

# Neural Network Control of Robot Manipulators and Nonlinear Systems

---

F. L. Lewis, S. Jagannathan  
and A. Yeşildirek



---

**Neural Network Control of Robot  
Manipulators and Nonlinear Systems**

---

**UK** Taylor & Francis Ltd, 1 Gunpowder Square, London, EC4A 3DE  
**USA** Taylor & Francis Inc., 325 Chestnut Street, Philadelphia PA 19106

Copyright © Taylor & Francis 1999

*All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, electrostatic, magnetic tape, mechanical, photocopying, recording or otherwise, without the prior permission of the copyright owner.*

**British Library Cataloguing in Publication Data**

A catalogue record for this book is available from the British Library.  
ISBN 0-7484-0596-8 (cased)

**Library of Congress Cataloguing-in-Publication Data are available**

Cover design by Amanda Barragry

Printed and bound by T.J. International Ltd, Padstow, UK

Cover printed by Flexiprint, Lancing, West Sussex



---

# **Neural Network Control of Robot Manipulators and Nonlinear Systems**

---

F.L. LEWIS

*Automation and Robotics Research Institute  
The University of Texas at Arlington*

S. JAGANNATHAN

*Systems and Controls Research  
Caterpillar, Inc., Mossville*

A. YEŞILDIREK

*Manager, New Product Development  
Depsa, Panama City*



# Contents

<b>List of Tables of Design Equations</b>	<b>xi</b>
<b>List of Figures</b>	<b>xviii</b>
<b>Series Introduction</b>	<b>xix</b>
<b>Preface</b>	<b>xxi</b>
<b>1 Background on Neural Networks</b>	<b>1</b>
1.1 NEURAL NETWORK TOPOLOGIES AND RECALL . . . . .	2
1.1.1 Neuron Mathematical Model . . . . .	2
1.1.2 Multilayer Perceptron . . . . .	8
1.1.3 Linear-in-the-Parameter (LIP) Neural Nets . . . . .	10
1.1.4 Dynamic Neural Networks . . . . .	13
1.2 PROPERTIES OF NEURAL NETWORKS . . . . .	25
1.2.1 Classification, Association, and Pattern Recognition . . . . .	26
1.2.2 Function Approximation . . . . .	30
1.3 NEURAL NETWORK WEIGHT SELECTION AND TRAINING . . . . .	33
1.3.1 Direct Computation of the Weights . . . . .	34
1.3.2 Training the One-Layer Neural Network— Gradient Descent	36
1.3.3 Training the Multilayer Neural Network— Backpropagation Tuning . . . . .	42
1.3.4 Improvements on Gradient Descent . . . . .	51
1.3.5 Hebbian Tuning . . . . .	56
1.3.6 Continuous-Time Tuning . . . . .	58
1.4 REFERENCES . . . . .	61
1.5 PROBLEMS . . . . .	63
<b>2 Background on Dynamic Systems</b>	<b>69</b>
2.1 DYNAMICAL SYSTEMS . . . . .	69
2.1.1 Continuous-Time Systems . . . . .	70
2.1.2 Discrete-Time Systems . . . . .	73
2.2 SOME MATHEMATICAL BACKGROUND . . . . .	77
2.2.1 Vector and Matrix Norms . . . . .	77
2.2.2 Continuity and Function Norms . . . . .	79
2.3 PROPERTIES OF DYNAMICAL SYSTEMS . . . . .	80

2.3.1	Stability . . . . .	80
2.3.2	Passivity . . . . .	82
2.3.3	Observability and Controllability . . . . .	85
2.4	FEEDBACK LINEARIZATION AND CONTROL SYSTEM DESIGN	88
2.4.1	Input-Output Feedback Linearization Controllers . . . . .	89
2.4.2	Computer Simulation of Feedback Control Systems . . . . .	94
2.4.3	Feedback Linearization for Discrete-Time Systems . . . . .	96
2.5	NONLINEAR STABILITY ANALYSIS AND CONTROLS DESIGN	100
2.5.1	Lyapunov Analysis for Autonomous Systems . . . . .	100
2.5.2	Controller Design Using Lyapunov Techniques . . . . .	105
2.5.3	Lyapunov Analysis for Non-Autonomous Systems . . . . .	109
2.5.4	Extensions of Lyapunov Techniques and Bounded Stability .	111
2.6	REFERENCES . . . . .	117
2.7	PROBLEMS . . . . .	119
<b>3</b>	<b>Robot Dynamics and Control</b>	<b>125</b>
3.0.1	Commercial Robot Controllers . . . . .	125
3.1	KINEMATICS AND JACOBIANS . . . . .	126
3.1.1	Kinematics of Rigid Serial-Link Manipulators . . . . .	127
3.1.2	Robot Jacobians . . . . .	130
3.2	ROBOT DYNAMICS AND PROPERTIES . . . . .	131
3.2.1	Joint Space Dynamics and Properties . . . . .	132
3.2.2	State Variable Representations . . . . .	136
3.2.3	Cartesian Dynamics and Actuator Dynamics . . . . .	137
3.3	COMPUTED-TORQUE (CT) CONTROL AND COMPUTER SIMULATION . . . . .	138
3.3.1	Computed-Torque (CT) Control . . . . .	138
3.3.2	Computer Simulation of Robot Controllers . . . . .	140
3.3.3	Approximate Computed-Torque Control and Classical Joint Control . . . . .	145
3.3.4	Digital Control . . . . .	147
3.4	FILTERED-ERROR APPROXIMATION-BASED CONTROL . .	153
3.4.1	A General Controller Design Framework Based on Approximation . . . . .	156
3.4.2	Computed-Torque Control Variant . . . . .	159
3.4.3	Adaptive Control . . . . .	159
3.4.4	Robust Control . . . . .	164
3.4.5	Learning Control . . . . .	167
3.5	CONCLUSIONS . . . . .	169
3.6	REFERENCES . . . . .	170
3.7	PROBLEMS . . . . .	171
<b>4</b>	<b>Neural Network Robot Control</b>	<b>175</b>
4.1	ROBOT ARM DYNAMICS AND TRACKING ERROR DYNAMICS	177
4.2	ONE-LAYER FUNCTIONAL-LINK NEURAL NETWORK CONTROLLER . . . . .	181
4.2.1	Approximation by One-Layer Functional-Link NN . . . . .	182

4.2.2	NN Controller and Error System Dynamics . . . . .	183
4.2.3	Unsupervised Backpropagation Weight Tuning . . . . .	184
4.2.4	Augmented Unsupervised Backpropagation Tuning—Removing the PE Condition . . . . .	189
4.2.5	Functional-Link NN Controller Design and Simulation Example	192
4.3	TWO-LAYER NEURAL NETWORK CONTROLLER . . . . .	196
4.3.1	NN Approximation and the Nonlinearity in the Parameters Problem . . . . .	196
4.3.2	Controller Structure and Error System Dynamics . . . . .	198
4.3.3	Weight Updates for Guaranteed Tracking Performance . . . . .	200
4.3.4	Two-Layer NN Controller Design and Simulation Example .	208
4.4	PARTITIONED NN AND SIGNAL PREPROCESSING . . . . .	208
4.4.1	Partitioned NN . . . . .	210
4.4.2	Preprocessing of Neural Net Inputs . . . . .	211
4.4.3	Selection of a Basis Set for the Functional-Link NN . . . . .	211
4.5	PASSIVITY PROPERTIES OF NN CONTROLLERS . . . . .	214
4.5.1	Passivity of the Tracking Error Dynamics . . . . .	214
4.5.2	Passivity Properties of NN Controllers . . . . .	215
4.6	CONCLUSIONS . . . . .	218
4.7	REFERENCES . . . . .	219
4.8	PROBLEMS . . . . .	221
<b>5</b>	<b>Neural Network Robot Control: Applications and Extensions</b>	<b>223</b>
5.1	FORCE CONTROL USING NEURAL NETWORKS . . . . .	224
5.1.1	Force Constrained Motion and Error Dynamics . . . . .	225
5.1.2	Neural Network Hybrid Position/Force Controller . . . . .	227
5.1.3	Design Example for NN Hybrid Position/Force Controller .	234
5.2	ROBOT MANIPULATORS WITH LINK FLEXIBILITY, MOTOR DYNAMICS, AND JOINT FLEXIBILITY . . . . .	235
5.2.1	Flexible-Link Robot Arms . . . . .	235
5.2.2	Robots with Actuators and Compliant Drive Train Coupling	240
5.2.3	Rigid-Link Electrically-Driven (RLED) Robot Arms . . . . .	246
5.3	SINGULAR PERTURBATION DESIGN . . . . .	247
5.3.1	Two-Time-Scale Controller Design . . . . .	248
5.3.2	NN Controller for Flexible-Link Robot Using Singular Perturbations . . . . .	251
5.4	BACKSTEPPING DESIGN . . . . .	260
5.4.1	Backstepping Design . . . . .	260
5.4.2	NN Controller for Rigid-Link Electrically-Driven Robot Using Backstepping . . . . .	264
5.5	CONCLUSIONS . . . . .	272
5.6	REFERENCES . . . . .	272
5.7	PROBLEMS . . . . .	274

<b>6 Neural Network Control of Nonlinear Systems</b>	<b>279</b>
6.1 SYSTEM AND TRACKING ERROR DYNAMICS . . . . .	280
6.1.1 Tracking Controller and Error Dynamics . . . . .	281
6.1.2 Well-Defined Control Problem . . . . .	283
6.2 CASE OF KNOWN FUNCTION $g(\mathbf{x})$ . . . . .	283
6.2.1 Proposed NN Controller . . . . .	284
6.2.2 NN Weight Tuning for Tracking Stability . . . . .	285
6.2.3 Illustrative Simulation Example . . . . .	287
6.3 CASE OF UNKNOWN FUNCTION $g(\mathbf{x})$ . . . . .	288
6.3.1 Proposed NN Controller . . . . .	289
6.3.2 NN Weight Tuning for Tracking Stability . . . . .	291
6.3.3 Illustrative Simulation Examples . . . . .	298
6.4 CONCLUSIONS . . . . .	303
6.5 REFERENCES . . . . .	305
<b>7 NN Control with Discrete-Time Tuning</b>	<b>307</b>
7.1 BACKGROUND AND ERROR DYNAMICS . . . . .	308
7.1.1 Neural Network Approximation Property . . . . .	308
7.1.2 Stability of Systems . . . . .	310
7.1.3 Tracking Error Dynamics for a Class of Nonlinear Systems . .	310
7.2 ONE-LAYER NEURAL NETWORK CONTROLLER DESIGN . .	312
7.2.1 Structure of the One-layer NN Controller and Error System Dynamics . . . . .	313
7.2.2 One-layer Neural Network Weight Updates . . . . .	314
7.2.3 Projection Algorithm . . . . .	318
7.2.4 Ideal Case: No Disturbances or NN Reconstruction Errors . .	323
7.2.5 One-layer Neural Network Weight Tuning Modification for Relaxation of Persistency of Excitation Condition . . . . .	323
7.3 MULTILAYER NEURAL NETWORK CONTROLLER DESIGN . .	329
7.3.1 Structure of the NN Controller and Error System Dynamics . .	332
7.3.2 Multilayer Neural Network Weight Updates . . . . .	333
7.3.3 Projection Algorithm . . . . .	340
7.3.4 Multilayer Neural Network Weight Tuning Modification for Relaxation of Persistency of Excitation Condition . . . . .	342
7.4 PASSIVITY PROPERTIES OF THE NN . . . . .	353
7.4.1 Passivity Properties of the Tracking Error System . . . . .	353
7.4.2 Passivity Properties of One-layer Neural Networks and the Closed-Loop System . . . . .	354
7.4.3 Passivity Properties of Multilayer Neural Networks . . . . .	356
7.5 CONCLUSIONS . . . . .	357
7.6 REFERENCES . . . . .	357
7.7 PROBLEMS . . . . .	359
<b>8 Discrete-Time Feedback Linearization by Neural Networks</b>	<b>361</b>
8.1 SYSTEM DYNAMICS AND THE TRACKING PROBLEM . . . . .	362
8.1.1 Tracking Error Dynamics for a Class of Nonlinear Systems . .	362
8.2 NN CONTROLLER DESIGN FOR FEEDBACK LINEARIZATION	364

8.2.1	NN Approximation of Unknown Functions . . . . .	365
8.2.2	Error System Dynamics . . . . .	366
8.2.3	Well-Defined Control Problem . . . . .	368
8.2.4	Proposed Controller . . . . .	369
8.3	SINGLE-LAYER NN FOR FEEDBACK LINEARIZATION . . . . .	369
8.3.1	Weight Updates Requiring Persistence of Excitation . . . . .	370
8.3.2	Projection Algorithm . . . . .	377
8.3.3	Weight Updates not Requiring Persistence of Excitation . . . . .	378
8.4	MULTILAYER NEURAL NETWORKS FOR FEEDBACK LINEARIZATION . . . . .	385
8.4.1	Weight Updates Requiring Persistence of Excitation . . . . .	386
8.4.2	Weight Updates not Requiring Persistence of Excitation . . . . .	392
8.5	PASSIVITY PROPERTIES OF THE NN . . . . .	403
8.5.1	Passivity Properties of the Tracking Error System . . . . .	403
8.5.2	Passivity Properties of One-layer Neural Network Controllers	404
8.5.3	Passivity Properties of Multilayer Neural Network Controllers	406
8.6	CONCLUSIONS . . . . .	408
8.7	REFERENCES . . . . .	408
8.8	PROBLEMS . . . . .	409
<b>9</b>	<b>State Estimation Using Discrete-Time Neural Networks</b>	<b>413</b>
9.1	IDENTIFICATION OF NONLINEAR DYNAMICAL SYSTEMS . .	415
9.2	IDENTIFIER DYNAMICS FOR MIMO SYSTEMS . . . . .	415
9.3	MULTILAYER NEURAL NETWORK IDENTIFIER DESIGN . .	418
9.3.1	Structure of the NN Controller and Error System Dynamics .	418
9.3.2	Three-Layer Neural Network Weight Updates . . . . .	420
9.4	PASSIVITY PROPERTIES OF THE NN . . . . .	425
9.5	SIMULATION RESULTS . . . . .	427
9.6	CONCLUSIONS . . . . .	428
9.7	REFERENCES . . . . .	428
9.8	PROBLEMS . . . . .	430







# List of Tables

1.3.1	Basic Matrix Calculus and Trace Identities . . . . .	39
1.3.2	Backpropagation Algorithm Using Sigmoid Activation Functions: Two-Layer Net . . . . .	48
1.3.3	Continuous-Time Backpropagation Algorithm Using Sigmoid Ac- tivation Functions . . . . .	60
3.2.1	Properties of Robot Arm Dynamics . . . . .	133
3.3.1	Robot Manipulator Control Algorithms . . . . .	139
3.4.1	Filtered-Error Approximation-Based Control Algorithms . . . . .	155
4.1.1	Properties of Robot Arm Dynamics . . . . .	178
4.2.1	FLNN Controller for Ideal Case, or for Nonideal Case with PE .	186
4.2.2	FLNN Controller with Augmented Tuning to Avoid PE . . . . .	190
4.3.1	Two-Layer NN Controller for Ideal Case . . . . .	201
4.3.2	Two-Layer NN Controller with Augmented Backprop Tuning .	203
4.3.3	Two-Layer NN Controller with Augmented Hebbian Tuning . .	206
5.0.1	Properties of Robot Arm Dynamics . . . . .	224
5.1.1	NN Force/Position Controller. . . . .	230
5.3.1	NN Controller for Flexible-Link Robot Arm . . . . .	257
5.4.1	NN Backstepping Controller for RLED Robot Arm . . . . .	268
6.2.1	Neural Net Controller with Known $g(\mathbf{x})$ . . . . .	286
6.3.1	Neural Net Controller with Unknown $f(\mathbf{x})$ and $g(\mathbf{x})$ . . . . .	292
7.2.1	Discrete-Time Controller Using One-Layer Neural Net: PE Re- quired . . . . .	315
7.2.2	Discrete-Time Controller Using One-Layer Neural Net: PE not Required . . . . .	324
7.3.1	Discrete-Time Controller Using Three-Layer Neural Net: PE Re- quired . . . . .	334
7.3.2	Discrete-Time Controller Using Three-Layer Neural Net: PE not Required . . . . .	346
8.3.1	Discrete-Time Controller Using One-Layer Neural Net: PE Re- quired . . . . .	370

8.3.2	Discrete-Time Controller Using One-layer Neural Net: PE not Required . . . . .	379
8.4.1	Discrete-Time Controller Using Multilayer Neural Net: PE Required . . . . .	386
8.4.2	Discrete-Time Controller Using Multilayer Neural Net: PE not Required . . . . .	393
9.3.1	Multilayer Neural Net Identifier . . . . .	425

# List of Figures

1.1.1	Neuron anatomy. From B. Kosko (1992). . . . .	2
1.1.2	Mathematical model of a neuron. . . . .	3
1.1.3	Some common choices for the activation function. . . . .	4
1.1.4	One-layer neural network. . . . .	5
1.1.5	Output surface of a one-layer NN. (a) Using sigmoid activation function. (b) Using hard limit activation function. (c) Using radial basis function. . . . .	7
1.1.6	Two-layer neural network. . . . .	8
1.1.7	EXCLUSIVE-OR implemented using two-layer neural network. . . . .	9
1.1.8	Output surface of a two-layer NN. (a) Using sigmoid activation function. (b) Using hard limit activation function. . . . .	11
1.1.9	Two-dimensional separable gaussian functions for an RBF NN. . . . .	13
1.1.10	Receptive field functions for a 2-D CMAC NN with second-order splines. . . . .	14
1.1.11	Hopfield dynamical neural net. . . . .	15
1.1.12	Continuous-time Hopfield net hidden-layer neuronal processing element (NPE) dynamics. . . . .	15
1.1.13	Discrete-time Hopfield net hidden-layer NPE dynamics. . . . .	16
1.1.14	Continuous-time Hopfield net in block diagram form. . . . .	16
1.1.15	Hopfield net functions. (a) Symmetric sigmoid activation function. (b) Inverse of symmetric sigmoid activation function. . . . .	19
1.1.16	Hopfield net phase-plane plots; $x_2(t)$ versus $x_1(t)$ . . . . .	20
1.1.17	Lyapunov energy surface for an illustrative Hopfield net. . . . .	20
1.1.18	Generalized continuous-time dynamical neural network. . . . .	21
1.1.19	Phase-plane plot of discrete-time NN showing attractor. . . . .	23
1.1.20	Phase-plane plot of discrete-time NN with modified $V$ weights. . . . .	23
1.1.21	Phase-plane plot of discrete-time NN with modified $V$ weights showing limit-cycle attractor. . . . .	24
1.1.22	Phase-plane plot of discrete-time NN with modified $A$ matrix. . . . .	24
1.1.23	Phase-plane plot of discrete-time NN with modified $A$ matrix and $V$ matrix. . . . .	25
1.2.1	Decision regions of a simple one-layer NN. . . . .	26
1.2.2	Types of decision regions that can be formed using single- and multi-layer NN. From R.P. Lippmann (1987). . . . .	27

1.2.3	Output error plots versus weights for a neuron. (a) Error surface using sigmoid activation function. (b) Error contour plot using sigmoid activation function. (c) Error surface using hard limit activation function. . . . .	29
1.3.1	Pattern vectors to be classified into 4 groups: $+, \circ, \times, *$ . Also shown are the initial decision boundaries. . . . .	41
1.3.2	NN decision boundaries. (a) After three epochs of training. (b) After six epochs of training. . . . .	43
1.3.3	Least-squares NN output error versus epoch . . . . .	44
1.3.4	The adjoint (backpropagation) neural network. . . . .	49
1.3.5	Function $y = f(x)$ to be approximated by two-layer NN and its samples for training. . . . .	50
1.3.6	Samples of $f(x)$ and actual NN output. (a) Using initial random weights. (b) After training for 50 epochs. . . . .	52
1.3.6	Samples of $f(x)$ and actual NN output (cont'd). (c) After training for 200 epochs. (d) After training for 873 epochs. . . . .	53
1.3.7	Least-squares NN output error as a function of training epoch. .	54
1.3.8	Typical 1-D NN error surface $e = Y - \sigma(V^T X)$ . . . . .	55
1.5.1	A dynamical neural network with internal neuron dynamics. . .	64
1.5.2	A dynamical neural network with outer feedback loops. . . . .	64
2.1.1	Continuous-time single-input Brunovsky form. . . . .	71
2.1.2	Van der Pol Oscillator time history plots. (a) $x_1(t)$ and $x_2(t)$ versus $t$ . (b) Phase-plane plot $x_2$ versus $x_1$ showing limit cycle. .	74
2.1.3	Discrete-time single-input Brunovsky form. . . . .	75
2.3.1	Illustration of uniform ultimate boundedness (UUB). . . . .	82
2.3.2	System with measurement nonlinearity. . . . .	84
2.3.3	Two passive systems in feedback interconnection. . . . .	85
2.4.1	Feedback linearization controller showing PD outer loop and nonlinear inner loop. . . . .	91
2.4.2	Simulation of feedback linearization controller, $T = 10$ sec. (a) Actual output $y(t)$ and desired output $y_d(t)$ . (b) Tracking error $e(t)$ . (c) Internal dynamics state $x_2(t)$ . . . . .	97
2.4.3	Simulation of feedback linearization controller, $T = 0.1$ sec. (a) Actual output $y(t)$ and desired output $y_d(t)$ . (b) Tracking error $e(t)$ . (c) Internal dynamics state $x_2(t)$ . . . . .	98
2.5.1	Sample trajectories of system with local asymptotic stability. (a) $x_1(t)$ and $x_2(t)$ versus $t$ . (b) Phase-plane plot of $x_2$ versus $x_1$ . . . . .	103
2.5.2	Sample trajectories of SISL system. (a) $x_1(t)$ and $x_2(t)$ versus $t$ . (b) Phase-plane plot of $x_2$ versus $x_1$ . . . . .	104
2.5.3	A function satisfying the condition $xc(x) > 0$ . . . . .	105
2.5.4	Signum function. . . . .	106
2.5.5	Depiction of a time-varying function $L(x, t)$ that is positive definite ( $L_0(x) < L(x, t)$ ) and decrescent ( $L(x, t) \leq L_1(x)$ ). . . . .	110
2.5.6	Sample trajectories of Mathieu system. (a) $x_1(t)$ and $x_2(t)$ versus $t$ . (b) Phase-plane plot of $x_2$ versus $x_1$ . . . . .	112
2.5.7	Sample closed-loop trajectories of UUB system. . . . .	117

3.1.1	Basic robot arm geometries. (a) Articulated arm, revolute coordinates (RRR). (b) Spherical coordinates (RRP). (c) SCARA arm (RRP). . . . .	128
3.1.1	Basic robot arm geometries (cont'd.). (d) Cylindrical coordinates (RPP). (e) Cartesian arm, rectangular coordinates (PPP). . . . .	129
3.1.2	Denavit-Hartenberg coordinate frames in a serial-link manipulator. . . . .	129
3.2.1	Two-link planar robot arm. . . . .	134
3.3.1	PD computed-torque controller. . . . .	140
3.3.2	PD computed-torque controller, Part I. . . . .	142
3.3.3	Joint tracking errors using PD computed-torque controller under ideal conditions. . . . .	144
3.3.4	Joint tracking errors using PD computed-torque controller with constant unknown disturbance. . . . .	145
3.3.5	PD classical joint controller. . . . .	146
3.3.6	Joint tracking errors using PD-gravity controller. . . . .	148
3.3.7	Joint tracking errors using classical independent joint control. . . . .	148
3.3.8	Digital controller, Part I: Routine robot.m Part I. . . . .	150
3.3.9	Joint tracking errors using digital computed-torque controller, $T = 20$ msec. . . . .	153
3.3.10	Joint 2 control torque using digital computed-torque controller, $T = 20$ msec. . . . .	154
3.3.11	Joint tracking errors using digital computed-torque controller, $T = 100$ msec. . . . .	154
3.3.12	Joint 2 control torque using digital computed-torque controller, $T = 100$ msec. . . . .	156
3.4.1	Filtered error approximation-based controller. . . . .	158
3.4.2	Adaptive controller. . . . .	162
3.4.3	Response using adaptive controller. (a) Actual and desired joint angles. (b) Mass estimates. . . . .	163
3.4.4	Response using adaptive controller with incorrect regression matrix, showing the effects of unmodelled dynamics. (a) Actual and desired joint angles. (b) Mass estimates. . . . .	164
3.4.5	Robust controller. . . . .	168
3.4.6	Typical behavior of robust controller. . . . .	169
3.7.1	Two-link polar robot arm. . . . .	172
3.7.2	Three-link cylindrical robot arm. . . . .	172
4.0.1	Two-layer neural net. . . . .	176
4.1.1	Filtered error approximation-based controller. . . . .	179
4.2.1	One-layer functional-link neural net. . . . .	181
4.2.2	Neural net control structure. . . . .	183
4.2.3	Two-link planar elbow arm. . . . .	192
4.2.4	Response of NN controller with backprop weight tuning: actual and desired joint angles. . . . .	193
4.2.5	Response of NN controller with backprop weight tuning: representative weight estimates. . . . .	194

4.2.6	Response of NN controller with improved weight tuning: actual and desired joint angles. . . . .	194
4.2.7	Response of NN controller with improved weight tuning: representative weight estimates. . . . .	195
4.2.8	Response of controller without NN. actual and desired joint angles. . . . .	195
4.3.1	Multilayer NN controller structure. . . . .	199
4.3.2	Response of NN controller with improved weight tuning: actual and desired joint angles. . . . .	209
4.3.3	Response of NN controller with improved weight tuning: representative weight estimates. . . . .	209
4.4.1	Partitioned neural net. . . . .	211
4.4.2	Neural subnet for estimating $M(q)\zeta_1(t)$ . . . . .	213
4.4.3	Neural subnet for estimating $V_m(q, \dot{q})\zeta_2(t)$ . . . . .	213
4.4.4	Neural subnet for estimating $G(q)$ . . . . .	213
4.4.5	Neural subnet for estimating $F(\dot{q})$ . . . . .	214
4.5.1	Two-layer neural net closed-loop error system. . . . .	216
5.1.1	Two-layer neural net. . . . .	228
5.1.2	Neural net hybrid position/force controller. . . . .	229
5.1.3	Closed-loop position error system. . . . .	233
5.1.4	Two-link planar elbow arm with circle constraint. . . . .	234
5.1.5	NN force/position controller simulation results. (a) Desired and actual motion trajectories $q_{1_d}(t)$ and $q_1(t)$ . (b) Force trajectory $\lambda(t)$ . . . . .	236
5.2.1	Acceleration/deceleration torque profile $\tau(t)$ . . . . .	240
5.2.2	Open-loop response of flexible arm: tip position $q_r(t)$ (solid) and velocity (dashed). . . . .	241
5.2.3	Open-loop response of flexible arm: flexible modes $q_{f_1}(t), q_{f_2}(t)$ . . . . .	241
5.2.4	Two canonical control problems with high-frequency modes. (a) Flexible-link robot arm. (b) Flexible-joint robot arm. . . . .	243
5.2.5	DC motor with shaft compliance. (a) Electrical subsystem. (b) Mechanical subsystem. . . . .	244
5.2.6	Step response of DC motor with no shaft flexibility. Motor speed in rad/s. . . . .	246
5.2.7	Step response of DC motor with very flexible shaft. . . . .	247
5.3.1	Neural net controller for flexible-link robot arm. . . . .	254
5.3.2	Response of flexible arm with NN and boundary layer correction. Actual and desired tip positions and velocities, $\varepsilon = 0.26$ . . . . .	258
5.3.3	Response of flexible arm with NN and boundary layer correction. Flexible modes, $\varepsilon = 0.26$ . . . . .	258
5.3.4	Response of flexible arm with NN and boundary layer correction. Actual and desired tip positions and velocities, $\varepsilon = 0.1$ . . . . .	259
5.3.5	Response of flexible arm with NN and boundary layer correction. Flexible modes, $\varepsilon = 0.1$ . . . . .	260
5.4.1	Backstepping controller. . . . .	261
5.4.2	Backstepping neural network controller. . . . .	267

5.4.3	Response of RLED controller with only PD control. (a) Actual and desired joint angle $q_1(t)$ . (b) Actual and desired joint angle $q_2(t)$ . (c) Tracking errors $e_1(t), e_2(t)$ . (d) Control torques $K_T i(t)$ .	270
5.4.4	Response of RLED backstepping NN controller. (a) Actual and desired joint angle $q_1(t)$ . (b) Actual and desired joint angle $q_2(t)$ . (c) Tracking errors $e_1(t), e_2(t)$ . (d) Control torques $K_T i(t)$ .	271
6.2.1	Neural network controller with known $g(\mathbf{x})$ .	285
6.2.2	Open-loop state trajectory of the Van der Pol's system.	288
6.2.3	Actual and desired state $x_1$ .	288
6.2.4	Actual and desired state $x_2$ .	289
6.3.1	NN controller with unknown $f(\mathbf{x})$ and $g(\mathbf{x})$ .	291
6.3.2	Illustration of the upper bound on $\ \tilde{\Theta}_g\ $ .	296
6.3.3	Illustration of the invariant set.	298
6.3.4	Actual and desired states.	299
6.3.5	Control input.	300
6.3.6	Actual and desired states.	300
6.3.7	Control input.	301
6.3.8	Chemical stirred-tank reactor (CSTR) process.	301
6.3.9	CSTR open-loop response to a disturbance.	303
6.3.10	Response with NN controller. Reactant temperature, $T$ .	304
6.3.11	Response with NN controller. The state $x_1(t)$ .	304
7.1.1	A multilayer neural network.	309
7.2.1	One-layer discrete-time neural network controller structure.	314
7.2.2	Response of neural network controller with delta-rule weight tuning and small $\alpha$ . (a) Actual and desired joint angles. (b) Neural network outputs	319
7.2.3	Response of neural network controller with delta-rule weight tuning and projection algorithm. (a) Actual and desired joint angles. (b) Neural network outputs.	321
7.2.4	Response of neural network controller with delta-rule weight tuning and large $\alpha$ . (a) Actual and desired joint angles. (b) Neural network outputs.	322
7.2.5	Response of neural network controller with improved weight tuning and projection algorithm. (a) Actual and desired joint angles. (b) Neural network outputs.	328
7.2.6	Response of the PD controller.	329
7.2.7	Response of neural network controller with improved weight tuning and projection algorithm. (a) Desired and actual state 1. (b) Desired and actual state 2.	330
7.2.8	Response of neural network controller with improved weight tuning in the presence of bounded disturbances. (a) Desired and actual state 1. (b) Desired and actual state 2.	331
7.3.1	Multilayer neural network controller structure.	333

7.3.2	Response of multilayer neural network controller with delta-rule weight tuning and small $\alpha_3$ . (a) Desired and actual trajectory. (b) Representative weight estimates. . . . .	339
7.3.3	Response of multilayer neural network controller with delta-rule weight tuning and projection algorithm with large $\alpha_3$ . (a) Desired and actual trajectory. (b) Representative weight estimates. . . . .	343
7.3.4	Response of multilayer neural network controller with delta-rule weight tuning and large $\alpha_3$ . (a) Desired and actual trajectory. (b) Representative weight estimates. . . . .	344
7.3.5	Response of multilayer neural network controller with delta-rule weight tuning and projection algorithm with small $\alpha_3$ . (a) Desired and actual trajectory. (b) Representative weight estimates. . . . .	345
7.3.6	Response of multilayer neural network controller with improved weight tuning and projection algorithm with small $\alpha_3$ . (a) Desired and actual trajectory. (b) Representative weight estimates. . . . .	348
7.3.7	Response of the PD controller. Desired and actual trajectory. . . . .	349
7.3.8	Response of multilayer neural network controller with improved weight tuning and projection algorithm. (a) Desired and actual state 1. (b) Desired and actual state 2. . . . .	351
7.3.9	Response of multilayer neural network controller with projection algorithm and large $\alpha_3$ in the presence of bounded disturbances. (a) Desired and actual state 1. (b) Desired and actual state 2. . . . .	352
7.4.1	Neural network closed-loop system using an $n$ -layer neural network. . . . .	354
8.2.1	Discrete-time neural network controller structure for feedback linearization. . . . .	366
8.4.1	Response of NN controller with delta-rule based weight tuning. (a) Actual and desired joint angles. (b) Neural network outputs . . . . .	392
8.4.2	Response of the NN controller with improved weight tuning and projection algorithm. (a) Actual and desired joint angles. (b) Neural network outputs. . . . .	402
8.4.3	Response of the PD controller. . . . .	402
8.4.4	Response of the NN controller with improved weight tuning and projection algorithm. (a) Desired and actual state 1. (b) Desired and actual state 2. . . . .	403
8.4.5	Response of NN controller with improved weight tuning in the presence of bounded disturbances. (a) Desired and actual state 1. (b) Desired and actual state 2. . . . .	403
8.5.1	The NN closed-loop system using a one-layer neural nework. . . . .	404
9.1.1	Multilayer neural network identifier models. . . . .	416
9.3.1	Multilayer neural network identifier structure. . . . .	420
9.4.1	Neural network closed-loop identifier system. . . . .	426
9.5.1	Response of neural network identifier with projection algorithm in the presence of bounded disturbances. (a) Desired and actual state 1. (b) Desired and actual state 2. . . . .	429

# Series Introduction

Control systems has a long and distinguished tradition stretching back to the nineteenth-century dynamics and stability theory. Its establishment as a major engineering discipline in the 1950s arose, essentially, from Second World War-driven work on frequency response methods by, amongst others, Nyquist, Bode and Wiener. The intervening 40 years has seen quite unparalleled developments in the underlying theory with applications ranging from the ubiquitous PID controller, widely encountered in the process industries, through to high-performance fidelity controllers typical of aerospace applications. This development has been increasingly underpinned by rapid development in the, essentially enabling, technology of computing software and hardware.

This view of mathematically model-based systems and control as a mature discipline masks relatively new and rapid developments in the general area of robust control. Here an intense research effort is being directed to the development of high-performance controllers which (at least) are robust to specified classes of plant uncertainty. One measure of this effort is the fact that, after a relatively short period of work, near world test of classes of robust controllers have been undertaken in the aerospace industry. Again, this work is supported by computing hardware and software developments, such as the toolboxes available within numerous commercially marketed controller design/simulation packages.

Recently, there has been increasing interest in the use of so-called "intelligent" control techniques such as fuzzy logic and neural networks. Basically, these rely on learning (in a prescribed manner) the input-output behavior of the plant to be controlled. Already, it is clear that there is little to be gained by applying these techniques to cases where mature mathematical model-based approaches yield high-performance control. Instead, their role (in general terms) almost certainly lies in areas where the processes encountered are ill-defined, complex, nonlinear, time-varying and stochastic. A detailed evaluation of their (relative) potential awaits the appearance of a rigorous supporting base (underlying theory and implementation architectures, for example) the essential elements of which are beginning to appear in learned journals and conferences.

Elements of control and systems theory/engineering are increasingly finding use outside traditional numerical processing environments. One such general area is intelligent command and control systems which are central, for example, to innovative manufacturing and the management of advanced transportation systems. Another is discrete event systems which mix numeric and logic decision making.

It was in response to these exciting new developments that the present Systems

and Control book series was conceived. It publishes high-quality research texts and reference works in the diverse areas which systems and control now includes. In addition to basic theory, experimental and/or application studies are welcome, as are expository texts where theory, verification and applications come together to provide a unifying coverage of a particular topic or topics.

The book series itself arose out of the seminal text: the 1992 centenary first English translation of Lyapunov's memoir *The General Problem of the Stability of Motion* by A. T. Fuller, and was followed by the 1994 publication of *Advances in Intelligent Control* by C. J. Harris. Since then a number of titles have been published and many more are planned.

A full list of books in this series is given below:

*Advances in Intelligent Control*, edited by C.J. Harris

*Intelligent Control in Biomedicine*, edited by D.A. Linkens

*Advances in Flight Control*, edited by M.B. Tischler

*Multiple Model Approaches to Modelling and Control*, edited by R. Murray-Smith and T.A. Johansen

*A Unified Algebraic Approach to Control Design*, R.E. Skelton, T. Iwasaki and K.M. Grigoriadis

*Neural Network Control of Robot Manipulators and Nonlinear Systems*, F.L. Lewis, S. Jagannathan and A. Yeşildirek

Forthcoming:

*Sliding Mode Control: Theory and Applications*, C. Edwards and S.K. Spurgeon

*Generalized Predictive Control with Applications to Medicine*, H. Mahfouf and D.A. Linkens

*Sliding Mode Control in Electro-Mechanical Systems*, V.I. Utkin, J. Guldner and J. Shi

*From Process Data to PID Controller Design*, L. Wang and W.R. Cluett

E. ROGERS  
J. O'REILLY

# Preface

The function of a feedback controller is to fundamentally change the behavior of a system. Feedback control systems sample the outputs of a system, compare them with a set of desired outputs, and then use the resulting error signals to compute the control inputs to the system in such a fashion that the errors become small. Man-made feedback control systems are responsible for advances in today's aerospace age and are used to control industrial, automotive, and aerospace systems. Naturally occurring feedback controllers are ubiquitous in biological systems. The cell, among the most basic of all life-forms, uses feedback to maintain its homeostasis by regulating the potential difference across the cell membrane. Volterra showed that feedback was responsible for regulating interacting populations of fish in a pond. Darwin showed that feedback over long timeframes was responsible for natural selection. Adam Smith effectively used feedback principles in his study of large-scale economical systems.

The complexity of today's man-made systems has placed severe strains on existing feedback design techniques. Many controls design approaches require known mathematical models of the system, or make assumptions that are violated by actual industrial and aerospace systems. Many feedback controllers designed using today's technology do not learn or adapt to new situations. Natural biological organisms have developed feedback controllers and information processing systems that are extremely efficient, highly redundant, and robust to unexpected disturbances. Chief characteristics of such systems are their ability to learn and adapt to varying environmental conditions and uncertainties and variations in the controlled system. It would be very desirable to use some features of naturally occurring systems in the design of man-made feedback controllers.

Among the most complex, efficient, and adaptive of all biological systems are immense networks of interconnected nerve cells. Such a network is the human nervous system, including the motor control system, basal ganglia, cerebellum, and motor cortex. Artificial neural networks (NN) based on biological neuronal systems have been studied for years, and their properties of learning and adaptation, classification, function approximation, feature extraction, and more have made them of extreme use in signal processing and system identification applications. These are *open-loop* applications, and the theory of NN has been very well developed in this area.

The applications of NN in *closed-loop* feedback control systems have only recently been rigorously studied. When placed in a feedback system, even a static NN becomes a dynamical system and takes on new and unexpected behaviors. As a

result, properties such as the internal stability, passivity, and robustness of the NN must be studied before conclusions about the closed-loop performance can be made. Early papers on neurocontrol failed to study these effects, and only proposed some control topologies, employed some standard weight tuning techniques, and presented some computer simulations indicating good performance.

In this book, the closed-loop applications and properties of NN are studied and developed in rigorous detail using mathematical stability proof techniques that both show how to design neurocontrollers and at the same time provide guaranteed stability and performance. A family of *multiloop neurocontrollers* for various applications are methodically developed based on feedback control engineering principles. The NN controllers are adaptive learning systems, but they do not need usual assumptions made in adaptive control theory such as linearity in the parameters and availability of a known regression matrix. It is shown how to design NN controllers for robot systems, a general class of nonlinear systems, complex industrial systems with vibrations and flexibility effects, force control, motor dynamics control, and more. Both continuous-time and discrete-time weight tuning techniques are presented. The book is designed for a second course in control theory, or for engineers in academia and industry who design feedback controllers for complex systems such as those occurring in actual commercial, industrial, and military applications. The various sorts of neurocontrollers are placed in tables which makes for easy reference on design techniques.

Other books on neurocontrol are by now in print. Generally, these are edited collections of papers and not textbooks. This is among the first textbooks on neurocontrol. The appearance of textbooks can be regarded as indicating when an area of scientific research has reached a certain level of maturity; the first textbooks on classical control theory, for instance, were published after World War II. Thus, one might say that the field of neurocontrol has at last taken on a ‘paradigm’ in the sense of Thomas Kuhn in *The Structure of Scientific Revolution*.

The book can be seen as having three natural sections. Section I provides a background, with Chapter 1 outlining properties and characteristics of neural networks that are important from a feedback control point of view. Included are NN structures, properties, and training. Chapter 2 gives a background on dynamical systems along with some mathematical tools, and a foundation in feedback linearization design and nonlinear stability design. In chapter 3 are discussed robot systems and their control, particularly computed-torque-like control. A general approach for robot control given there based on ‘filtered error’ can be used to unify several different control techniques, including adaptive control, robust control, and the neural net techniques given in this book.

Section II presents a family of neural network feedback controllers for some different sorts of systems described in terms of continuous-time dynamical equations. Chapter 4 lays the foundation of NN control used in the book by deriving neural network controllers for basic rigid robotic systems. Both one-layer and two-layer NN controllers are given. The properties of these controllers are described in terms of stability, robustness, and passivity. Extensions are made in Chapter 5 to more complex practical systems. Studied there are neurocontrollers for force control, link flexibility, joint flexibility, and high-frequency motor dynamics. The two basic approaches to control design employed there are singular perturbations and back-

stepping. Neurocontrollers are designed for a class of nonlinear systems in Chapter 6.

In Section III neurocontrollers are designed for systems described in terms of discrete-time dynamical models. Controllers designed in discrete time have the important advantage that they can be directly implemented in digital form on modern-day microcontrollers and computers. Unfortunately, discrete-time design is far more complex than continuous-time design. One sees this in the complexity of the stability proofs, where completing the square several times with respect to different variables is needed. Chapter 7 provides the basic discrete-time neurocontroller topology and weight tuning algorithms. Chapter 8 confronts the additional complexity introduced by uncertainty in the control influence coefficient, and discrete feedback linearization techniques are presented. Finally, Chapter 9 shows how to use neural networks to identify some important classes of nonlinear systems.

This work was supported by the National Science Foundation under grant ECS-9521673.

Frank L. Lewis  
Arlington, Texas

*This book is dedicated to Christopher*

## Chapter 1

# Background on Neural Networks

In this chapter we provide a brief background on neural networks (NN), covering mainly the topics that will be important in a discussion of NN applications in closed-loop control of dynamical systems. Included are NN topologies and recall, properties, and training techniques. Applications are given in classification, pattern recognition, and function approximation, with examples provided using the MATLAB NN Toolbox (Matlab 1994, Matlab NN Toolbox 1995).

Surveys of NN are given, for instance, by Lippmann (1987), Simpson (1992), and Hush and Horne (1993); many books are also available, as exemplified by Haykin (1994), Kosko (1992), Kung (1993), Levine (1991), Peretto (1992) and other books too numerous to mention. It is not necessary to have an exhaustive knowledge of NN digital signal processing applications for feedback control purposes. Only a few network topologies, tuning techniques, and properties are important, especially the *NN function approximation property*. These are the topics of this chapter.

The uses of NN in closed-loop control are dramatically distinct from their uses in open-loop applications, which are mainly in digital signal processing (DSP). The latter include classification, pattern recognition, and approximation of nondynamic functions (e.g. with no integrators or time delays). In DSP applications, NN usage is backed up by a body of knowledge developed over the years that shows how to choose network topologies and select the weights to yield guaranteed performance. The issues associated with weight training algorithms are well understood. By contrast, in closed-loop control of dynamical systems, most applications have been ad hoc, with open-loop techniques (e.g. backpropagation weight tuning) employed in a naive yet hopeful manner to solve problems associated with dynamic neural net evolution within a feedback loop, where the NN must provide stabilizing controls for the system as well as maintain all its weights bounded. Most published papers have consisted of some loose discussion followed by some simulation examples. By now, several researchers have begun to provide rigorous mathematical analyses of NN in closed-loop control applications (see Chapter 4). The background for these efforts was provided by Narendra and co-workers in several seminal works (see references). It has generally been discovered that standard open-loop weight tuning algorithms

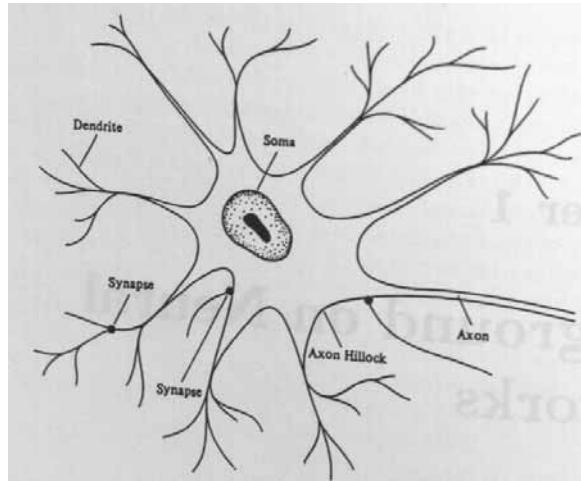


Figure 1.1.1: Neuron anatomy. From B. Kosko (1992).

such as backpropagation or Hebbian tuning must be modified to provide guaranteed stability and tracking in feedback control systems.

## 1.1 NEURAL NETWORK TOPOLOGIES AND RECALL

Neural networks are closely modeled on biological processes for information processing, including specifically the nervous system and its basic unit, the neuron. Signals are propagated in the form of potential differences between the inside and outside of cells. The main components of a neuronal cell are shown in Fig. 1.1.1. Dendrites bring signals from other neurons into the cell body or soma, possibly multiplying each incoming signal by a transfer *weighting coefficient*. In the soma, cell capacitance integrates the signals which collect in the axon hillock. Once the composite signal exceeds a cell *threshold*, a signal, the action potential, is transmitted through the axon. Cell nonlinearities make the composite action potential a *nonlinear function* of the combination of arriving signals. The axon connects through synapses with the dendrites of subsequent neurons. The synapses operate through the discharge of neurotransmitter chemicals across intercellular gaps, and can be either excitatory (tending to fire the next neuron) or inhibitory (tending to prevent firing of the next neuron).

