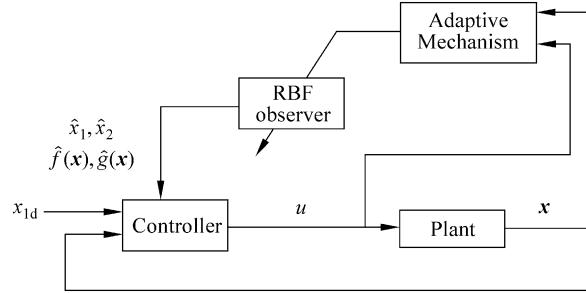
11.2.1 Sliding Mode Controller Design

Consider a second-order nonlinear SISO system as

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{b}[f(\mathbf{x}) + g(\mathbf{x})u + d(t)] \\ y &= \mathbf{C}^T\mathbf{x}\end{aligned}\tag{11.14}$$

where $\mathbf{x} = [x_1 \quad x_2]^T$, $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$, and $\mathbf{b} = [0 \quad 1]^T$, $\mathbf{C} = [1 \quad 0]^T$, $d(t) = 0$.

Fig. 11.5 Block diagram of the control scheme with RBF observer



The tracking error and its derivative value are $e = \hat{x}_1 - x_{1d}$ and $\dot{e} = \hat{x}_2 - \dot{x}_{1d}$, where x_{1d} is ideal position signal.

Firstly, we design the sliding mode function as

$$s(t) = ce(t) + \dot{e}(t) \quad (11.15)$$

where c must satisfy Hurwitz condition, $c > 0$.

Design Lyapunov function as

$$V_c = \frac{1}{2}s^2. \quad (11.16)$$

Therefore, we have

$$\dot{s}(t) = c(\dot{\hat{x}}_1 - \dot{x}_{1d}) + (\dot{\hat{x}}_2 - \ddot{x}_{1d}). \quad (11.17)$$

From the observer (11.6), we get

$$\dot{\hat{x}}_1 = \hat{x}_2 + K_1(x_1 - \hat{x}_1) \quad (11.18a)$$

$$\dot{\hat{x}}_2 = \hat{f}(\mathbf{x}) + \hat{g}(\mathbf{x})u - v(t) + K_2(x_1 - \hat{x}_1). \quad (11.18b)$$

Substituting (11.18a) and (11.18b) into (11.17), we can get

$$\dot{s}(t) = c((\hat{x}_2 + K_1(x_1 - \hat{x}_1)) - \dot{x}_{1d}) + \hat{f}(\mathbf{x}) + \hat{g}(\mathbf{x})u - v(t) + K_2(x_1 - \hat{x}_1) - \ddot{x}_{1d}.$$

To guarantee $\dot{V}_c = ss < 0$, we design the sliding mode controller as

$$u = \frac{1}{\hat{g}}(-c(\hat{x}_2 + K_1(x_1 - \hat{x}_1) - \dot{x}_{1d}) - \hat{f} + v(t) - K_2(x_1 - \hat{x}_1) + \ddot{x}_{1d} - \eta \text{sgn}(s)) \quad (11.19)$$

where $\eta > 0$.

Then we get

$$s\dot{s} = -\eta|s| < 0.$$

Thus

$$\dot{V} \leq 0 (\dot{V} = 0 \text{ when } s = 0).$$

Define Lyapunov function as

$$V = V_o + V_c.$$

Then $\dot{V} = \dot{V}_o + V_c \leq 0$.

11.2.2 Simulation Example

Using the model in paper [1], consider a single-link robot equation as

$$Mq + \frac{1}{2}mgl \sin q = u, \quad y = q$$

where q is the angle, u is the control input, M is the moment of inertia, g is the acceleration, and m and l are the mass and the length of the link.