### 1.1.1 Neuron Mathematical Model

A mathematical model of the neuron is depicted in Fig. 1.1.2, which shows the dendrite weights  $v_j$ , the firing threshold  $v_0$  (also called the ‘bias’), the summation of weighted incoming signals, and the nonlinear function  $\sigma(\cdot)$ . The cell inputs are the  $n$  time signals  $x_1(t), x_2(t), \dots, x_n(t)$  and the output is the scalar  $y(t)$ , which can

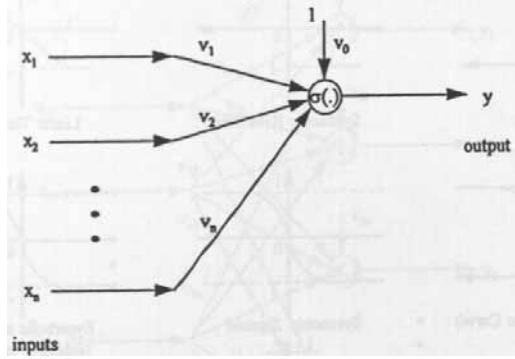


Figure 1.1.2: Mathematical model of a neuron.

be expressed as

$$y(t) = \sigma \left( \sum_{j=1}^n v_j x_j(t) + v_0 \right). \quad (1.1.1)$$

Positive weights  $v_j$  correspond to excitatory synapses and negative weights to inhibitory synapses. This was called the *perceptron* by Rosenblatt in 1959 (Haykin 1994).

The cell function  $\sigma(\cdot)$  is known as the *activation function*, and is selected differently in different applications; some common choices are shown in Fig. 1.1.3. The intent of the activation function is to model the behavior of the cell where there is no output below a certain value of the argument of  $\sigma(\cdot)$  and the output takes a specified magnitude above that value of the argument. A general class of monotonically nondecreasing functions taking on bounded values at  $-\infty$  and  $+\infty$  is known as the *sigmoid functions*. It is noted that, as the threshold or bias  $v_0$  changes, the activation functions shift left or right. For many training algorithms (including backpropagation), the derivative of  $\sigma(\cdot)$  is needed so that the activation function selected must be differentiable.

The expression for the neuron output  $y(t)$  can be streamlined by defining the column vector of input signals  $\bar{x}(t) \in \Re^n$  and the column vector of NN weights  $\bar{v}(t) \in \Re^n$  as

$$\bar{x}(t) = [x_1 \ x_2 \dots x_n]^T, \quad \bar{v}(t) = [v_1 \ v_2 \dots v_n]^T. \quad (1.1.2)$$

Then, one may write in matrix notation

$$y = \sigma(\bar{v}^T \bar{x} + v_0). \quad (1.1.3)$$

A final refinement is achieved by defining the augmented input column vector  $x(t) \in \Re^{n+1}$  and NN weight column vector  $v(t) \in \Re^{n+1}$  as

$$x(t) = [1 \ \bar{x}^T]^T = [1 \ x_1 \ x_2 \dots x_n]^T, \quad v(t) = [v_0 \ \bar{v}^T]^T = [v_0 \ v_1 \ v_2 \dots v_n]^T. \quad (1.1.4)$$

Then, one may write

$$y = \sigma(v^T x). \quad (1.1.5)$$

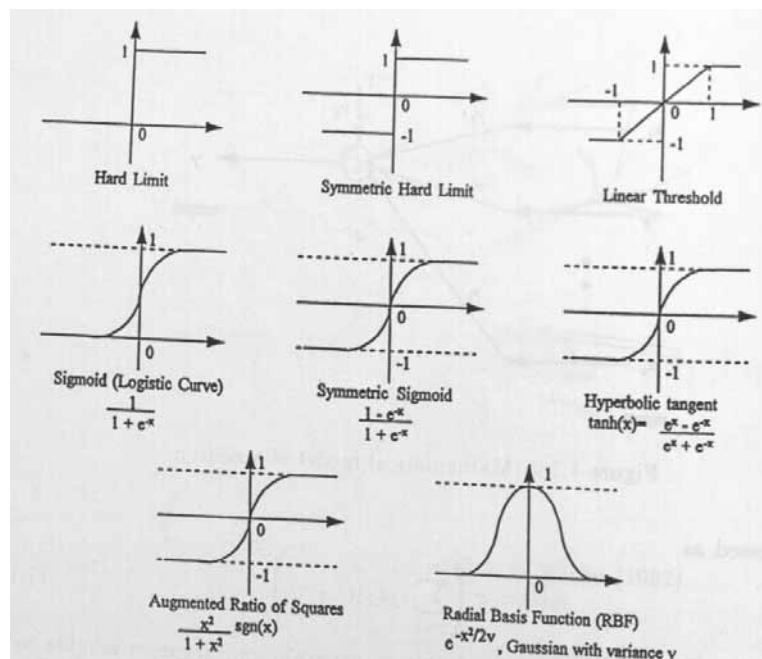


Figure 1.1.3: Some common choices for the activation function.

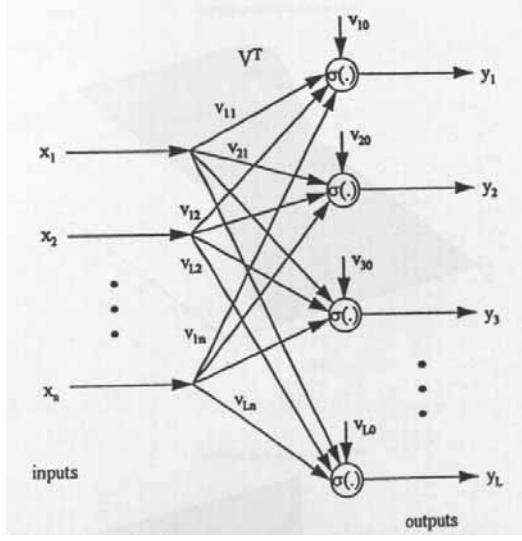


Figure 1.1.4: One-layer neural network.

Though the input vector  $\bar{x}(t) \in \Re^n$  and the vector of weights  $\bar{v} \in \Re^n$  have been augmented by 1 and  $v_0$  respectively to include the threshold, we may at times loosely say that  $x(t)$  and  $v$  are elements of  $\Re^n$ .

These expressions for the neuron output  $y(t)$  are referred to as the *cell recall mechanism*. They describe how the output is reconstructed from the input signals and the values of the cell parameters.

In Fig. 1.1.4 is shown a *neural network (NN)* consisting Of  $L$  cells, all fed by the same input signals  $x_j(t)$  and producing one output  $y_\ell(t)$  per neuron. We call this a *one-layer NN*. The recall equation for this network is given by

$$y_\ell = \sigma \left( \sum_{j=1}^n v_{\ell j} x_j + v_{\ell 0} \right); \quad \ell = 1, 2, \dots, L. \quad (1.1.6)$$

By defining the matrix of weights and the vector of thresholds as

$$\bar{V}^T \equiv \begin{bmatrix} v_{11} & v_{12} & \dots & v_{1n} \\ v_{21} & v_{22} & \dots & v_{2n} \\ \vdots & \vdots & & \vdots \\ v_{L1} & v_{L2} & \dots & v_{Ln} \end{bmatrix}, \quad b_v \equiv \begin{bmatrix} v_{10} \\ v_{20} \\ \vdots \\ v_{L0} \end{bmatrix}, \quad (1.1.7)$$

one may write the output vector  $y = [y_1 \ y_2 \ \dots \ y_L]^T$  as

$$y = \bar{\sigma}(\bar{V}^T \bar{x} + b_v). \quad (1.1.8)$$

The vector activation function is defined for a vector  $w \equiv [w_1 \ w_2 \ \dots \ w_L]^T$  as

$$\bar{\sigma}(w) \equiv [\sigma(w_1) \ \sigma(w_2) \ \dots \ \sigma(w_L)]^T. \quad (1.1.9)$$

A final refinement may be achieved by defining the augmented matrix of weights as

$$V^T = \begin{bmatrix} v_{10} & v_{11} & v_{12} & \dots & v_{1n} \\ v_{20} & v_{21} & v_{22} & \dots & v_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ v_{L0} & v_{L1} & v_{L2} & \dots & v_{Ln} \end{bmatrix}, \quad (1.1.10)$$

which contains the thresholds in the first column. Then, the NN outputs may be expressed in terms of the augmented input vector  $x(t)$  as

$$y = \bar{\sigma}(V^T x). \quad (1.1.11)$$

In other works (e.g. the Matlab NN Toolbox) the matrix of weights may be defined as the transpose of our version; our definition conforms more closely to usage in the control system literature.

### Example 1.1.1 (Output Surface for one-layer Neural Network) :

A perceptron with two inputs and one output is given by

$$y = \sigma(-4.79x_1 + 5.90x_2 - 0.93) \equiv \sigma(vx + b),$$

where  $v \equiv \bar{V}^T$ . Plots of the NN output surface  $y$  as a function of the inputs  $x_1, x_2$  over the grid  $[-2, 2] \times [-2, 2]$  are given in Fig. 1.1.5. Different outputs are shown corresponding to the use of different activation functions.

Though this plot of the output surface versus  $x$  is informative, it is not typically used in NN research. In fuzzy logic design it is known as the *reasoning surface* and is a standard tool. To make this plot the MATLAB NN Toolbox (1995) was used. The following sequence of commands was used:

```
% set up NN weights:  
v= [-4.79 5.9];  
b= [-0.93];  
% set up plotting grid for sampling x:  
[x1,x2]= meshgrid(-2 : 0.1 : 2);  
% compute NN input vectors p and simulate NN using sigmoid:  
p1= x1(:);  
p2= x2(:);  
p= [p1'; p2'];  
a= simuff(p,v,b,'sigmoid');  
% format results for using 'mesh' or 'surfl' plot routines:  
a1= eye(41);  
a1(:)= a';  
mesh(x1,x2,a1);
```

It is important to be aware of several factors in this MATLAB code: One should read the *MATLAB User's Guide* to become familiar with the use of the colon in matrix formatting. The prime on vectors or matrices (e.g.  $p1'$ ) means matrix transpose. The semicolon at the end of a command suppresses printing of the result. The symbol % means that the rest of the line is a comment. It is important to note that MATLAB defines the NN weight matrices as the *transposes* of our weight matrices; therefore, in all examples, the MATLAB convention is followed (we use lowercase letters here to help make the distinction). There are routines that compute the outputs of various NN given the inputs; in this instance SIMUFF() is used. The functions of 3-D plotting routines MESH and SURFL should be studied.  $\square$

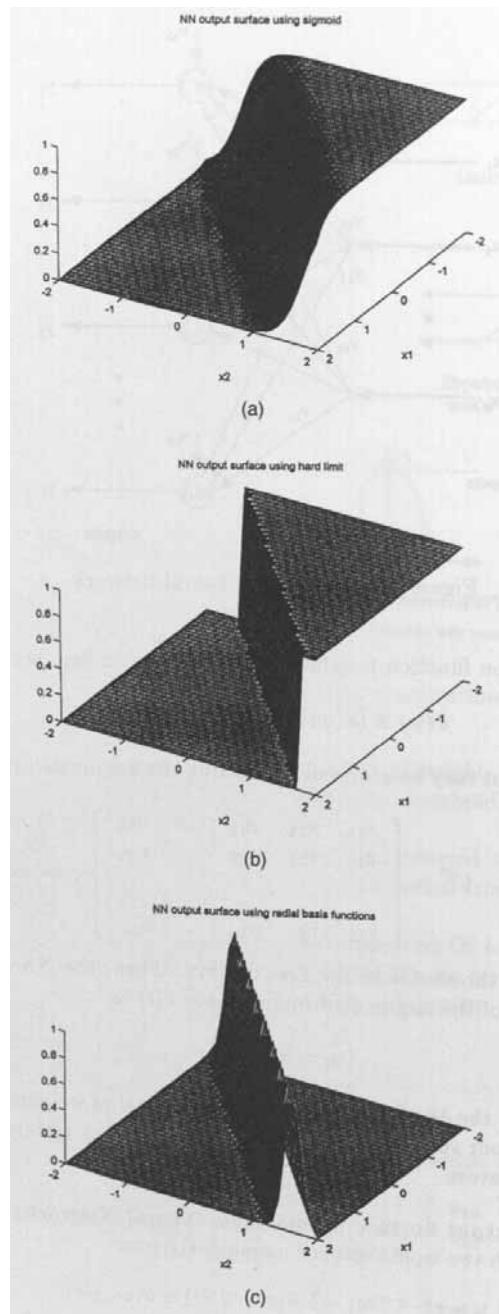


Figure 1.1.5: Output surface of a one-layer NN. (a) Using sigmoid activation function. (b) Using hard limit activation function. (c) Using radial basis function.

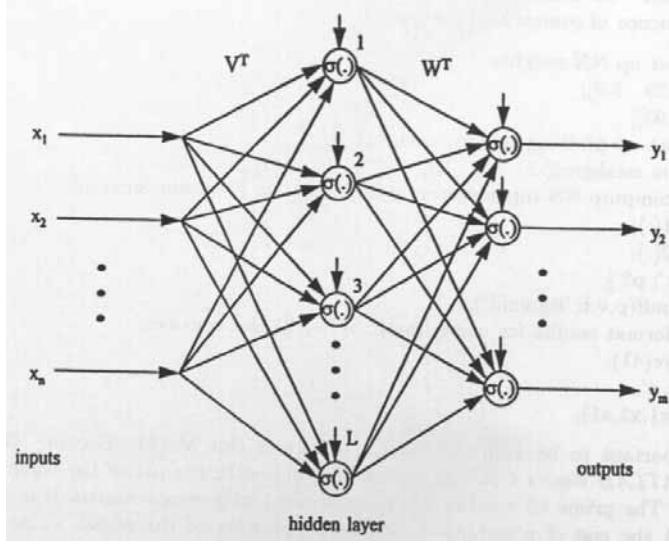


Figure 1.1.6: Two-layer neural network.

### 1.1.2 Multilayer Perceptron

A two-layer NN is depicted in Fig. 1.1.6, where there are two layers of neurons, with one layer having  $L$  neurons feeding into a second layer having  $m$  neurons. The first layer is known as the *hidden layer*, with  $L$  the *number of hidden-layer neurons*; the second layer is known as the *output layer*. NN with multiple layers are called *multilayer perceptrons*; their computing power is significantly enhanced over the one-layer NN. With one-layer NN it is possible to implement digital operations such as AND, OR, and COMPLEMENT (see Problems section). However, developments in NN were arrested many years ago when it was shown that the one-layer NN is incapable of performing the *EXCLUSIVE-OR* operation, which is basic to digital logic design. When it was realized that the two-layer NN can implement the EXCLUSIVE-OR (X-OR), NN research again accelerated. One solution to the X-OR operation is shown in Fig. 1.1.7, where sigmoid activation functions were used (Hush and Horne 1993).

The output of the two-layer NN is given by the recall equation

$$y_i = \sigma \left( \sum_{\ell=1}^L w_{i\ell} \sigma \left( \sum_{j=1}^n v_{\ell j} x_j + v_{\ell 0} \right) + w_{i0} \right) ; \quad i = 1, 2, \dots, m. \quad (1.1.12)$$

Defining the *hidden-layer outputs*  $z_\ell$  allows one to write

$$z_\ell = \sigma \left( \sum_{j=1}^n v_{\ell j} x_j + v_{\ell 0} \right) ; \quad \ell = 1, 2, \dots, L \quad (1.1.13)$$

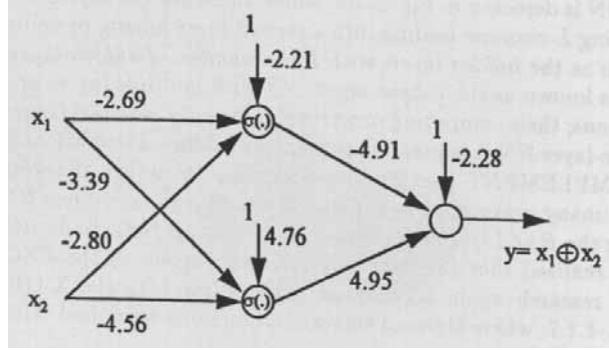


Figure 1.1.7: EXCLUSIVE-OR implemented using two-layer neural network.

$$y_i = \sigma \left( \sum_{\ell=1}^L w_{i\ell} z_\ell + w_{i0} \right) ; \quad i = 1, 2, \dots, m. \quad (1.1.14)$$

Defining first-layer weight matrices  $\bar{V}$  and  $V$  as in the previous subsection, and second-layer weight matrices as

$$\begin{aligned} \bar{W}^T &\equiv \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1L} \\ w_{21} & w_{22} & \dots & w_{2L} \\ \vdots & \vdots & & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mL} \end{bmatrix}, \quad b_w \equiv \begin{bmatrix} w_{10} \\ w_{20} \\ \vdots \\ w_{m0} \end{bmatrix}, \\ W^T &= \begin{bmatrix} w_{10} & w_{11} & w_{12} & \dots & w_{1L} \\ w_{20} & w_{21} & w_{22} & \dots & w_{2L} \\ \vdots & \vdots & \vdots & & \vdots \\ w_{m0} & w_{m1} & w_{m2} & \dots & w_{mL} \end{bmatrix}, \end{aligned} \quad (1.1.15)$$

one may write the NN output as

$$y = \bar{\sigma}(\bar{W}^T \bar{\sigma}(\bar{V}^T \bar{x} + b_v) + b_w) \quad (1.1.16)$$

or, in streamlined form as

$$y = \bar{\sigma}(W^T \sigma(V^T x)). \quad (1.1.17)$$

In these equations, the notation  $\bar{\sigma}$  means the vector defined in accordance with (1.1.9). In (1.1.17) it is necessary to use the augmented vector

$$\sigma(w) \equiv [1 \ \bar{\sigma}(w)^T]^T = [1 \ \sigma(w_1) \ \sigma(w_2) \dots \sigma(w_L)]^T; \quad (1.1.18)$$

where a 1 is placed as the first entry, to allow the incorporation of the thresholds  $w_{i0}$  as the first column of  $W^T$ . In terms of the hidden-layer output vector  $z \in \Re^L$  one may write

$$\bar{z} = \bar{\sigma}(V^T x) \quad (1.1.19)$$

$$y = \sigma(W^T z), \quad (1.1.20)$$

where  $z \equiv [1 \ z^T]^T$ .

In the remainder of this book we shall not show the overbar on vectors—the reader will be able to determine by the context whether the leading ‘1’ is required. We shall generally be concerned in later chapters with two-layer NN with linear activation functions on the output layer, so that

$$y = W^T \sigma(V^T x). \quad (1.1.21)$$

#### Example 1.1.2 (Output Surface for Two-Layer Neural Network) :

A two-layer NN with two inputs and one output is given by

$$y = \bar{W}^T \sigma(\bar{V}^T x + b_v) + b_w \equiv w\sigma(vx + b_v) + b_w$$

with weight matrices and thresholds given by

$$v = \bar{V}^T = \begin{bmatrix} -2.69 & -2.80 \\ -3.39 & -4.56 \end{bmatrix}, \quad b_v = \begin{bmatrix} -2.21 \\ 4.76 \end{bmatrix}$$

$$w = \bar{W}^T = [-4.91 \ 4.95], \quad b_w = [-2.28].$$

Plots of the NN output surface  $y$  as a function of the inputs  $x_1, x_2$  over the grid  $[-2, 2] \times [-2, 2]$  are given in Fig. 1.1.8. Different outputs are shown corresponding to the use of different activation functions. To make this plot the MATLAB NN Toolbox (1995) was used with the sequence of commands given in Example 1.1.1.

Of course, this is the EXCLUSIVE-OR network from Fig. 1.1.7. The X-OR is normally used only with binary input vectors  $x$ , however, plotting the NN output surface over a region of values for  $x$  reveals graphically the decision boundaries of the network and aids in visualization.  $\square$

#### 1.1.3 Linear-in-the-Parameter (LIP) Neural Nets

If the first-layer weights and thresholds  $V$  in (1.1.21) are predetermined by some *a priori* method, then only the second-layer weights and thresholds  $W$  are considered to define the NN, so that the NN has only one layer of weights. One may then define the fixed function  $\phi(x) \equiv \sigma(V^T x)$  so that such a one-layer NN has the recall equation

$$y = W^T \phi(x), \quad (1.1.22)$$

where  $x \in \Re^n$  (recall that technically  $x$  is augmented by a ‘1’),  $y \in \Re^m$ ,  $\phi(\cdot) : \Re^n \rightarrow \Re^L$ , and  $L$  is the number of hidden-layer neurons. This NN is *linear* in the NN parameters  $W$ , so that it is far easier to deal with in subsequent chapters. Specifically, it is easier to train the NN by tuning the weights. This one-layer having only output-layer weights  $W$  should be contrasted with the one-layer NN discussed in (1.1.11), which had only input-layer weights  $V$ .

More generality is gained if  $\sigma(\cdot)$  is not diagonal, e.g. as defined in (1.1.9), but  $\phi(\cdot)$  is allowed to be a general function from  $\Re^n$  to  $\Re^L$ . This is called a *functional-link neural net (FLNN)* (Sadegh 1993). Some special FLNN are now discussed. We often use  $\sigma(\cdot)$  in place of  $\phi(\cdot)$ , with the understanding that, for LIP nets, this activation function vector is not diagonal, but is a general function from  $\Re^n$  to  $\Re^L$ .

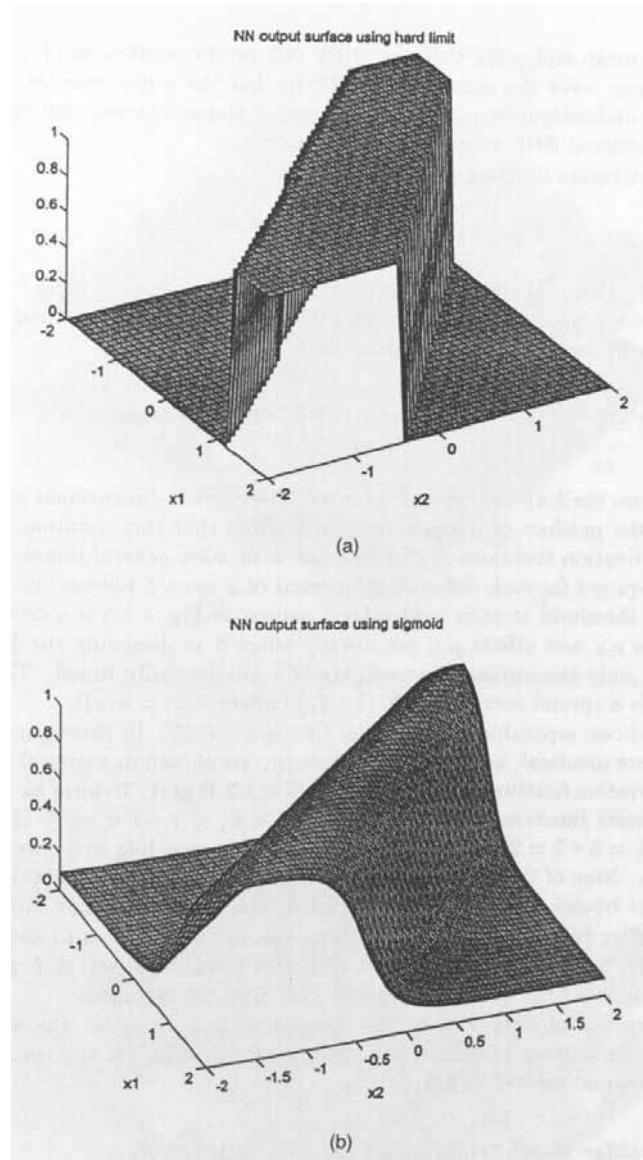


Figure 1.1.8: Output surface of a two-layer NN. (a) Using sigmoid activation function. (b) Using hard limit activation function.

### 1.1.3.1 Gaussian or Radial Basis Function (RBF) Networks

The selection of a suitable set of activation functions is considerably simplified in various sorts of *structured nonlinear networks*, including radial basis function, CMAC, and fuzzy logic nets. It will be shown here that the key to the design of such structured nonlinear nets lies in a *more general set of NN thresholds* than allowed in the standard equation (1.1.12), and in their appropriate selection.

A NN activation function often used is the gaussian or radial basis function (RBF) (Sanner and Slotine 1991) given when  $x$  is a scalar as

$$\sigma(x) = e^{-(x-\mu)^2/2p}, \quad (1.1.23)$$

where  $\mu$  is the mean and  $p$  the variance. RBF NN can be written as (1.1.21), but have an advantage over the usual sigmoid NN in that the  $n$ -dimensional gaussian function is well understood from probability theory, Kalman filtering, and elsewhere, so that  $n$ -dimensional RBF are easy to conceptualize.

The  $j$ -th activation function can be written as

$$\sigma_j(x) = e^{-\frac{1}{2}(x-\mu_j)^T P_j^{-1}(x-\mu_j)} \quad (1.1.24)$$

with  $x, \mu_j \in \Re^n$ . Define the vector of activation functions as  $\sigma(x) \equiv [\sigma_1(x) \ \sigma_2(x) \ \dots \ \sigma_L(x)]^T$ . If the covariance matrix is diagonal so that  $P_j = \text{diag}\{p_{jk}\}$ , then (1.1.24) becomes *separable* and may be decomposed into components as

$$\sigma_j(x) = e^{-\frac{1}{2} \sum_{k=1}^n -(x_k - \mu_{jk})^2 / p_{jk}} = \prod_{k=1}^n e^{-\frac{1}{2} (x_k - \mu_{jk})^2 / p_{jk}}, \quad (1.1.25)$$

where  $x_k, \mu_{jk}$  are the  $k$ -th components of  $x, \mu_j$ . Thus, the  $n$ -dimensional activation functions are the product of  $n$  scalar functions. Note that this equation is of the form of the activation functions in (1.1.12), but with *more general thresholds*, as a threshold is required for each different component of  $x$  at each hidden layer neuron  $j$ ; that is, the threshold at each hidden-layer neuron in Fig. 1.1.6 is a *vector*. The RBF variances  $p_{jk}$  and offsets  $\mu_{jk}$  are usually selected in designing the RBF NN and left fixed; only the output-layer weights  $W^T$  are generally tuned. Therefore, the RBF NN is a special sort of FLNN (1.1.22) (where  $\phi(x) = \sigma(x)$ ).

Fig. 1.1.9 shows separable gaussians for the case  $x \in \Re^2$ . In this figure, all the variances  $p_{jk}$  are identical, and the mean values  $\mu_{jk}$  are chosen in a special way that spaces the activation functions at the node points of a 2-D grid. To form an RBF NN that approximates functions over the region  $\{-1 < x_1 \leq 1, -1 < x_2 \leq 1\}$  one has here selected  $L = 5 \times 5 = 25$  hidden-layer neurons, corresponding to 5 cells along  $x_1$  and 5 along  $x_2$ . Nine of these neurons have 2-D gaussian activation functions, while those along the boundary require the illustrated ‘one-sided’ activation functions.

An alternative to selecting the gaussian means and variances is to use random choice. In 2-D, for instance (c.f. Fig. 1.1.9), this produces a set of  $L$  gaussians scattered at random over the  $(x_1, x_2)$  plane with different variances.

The importance of RBF NN is that they show how to select the activation functions and the number of hidden-layer neurons for specific NN applications (e.g. function approximation— see below).

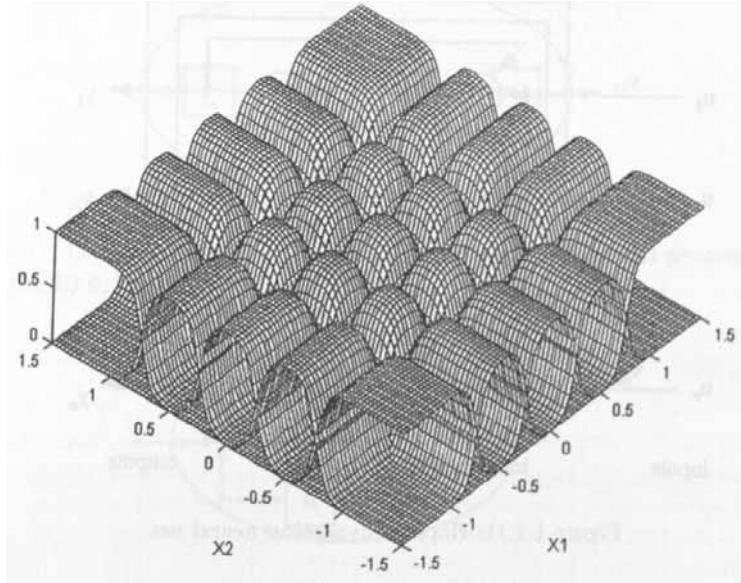


Figure 1.1.9: Two-dimensional separable gaussian functions for an RBF NN.

#### 1.1.3.2 Cerebellar Model Articulation Controller (CMAC) Nets

A CMAC NN (Albus 1975) has separable activation functions generally composed of splines. The activation functions of a 2-D CMAC composed of second-order splines (e.g. triangle functions) are shown in Fig. 1.1.10, where  $L = 5 \times 5 = 25$ . The activation functions of a CMAC NN are called *receptive field functions* in analogy with the optical receptor fields of the eye. An advantage of CMAC NN is that the receptive field functions based on splines have *finite support* so that they may be efficiently evaluated. An additional computational advantage is provided by the fact that higher-order splines may be computed recursively from lower-order splines.

#### 1.1.4 Dynamic Neural Networks

The NN we have discussed so far are *nondynamic* in that they contain no memory—that is, no integrators or time-delay elements. There are many sorts of *dynamic NN*, or recurrent NN, where some signals in the NN are either integrated or delayed and fed back into the net. The seminal work of Narendra and co-workers (see References) should be explored for more details.

##### 1.1.4.1 Hopfield Net

Perhaps the most familiar dynamic NN is the *Hopfield net*, shown in Fig. 1.1.11, a special form of two-layer NN where the output  $y_i$  is fed back into the hidden-layer neurons (Haykin 1994). In the Hopfield net, the first-layer weight matrix  $V$  is the identity matrix  $I$ , the second-layer weight matrix  $W$  is square, and the output-layer

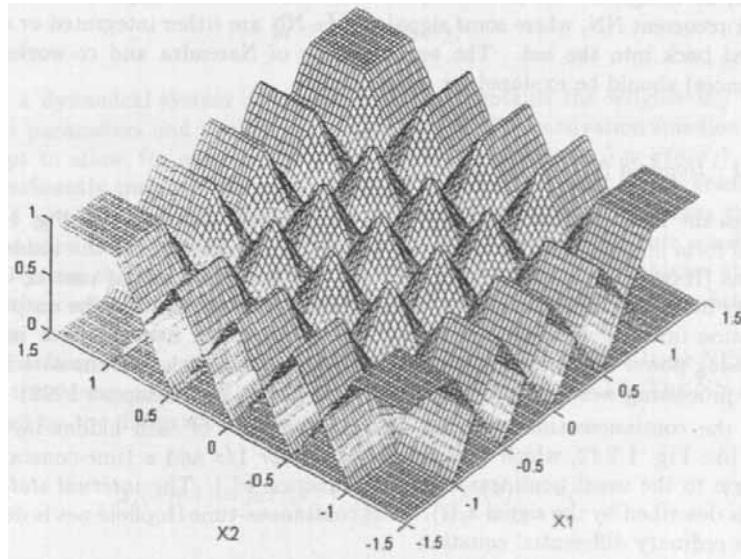


Figure 1.1.10: Receptive field functions for a 2-D CMAC NN with second-order splines.

activation function is linear. Moreover, the hidden-layer neurons have increased processing power in the form of a *memory*. We may call such neurons with internal signal processing *neuronal processing elements* (*NPE*) (c.f. Simpson 1992).

In the continuous-time case the *internal dynamics* of each hidden-layer NPE looks like Fig. 1.1.12, which contains an *integrator*  $1/s$  and a time-constant  $\tau_i$  in addition to the usual nonlinear activation function  $\sigma(\cdot)$ . The *internal state* of the NPE is described by the signal  $x_i(t)$ . The continuous-time Hopfield net is described by the ordinary differential equation

$$\tau_i \dot{x}_i = -x_i + \sum_{j=1}^n w_{ij} \sigma_j(x_j) + u_i \quad (1.1.26)$$

with output equation

$$y_i = \sum_{j=1}^n w_{ij} \sigma_j(x_j). \quad (1.1.27)$$

This is a dynamical system of special form that contains the weights  $w_{ij}$  as adjustable parameters and positive time constants  $\tau_i$ . The activation function has a subscript to allow, for instance, for scaling terms  $g_j$  as in  $\sigma_j(x_j) \equiv \sigma(g_j x_j)$ , which can significantly improve the performance of the Hopfield net. In the traditional Hopfield net the threshold offsets  $u_i$  are constant bias terms. The reason that we have renamed the offsets as  $u_i$  is that (1.1.26) has the form of a *state equation* in control system theory, where the internal state is labeled  $x(t)$ . The biases play the role of the control input term, which is labeled  $u(t)$ . In traditional Hopfield NN, the term ‘input pattern’ refers to the initial state components  $x_i(0)$ .

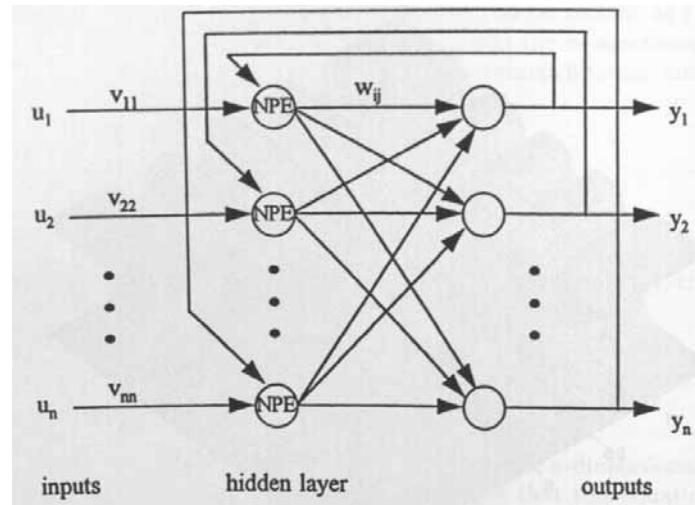


Figure 1.1.11: Hopfield dynamical neural net.

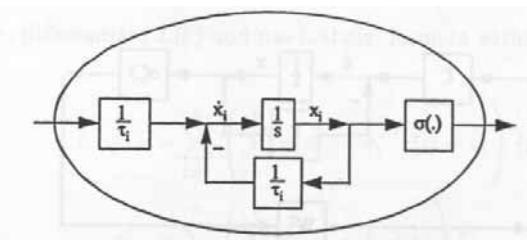


Figure 1.1.12: Continuous-time Hopfield net hidden-layer neuronal processing element (NPE) dynamics.

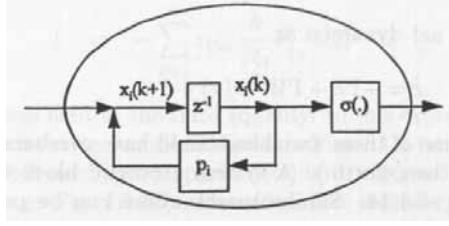


Figure 1.1.13: Discrete-time Hopfield net hidden-layer NPE dynamics.

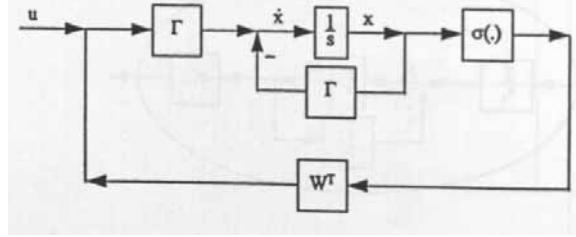


Figure 1.1.14: Continuous-time Hopfield net in block diagram form.

In the discrete-time case, the internal dynamics of each hidden-layer NPE contains a time delay instead of an integrator, as shown in Fig. 1.1.13. The NN is now described by the difference equation

$$x_i(k+1) = p_i x_i(k) + \sum_{j=1}^n w_{ij} \sigma_j(x_j(k)) + u_i(k), \quad (1.1.28)$$

with  $p_i < 1$ . This is a discrete-time dynamical system with time index  $k$ .

Defining the NN weight matrix  $W^T$ , vectors  $x \equiv [x_1 \ x_2 \dots x_n]^T$  and  $u \equiv [u_1 \ u_2 \dots u_n]^T$ , and the matrix  $\Gamma = \text{diag}\left\{\frac{1}{\tau_1}, \frac{1}{\tau_2}, \dots, \frac{1}{\tau_n}\right\}$ , one may write the continuous-time Hopfield net dynamics as

$$\dot{x} = -\Gamma x + \Gamma W^T \sigma(x) + \Gamma u. \quad (1.1.29)$$

(Note that technically some of these variables should have overbars. We shall generally drop the overbars henceforth.) A system theoretic block diagram of this dynamics is given in Fig. 1.1.14. Similar machinations can be performed in the discrete-time case (see Problems section).

#### 1.1.4.2 Hopfield Net Energy Function and Stability

The Hopfield net can be used as a classifier/pattern recognizer (see Section 1.2.1). In the original form used by Hopfield, the diagonal entries  $w_{ii}$  were set equal to zero and it was shown that, as long as the weight matrix  $[w_{ij}]$  is symmetric, the NN converges to stable local equilibrium points depending on the values of the initial state  $x(0)$ . These stable local equilibrium points represent the *information stored*

*in the net*, that is, the exemplar patterns to be recognized, and are determined by the values  $w_{ij}$ . Techniques for selecting the weights for a specific set of exemplar patterns are given in Section 1.3.1.

To show that the state of the Hopfield net is stable, converging to bounded values, one may use the notion of the *Lyapunov function* of a dynamic system from Chapter 2. Thus, consider the continuous-time Hopfield net and define the intermediate vector  $\xi \in \Re^n$  with components

$$\xi_i \equiv \sigma_i(x_i). \quad (1.1.30)$$

Associate to the NN an *energy function* or Lyapunov function

$$L(\xi) = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} \xi_i \xi_j + \sum_{i=1}^n \int_0^{\xi_i} \sigma_i^{-1}(z) dz - \sum_{i=1}^n u_i \xi_i. \quad (1.1.31)$$

It can be shown that  $L(\xi)$  is bounded below. Thus, if one can prove that its derivative is negative, then this energy function is decreasing, so that the energy, and hence also the magnitude of  $\xi$ , decreases. This will demonstrate stability of the state.

Therefore, differentiate  $L(\xi)$  and use Leibniz' formula with the integral term to obtain

$$\begin{aligned} \dot{L} &= -\sum_{i=1}^n \left( \sum_{j=1}^n w_{ij} \xi_j - \sigma_i^{-1}(\xi_i) + u_i \right) \dot{\xi}_i \\ &= -\sum_{i=1}^n \left( \sum_{j=1}^n w_{ij} \xi_j - x_i + u_i \right) \dot{\xi}_i \\ &= -\sum_{i=1}^n (\tau_i \dot{x}_i) \dot{\xi}_i \\ \dot{L} &= -\sum_{i=1}^n \tau_i \dot{\xi}_i^2 \frac{\partial}{\partial \xi_i} \sigma_i^{-1}(\xi_i) \end{aligned} \quad (1.1.32)$$

where (1.1.26) was used at the third equality. In this expression, one notes that, for all continuous activation functions shown in Fig. 1.1.3 (with the exception of RBF, which are not often used with Hopfield nets) both  $\sigma_i(\cdot)$  and  $\sigma_i^{-1}(\cdot)$  are monotonically increasing (see e.g. Fig. 1.1.15). Therefore,  $\dot{L}$  is negative and the Hopfield state is stable.

**Example 1.1.3 (Dynamics and Lyapunov Surface of Hopfield Network) :**

Select  $x = [x_1 \ x_2]^T \in \Re^2$  and choose parameters so that the Hopfield net is

$$\dot{x} = -\frac{1}{2}x + \frac{1}{2}W^T \sigma(x) + \frac{1}{2}u$$

with weight matrix

$$W = W^T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Select the symmetric sigmoid activation function in Fig. 1.1.3 so that

$$\xi_i = \sigma_i(x_i) \equiv \sigma(g_i x_i) = \frac{1 - e^{-g_i x_i}}{1 + e^{-g_i x_i}}.$$

Then,

$$x_i = \sigma_i^{-1}(\xi_i) = -\frac{1}{g_i} \ln \left( \frac{1 - \xi_i}{1 + \xi_i} \right).$$

Using sigmoid decay constants of  $g_1 = g_2 = 100$ , these functions are plotted in Fig. 1.1.15.

#### a. State Trajectory Phase-Plane Plots

State trajectory phase-plane plots for various initial condition vectors  $x(0)$  and  $u = 0$  are shown in Fig. 1.1.16, which plots  $x_2(t)$  vs.  $x_1(t)$ . All initial conditions converge to the vicinity of either point  $(-1, -1)$  or point  $(1, 1)$ . As seen in Section 1.3.1, these are the exemplar patterns stored in the weight matrix  $W$ . Techniques for selecting the weights for desired performance are given in Section 1.3.1.

The state trajectories are plotted with MATLAB using the function ‘ode23(·)’, which requires the following M file to describe the system dynamics:

```
% hopfield.m: Matlab M file for Hopfield net dynamics
function xdot= hopfield(t,x)
g=100;
tau= 2;
u= [0 ; 0];
w= [0 1
     1 0];
xi= (1-exp(-g*x)) ./ (1+exp(-g*x));
xdot= (-x + w*v + u)/tau;
```

In MATLAB, an operator preceded by a period denotes the element-by-element matrix operation; thus ‘./’ denotes element-by-element vector division.

#### b. Lyapunov Energy Surface

To investigate the time history behavior, examine the Lyapunov energy function (1.1.31) for this system. Using the inverse of the symmetric sigmoid activation function one computes

$$L(\xi) = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} \xi_i \xi_j - \sum_{i=1}^n u_i \xi_i + \frac{1}{g_i} \sum_{i=1}^n [(1 + \xi_i) \ln(1 + \xi_i) + (1 - \xi_i) \ln(1 - \xi_i)].$$

This surface is plotted versus  $x_1, x_2$  in Fig. 1.1.17 where it is clearly seen that the minima over the region  $[-1, 1] \times [-1, 1]$  occur near the points  $(-1, -1)$  and  $(1, 1)$ . This explains the behavior seen in the time histories; the state trajectories are such that  $x(t)$  moves downhill on the Lyapunov surface at each time instant. The exact equilibrium points may be computed by solving the equation  $\dot{x} = 0$ .  $\square$

#### 1.1.4.3 Generalized Recurrent Neural Network

A generalized dynamical NN is shown in Fig. 1.1.18 (c.f. work of Narendra, see References). In this figure,  $H(s) = C(sI - A)^{-1}B$  represents the transfer function of a linear dynamical system or plant given by

$$\begin{aligned} \dot{x} &= Ax + B\mu \\ \zeta &= Cx \end{aligned} \tag{1.1.33}$$

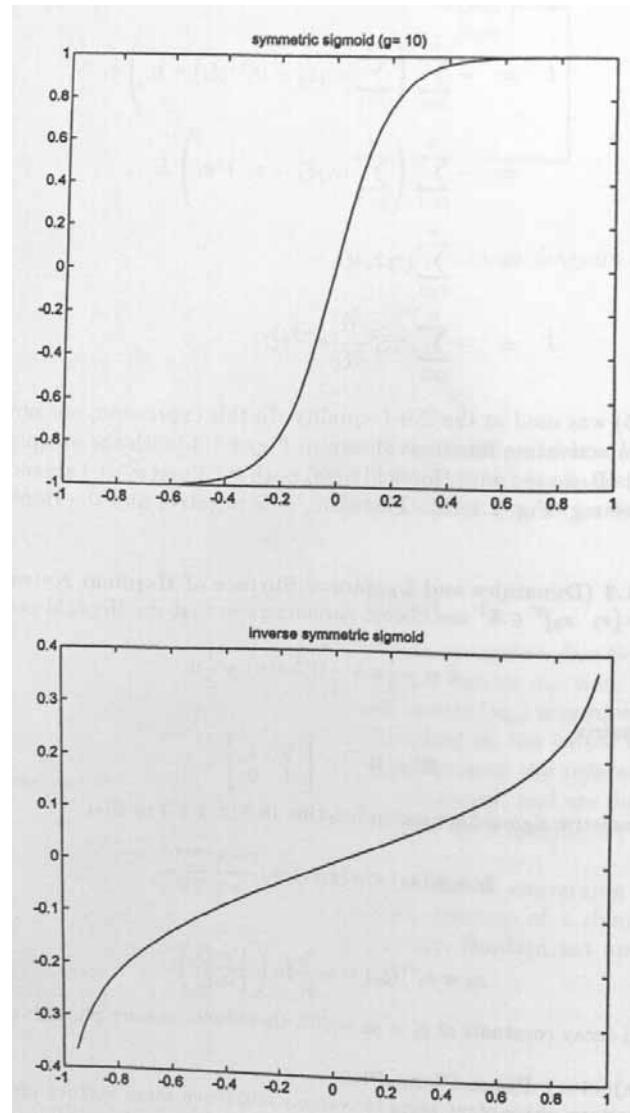


Figure 1.1.15: Hopfield net functions. (a) Symmetric sigmoid activation function.  
(b) Inverse of symmetric sigmoid activation function.