Letting $x_1 = q$, $x_2 = \dot{q}$ yields the state-space equation as

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \left(-\frac{1}{2} \frac{mgl \sin x_1}{M} + \frac{1}{M} u \right)$$

$$y = x_1$$

From (11.13), we can get $f(\mathbf{x}) = -\frac{1}{2} \frac{mgl \sin x_1}{M}$ and $g(\mathbf{x}) = \frac{1}{M}$.

The input vector of RBF is $[\hat{x}_1 \quad \hat{x}_2]^T$, and the network structure 2-7-1 is used. In this example, according to the practical scope of x_1 and x_2 , for each Gaussian function, the parameters are designed as $\mathbf{c}_1 = \frac{1}{3}[-3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3]$, $\mathbf{c}_2 = \frac{1}{3}[-3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3]$, and $b_j = 5.0$, $j = 1, \dots, 7$. The initial weight value is chosen as zero.

In the simulation, the robot parameters are $m = 1$, $l = 1$, $M = 0.5$, $g = 9.8$. The adaptive observers (11.18a) and (11.18b) are used; choose $L^{-1}(s) = \frac{1}{s+0.5}$, $\mathbf{K} = [400 \quad 800]$, $\mathbf{F}_1 = \text{diag}[500]$, $\mathbf{F}_2 = \text{diag}[0.5]$, $\kappa_1 = \kappa_2 = 0.001$, $D = 0.8$, $\mathbf{x}(0) = [0.2 \quad 0]^T$, $\hat{\mathbf{x}}(0) = [0.1 \quad 0]^T$.

The ideal position signal is $x_d(t) = \sin(t)$, and the controller is (11.19); choose $c = 20$, $\eta = 0.10$. The results are shown from Figs. 11.6, 11.7, 11.8, and 11.9.

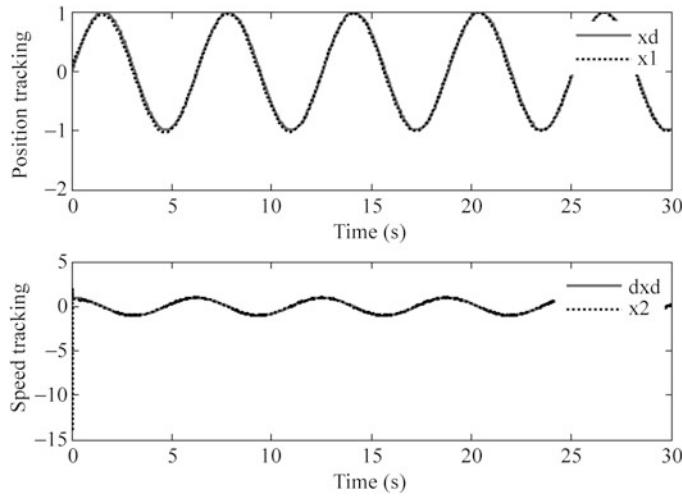


Fig. 11.6 Position and speed tracking

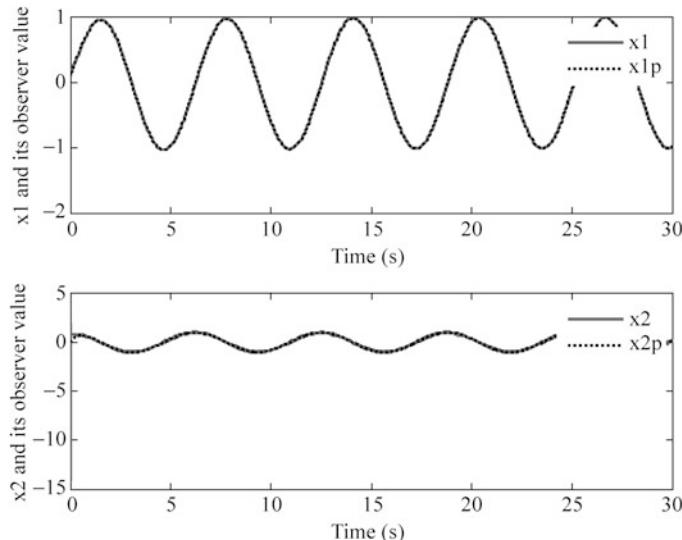


Fig. 11.7 States estimation of x_1 and x_2

In the two above examples, the variation of $\hat{f}(\cdot)$ and $\hat{g}(\cdot)$ do not converge to $f(\cdot)$ and $g(\cdot)$. This is due to the fact that the input signal is not persistently exciting and the observer error convergence can be achieved by many possible values of $\hat{f}(\cdot)$ and $\hat{g}(\cdot)$, besides the true $f(\cdot)$ and $g(\cdot)$ value, which has been explained in Sect. 1.5.

The Simulink program of this example is chap11_4sim.mdl; the Matlab programs of the example are given in the [Appendix](#).