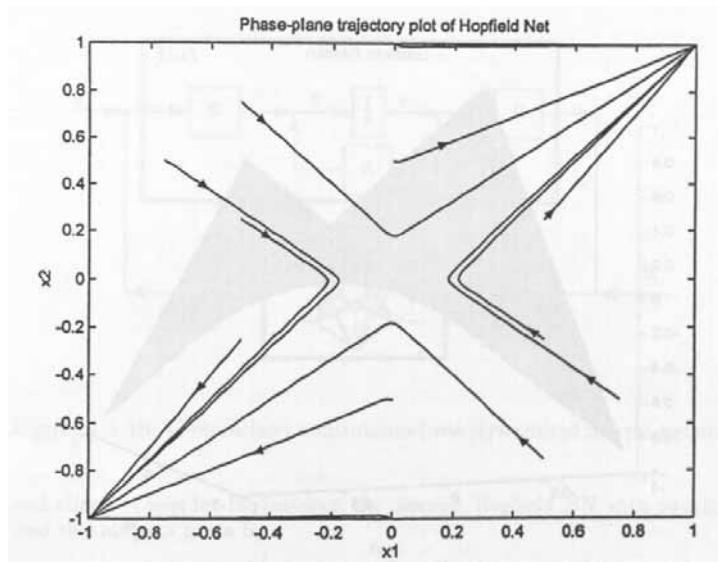


Figure 1.1.16: Hopfield net phase-plane plots;  $x_2(t)$  versus  $x_1(t)$ .

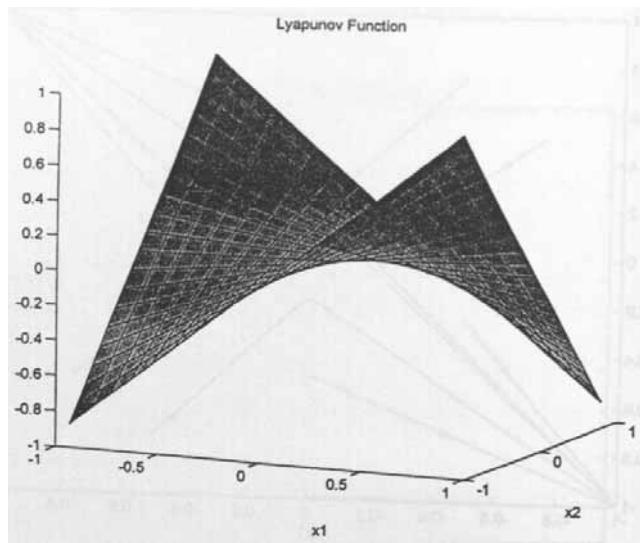


Figure 1.1.17: Lyapunov energy surface for an illustrative Hopfield net.

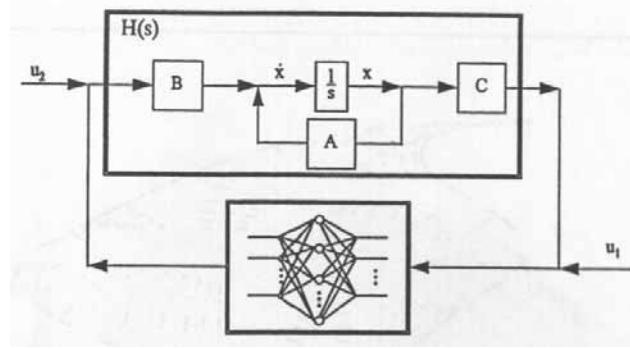


Figure 1.1.18: Generalized continuous-time dynamical neural network.

with internal state  $x(t) \in \mathbb{R}^n$ , control input  $\mu(t)$ , and output  $\zeta(t)$ . The NN can be a two-layer net described by (1.1.12), (1.1.16), (1.1.17). This dynamic NN is described by the equation

$$\dot{x} = Ax + B[\sigma(W^T\sigma(V^T(Cx + u_1)))] + Bu_2. \quad (1.1.34)$$

From examination of (1.1.29) it is plain that the Hopfield net is a special case of this equation, which is also true of many other dynamical NN in the literature. A similar version holds for the discrete-time case. If the system matrices  $A, B, C$  are diagonal, then the dynamics can be interpreted as residing *within the neurons*, and one can speak of neuronal processing elements with increased computing power and internal memory. Otherwise, there are additional dynamical interconnections around the NN as a whole (see the problems).

**Example 1.1.4 (Chaotic Behavior of Neural Networks) :**

This example was provided by Professor Chaouki Abdallah (1995) of the University of New Mexico based on some discussions in Becker and Dörfler (1988). Even in simple neural networks it is possible to observe some very interesting behavior, including limit cycles and chaos. Consider for instance the discrete Hopfield NN with two inputs, two states, and two outputs given by

$$x_{k+1} = Ax_k + W^T\sigma(V^T x_k) + u_k,$$

which is a discrete-time system of the form (1.1.34).

**a. Starfish Attractor— Changing the NN Weights**

Select the system matrices as

$$A = \begin{bmatrix} -0.1 & 1 \\ -1 & 0.1 \end{bmatrix}, \quad w = W^T = \begin{bmatrix} \pi & 1 \\ 1 & -1 \end{bmatrix}, \quad v = V^T = \begin{bmatrix} 1.23456 & 2.23456 \\ 1.23456 & 2.23456 \end{bmatrix}$$

and the input as  $u_k = [1 \ 1]^T$ .

It is straightforward to simulate the time performance of the DT Hopfield system using the following MATLAB code.

```
% MATLAB function file for simulation of discrete Hopfield NN
function [x,y] = starfish(N)
x1(1)=rand;
x2(1)=rand;
a11= -0.1; a12= 1;
a21= -1; a22= 0.1;
w11= pi; w12= 1;
w21= 1; w22= -1;
u1= 1;
u2= -1;
v11= 1.23456; v12= 2.23456;
v21= 1.23456; v22= 2.23456;
for k=1:N
    x1(k+1)=a11*x1(k) + a12*x2(k) + w11*tanh(v11*x1(k)) + w12*tanh(v12*x2(k))
    + u1;
    x2(k+1)=a21*x1(k) + a22*x2(k) + w21*tanh(v21*x1(k)) + w22*tanh(v22*x2(k))
    + u2;
end
end
```

where the argument  $N$  is the number of time iterations to be performed. The system is initialized at random initial state  $x_0$ . The tanh activation function is used.

The result of the simulation is plotted using the MATLAB function `plot(x1,x2,'.)`; it is shown for  $N= 2000$  points in Fig. 1.1.19. After an initial transient, the time history is attracted to a shape reminiscent of a starfish. Lyapunov exponent analysis allows the determination of the dimension of the attractor. If the attractor has non-integer dimension, it is said to be a *strange attractor* and the NN exhibits chaos.

Changing the NN weight matrices results in different behavior. Setting

$$v = V^T = \begin{bmatrix} 2 & 3 \\ 2 & 3 \end{bmatrix}$$

yields the plot shown in Fig. 1.1.20. It is very easy to destroy the chaotic-type behavior. For instance, setting

$$v = V^T = \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}$$

yields the plot shown in Fig. 1.1.21, where the attractor is a stable limit cycle.

### b. Anemone Attractor— Changing the Plant $A$ Matrix

Changes in the plant matrices  $(A, B, C)$  also influence the characteristics of the attractor. Setting

$$A = \begin{bmatrix} 1 & 1 \\ -1 & 0.1 \end{bmatrix}$$

yields the phase-plane plot shown in Fig. 1.1.22. Also changing the NN first-layer weight matrix to

$$v = V^T = \begin{bmatrix} 2 & 3 \\ 2 & 3 \end{bmatrix}$$

yields the behavior shown in Fig. 1.1.23. □

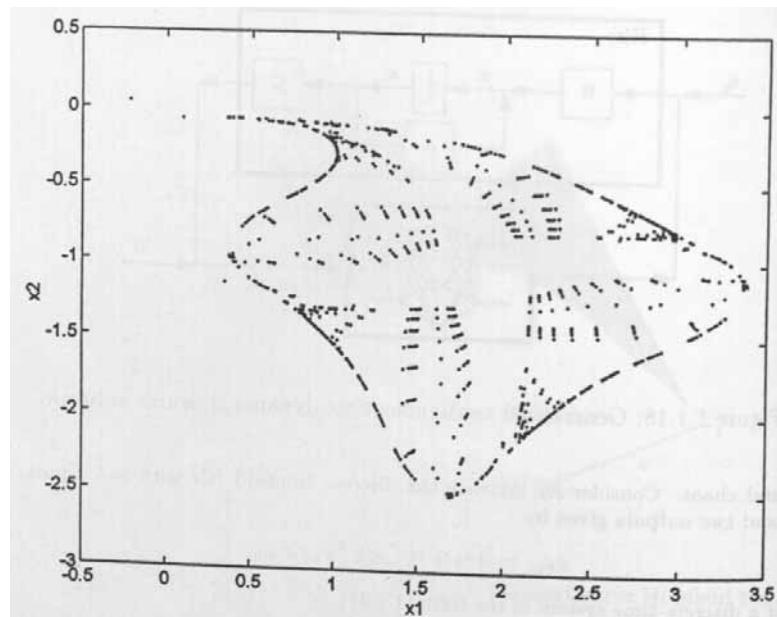


Figure 1.1.19: Phase-plane plot of discrete-time NN showing attractor.

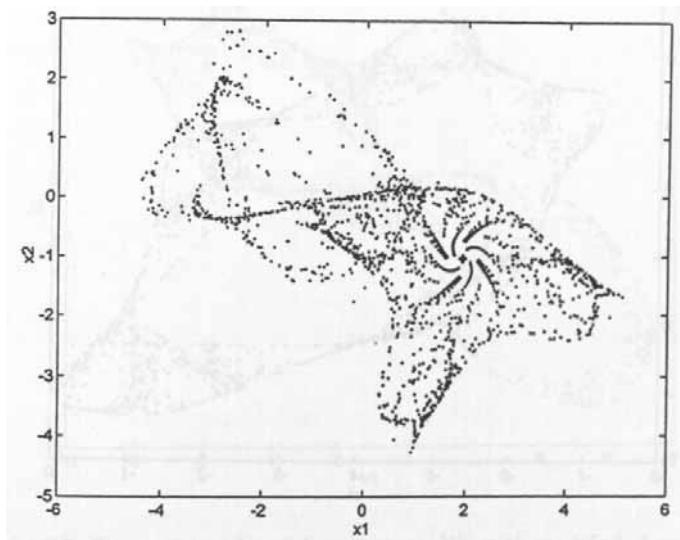


Figure 1.1.20: Phase-plane plot of discrete-time NN with modified  $V$  weights.

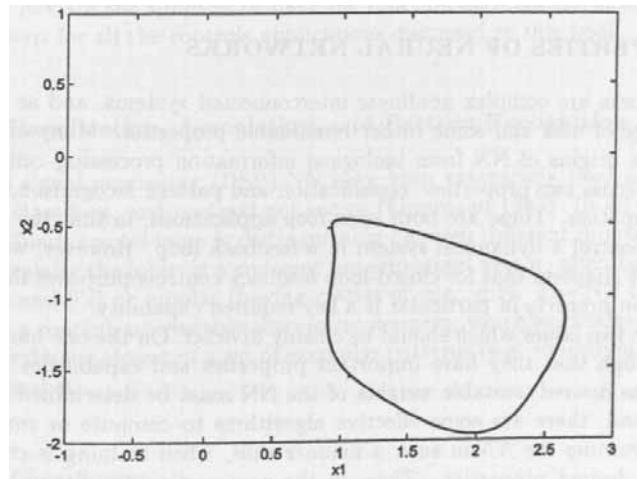


Figure 1.1.21: Phase-plane plot of discrete-time NN with modified  $V$  weights showing limit-cycle attractor.

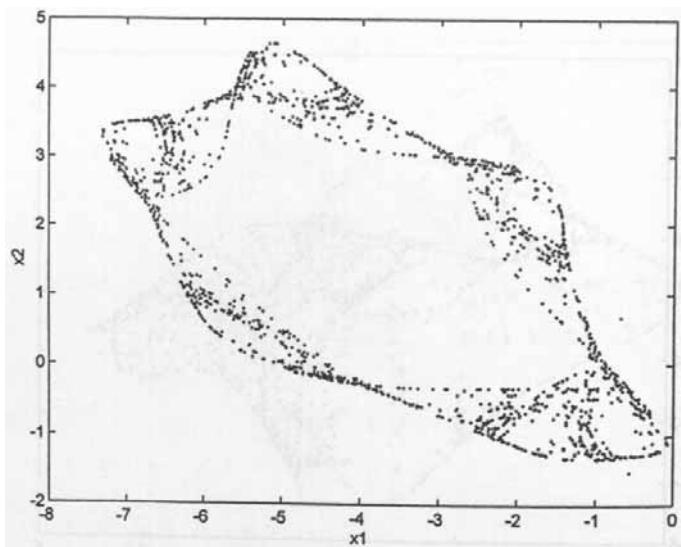


Figure 1.1.22: Phase-plane plot of discrete-time NN with modified  $A$  matrix.

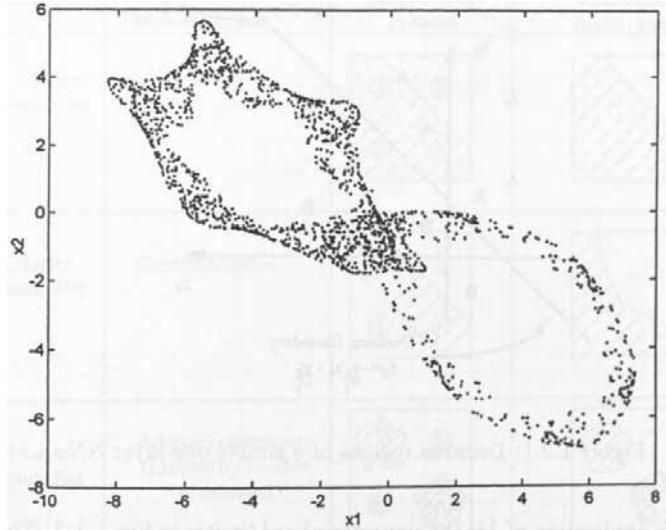


Figure 1.1.23: Phase-plane plot of discrete-time NN with modified  $A$  matrix and  $V$  matrix.

## 1.2 PROPERTIES OF NEURAL NETWORKS

Neural networks are complex nonlinear interconnected systems, and as such have a broad range of uses and some rather remarkable properties. Many of these derive from the origins of NN from biological information processing cells. In this section we discuss two properties: classification and pattern recognition, and function approximation. These are both *open-loop* applications, in that the NN is not required to control a dynamical system in a feedback loop. However, we shall see in subsequent chapters that for closed-loop feedback controls purposes the *function approximation property* in particular is a key required capability.

There are two issues which should be cleanly divided. On the one hand, NN are complex enough that they have important properties and capabilities. However, to function as desired, suitable weights of the NN must be determined. Thus, on the other hand, there are some effective algorithms to compute or *tune the NN weights by training the NN* in such a manner that, when training is complete, it exhibits the desired properties. Thus, in the next section we discuss techniques of weight selection and tuning so that the NN performs as a classifier, a function approximator, and so on.

It is finally emphasized that, though it is possible to construct NN with multiple hidden layers, the computational burden increases with the number of hidden layers. A NN with two hidden layers (three-layer net) can form the most complex decision regions for classification. However, in many practical situations it is usually found that *the two-layer NN (e.g. with one hidden layer) is sufficient*. Specifically, since two-layer NN are the simplest to have the function approximation capability, they are sufficient for all the controls applications discussed in this book.

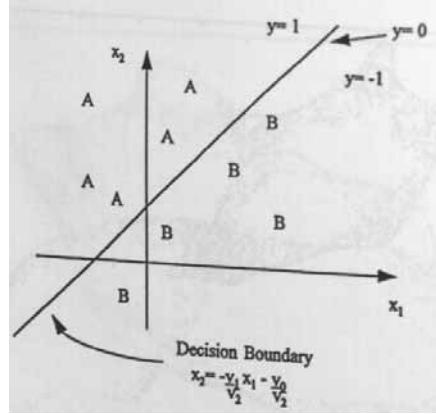


Figure 1.2.1: Decision regions of a simple one-layer NN.

### 1.2.1 Classification, Association, and Pattern Recognition

In digital signal processing (DSP) NN have been extensively used as pattern recognizers, classifiers, and contrast enhancers (Lippmann 1987). In all these applications the fundamental issue is distinguishing between different inputs presented to the NN; usually the input is a *constant* time-invariant vector, often binary (consisting of 1's and 0's) or bipolar (having entries of, e.g.,  $\pm 1$ ). The NN in such uses is known as a content-addressable *associative memory*, which associates various input patterns with the closest of a set of exemplar patterns (e.g. identifying noisy letters of the alphabet).

#### 1.2.1.1 Classification

A one-layer NN with two inputs  $x_1, x_2$  and one output is given by

$$y = \sigma(v_0 + v_1 x_1 + v_2 x_2), \quad (1.2.1)$$

where in this application  $\sigma(\cdot)$  is the symmetric hard limiter in Fig. 1.1.3. The output can take on values of  $\pm 1$ . When  $y$  is zero, there holds the relation

$$\begin{aligned} 0 &= v_0 + v_1 x_1 + v_2 x_2 \\ x_2 &= -\frac{v_1}{v_2} x_1 - \frac{v_0}{v_2}. \end{aligned} \quad (1.2.2)$$

As illustrated in Fig. 1.2.1, this is a line partitioning  $\mathbb{R}^2$  into two *decision regions*, with  $y$  taking a value of  $+1$  in one region and  $-1$  in the other. Therefore, if the input vectors  $x = [x_1 \ x_2]^T$  take on values as shown by the  $A$ 's and  $B$ 's, they can be partitioned into the two classes  $A$  and  $B$  by examining the values of  $y$  resulting when the values of  $x$  are presented to the NN.

Given the two regions into which the values of  $x$  should be classified, it is obviously necessary to know how to select the weights and threshold to draw a line between the two. Weight selection and NN training are discussed in the next section.

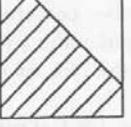
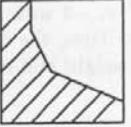
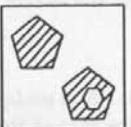
Structure	Types of Decision Regions	Exclusive Or Problem	Most General Region Shapes
1-Layer Neural Net	Hyperplane		
2-Layer Neural Net	Convex Regions		
3-Layer Neural Net	Arbitrary complexity (Limited by number of neurons)		

Figure 1.2.2: Types of decision regions that can be formed using single- and multi-layer NN. From R.P. Lippmann (1987).

In the general case of  $n$  inputs  $x_j$  and  $L$  outputs  $y_\ell$ , the one-layer (Fig. 1.1.4) NN partitions  $\Re^n$  using  $L$  hyperplanes (subspaces of dimension  $n - 1$ ). Clearly, if the values of  $x$  do not fall into regions that are separable using hyperplanes, they cannot be classified using a one-layer NN. Fig. 1.2.2 shows some classification problems that are undecidable for the one-layer NN—a specific example is the EXCLUSIVE-OR.

The two-layer NN with  $n$  inputs,  $L$  hidden-layer neurons, and  $m$  outputs (Fig. 1.1.6) can implement more complex decision boundaries than the one-layer NN. Specifically, the first layer forms  $L$  hyperplanes (each of dimension  $n - 1$ ), and the second layer combines them into  $m$  decision regions by taking various intersections of the regions defined by the hyperplanes, depending on the output-layer weights. Thus, the two-layer NN can form open or closed *convex* decision regions, as shown in Fig. 1.2.2. The X-OR problem can be solved using a two-layer NN. The three-layer NN can form arbitrary decision regions, not necessarily convex, and suffices for the most complex classification problems. This discussion has assumed hard limiters; if smooth activation functions are used the decision boundaries are smooth as well. With smooth activation functions, moreover, the backpropagation training algorithm given in the next section can be used to determine the weights needed to solve any specific classification problem.

The NN structure should be complex enough for the decision problem at hand; too complex a net makes for additional computation time that is not necessary. The number of nodes in the hidden layer should typically be sufficient to provide three or more edges for each decision region generated by the output-layer nodes.

An illustration of NN classification is given in Example 1.3.2.

### 1.2.1.2 Association and Pattern Recognition

In the *association problem* there are prescribed input vectors  $X^p \in \Re^n$  along with output *target* vectors  $Y^p \in \Re^m$ , each of which is to be associated with its corresponding  $X^p$ . In practical situations there might be multiple input vectors prescribed by the user, each with an associated desired output vector. Thus, there might be prescribed  $P$  desired *exemplar* input/output pairs  $(X^1, Y^1), (X^2, Y^2), \dots, (X^P, Y^P)$  for the NN.

Pattern recognition is often a special case of association. In illustration,  $X^p, p = 1, \dots, 26$  could be the letters of the alphabet drawn on a  $7 \times 5$  grid of 1's and 0's (e.g. 0 means light, 1 means dark), and  $Y^1$  could be  $A$ ,  $Y^2$  would be  $B$ , and so on. For presentation to the NN as vectors,  $X^p$  might be encoded as the columns of the  $7 \times 5$  grid stacked on top of one another to produce a 35-vector of ones and zeros, while  $Y^p$  might be the  $p$ -th column of the  $26 \times 26$  identity matrix, so that  $A$  would be encoded as  $[1 \ 0 \ 0 \dots]^T$ , and so on. Then, the NN should associate pattern  $X^p$  with target output  $Y^p$  to classify the letters.

To solve pattern recognition and association problems for a given set Of input/output pairs  $(X^p, Y^p)$ , it is necessary to select the correct weights and thresholds for the NN, as illustrated by the next example.

**Example 1.2.1 (Optimal NN Weights and Biases for Pattern Association) :**

It is desired to design a one-layer NN with one input  $x$  and one output  $y$  that associates input  $X^1 = -3$  with the target output  $Y^1 = 0.4$  and input  $X^2 = 2$  with target output  $Y^2 = 0.8$ . Thus, the desired i/o pairs to be associated are  $(-3, 0.4), (2, 0.8)$ . The NN has only one weight and one bias and the recall equation is

$$y = \sigma(vx + b).$$

Denote the *actual NN outputs* when the input exemplar patterns are presented as

$$y^1 = \sigma(vX^1 + b), \quad y^2 = \sigma(vX^2 + b).$$

When the NN is performing as prescribed, one should have  $y^1 = Y^1, y^2 = Y^2$ . To measure the performance of the NN, define the *least-squares output error* as

$$E = \frac{1}{2} [(Y^1 - y^1)^2 + (Y^2 - y^2)^2].$$

When  $E$  is small, the NN is performing well.

Using the MATLAB NN Toolbox, it is straightforward to plot the least-squares output error  $E$  as a function of the weight  $w$  and bias  $b$ . The result is shown in Fig. 1.2.3 for the sigmoid and the hard limit activation functions.

To design the NN, it is necessary to select  $w$  and  $b$  to minimize the error  $E$ . It is seen that for the hard limit,  $E$  is minimized for a range of weight/bias values. On the other hand, for the sigmoid the error is minimized for  $(w, b)$  in the vicinity of  $(0.3, 0.6)$ . Moreover, the sigmoid allows a smaller minimum value of  $E$  than does the hard limit. Since the error surface plot using the sigmoid is smooth, conventional gradient-based techniques can be used to determine the optimal weight and bias. This topic is discussed in Section 1.3.2 for the one-layer NN and Section 1.3.3 for the multilayer NN.

To make, for instance, Fig. 1.2.3a, the following MATLAB commands were used:

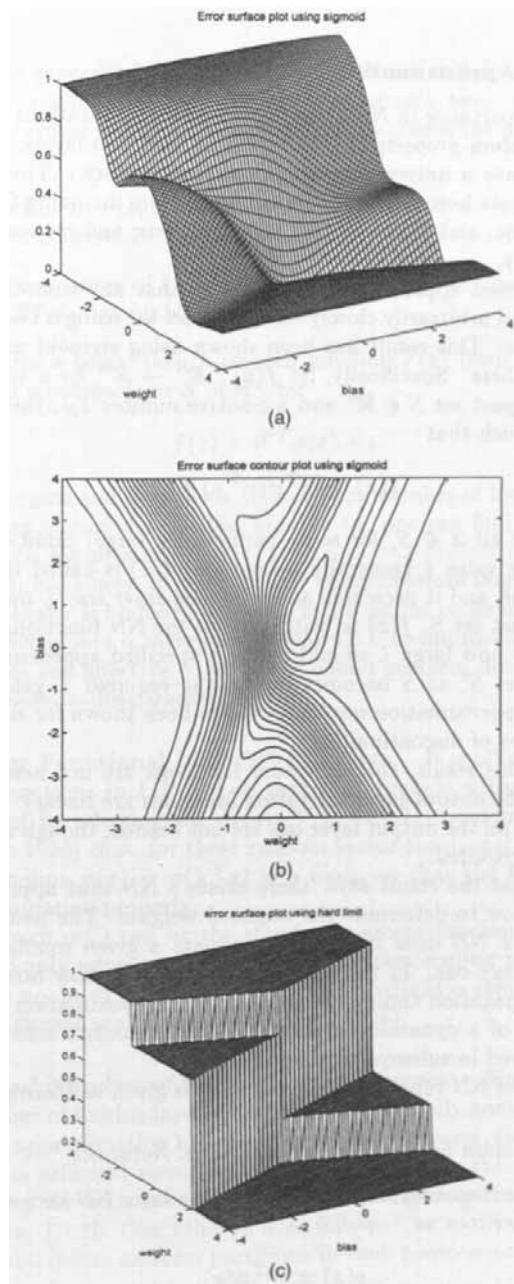


Figure 1.2.3: Output error plots versus weights for a neuron. (a) Error surface using sigmoid activation function. (b) Error contour plot using sigmoid activation function. (c) Error surface using hard limit activation function.

```
% set up input patterns, output targets, and weight/bias ranges:
p=[-3 2];
t=[0.4 0.8];
wv=-4 : 0.1 : 4;
bv=-4 : 0.1 : 4;
% compute output error surface:
es=errsurf(p,t,wv,bv,'logsig');
% plot and label error surface:
mesh(wv,bv,es)
view(60,30)
set(gca,' xlabel',text(0,0,'weight'))
set(gca,' ylabel',text(0,0,'bias'))
title('Error surface plot using sigmoid')
```

Note that the input patterns are stored in a vector  $p$  and the target outputs are stored in a vector  $t$  with corresponding entries.  $\square$

### 1.2.2 Function Approximation

Of fundamental importance in NN closed-loop control applications is the *universal function approximation property* of NN having at least two layers. (One-layer NN do not generally have a universal approximation capability.) The approximation capabilities of NN have been studied by many researchers including Cybenko (1989); Hornik, Stinchcombe, and White (1989); and Sandberg and co-workers (e.g. Park and Sandberg 1991).

The basic universal approximation result says that any smooth function  $f(x)$  can be approximated arbitrarily closely on a compact set using a two-layer NN with appropriate weights. This result has been shown using sigmoid activations, RBF activations, and others. Specifically, let  $f(x) : \Re^n \rightarrow \Re^m$  be a smooth function. Then, given a compact set  $S \in \Re^n$  and a positive number  $\varepsilon_N$ , there exists a two-layer NN (1.1.21) such that

$$f(x) = W^T \sigma(V^T x) + \varepsilon \quad (1.2.3)$$

with  $\|\varepsilon\| < \varepsilon_N$  for all  $x \in S$ , for some (sufficiently large) number  $L$  of hidden-layer neurons. The value  $\varepsilon$  (generally a function of  $x$ ) is called the *NN function approximation error*, and it decreases as the hidden-layer size  $L$  increases. We say that, on the compact set  $S$ ,  $f(x)$  is ‘within  $\varepsilon_N$  of the NN functional range’. Some recent results show how large  $L$  should be for a specified approximation accuracy  $\varepsilon_N$  and compact set  $S$ ; as  $S$  becomes larger, the required  $L$  generally increases correspondingly. Approximation results have also been shown for smooth functions with a finite number of discontinuities.

Note that, in this result, the activation functions are not needed on the NN output layer (i.e. the output-layer activation functions are linear). It also happens that the thresholds on the output layer  $w_{i0}$  are not needed, though the hidden-layer thresholds  $v_{\ell 0}$  are required.

Note further that the result says ‘there exists a NN that approximates  $f(x)$ ’, it does not show how to determine the required weights. The issue of finding the weights such that a NN does indeed approximate a given function  $f(x)$  closely enough is not an easy one. In the next section we shall show how to accomplish

this using backpropagation tuning. If the function approximation is to be carried out in the context of a dynamic closed-loop feedback control scheme, the issue is thornier and is solved in subsequent chapters.

An illustration of NN function approximation is given in Example 1.3.3.

### 1.2.2.1 Approximation by Functional-Link Neural Networks

In Section 1.1.3 was discussed a special class of one-layer NN known as Functional-Link NN (FLNN) written as

$$y(x) = W^T \phi(x), \quad (1.2.4)$$

with  $W$  the NN output weights (including thresholds) and  $\phi(\cdot)$  a general function from  $\Re^n$  to  $\Re^L$ . In subsequent chapters, these NN have a great advantage in that they are easier to train than general two-layer NN since they are *linear in the tunable parameters* (LIP). Unfortunately, for LIP NN, the functional approximation property does not generally hold. However, a FLNN can still approximate functions as long as the activation functions  $\phi(\cdot)$  are selected as a *basis*, which must satisfy the following two requirements on a compact simply-connected set  $\mathcal{S}$  of  $\Re^n$  (Sadegh 1993):

1. A constant function on  $\mathcal{S}$  can be expressed as (1.2.4) for a finite number  $L$  of hidden-layer neurons.
2. The functional range of (1.2.4) is dense in the space of continuous functions from  $\mathcal{S}$  to  $\Re^m$  for countable  $L$ .

If  $\phi(\cdot)$  provides a basis, then a smooth function  $f(x)$  from  $\Re^n$  to  $\Re^m$  can be approximated on a compact set  $\mathcal{S}$  of  $\Re^n$ , by

$$f(x) = W^T \phi(x) + \epsilon \quad (1.2.5)$$

for some ideal weights and thresholds  $W$  and some number of hidden-layer neurons  $L$ . In fact, for any choice of a positive number  $\epsilon_N$ , one can find a feedforward NN such that  $\|\epsilon\| < \epsilon_N$  for all  $x$  in  $\mathcal{S}$ .

Barron (1993) has shown that for all LIP approximators there is a fundamental lower bound, so that  $\epsilon$  is bounded below by terms on the order of  $1/L^{2/n}$ . Thus, as the number of NN inputs  $n$  increases, increasing  $L$  to improve the approximation accuracy becomes less effective. This lower bound problem does not occur in the multilayer nonlinear-in-the-parameters nets.

**Random Vector Functional Link (RVFL) Nets.** It is often difficult to select the activation functions in LIP NN so that they provide a basis. This problem may be addressed by selecting the matrix  $V$  in (1.1.21) randomly. It is shown in (Igelnik and Pao 1995) that, for these random vector functional link (RVFL) nets, the resulting function  $\phi(x) = \sigma(V^T x)$  is a basis, so that the RVFL NN has the universal approximation property.

In this approach,  $\sigma(\cdot)$  can be the standard sigmoid functions. This approach amounts to randomly selecting the activation function scaling parameters  $v_{\ell j}$  and shift parameters  $v_{\ell 0}$  in  $\sigma(\sum_j v_{\ell j} x_j + v_{\ell 0})$ . This produces a family of  $L$  activation functions with different scaling and shifts (Kim 1996).

**On the Required Number of Hidden-Layer Neurons.** The problem of determining the number of hidden-layer neurons for general fully-connected NN (1.1.21) for good enough approximation has not been solved. However, for NN such as RBF or CMAC there is sufficient structure to allow a solution to this problem. The key hinges on selecting the activation functions close enough together in situations like Fig. 1.1.9 and Fig. 1.1.10. One solution is as follows.

Let  $x \in \Re^n$  and define uniform partitions in each component  $x_j$ . Let  $\delta_j$  be the partition interval for  $x_j$  and  $\delta \equiv \sqrt{\sum_{j=1}^n \delta_j^2}$ . In illustration, in Fig. 1.1.10 where  $n = 2$ , one has  $\delta_1 = \delta_2 = 0.5$ . The next result shows the maximum partition size  $\delta$  allowed for approximation with a desired accuracy  $\epsilon$  (Commuri 1996).

**Theorem 1.2.1 (Partition Interval for CMAC Approximation) :**

Let a function  $f(x) : \Re^n \rightarrow \Re^m$  be continuous with Lipschitz constant  $\lambda$  so that

$$\|f(x) - f(z)\| \leq \lambda \|x - z\|$$

for all  $x, z$  in some compact set  $\mathcal{S}$  of  $\Re^n$ . Construct a CMAC with triangular receptive field functions  $\phi(\cdot)$  in the recall equation (1.2.4). Then there exist weights  $W$  such that

$$\|f(x) - y(x)\| \leq \epsilon$$

for all  $x \in \mathcal{S}$  if the CMAC is designed so that

$$\delta \leq \frac{\epsilon}{m\lambda}. \quad (1.2.6)$$

□

In fact, CMAC designed with this partition interval can approximate on  $\mathcal{S}$  any continuous function smooth enough to satisfy the Lipschitz condition for the given  $\lambda$ . Now, given limits on the dimensions of  $\mathcal{S}$  one can translate this upper bound on  $\delta$  to a lower bound on the number  $L$  of hidden-layer neurons. Note that as the functions  $f(x)$  become less smooth, so that  $\lambda$  increases, the grid nodes become more finely spaced so that the required number  $L$  of hidden-layer neurons increases.

In Sanner and Slotine (1991) is given a similar result for designing RBF which selects the fineness of the grid partition based on a frequency-domain smoothness measure for  $f(x)$  instead of a Lipschitz constant smoothness measure.

### 1.3 NEURAL NETWORK WEIGHT SELECTION AND TRAINING

We have studied the topology of NN, and shown that they possess some important properties including classification and function approximation capabilities. For a NN to function as desired, however, it is necessary to determine suitable weights and thresholds. For years this was a problem, especially for multilayer nets, where it was not known how to apportion resulting errors to different layers and force the appropriate weights to change to reduce the errors—this was known as the error ‘credit assignment problem’. Today, these problems have for the most part been solved and there are very good algorithms for NN weight selection and/or tuning. References for this section include Haykin (1994), Kung (1993), Peretto (1992), and Hush and Horne (1993).

**Direct Computation Versus Training.** There are two basic approaches to determining NN weights: direct analytic computation and NN training by recursive update techniques. In the Hopfield net, for instance, the weights can be directly computed in terms of the desired outputs of the NN. In many other applications of static NN, the weights are tuned by a recursive NN training procedure. In all of this chapter we are talking about NN in *open-loop applications*. That is, not until later chapters do we get into the issues of tuning the NN weights while the NN is simultaneously performing in the capacity of a feedback controller to stabilize a dynamical plant.

**Classification of Learning Schemes.** Updating the weights by training the NN is known as the *learning feature* of NN. Learning algorithms may be carried out in continuous-time (via differential equations for the weights), or in discrete form (via difference equations for the weights). There are many learning algorithms, and they fall into three categories. In *supervised learning*, all the information needed for training is available *a priori*, for instance, the inputs  $x$  and the desired outputs  $y$  they should produce. This global information does not change, and is used to compute errors that can be used for updating the weights. It is said that there is a ‘teacher’ that knows the desired outcomes and tunes the weights accordingly. On the other hand, in *unsupervised learning* (also called self-organizing behavior) the desired NN output is not known, so there is no teacher with global information. Instead, local data is examined and organized according to emergent collective properties. Finally, in *reinforcement learning*, the weights associated with a particular neuron are not changed proportionally to the output error of that neuron, but instead are changed in proportion to some global reinforcement signal.

**Learning and Operational Phases.** There is a distinction between the *learning phase*, when the NN weights are selected (often through training), and the *operational phase*, when the weights are generally held constant and inputs are presented to the NN as it performs its design function. During training the weights are often selected using prescribed exemplar inputs and outputs for the NN. In the operational phase, it is often the case that the inputs do not belong to the exemplar training set. However, in classification, for instance, the NN is able to provide the output corresponding to the exemplar to which any given input is *closest* in some specified norm (e.g. a noisy ‘A’ should be classified as an ‘A’). This ability to process inputs not necessarily in the exemplar set and provide meaningful outputs is known as the *generalization* property of NN, and is closely connected to the property of associative memories that close inputs should provide close outputs.

**Off-Line versus On-Line Learning.** Finally, learning may be *off-line*, where the preliminary learning phase occurs prior to applying the NN in its operational capacity (during which the weights are held constant), or *on-line*, where the NN functions in its intended operational capacity while simultaneously learning the weights. Examples of off-line learning include open-loop applications such as classification and pattern recognition. On-line learning is a very difficult problem, and is exemplified by closed-loop feedback control applications. There, the NN must keep a dynamical plant stable while simultaneously learning and ensuring that its own internal state (the weights) remains bounded. Various techniques from *adaptive control theory* are needed to successfully confront this problem.

### 1.3.1 Direct Computation of the Weights

In the Hopfield net, the weights can be initialized by direct computation of outer products between the desired outputs. The continuous-time Hopfield net has dynamics

$$\tau_i \dot{x}_i = -x_i + \sum_{j=1}^n w_{ij} \sigma_j(x_j) + u_i \quad (1.3.1)$$

or

$$\dot{x} = -\Gamma x + \Gamma W^T \sigma(x) + \Gamma u, \quad (1.3.2)$$

with  $x \in \Re^n$ .

Suppose it is desired to design a Hopfield net that can discriminate between  $P$  prescribed bipolar pattern vectors  $X^1, X^2, \dots, X^P$ , e.g., each having  $n$  entries of either  $+1$  or  $-1$ . Thus, the Hopfield net should perform as an associative memory that discriminates among bipolar vectors, matching each input vector  $x(0)$  presented as an initial condition with one of the  $P$  *exemplar patterns*  $X^p$ . It was shown by Hopfield that weights solving this problem may be selected using the Hebbian philosophy of learning as the *outer products* of the exemplar vectors

$$W = \frac{1}{n} \sum_{p=1}^P X^p (X^p)^T - \frac{1}{n} PI, \quad (1.3.3)$$

where  $I$  is the identity matrix. The purpose of the term  $PI$  is to zero out the diagonal. Note that this weight matrix  $W$  is symmetric. This formula effectively encodes the exemplar patterns in the weights of the NN. Though there is no weight tuning, technically this formula is an example of supervised learning, as the desired outputs are used to compute the weights.

It can be shown that, with these weights, there are  $P$  equilibrium points in  $\Re^n$ , one at each of the exemplar vectors  $X^p$  (see Problems Section and, for instance, Hush and Horne 1993, Haykin 1994). Once the weights have been computed (the training phase), the net can be used in its operational phase, where an unknown vector  $x(0)$  is presented as an initial condition, and the net state is computed as a function of time using (1.3.2). The net will converge to the equilibrium point  $X^p$  to which the input vector  $x(0)$  is closest. (If the symmetric hard limit activation functions are used, the ‘closest’ vector is defined in terms of the Hamming distance.) It is intriguing to note that the information is *stored* in the net using (1.3.3) (during the ‘training’ phase), and *recalled* from the net using (1.3.2) (during the ‘operational’ phase). Thus, the NN functions as a biologically inspired memory device.

It can be shown that, with  $n$  the size of the Hopfield net, one can obtain perfect recall if the number of stored exemplar patterns satisfies  $P \leq n/(4 \ln n)$ . For example, if there are 256 neurons in the net, then the maximum number of exemplar patterns allowed is  $P = 12$ . However, if a small fraction of the bits in the recalled pattern are allowed to be in error, then the capacity increases to  $P \leq 0.138n$ . If  $P = 0.138n$  then approximately 1.6% of the bits in the recalled pattern are in error. Other weight selection techniques allow improved storage capacity in the Hopfield net, in fact, with proper computation of  $W$  the net capacity can approach  $P = n$ .

#### Example 1.3.1 (Hopfield Net Weight Selection) :

In Example 1.1.3 was considered the Hopfield net

$$\dot{x} = -\frac{1}{2}x + \frac{1}{2}W^T \sigma(x) + \frac{1}{2}u$$

with  $x \in \Re^2$  and symmetric sigmoid activations having decay constants  $g_1 = g_2 = 100$ .

Suppose the prescribed exemplar patterns are

$$X^1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, X^2 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}.$$

Then, according to the ‘training’ equation (1.3.3), one has the weight matrix

$$W = W^T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Using these weights, state trajectory phase-plane plots for various initial condition vectors  $x(0)$  were shown in Fig. 1.1.16. Indeed, in all cases, the state trajectories converged either to the point  $(-1, -1)$  or to  $(1, 1)$ .  $\square$

### 1.3.2 Training the One-Layer Neural Network— Gradient Descent

In this subsection will be considered the problem of tuning the weights in the one-layer NN shown in Fig. 1.1.4 and described by the recall equation

$$y_\ell = \sigma \left( \sum_{j=1}^n v_{\ell j} x_j + v_{\ell 0} \right) ; \quad \ell = 1, 2, \dots, L, \quad (1.3.4)$$

or in matrix form

$$y = \sigma(V^T x), \quad (1.3.5)$$

with  $x = [1 \ x_1 \ x_2 \dots x_n]^T \in \Re^{n+1}$ ,  $y \in \Re^L$ , and  $V$  the matrix of weights and thresholds. A tuning algorithm for this ‘single-layer perceptron’ was first derived by Rosenblatt in 1959; he used the symmetric hard limiter activation function. Widrow and Hoff studied the case of linear  $\sigma(\cdot)$  in 1960 (Haykin 1994).

There are many sorts of training algorithms for NN; the basic type we shall discuss is *error correction training*. We shall introduce a matrix-calculus-based approach that is very convenient for formulating NN training algorithms. Since NN training is usually performed using digital computers, a convenient form of weight update equation is expressed in terms of *discrete iterations*, where the weights are updated by discrete iteration steps. Such *digital update algorithms* are extremely convenient for computer implementation and are considered in this subsection. Continuous-time weight updating is discussed in Section 1.3.6.

#### 1.3.2.1 Gradient Descent Tuning

In this discussion the iteration index is denoted as  $k$ . One should not think of  $k$  as a time index as the iteration index is not necessarily the same as the time index. Let  $v_{\ell j}(k)$  be the NN weights at iteration  $k$  so that

$$y_\ell(k) = \sigma \left( \sum_{j=1}^n v_{\ell j}(k) X_j + v_{\ell 0}(k) \right) ; \quad \ell = 1, 2, \dots, L. \quad (1.3.6)$$

In this equation,  $X_j$  are the components of a prescribed *constant* input vector  $X$  that stays the same during training of the NN. A general class of weight update algorithms is given by the recursive update equation

$$v_{\ell j}(k+1) = v_{\ell j}(k) - \eta \frac{\partial E(k)}{\partial v_{\ell j}(k)} \quad (1.3.7)$$

where  $E(k)$  is a *cost function* that is selected depending on the application. In this algorithm, the weights  $v_{\ell j}$  are updated at each iteration number  $k$  in such a manner that the prescribed cost function decreases. This is accomplished by going ‘downhill’ against the gradient  $\frac{\partial E(k)}{\partial v_{\ell j}(k)}$ . The positive step size parameter  $\eta$  is taken as less than 1 and is called the *learning rate*.

To see that the gradient descent algorithm decreases the cost function, note that  $\Delta v_{\ell j}(k) \equiv v_{\ell j}(k+1) - v_{\ell j}(k)$  and, to first order

$$\begin{aligned}\Delta E(k) &\equiv E(k+1) - E(k) \\ &\approx \sum_{\ell,j} \frac{\partial E(k)}{\partial v_{\ell j}(k)} \Delta v_{\ell j}(k) \\ &= -\eta \sum_{\ell,j} \left( \frac{\partial E(k)}{\partial v_{\ell j}(k)} \right)^2.\end{aligned}\quad (1.3.8)$$

Techniques such as ‘conjugate gradient’ take into account second-order and higher terms in this Taylor series expansion.

A specific gradient descent algorithm is derived by taking the cost function as the least-squares NN output-error. Thus, let a prescribed pattern vector  $X$  be input to the NN and the desired target output associated with  $X$  be  $Y$  (c.f. Example 1.2.1). Then, at iteration number  $k$  the  $\ell$ -th component of the output error is

$$e_\ell(k) = Y_\ell - y_\ell(k), \quad (1.3.9)$$

where  $Y_\ell$  is the desired output and  $y_\ell(k)$  is the actual output with input  $X$ . Define the least-squares output-error cost as

$$E(k) = \frac{1}{2} \sum_{\ell=1}^L e_\ell^2(k) = \frac{1}{2} \sum_{\ell=1}^L (Y_\ell - y_\ell(k))^2. \quad (1.3.10)$$

Note that the components  $X_j$  of the input  $X$  and the desired NN output components  $Y_\ell$  are not functions of the iteration number  $k$  (see the subsequent discussion on series versus batch updating).