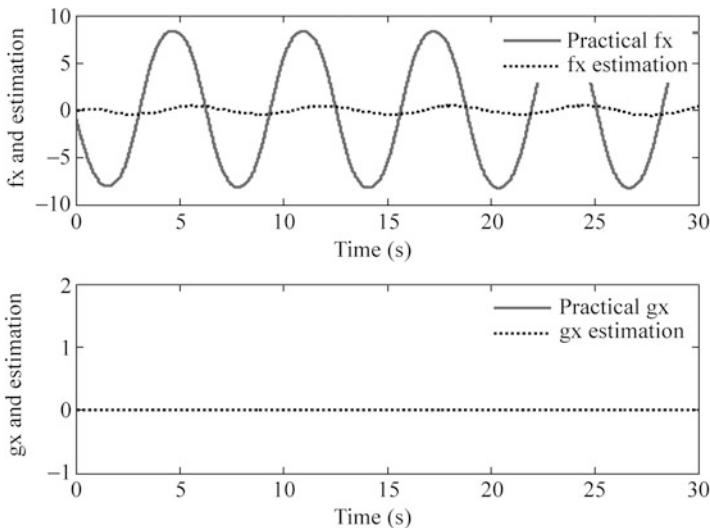


Fig. 11.8 Estimation of $f(x)$ and $g(x)$

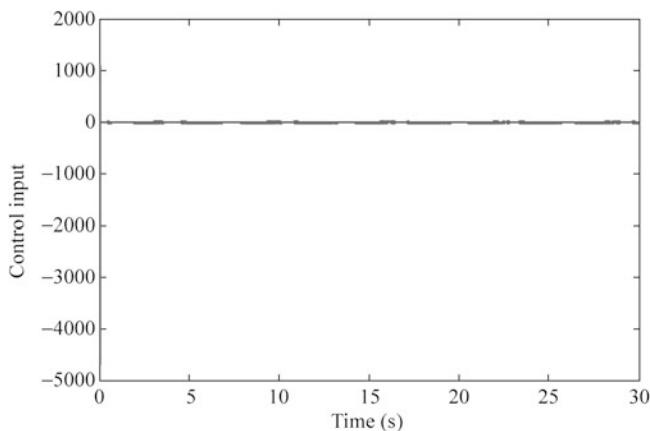


Fig. 11.9 Control input

Appendix

Programs for Sect. 11.1.3.1

The program for SPR testing of $H(s)$: chap11_1.m

```
%System Analysis
clear all;
close all;

A=[0 1; 0 0];
K1=400; K2=800;
```

```

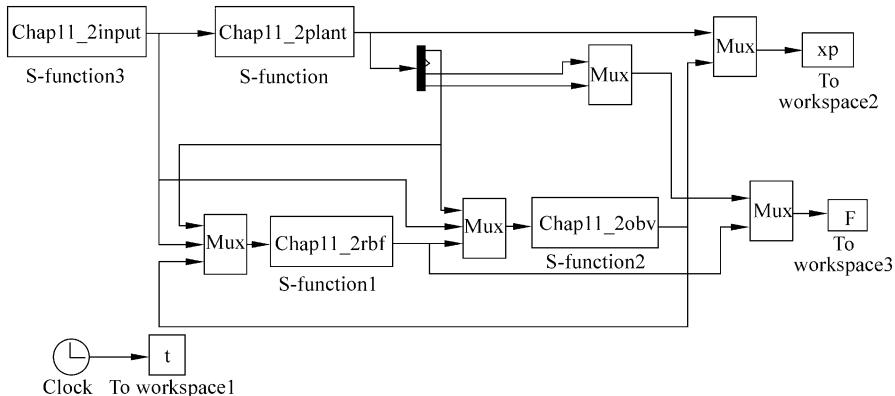
K=[K1 K2]';
b=[0 1]';
C=[1 0]';

%For dot(x)=Ax+Bu, y=Cx, see >help ss2tf
A=A-K*C';
B=b; C=C'; D=0;

[num,den]=ss2tf(A,B,C,D);
H=tf(num,den) %Plant
pole(H)
L=tf(1,[1 3]) %Low filter
sys=series(H,inv(L)) %Series with Low filter

```

Main Simulink program: chap11_2sim.mdl



Input program: chap11_2input.m

```

function [sys,x0,str,ts]=obser(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 = 1;

```

```
sizes.NumInputs = 0;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[];
str=[];
ts=[];
function sys=mdlOutputs(t,x,u)
sys(1)=sin(2*t)+cos(20*t);
```

Observer program: chap11_2obv.m

```
function [sys,x0,str,ts] = obser(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 {1,2,4,9}
    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 = 4;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0=[0 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
K1=400;K2=800;
y=u(1);
ut=u(2);
fzp=u(3);
gxp=u(4);
A=[0 1;0 0];
b=[0 1]';
C=[1 0];
K=[K1 K2]';
```

```

ye=y-x(1);
D=1.50;
v=-D*sign(ye);

dx=A*x+b*(f*x+g*x*u-v)+K*(y-C*x);
sys(1)=dx(1);
sys(2)=dx(2);
function sys=mdlOutputs(t,x,u)
sys(1) = x(1);
sys(2) = x(2);

```

RBF Approximation program: chap11_2rbf.m

```

function [sys,x0,str,ts] = obser(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 {1,2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
global c b
sizes = simsizes;
sizes.NumContStates = 21;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 4;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0=zeros(1,21);
str=[];
ts=[];
c1=1/3*[-3 -2 -1 0 1 2 3];
c2=2/3*[-3 -2 -1 0 1 2 3];
c=[c1;c2];
b=5;
function sys=mdlDerivatives(t,x,u)
global c b
y=u(1);
ut=u(2);
x1p=u(3);

```

```

x2p=u(4);
xp=[x1p x2p]';
yp=x1p;
ye=y-yp;

h=zeros(7,1);
for j=1:1:7
    h(j)=exp(-norm(xp-c(:,j))^2/(2*b^2));
end
h_bar=x(15:1:21);

F1=500000*eye(7);
F2=50000*eye(7);

k1=0.001;k2=0.001;
W1=[x(1) x(2) x(3) x(4) x(5) x(6) x(7)];
W2=[x(8) x(9) x(10) x(11) x(12) x(13) x(14)];

dW1=F1*h_bar*ye-k1*F1*abs(ye)*W1';
dW2=F2*h_bar*ye*ut-k2*F2*abs(ye)*W2';
for i=1:1:7
    sys(i)=dW1(i);
    sys(i+7)=dW2(i);
end
for i=15:1:21
    sys(i)=h(i-14)-3*x(i);
end

function sys=mdlOutputs(t,x,u)
global c b
W1=[x(1) x(2) x(3) x(4) x(5) x(6) x(7)];
W2=[x(8) x(9) x(10) x(11) x(12) x(13) x(14)];
h_bar=x(15:1:21);

fxp=W1*h_bar;
gxpxp=W2*h_bar;

sys(1)=fxp;
sys(2)=gxpxp;

```

Plant program: chap11_2plant.m

```

function [sys,x0,str,ts]=obser(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 = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.2 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
m=1;l=1;M=0.5;g=9.8;
fx=-0.5*m*g*l*sin(x(1))/M;
gx=1/M;

sys(1)=x(2);
sys(2)=fx+gx*u;
function sys=mdlOutputs(t,x,u)
m=1;l=1;M=0.5;g=9.8;
fx=-0.5*m*g*l*sin(x(1))/M;
gx=1/M;

y=x(1);
sys(1)=y;
sys(2)=x(2);
sys(3)=fx;
sys(4)=gx;

```

Plot program: chap11_2plot.m

```

close all;
figure(1);
subplot(211);
plot(t,xp(:,1),'r',t,xp(:,5),'k:','linewidth',2);
xlabel('time');ylabel('x1 and its observer value');
legend('Practical x1','x1 estimation');
subplot(212);
plot(t,xp(:,2),'r',t,xp(:,6),'k:','linewidth',2);
xlabel('time');ylabel('x2 and its observer value');
legend('Practical x2','x2 estimation');

```

```

figure(2);
subplot(211);
plot(t,F(:,1),'r',t,F(:,3),'k:','linewidth',2);
xlabel('time');ylabel('fx and estimation');
legend('Practical fx','fx estimation');
subplot(212);
plot(t,F(:,2),'r',t,F(:,4),'k:','linewidth',2);
xlabel('time');ylabel('gx and estimation');
legend('Practical gx','gx estimation');

```

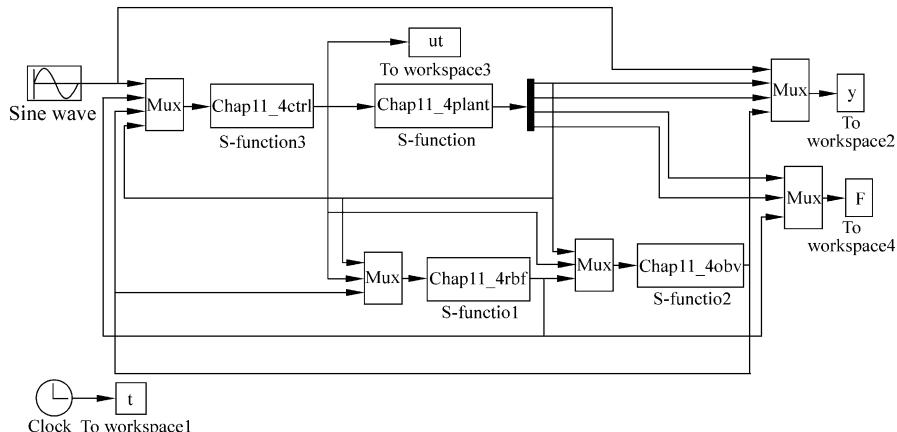
Programs for Sect. 11.1.3.2

Main Simulink program: chap11_3sim.mdl

Input program: chap11_3input.m
 Observer program: chap11_3obv.m
 RBF Approximation program: chap11_3rbf.m
 Plant program: chap11_3plant.m
 Plot program: chap11_3plot.m

Programs for Sect. 11.2.2

Main Simulink program: chap11_4sim.mdl



Observer program: chap11_4obv.m

```

function [sys,x0,str,ts] = obser(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 {1,2,4,9}
    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 = 4;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0=[0.1 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
K1=400;K2=800;
y=u(1);
ut=u(2);
fxp=u(3);
gxp=u(4);

A=[0 1;0 0];
b=[0 1]';
C=[1 0];
K=[K1 K2]';

ye=y-x(1);
D=0.8;
v=-D*sign(ye);

dx=A*x+b*(fxp+gxp*ut-v)+K*(y-C*x);
sys(1)=dx(1);
sys(2)=dx(2);
function sys=mdlOutputs(t,x,u)
sys(1) = x(1);
sys(2) = x(2);