To derive the gradient descent algorithm with least-squares output-error cost, the gradients with respect to the weights and thresholds are computed using the product rule and the chain rule as

$$\frac{\partial E(k)}{\partial v_{\ell j}(k)} = -e_\ell(k) \sigma' \left( \sum_{j=1}^n v_{\ell j}(k) X_j + v_{\ell 0}(k) \right) X_j \quad (1.3.11)$$

$$\frac{\partial E(k)}{\partial v_{\ell 0}(k)} = -e_\ell(k) \sigma' \left( \sum_{j=1}^n v_{\ell j}(k) X_j + v_{\ell 0}(k) \right), \quad (1.3.12)$$

where equations (1.3.10) and (1.3.6) were used. The notation  $\sigma'(\cdot)$  denotes the derivative of the activation function evaluated at the argument. Therefore, the gradient descent algorithm for the least-squares output-error case yields the weight updates

$$v_{\ell j}(k+1) = v_{\ell j}(k) + \eta e_\ell(k) \sigma' \left( \sum_{j=1}^n v_{\ell j}(k) X_j + v_{\ell 0}(k) \right) X_j \quad (1.3.13)$$

and the threshold updates

$$v_{\ell 0}(k+1) = v_{\ell 0}(k) + \eta e_{\ell}(k) \sigma' \left( \sum_{j=1}^n v_{\ell j}(k) X_j + v_{\ell 0}(k) \right). \quad (1.3.14)$$

Historically, the derivative of the activation functions was not used to update the weights prior to the 1970s (see next section). Widrow and Hoff took linear activation functions so that the tuning algorithm becomes the linear mean-square (LMS) algorithm

$$v_{\ell j}(k+1) = v_{\ell j}(k) + \eta e_{\ell}(k) X_j \quad (1.3.15)$$

$$v_{\ell 0}(k+1) = v_{\ell 0}(k) + \eta e_{\ell}(k). \quad (1.3.16)$$

This is the algorithm generally used for training the one-layer perceptron even if nonlinear activation functions are used. It is called the ‘perceptron training algorithm’ or the ‘delta rule’. Rosenblatt showed that, using the symmetric hard limit activation functions, if the classes of input vectors are *separable* using linear decision boundaries, then this algorithm converges to the correct weights (Haykin 1994).

**Matrix Calculus Formulation.** A matrix calculus approach can be used to derive the delta rule by a streamlined method that is well suited for simplifying notation. Thus, given the input-output pair  $(X, Y)$  that the NN should associate, define the NN output error vector as

$$e(k) = Y - y(k) = Y - \sigma(V^T(k)X) \in \Re^L \quad (1.3.17)$$

and the least-squares output-error cost as

$$E(k) = \frac{1}{2} e^T(k) e(k) = \frac{1}{2} \text{tr}\{e(k) e^T(k)\}. \quad (1.3.18)$$

The *trace* of a square matrix  $\text{tr}\{\cdot\}$  is defined as the sum of the diagonal elements. One uses the expression involving the trace  $\text{tr}\{ee^T\}$  due to the fact that derivatives of the trace with respect to matrices are very convenient to evaluate. On the other hand, evaluating gradients of  $e^T e$  with respect to weight matrices involves the use of third-order tensors, which must be managed using the Kronecker product (Lewis, Abdallah, and Dawson 1993) or other machinations. A few matrix calculus identities are very useful; they are given in Table 1.3.1.

In terms of matrices the gradient descent algorithm is

$$V(k+1) = V(k) - \eta \frac{\partial E(k)}{\partial V(k)}. \quad (1.3.19)$$

Write

$$E(k) = \frac{1}{2} \text{tr} \left\{ (Y - \sigma(V^T(k)X)) (Y - \sigma(V^T(k)X))^T \right\}, \quad (1.3.20)$$

where  $e(k)$  is the NN output error associated with input vector  $X$  using the weights  $V(k)$  determined at iteration  $k$ . Assuming linear activation functions  $\sigma(\cdot)$  one has

$$E(k) = \frac{1}{2} \text{tr} \left\{ (Y - V^T(k)X) (Y - V^T(k)X)^T \right\}. \quad (1.3.21)$$

Table 1.3.1: Basic Matrix Calculus and Trace Identities

Let  $r, s$  be scalars;  $A, B, C$  be matrices; and  $x, y, z$  be vectors, all dimensioned so that the following formulae are compatible. Then:

$$\text{tr}\{AB\} = \text{tr}\{BA\}, \quad \text{when the matrices have compatible dimensions} \quad (1.3.25)$$

$$\frac{\partial \text{tr}\{BAC\}}{\partial A} = B^T C^T \quad (1.3.26)$$

$$\frac{\partial \text{tr}\{ABA^T\}}{\partial A} = 2AB \quad (1.3.27)$$

$$\frac{\partial s}{\partial A^T} = \left[ \frac{\partial s}{\partial A} \right]^T \quad (1.3.28)$$

$$\frac{\partial AB}{\partial s} = \frac{\partial A}{\partial s} B + A \frac{\partial B}{\partial s}, \quad \text{product rule} \quad (1.3.29)$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial x} \quad \text{chain rule} \quad (1.3.30)$$

$$\frac{\partial s}{\partial t} = \text{tr} \left\{ \frac{\partial s}{\partial A} \cdot \frac{\partial A^T}{\partial t} \right\}, \quad \text{chain rule} \quad (1.3.31)$$

Now, using the identities in Table 1.3.1 (especially (1.3.26)) one easily determines (see Problems Section) that

$$\frac{\partial E(k)}{\partial V(k)} = -X e^T(k) \quad (1.3.22)$$

so that the gradient descent tuning algorithm is written as

$$V(k+1) = V(k) + \eta X e^T(k), \quad (1.3.23)$$

which updates both the weights and the thresholds. Recall that the first column of  $V^T$  consists of the thresholds and the first entry of  $X$  is 1. Therefore, the threshold vector  $b_v$  in (1.1.7) is updated according to

$$b_v(k+1) = b_v(k) + \eta e(k). \quad (1.3.24)$$

It is interesting to note that the weights are updated according to the *outer product* of the prescribed pattern vector  $X$  and the NN output error  $e$ .

### 1.3.2.2 Series versus Batch Updating

We have just discussed NN weight training when one input-vector/desired-output-vector pair  $(X, Y)$  is given for a NN. In practical situations there might be multiple input vectors prescribed by the user, each with an associated desired output vector. Thus, suppose there are prescribed  $P$  desired input/output pairs  $(X^1, Y^1), (X^2, Y^2), \dots, (X^P, Y^P)$  for the NN.

In such situations the NN must be trained to associate *each* input vector with its prescribed output vector. There are many strategies for training the net in this scenario; at the two extremes are *series updating* and *batch upadating*, also called parallel or block updating. For this discussion we shall use matrix updates, defining for  $p = 1, 2, \dots, P$  the quantities:

$$y^p(k) = \sigma(V^T(k)X^p) \quad (1.3.32)$$

$$e^p(k) = Y^p - y^p(k) = Y^p - \sigma(V^T(k)X^p) \quad (1.3.33)$$

$$E^p(k) = \frac{1}{2}(e^p(k))^T e^p(k) = \frac{1}{2} \text{tr}\{e^p(k)(e^p(k))^T\}. \quad (1.3.34)$$

In series updating the vectors  $(X^p, Y^p)$  are sequentially presented to the NN. At each presentation, one step of the training algorithm is performed so that

$$V(k+1) = V(k) + \eta X^p (e^p(k))^T, \quad p = 1, 2, \dots, P, \quad (1.3.35)$$

which updates both the weights and thresholds (see (1.1.10)). An *epoch* is defined as one complete run through all the  $P$  associated pairs. When one epoch has been completed, the pair  $(X^1, Y^1)$  is presented again and another run through all the  $P$  pairs is performed. Hopefully, after many epochs, the output error will be small enough.

In batch updating, all  $P$  pairs are presented to the NN (one at a time) and a cumulative error is computed after all have been presented. At the end of this procedure, the NN weights are updated once. The result (see Problems section) is

$$V(k+1) = V(k) + \eta \sum_{p=1}^P X^p (e^p(k))^T. \quad (1.3.36)$$

In batch updating, the iteration index  $k$  corresponds to the number of times the set of  $P$  patterns is presented and the cumulative error computed. That is,  $k$  corresponds to the epoch number.

There is a very convenient way to perform batch NN weight updating using matrix manipulations. Thus, define the matrices

$$X \equiv [X^1 \ X^2 \ \dots \ X^P], \quad Y \equiv [Y^1 \ Y^2 \ \dots \ Y^P], \quad (1.3.37)$$

which contain all  $P$  of the prescribed input/ouput vectors, and the batch error matrix

$$e(k) \equiv [e^1(k) \ e^2(k) \ \dots \ e^P(k)]. \quad (1.3.38)$$

It is now very easy to see that the NN recall can be computed using the equation

$$y(k) = \sigma(V^T(k)X) \quad (1.3.39)$$

where the batch output matrix is  $y(k) \equiv [y^1(k) \ y^2(k) \ \dots \ y^P(k)]$ . Therefore, the batch weight update can be written as

$$V(k+1) = V(k) + \eta X e^T(k). \quad (1.3.40)$$

This method involves the concept of presenting all  $P$  of the prescribed inputs  $X_p$  to the NN *simultaneously*.

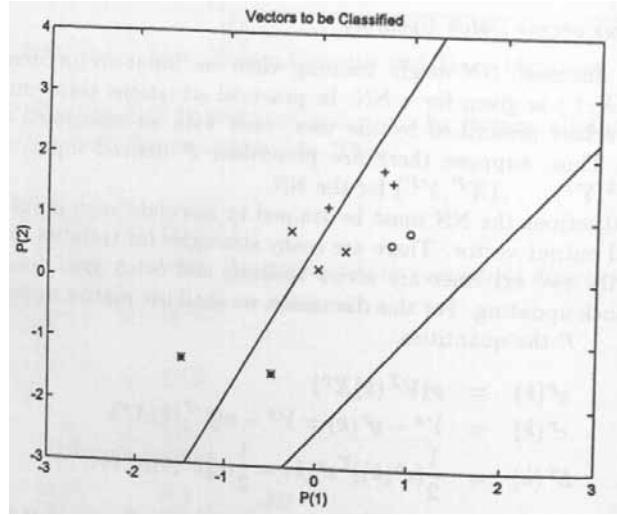


Figure 1.3.1: Pattern vectors to be classified into 4 groups:  $+$ ,  $\circ$ ,  $\times$ ,  $*$ . Also shown are the initial decision boundaries.

It has been mentioned that the update iteration index  $k$  is not necessarily the same as the time index. In fact, one now realizes that the relation between  $k$  and the time is dependent on how one chooses to process multiple prescribed input-output pairs.

**Example 1.3.2 (NN Training— a Simple Classification Example) :**

It is desired to design a one-layer NN with two inputs and two outputs that classifies the following 10 points in  $\Re^2$  into the four groups shown:

Group 1:

$$\overline{(0.1, 1.2)}, (0.7, 1.8), (0.8, 1.6)$$

Group 2:

$$\overline{(0.8, 0.6)}, (1.0, 0.8)$$

Group 3

$$\overline{(0.3, 0.5)}, (0.0, 0.2), (-0.3, 0.8)$$

Group 4

$$\overline{(-0.5, -1.5)}, (-1.5, -1.3)$$

These points are shown in Fig. 1.3.1, where the groups are denoted respectively by  $+$ ,  $\circ$ ,  $\times$ ,  $*$ . The hard limit activation function will be used as it is suitable for classification problems.

To cast this in terms tractable for NN design, encode the four groups respectively by 10, 00, 11, 01. Then, define the input pattern matrix as

$$p = [X^1 \ X^2 \ \dots \ X^{10}] \\ = \begin{bmatrix} 0.1 & 0.7 & 0.8 & 0.8 & 1.0 & 0.3 & 0.0 & -0.3 & -0.5 & -1.5 \\ 1.2 & 1.8 & 1.6 & 0.6 & 0.8 & 0.5 & 0.2 & 0.8 & -1.5 & -1.3 \end{bmatrix}$$

and the target vector as

$$t = [Y^1 \ Y^2 \ \dots \ Y^{10}]$$

$$= \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Then, the three points associated with target vector  $[1 \ 0]^T$  will be assigned to the same group, and so on.

The design will be carried out using the MATLAB NN Toolbox. The one-layer NN weights  $v$  and biases  $b$  are initialized using

$[v,b] = \text{initp}(p,t)$ ,

which assigns random values between -1 and 1 to each element. The result is

$$v = \begin{bmatrix} -0.5621 & 0.3577 \\ -0.9059 & 0.3586 \end{bmatrix}, \quad b = \begin{bmatrix} 0.8694 \\ -0.2330 \end{bmatrix}.$$

Each output  $y_\ell$  of the NN yields one decision line in the  $\Re^2$  plane, as shown in Example 1.1.1. The two lines given by the random initial weights are drawn using the commands

```
plotpv(p,t)      % draws the points corresponding to the 10 input vectors
plotpc(v,b)      % superimposes the decision lines corresponding to weight v
and bias b
```

The initial decision lines are shown in Fig. 1.3.1.

Now, the NN was trained using the batch updating algorithm (1.3.40). The MATLAB commands are

```
tp = [1 3]
[v,b] = trainp(v,b,p,t,tp)      % batch weight/bias update algorithm
```

where  $tp$  is a training parameter vector whose first entry indicates how often the output error should be displayed, and whose second entry indicates the number of epochs training should continue. Recall that an *epoch* is one complete presentation of all 10 patterns to the NN (in this case all 10 are presented simultaneously using the batch update techniques discussed in connection with (1.3.40)).

After 3 epochs, the weights and bias are

$$v = \begin{bmatrix} -0.5621 & 6.4577 \\ -1.2059 & -1.6414 \end{bmatrix}, \quad b = \begin{bmatrix} 0.8694 \\ 1.7670 \end{bmatrix}.$$

The corresponding decision lines are shown in Fig. 1.3.2a.

Now the vector  $tp$  was reset to  $[1 \ 20]$  and the NN training was continued. After 3 further epochs (e.g. 6 epochs in all) the error was small enough and training ceased. The final weights and biases are

$$v = \begin{bmatrix} -3.8621 & 4.5577 \\ -1.2059 & -1.6414 \end{bmatrix}, \quad b = \begin{bmatrix} -0.1306 \\ 1.7670 \end{bmatrix}$$

and the final decision boundaries are shown in Fig. 1.3.2b. The plot of least-squares output error (1.3.34) vs. epoch is shown in Fig. 1.3.3.  $\square$

### 1.3.3 Training the Multilayer Neural Network—Backpropagation Tuning

A one-layer NN can neither approximate general functions nor perform the EXCLUSIVE-OR operation, which is basic to digital logic implementations. When it was demonstrated that the two-layer NN has both these capabilities, and that a three-layer NN is sufficient for the most general pattern classification applications, there was

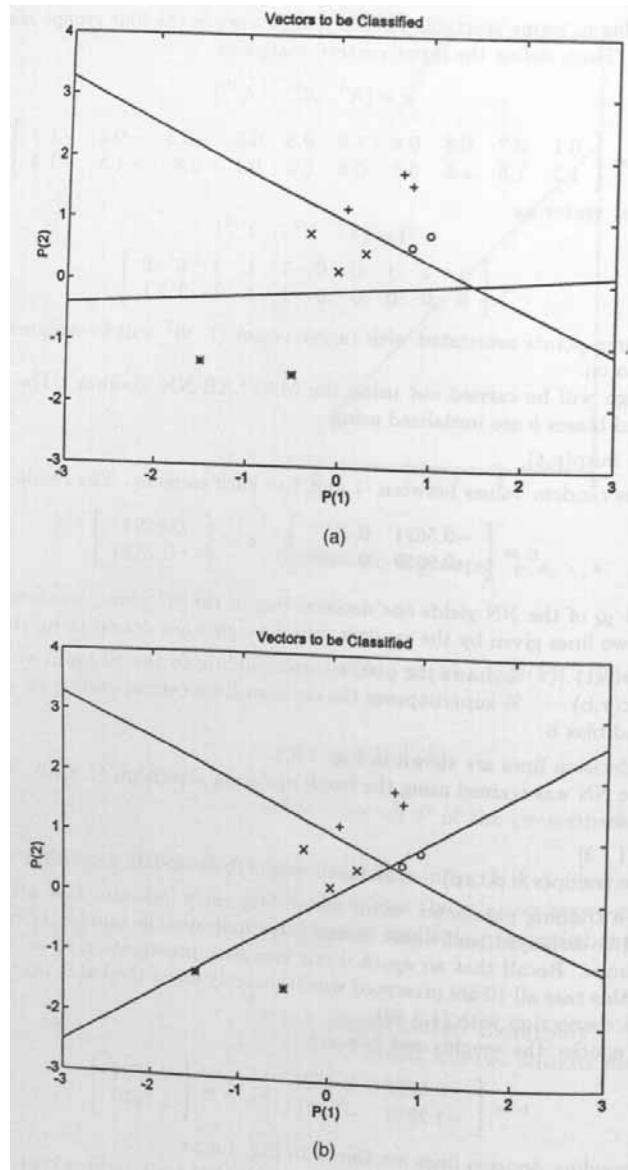


Figure 1.3.2: NN decision boundaries. (a) After three epochs of training. (b) After six epochs of training.

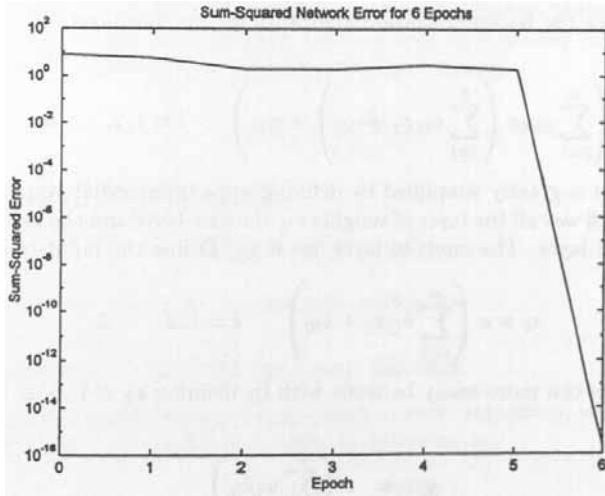


Figure 1.3.3: Least-squares NN output error versus epoch

a sudden intense interest in multilayer NN. Unfortunately, for years it was not understood how to train a multilayer net.

The problem was to assign each weight part of the credit for NN output errors in order to determine how to tune that weight. This so-called ‘credit assignment’ problem was finally solved by several researchers (Werbos (1974, 1989); Rumelhart, Hinton, and Williams (1986)), who derived the *Backpropagation Training Algorithm*. The solution is surprisingly straightforward in retrospect, hinging on a simple application of calculus using the chain rule.

In Section 1.3.2 it was shown how to train a one-layer NN. There, the delta rule was derived ignoring the nonlinearity of the activation function. In this section we show how to derive the full NN weight update rule for a multilayer NN including all activation function nonlinearities. For this application, *the activation functions selected must be differentiable*. Though the backpropagation algorithm enjoys great success, one must remember that it is still a gradient-based technique, so that the usual caveats associated with step sizes, local minima and so on must be kept in mind when using it (see Section 1.3.4).

#### 1.3.3.1 Background

We shall derive the backpropagation algorithm for the two-layer NN in Fig. 1.1.6 described by

$$y_i = \sigma \left( \sum_{\ell=1}^L w_{i\ell} \sigma \left( \sum_{j=1}^n v_{\ell j} x_j + v_{\ell 0} \right) + w_{i0} \right) ; \quad i = 1, 2, \dots, m. \quad (1.3.41)$$

The derivation is greatly simplified by defining some intermediate quantities.

In Fig. 1.1.6 we call the layer of weights  $v_{\ell j}$  the first layer and the layer of weights  $w_{i\ell}$  the second layer. The input to layer one is  $x_j$ . Define the input to layer two as

$$z_\ell = \sigma \left( \sum_{j=1}^n v_{\ell j} x_j + v_{\ell 0} \right) ; \quad \ell = 1, 2, \dots, L. \quad (1.3.42)$$

The thresholds can more easily be dealt with by defining  $x_0 \equiv 1, z_0 \equiv 1$ . Then one can say

$$y_i = \sigma \left( \sum_{\ell=0}^L w_{i\ell} z_\ell \right) \quad (1.3.43)$$

$$z_\ell = \sigma \left( \sum_{j=0}^n v_{\ell j} x_j \right). \quad (1.3.44)$$

It is convenient at this point to begin thinking in terms of moving *backward* through the NN, hence the ordering of this and subsequent lists of equations. Define the outputs of layers two and one respectively as

$$u_i^2 = \sum_{\ell=0}^L w_{i\ell} z_\ell \quad (1.3.45)$$

$$u_\ell^1 = \sum_{j=0}^n v_{\ell j} x_j. \quad (1.3.46)$$

Then we can write

$$y_i = \sigma(u_i^2) \quad (1.3.47)$$

$$z_\ell = \sigma(u_\ell^1). \quad (1.3.48)$$

In deriving the backpropagation algorithm we shall have occasion to differentiate the activation functions. Note therefore that

$$\frac{\partial y_i}{\partial w_{i\ell}} = \sigma'(u_i^2) z_\ell \quad (1.3.49)$$

$$\frac{\partial y_i}{\partial z_\ell} = \sigma'(u_i^2) w_{i\ell} \quad (1.3.50)$$

$$\frac{\partial z_\ell}{\partial v_{\ell j}} = \sigma'(u_\ell^1) x_j \quad (1.3.51)$$

$$\frac{\partial z_\ell}{\partial x_j} = \sigma'(u_\ell^1) v_{\ell j}, \quad (1.3.52)$$

where  $\sigma'(\cdot)$  is the derivative of the activation function.

Part of the power of the backpropagation algorithm soon to be derived is the fact that the evaluation of the activation function derivative is very easy for common  $\sigma(\cdot)$ . Specifically, selecting the sigmoid activation function

$$\sigma(s) = \frac{1}{1 + e^{-s}} \quad (1.3.53)$$

one obtains (see Problems section)

$$\sigma'(s) = \sigma(s)(1 - \sigma(s)) \quad (1.3.54)$$

which is very easy to compute using simple multipliers.

### 1.3.3.2 Derivation of the Backpropagation Algorithm

Backpropagation weight tuning is a gradient descent algorithm, so the weights in layers two and one respectively are updated according to

$$w_{i\ell} = w_{i\ell} - \eta \frac{\partial E}{\partial w_{i\ell}} \quad (1.3.55)$$

$$v_{\ell j} = v_{\ell j} - \eta \frac{\partial E}{\partial v_{\ell j}}, \quad (1.3.56)$$

with  $E$  a prescribed cost function. In this discussion we shall conserve simplicity of notation by dispensing with the iteration index  $k$  (c.f. Section 1.3.2), interpreting these equalities as replacements. The learning rates  $\eta$  in the two layers can of course be selected as different.

Let there be prescribed an input vector  $X$  and an associated desired output vector  $Y$  for the network. Define the least-squares NN output error as

$$E = \frac{1}{2} e^T e = \frac{1}{2} \sum_{i=1}^m e_i^2 \quad (1.3.57)$$

$$e_i = Y_i - y_i, \quad (1.3.58)$$

where  $y_i$  is evaluated using (1.3.41) with the components of the input pattern  $X_j$  as the NN inputs  $x_j$ .

The required gradients of the cost  $E$  with respect to the weights are now very easily determined using the chain rule. Specifically, for the second-layer weights

$$\frac{\partial E}{\partial w_{i\ell}} = \frac{\partial E}{\partial u_i^2} \frac{\partial u_i^2}{\partial w_{i\ell}} = \left[ \frac{\partial E}{\partial e_i} \frac{\partial e_i}{\partial y_i} \frac{\partial y_i}{\partial u_i^2} \right] \frac{\partial u_i^2}{\partial w_{i\ell}} \quad (1.3.59)$$

and using the above equalities one obtains

$$\frac{\partial E}{\partial u_i^2} = -\sigma'(u_i^2)e_i \quad (1.3.60)$$

$$\frac{\partial E}{\partial w_{i\ell}} = -z_\ell [\sigma'(u_i^2)e_i]. \quad (1.3.61)$$

Similarly, for the first-layer weights

$$\frac{\partial E}{\partial v_{\ell j}} = \frac{\partial E}{\partial u_\ell^1} \frac{\partial u_\ell^1}{\partial v_{\ell j}} = \left[ \sum_{i=1}^m \frac{\partial E}{\partial u_i^2} \frac{\partial u_i^2}{\partial z_\ell} \frac{\partial z_\ell}{\partial u_\ell^1} \right] \frac{\partial u_\ell^1}{\partial v_{\ell j}} \quad (1.3.62)$$

and using the above equalities one obtains

$$\frac{\partial E}{\partial u_\ell^1} = -\sigma'(u_\ell^1) \sum_{i=1}^m w_{i\ell} [\sigma'(u_i^2)e_i] \quad (1.3.63)$$

$$\frac{\partial E}{\partial v_{\ell j}} = -X_j \left[ \sigma'(u_\ell^1) \sum_{i=1}^m w_{i\ell} [\sigma'(u_i^2)e_i] \right]. \quad (1.3.64)$$

These equations can be considerably simplified by introducing the notion of a *backward recursion* through the network. Thus, define the *backpropagated error* for layers 2 and 1 respectively as

$$\delta_i^2 \equiv -\frac{\partial E}{\partial u_i^2} = \sigma'(u_i^2)e_i \quad (1.3.65)$$

$$\delta_\ell^1 \equiv -\frac{\partial E}{\partial u_\ell^1} = \sigma'(u_\ell^1) \sum_{i=1}^m w_{i\ell} \delta_i^2. \quad (1.3.66)$$

Assuming the sigmoid activation functions are used, the backpropagated errors can be computed as

$$\delta_i^2 = y_i(1-y_i)e_i \quad (1.3.67)$$

$$\delta_\ell^1 = z_\ell(1-z_\ell) \sum_{i=1}^m w_{i\ell} \delta_i^2. \quad (1.3.68)$$

Combining these equations one obtains the backpropagation algorithm given in Table 1.3.2. There, the algorithm is given in terms of a *forward recursion* through the NN to compute the output, then a *backward recursion* to determine the backpropagated errors, and finally a step to determine the weight updates. Such *two-pass algorithms* are standard in digital signal processing and optimal estimation theory. In fact, one should particularly examine optimal smoothing algorithms contained, for instance, in Lewis (1986). The backpropagation algorithm may be employed using series or batch processing of multiple input/output patterns (previous subsection), and may be modified to use adaptive step size  $\eta$  or momentum training (next subsection).

Note that the threshold updates are given by

$$w_{i0} = w_{i0} + \eta \delta_i^2 \quad (1.3.76)$$

$$v_{\ell 0} = v_{\ell 0} + \eta \delta_\ell^1. \quad (1.3.77)$$

In many applications the NN has no activation functions in the output layer (e.g. the activation function is linear in (1.3.70)). Then one must use simply  $\delta_i^2 = e_i$  in the equations for backpropagation.

In terms of signal vectors and weight matrices one may write the backpropagation algorithm as follows (see Problems section). The forward recursion becomes

$$z = \sigma(V^T X) \quad (1.3.78)$$

$$y = \sigma(W^T z), \quad (1.3.79)$$

the backward recursion is

$$e = Y - y \quad (1.3.80)$$

$$\delta^2 = \text{diag}\{y\} (I - \text{diag}\{y\}) e \quad (1.3.81)$$

$$\delta^1 = \text{diag}\{z\} (I - \text{diag}\{z\}) W \delta^2, \quad (1.3.82)$$

where, with  $y$  an  $m$ -vector,  $\text{diag}\{y\}$  is an  $m \times m$  diagonal matrix having the entries  $y_1, y_2, \dots, y_m$  on the diagonal. The weight and threshold updates are

$$W = W + \eta z (\delta^2)^T \quad (1.3.83)$$

$$V = V + \eta X (\delta^1)^T. \quad (1.3.84)$$

Table 1.3.2: Backpropagation Algorithm Using Sigmoid Activation Functions: Two-Layer Net

The following iterative procedure should be repeated until the NN output error has become sufficiently small. Series or batch processing of multiple input/output patterns ( $X, Y$ ) may be used. Adaptive learning rate  $\eta$  and momentum terms may be added.

*Forward Recursion to Compute NN Output:*

Present input pattern  $X$  to the NN and compute the NN output using:

$$z_\ell = \sigma \left( \sum_{j=0}^n v_{\ell j} X_j \right) ; \quad \ell = 1, 2, \dots, L \quad (1.3.69)$$

$$y_i = \sigma \left( \sum_{\ell=0}^L w_{i\ell} z_\ell \right) ; \quad i = 1, 2, \dots, m \quad (1.3.70)$$

with  $X_0 = 1$  and  $z_0 = 1$ , where  $Y$  is the desired output pattern.

*Backward Recursion for Backpropagated Errors:*

$$e_i = Y_i - y_i ; \quad i = 1, 2, \dots, m \quad (1.3.71)$$

$$\delta_i^2 = y_i(1-y_i)e_i ; \quad i = 1, 2, \dots, m \quad (1.3.72)$$

$$\delta_\ell^1 = z_\ell(1-z_\ell) \sum_{i=1}^m w_{i\ell} \delta_i^2 ; \quad \ell = 1, 2, \dots, L \quad (1.3.73)$$

*Computation of the NN Weight and Threshold Updates:*

$$w_{i\ell} = w_{i\ell} + \eta z_\ell \delta_i^2 ; \quad i = 1, 2, \dots, m; \quad \ell = 0, 1, \dots, L \quad (1.3.74)$$

$$v_{\ell j} = v_{\ell j} + \eta X_j \delta_\ell^1 ; \quad \ell = 1, 2, \dots, L; \quad j = 0, 1, \dots, n \quad (1.3.75)$$

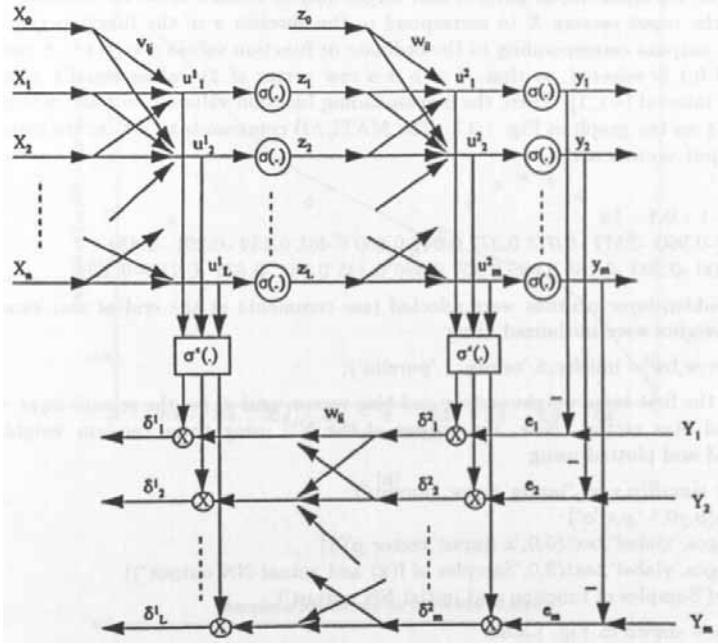


Figure 1.3.4: The adjoint (backpropagation) neural network.

At this point one notices quite an interesting occurrence. The forward recursion of the backpropagation algorithm is based, of course, on the NN weight matrices, however, the backward recursion is based on the *transposes of the weight matrices*. Moreover, it is accomplished by working *backward through the transposed NN*. In system theory the dual, backward system is known as the *adjoint system*. This system enjoys some very special properties in relation to the original system, many associated with determining solutions to optimality and control problems (Lewis and Syrmos 1995). Such notions have not yet been fully explored in the context of NN.

An intriguing concept is that of the *adjoint NN for training*. This ‘backpropagation network’ was discussed by Narendra and Parthasarathy (1990) and is depicted in Fig. 1.3.4. The adjoint training net is based on the transposes of the NN weight matrices and contains *multipliers*. In this respect, it is very similar to various optimal control and adaptive filtering and control schemes wherein the computation and/or tuning of the feedback control gains is carried out in outer loops containing multipliers. The multiplier is fundamental to higher-level and intelligent control. In the 1940’s Norbert Wiener introduced his new field of *Cybernetics*. It was he who said that developments on two fronts were required prior to further advances in system theory: increased computing power and the theory of the multiplier (Wiener 1948).

By now several improvements have been made on the backpropagation algorithm given here. A major increase in speed is offered by the *Levenberg-Marquardt*

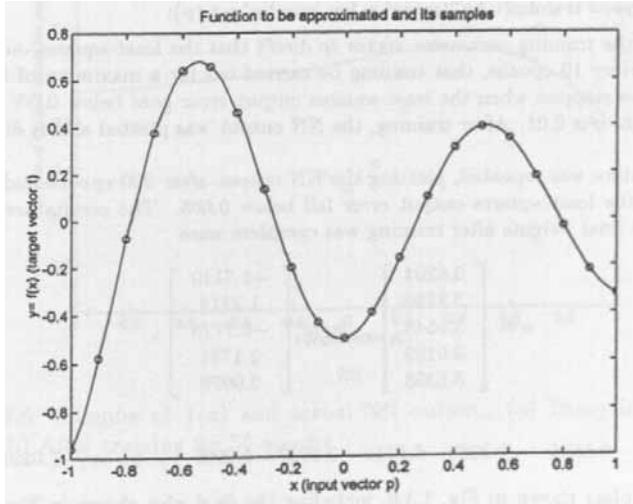


Figure 1.3.5: Function  $y = f(x)$  to be approximated by two-layer NN and its samples for training.

algorithm, which combines gradient descent and the Gauss-Newton algorithm. The next section discusses some other techniques for improving backpropagation.

**Example 1.3.3 (NN Function Approximation) :**

It is known that a two-layer NN with sigmoid activation functions can approximate arbitrarily accurately any smooth function (see Section 1.2.2.). In this example it is desired to design a two-layer NN to approximate the function shown in Fig. 1.3.5, so that the NN has one input  $x$  and one output  $y$ . The hidden-layer activation functions will be the hyperbolic tangent and the output-layer activation functions will be linear.

The NN weights will be determined using backpropagation training with batch updates. First, exemplar input pattern and target output vectors must be selected. Select therefore the input vectors  $X$  to correspond to the abscissa  $x$  of the function graph and the target outputs corresponding to the ordinate or function values  $y = f(x)$ . A sampling interval of 0.1 is selected, so that  $X = p$  is a row vector of 21 values equally spaced at 0.1 on the interval  $[-1, 1]$ . Then, the corresponding function values  $Y = t$  are determined, shown by  $\circ$  on the graph in Fig. 1.3.5. The MATLAB commands to set up the input and target output vectors are

```
p= -1 : 0.1 : 1 ;
t= [-0.960 -0.577 -0.073 0.377 0.641 0.660 0.461 0.134 -0.201 -0.434
-0.500 -0.393 -0.165 0.099 0.307 0.396 0.345 0.182 -0.031 -0.219 -0.320] ;
```

Five hidden-layer neurons were selected (see comments at the end of this example). The NN weights were initialized using

```
[v,bv,w,bw]= initff(p,5,'tansig',1,'purelin');
```

with  $v, bv$  the first-layer weight matrix and bias vector, and  $w, bw$  the second-layer weight matrix and bias vector. Now, the output of the NN using these random weights was determined and plotted using

```

y0= simuff(p,v,bv,'tansig',w,bw,'purelin');
plot(p,y0,'-',p,t,'o')
set(gca,' xlabel',text(0,0,'x (input vector p)'))
set(gca,' ylabel',text(0,0,'Samples of f(x) and actual NN output'))
title('Samples of function and initial NN output')

```

The result is shown in Fig. 1.3.6a.

The NN was now trained using the backpropagation algorithm (1.3.80)-(1.3.84) with batch updating (see (1.3.40)). The MATLAB command is

```

tp= [10 50 .005 .01];
[v,bv,w,bw]= trainbp(v,bv,'tansig',w,bw,'purelin',p,t,tp);

```

The entries of the training parameter vector  $tp$  direct that the least-squares output error be computed every 10 epochs, that training be carried out for a maximum of 50 epochs, that training be stopped when the least-squares output error goes below 0.005, and that the learning rate  $\eta$  is 0.01. After training, the NN output was plotted and is displayed in Fig. 1.3.6b.

This procedure was repeated, plotting the NN output after 200 epochs and after 873 epochs, when the least-squares output error fell below 0.005. The results are shown in Fig. 1.3.6. The final weights after training was complete were

$$v = \begin{bmatrix} 3.6204 \\ 3.8180 \\ 3.5548 \\ 3.0169 \\ 3.6398 \end{bmatrix}, \quad bv = \begin{bmatrix} -2.7110 \\ 1.2214 \\ -0.7778 \\ 2.1751 \\ 2.9979 \end{bmatrix},$$

$$w = [-0.6334 \quad -1.2985 \quad 0.8719 \quad 0.5937 \quad 0.9906], \quad bw = [-1.0295].$$

To obtain the plots shown in Fig. 1.3.6, including the final plot shown in Fig. 1.3.6d, a refined input vector  $p$  was used corresponding to samples at a uniform spacing of 0.01 on the interval  $[-1, 1]$ . The NN output was simply obtained by using MATLAB function *simuff()* with the new  $p$  vector. This shows clearly that, after training, the NN will interpolate between values used in the original  $p$  that was used for training, *determining correct outputs for samples not in the training data*. This important property is known as the *generalization property*, and is closely connected to the associative memory property that close inputs should produce close NN outputs.

The least-squares output error is plotted as a function of training epoch in Fig. 1.3.7.

This example was initially performed using three hidden-layer neurons. It was found that even after several thousand epochs of training, the NN was unable to approximate the function. Therefore, the number of hidden-layer neurons was increased to five and the procedure was repeated. Using MATLAB, it took about 15 minutes to run this entire example and make all plots.  $\square$

#### 1.3.4 Improvements on Gradient Descent

Several improvements can be made to correct deficiencies in gradient descent NN training algorithms. These can be applied at each layer of a multilayer NN when using backpropagation tuning. The two basic issues are that gradient-based minimization algorithms provide only a *local minimum*, and that the verification (1.3.8) that gradient descent decreases the cost function is based on an approximation.

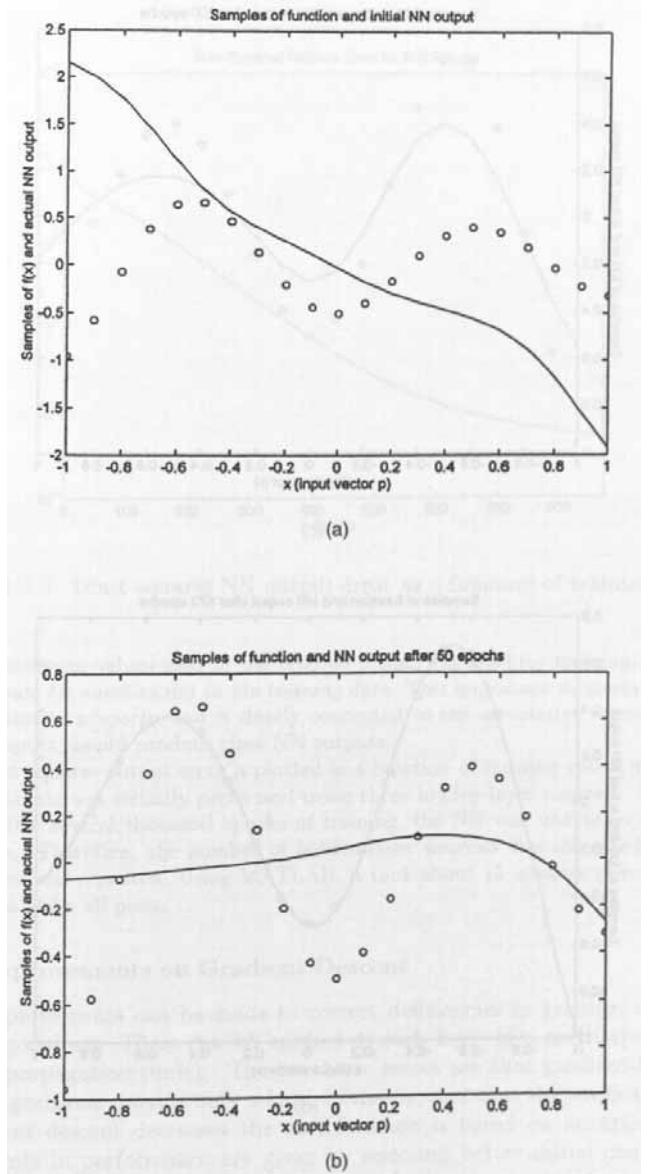


Figure 1.3.6: Samples of  $f(x)$  and actual NN output. (a) Using initial random weights. (b) After training for 50 epochs.

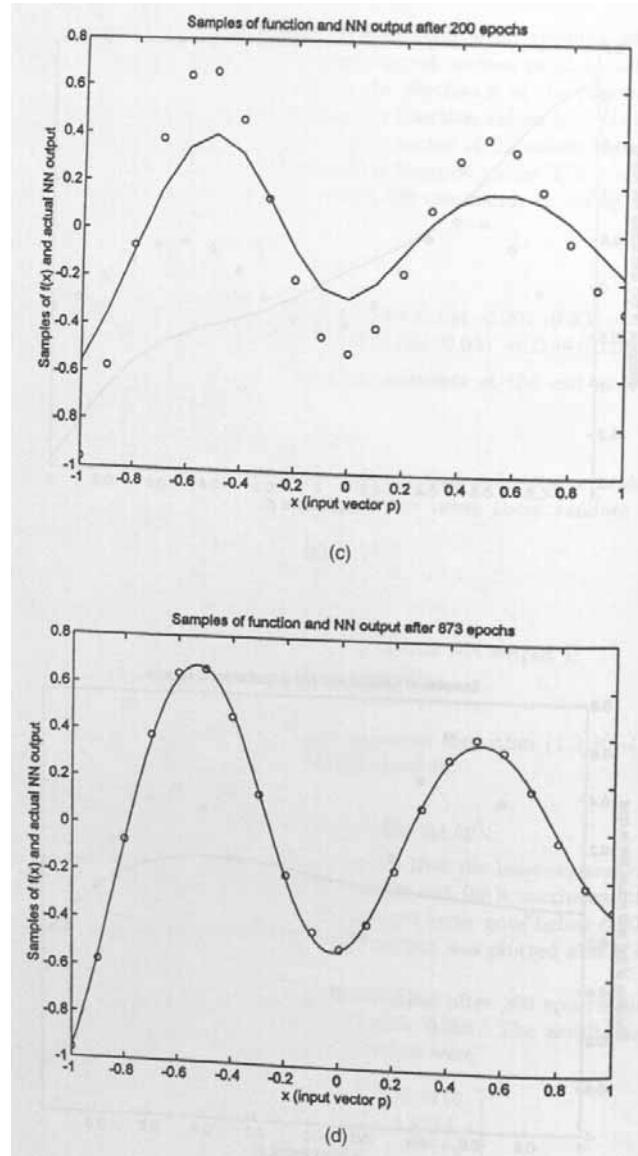


Figure 1.3.6: Samples of  $f(x)$  and actual NN output (cont'd). (c) After training for 200 epochs. (d) After training for 873 epochs.

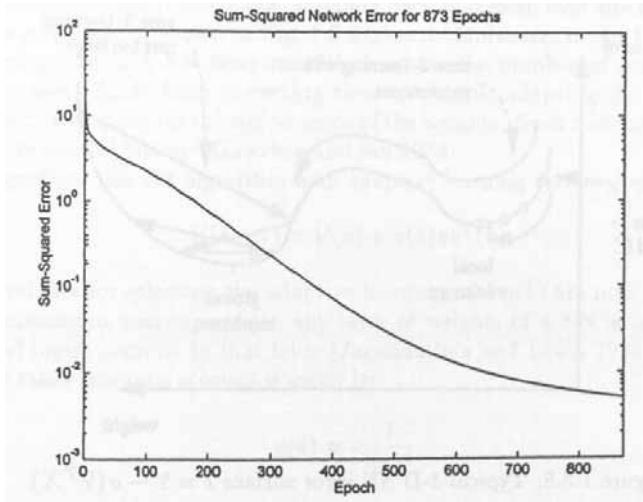


Figure 1.3.7: Least-squares NN output error as a function of training epoch.

Improvements in performance are given by selecting better initial conditions, using learning with ‘momentum’, and using an adaptive learning rate  $\eta$ . References for this section include Goodwin and Sin (1984), Haykin (1994), Kung (1993), and Peretto (1992). All these refinements are available in the MATLAB NN Toolbox (1995).

**Better Initial Conditions.** The NN weights and thresholds are typically initialized to small random (positive and negative) values. A typical error surface graph in 1-D is given in Fig. 1.3.8, which shows a local minimum and a global minimum. If the weight is initialized as shown in Case 1, there is a possibility that the gradient descent algorithm might find the local minimum, rolling downhill to the shallow bowl. Several authors have determined better techniques to initialize the weights than by random selection, particularly for the multilayer NN. Among these are Nguyen and Widrow, whose techniques are used, for instance, in MATLAB. Such improved initialization techniques can also significantly speed up convergence of the weights to their final values.

**Learning with Momentum.** An improved version of gradient descent is given by the *Momentum Gradient Algorithm*

$$V(k+1) = \beta V(k) + \eta(1 - \beta)Xe^T(k), \quad (1.3.85)$$

with positive momentum parameter  $\beta < 1$  and positive learning rate  $\eta < 1$ ;  $\beta$  is generally selected near 1 (e.g. 0.95). This corresponds in discrete-time dynamical system terms to moving the system pole from  $z = 1$  to the interior of the unit circle, and adds stability in a manner similar to friction effects in mechanical systems.