```

RBF Approximation program: chap11_4rbf.m

```

function [sys,x0,str,ts] = obser(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 {1,2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
global c b
sizes=simsizes;
sizes.NumContStates=21;
sizes.NumDiscStates=0;
sizes.NumOutputs=2;
sizes.NumInputs=4;
sizes.DirFeedthrough=1;
sizes.NumSampleTimes=0;
sys=simsizes(sizes);
x0=zeros(1,21);
str=[];
ts=[];
c1=1/3*[-3 -2 -1 0 1 2 3];
c2=1/3*[-3 -2 -1 0 1 2 3];
c=[c1;c2];
b=5;
function sys=mdlDerivatives(t,x,u)
global c b
y=u(1);
ut=u(2);
x1p=u(3);
x2p=u(4);
xp=[x1p x2p]';
yp=x1p;
ye=y-yp;

h=zeros(7,1);
for j=1:1:7
    h(j)=exp(-norm(xp-c(:,j))^2/(2*b^2));
end
h_bar=x(15:1:21);

F1=500*eye(7);
F2=0.50*eye(7);

k1=0.01;k2=0.01;
W1=[x(1) x(2) x(3) x(4) x(5) x(6) x(7)];
W2=[x(8) x(9) x(10) x(11) x(12) x(13) x(14)];
```

```

dW1=F1*h_bar*ye-k1*F1*abs(ye)*W1';
dW2=F2*h_bar*ye*ut-k2*F2*abs(ye)*W2';
for i=1:1:7
    sys(i)=dW1(i);
    sys(i+7)=dW2(i);
end
for i=15:1:21
    sys(i)=h(i-14)-0.5*x(i);
end

function sys=mdlOutputs(t,x,u)
global c b
W1=[x(1) x(2) x(3) x(4) x(5) x(6) x(7)];
W2=[x(8) x(9) x(10) x(11) x(12) x(13) x(14)];
h_bar=x(15:1:21);

fxp=W1*h_bar;
gxp=W2*h_bar;

sys(1)=fxp;
sys(2)=gxp;

```

Plant program: chap11_4plant.m

```

function [sys,x0,str,ts]=obser(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 = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.2 0];

```

```
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
m=1;l=1;M=0.5;g=9.8;
fx=-0.5*m*g*l*sin(x(1))/M;
gx=1/M;

sys(1)=x(2);
sys(2)=fx+gx*u;
function sys=mdlOutputs(t,x,u)
m=1;l=1;M=0.5;g=9.8;
fx=-0.5*m*g*l*sin(x(1))/M;
gx=1/M;

y=x(1);
sys(1)=y;
sys(2)=x(2);
sys(3)=fx;
sys(4)=gx;
```

Plot program: chap11_4plot.m

```
close all;

figure(1);
subplot(211);
plot(t,y(:,1),'r',t,y(:,2),'k:','linewidth',2);
xlabel('time');ylabel('Position tracking');
legend('xd','x1');
subplot(212);
plot(t,cos(t),'r',t,y(:,3),'k:','linewidth',2);
xlabel('time');ylabel('Speed tracking');
legend('dxd','x2');

figure(2);
subplot(211);
plot(t,y(:,2),'r',t,y(:,4),'k:','linewidth',2);
xlabel('time');ylabel('x1 and its observer value');
legend('x1','x1p');
subplot(212);
plot(t,y(:,3),'r',t,y(:,5),'k:','linewidth',2);
xlabel('time');ylabel('x2 and its observer value');
legend('x2','x2p');

figure(3);
subplot(211);
plot(t,F(:,1),'r',t,F(:,3),'k:','linewidth',2);
xlabel('time');ylabel('fx and estimation');
```

```
legend('Practical fx','fx estimation');
subplot(212);
plot(t,F(:,2),'r',t,F(:,4),'k:','linewidth',2);
xlabel('time');ylabel('gx and estimation');
legend('Practical gx','gx estimation');

figure(4);
plot(t,ut,'r','linewidth',2);
xlabel('time');ylabel('Control input');
```

References

1. Young HK, Lewis FL, Chaouki TA (1997) A dynamic recurrent neural network based adaptive observer for a class of nonlinear systems. *Automatica* 33(8):1539–1543
2. Huang SN, Tan KK, Lee TH (2005) Further result on a dynamic recurrent neural network based adaptive observer for a class of nonlinear systems. *Automatica* 41:2161–2162
3. Young HK, Lewis FL (1999) Neural network output feedback control of robot manipulators. *IEEE Trans Robot Automat* 15(2):301–309
4. Abdollahi F, Talebi HA, Patel RV (2006) A stable neural network-based observer with application to flexible-joint manipulators. *IEEE Trans Neural Netw* 17(1):118–129

Index

A

Adaptive control, 3–5, 7–11, 58–60, 62, 71–92, 120, 133–144, 203, 257, 265, 295–298, 311, 312, 321

Adaptive law, 6, 8, 9, 73, 75–77, 80, 82, 83, 87, 88, 118, 122, 123, 136–140, 143, 144, 147, 149, 150, 155, 156, 158, 160, 171, 182, 186, 200, 202, 203, 205, 212–214, 257–259, 265, 271, 272, 274, 283, 287, 296, 297, 299, 322, 323, 325, 326, 328, 341, 345

Adaptive neural sliding mode control, 118, 120, 122, 123, 144–152, 347–351

Adaptive RBF control, 3–5, 133–191, 197–205, 298–299, 319–329, 331–337

Adaptive sliding mode control, 339–362

Asymptotical convergence, 320

182–191, 197, 200, 203, 205, 211, 214, 218–220, 224–227, 232–236, 241–245, 252, 254, 257, 259, 265, 267, 269, 272, 274, 276–277, 279–281, 283–290, 297, 299, 313, 320, 321, 325, 326, 328

Convergence analysis, 9, 341, 342

Coordination transformation, 208

D

Desired feedback control, 86–87

Digital control, 295, 296

Direct robust adaptive control, 83–93

Discrete RBF neural network, 311–337

Discrete system, 311, 319

Dynamic equation, 5, 6, 76, 133, 134, 142, 144, 153, 155, 156, 193, 196, 197, 204, 208, 209, 213, 260, 273, 299, 312

B

Backstepping control, 251–291

Barbalat’s lemma, 9, 211, 213

Basis functions, 209, 213, 321, 340

Feedback linearization, 320

Flexible joint robot, 251, 260–275, 282–291

Function approximation, 1, 3, 86–87

C

Cartesian coordinates, 208, 214

Center vector of Gaussian function, 25–28

Centrifugal and Coriolis term, 134, 145, 153, 193, 197, 209

Complex plane, 72, 79

Control law, 7, 9, 11, 61, 72, 73, 76, 77, 79, 80, 83, 106–112, 115, 116, 118, 123, 125–132, 134, 140, 143, 147–150, 154–156, 158, 160–163, 171–174,

G

Gaussian function, 19, 23–28, 30–35, 55, 58, 59, 61, 73, 76, 80, 83, 90, 92, 117, 121, 123, 140, 144, 149, 150, 154–157, 203, 205, 214, 257, 259, 269, 297, 299, 313, 316, 318, 325–327

Gaussian function parameter, 25–28

Ge–Lee (GL) matrices, 193, 210

Global approximation, 133–191

Gradient descent method, 22–25, 33, 56, 59, 62, 71
 Gravitational force, 134, 145, 153, 193, 197, 209