Momentum adds a memory effect so that the NN in effect responds not only to the local gradient, but also to recent trends in the error surface. As shown by the

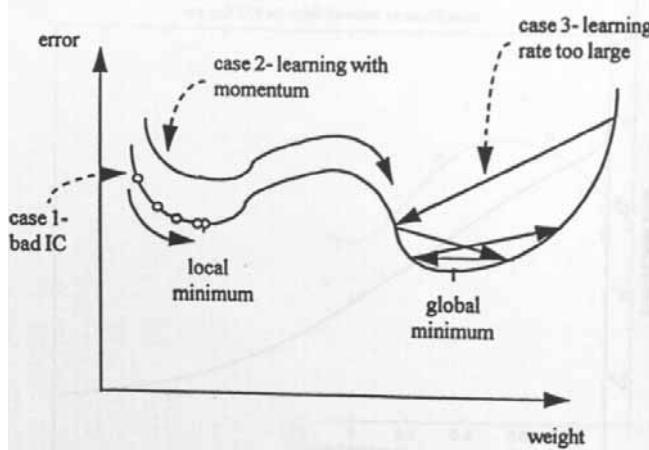


Figure 1.3.8: Typical 1-D NN error surface  $e = Y - \sigma(V^T X)$ .

next example, without momentum the NN can get stuck in a local minimum; adding momentum can help the NN ‘ride through’ local minima. For instance, referring to Fig. 1.3.8, using momentum as in Case 2 will cause the NN to slide through the local minimum, coming to rest at the global minimum. In the MATLAB Neural Network Toolbox are some examples showing that learning with momentum can significantly speed up and improve the performance of backpropagation.

**Adaptive Learning Rate.** If the learning rate  $\eta$  is too large, then the NN can overshoot the minimum cost value, jumping back and forth over the minimum and failing to converge, as shown in Fig. 1.3.8 Case 3. Moreover, it can be shown that the learning rate in a NN layer must decrease as the number of neurons in that layer increases. Aside from correcting these problems, adapting the learning rate can significantly speed up the convergence of the weights. Such notions are standard in adaptive control theory (Goodwin and Sin 1984).

The gradient descent algorithm with adaptive learning rate is given by

$$V(k+1) = V(k) + \eta(k)x e^T(k). \quad (1.3.86)$$

Two techniques for selecting the adaptive learning rate  $\eta(k)$  are now given.

The maximum learning rate in any layer of weights of a NN is limited by the number of input neurons to that layer (Jagannathan and Lewis 1995). A learning rate that takes this into account is given by

$$\eta(k) = \nu \frac{1}{\|z\|^2}, \quad (1.3.87)$$

where  $0 < \nu < 1$  and  $z$  is the input vector to the layer. As the number of input neurons to the layer increases, the norm gets larger (note that  $z \in \mathbb{R}^{L+1}$ , with  $L$  the number of neurons in the input to the layer). This is nothing but the standard ‘projection method’ in adaptive control (Goodwin and Sin 1984).

Another technique to adapt  $\eta$  is given as follows. If the learning rate is too large, the NN can overshoot the minimum and never converge (see Fig. 1.3.8 Case 3). Various standard techniques from optimization theory can be used to correct this problem; they generally rely on reducing the learning rate as a minimum is approached. The following technique increases the learning rate if the cost  $E(k)$  (see (1.3.18)) is decreasing. If the cost increases during any iteration, however, the old weights are retained and the learning step size is repeatedly reduced until the cost decreases on that iteration.

```

1    $V(k+1) = V(k) + \eta(k)x e^T(k)$ 
    If  $E(k+1) < E(k)$  ; retain  $V(k+1)$  and increase learning step size
         $\eta(k+1) = (1+\alpha)\eta(k)$ 
        Go to 2
    If  $E(k+1) > E(k)$  ; reject  $V(k+1)$  and decrease learning step size
         $\eta(k) = (1-\alpha)\eta(k)$ 
        Go to 1
2    $k = k + 1$ 
    Go to next iteration

```

(1.3.88)

The positive parameter  $\alpha$  is generally selected as about 0.05. Various modifications of this technique are possible.

**‘Safe’ Learning Rate.** A ‘safe learning rate’ can be derived as follows. Let  $z$  be the input vector to the layer of weights being tuned, and the number of neurons in the input be  $L$  so that  $z \in \mathbb{R}^{L+1}$ . If the activation function is bounded by 1 (see Fig. 1.1.3), then  $\|z\|^2 < L + 1$ , and the adaptive learning rate (1.3.87) is always bounded below by

$$\eta(k) = \nu \frac{1}{L+1}. \quad (1.3.89)$$

That is, taking  $\nu = 1$  in (1.3.89) provides a safe maximum allowed learning rate in a NN layer with  $L$  input neurons; a safe learning rate  $\eta$  for that layer is less than  $1/(L+1)$ .

### 1.3.5 Hebbian Tuning

In the 1940’s D. O. Hebb proposed a tuning algorithm based on classical conditioning experiments in psychology and by the associative memory paradigm which these observations suggest (Peretto 1992). In this subsection we shall dispense with the iteration index  $k$ , interpreting the weight update equations as replacements. Consider the *one-layer* NN in Fig. 1.1.4 with recall equation

$$y_\ell = \sigma \left( \sum_{j=1}^n v_{\ell j} x_j + v_{\ell 0} \right); \quad \ell = 1, 2, \dots, L. \quad (1.3.90)$$

Suppose first that the NN is to discriminate among  $P$  patterns  $X^1, X^2, \dots, X^P$ , each in  $\mathbb{R}^n$  and having components  $X_i^p; i = 1, 2, \dots, n$ . In this application, the net

is *square* so that  $L = n$ . A pattern  $X^p$  is *stable* if its *stabilization parameters* are all positive:

$$\sum_{j \neq \ell} v_{\ell j} X_{\ell}^p X_j^p > 0 ; \quad \ell = 1, 2, \dots, n. \quad (1.3.91)$$

The stabilization parameters are a measure of how well imprinted the pattern  $X^p$  is with respect to the  $\ell$ -th neuron in a given NN. Define therefore the cost as

$$E = - \sum_{p=1}^P \sum_{j, \ell=1}^n v_{\ell j} X_{\ell}^p X_j^p \quad (1.3.92)$$

which, if minimized, gives large stabilization parameters.

Using this cost in the gradient algorithm (1.3.7) yields the Hebbian tuning rule

$$v_{\ell j} = v_{\ell j} + \eta \sum_{p=1}^P X_{\ell}^p X_j^p. \quad (1.3.93)$$

In matrix terms this may be written as

$$V = V + \eta \sum_{p=1}^P X^p (X^p)^T, \quad (1.3.94)$$

whence it is seen that the update for the weight matrix is given in terms of the *outer product* of the desired pattern vectors. This is a recursive technique in the same spirit as Hopfield's direct computation formula (1.3.3).

Various extensions have been made to this Hebbian or *outer product* training technique in the case of nonsquare NN and multilayer NN. For instance, if  $L \neq n$  in a one-layer net, and the NN is to associate  $P$  patterns  $X^p$ , each in  $\Re^n$ , with  $P$  target outputs  $Y^p$ , each in  $\Re^L$ , a modified Hebbian tuning rule is given by

$$V = V + \eta \sum_{p=1}^P X^p (Y^p)^T, \quad (1.3.95)$$

or by

$$V = V + \eta \sum_{p=1}^P X^p (e^p)^T, \quad (1.3.96)$$

where the output error for pattern  $p$  is given by  $e^p = Y^p - y^p$ , with  $y^p$  the actual NN output given when the NN input is  $X^p$ .

The two-layer NN of Fig. 1.1.6 has the recall equation

$$z = \sigma(V^T x) \quad (1.3.97)$$

$$y = \sigma(W^T z), \quad (1.3.98)$$

with  $z \in \Re^L$  the hidden-layer output vector. Suppose the NN is to associate the input pattern  $X$  to the output vector  $Y$ . Define the output error as  $e = Y - y$ , with

$y$  the output when  $x = X$ . Then, a tuning rule based on the Hebbian philosophy is given by

$$W = W + \eta z e^T \quad (1.3.99)$$

$$V = V + \eta X z^T. \quad (1.3.100)$$

Unfortunately, this multilayer Hebbian training algorithm has not been shown to converge, and has often been documented as leading to problems.

### 1.3.6 Continuous-Time Tuning

We have seen how to update the NN weights by training using discrete iterations in an iteration index  $k$ . This is very convenient for NN training using a digital computer, where iterations are a natural operation. This book is concerned with NN feedback control of dynamical systems, often called the *plant*, of the form

$$\dot{x} = f(x, u) \quad (1.3.101)$$

in continuous time, or

$$x(K+1) = f(x(K), u(K)) \quad (1.3.102)$$

in discrete time, where  $f(\cdot)$  is a generally nonlinear function. Therefore, it is important to understand how the NN weights can be adapted in both discrete-iteration and continuous-time formulations. In fact, the combination of the adaptive NN and the plant being controlled forms a composite dynamical system whose properties can only be understood by analyzing the NN and the plant as a single entity in a feedback control configuration.

The discrete iteration weight update index  $k$  is not necessarily the same as the discrete time index  $K$ , as we have seen in the subsection on batch versus series processing of multiple patterns (end of Section 1.3.2). In continuous-time tuning, on the other hand, the weight tuning is expressed as a differential equation in terms of the time variable  $t$ . All the NN training algorithms derived for the case of discrete iterations can also be expressed in continuous-time terms. Obviously, the equations given in Sections 1.1.1 and 1.1.2 for the static NN are the same however training is accomplished. It is important to note specifically that the properties in Section 1.2 hold no matter how training is performed. In Section 1.1.4 we covered dynamical NN for both the discrete-time and continuous-time cases. In this section we shall show continuous-time formulations of various training algorithms.

#### 1.3.6.1 Gradient Descent Tuning in Continuous Time

Most of the discussion in Section 1.3.2 still holds in continuous-time. Now, the gradient descent algorithm for the one-layer NN becomes

$$\dot{V} = -\eta \frac{\partial E}{\partial V} \quad (1.3.103)$$

with  $E(t)$  the prescribed cost. Given the input-output pair  $(X, Y)$  to be associated by the NN, define the output error

$$e(t) = Y - y(t) = Y - \sigma(V^T(t)X) \in \Re^L \quad (1.3.104)$$

and select the least-squares output-error cost

$$E(t) = \frac{1}{2}e^T(t)e(t) = \frac{1}{2}\text{tr}\{e(t)e^T(t)\}. \quad (1.3.105)$$

Both these quantities are now evaluated continuously as functions of time.

Evaluating the partial derivative and ignoring the derivative of  $\sigma(\cdot)$  (or, equivalently, assuming linear activation functions) yields the continuous-time perceptron training rule

$$\dot{V} = \eta X e^T. \quad (1.3.106)$$

In the case of multiple input/output patterns to be associated, batch updating techniques should be used in continuous time.

Various modifications to this training algorithm are possible, exactly as in Section 1.3.4. Momentum can be added in the form

$$\dot{V} = -\beta V + \eta X e^T, \quad (1.3.107)$$

which, in system theory terms, moves the pole from  $s = 0$  into the left-half plane and adds stability. Now,  $\beta$  should be selected as a small positive number. Various schemes are available for adapting the learning rate  $\eta$  (see Åström and Wittenmark 1989).

### 1.3.6.2 Backpropagation Tuning in Continuous Time

The discussion in Section 1.3.3 still holds in continuous time. It follows that a matrix formulation of the backpropagation algorithm is as given in Table 1.3.3. If the output layer of the NN has no activation functions (e.g. the activation functions in (1.3.109) are linear), then one should use simply  $\delta^2 = e$ .

It is interesting to note that in continuous-time one loses the notion of a forward iteration through the NN followed by a backward iteration through the backpropagation network. However, a backprop network may still be drawn, though the NN and the backprop network now interact continuously through time and all signals are evaluated in a continuous fashion.

### 1.3.6.3 Continuous-Time Hebbian Tuning

The discussion in Section 1.3.5 still holds here. A continuous version of the Hebbian tuning rule given by

$$\dot{V} = \eta \sum_{p=1}^P X^p (X^p)^T \quad (1.3.115)$$

trains a one-layer square (e.g.  $L = n$ ) NN to discriminate among  $P$  patterns  $X^1, X^2, \dots, X^P$ , each in  $\Re^n$ .

Various extensions of this outer-product rule are possible. If  $L \neq n$  in a one-layer net, and the NN is to associate  $P$  patterns  $X^p$ , each in  $\Re^n$ , with  $P$  target outputs  $Y^p$ , each in  $\Re^L$ , a modified Hebbian tuning rule is given by

$$\dot{V} = \eta \sum_{p=1}^P X^p (Y^p)^T, \quad (1.3.116)$$

Table 1.3.3: Continuous-Time Backpropagation Algorithm Using Sigmoid Activation Functions

**Compute NN Output:**

$$z = \sigma(V^T X) \quad (1.3.108)$$

$$y = \sigma(W^T z), \quad (1.3.109)$$

**Compute Filtered Errors:**

$$e = Y - y \quad (1.3.110)$$

$$\delta^2 = \text{diag}\{y\} (I - \text{diag}\{y\}) e \quad (1.3.111)$$

$$\delta^1 = \text{diag}\{z\} (I - \text{diag}\{z\}) W \delta^2, \quad (1.3.112)$$

where, with  $y$  an  $m$ -vector,  $\text{diag}\{y\}$  is an  $m \times m$  diagonal matrix having the entries  $y_1, y_2, \dots, y_m$  on the diagonal.

**Weight Updates:**

$$\dot{W} = \eta z (\delta^2)^T \quad (1.3.113)$$

$$\dot{V} = \eta X (\delta^1)^T. \quad (1.3.114)$$

or by

$$\dot{V} = \eta \sum_{p=1}^P X^p (e^p)^T, \quad (1.3.117)$$

where the output error for pattern  $p$  is given by  $e^p(t) = Y^p - y^p(t)$ , with  $y^p(t) = \sigma(V^T(t)X^p)$  the actual NN output given when the NN input is  $X^p$ .

The two-layer NN of Fig. 1.1.6 has the recall equations

$$z = \sigma(V^T x) \quad (1.3.118)$$

$$y = \sigma(W^T z), \quad (1.3.119)$$

with  $z \in \Re^L$  the hidden-layer output vector. Suppose the NN is to associate the input pattern  $X$  to the output vector  $Y$ . Define the output error as  $e = Y - y$ , with  $y$  the output when  $x = X$ . Then, a continuous-time tuning rule based on the Hebbian philosophy is given by

$$\dot{W} = \eta z e^T \quad (1.3.120)$$

$$\dot{V} = \eta X z^T. \quad (1.3.121)$$

This multilayer Hebbian training algorithm has not been shown to converge.

#### 1.4 REFERENCES

- Abdallah, C.T., "Engineering Applications of Chaos," lecture and personal communication, Nov. 1995.
- Albus, J.S., "A new approach to manipulator control: the Cerebellar Model Articulation Controller (CMAC)," *Trans. ASME J. Dynam. Sys., Meas., Control*, vol. 97, no. 3, pp. 220-227, Sept. 1975.
- Åström, K.J., and B. Wittenmark, *Adaptive Control*, Addison-Wesley, Reading, MA, 1989.
- Barron, A.R., "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Trans. Info. Theory*, vol. 39, no. 3, pp. 930-945, May 1993.
- Becker, K.-H., and M. Dörfler, *Dynamical Systems and Fractals*, Cambridge University Press, Cambridge, 1988.
- Commuri, S., *A Framework for Intelligent Control of Nonlinear Systems*, Ph.D. Dissertation, Dept. Elect. Eng., Univ. Texas at Arlington, Arlington, Texas 76019, May 1996.
- Cybenko, G., "Approximation by superpositions of a sigmoidal function," *Mathematics of Control. Signals and Systems*, vol. 2, no. 4, pp. 303-314, 1989.
- Goodwin, C.G., and K.S. Sin, *Adaptive Filtering, Prediction, and Control*, Prentice-Hall, New Jersey, 1984.
- Haykin, S., *Neural Networks*, IEEE Press and Macmillan, New York, 1994.
- Hornik, K., M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359-366, 1989.
- Hush, D.R., and B.G. Horne, "Progress in supervised neural networks," *IEEE Signal Processing Magazine*, pp. 8-39, Jan. 1993.
- Igelnik, B., and Y.-H. Pao, "Stochastic choice of basis functions in adaptive function approximation and the functional-link net," *IEEE Trans. Neural Networks*, vol. 6, no. 6, pp. 1320-1329, Nov. 1995.
- Jagannathan, S., and F.L. Lewis, "Multilayer discrete-time neural network controller for a class of nonlinear systems", *Proc. IEEE Int. Symp. Intelligent Control*, Monterey, CA, Aug. 1995.
- Kim, Y.H., *Intelligent Closed-Loop Control Using Dynamic Recurrent Neural Network and Real-Time Adaptive Critic*, Ph.D. Dissertation Proposal, Dept. Electrical Engineering, The University of Texas at Arlington, Arlington, Texas 76019, Sept. 1996.

- Kosko, B., *Neural Networks and Fuzzy Systems*, Prentice Hall, New Jersey, 1992.
- Kung, S.Y., *Digital Neural Networks*, Prentice-Hall, New Jersey, 1993.
- Levine, D.S., *Introduction to Neural and Cognitive Modeling*, Lawrence Erlbaum Pub., Hillsdale, New Jersey, 1991.
- Lewis, F.L., *Optimal Estimation*, Wiley, New York, 1986.
- Lewis, F.L., and V.L. Syrmos, *Optimal Control*, second edition, Wiley, New York, 1995.
- Lewis, F.L., C.T. Abdallah, and D.M. Dawson, *Control of Robot Manipulators*, Macmillan, New York, 1993.
- Lippmann, R.P., "An introduction to computing with neural nets," *IEEE ASSP Magazine*, pp. 4-22, April 1987.
- MATLAB version 4.2, July 1994, The Mathworks, Inc., 24 Prime Park Way, Natick, MA 01760, USA.
- MATLAB Neural Network Toolbox, version 2.0, 1995, The Mathworks, Inc., 24 Prime Park Way, Natick, MA 01760, USA.
- Narendra, K.S., "Adaptive control using neural networks," in *Neural Networks for Control*, pp. 115-142. ed. W.T. Miller, R.S. Sutton, P.J. Werbos, Cambridge: MIT Press, 1991.
- Narendra, K.S., "Adaptive Control of dynamical systems using neural networks," in *Handbook of Intelligent Control*, pp. 141-183, ed. D.A. White and D.A. Sofge, New York: Van Nostrand Reinhold, 1992.
- Narendra, K.S., and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4-27, Mar. 1990.
- Narendra, K.S., and K. Parthasarathy, "Gradient methods for the optimization of dynamical systems containing neural networks," *IEEE Trans. Neural Networks*, vol. 2, no. 2, pp. 252-262, Mar. 1991.
- Park, J., and I.W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Comp.*, vol. 3, pp. 246-257, 1991.
- Peretto, P., *An Introduction to the Modeling of Neural Networks*, Cambridge University Press, 1992.
- Rumelhart, D.E., G.E. Hinton, and R.J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, ed. D.E. Rumelhart and J.L. McClelland, Cambridge, MA: MIT Press, 1986.
- Sadegh, N., "A perceptron network for functional identification and control of nonlinear systems," *IEEE Trans. Neural Networks*, vol. 4, no. 6, pp. 982-988, Nov. 1993.

Sanner, R.M., and J.-J.E. Slotine, "Stable adaptive control and recursive identification using radial gaussian networks," *Proc. IEEE Conf. Decision and Control*, Brighton, 1991.

Simpson, P.K., "Foundations of neural networks," in *Artificial Neural Networks, Paradigms, Applications, and Hardware Implementation*, pp. 3-24, ed. E. Sanchez-Sinencio, IEEE Press, 1992.

Werbos, P.J., *Beyond Regression: New Tools for Prediction and Analysis in the Behavior Sciences*, Ph.D. Thesis, Committee on Appl. Math. Harvard Univ., 1974.

Werbos, P.J., "Back propagation: past and future," *Proc. 1988 Int. Conf. Neural Nets*, vol. 1, pp. I343-I353, 1989.

Wiener, N., *Cybernetics: Or Control and Communication in the Animal and the Machine*, MIT Press, Cambridge, 1948.

## 1.5 PROBLEMS

### Section 1.1

**Problem 1.1-1 : Logical Operations Using NN.** A neuron with linear activation function is described by  $y = v_1x_1 + v_2x_2 + v_0$ . Select the weights to design one-layer NN that implement: (a.) the AND operation, (b.) the OR operation, and (c.) the COMPLEMENT.

**Problem 1.1-2 : Discrete-Time Hopfield Net.** Write the discrete-time version of (1.1.29) and draw the corresponding block diagram to Fig. 1.1.14

**Problem 1.1-3 : Hopfield Net Lyapunov Function.** Derive the expression (1.1.32) for the energy function derivative.

**Problem 1.3-4 : Hopfield Net MATLAB Simulation.** Perform a MATLAB simulation of the Hopfield net in Example 1.1.3. Make phase-plane plots for representative initial conditions  $x(0)$  of the vector  $\xi(t)$ . Compare to the phase-plane plots of  $x(t)$  given in the example.

**Problem 1.3-5 : Hopfield Net Lyapunov Function.** Verify the expression for the Lyapunov energy function given in Example 1.1.3.

**Problem 1.1-6 : Discrete-Time Dynamical Neural Net.** Write the discrete-time form of the general dynamics (1.1.34). Draw the corresponding figure.

**Problem 1.1-7 : Neural Net with Internal Neuron Dynamics.** A dynamical NN is given in Fig. 1.5.1. Write down the dynamical equations and show that this is a special case of the general dynamical NN in Fig. 1.1.18.

**Problem 1.1-8 : Neural Net with Outer Feedback Loops.** A dynamical NN is given in Fig. 1.5.2. Write down the dynamical equations and show that this is a special case of the general dynamical NN in Fig. 1.1.18.

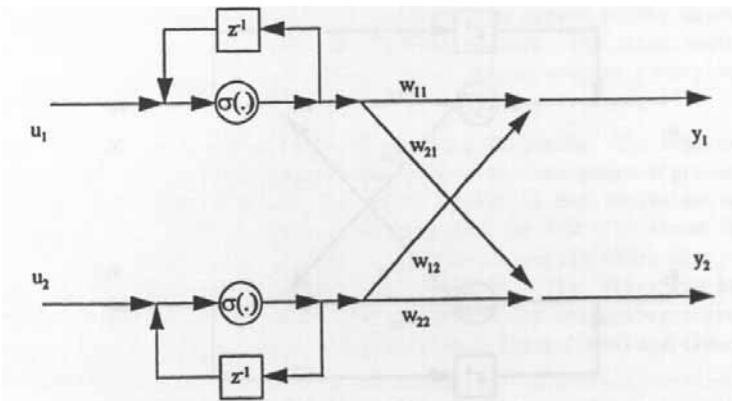


Figure 1.5.1: A dynamical neural network with internal neuron dynamics.

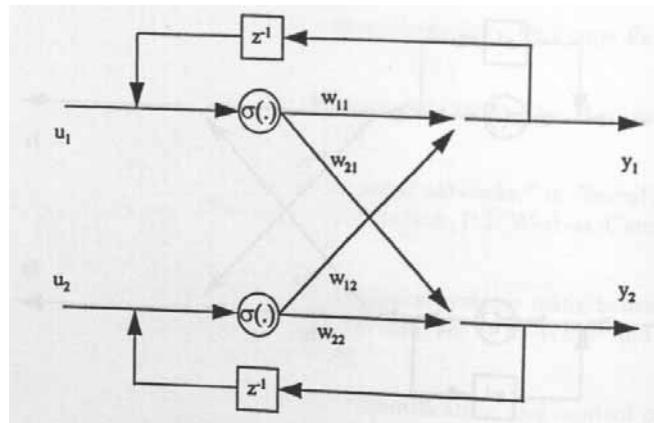


Figure 1.5.2: A dynamical neural network with outer feedback loops.

**Problem 1.1-9 : Cohen-Grossberg Dynamical Net.** The equations of the Cohen-Grossberg net are given by

$$\dot{x}_i = f_i(x_i) \left[ g_i(x_i) - \sum_{j=1}^n w_{ij}\sigma(x_j) \right], \quad (1.5.1)$$

with  $f_i(\cdot), g_i(\cdot)$  nonlinear functions specified by the designer. Draw a figure of the NN. Is this a special case of the general dynamics (1.1.34)?

**Problem 1.1-10 : Chaotic Behavior in Discrete-Time Neural Networks.** In Example 1.1.4 some chaotic-type behavior was displayed for a simple discrete-time NN. Perform some experimentation with this system, making phase plane plots for various modifications of the plant and NN weight matrices. Try different activation functions. Try plots using  $N= 2000$ ,  $N= 4000$ , etc. Is this in fact chaotic behavior? Extended study of this topic could easily lead to a Ph.D. thesis.

### Section 1.3

**Problem 1.3-1 : Hopfield Net Equilibria.** Consider (1.3.2) with  $\sigma(\cdot)$  the symmetric hard limiter. Show that, using the weights (1.3.3), the equilibrium points of the Hopfield net are given by the exemplar pattern vectors  $X^p$ . Note that at steady-state  $\dot{x} = 0$ .

**Problem 1.3-2 : Hopfield Net Equilibria.** Repeat the previous problem using an energy or Lyapunov function approach. Take the Hopfield net with symmetric sigmoid activation functions in Examples 1.1.3 and 1.3.1. Show that with the selected weight matrix, the minima of  $L = -x^T W x$  in the region  $[-1, 1] \times [-1, 1]$  occur at  $x = [-1 \ -1]^T, x = [1 \ 1]^T$ .

**Problem 1.3-3 : EXCLUSIVE-OR.** A two-layer NN that implements the X-OR operation is given in Fig. 1.1.7. Use the MATLAB NN Toolbox to train a NN to implement this operation and compare your answer to the figure. Begin with random weights and train using backpropagation. The input vectors  $x$  are  $[0 \ 0]^T, [0 \ 1]^T, [1 \ 0]^T, [1 \ 1]^T$ , and the associated desired outputs  $y$  are given by the definition of X-OR.

**Problem 1.3-4 : Second-Order NN Tuning Methods.** The arguments connected with (1.3.8) were designed to demonstrate the convergence of gradient-based NN tuning algorithms. However, the second equality in that derivation is only an approximate one. Write a Taylor series expansion for  $E(k+1)$  about  $E(k)$  that includes second-order terms. Propose a modified tuning algorithm that takes into account the Hessian matrix of  $E(k)$  with respect to  $v_{\ell j}(k)$ . When you have completed this problem, examine the Newton's method and conjugate-gradient tuning algorithms as given, for instance, in Haykin (1994), Kung (1993) and Goodwin and Sin (1984).

**Problem 1.3-5 : Matrix-Calculus-Based Derivation of Gradient Descent.** Perform in detail all steps in the derivation of (1.3.23) using matrix techniques.

**Problem 1.3-6 : Batch NN Weight Updating.** Let  $P$  desired input/output pairs  $(X^1, Y^1), (X^2, Y^2), \dots, (X^P, Y^P)$  be prescribed for the NN. In batch updating, all  $P$  pairs are presented to the NN and a cumulative error is computed. At

the end of this procedure, the NN weights are updated once using (1.3.36). Derive this update law by defining the cumulative error for one epoch as

$$E(k) = \sum_{p=1}^P E^p(k) \quad (1.5.2)$$

with  $E^p(k)$  given in (1.3.34). Use matrix calculus for the derivation.

**Problem 1.3-7 : NN Output Surface as a Function of Training Epoch.** In Example 1.1.1 it was shown how to plot the output surface, (i.e. the output plotted as a function of the inputs) for a neural network with two inputs  $x_1, x_2$  and one output  $y$ . Example 1.3.2 provided a classification example. For that example, plot the output surfaces for  $y_1$  and  $y_2$  for several stages in the training. For instance, plot the output surfaces after 0, 3, and 6 epochs.

**Problem 1.3-8 : Programming the Perceptron Algorithm.** Though the MATLAB Neural Network Toolbox (1995) has functions that perform the algorithms required for training the one-layer NN, it is instructive to write one's own program. This is very simple using matrix formulations. Thus, write a MATLAB M file to implement the perceptron training algorithm (1.3.23). Using your program, duplicate Example 1.3.2.

**Problem 1.3-9 : Activation Function Derivatives.** The activation function derivatives are required to implement backpropagation training. (a) Derive the sigmoid derivative (1.3.54). (b) Derive the tanh derivative. (c) Derive the RBF derivative.

**Problem 1.3-10 : Backpropagation Using Tanh and RBF Activation Functions.** Derive the backpropagation algorithm using: (a) Tanh activation functions. (b) RBF activation functions.

**Problem 1.3-11 : Matrix Formulation of the Backpropagation Algorithm.** Verify the matrix formulation of backpropagation in Equations (1.3.80)ff.

**Problem 1.3-12 : Backpropagation Derivation Using Matrix Calculus.** Use a matrix calculus approach to derive the backpropagation algorithm as was done for gradient descent (c.f. (1.3.23)).

**Problem 1.3-13 : Programming the Backpropagation Algorithm.** Though the MATLAB Neural Network Toolbox (1995) has functions that perform the algorithms required for training the multilayer NN, it is instructive to write one's own program. This is straightforward using matrix formulations. Thus, write a MATLAB M file to implement the backpropagation training algorithm given in Table 1.3.2. Using your program, duplicate Example 1.3.3.

**Problem 1.3-14 : Modified Backpropagation Algorithm.** A modified backpropagation algorithm is obtained by changing the order of the operations in Table 1.3.2. Thus, suppose the backpropagated error and updated weights are computed in the interleaved fashion

$$\delta_i^2 = y_i(1 - y_i)e_i ; \quad i = 1, 2, \dots, m \quad (1.5.3)$$

$$w_{i\ell} = w_{i\ell} + \eta z_\ell \delta_i^2 ; \quad i = 1, 2, \dots, m; \quad \ell = 0, 1, \dots, L \quad (1.5.4)$$

$$\delta_\ell^1 = z_\ell(1 - z_\ell) \sum_{i=1}^m w_{i\ell} \delta_i^2 ; \quad \ell = 1, 2, \dots, L \quad (1.5.5)$$

$$v_{\ell j} = v_{\ell j} + \eta X_j \delta_\ell^1 ; \quad \ell = 1, 2, \dots, L; \quad j = 0, 1, \dots, n \quad (1.5.6)$$

where the *new* layer-2 weights  $w_{i\ell}$  are used to compute the layer-1 backpropagated error  $\delta_\ell^1$ . Justify this algorithm (or argue against it) using partial derivative/chain rule arguments. Would you expect this algorithm to perform better or worse than standard backpropagation?

**Problem 1.3-15 : Programming the Modified Backpropagation Algorithm.** Write a MATLAB M file to implement the modified backpropagation training algorithm given in the previous problem . Using your program, perform Example 1.3.3. Does the modified algorithm converge faster than standard backpropagation?

**Problem 1.3-16 : Backpropagation for  $N$ -layer Neural Network.** Streamline the notation in Table 1.3.2, for instance by defining an  $N$ -layer NN with weights  $w_{ij}^q$ , or weight matrices  $W^q$ , in layer  $q$ . Derive the backpropagation algorithm for the  $N$ -layer NN, which should be simply a recursion (c.f. Do loop) in the index  $p$ .

**Problem 1.3-17 : Backpropagation and Optimal Smoothing.** The forward/backward nature of the backpropagation algorithm as given in Table 1.3.2 is very similar to optimal smoothing algorithms. Examine a book on Optimal Estimation (e.g. Lewis 1986) and pursue this further. In particular, can you derive a streamlined formulation of the backpropagation algorithm that is similar to the Rauch-Tung-Striebel smoother?

**Problem 1.3-18 : Continuous-Time Backpropagation Algorithm.** Derive the continuous-time backpropagation algorithm given in Table 1.3.3.



## Chapter 2

# Background on Dynamic Systems

In this chapter we provide a brief background on dynamical systems, mainly covering the topics that will be important in a discussion of neural network (NN) applications in closed-loop control of dynamical systems. It is common for computer science engineers working in NN system and control applications to have little understanding of feedback control and dynamical systems. Many of the phenomena they observe are due not to properties of NN but to properties of feedback control systems. NN applications in dynamical systems is a complex area with several facets, an incomplete understanding of any one of which leads to incorrect conclusions being drawn, with inaccurate attributions of causes—many are convinced that the often exceptional regulatory and behavioral phenomena observed in NN control systems are completely due to the NN, while in fact most are due to the rather remarkable nature of feedback control in itself.

Included in this chapter are continuous-time and discrete-time systems, computer simulation, norms, stability and passivity definitions, stability analysis techniques including Lyapunov approaches and the Bellman-Gronwall Lemma, and feedback linearization. More information is available, for instance, in Khalil (1992), Vidyasagar (1993), Lewis, Abdallah, and Dawson (1993), Slotine and Li (1991), Ioannou and Sun (1996), and Qu and Dawson (1996).

### 2.1 DYNAMICAL SYSTEMS

Many systems in nature, including biological systems, are *dynamical* in the sense that they are acted upon by external inputs, have internal memory, and behave in certain ways that can be captured by the notion of the development of activities through time. The notion of *system* was formalized in the early 1900's by Alfred North Whitehead (1953) and L. von Bertalanffy (1968). A system is viewed here as an entity distinct from its environment, whose interactions with the environment can be characterized through *input* and *output* signals. An intuitive feel for dynamic systems is provided by Luenberger (1979), which has many excellent examples.

### 2.1.1 Continuous-Time Systems

A very general class of continuous-time systems can be described by the nonlinear ordinary differential equation in *state-space form*

$$\begin{aligned}\dot{x} &= F(x, u) \\ y &= H(x, u),\end{aligned}\tag{2.1.1}$$

where  $x(t) \in \mathbb{R}^n$  is the internal *state vector*,  $u(t) \in \mathbb{R}^m$  is the *control input*, and  $y(t) \in \mathbb{R}^p$  is the measured *system output*. Overdot represents differentiation with respect to time  $t$ . The first equation, the *state equation*, captures the dynamical portion of the system and has memory inherent in the  $n$  integrators. It may be derived, for instance, from the physics of the system by using Lagrangian or Hamiltonian dynamics. The second equation, called the *output* or measurement equation, represents how we chose to measure the system variables; it depends on the type and availability of sensors. This state equation can describe a variety of dynamical behaviors, including mechanical and electrical systems, earth atmosphere dynamics, planetary orbital dynamics, aircraft systems, population growth dynamics, and chaotic behavior (see Problems section).

#### 2.1.1.1 Brunovsky Canonical Form

Letting  $x = [x_1 \ x_2 \ \dots \ x_n]^T$ , a special form of nonlinear continuous-time dynamics is given by the class of systems in *Brunovsky canonical form*

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= x_3 \\ &\vdots \\ \dot{x}_n &= f(x) + g(x)u\end{aligned}\tag{2.1.2}$$

$$y = h(x).\tag{2.1.3}$$

As seen from Fig. 2.1.1 this is a chain or cascade of integrators  $\frac{1}{s}$ ; each integrator stores information and requires an initial condition. The internal state can be viewed as the initial information required to specify a unique solution of the differential equation. The measured output  $y(t)$  can be a general function of the states as shown, or can have more specialized forms such as

$$y = h(x_1).\tag{2.1.4}$$

The Brunovsky canonical form may equivalently be written as

$$\dot{x} = Ax + bf(x) + bg(x)u\tag{2.1.5}$$

where

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ & & \vdots & & \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}, b = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}.\tag{2.1.6}$$

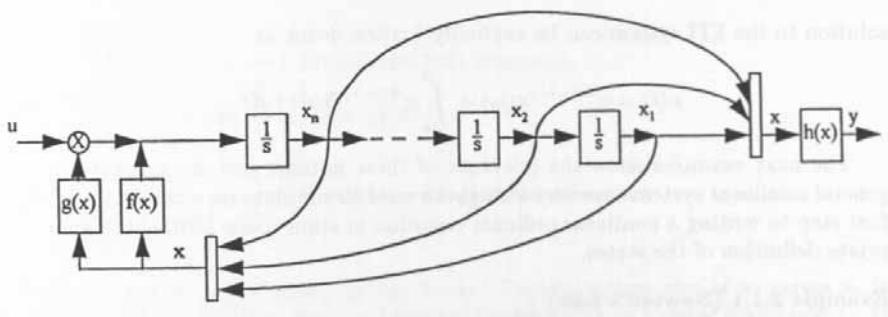


Figure 2.1.1: Continuous-time single-input Brunovsky form.

A more general Brunovsky canonical form occurs where the input is an  $m$ -vector

$$u(t) = [u_1 \ u_2 \dots u_m]^T \text{ and}$$

$$\begin{aligned} \dot{x}_1 &= x_2 & \dot{x}_{n_1+1} &= x_{n_1+2} \\ \dot{x}_2 &= x_3 & \dot{x}_{n_1+2} &= x_{n_1+3} \\ &\vdots & &\vdots \\ \dot{x}_{n_1} &= f_1(x) + g_1(x)u_1 & \dot{x}_{n_1+n_2} &= f_2(x) + g_2(x)u_2 \\ && \vdots & \\ && \dot{x}_{n_1+n_2+\dots+n_{m-1}+1} &= x_{n_1+n_2+\dots+n_{m-1}+2} \\ && \dot{x}_{n_1+n_2+\dots+n_{m-1}+2} &= x_{n_1+n_2+\dots+n_{m-1}+3} \\ && &\vdots \\ && \dot{x}_{n_1+n_2+\dots+n_m} &= f_m(x) + g_m(x)u_m. \end{aligned} \quad (2.1.7)$$

The *multi-input Brunovsky form* is a system with  $m$ -parallel chains of integrators of lengths  $n_1, n_2, \dots$ , each driven by one of the control inputs. Note that  $n = n_1 + n_2 + \dots + n_m$ .

Many systems occur naturally in Brunovsky form (see subsequent examples). Moreover, it is often possible to transform general systems of the form (2.1.1) to Brunovsky form. This is accomplished by finding a suitable state-space transformation followed by an input transformation (Slotine and Li 1991, Isidori 1989). To be transformable to Brunovsky form, the system must satisfy two properties: a controllability condition (Section 2.4) and a condition known as involutivity. Gen-

erally, the controllability condition holds for practically occurring systems. The involutivity condition is more difficult to satisfy but holds for many useful systems.

### 2.1.1.2 Linear Systems

A special and important class of dynamical systems are the *linear time-invariant (LTI) systems*

$$\dot{x} = Ax + Bu \quad (2.1.8)$$

$$y = Cx, \quad (2.1.9)$$

with  $A, B, C$  constant matrices of general form (e.g., not restricted to (2.1.6)). An LTI system is denoted  $(A, B, C)$ . Given an initial time  $t_0$  and initial state  $x(t_0)$  the solution to the LTI system can be explicitly written down as

$$x(t) = e^{A(t-t_0)}x(t_0) + \int_{t_0}^t e^{A(t-\tau)}Bu(\tau) d\tau. \quad (2.1.10)$$

The next examples show the relevance of these notions and demonstrate that general nonlinear systems are very straightforward to simulate on a computer. The first step to writing a nonlinear ordinary equation in state-space form is the appropriate definition of the states.

#### Example 2.1.1 (Newton's Law) :

Newton's law  $F = ma$  can be written as

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= u\end{aligned}$$

where the states are defined as  $x_1 = \text{position}$ ,  $x_2 = \text{velocity}$ , and the control input is  $u = F/m$ , with  $m$  the mass and  $F(t)$  the input force.  $\square$

#### Example 2.1.2 (Simulation of Dynamical Systems— Van der Pol Oscillator) :

One of the most popular examples of nonlinear systems is the van der Pol oscillator which has dynamics

$$\ddot{y} + \alpha(y^2 - 1)\dot{y} + y = u.$$

##### a. Brunovsky Canonical Form

The van der Pol dynamics can be written in Brunovsky form by defining the states as  $x_1 = \text{position}$ ,  $x_2 = \text{velocity}$ . Then,

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= \alpha(1-x_1^2)x_2 - x_1 + u.\end{aligned}$$

##### b. Computer Simulation

The nonlinear state-space equations are very easy to simulate on a digital computer. First, it is necessary to have a *numerical integration routine* such as Runge-Kutta. Techniques such as Euler's method, Adams-Bashforth and so on are usually not sufficiently accurate. Integration routines can easily be written in FORTRAN or C. MATLAB (1994) provides the Runge-Kutta routines ODE23 (third-order) and ODE45 (fourth order); the former usually suffices. These routines have adaptive step sizes.

To use the MATLAB numerical integration routines, the system dynamics must be written into an M file. The state-space description makes this very direct; in fact, ODE23 requires the dynamics in state-space form (2.1.1). For the van der Pol oscillator the required M file is:

```
% MATLAB .M file for van der Pol oscillator
function xdot= vdpol(t,x)
alpha= 0.8; u= 0;
xdot= [x(2) ; alpha*(1-x(1)^2)*x(2) - x(1) + u];
```

where it is assumed that  $u(t) = 0$ . Now, the sequence of commands required to invoke ODE23 and obtain time history plots, for instance over a time horizon of 50 sec, is:

```
t0=0 ; tf= 50; % time horizon
x0= [0.1 ; 0.1]'; % initial conditions
[t,x]= ode23('vdpol',t0,tf,x0);
% Time History Plot:
plot(t,x)
title('Van der Pol Oscillator Time Histories')
xlabel('time (sec)')
ylabel('states x1(t) and x2(t)')
% Phase-Plane Plot:
plot(x(:,1),x(:,2))
title('Van der Pol Oscillator Phase Plane Plot')
xlabel('x(1)')
ylabel('x(2)')
```

The time history plot is shown in Fig. 2.1.2a. The phase-plane plot of  $x_2$  versus  $x_1$  is shown in Fig. 2.1.2b. Note the convergence of the trajectory to a stable limit cycle.  $\square$

### 2.1.2 Discrete-Time Systems

If the time index is an integer  $k$  instead of a real number  $t$  the system is said to be of *discrete-time*. A very general class of discrete-time systems can be described by the nonlinear ordinary difference equation in *discrete state-space form*

$$\begin{aligned} x(k+1) &= F(x(k), u(k)) \\ y(k) &= H(x(k), u(k)), \end{aligned} \quad (2.1.11)$$

where  $x(k) \in \mathbb{R}^n$  is the internal state vector,  $u(k) \in \mathbb{R}^m$  is the control input, and  $y(k) \in \mathbb{R}^p$  is the measured system output.

These equations may either be derived directly from an analysis of the dynamical process being studied, or they may be *sampling* or discretized versions of continuous-time dynamics in the form (2.1.1). Today, controllers are generally implemented in digital form so that a discrete-time description of the controller is needed. This may be determined by design based on the discrete-time system dynamics. Sampling of linear systems is well understood since the work of Ragazzini, Franklin, and others in the 1950s, with many design techniques available. However, sampling of nonlinear systems is not an easy topic. In fact, the exact discretization of nonlinear continuous dynamics is based on the Lie derivatives and leads to an infinite series representation (see e.g. Kalkkuhl and Hunt 1996). Various approximate discretization techniques use truncated versions of the exact series.

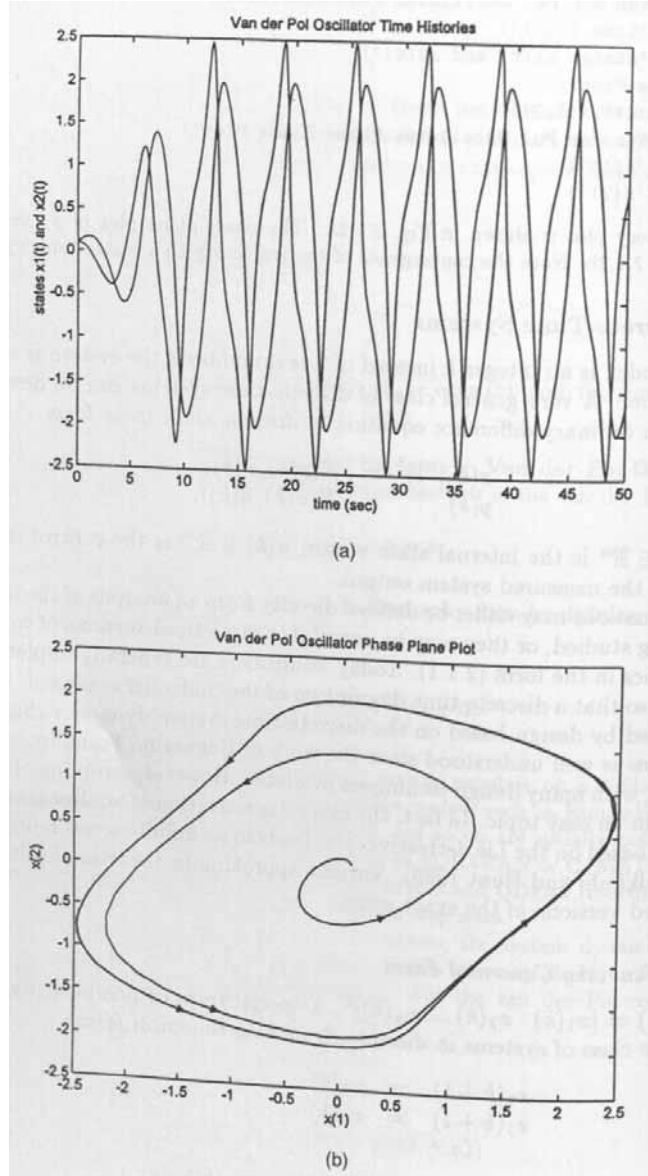


Figure 2.1.2: Van der Pol Oscillator time history plots. (a)  $x_1(t)$  and  $x_2(t)$  versus  $t$ . (b) Phase-plane plot  $x_2$  versus  $x_1$  showing limit cycle.

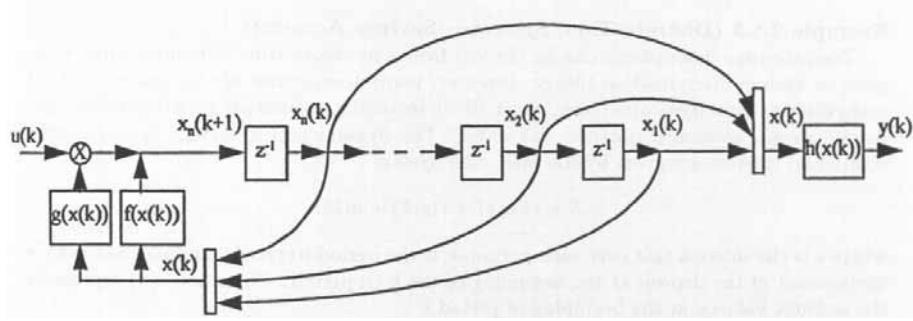


Figure 2.1.3: Discrete-time single-input Brunovsky form.

2.1.2.1 *Brunovsky Canonical Form*

Letting  $x(k) = [x_1(k) \ x_2(k) \dots x_n(k)]^T$ , a special form of nonlinear dynamics is given by the class of systems in *discrete Brunovsky canonical form*

$$\begin{aligned} x_1(k+1) &= x_2(k) \\ x_2(k+1) &= x_3(k) \\ &\vdots \\ x_n(k+1) &= f(x(k)) + g(x(k))u(k) \end{aligned} \tag{2.1.12}$$

$$y(k) = h(x(k)). \tag{2.1.13}$$

As seen from Fig. 2.1.3 this is a chain or cascade of unit delay elements  $z^{-1}$ , i.e. a *shift register*. Each delay element stores information and requires an initial condition. The measured output  $y(k)$  can be a general function of the states as shown, or can have more specialized forms such as

$$y(k) = h(x_1(k)). \tag{2.1.14}$$

The discrete Brunovsky canonical form may equivalently be written as

$$x(k+1) = Ax(k) + bf(x(k)) + bg(x(k))u(k) \tag{2.1.15}$$

where  $A, b$  are given by (2.1.6). A discrete-time form of the more general version (2.1.7) may also be written (see Problems section). It is a system with  $m$ -parallel chains of delay elements of lengths  $n_1, n_2, \dots$  (e.g.  $m$  shift registers), each driven by one of the control inputs.

Many practical systems occur in the *continuous-time* Brunovsky form. However, if a system of the continuous Brunovsky form (2.1.2) is sampled, the result is not generally a system in discrete Brunovsky form (2.1.12), but a discrete system in the general form (2.1.11). Under certain conditions, general discrete-time systems of the form (2.1.11) can be converted to discrete Brunovsky canonical form systems (see e.g. Kalkkuhl and Hunt 1996).

### 2.1.2.2 Linear Systems

A special and important class of dynamical systems are the *discrete-time linear time-invariant (LTI) systems*

$$x(k+1) = Ax(k) + Bu(k) \quad (2.1.16)$$

$$y(k) = Cx(k), \quad (2.1.17)$$

with  $A, B, C$  constant matrices of general form (e.g., not restricted to (2.1.6)). An LTI system is denoted  $(A, B, C)$ . Given an initial state  $x(0)$  the solution to the LTI system can be explicitly written as

$$x(k) = A^k x(0) + \sum_{j=0}^{k-1} A^{k-j-1} Bu(j). \quad (2.1.18)$$

The next example shows the relevance of these notions and demonstrates that general discrete-time nonlinear systems are even easier to simulate on a computer than continuous-time systems, as no integration routine is needed.

#### **Example 2.1.3 (Discrete-Time System— Savings Account) :**

Discrete-time descriptions can be derived from continuous-time dynamics using sampling or system discretization theory. However, many phenomena are naturally modeled using discrete-time dynamics (Luenberger 1979), including population growth/decline, epidemic spread, economic systems, and so on. The dynamics of a savings account using compound interest are given by the first-order system

$$x(k+1) = (1+i)x(k) + u(k),$$

where  $i$  is the interest rate over each period,  $k$  is the period iteration number, and  $u(k)$  is the amount of the deposit at the beginning of the  $k$ -th period. The state  $x(k)$  represents the account balance at the beginning of period  $k$ .

#### a. Analysis

According to (2.1.18), if equal annual deposits are made of  $u(k) = d$ , the account balance is

$$x(k) = (1+i)^k x(0) + \sum_{j=0}^{k-1} (1+i)^{k-j-1} d,$$

with  $x(0)$  the initial amount in the account. Using the standard series summation formula

$$\sum_{j=0}^{k-1} a^j = \frac{1-a^k}{1-a}$$

one derives

$$\begin{aligned} x(k) &= (1+i)^k x(0) + d(1+i)^{k-1} \sum_{j=0}^{k-1} \frac{1}{(1+i)^j} \\ x(k) &= (1+i)^k x(0) + d(1+i)^{k-1} \left[ \frac{1 - \frac{1}{(1+i)^k}}{1 - \frac{1}{(1+i)}} \right] \\ x(k) &= (1+i)^k x(0) + d \left[ \frac{(1+i)^k - 1}{i} \right], \end{aligned}$$

the standard formula for complex interest with constant annuities of  $d$ .

### b. Simulation

It is very easy to simulate a discrete-time system. No numerical integration driver program is needed, in contrast to the continuous-time case. Instead, a simple ‘do loop’ can be used. A complete MATLAB program that simulates the compound interest dynamics is given by:

```
% Discrete-Time Simulation program for Compound Interest Dynamics
d= 100; i= .08; % 8% interest rate
x(1)= 1000;
for k= 1:100;
    x(k+1)= (1+i)*x(k) + d;
end
k=[1:101];
plot(k,x)
```

□

## 2.2 SOME MATHEMATICAL BACKGROUND

### 2.2.1 Vector and Matrix Norms

We assume the reader is familiar with norms, both vector and induced matrix norms (Lewis, Abdallah, and Dawson 1993). We denote any suitable vector norm by  $\|\cdot\|$ . When it is required to be specific we denote the  $p$ -norm by  $\|\cdot\|_p$ . Recall that for any vector  $x \in \Re^n$

$$\|x\|_1 = \sum_{i=1}^n |x_i| \quad (2.2.1)$$

$$\|x\|_p = \left( \sum_{i=1}^n |x_i|^p \right)^{1/p} \quad (2.2.2)$$

$$\|x\|_\infty = \max_i |x_i|. \quad (2.2.3)$$

The 2-norm is the standard Euclidean norm.

Given a matrix  $A$ , its induced  $p$ -norm is denoted  $\|A\|_p$ . Letting  $A = [a_{ij}]$ , recall that the induced 1-norm is the maximum absolute column sum

$$\|A\|_1 = \max_j \sum_i |a_{ij}| \quad (2.2.4)$$

and the induced  $\infty$ -norm is the maximum absolute row sum

$$\|A\|_\infty = \max_i \sum_j |a_{ij}|. \quad (2.2.5)$$

The induced matrix  $p$ -norm satisfies the inequality, for any vector  $x$ ,

$$\|Ax\|_p \leq \|A\|_p \|x\|_p, \quad (2.2.6)$$

and for any two matrices  $A, B$  one also has

$$\|AB\|_p \leq \|A\|_p \|B\|_p. \quad (2.2.7)$$

Given a matrix  $A = [a_{ij}]$ , the *Frobenius norm* is defined as the root of the sum of the squares of all elements:

$$\|A\|_F^2 \equiv \sum a_{ij}^2 = \text{tr}(A^T A), \quad (2.2.8)$$

with  $\text{tr}(\cdot)$  the matrix *trace* (i.e., sum of diagonal elements). Though the Frobenius norm is not an induced norm, it is *compatible* with the vector 2-norm so that

$$\|Ax\|_2 \leq \|A\|_F \|x\|_2. \quad (2.2.9)$$

**Singular Value Decomposition.** The matrix norm  $\|A\|_2$  induced by the vector 2-norm is the *maximum singular value* of  $A$ . For a general  $m \times n$  matrix  $A$ , one may write the *singular value decomposition (SVD)*

$$A = U\Sigma V^T \quad (2.2.10)$$

where  $U$  is  $m \times m$ ,  $V$  is  $n \times n$ , and both are *orthogonal*, that is

$$\begin{aligned} U^T U &= UU^T = I_m \\ V^T V &= VV^T = I_n \end{aligned} \quad (2.2.11)$$

with  $I_n$  the  $n \times n$  identity matrix. The  $m \times n$  singular value matrix has the structure

$$\Sigma = \text{diag}\{\sigma_1, \sigma_2, \dots, \sigma_r, 0, \dots, 0\} \quad (2.2.12)$$

where  $r$  is the rank of  $A$  and  $\sigma_i$  are the *singular values* of  $A$ . It is traditional to arrange the singular values in nonincreasing order, so that the *largest singular value* is  $\sigma_{\max}(A) = \sigma_1$ . If  $A$  has full rank, then  $r$  is equal to either  $m$  or  $n$ , whichever is smaller. Then the minimum singular value is  $\sigma_{\min}(A) = \sigma_r$  (otherwise the minimum singular value is equal to zero).

The SVD generalizes the notion of eigenvalues to general nonsquare matrices. The singular values of  $A$  are the (positive) square roots of the nonzero eigenvalues of  $AA^T$ , or equivalently of  $A^T A$  (see Problems section).

**Quadratic Forms and Definiteness.** Given an  $n \times n$  matrix  $Q$  the *quadratic form*  $x^T Q x$ , with  $x$  an  $n$ -vector, will be important for stability analysis in this book. The quadratic form can in some cases have certain properties that are independent of the vector  $x$  selected. Four important definitions are:

$$\begin{array}{lll} Q \text{ is positive definite, denoted } Q > 0, \text{ if} & x^T Q x > 0, \forall x \neq 0 \\ Q \text{ is positive semidefinite, denoted } Q \geq 0, \text{ if} & x^T Q x \geq 0, \forall x \\ Q \text{ is negative definite, denoted } Q < 0, \text{ if} & x^T Q x < 0, \forall x \neq 0 \\ Q \text{ is negative semidefinite, denoted } Q \leq 0, \text{ if} & x^T Q x \leq 0, \forall x. \end{array} \quad (2.2.13)$$

If  $Q$  is symmetric, then it is positive definite if, and only if, all its eigenvalues are positive, and positive semidefinite if, and only if, all eigenvalues are nonnegative. If  $Q$  is not symmetric, tests are more complicated and involve determining the minors of the matrix. Tests for negative definiteness and semidefiniteness may be found by noting that  $Q$  is negative (semi)definite if, and only if,  $-Q$  is positive (semi)definite.

If  $Q$  is a symmetric matrix, its singular values are the magnitudes of its eigenvalues. If  $Q$  is a symmetric positive semidefinite matrix, its singular values and its eigenvalues are the same. If  $Q$  is positive semidefinite then, for any vector  $x$  one has the useful inequality

$$\sigma_{\min}(Q)\|x\|^2 \leq x^T Q x \leq \sigma_{\max}(Q)\|x\|^2. \quad (2.2.14)$$

### 2.2.2 Continuity and Function Norms

Given a subset  $\mathcal{S} \subset \Re^n$ , a function  $f(x) : \mathcal{S} \rightarrow \Re^m$  is *continuous on  $\mathcal{S}$*  if for every  $x_0 \in \mathcal{S}$  and for every  $\epsilon > 0$  there exists a  $\delta(\epsilon, x_0) > 0$  such that  $\|x - x_0\| < \delta(\epsilon, x_0)$  implies that  $\|f(x) - f(x_0)\| < \epsilon$ .

If  $\delta$  is independent of  $x_0$  then the function is said to be uniformly continuous. Uniform continuity is often difficult to test. However, if  $f(x)$  is continuous and its derivative  $f'(x)$  is bounded, then it is uniformly continuous.

A function  $f(x) : \Re^n \rightarrow \Re^m$  is differentiable if its derivative  $f'(x)$  exists. It is continuously differentiable if its derivative exists and is continuous.  $f(x)$  is said to be locally Lipschitz if, for all  $x, z \in \mathcal{S} \subset \Re^n$  one has

$$\|f(x) - f(z)\| \leq L\|x - z\| \quad (2.2.15)$$

for some finite constant  $L(\mathcal{S})$ .  $L$  is known as a Lipschitz constant. If  $\mathcal{S} = \Re^n$  the function is globally Lipschitz.

If  $f(x)$  is globally Lipschitz then it is uniformly continuous. If it is continuously differentiable it is locally Lipschitz. If it is differentiable it is continuous. For example,  $f(x) = x^2$  is continuously differentiable. It is locally but not globally Lipschitz. It is continuous but not uniformly continuous.

Given a function  $f(t) : [0, \infty) \rightarrow \Re^n$ , according to Barbalat's Lemma, if

$$\int_0^\infty f(t)dt \leq \infty, \quad (2.2.16)$$

and  $f(t)$  is uniformly continuous, then  $f(t) \rightarrow 0$  as  $t \rightarrow \infty$ .

**Function Norms.** Given a continuous-time function  $f(t) : [0, \infty) \rightarrow \Re^n$ , its  $L_p$  (function) norm is given in terms of the vector norm  $\|f(t)\|_p$  at each value of  $t$  by

$$\|f(\cdot)\|_p = \left( \int_0^\infty \|f(t)\|_p^p dt \right)^{1/p}, \quad (2.2.17)$$

and if  $p = \infty$

$$\|f(\cdot)\|_\infty = \sup_t \|f(t)\|_\infty. \quad (2.2.18)$$

If the  $L_p$  norm is finite we say  $f(t) \in L_p$ . Note that a function is in  $L_\infty$  if, and only if, it is bounded.

In the discrete-time case, let  $Z_+ = \{0, 1, 2, \dots\}$  be the set of natural numbers and  $f(k) : Z_+ \rightarrow \Re^n$ . The  $l_p$  (function) norm is given in terms of the vector norm  $\|f(k)\|_p$  at each value of  $k$  by

$$\|f(\cdot)\|_p = \left( \sum_{k=0}^{\infty} \|f(k)\|_p^p \right)^{1/p}, \quad (2.2.19)$$

and if  $p = \infty$

$$\|f(\cdot)\|_{\infty} = \sup_k \|f(k)\|_{\infty}. \quad (2.2.20)$$

If the  $l_p$  norm is finite we say  $f(k) \in l_p$ . Note that a function is in  $l_{\infty}$  if, and only if, it is bounded.

## 2.3 PROPERTIES OF DYNAMICAL SYSTEMS

In this section are discussed some properties of dynamical systems, including stability, passivity, observability, and controllability. On the one hand, if the original open-loop system is controllable and observable, then feedback control systems can be designed to afford desired performance. If the system has certain passivity properties, this design procedure is simplified and additional closed-loop properties such as robustness can be guaranteed. On the other hand, properties such as stability may not be present in the original open-loop system, but are design requirements for the closed-loop performance.

### 2.3.1 Stability

Stability, along with robustness (next subsection), is a performance requirement for closed-loop systems. That is, though the open-loop stability properties of the original system may not be satisfactory, it is desired to design a feedback control system such that the closed-loop stability is adequate. We discuss stability for continuous-time systems, but the same definitions also hold for discrete-time systems with obvious modifications.

Consider the dynamical system

$$\dot{x} = f(x, t), \quad (2.3.1)$$

with  $x \in \Re^n$ , which might represent either an uncontrolled open-loop system, or a closed-loop system after the control input  $u(t)$  has been specified in terms of the state  $x(t)$ . Let the initial time be  $t_0$ , and the initial condition be  $x_0 \equiv x(t_0)$ . This system is said to be non-autonomous since the time  $t$  appears explicitly. If  $t$  does not appear explicitly in  $f(\cdot)$ , the system is said to be autonomous. A primary cause of explicit time dependence in control systems is the presence of time-dependent disturbances  $d(t)$ .

A state  $x_e$  is an equilibrium point of the system if  $f(x_e, t) = 0, t \geq t_0$ . If  $x_0 = x_e$ , so that the system starts out in the equilibrium state, then it will forever remain there. For linear systems, the only possible equilibrium point is  $x_e = 0$ ; for nonlinear systems,  $x_e$  may be nonzero. In fact, there may be an equilibrium set, such as a limit cycle.

**Asymptotic Stability.** An equilibrium point  $x_e$  is locally asymptotically stable (AS) at  $t_0$  if there exists a compact set  $\mathcal{S} \subset \mathbb{R}^n$  such that, for every initial condition  $x_0 \in \mathcal{S}$ , one has  $\|x(t) - x_e\| \rightarrow 0$  as  $t \rightarrow \infty$ . That is, the state  $x(t)$  converges to  $x_e$ . If  $\mathcal{S} = \mathbb{R}^n$  so that  $x(t) \rightarrow x_e$  for all  $x(t_0)$ , then  $x_e$  is said to globally asymptotically stable (GAS) at  $t_0$ . If the conditions hold for all  $t_0$ , the stability is said to uniform (e.g. UAS, GUAS).

Asymptotic stability is a very strong property that is generally extremely difficult to achieve in closed-loop systems, even using advanced feedback controller design techniques. The primary reason is the presence of unknown but bounded system disturbances. A milder requirement is provided as follows.

**Lyapunov Stability.** An equilibrium point  $x_e$  is stable in the sense of Lyapunov (SISL) at  $t_0$  if for every  $\epsilon > 0$  there exists a  $\delta(\epsilon, t_0) > 0$  such that  $\|x_0 - x_e\| < \delta(\epsilon, t_0)$  implies that  $\|x(t) - x_e\| < \epsilon$  for  $t \geq t_0$ . The stability is said to be uniform (e.g. uniformly SISL) if  $\delta(\cdot)$  is independent of  $t_0$ ; that is, the system is SISL for all  $t_0$ .

It is extremely interesting to compare these definitions to those of function continuity and uniform continuity. SISL is a notion of continuity for dynamical systems. Note that for SISL there is a requirement that the state  $x(t)$  be kept arbitrarily close to  $x_e$  by starting sufficiently close to it. This is still too strong a requirement for closed-loop control in the presence of unknown disturbances. Therefore, a practical definition of stability to be used as a performance objective for feedback controller design in this book is as follows.

**Boundedness.** This definition is illustrated in Fig. 2.3.1. The equilibrium point  $x_e$  is said to be uniformly ultimately bounded (UUB) if there exists a compact set  $\mathcal{S} \subset \mathbb{R}^n$  so that for all  $x_0 \in \mathcal{S}$  there exists a bound  $B$  and a time  $T(B, x_0)$  such that  $\|x(t) - x_e\| \leq B$  for all  $t \geq t_0 + T$ . The intent here is to capture the notion that for all initial states in the compact set  $\mathcal{S}$ , the system trajectory eventually reaches, after a lapsed time of  $T$ , a bounded neighborhood of  $x_e$ .

The difference between UUB and SISL is that in UUB the bound  $B$  cannot be made arbitrarily small by starting closer to  $x_e$ . In fact, the van der Pol oscillator in Example 2.1.2 is UUB but not SISL. In practical closed-loop systems,  $B$  depends on the disturbance magnitudes and other factors. If the controller is suitably designed, however,  $B$  will be small enough for practical purposes. The term uniform indicates that  $T$  does not depend on  $t_0$ . The term ultimate indicates that the boundedness property holds after a time lapse  $T$ . If  $\mathcal{S} = \mathbb{R}^n$  the system is said to be globally UUB (GUUB).

**A Note on Autonomous Systems and Linear Systems.** If the system is autonomous so that

$$\dot{x} = f(x) \quad (2.3.2)$$

where  $f(x)$  is not an explicit function of time, then the state trajectory is independent of the initial time. This means that if an equilibrium point is stable by any of the three definitions, the stability is automatically uniform. Non-uniformity is only a problem with non-autonomous systems.

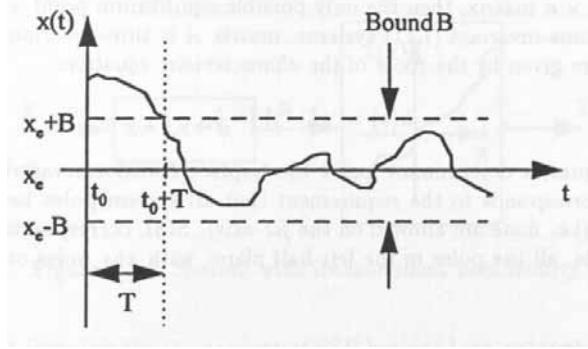


Figure 2.3.1: Illustration of uniform ultimate boundedness (UUB).

If the system is linear so that

$$\dot{x} = A(t)x \quad (2.3.3)$$

with  $A(t)$  an  $n \times n$  matrix, then the only possible equilibrium point is the origin.

For linear time-invariant (LTI) systems, matrix  $A$  is time-invariant. Then, the system poles are given by the roots of the characteristic equation

$$\Delta(s) = |sI - A| = 0, \quad (2.3.4)$$

with  $\|\cdot\|$  the matrix determinant and  $s$  the Laplace transform variable. For LTI systems, AS corresponds to the requirement that all system poles be in the open left-half plane (i.e. none are allowed on the  $j\omega$ -axis). SISL corresponds to marginal stability, that is, all the poles in the left-half plane, with any poles on the  $j\omega$ -axis nonrepeated.

### 2.3.2 Passivity

Passive systems are important in robust control where a feedback control system must be designed to offset the effects of bounded disturbances or unmodelled dynamics. Since we intend to define some new passivity properties of NN, we discuss here some notions of passivity (Goodwin and Sin 1984; Landau 1979; Lewis, Abdallah, and Dawson 1993; Slotine and Li 1991). Passivity is extensively used in the theory of networks and  $n$ -port devices.

#### 2.3.2.1 Passivity of Continuous-Time Systems

A continuous-time system (e.g. (2.1.1)) with input  $u(t)$  and output  $y(t)$  is said to be passive if it verifies an equality of the so-called power form

$$\dot{L}(t) = y^T u - g(t) \quad (2.3.5)$$

for some  $L(t)$  that is lower bounded and some  $g(t) \geq 0$ . That is (see Problems section),

$$\int_0^T y^T(\tau)u(\tau)d\tau \geq \int_0^T g(\tau)d\tau - \gamma^2 \quad (2.3.6)$$

for all  $T \geq 0$  and some  $\gamma \geq 0$ . Often,  $L(t)$  is the total energy, kinetic plus potential; then, the power input to the system is  $y^T u$  and  $g(t)$  is the dissipated power.

We say the system is dissipative if it is passive and in addition

$$\int_0^\infty y^T(\tau)u(\tau)d\tau \neq 0 \text{ implies } \int_0^\infty g(\tau)d\tau > 0. \quad (2.3.7)$$

A special sort of dissipativity occurs if  $g(t)$  is a monic quadratic function of  $\|x\|$  with bounded coefficients, where  $x(t)$  is the internal state of the system. We call this state strict passivity (SSP). Then,

$$\int_0^T y^T(\tau)u(\tau)d\tau \geq \int_0^T (\|x\|^2 + LOT) d\tau - \gamma^2 \quad (2.3.8)$$

for all  $T \geq 0$  and some  $\gamma \geq 0$ , where  $LOT$  denotes lower-order terms in  $\|x\|$ . Then, the  $L_2$  norm of the state is overbounded in terms of the  $L_2$  inner product of output and input (i.e. the power delivered to the system).

Somewhat surprisingly, the concept of SSP has not been extensively used in the literature (Lewis, Liu, and Yeşildirek 1993; Seron et al. 1994), though see Goodwin and Sin (1984) where input and output strict passivity are defined. We use SSP to advantage in subsequent chapters to conclude some internal boundedness properties of neural network controllers without the usual assumptions of observability (e.g. persistence of excitation) that are required in standard adaptive control approaches (see Chapter 4).

### Example 2.3.1 (Passivity of System with Nonlinearity) :

Many practical systems have nonlinear and/or discontinuous measurements or actuators, including backlash, deadzones, saturation limits, and so on. The time-varying system with nonlinear measurements (Slotine and Li 1991)

$$\begin{aligned} \dot{x} + \lambda(t)x &= u \\ y &= h(x) \end{aligned} \quad (2.3.9)$$

$\lambda(t) \geq 0$ , is depicted in Fig. 2.3.2. The nonlinearity  $h(x)$  satisfies the positivity condition

$$xh(x) > 0 \text{ for } x \neq 0$$

which means it has the same sign as its argument. Otherwise, it is arbitrary and can even be discontinuous.

Select the trial function

$$L = \int_0^x h(z)dz \geq 0$$

and, using Leibniz' rule, differentiate to determine

$$\dot{L} = h(x)\dot{x} = yu - h(x)\lambda(t)x \equiv yu - g(t),$$

which is in power form. Therefore, the system is passive. Since the condition (2.3.7) holds if  $\lambda(t)$  is not identically zero, the system is also dissipative.  $\square$

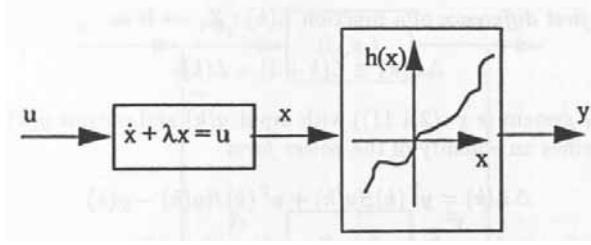


Figure 2.3.2: System with measurement nonlinearity.

### 2.3.2.2 Passivity of Discrete-Time Systems

The passivity notions defined here are used later in Lyapunov proofs of stability. Discrete-time Lyapunov proofs are considerably more complex than their continuous-time counterparts; therefore, the required passivity notions in discrete-time are more complex.

Define the first difference of a function  $L(k) : Z_+ \rightarrow \mathbb{R}$  as

$$\Delta L(k) \equiv L(k+1) - L(k). \quad (2.3.10)$$

A discrete-time system (e.g. (2.1.11)) with input  $u(k)$  and output  $y(k)$  is said to be passive if it verifies an equality of the power form

$$\Delta L(k) = y^T(k)Su(k) + u^T(k)Ru(k) - g(k) \quad (2.3.11)$$

for some  $L(k)$  that is lower bounded, some function  $g(k) \geq 0$ , and appropriately defined matrices  $R, S$ . That is (see Problems section),

$$\sum_{k=0}^T (y^T(k)Su(k) + u^T(k)Ru(k)) \geq \sum_{k=0}^T g(k) - \gamma^2 \quad (2.3.12)$$

for all  $T \geq 0$  and some  $\gamma \geq 0$ .

We say the system is dissipative if it is passive and in addition

$$\sum_{k=0}^T (y^T(k)Su(k) + u^T(k)Ru(k)) \neq 0 \text{ implies } \sum_{k=0}^T g(k) > 0, \quad (2.3.13)$$

for all  $T \geq 0$ .

A special sort of dissipativity occurs if  $g(k)$  is a monic quadratic function of  $\|x\|$  with bounded coefficients, where  $x(k)$  is the internal state of the system. We call this state strict passivity (SSP). Then,

$$\sum_{k=0}^T (y^T(k)Su(k) + u^T(k)Ru(k)) \geq \sum_{k=0}^T (\|x\|^2 + LOT) - \gamma^2 \quad (2.3.14)$$

for all  $T \geq 0$  and some  $\gamma \geq 0$ , where  $LOT$  denotes lower-order terms in  $\|x\|$ . Then, the  $l_2$  norm of the state is overbounded in terms of the  $l_2$  inner product of

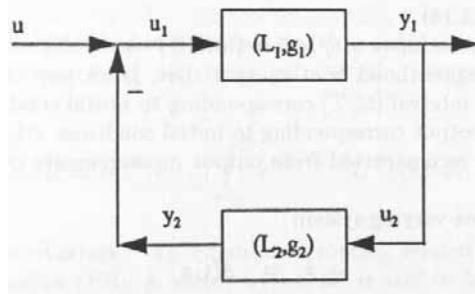


Figure 2.3.3: Two passive systems in feedback interconnection.

*output and input (i.e. the power delivered to the system). We use SSP to conclude some internal boundedness properties of the system without the usual assumptions of observability (e.g. persistence of excitation) that are required in standard adaptive control approaches.*

#### 2.3.2.3 Interconnections of Passive Systems

*To get an indication of the importance of passivity, suppose two passive systems are placed into a feedback configuration as shown in Fig. 2.3.3. Then,*

$$\begin{aligned}\dot{L}_1(t) &= y_1^T u_1 - g_1(t) \\ \dot{L}_2(t) &= y_2^T u_2 - g_2(t) \\ u_1 &= u - y_2 \\ u_2 &= y_1\end{aligned}\tag{2.3.15}$$

*and it is very easy to verify (see Problems section) that*

$$\frac{d}{dt}(L_1 + L_2) = y_1^T u - (g_1 + g_2).\tag{2.3.16}$$

*That is, the feedback configuration is also in power form and hence passive. Properties that are preserved under feedback are extremely important for controller design.*

*If both systems in Fig. 2.3.3 are state strict passive, then the closed-loop system is SSP. However, if only one subsystem is SSP and the other only passive, the combination is only passive and not generally SSP (see Problems section).*

*It also turns out that parallel combinations of systems in power form are still in power form. These results are particular cases of what is known in circuit theory as Tellegen's power conservation theorem (Slotine and Li 1991). Series interconnection does not generally preserve passivity (see Problems section).*

#### 2.3.3 Observability and Controllability

*Observability and controllability are properties of the open-loop system— when they hold, it is possible to design feedback controllers to fulfill desired closed-loop performance specifications (e.g. track a reference trajectory and keep all internal states stable).*

The discussion in this subsection centers around the nonlinear continuous-time system

$$\dot{x} = f(x) + g(x)u \quad (2.3.17)$$

$$y = h(x), \quad (2.3.18)$$

which is said to be affine in the control input  $u(t)$ , and the linear time-invariant (LTI) system

$$\dot{x} = Ax + Bu \quad (2.3.19)$$

$$y = Cx, \quad (2.3.20)$$

which is denoted  $(A, B, C)$ . Let  $x \in \mathbb{R}^n, u \in \mathbb{R}^m, y \in \mathbb{R}^p$ . The definitions extend in a straightforward manner to discrete-time systems.

### 2.3.3.1 Observability

Observability properties refer to the suitability of the measurements taken in a system; that is, the suitability of the choice of the measurement function  $h(\cdot)$  in the output equation (2.3.18).

A system with zero input  $u(t) = 0$  is (locally) observable at an initial state  $x_0$  if there exists a neighborhood  $\mathcal{S}$  of  $x_0$  such that, given any other state  $x_1 \in \mathcal{S}$ , the output over an interval  $[t_0, T]$  corresponding to initial condition  $x(t_0) = x_0$  is different from the output corresponding to initial condition  $x(t_0) = x_1$ . Then, the initial state can be reconstructed from output measurements over a time interval  $[t_0, T]$ .

Consider the time-varying system

$$\dot{x} = A(t)x \quad (2.3.21)$$

$$y = C(t)x \quad (2.3.22)$$

and define the state-transition matrix  $\Phi(t, t_0) \in \mathbb{R}^{n \times n}$  by

$$\frac{d}{dt}\Phi(t, t_0) = A(t)\Phi(t, t_0), \quad \Phi(t_0, t_0) = I. \quad (2.3.23)$$

The key object in observability analysis is the observability gramian given by

$$N(t_0, T) = \int_{t_0}^T \Phi^T(\tau, t_0)C^TC\Phi(\tau, t_0) d\tau. \quad (2.3.24)$$

The system is said to be uniformly completely observable (UCO) (Sastry and Bodson 1989) if there exist positive constants  $\delta, \alpha_1, \alpha_2$  such that

$$\alpha_1 I \leq N(t_0, t_0 + \delta) \leq \alpha_2 I \quad (2.3.25)$$

for all  $t_0 \geq 0$ .

In the linear time-invariant case  $(A, B, C)$ , observability tests are straightforward (Kailath 1980). The state transition matrix for linear systems is

$$\Phi(t, t_0) = e^{A(t-t_0)} \quad (2.3.26)$$

and the observability gramian is

$$N(t_0, T) = \int_{t_0}^T e^{A^T(\tau-t_0)} C^T C e^{A(\tau-t_0)} d\tau. \quad (2.3.27)$$

The system is observable if, and only if,  $N(t_0, T)$  has full rank  $n$ . This observability condition can be shown equivalent to the requirement that the observability matrix

$$V = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^n \end{bmatrix} \quad (2.3.28)$$

have full rank  $n$ . Note that matrix  $B$  does not enter into these requirements. Matrix  $V$  is of full rank  $n$  if, and only if, the discrete-time observability gramian

$$G_o = V^T V \quad (2.3.29)$$

is nonsingular.

If the system is observable and the input  $u(t)$  is zero, the initial state can be reconstructed from the output  $y(t)$  measured over the interval  $[t_0, T]$  using the functional operator

$$x(t_0) = N^{-1}(t_0, T) \int_{t_0}^T e^{A^T(\tau-t_0)} C^T y(\tau) d\tau. \quad (2.3.30)$$

**Persistence of Excitation.** An important property related to observability is persistence of excitation (PE). A vector  $w(t) \in \Re^p$  is said to be PE if there exist positive constants  $\delta, \alpha_1, \alpha_2$  such that

$$\alpha_1 I \leq \int_{t_0}^{t_0+\delta} w(\tau) w^T(\tau) d\tau \leq \alpha_2 I \quad (2.3.31)$$

for all  $t_0 \geq 0$ . The integral may be interpreted as a gramian and PE could be compared to the definition of uniform complete observability. PE is a notion of a time signal's containing 'sufficient richness' so that the matrix defined by the  $L_2$  outer product in the definition is nonsingular. Note that the  $p \times p$  vector outer product matrix  $w(t)w^T(t)$  has rank of only one for any given  $t$ . However, when integrated over the interval  $[t_0, t_0 + \delta]$  the requirement is that the resulting matrix be nonsingular.

Roughly speaking, if  $w(t)$  is a  $p$ -vector, it should have at least  $p$  distinct complex frequencies to be PE. For example, if  $p = 4$ ,  $w(t)$  could be the sum of four real exponentials, or contain sinusoidal components at two frequencies, etc.

### 2.3.3.2 Controllability

Controllability properties refer to the suitability of the control inputs selected for a system; that is, the suitability of the choice of the input function  $g(\cdot)$  in the state equation (2.3.17).

A system is (locally) controllable at an equilibrium state  $x_e$  if there exists a neighborhood  $\mathcal{S}$  of  $x_e$  such that, given any initial state  $x(t_0) \in \mathcal{S}$ , there exists a final time  $T$  and a control input  $u(t)$  on  $[0, T]$  that drives the state from  $x(t_0)$  to  $x_e$ . A system is (locally) reachable at a given initial state  $x(t_0)$  if there exists a neighborhood  $\mathcal{S}$  of  $x(t_0)$  such that, given any prescribed final state  $x_d(T) \in \mathcal{S}$ , there exists a final time  $T$  and a control input  $u(t)$  on  $[0, T]$  that drives the state from  $x(t_0)$  to  $x_d(T)$ . (Vidyasagar 1993).

In the linear time-invariant case  $(A, B, C)$  one may give tests for controllability and reachability that are easy to perform (Kailath 1980). Then, local and global controllability properties are the same. For continuous LTI systems, reachability and controllability are the same and the key object in analysis is the controllability gramian

$$M(t_0, T) = \int_{t_0}^T e^{A(T-\tau)} B B^T e^{A^T(T-\tau)} d\tau. \quad (2.3.32)$$

The system is controllable (equivalently reachable) if  $M(t_0, T)$  has full rank. It can be shown that if  $M(t_0, T)$  has full rank for any  $T > t_0$ , then it has full rank for all  $T > t_0$ . This controllability condition can be shown to reduce to a full-rank condition on the controllability matrix

$$U = [B \ AB \ A^2B \dots A^nB]. \quad (2.3.33)$$

Note that matrix  $C$  does not enter into these requirements. Matrix  $U$  is of full rank  $n$  if, and only if, the discrete-time controllability gramian

$$G_c = UU^T \quad (2.3.34)$$

is nonsingular. If the system is controllable, the initial state  $x(t_0)$  can be driven to any desired final state  $x_d(T)$  using the control input computed according to (see Problems section)

$$u(t) = B^T e^{A^T(T-t)} M^{-1}(t_0, T) [x_d(T) - e^{AT} x(t_0)], \quad t \in [t_0, T]. \quad (2.3.35)$$

In the case of discrete LTI systems a similar analysis holds, but then reachability is stronger than controllability. That is, for discrete systems it is easier to drive nonzero initial states to zero than it is to drive them to prescribed nonzero final values.

## 2.4 FEEDBACK LINEARIZATION AND CONTROL SYSTEM DESIGN

For linear time-invariant (LTI) systems there are a wide variety of controller design techniques that achieve a range of performance objectives including state regulation, tracking of desired trajectories, and so on. Design techniques include the linear quadratic regulator,  $H$ -infinity and other robust control techniques, adaptive control, classical approaches such as root-locus and Bode design, and so on. Generally, as long as the system is controllable it is possible to design a controller using full state-feedback that gives good closed-loop performance. Some problems occur with non-minimum phase systems, but several techniques are now available for confronting

these. If only the outputs can be measured, then good performance can be achieved by using a dynamic regulator as long as the system is both controllable and observable (e.g. linear quadratic gaussian design using a Kalman filter).

Unfortunately, for nonlinear systems controls design is very much more complex. There are no universal techniques that apply for all nonlinear systems; each nonlinear system must generally be considered as a separate design problem. Though there are techniques available such as Lyapunov, passivity, hyperstability, and variable-structure (e.g. sliding mode) approaches, considerable design insight is still required. Feedback linearization techniques offer a widely applicable set of design tools that are useful for broad classes of nonlinear systems. They function by basically converting the nonlinear problem into a related linear controls design problem. They are more powerful, where they apply, than standard classical linearization techniques such as Lyapunov's indirect method where the nonlinear system is linearized using Jacobian techniques. References for this section include (Slotine and Li 1991, Isidori 1989, Khalil 1992, Vidyasagar 1993).

### 2.4.1 Input-Output Feedback Linearization Controllers

There are basically two feedback linearization techniques— input-state feedback linearization and input-output (i/o) feedback linearization. The former requires a complex set of mathematical tools including Frobenius' Theorem and Lie algebra. The control laws derived are often complex due to the need to determine nonlinear state-space transformations and their inverses. On the other hand, i/o feedback linearization is direct to apply and represents more of an engineering approach to control systems design. It is very useful for large classes of nonlinear controls problems including those treated in this book, which encompass robot manipulators, mechanical systems, and other Lagrangian systems.

#### 2.4.1.1 Feedback Linearization Controller Design

Here, we discuss i/o feedback linearization as a controller design technique for systems of the form

$$\begin{aligned}\dot{x} &= F(x, u) \\ y &= h(x).\end{aligned}\tag{2.4.1}$$

The technique is introduced through a sample design.

**Sample Plant and Problem Specification.** Given the system or plant dynamics

$$\begin{aligned}\dot{x}_1 &= x_1 x_2 + x_3 \\ \dot{x}_2 &= -2x_2 + x_1 u \\ \dot{x}_3 &= \sin x_1 + 2x_1 x_2 + u\end{aligned}\tag{2.4.2}$$

it is desired to design a tracking controller that causes  $x_1(t)$  to follow a desired trajectory  $y_d(t)$  which is prescribed by the user. This is a complex design problem that is not approachable using any of the LTI techniques mentioned above.

**I/O Feedback Linearization Step.** The tracking problem is easily approached using i/o feedback linearization. The procedure is as follows. Select the output

$$y(t) = x_1(t). \quad (2.4.3)$$

Note that the output is defined by the performance specifications. Differentiate  $y(t)$  repeatedly and substitute state derivatives from (2.4.2) until the control input  $u(t)$  appears. This step yields

$$\begin{aligned} \dot{y} &= \dot{x}_1 = x_1 x_2 + x_3 \\ \ddot{y} &= x_1 \dot{x}_2 + \dot{x}_1 x_2 + \dot{x}_3 \\ &= [\sin x_1 + x_2 x_3 + x_1 x_2^2] + [1 + x_1^2] u \\ &\equiv f(x) + g(x)u. \end{aligned} \quad (2.4.4)$$

Now define variables as  $z_1 \equiv y, z_2 \equiv \dot{y}$  so that

$$\begin{aligned} \dot{z}_1 &= z_2 \\ \dot{z}_2 &= f(x) + g(x)u. \end{aligned} \quad (2.4.5)$$

This may be converted to a linear system by redefinition of the input as

$$v(t) \equiv f(x) + g(x)u(t) \quad (2.4.6)$$

so that

$$u(t) = \frac{1}{g(x)}(-f(x) + v(t)), \quad (2.4.7)$$

for then one obtains

$$\begin{aligned} \dot{z}_1 &= z_2 \\ \dot{z}_2 &= v, \end{aligned} \quad (2.4.8)$$

which is equivalent to

$$\ddot{y} = v. \quad (2.4.9)$$

This is known as the feedback linearized system.

**Controller Design Step.** Standard linear system techniques can now be used to design a tracking controller for the feedback linearized system. For instance, one possibility is the proportional-plus-derivative (PD) tracking control

$$v = \ddot{y}_d + K_d \dot{e} + K_p e \quad (2.4.10)$$

where the tracking error is defined as

$$e(t) \equiv y_d(t) - y(t). \quad (2.4.11)$$

Substituting this control  $v(t)$  into (2.4.9) yields the closed-loop system

$$\ddot{e} + K_d \dot{e} + K_p e = 0, \quad (2.4.12)$$

or equivalently, in state-space form

$$\frac{d}{dt} \begin{bmatrix} e \\ \dot{e} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -K_p & -K_d \end{bmatrix} \begin{bmatrix} e \\ \dot{e} \end{bmatrix}. \quad (2.4.13)$$

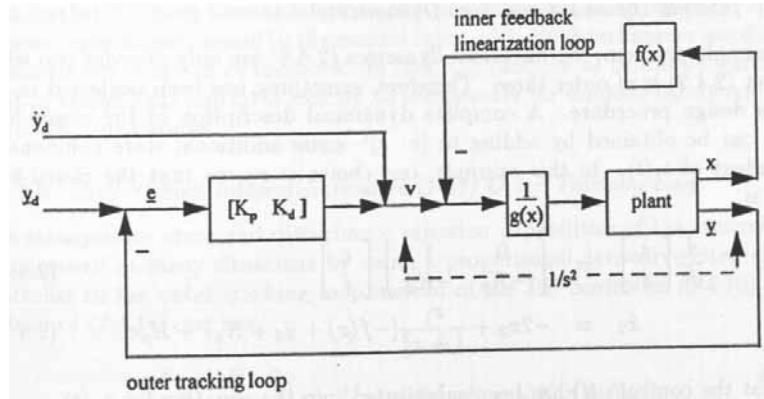


Figure 2.4.1: Feedback linearization controller showing PD outer loop and nonlinear inner loop.

*As long as the PD gains are positive, the tracking error converges to zero. The PD gains should be selected for suitable percent overshoot and rise time.*

*According to (2.4.7) and (2.4.10) the complete controller implied by this technique is given by*

$$u(t) = \frac{1}{g(x)}[-f(x) + \ddot{y}_d + K_d\dot{e} + K_p e], \quad (2.4.14)$$

*where the nonlinear functions  $f(x), g(x)$  are defined in (2.4.4).*

#### 2.4.1.2 Structure of I/O Feedback Linearization Controller

*The structure of the i/o feedback linearization controller is depicted in Fig. 2.4.1, where  $e \equiv [e \ \dot{e}]^T$ ,  $\underline{y} \equiv [y \ \dot{y}]^T$ ,  $\underline{y}_d \equiv [y_d \ \dot{y}_d]^T$ . It consists of a PD outer tracking loop plus a nonlinear inner linearization loop. The function of the inner feedback loop is to linearize the plant so that the system, between the points shown, looks like  $1/s^n$  (in this example,  $n=2$ ). Then the PD controller, the design of which is based on the system  $1/s^n$ , achieves tracking behavior. Note that the controller incorporates feedforward acceleration compensation through the term  $\ddot{y}_d$ ; this is a form of predictive control.*

*A major advantage of the feedback linearization controller is that it contains a unity-gain outer tracking loop, which provides robustness and is highly desirable in many practical applications (e.g. aircraft control system design). This design technique also decouples the nonlinear compensation design step from the tracking performance specification design step. Note that the feedback linearization controller generally requires full state feedback in computing  $g(x), f(x)$ .*

*It is to be emphasized that the feedback linearization controller generally has performance far exceeding that of classical linearization controllers based on Jacobian linearization techniques. No approximation is involved in feedback linearization design.*

#### 2.4.1.3 Ill-Defined Relative Degree

For this technique to work, the function  $g(x)$  multiplying the control input  $u(t)$  in (2.4.5) must never be zero (see (2.4.7)). In this example  $g(x) = 1 + x_1^2$ . If  $g(x)$  can be zero in a particular plant, the plant is said to have ill-defined relative degree. Otherwise, it has well-defined relative degree.

Even if the system is ill defined, i/o feedback linearization may still be applied under some circumstances. In fact, it can be shown that as long as the closed-loop system

$$\begin{aligned}\dot{z}_1 &= z_2 \\ \dot{z}_2 &= f(x) + g(x)u \\ \zeta &= g(x).\end{aligned}\tag{2.4.15}$$

with output  $\zeta(t)$  is observable, then a modification of (2.4.7) still works (Commuri and Lewis 1994). The observability requirement means that the control influence coefficient  $g(x)$  may only be small, so that control effectiveness is reduced, when the state  $x(t)$  is also small.