H

Hamilton–Jacobi inequality (HJI), 153–159, 182–191
 Hidden layer, 3, 4, 19, 36, 56, 58, 62, 72, 80, 117, 120, 198, 214, 259, 299
 Hidden neural net, 20–22, 36, 72, 80
 Hurwitz, 85, 114, 146, 348

I

Inertia matrix, 134, 145, 153, 193, 197, 209
 Initial weight value, 24, 25, 28, 34, 60, 63, 76, 77, 83, 90, 123, 140, 144, 149, 150, 203, 205, 214, 259, 274, 297, 325, 326, 328, 344, 346
 Input layer, 3, 36, 72, 80
 Inverted pendulum, 76, 77, 118, 122, 252–260, 276–282, 345

J

Jacobian matrix, 205, 209, 211, 214
 Joint space, 208, 209
 Joint torque, 134, 145, 154, 193, 197, 209

L

L^n_∞ , 196, 202, 213
 L_2 , 153, 155
 Learning rate, 24, 34, 56, 59
 Local approximation, 193–249
 Lyapunov stability theory, 71, 114
 Lyapunov theorem, 86

M

Maximum eigenvalue, 138
 Minimum eigenvalue, 138, 196, 202, 258, 271
 Model reference adaptive control, 58–60
 Momentum factor, 24, 34, 56, 59, 62

N

Neural approximation, 71–83
 Neural net, 19–23, 26, 33, 36, 259, 340

Neural network control, v, vi, 1, 19, 76, 133, 153, 205, 252, 320
 Neural network modeling, 208–210
 Nominal model, 134, 140, 144, 193–197
 Nonlinear function, 3, 36, 71, 73, 79, 80, 252, 312

Nonlinear smooth function, 320

O

Observer, 339–362
 Optimal weight vector, 321

P

Polynomial, 72, 79, 85, 114
 Positive definite matrix, 210, 258, 270

R

Radial basis function, v, 3, 134
 RBF approximation, 25–32, 46–49, 135, 147, 269–272, 321, 354–355, 358–360
 RBF neural network, 3–5, 19–53, 55–69, 71–112, 116, 117, 120, 121, 134–144, 149–150, 153–191, 203, 205, 269, 293–309, 312, 313, 316, 318, 320, 325, 327, 339
 structure, 20, 63
 RBF observer, 339–362
RKM See Runge–Kutta–Merson (RKM)
 Robotic manipulators, 4, 134–144, 195, 197–205, 295, 298
 Robust control, 113, 153–159, 182–191, 193–197, 252
 Robust term, 144–152, 194, 200, 340, 341
 Runge–Kutta–Merson (RKM), 293–297

S

Schwarz inequality, 271
 Self adjust control, 61–63, 68–69
 Single link flexible joint robot, 260–275, 279–291
 Single-link robot, 343, 349
 Singularity, 82, 259, 271
 SISO system, 211, 295–298, 343, 347
 Skew symmetric, 148, 155, 195, 201, 209, 213
 Sliding mode control, 113–132, 399–362
 Sliding mode function, 114, 116, 120, 145, 194, 348

Stability analysis, 3, 37, 73–75, 80–83, 147–148, 195–196, 200–203, 312–316, 322–324, 345

Supervisory control law, 55–57

Symmetric and positive definite, 201, 212

T

Task space, 205–217, 240–249

Transfer function, 211, 341

V

Virtual control, 263, 264, 266, 267

W

Weight value, 20, 23–25, 28, 33–35, 55, 58, 60, 63, 72, 74, 76, 77, 80, 81, 83, 90, 92, 123, 135, 140, 144, 147, 149, 150, 154, 155, 198, 203, 205, 212, 214, 257, 259, 269, 274, 297, 313, 325, 326, 328, 340, 344, 346

Weight vector, 35–37, 55, 56, 58, 61, 87, 135, 299, 321

Width of Gaussian function, 25

Width vector, 23, 33

Z

Zhang, J., 314