#### 2.4.1.4 Internal Dynamics and Zero Dynamics

In the sample problem, the linearized dynamics (2.4.8) are only of order two while the plant (2.4.2) is of order three. Therefore, something has been neglected in the controls design procedure. A complete dynamical description of the closed-loop system can be obtained by adding to  $[e \ \dot{e}]^T$  some additional state components independent of  $z_i(t)$ . In this example, one choice is  $x_2$ , so that the closed-loop system is

$$\frac{d}{dt} \begin{bmatrix} e \\ \dot{e} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -K_p & -K_d \end{bmatrix} \begin{bmatrix} e \\ \dot{e} \end{bmatrix} \tag{2.4.16}$$

$$\dot{x}_2 = -2x_2 + \frac{x_1}{1+x_1^2}[-f(x) + \ddot{y}_d + K_d\dot{e} + K_p e]. \tag{2.4.17}$$

Note that the control  $u(t)$  has been substituted into the equation for  $\dot{x}_2(t)$ .

The additional dynamics neglected in the feedback linearization design are made unobservable by this design procedure; that is, with respect to the design output  $y(t) = x_1(t)$  they are unobservable. They are known as the internal dynamics. A different choice of  $y(t)$  results in different internal dynamics (see Problems section). The zero dynamics is defined as the internal dynamics when the control input is selected to keep the output  $y(t)$  equal to zero. If the zero dynamics are unstable, the system is said to be non-minimum phase. The zero dynamics extends to nonlinear systems the concept of system zeros. In the LTI case, the zero dynamics define the system zeros. For i/o feedback linearization to function correctly, the internal dynamics must be stable. Then, the controller given in Fig. 2.4.1 performs in an adequate manner.

In this example the zero dynamics are

$$\dot{x}_2 = -2x_2, \tag{2.4.18}$$

which says that  $x_2(t) = e^{-2t}x_2(0)$ ; this is an asymptotically stable (AS) system. Therefore, the i/o feedback linearization controller performs correctly. Specifically,

the internal dynamics (2.4.17) represent an AS system driven by additional signals. If the PD feedback linearization controller operates correctly, these additional signals are all bounded. The state of an AS system with bounded input is also bounded, so all is well. One can see that, as the desired acceleration  $\ddot{y}_d(t)$  increases, the magnitude of state  $x_2(t)$  will increase, so that internal dynamics can fundamentally limit the performance capabilities of nonlinear systems. If the pole at  $s = 2$  were further to the left in the  $s$ -plane, the situation would be improved and faster prescribed trajectories could be followed.

#### 2.4.1.5 Modelling Errors, Disturbances, and Robustness

In the sample design it was assumed that all the dynamics are exactly known and there are no disturbances. However, in practical situations there can be unmodelled dynamics or unknown disturbances. Such effects degrade the performance of the feedback linearization controller and are investigated in the Problems section. However, the controller is surprisingly robust to such effects. This is in great measure due to the outer PD tracking loop. The robustness properties can be improved even further by using various adaptive control techniques, or by adding a specially designed robustifying signal to the control input  $u(t)$ . Such techniques are discussed in subsequent chapters of the book. In fact, the function of neural networks when used in closed-loop control is exactly to compensate for unmodelled dynamics and unknown disturbances.

#### 2.4.1.6 Proportional-Integral-Derivative (PID) Outer Tracking Loop

The steady-state error and disturbance rejection capabilities of the controller can be improved in many situations by using a proportional-derivative-integral (PID) controller in the outer tracking loop instead of the PD controller (2.4.10). Then, instead of (2.4.14) one has

$$\begin{aligned}\dot{\varepsilon} &= e \\ u(t) &= \frac{1}{g(x)}[-f(x) + \ddot{y}_d + K_d \dot{e} + K_p e + K_i \varepsilon].\end{aligned}\tag{2.4.19}$$

This is an important example of a controller with its own dynamics—the controller now has a state  $\varepsilon(t)$  associated with it, so that it has some internal memory.

#### 2.4.1.7 Feedback Linearization for Systems in Brunovsky Form

If the system is in Brunovsky form, i/o feedback linearization is very easy. For the system

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= x_3 \\ &\vdots \\ \dot{x}_n &= f(x) + g(x)u\end{aligned}\tag{2.4.20}$$

$$y = x_1\tag{2.4.21}$$

the design is direct, for the linearization step is not needed. In fact, the system is already in the chained form (2.4.5). Therefore, the input redefinition

$$u(t) = \frac{1}{g(x)}(-f(x) + v) \quad (2.4.22)$$

is the feedback linearization controller, for then the closed-loop system is simply

$$y^{(n)} = v, \quad (2.4.23)$$

where superscript  $(j)$  denotes the  $j$ -th derivative.

The outer-loop control  $v(t)$  may be selected using an extension of PD or PID control to obtain a control structure like that in Fig. 2.4.1. One possibility is

$$u(t) = \frac{1}{g(x)}[-f(x) + y_d^{(n)} + K_n e^{(n-1)} + \dots + K_1 e], \quad (2.4.24)$$

with  $y_d(t)$  the desired trajectory and  $e(t) = y_d(t) - y(t)$  the tracking error. Note that feedback of the error and its first  $n-1$  derivatives is needed, along with feedforward of  $y_d^{(n)}(t)$ . The closed-loop dynamics is

$$e^{(n)} + K_n e^{(n-1)} + \dots + K_1 e = 0, \quad (2.4.25)$$

which is stable if the feedback gains  $[K_n \dots K_1]$  are suitably selected. Here, there are no internal dynamics.

One can write this in matrix form by defining the error vector

$$\underline{e} \equiv \begin{bmatrix} e \\ \dot{e} \\ \vdots \\ e^{(n-1)} \end{bmatrix} \quad (2.4.26)$$

and the gain vector

$$K \equiv [K_1 \ K_2 \dots K_n]. \quad (2.4.27)$$

Then the closed-loop error dynamics may be written as

$$\begin{aligned} \dot{\underline{e}} &= (A - bK)\underline{e} \\ \dot{\xi} &= \Phi(\underline{e}, \xi, y_d) \end{aligned} \quad (2.4.28)$$

where  $A, b$  are the canonical form matrices (2.1.6). An equation has been added for the internal dynamics  $\xi(t)$ , which may be present in some examples.

A similar procedure may be used to design feedback linearization for multivariable Brunovsky canonical form systems (2.1.7).

## 2.4.2 Computer Simulation of Feedback Control Systems

In Example 2.1.2 it was shown how to simulate an open-loop nonlinear system using the state-space description and a Runge-Kutta integrator. A conscientious engineering procedure for design of feedback control systems involves analysis of the

*open-loop dynamics to determine the system properties, then controller design, then computer simulation of the closed-loop system prior to implementing the controller on the actual plant. The simulation step is essential to verify closed-loop performance and ensure that nothing has been overlooked in the design step. Continuous-time dynamical systems with feedback controllers are straightforward to simulate. A Runge-Kutta integrator is required, such as ODE23 in MATLAB. Then, a subroutine must be written that has two parts: the computation of the control input  $u(t)$  followed by the plant dynamics. The technique is illustrated using the sample design problem.*

**Example 2.4.1 (Simulation of Feedback Linearization Controller) :**

For the sample design problem (2.4.2), the complete closed-loop description is given by the plant dynamics

$$\begin{aligned}\dot{x}_1 &= x_1x_2 + x_3 \\ \dot{x}_2 &= -2x_2 + x_1u \\ \dot{x}_3 &= \sin x_1 + 2x_1x_2 + u\end{aligned}$$

and the controller

$$\begin{aligned}f(x) &= \sin x_1 + x_2x_3 + x_1x_2^2 \\ g(x) &= 1 + x_1^2 \\ y &= x_1 \\ e &= y_d - y \\ u(t) &= \frac{1}{g(x)}[-f(x) + \ddot{y}_d + K_d\dot{e} + K_p e],\end{aligned}$$

where the desired trajectory  $y_d(t)$  is prescribed by the user.

All this information must be written into a MATLAB M file that is called by ODE23. Suppose that it is desired for the plant output  $y(t)$  to follow the desired trajectory

$$y_d = \sin(2\pi t/T).$$

Then, the MATLAB M file for ODE23 is:

```
% MATLAB file for closed-loop system simulation
function xdot= fblinct(t,x)
global T
% Computation of the desired trajectory
yD= sin(2*pi*t/T) ;
yDdot= (2*pi/T) * cos(2*pi*t/T) ;
yDddot= -(2*pi/T)^2 * sin(2*pi*t/T) ;
% Computation of the control input
kp= 100 ;
kd= 14.14 ;
f = sin(x(1)) + x(2)*x(3) + x(1)*x(2)^2 ;
g = 1 + x(1)^2 ;
y = x(1) ;
ydot= x(1)*x(2) + x(3) ;
e = yD - y ;
edot= yDdot - ydot ;
u = (-f + yDddot + kd*edot + kp*e) / g ;
% Plant dynamics
xdot(1) = x(1)*x(2) + x(3) ;
xdot(2) = -2*x(2) + x(1)*u ;
xdot(3) = sin(x(1)) + 2*x(1)*x(2) + u ;
```

Note how closely the structure of the M file follows the controller and plant equations.

The selection of the PD gains as  $K_p = 100, K_d = 14.14$  yields a closed-loop characteristic polynomial of  $s^2 + K_d s + K_p = s^2 + 14.14s + 100 \equiv s^2 + 2\zeta\omega_n s + \omega_n^2$ , so that the closed-loop system has a natural frequency of  $\omega_n = 10$  rad/s and a damping ratio of  $\zeta = 1/\sqrt{2} = 0.707$ .

The script session needed to run the simulation and make the plots is:

```
T= 10
t0= 0 ; tf=50 ;
x0= [1 1 1]' ;
[t,x]= ode23('fblinct',t0,tf,x0) ;
yd= sin(2*pi*t/T) ;
e= yd - y ;
plot(t,[yd,x(:,1)])
plot(t,e)
plot(t,x(:,2))
```

where the graph labeling commands are not shown.

The time constant of the internal dynamics is  $\tau = 0.5\text{sec}$ . The simulation results for a desired period of  $T = 10$  sec are plotted in Fig. 2.4.2. The tracking behavior of  $x_1(t)$  and internal stability of  $x_2(t)$  are excellent. The performance for  $T = 1$  sec is shown in Fig. 2.4.3. This value of  $T$  is of the order of the time constant of the internal dynamics, so problems may be anticipated. Indeed, note that  $x_2(t)$  is larger since the desired acceleration has increased (recall that the internal dynamics are excited by the desired acceleration). This makes the tracking error larger.  $\square$

### 2.4.3 Feedback Linearization for Discrete-Time Systems

*This discussion involves i/o feedback linearization controller design of discrete-time systems in the general form*

$$\begin{aligned} x(k+1) &= F(x(k), u(k)) \\ y(k) &= h(x(k)). \end{aligned} \quad (2.4.29)$$

*The procedure is very much the same as for continuous-time systems.*

#### 2.4.3.1 I/O Linearization Step

*One may proceed by several approaches to obtain the discrete-time chained Brunovsky form*

$$\begin{aligned} z_1(k+1) &= z_2(k) \\ z_2(k+1) &= z_3(k) \\ &\vdots \\ z_n(k+1) &= f(x(k)) + g(x(k))u(k) \end{aligned} \quad (2.4.30)$$

$$y(k) = z_1(k). \quad (2.4.31)$$

*Specifically, one could advance the prescribed output  $y(k) = h(x(k))$  repeatedly, defining new states as  $z_1(k) \equiv y(k), z_2(k) \equiv y(k+1), z_3(k) \equiv y(k+2), \dots$ , until  $u(k)$  appears. The result is the chained form, possibly with some additional internal dynamics.*

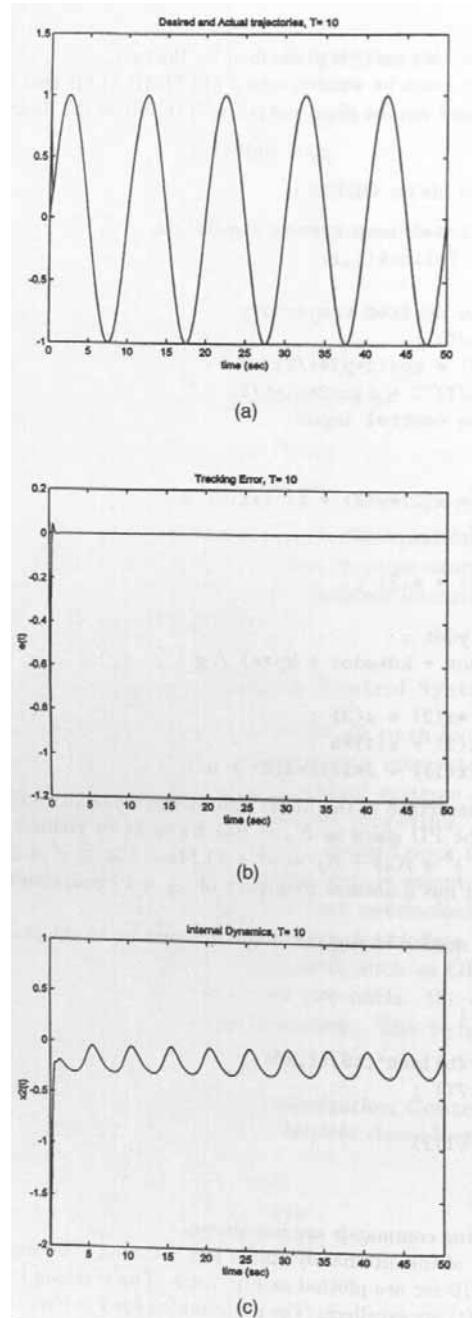


Figure 2.4.2: Simulation of feedback linearization controller,  $T = 10$  sec. (a) Actual output  $y(t)$  and desired output  $y_d(t)$ . (b) Tracking error  $e(t)$ . (c) Internal dynamics state  $x_2(t)$ .

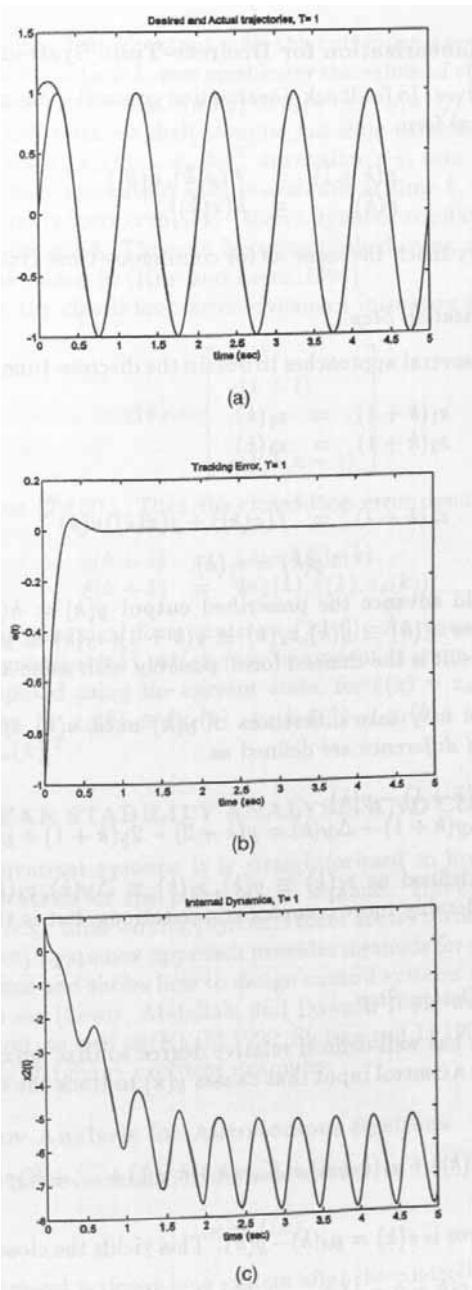


Figure 2.4.3: Simulation of feedback linearization controller,  $T = 0.1$  sec. (a) Actual output  $y(t)$  and desired output  $y_d(t)$ . (b) Tracking error  $e(t)$ . (c) Internal dynamics state  $x_2(t)$ .

Alternatively, one may take differences of  $y(k)$  until  $u(k)$  appears—the first difference and second difference are defined as

$$\begin{aligned}\Delta y(k) &\equiv y(k+1) - y(k) \\ \Delta^2 y(k) &\equiv \Delta y(k+1) - \Delta y(k) = y(k+2) - 2y(k+1) + y(k).\end{aligned}\quad (2.4.32)$$

The new states are defined as  $z_1(k) \equiv y(k), z_2(k) \equiv \Delta y(k), z_3(k) \equiv \Delta^2 y(k), \dots$ . This results in considerably more complex computations, but is theoretically more appropriate.

#### 2.4.3.2 Controller Design Step

Providing the system has well-defined relative degree so that  $\|g(x_k)\| > 0$  for all  $k$ , the obvious choice for a control input that causes  $y(k)$  to track the desired trajectory  $y_d(k)$  is now

$$u(k) = \frac{1}{g(x(k))} [-f(x(k)) + y_d(k+n) + K_n e(k+n-1) + \dots + K_2 e(k+1) + K_1 e(k)] \quad (2.4.33)$$

where the tracking error is  $e(k) = y_d(k) - y(k)$ . This yields the closed-loop dynamics

$$e(k+n) + K_n e(k+n-1) + \dots + K_2 e(k+1) + K_1 e(k) = 0, \quad (2.4.34)$$

which is stable as long as the gains are appropriately selected. This is a controller with an outer tracking loop and an inner nonlinear feedback linearization loop, exactly as its continuous-time counterpart in Fig. 2.4.1.

With respect to the defined output  $y(k)$ , this controller is noncausal. To compute the control input  $u(k)$  at time  $k$ , one must know the values of  $e(k)$  as well as its  $n-1$  future values. However, note that  $e(k+j) = y_d(k+j) - y(k+j) = y_d(k+j) - z_{j+1}(k)$ , for  $0 < j < n$ . In this work we shall assume full state variable feedback, that is, all the states  $x(k) = [x_1(k) \ x_2(k) \ \dots \ x_n(k)]^T$  are available at time  $k$  for computation of the control  $u_k$ . If only the output  $y(k)$  is available at time  $k$ , the controller design problem is considerably more complex—then a dynamic regulator containing a state observer must be designed. This can be accomplished using recurrent or dynamic neural networks as shown in (Kim and Lewis 1996).

One can write the closed-loop error dynamics in matrix form by defining the error vector

$$\underline{e}(k) \equiv \begin{bmatrix} e(k) \\ e(k+1) \\ \vdots \\ e(k+n-1) \end{bmatrix} \quad (2.4.35)$$

and the gain vector (2.4.27). Then the closed-loop error dynamics may be written as

$$\begin{aligned}\underline{e}(k+1) &= (A - bK)\underline{e}(k) \\ \xi(k+1) &= \Phi(\underline{e}(k), \xi(k), z_d(k))\end{aligned}\quad (2.4.36)$$

where  $A, b$  are the canonical form matrices (2.1.6). An equation has been added for the internal dynamics  $\xi(k)$ , which may be present in some examples. Note that  $\underline{e}(k)$  may be computed using the current state, for  $\underline{e}(k) = z_d(k) - x(k)$ , with the desired state given by  $z_d(k) \equiv [y_d(k) \ y_d(k+1) \ \dots \ y_d(k+n-1)]^T$  and  $z(k) \equiv [z_1(k) \ z_2(k) \ \dots \ z_n(k)]^T$ .

## 2.5 NONLINEAR STABILITY ANALYSIS AND CONTROLS DESIGN

For linear time-invariant systems it is straightforward to investigate stability by examining the locations of the poles in the  $s$ -plane. However, for nonlinear or non-autonomous (e.g. time-varying) systems there are no corresponding direct techniques. The (direct) Lyapunov approach provides methods for studying the stability of nonlinear systems and shows how to design control systems for such systems. For more information see (Lewis, Abdallah, and Dawson 1993) which deals with robot manipulator control, as well as (Khalil 1992, Slotine and Li 1991, Vidyasagar 1993), which have proofs and many excellent examples.

### 2.5.1 Lyapunov Analysis for Autonomous Systems

The autonomous (time-invariant) dynamical system

$$\dot{x} = f(x), \quad (2.5.1)$$

$x \in \mathbb{R}^n$ , could represent a closed-loop system after the controller has been designed. In Section 2.3.1 we defined several sorts of stability. We shall show here how to examine the stability properties using a generalized energy approach. An isolated equilibrium point  $x_e$  can always be brought to the origin by redefinition of coordinates; therefore, let us assume without loss of generality that the origin is an equilibrium point. First, we give some definitions and results. Then, some examples are presented to illustrate the power of the Lyapunov approach.

Let  $L(x) : \mathbb{R}^n \rightarrow \mathbb{R}$  be a scalar function such that  $L(0) = 0$ , and  $\mathcal{S}$  be a compact subset of  $\mathbb{R}^n$ . Then,  $L(x)$  is said to be:

Locally positive definite if  $L(x) > 0$  when  $x \neq 0$ , for all  $x \in \mathcal{S}$ . (Denoted  $L(x) > 0$ .)

Locally positive semidefinite if  $L(x) \geq 0$  for all  $x \in \mathcal{S}$ . (Denoted  $L(x) \geq 0$ .)

Locally negative definite if  $L(x) < 0$  when  $x \neq 0$ , for all  $x \in \mathcal{S}$ . (Denoted  $L(x) < 0$ .)

Locally negative semidefinite if  $L(x) \leq 0$  for all  $x \in \mathcal{S}$ . (Denoted  $L(x) \leq 0$ .)

An example of a positive definite function is the quadratic form  $L(x) = x^T Px$ , with  $P$  any matrix that is symmetric and positive definite. A definite function is allowed to be zero only when  $x = 0$ , a semidefinite function may vanish at points where  $x \neq 0$ . All these definitions are said to hold globally if  $\mathcal{S} = \mathbb{R}^n$ .

A function  $L(x) : \mathbb{R}^n \rightarrow \mathbb{R}$  with continuous partial derivatives is said to be a Lyapunov function for the system (2.5.1), if, for some compact set  $\mathcal{S} \subset \mathbb{R}^n$ , one has locally:

$$L(x) \text{ is positive definite, } L(x) > 0 \quad (2.5.2)$$

$$\dot{L}(x) \text{ is negative semidefinite, } \dot{L}(x) \leq 0 \quad (2.5.3)$$

where  $\dot{L}(x)$  is evaluated along the trajectories of (2.5.1) (as shown in an upcoming example). That is,

$$\dot{L}(x) = \frac{\partial L}{\partial x} \dot{x} = \frac{\partial L}{\partial x} f(x). \quad (2.5.4)$$

**Theorem 2.5.1 (Lyapunov Stability)** : If there exists a Lyapunov function for system (2.5.1), then the equilibrium point is stable in the sense of Lyapunov (SISL).  $\square$

This powerful result allows one to analyze stability using a generalized notion of energy. The Lyapunov function performs the role of an energy function. If  $L(x)$  is positive definite and its derivative is negative semidefinite, then  $L(x)$  is nonincreasing, which implies that the state  $x(t)$  is bounded. The next result shows what happens if the Lyapunov derivative is negative definite—then  $L(x)$  continues to decrease until  $\|x(t)\|$  vanishes.

**Theorem 2.5.2 (Asymptotic Stability)** : If there exists a Lyapunov function  $L(x)$  for system (2.5.1) with the strengthened condition on its derivative

$$\dot{L}(x) \text{ is negative definite, } \dot{L}(x) < 0 \quad (2.5.5)$$

then the equilibrium point is asymptotically stable (AS).  $\square$

To obtain global stability results one needs to expand the set  $S$  to all of  $\mathbb{R}^n$ , but also required is an additional radial unboundedness property.

**Theorem 2.5.3 (Global Stability)** :

a. *Globally SISL*. If there exists a Lyapunov function  $L(x)$  for system (2.5.1) such that (2.5.2) and (2.5.3) hold globally and

$$L(x) \rightarrow \infty \text{ as } \|x\| \rightarrow \infty \quad (2.5.6)$$

then the equilibrium point is globally SISL.

b. *Globally AS*. If there exists a Lyapunov function  $L(x)$  for system (2.5.1) such that (2.5.2) and (2.5.5) hold globally and also the unboundedness condition (2.5.6) holds, then the equilibrium point is globally AS (GAS).  $\square$

The global nature of this result of course implies that the equilibrium point mentioned is the only equilibrium point.

The next examples show the utility of the Lyapunov approach and make several points. Among the points of emphasis are that the Lyapunov function is intimately related to the energy properties of a system, and that Lyapunov techniques are closely related to the passivity notions in Section 2.3.2.

**Example 2.5.1 (Asymptotic Stability)** :

a. **Local Stability**

Consider the system

$$\begin{aligned}\dot{x}_1 &= x_1 x_2^2 + x_1(x_1^2 + x_2^2 - 3) \\ \dot{x}_2 &= -x_1^2 x_2 + x_2(x_1^2 + x_2^2 - 3).\end{aligned}$$

Stability may often be examined for nonlinear systems by selecting the quadratic Lyapunov function candidate

$$L(x) = \frac{1}{2}(x_1^2 + x_2^2)$$

which is a direct generalization of an energy function and has derivative

$$\dot{L}(x) = x_1 \dot{x}_1 + x_2 \dot{x}_2.$$

Evaluating this along the trajectories of the system simply involves substituting the state derivatives from the dynamics to obtain, in this case,

$$\dot{L}(x) = -(x_1^2 + x_2^2)(3 - x_1^2 - x_2^2)$$

which is negative as long as

$$\|x\| = x_1^2 + x_2^2 < 3.$$

Therefore,  $L(x)$  serves as a (local) Lyapunov function for the system, which is *locally* asymptotically stable. The system is said to have a *domain of attraction* with radius of 3.

The system trajectories are easily plotted by writing an M file for MATLAB (see Example 2.1.2 and Section 2.4.2). The result is shown in Fig. 2.5.1. Trajectories beginning outside  $\|x\| = 3$  in the phase plane cannot be guaranteed to converge.

### b. Global Stability

Consider now the system

$$\begin{aligned}\dot{x}_1 &= x_1 x_2^2 - x_1(x_1^2 + x_2^2) \\ \dot{x}_2 &= -x_1^2 x_2 - x_2(x_1^2 + x_2^2).\end{aligned}$$

Selecting again the Lyapunov function candidate

$$L(x) = \frac{1}{2}(x_1^2 + x_2^2)$$

yields

$$\dot{L}(x) = -(x_1^2 + x_2^2)^2$$

which is negative. The function  $L(x)$  is therefore a Lyapunov function and the system is *globally* asymptotically stable.  $\square$

### Example 2.5.2 (Lyapunov Stability) :

For the system

$$\begin{aligned}\dot{x}_1 &= x_1 x_2^2 - x_1 \\ \dot{x}_2 &= -x_1^2 x_2\end{aligned}$$

select the quadratic Lyapunov function candidate

$$L(x) = \frac{1}{2}(x_1^2 + x_2^2),$$

which has

$$\dot{L}(x) = -x_1^2.$$

This is only negative *semidefinite* (note that  $\dot{L}$  can be zero when  $x_2 \neq 0$ ). Therefore,  $L(x)$  is a Lyapunov function, but the system is only shown by this method to be SISL—that is  $\|x_1\|, \|x_2\|$  are both bounded.

The time histories shown in Fig. 2.5.2 were obtained using MATLAB.  $\square$

### Example 2.5.3 (Energy-Related Lyapunov Functions and Passivity) :

In many situations the simple quadratic Lyapunov functions do not suffice; it can be extremely difficult to find a Lyapunov function for complex systems. Failure to find a Lyapunov function may be because the system is not stable, or because the designer simply lacks insight and experience. The Lyapunov function is closely connected to the deep physical properties of the system, which can often aid in selecting a suitable candidate.

Consider the following system (Slotine and Li 1991)

$$\ddot{x} + b(\dot{x}) + c(x) = 0$$

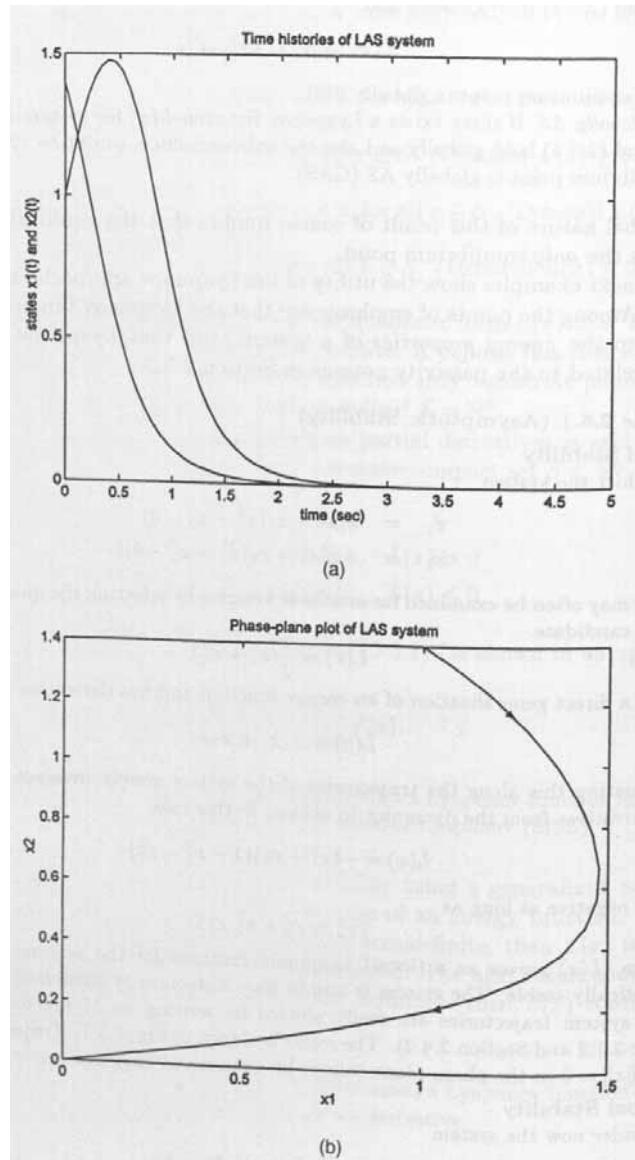


Figure 2.5.1: Sample trajectories of system with local asymptotic stability. (a)  $x_1(t)$  and  $x_2(t)$  versus  $t$ . (b) Phase-plane plot of  $x_2$  versus  $x_1$ .

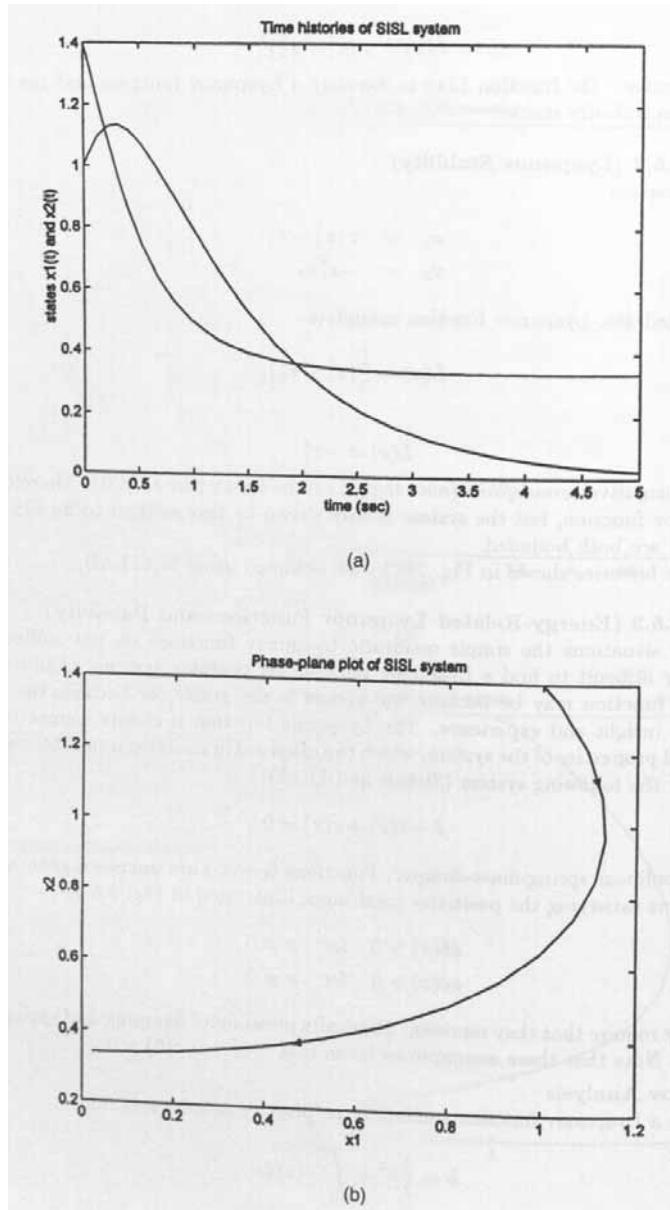


Figure 2.5.2: Sample trajectories of SISL system. (a)  $x_1(t)$  and  $x_2(t)$  versus  $t$ . (b) Phase-plane plot of  $x_2$  versus  $x_1$ .

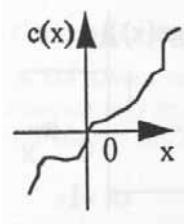


Figure 2.5.3: A function satisfying the condition  $xc(x) > 0$ .

which is a nonlinear spring-mass-damper. Functions  $b(\cdot), c(\cdot)$  are unknown general continuous functions satisfying the positivity conditions, illustrated in Fig. 2.5.3,

$$\begin{aligned} \dot{x}b(\dot{x}) &> 0 \quad \text{for } x \neq 0 \\ xc(x) &> 0 \quad \text{for } x \neq 0 \end{aligned}$$

which simply require that they represent physically meaningful damping and spring effects, respectively. Note that these assumptions mean that  $b(0) = 0, c(0) = 0$ .

#### a. Lyapunov Analysis

Select as a Lyapunov function candidate the positive definite function

$$L = \frac{1}{2}\dot{x}^2 + \int_0^x c(z)dz,$$

which is the sum of the kinetic and potential energy of the system. Differentiating using Leibniz' rule yields

$$\dot{L} = \dot{x}\ddot{x} + c(x)\dot{x} = -b(\dot{x})\dot{x} \leq 0,$$

where  $\ddot{x}$  was eliminated by substitution from the system dynamics. The entire state is  $[x \dot{x}]^T$ , however, only  $\dot{x}$  appears explicitly in  $\dot{L}$ , which is therefore only negative *semidefinite*. Therefore, this shows the system is SISL. In the Problems section it is shown that the system is actually AS by a technique based on Barbalat's Lemma introduced in Section 2.5.4.1. The function  $\dot{L}(x)$  is the power dissipated in the system.

#### b. Passivity

Lyapunov functions are closely related to the passivity notions introduced in Section 2.3.2. Considering the forced system

$$\ddot{x} + b(\dot{x}) + c(x) = F$$

one sees that

$$\dot{L} = \dot{x}F - b(\dot{x})\dot{x},$$

which is of the power form (2.3.5). Thus, the system from input  $F$  to output  $\dot{x}$  is passive.

□

### 2.5.2 Controller Design Using Lyapunov Techniques

Though we have presented Lyapunov analysis only for unforced systems in the form (2.5.1), which have no control input, these techniques also provide a powerful set of tools for designing feedback control systems for systems of the form

$$\dot{x} = f(x) + g(x)u. \tag{2.5.7}$$

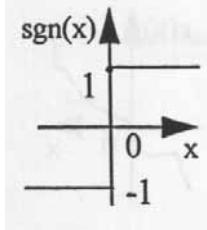


Figure 2.5.4: Signum function.

Thus, select a Lyapunov function candidate  $L(x) > 0$  and differentiate along the system trajectories to obtain

$$\dot{L}(x) = \frac{\partial L}{\partial x} \dot{x} = \frac{\partial L}{\partial x} [f(x) + g(x)u]. \quad (2.5.8)$$

Then, it is often possible to ensure that  $\dot{L} \leq 0$  by appropriate selection of  $u(t)$ . When this is possible, it generally yields controllers in state-feedback form, that is, where  $u(t)$  is a function of the states  $x(t)$ .

Practical systems with actuator limits and saturation often contain discontinuous functions including the signum function defined for scalars  $x \in \mathbb{R}$  as

$$sgn(x) = \begin{cases} 1, & x \geq 0 \\ -1, & x < 0 \end{cases}, \quad (2.5.9)$$

shown in Fig. 2.5.4, and for vectors  $x = [x_1 \ x_2 \dots x_n]^T \in \mathbb{R}^n$  as

$$sgn(x) = [sgn(x_i)] \quad (2.5.10)$$

where  $[z_i]$  denotes a vector  $z$  with components  $z_i$ . The discontinuous nature of such functions often makes it impossible to apply i/o feedback linearization where differentiation is required. In some cases, controller design can be carried out for systems containing discontinuities using Lyapunov techniques.

#### Example 2.5.4 (Controller Design by Lyapunov Analysis) :

Consider the system

$$\begin{aligned} \dot{x}_1 &= x_2 sgn(x_1) \\ \dot{x}_2 &= x_1 x_2 + u \end{aligned}$$

which has an actuator nonlinearity. A control input may not be designed using feedback linearization techniques, as differentiation of the signum function would be required. However, a stabilizing controller can easily be designed using Lyapunov techniques.

Select the Lyapunov function candidate

$$L(x) = \frac{1}{2}(x_1^2 + x_2^2),$$

and evaluate

$$\dot{L}(x) = x_1 \dot{x}_1 + x_2 \dot{x}_2 = x_2[x_1 sgn(x_1) + x_1 x_2 + u].$$

Now selecting the feedback control

$$u = -kx_2 - x_1x_2 - x_1 \operatorname{sgn}(x_1)$$

yields

$$\dot{L}(x) = -kx_2^2$$

so that  $L(x)$  is rendered a (closed-loop) Lyapunov function. Since  $\dot{L}$  is negative semidefinite, the closed-loop system with this controller is SISL.

Note that this controller has elements of feedback linearization in that the control input  $u(t)$  is selected to cancel some nonlinearities, with an additional proportional gain term  $-kx_2$  to provide the stabilizing effect. However, no differentiation of the right-hand sides of the state equation is needed in the Lyapunov approach. There are some issues in this specific example, such as the selection of a discontinuous control signal, which could cause chattering (as well as violation of the Caratheodory conditions (Khalil 1992)). In practice, however, the system dynamics often provide low-pass filtering, so that the controllers work well.  $\square$

#### 2.5.2.1 Lyapunov Analysis and Controls Design for Linear Systems

For general nonlinear systems it is not always easy to find a Lyapunov function. Thus, failure to find a Lyapunov function may be because the system is not stable, or because the designer simply lacks insight and experience. However, in the case of linear time-invariant systems

$$\dot{x} = Ax \quad (2.5.11)$$

Lyapunov analysis is simplified, and a Lyapunov function is easy to find, if one exists.

**Stability Analysis.** Select as a Lyapunov function candidate the quadratic form

$$L(x) = \frac{1}{2}x^T Px, \quad (2.5.12)$$

where  $P$  is a constant symmetric positive definite matrix. Since  $P > 0$ , then  $x^T Px$  is a positive definite function. This function is a generalized norm, which serves as a system energy function. Then,

$$\dot{L}(x) = \frac{1}{2}[\dot{x}^T Px + x^T P\dot{x}] \quad (2.5.13)$$

$$= \frac{1}{2}x^T[A^T P + PA]x. \quad (2.5.14)$$

For stability one requires negative semidefiniteness. Thus, there must exist a symmetric positive semidefinite matrix  $Q$  such that

$$\dot{L}(x) = -x^T Qx. \quad (2.5.15)$$

This results in the next theorem.

**Theorem 2.5.4 (Lyapunov Theorem for Linear Systems) :**

The system (2.5.11) is SISL if there exist matrices  $P > 0, Q \geq 0$  that satisfy the Lyapunov equation

$$A^T P + PA = -Q. \quad (2.5.16)$$

If there exists a solution such that both  $P$  and  $Q$  are positive definite, the system is AS.  $\square$

*It can be shown that this theorem is both necessary and sufficient. That is, for LTI systems, if there is no Lyapunov function of the quadratic form (2.5.12), then there is no Lyapunov function. This result provides an alternative to examining the eigenvalues of the  $A$  matrix.*

**Lyapunov Design of LTI Feedback Controllers.** These notions offer a valuable procedure for LTI control system design. Note that the closed-loop system with state feedback

$$\dot{x} = Ax + Bu \quad (2.5.17)$$

$$u = -Kx \quad (2.5.18)$$

is SISL if, and only if, there exist matrices  $P > 0, Q \geq 0$  that satisfy the closed-loop Lyapunov equation

$$(A - BK)^T P + P(A - BK) = -Q. \quad (2.5.19)$$

If there exists a solution such that both  $P$  and  $Q$  are positive definite, the system is AS.

Now suppose there exist  $P > 0, Q > 0$  that satisfy the matrix Riccati equation

$$A^T P + PA + Q - PBR^{-1}B^T P = 0 \quad (2.5.20)$$

for some matrix  $R > 0$ . Select now the feedback gain as

$$K = R^{-1}B^T P. \quad (2.5.21)$$

Then, one may write (2.5.20) in terms of  $K$  as

$$\begin{aligned} 0 &= A^T P + PA + Q - PBR^{-1}B^T P \\ 0 &= A^T P + PA + Q - PBR^{-1}B^T P - PBR^{-1}B^T P + PBR^{-1}B^T P \\ 0 &= A^T P + PA + Q - K^T B^T P - PBK + K^T RK \\ 0 &= (A - BK)^T P + P(A - BK) + [Q + K^T RK]. \end{aligned} \quad (2.5.22)$$

The last equation is a closed-loop Lyapunov equation, and verifies that this selection for the feedback gain matrix  $K$  guarantees closed-loop asymptotic stability.

Note that the Riccati equation depends only on known matrices—the system  $(A, B)$  and two symmetric design matrices  $Q, R$  that need only be selected positive definite. There are many good routines that can find the solution  $P$  to this equation providing only that  $(A, B)$  is controllable (e.g. MATLAB). Then, a stabilizing gain is given by (2.5.21). If different design matrices  $Q, R$  are selected, different closed-loop poles will result. This approach goes far beyond classical frequency domain or root locus design techniques in that it allows the determination of stabilizing feedbacks for complex multivariable systems by simply solving a matrix design equation. For more details on this linear quadratic (LQ) design techniques see (Lewis and Syrmos 1995).

### 2.5.3 Lyapunov Analysis for Non-Autonomous Systems

We now consider non-autonomous (time-varying) dynamical systems of the form

$$\dot{x} = f(x, t), \quad t \geq t_0 \quad (2.5.23)$$

$x \in \mathbb{R}^n$ . Assume again that the origin is an equilibrium point. For non-autonomous systems the basic concepts just introduced still hold, but the explicit time dependence of the system must be taken into account. The basic issue is that the Lyapunov function may now depend on time. In this situation, the definitions of definiteness must be modified, and the notion of ‘decrecence’ is needed.

Let  $L(x, t) : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}$  be a scalar time-varying function such that  $L(0, t) = 0$ , and  $\mathcal{S}$  be a compact subset of  $\mathbb{R}^n$ . Then,  $L(x, t)$  is said to be:

Locally positive definite if  $L(x, t) \geq L_0(x)$  for some time-invariant positive definite  $L_0(x)$ , for all  $t \geq 0$  and  $x \in \mathcal{S}$ . (Denoted  $L(x, t) > 0$ .)

Locally positive semidefinite if  $L(x, t) \geq L_0(x)$  for some time-invariant positive semidefinite  $L_0(x)$ , for all  $t \geq 0$  and  $x \in \mathcal{S}$ . (Denoted  $L(x, t) \geq 0$ .)

Locally negative definite if  $L(x, t) \leq L_0(x)$  for some time-invariant negative definite  $L_0(x)$ , for all  $t \geq 0$  and  $x \in \mathcal{S}$ . (Denoted  $L(x, t) < 0$ .)

Locally negative semidefinite if  $L(x, t) \leq L_0(x)$  for some time-invariant negative semidefinite  $L_0(x)$ , for all  $t \geq 0$  and  $x \in \mathcal{S}$ . (Denoted  $L(x, t) \leq 0$ .)

Thus, for definiteness of time-varying functions, a time-invariant definite function must be dominated. All these definitions are said to hold globally if  $\mathcal{S} = \mathbb{R}^n$ .

A time-varying function  $L(x, t) : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}$  is said to be decrescent if  $L(0, t) = 0$ , and there exists a time-invariant positive definite function  $L_1(x)$  such that

$$L(x, t) \leq L_1(x), \quad \forall t \geq 0. \quad (2.5.24)$$

The notions of decrecence and positive definiteness for time-varying functions are depicted in Fig. 2.5.5.

#### Example 2.5.5 (Decrescent Function) :

Consider the time-varying function

$$L(x, t) = x_1^2 + \frac{x_2^2}{2 + \sin t}.$$

Note that  $1 \leq 2 + \sin t \leq 3$ , so that

$$L(x, t) \geq L_0(x) \equiv x_1^2 + \frac{x_2^2}{3},$$

and  $L(x, t)$  is globally positive definite. Also,

$$L(x, t) \leq L_1(x) \equiv x_1^2 + x_2^2$$

so that it is decrescent. □

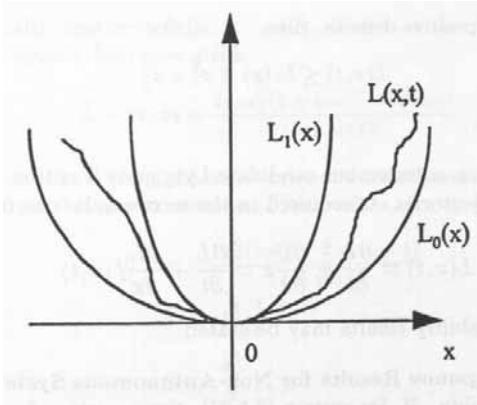


Figure 2.5.5: Depiction of a time-varying function  $L(x, t)$  that is positive definite ( $L_0(x) < L(x, t)$ ) and decrescent ( $L(x, t) \leq L_1(x)$ ).

To evaluate the non-autonomous candidate Lyapunov function derivative  $\dot{L}(x, t)$  along the system trajectories, as required in the next result, one must use

$$\dot{L}(x, t) = \frac{\partial L}{\partial t} + \frac{\partial L}{\partial x} \dot{x} = \frac{\partial L}{\partial t} + \frac{\partial L}{\partial x} f(x, t). \quad (2.5.25)$$

Now the following stability results may be stated.

**Theorem 2.5.5 (Lyapunov Results for Non-Autonomous Systems) :**

a. *Lyapunov Stability.* If, for system (2.5.23), there exists a function  $L(x, t)$  with continuous partial derivatives, such that for  $x$  in a compact set  $\mathcal{S} \subset \Re^n$

$$L(x, t) \text{ is positive definite, } L(x, t) > 0 \quad (2.5.26)$$

$$\dot{L}(x, t) \text{ is negative semidefinite, } \dot{L}(x, t) \leq 0 \quad (2.5.27)$$

then the equilibrium point is SISL.

b. *Asymptotic Stability.* If, furthermore, condition (2.5.27) is strengthened to

$$\dot{L}(x, t) \text{ is negative definite, } \dot{L}(x, t) < 0 \quad (2.5.28)$$

then the equilibrium point is AS.

c. *Global Stability.* If the equilibrium point is SISL or AS, if  $\mathcal{S} = \Re^n$ , and in addition the radial unboundedness condition holds:

$$L(x, t) \rightarrow \infty \quad \forall t \text{ as } \|x\| \rightarrow \infty \quad (2.5.29)$$

then the stability is global.

d. *Uniform Stability.* If the equilibrium point is SISL or AS, and in addition  $L(x, t)$  is decrescent (e.g. (2.5.24) holds), then the stability is uniform (e.g. independent of  $t_0$ ).  $\square$

The equilibrium point may be both uniformly and globally stable— e.g. if all the conditions of the theorem hold, then one has GUAS.

**Example 2.5.6 (Damped Mathieu Equation) :**

This example illustrates that Lyapunov functions may often be complex functions that are not easy to find. The damped Mathieu equation is

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= -x_2 - (2 + \sin t)x_1.\end{aligned}$$

To analyze its stability, postulate the Lyapunov function candidate

$$L(x, t) = x_1^2 + \frac{x_2^2}{2 + \sin t}.$$

Note that  $\beta \equiv 2 + \sin t \leq 3$ , so that

$$L(x, t) \geq L_0(x) \equiv x_1^2 + \frac{x_2^2}{3},$$

and  $L(x, t)$  is globally positive definite.

Taking the Lyapunov derivative yields

$$\dot{L} = 2x_1\dot{x}_1 + \frac{2x_2\dot{x}_2(2 + \sin t) - x_2^2 \cos t}{(2 + \sin t)^2}.$$

At this point, to evaluate  $\dot{L}$  along the trajectories of the system one simply substitutes the state derivatives from the system equation. The result is

$$\begin{aligned}\dot{L} &= -\frac{x_2^2(\cos t + 2 \sin t + 4)}{(2 + \sin t)^2} \\ &= -x_2^2 \frac{\alpha}{\beta^2} \\ &\leq -\frac{x_2^2}{9}\end{aligned}$$

since  $\alpha \equiv \cos t + 2 \sin t + 4 \geq 1$  and  $\beta \leq 3$ . Therefore,  $\dot{L}$  is negative semidefinite. This shows that the Lyapunov candidate is indeed a Lyapunov function so that the system is SISL. In fact  $\beta \geq 1$ , implying that  $L(x, t) \leq x_1^2 + x_2^2$  so that it is decrescent. This shows the system to be uniformly SISL.

The time histories shown in Fig. 2.5.6 were obtained using MATLAB. □

#### 2.5.4 Extensions of Lyapunov Techniques and Bounded Stability

*The Lyapunov results so far presented have allowed the determination of SISL, if there exists a function such that  $L(x, t) > 0$ ,  $\dot{L}(x, t) \leq 0$ , and AS, if there exists a function such that  $L(x, t) > 0$ ,  $\dot{L}(x, t) < 0$ . Various extensions of these results allow one to determine more about the stability properties by further examining the deeper structure of the system dynamics (Slotine and Li 1991).*

##### 2.5.4.1 Barbalat's Lemma Extension of Lyapunov Analysis

*The first result is based on Barbalat's Lemma (Section 2.2.2) applied to the Lyapunov derivative  $\dot{L}$ . It gives a condition under which  $\dot{L} \rightarrow 0$ .*

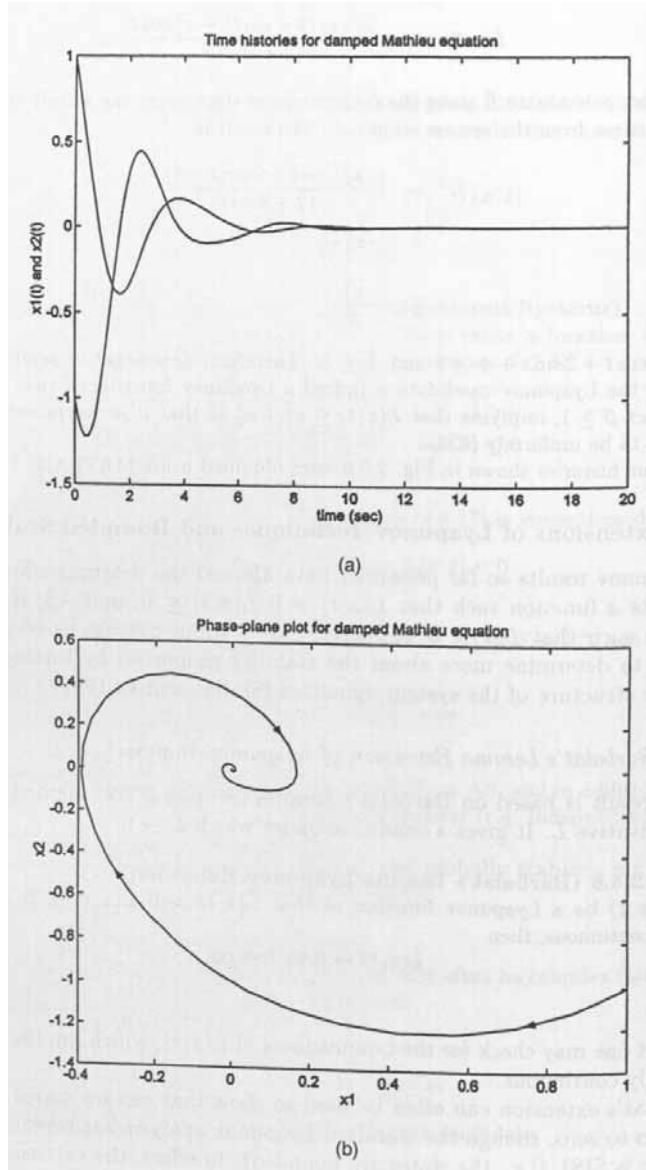


Figure 2.5.6: Sample trajectories of Mathieu system. (a)  $x_1(t)$  and  $x_2(t)$  versus  $t$ .  
(b) Phase-plane plot of  $x_2$  versus  $x_1$ .

**Theorem 2.5.6 (Barbalat's Lemma Lyapunov Extension) :**

Let  $L(x, t)$  be a Lyapunov function so that  $L(x, t) > 0, \dot{L}(x, t) \leq 0$ . If  $\dot{L}(x, t)$  is uniformly continuous, then

$$\dot{L}(x, t) \rightarrow 0 \text{ as } t \rightarrow \infty.$$

□

Recall that one may check for the boundedness of  $\ddot{L}(x, t)$ , which implies that  $\dot{L}(x, t)$  is uniformly continuous.

Barbalat's extension can often be used to show that certain states of a system actually go to zero, though the standard Lyapunov analysis has revealed only that the system is SISL (i.e., the states are bounded). In effect, the extension can show that the states converge to a smaller region than that implied by Lyapunov analysis, which in the case of SISL only demonstrates that they converge to a bounded region. In the next example the states are shown to converge to the line  $x_1 = 0$ , and in the second example they are shown to converge to the origin  $x_1 = x_2 = 0$ , so that the system is actually AS.

**Example 2.5.7 (Asymptotic Stability Using Barbalat's Extension) :**

The system in Example 2.5.2 was shown to be SISL using Lyapunov techniques, so that  $\|x_1\|, \|x_2\|$  are both bounded. The Lyapunov derivative was

$$\dot{L}(x) = -x_1^2.$$

To apply the Barbalat's extension, one must verify that  $\dot{L}$  is uniformly continuous. To accomplish this, differentiate again to determine

$$\ddot{L} = -2x_1\dot{x}_1 = -2x_1^2x_2^2 + 2x_1^2,$$

where the system dynamics were used to substitute for  $\dot{x}_1$ . Since Lyapunov analysis has shown  $\|x\|$  to be bounded, it follows that  $\ddot{L}$  is bounded. Therefore,  $\dot{L}$  is uniformly continuous.

Barbalat's Lemma now shows that  $\dot{L}$  goes to zero with time. Therefore,  $x_1(t) \rightarrow 0$ . That is, state component  $x_1(t)$  actually goes to zero, though standard Lyapunov analysis only showed it to be bounded. The time history plots given in Example 2.5.2 corroborate this conclusion. □

**Example 2.5.8 (Barbalat's Extension to Show AS for Mathieu Equation) :**

In Example 2.5.6 it was shown using standard Lyapunov analysis that the damped Mathieu equation is SISL, so that  $\|x\|$  is bounded. The Lyapunov derivative was

$$\dot{L} = -x_2^2 \frac{\alpha}{\beta^2}$$

with  $\alpha \equiv \cos t + 2 \sin t + 4, \beta \equiv 2 + \sin t$ .

To apply the Barbalat's extension, one must verify that  $\dot{L}(x, t)$  is uniformly continuous. To accomplish this, differentiate again to determine

$$\ddot{L} = [-2\alpha x_2 \dot{x}_2 - x_2^2(-\sin t + 2 \cos t)]/\beta^2 + [2\alpha x_2^2 \cos t]/\beta^3.$$

Now, substitute from the system dynamics for  $\dot{x}_2$  to obtain

$$\ddot{L} = [x_2^2 + 2\alpha x_1 x_2]/\beta + [4x_2^2 \sin t]/\beta^2 + [2\alpha x_2^2 \cos t]/\beta^3.$$

Since Lyapunov analysis has shown  $\|x\|$  to be bounded, and  $\alpha, \beta$  are both upper and lower bounded, it follows that  $\ddot{L}$  is bounded. Therefore,  $\dot{L}$  is uniformly continuous.

Barbalat's Lemma now shows that  $\dot{L}$  goes to zero with time. Therefore,  $x_2(t) \rightarrow 0$ . However, in this example Barbalat opens the door for an even deeper analysis by appealing yet again to the dynamics. Specifically, according to the dynamics, one thus has in the limit

$$\begin{aligned}\dot{x}_1 &= 0 \\ 0 = \dot{x}_2 &= -(2 + \sin t)x_1\end{aligned}$$

since  $(2 + \sin t) \neq 0$ , it follows that  $x_1(t) \rightarrow 0$ . Therefore, the system is AS. This conclusion is actually verified by the time history plots given in Example 2.5.6.  $\square$

#### 2.5.4.2 UUB Analysis and Controls Design

We have seen how to demonstrate that a system is SISL or AS using Lyapunov techniques. However, in practical applications there are often unknown disturbances or modeling errors, so that even SISL is too strong to expect in closed-loop systems. Typical examples are systems of the form

$$\dot{x} = f(x, t) + d(t), \quad (2.5.30)$$

with  $d(t)$  an unknown but bounded disturbance. A more practical notion of stability is uniform ultimate boundedness (UUB). The next result shows that UUB is guaranteed if the Lyapunov derivative is negative outside some bounded region of  $\mathbb{R}^n$ .

**Theorem 2.5.7 (UUB by Lyapunov Analysis) :**

If, for system (2.5.30), there exists a function  $L(x, t)$  with continuous partial derivatives such that for  $x$  in a compact set  $\mathcal{S} \subset \mathbb{R}^n$

$$L(x, t) \text{ is positive definite, } L(x, t) > 0$$

$$\dot{L}(x, t) < 0 \text{ for } \|x\| > R$$

for some  $R > 0$  such that the ball of radius  $R$  is contained in  $\mathcal{S}$ , then the system is UUB, and the norm of the state is bounded to within a neighborhood of  $R$ .  $\square$

In this result note that  $\dot{L}$  must be strictly less than zero outside the ball of radius  $R$ . If one has only that  $\dot{L}(x, t) \leq 0$  for  $\|x\| > R$ , then nothing may be concluded about the system stability.

For systems of the sort satisfying the theorem, there may be some disturbance effects that push the state away from the equilibrium. However, if the state becomes too large, the dynamics tend to pull it back towards the equilibrium. Due to these two opposing effects that balance when  $\|x\| \approx R$ , the time histories tend to remain in the vicinity of  $\|x\| = R$ . In effect, the norm of the state is effectively or practically bounded by  $R$ .

The notion of the ball outside which  $\dot{L}$  is negative should not be confused with that of domain of attraction—in Example 2.5.1a. It was shown there that the system is AS as long as one has  $\|x_0\| < 3$ , defining a domain of attraction of radius 3.

The next examples show how to use this result. They make the point that it can also be used as a control design technique where the control input is selected to guarantee that the conditions of the theorem hold.

**Example 2.5.9 (UUB Lyapunov Extension) :**

The system

$$\begin{aligned}\dot{x}_1 &= x_1 x_2^2 - x_1(x_1^2 + x_2^2 - 3) \\ \dot{x}_2 &= -x_1^2 x_2 - x_2(x_1^2 + x_2^2 - 3)\end{aligned}$$

is closely related to the systems in Example 2.5.1. Select the Lyapunov function candidate

$$L(x) = \frac{1}{2}(x_1^2 + x_2^2)$$

and evaluate

$$\begin{aligned}\dot{L}(x) &= x_1 \dot{x}_1 + x_2 \dot{x}_2 \\ &= -(x_1^2 + x_2^2)(x_1^2 + x_2^2 - 3),\end{aligned}$$

which is negative as long as

$$\|x\| = x_1^2 + x_2^2 > 3.$$

Standard Lyapunov techniques fail in this example since  $\dot{L}$  is not even negative semidefinite. However, the UUB extension shows that the system is UUB, so that after a finite settling time the trajectories will remain in the vicinity of  $\|x\| = 3$ .  $\square$

**Example 2.5.10 (UUB of Linear System with Disturbance) :**

It is usual in practical systems to have unknown disturbances, which are often bounded by some known amount. Such disturbances result in UUB and require the UUB extension for analysis. Suppose the system

$$\dot{x} = Ax + d$$

has  $A$  stable and a disturbance  $d(t)$  that is unknown but bounded so that  $\|d\| < d_M$ , with the bound  $d_M$  known.

Select the Lyapunov function candidate

$$L(x) = \frac{1}{2}x^T Px$$

and evaluate

$$\begin{aligned}\dot{L}(x) &= \frac{1}{2}(\dot{x}^T Px + x^T P \dot{x}) \\ &= \frac{1}{2}x^T(A^T P + PA)x + x^T Pd \\ &= -\frac{1}{2}x^T Qx + x^T Pd\end{aligned}$$

where  $(P, Q)$  satisfy the Lyapunov equation

$$A^T P + PA = -Q.$$

One may now use norm inequalities (e.g. (2.2.14)) to write

$$\begin{aligned}\dot{L}(x) &\leq -\frac{1}{2}\sigma_{min}(Q)\|x\|^2 + \sigma_{max}(P)\|x\|\cdot\|d\| \\ &\leq -\|x\|\left[\frac{1}{2}\sigma_{min}(Q)\|x\| - \sigma_{max}(P)\|d\|\right]\end{aligned}$$

which is negative as long as

$$\|x\| \geq 2 \frac{\sigma_{\max}(P)d_M}{\sigma_{\min}(Q)}.$$

Thus, if the disturbance magnitude bound increases, the norm of the state will also increase. There are standard results on norms of Lyapunov solutions that can be used to argue that as the system becomes more stable, so that the system poles move further into the left-half plane, the ratio  $\frac{\sigma_{\max}(P)}{\sigma_{\min}(Q)}$  decreases.  $\square$

**Example 2.5.11 (UUB of Closed-Loop System) :**

The UUB extension can be used to design stable closed-loop systems. The system

$$\begin{aligned}\dot{x}_1 &= x_1 x_2^2 - 10x_1 + d \\ \dot{x}_2 &= -x_1^2 x_2 - x_2 \sin x_1 + u\end{aligned}$$

is excited by an unknown disturbance whose magnitude is bounded so that  $\|d\| < d_M$ . To find a control that stabilizes the system and mitigates the disturbance effect, select

$$L = \frac{1}{2}(x_1^2 + x_2^2)$$

so that

$$\dot{L} = -10x_1^2 + x_1 d - x_2^2 \sin x_1 + x_2 u.$$

Selecting now the control input

$$u = x_2 \sin x_1 - kx_2$$

cancels the sinusoidal nonlinearity and provides a stabilizing term, yielding the closed-loop Lyapunov derivative

$$\begin{aligned}\dot{L} &= -10x_1^2 + x_1 d - kx_2^2 \\ &= -x^T \begin{bmatrix} 10 & 0 \\ 0 & k \end{bmatrix} x + x^T \begin{bmatrix} d \\ 0 \end{bmatrix} \\ &\equiv -x^T Q x + x^T \underline{d}.\end{aligned}$$

Therefore,

$$\begin{aligned}\dot{L} &\leq -\sigma_{\min}(Q)\|x\|^2 + \|x\| \cdot \|d\| \\ \dot{L} &\leq -\sigma_{\min}(Q)\|x\|^2 + d_M\|x\|,\end{aligned}$$

which is negative as long as

$$\|x\| > \frac{d_M}{\sigma_{\min}(Q)}.$$

However,  $\sigma_{\min}(Q) = \min\{10, k\}$ , so that the UUB bound is made smaller by increasing the feedback gain  $k$  up to the limit when  $k = 10$ . After that, increasing  $k$  does not decrease the bound.

The system trajectories shown in Fig. 2.5.7 display the expected closed-loop behavior. They were made using  $k = 20$  with  $d(t)$  a random signal uniformly distributed on  $[0, 5]$  so that  $d_M = 5$ . The initial condition was  $x(0) = [1 \ -1]'$ .  $\square$

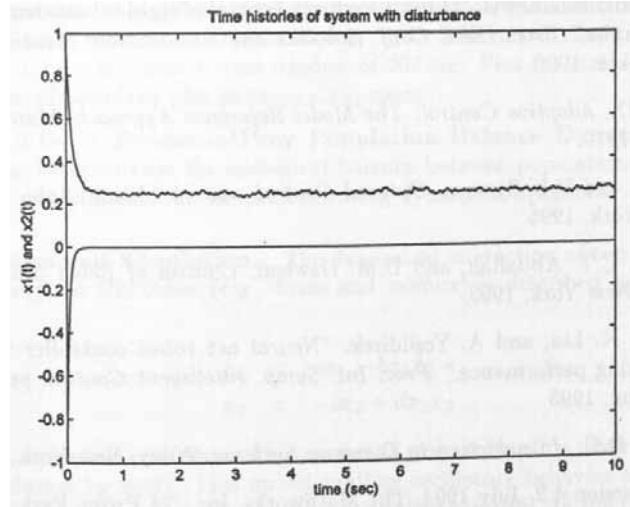


Figure 2.5.7: Sample closed-loop trajectories of UUB system.

#### 2.5.4.3 Bellman-Gronwall Lemma

Another result useful in showing boundedness is the following.

**Theorem 2.5.8 (Bellman-Gronwall Lemma) :**

Suppose that for all  $t \geq t_0$  the function  $x(t)$  satisfies the inequality

$$x(t) \leq \beta(t) + \int_{t_0}^t \alpha(\tau)x(\tau)d\tau \quad (2.5.31)$$

with  $\alpha(\cdot), \beta(\cdot)$  nonnegative piecewise continuous functions. Then, a bound on  $x(t), t \geq t_0$  is given by

$$x(t) \leq \beta(t) + \int_{t_0}^t \alpha(\tau)\beta(\tau)\Gamma(\tau, t) d\tau \quad (2.5.32)$$

where

$$\Gamma(\tau, t) = e^{\int_{\tau}^t \alpha(\sigma) d\sigma}. \quad (2.5.33)$$

Moreover, if  $\beta$  is a constant, then

$$x(t) \leq \beta\Gamma(t_0, t). \quad (2.5.34)$$

□

The importance of this result lies in the fact that in (2.5.31) the signal  $x(t)$  appears on both sides of the inequality. However, the special form allows one to derive the bound (2.5.32), where  $x(t)$  appears only on the left.

## 2.6 REFERENCES

von Bertalanffy, L., *General System Theory*, Braziller, New York, 1968.

*Commuri, S., and F.L. Lewis, "Robust practical stabilization of nonlinear systems with ill-defined relative degree," Proc. IEEE Mediterranean Symp. New Directions in Control and Automation, pp. 299-306, June 1994.*

*Goodwin, C.G., and K.S. Sin, Adaptive Filtering, Prediction, and Control, Prentice-Hall, New Jersey, 1984.*

*Ioannou, P.A., and J. Sun, Robust Adaptive Control, Prentice-Hall, New Jersey, 1996.*

*Isidori, A., Nonlinear Control Systems, second edition, Springer-Verlag, Berlin, 1989*

*Kailath, T., Linear Systems, Prentice-Hall, New Jersey, 1980.*

*Kalckuhl, J.C., and K.J. Hunt, "Discrete-time neural model structures for continuous-time nonlinear systems," Neural Adaptive Control Technology, ed. R. Zbikowski and K.J. Hunt, Chapter 1, World Scientific, Singapore, 1996.*

*Khalil, H.K., Nonlinear Systems, Macmillan, New York, 1992.*

*Kim, Y.H., and F.L. Lewis, "Output feedback control of rigid robots using dynamic neural networks," Proc. IEEE Conf. Robotics and Automation, Minneapolis, pp. 1923-1928, Apr. 1996.*

*Landau, Y.D., Adaptive Control: The Model Reference Approach, Marcel Dekker, Inc., Basel, 1979.*

*Lewis, F.L., and V.L. Syrmos, Optimal Control, second edition, John Wiley and Sons, New York, 1995.*

*Lewis, F.L., C.T. Abdallah, and D.M. Dawson, Control of Robot Manipulators, Macmillan, New York, 1993.*

*Lewis, F.L., K. Liu, and A. Yeşildirek, "Neural net robot controller with guaranteed tracking performance," Proc. Int. Symp. Intelligent Control, pp. 225-231, Chicago., Aug. 1993.*

*Luenberger, D.G., Introduction to Dynamic Systems, Wiley, New York, 1979.*

*MATLAB version 4.2, July 1994, The Mathworks, Inc., 24 Prime Park Way, Natick, MA 01760, USA.*

*Sastry, S., and M. Bodson, Adaptive Control, Prentice-Hall, New Jersey, 1989.*

*Seron, M.M., D.J. Hill, and A.L. Fradkov, "Adaptive passification of nonlinear systems," Proc. IEEE Conf. Decision and Control, pp. 190-195, Dec. 1994.*

*Slotine, J.-J.E., and W. Li, Applied Nonlinear Control, Prentice-Hall, New Jersey, 1991.*

*Qu, Z., and D.M. Dawson, Robust Tracking Control of Robot Manipulators, IEEE Press, New York, 1996.*

Vidyasagar, M., *Nonlinear System Analysis*, second edition, Prentice-Hall, New Jersey, 1993.

Whitehead, A.N., *Science and the Modern World*, Lowell Lectures (1925), Macmillan, New York, 1953.

## 2.7 PROBLEMS

### Section 2.1

**Problem 2.1-1 : Simulation of Chaotic System.** The dynamics of the Lorenz attractor system are given by

$$\begin{aligned}\dot{x}_1 &= -\sigma(x_1 - x_2) \\ \dot{x}_2 &= rx_1 - x_2 - x_1x_3 \\ \dot{x}_3 &= -bx_3 + x_1x_2.\end{aligned}$$

The parameter  $\sigma$  is proportional to the Prandtl number in ordinary viscous fluids,  $r$  is comparable to the Rayleigh number in the Benard convection problem, and  $b$  is a positive constant. This system exhibits chaotic behavior. Simulate the trajectories using MATLAB with  $\sigma = 10, r = 28, b = 8/3$ . Use initial conditions of  $x_1 = 0.1, x_2 = 0.1, x_3 = 0.1$ , and a time window of 200 sec. Plot the states versus time, and also the phase-plane plot in  $(x_1, x_2, x_3)$ -space.

**Problem 2.1-2 : Predator/Prey Population Balance Dynamics.** It was observed by Volterra that the ecological balance between populations, as detailed by Darwin and Wallace, can be modeled using dynamical equations.

**a. Dynamics and Simulation.** The dynamical interaction of two populations, one predatory on the other (e.g. foxes and rabbits) is described by Luenberger (1979)

$$\begin{aligned}\dot{x}_1 &= ax_1 - bx_1x_2 \\ \dot{x}_2 &= -cx_2 + dx_1x_2\end{aligned}$$

with  $a, b, c, d$  positive constants. The number of prey at time  $t$  is described by  $x_1(t)$  and of predators by  $x_2(t)$ . This model exhibits oscillatory behavior corresponding to alternating periods where the prey is scarce then plentiful. In the first equation, the first term reveals that  $x_1(t)$  will increase if  $x_2$  is zero; the second term shows the effect of encounters between predator and prey, indicating that a positive  $x_2$  causes  $x_1(t)$  to decrease. In the second equation, the first term reveals that  $x_2(t)$  will decrease if  $x_1$  is zero; the second term shows that a positive  $x_1$  causes  $x_2(t)$  to increase.

Simulate this system in MATLAB using different values of the constants (begin on the first run with all values equal to 1). Plot  $x_1(t), x_2(t)$  versus  $t$ , and also the phase-plane plot  $x_1$  versus  $x_2$ . Be sure to use a sufficiently long simulation run time to observe all effects. Start with nonzero initial conditions.

**b. Effect of Overcrowding.** *The effects of overcrowding of prey in the presence of scarce food resources can be included by adding a term so that the model becomes*

$$\begin{aligned}\dot{x}_1 &= ax_1 - bx_1x_2 - ex_1^2 \\ \dot{x}_2 &= -cx_2 + dx_1x_2\end{aligned}$$

*with  $e > 0$ . Simulate this system in MATLAB and compare its behavior to the model in part (a).*

**Problem 2.1-3 : Spread of an Epidemic.** *The development of an epidemic can be described by (Luenberger 1979)*

$$\begin{aligned}\dot{x}_1 &= -\beta x_1 x_2 \\ \dot{x}_2 &= \beta x_1 x_2 - \gamma x_2 \\ \dot{x}_3 &= \gamma x_2\end{aligned}$$

*with  $x_1$  the number of susceptible individuals,  $x_2$  the number of infected individuals, and  $x_3$  the number of individuals who are either immune or removed by isolation or death. The infection rate constant is  $\beta > 0$ , and the removal rate constant is  $\gamma > 0$ .*

*Simulate this system in MATLAB using different values of the constants. Plot  $x_1(t), x_2(t), x_3(t)$  versus  $t$ , and also phase-plane plots  $x_i$  versus  $x_j$ . Plot the 3-D plot in  $x_1, x_2, x_3$ -space. Be sure to use a sufficiently long simulation run time to observe all effects. Start with nonzero initial conditions.*

**Problem 2.1-4 : Discrete-Time Multi-Input Brunovsky Canonical Form.** *Write the discrete-time version of (2.1.7). Draw a block diagram of the system.*

**Problem 2.1-5 : Simulation of Compound Interest System.** *Simulate the system of Example 2.1.3 and plot the state versus time.*

**Problem 2.1-6 : Genetics.** *Many congenital diseases can be explained as the result of both genes at a single location being the same recessive gene (Luenberger 1979). Under some assumptions, the frequency of the recessive gene at generation  $k$  is given by the recursion*

$$x(k+1) = \frac{x(k)}{1+x(k)}.$$

*Simulate in MATLAB using  $x(0)=80$ . Observe that  $x(k)$  converges to zero, but very slowly. This explains why deadly genetic diseases can remain active for hundreds of generations. Simulate the system starting for a small negative value of  $x(0)$  and observe that it tends away from zero.*

**Problem 2.1-7 : Discrete-Time System.** *Simulate the system*

$$\begin{aligned}x_1(k+1) &= \frac{x_2(k)}{1+x_2(k)^2} \\ x_2(k+1) &= \frac{x_1(k)}{1+x_2(k)^2}\end{aligned}$$

*using MATLAB. Plot  $x_1(k), x_2(k)$  versus  $k$  and the phase-plane plot.*

## Section 2.2

**Problem 2.2-1 : Singular Value Decomposition.**

(a) Verify that, in the SVD (2.2.10), the singular values of  $A$  are the square roots of the nonzero eigenvalues of  $AA^T$ , or equivalently of  $A^TA$ . This may be achieved by substituting from (2.2.10) into  $A^TA$  and  $AA^T$  and then using the properties of  $U, V$ . (b) Find the relation between the singular values and the eigenvalues of a (square) symmetric matrix. (c) Find the relation between the singular values and the eigenvalues of a general square matrix.

## Section 2.3

**Problem 2.3-1 : Passivity.** Verify the passivity of the nonlinear spring-mass-damper system

$$m\ddot{x} + x^2\dot{x}^3 + x^7 = F.$$

Select

$$L(t) = \frac{1}{2}\dot{x}^2 + \frac{1}{8}x^8$$

which is the sum of the kinetic and potential energy. Find the dissipated power.

**Problem 2.3-2 : Passivity.** Verify the passivity of the nonlinear system

$$\begin{aligned} \dot{x} + x^3 &= u \\ y &= x - \sin^2 x. \end{aligned}$$

Select

$$L(t) = \int_0^x (z - \sin^2 z) dz.$$

**Problem 2.3-3 : Passivity Definitions.** Verify that (2.3.5) and (2.3.6) are saying the same thing.

**Problem 2.3-4 : Discrete-Time Passivity Definitions.** Verify that (2.3.11) and (2.3.12) are saying the same thing. To accomplish this, sum both sides of the former equation and use the boundedness assumption on  $L(k)$ .

**Problem 2.3-5 : Passivity of Feedback Interconnection.** (a) Verify Equation (2.3.16). (b) Show that if both systems in the feedback connection are SSP, the closed-loop system is SSP. (c) Show that if one system is SSP but the other only passive, the closed-loop system is generally only passive.

**Problem 2.3-6 : Passivity of Interconnections.** (a) Show that parallel combinations of passive systems are passive. (b) Investigate the passivity of series combinations of passive systems.

**Problem 2.3-7 : Observability of LTI Systems.** Begin with the LTI solution (2.1.10) with  $u(t) = 0$ . Write down  $y(t)$  in terms of  $x(0)$  and verify that the initial state can be reconstructed using (2.3.30).

**Problem 2.3-8 : Controllability of LTI Systems.** Begin with the LTI solution (2.1.10). Verify that the initial state can be driven to the final state  $x(T)$  using (2.3.35). Hint— substitute the latter equation into the former.

## Section 2.4

**Problem 2.4-1 : Internal Dynamics.** Repeat the feedback linearization design for the system of (2.4.2) if the output is redefined as  $y = x_2$ ; that is,  $x_2(t)$  is required to track the desired trajectory  $y_d(t)$ . Find  $f(x)$ ,  $g(x)$ . Examine the internal dynamics; are they stable?

**Problem 2.4-2 : Feedback Linearization.** The system

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= x_3 \\ \dot{x}_3 &= -2x_1x_3 + \sin x_2 + 5u\end{aligned}$$

is in Brunovsky form. If  $x_1(t)$  is required to track a desired trajectory  $y_d(t)$  the feedback linearization design is very easy. However, in this example it is desired for  $y(t) \equiv x_2(t)$  to track  $y_d(t)$ . Perform the design and study the zero dynamics. Simulate the closed-loop system using MATLAB with initial conditions of  $x_1(0) = 1$ ,  $x_2(0) = 1$ ,  $x_3(0) = 1$ .

**Problem 2.4-3 : Effect of Modeling Uncertainties and Disturbances on Feedback Linearization Controller.** In the sample design of Subsection 2.4.1 and the computer simulation in Example 2.4.1 it was assumed that all the dynamics are exactly known and there are no disturbances. However, in practical situations there can be unmodelled dynamics or unknown disturbances. Suppose, therefore, that the controller was designed assuming the plant dynamics (2.4.2) but that the actual plant is given by

$$\begin{aligned}\dot{x}_1 &= x_1x_2 + x_3 + d(t) \\ \dot{x}_2 &= -2x_2 + x_1u \\ \dot{x}_3 &= \sin x_1 + cx_1x_2 + u\end{aligned}\tag{2.7.1}$$

where  $c$  is an inexactly known parameter and  $d(t)$  is a disturbance. Simulate the controller from Example 2.4.1 on this actual dynamics for: (a)  $d = 0, c = 1$ . (b)  $d = 0, c = 5$ . (c) The disturbance is the unit step,  $d(t) = u_{-1}(t) \equiv t, t \geq 0$ , and  $c = 2$ .

**Problem 2.4-4 : Integral Control Term for Disturbance Rejection.** The debilitating effects of disturbances can often be offset by including an integral term in the outer tracking loop. Consider the system (2.7.1) with  $c = 2$  and a unit step disturbance  $d(t) = u_{-1}(t)$ . Replace the PD outer tracking loop by a PID controller. Select suitable PID gains. Simulate the system using both PD and PID outer loops to compare the performance.

## Section 2.5

**Problem 2.5-1 : Lyapunov Stability Analysis.** Using Lyapunov techniques examine stability for the following systems. Plot time histories to substantiate your conclusions.

a.  $\begin{aligned}\dot{x}_1 &= x_1x_2^2 - x_1 \\ \dot{x}_2 &= -x_1^2x_2 - x_2.\end{aligned}$

b.

$$\begin{aligned}\dot{x}_1 &= x_2 \sin x_1 - x_1 \\ \dot{x}_2 &= -x_1 \sin x_1 - x_2.\end{aligned}$$

c.

$$\begin{aligned}\dot{x}_1 &= x_2 + x_1(x_1^2 - 2) \\ \dot{x}_2 &= -x_1\end{aligned}$$

**Problem 2.5-2 : Passivity and Lyapunov Functions.** Study the passivity of the following systems. Find the power dissipated  $g(t)$ .

a.

$$\begin{aligned}\dot{x}_1 &= x_1 x_2^2 - x_1 + u \\ \dot{x}_2 &= -x_1^2 x_2 - x_2 + u \\ y &= x_1 + x_2.\end{aligned}$$

b.

$$\begin{aligned}\dot{x}_1 &= x_2 \sin x_1 - x_1 + u \\ \dot{x}_2 &= -x_1 \sin x_1 - x_2 \\ y &= x_1.\end{aligned}$$

c.

$$\begin{aligned}\dot{x}_1 &= x_2 \sin x_1 - x_1 + u \\ \dot{x}_2 &= -x_1 \sin x_1 - x_2 \\ y &= x_2.\end{aligned}$$

**Problem 2.5-3 : Lyapunov Control Design.** Using Lyapunov techniques design controllers to stabilize the following systems. Plot time histories, both open-loop and closed-loop, to verify your designs. Verify that the closed-loop systems are passive. Check also for dissipativity.

a.

$$\begin{aligned}\dot{x}_1 &= x_1 x_2 \\ \dot{x}_2 &= x_1^2 - \sin x_1 + u.\end{aligned}$$

b.

$$\begin{aligned}\dot{x}_1 &= x_1 + (1 + x_2^2)u \\ \dot{x}_2 &= x_2 \sin x_1.\end{aligned}$$

c.

$$\begin{aligned}\dot{x}_1 &= x_1(x_1^2 - 2) \\ \dot{x}_2 &= \cos(x_1) + (2 + \sin(x_1))u.\end{aligned}$$

**Problem 2.5-4 : Barbalat's Lemma.** Using Barbalat's Lemma, show that the nonlinear spring-mass-damper in Example 2.5.3 is AS.

**Problem 2.5-5 : Bounded Stability Using Lyapunov Extensions.** Use the UUB extension to show that the van der Pol oscillator

$$\ddot{y} + \alpha(y^2 - \gamma)\dot{y} + y = u$$

is UUB. Find the radius of the region of boundedness. Simulate the system in

*MATLAB and plot phase-plane trajectories to verify the result.*

**Problem 2.5-6 : Stability of Limit Cycle.** Consider the system

$$\begin{aligned}\dot{x}_1 &= x_1x_2^2 - x_1(x_1^2 + x_2^2 - 3) \\ \dot{x}_2 &= -x_1^2x_2 - x_2(x_1^2 + x_2^2 - 3)\end{aligned}$$

which was shown to be UUB in Example 2.5.9. Select the Lyapunov function

$$L = (x_1^2 + x_2^2 - 3)^2$$

to demonstrate that this system has two equilibria: a stable limit cycle and an unstable equilibrium point at the origin. Simulate the system using MATLAB and make phase-plane plots for several initial conditions to convince yourself that, in this example, the limit cycle is the cause of the UUB nature of the stability.

**Problem 2.5-7 : Stability Improvement Using Feedback.** The system

$$\dot{x} = Ax + Bu + d$$

has a disturbance  $d(t)$  that is unknown but bounded so that  $\|d\| < d_M$ , with the bound  $d_M$  known. In Example 2.5.10 the system with no control input,  $B = 0$ , and  $A$  stable was shown to be UUB. Show that by selecting the control input as  $u(t) = -Kx(t)$  it is possible to improve the UUB stability properties of the system by making the bound on  $\|x\|$  smaller. In fact, if feedback is allowed, the initial system matrix  $A$  need not be stable as long as  $(A, B)$  is stabilizable.

# Chapter 3

# Robot Dynamics and Control

*This chapter deals with the real-time motion control of robot manipulators. Robot manipulators have complex nonlinear dynamics that might make accurate and robust control difficult. Fortunately, robots are in the class of Lagrangian dynamical systems, so that they have several extremely nice physical properties that make their control straightforward. In this chapter will be discussed several control techniques including computed-torque (e.g. feedback linearization), classical joint control, and digital control. A framework for the tracking control problem based on approximation of unknown nonlinear functions is provided that can be used to derive a broad family of controllers including adaptive, robust, and learning controllers. This approximation-based approach is used to design neural network controllers in the remainder of the book. More information may be found in Lewis, Abdallah, and Dawson (1993); Lewis, Fitzgerald, and Liu (1997). The advances that made possible this modern approach to robot control were made by Craig (1985), Slotine and Li (1991) Slotine (1985, 1988), Spong and Ortega (Spong et al. 1987, Spong and Vidyasagar 1989), and others.*

### 3.0.1 Commercial Robot Controllers

*Commercial robot controllers are specialized multiprocessor computing systems that provide four basic processes allowing integration of the robot into an automation system: motion trajectory generation and following, motion/process integration and sequencing, human user integration, and information integration.*

**Motion Trajectory Generation and Following.** *There are two important controller-related aspects of industrial robot motion generation. One is the extent of manipulation that can be programmed, the other is the ability to execute controlled programmed motion. A unique aspect of each robot system is its real-time servo-level motion control. The details of real-time control are typically not revealed to the user due to safety and proprietary information secrecy reasons. Each robot controller, through its operating system programs, converts digital data from higher level coordinators into coordinated arm motion through precise computation and high-speed distribution and communication of the individual axis motion commands which are executed by individual joint servo-controllers. Most commercial robot controllers*

operate at a sample period of 16 msec. The real-time motion controller invariably uses classical independent-joint proportional-integral-derivative (PID) control or simple modifications of PID. This makes commercially available controllers suitable for point-to-point motion, but most are not suitable for following continuous position/velocity profiles or exerting prescribed forces without considerable programming effort, if at all.

**Motion/Process Integration and Sequencing.** Motion/process integration involves coordinating manipulator motion with process sensors or other process controller devices. The most primitive process integration is through discrete digital input/output (i/o). For example a machine controller external to the robot controller might send a one-bit signal indicating that it is ready to be loaded by the robot. The robot controller must have the ability to read the signal and to perform logical operations (if then, wait until, do until, etc.) using the signal. Coordination with sensors (e.g. vision) is also often provided.

**Human Integration.** The controller's human interfaces are critical to the expeditious setup and programming of robot systems. Most robot controllers have two types of human interface available: computer style CRT/keyboard terminals for writing and editing program code off-line, and teach pendants, which are portable manual input terminals used to command motion in a telerobotic fashion via touch keys or joy sticks. Teach pendants are usually the most efficient means available for positioning the robot, and a memory in the controller makes it possible to play back the taught positions to execute motion trajectories. With practice, human operators can quickly teach a series of points which are chained together in playback mode. Most robot applications currently depend on the integration of human expertise during the programming phase for the successful planning and coordination of robot motion. These interface mechanisms are effective in unobstructed workspaces where no changes occur between programming and execution. They do not allow human interface during execution or adaptation to changing environments.

**Information Integration.** Information integration is becoming more important as the trend toward increasing flexibility and agility impacts robotics. Many commercial robot controllers now support information integration functions by employing integrated PC interfaces through the communications ports (e.g. RS-232), or through direct connections to the robot controller data bus.

### 3.1 KINEMATICS AND JACOBIANS

The primary focus of this chapter is on the dynamics of robot manipulators and their properties and control. However, it is usually important in robot control applications to have an appreciation of manipulator kinematic and Jacobian transformations. Therefore, these topics are discussed in this section.

### 3.1.1 Kinematics of Rigid Serial-Link Manipulators

The kinematics of the robot manipulator are concerned only with relative positioning and not with motion effects. The key notions involve position coordinate changes and include the link transformation  $A$  matrices, the arm  $T$  matrix, joint space versus Cartesian space coordinates, and kinematics versus inverse kinematics.

**Link A Matrices.** Robot manipulator geometries fall into five basic classes, illustrated in Fig. 3.1.1. Particularly useful for assembly and pick-and-place operations is the selected compliant articulated robot for assembly (SCARA) arm. Fixed-base serial-link rigid robot manipulators can be considered as a sequence of joints held together by links. Each joint  $i$  has a joint variable  $q_i$ , which is an angle (e.g.  $\theta_i$ , units of degrees) for revolute ( $R$ ) joints and a length (e.g.  $d_i$ , units of length) for prismatic ( $P$ ) or extensible joints. The joint vector of an  $n$ -link robot is defined as  $q = [q_1 \ q_2 \dots \ q_n]^T \in \mathbb{R}^n$ ; the joints are traditionally numbered from the base to the end-effector, with link 0 being the fixed base. For example, in Fig. 3.1.1d, the joint vector is  $q = [\theta \ h \ r]^T$ , with  $n = 3$ . A robot with  $n$  joints has  $n$  degrees of freedom, so that for complete freedom of positioning and orientation in our 3-D space  $\mathbb{R}^3$  one needs a six-link arm.

For analysis purposes, it is considered that a coordinate frame is affixed to each link. The base frame is attached to the manipulator base, link 0. The location of the coordinate frame on the link is often selected according to the Denavit-Hartenberg (DH) convention (Lewis, Abdallah, and Dawson 1993). The relation between the links, shown in Fig. 3.1.2, is given by the  $A$  matrix for link  $i$ , which has the form

$$A_i(q_i) = \begin{bmatrix} R_i & p_i \\ 0 & 1 \end{bmatrix}, \quad (3.1.1)$$

where  $R_i(q_i)$  is a  $3 \times 3$  rotation matrix ( $R_i^{-1} = R_i^T$ ) and  $p_i(q_i) = [x_i \ y_i \ z_i]^T \in \mathbb{R}^3$  is a translation vector.  $R_i$  specifies the rotation of the coordinate frame on link  $i$  with respect to the coordinate frame on link  $i-1$ ;  $p_i$  specifies the translation of the coordinate frame on link  $i$  with respect to the coordinate frame on link  $i-1$ . The  $4 \times 4$  homogeneous transformation  $A_i$  thus specifies completely the orientation and translation of link  $i$  with respect to link  $i-1$ .

The  $A$  matrix  $A_i(q_i)$  is a function of the joint variable, so that as  $q_i$  changes with robot motion,  $A_i$  changes correspondingly.  $A_i$  is also dependent on the parameters link twist and link length, which are fixed for each link. The  $A$  matrices are often given for a specific robot in the manufacturer's handbook.

**Robot T Matrix.** The position of the end-effector is given in terms of the base coordinate frame by the arm  $T$  matrix defined as the concatenation of  $A$  matrices

$$T(q) = A_1(q_1)A_2(q_2)\dots A_n(q_n) \equiv \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix}. \quad (3.1.2)$$

This  $4 \times 4$  homogeneous transformation matrix is a function of the joint variable vector  $q$ . The  $3 \times 3$  cumulative rotation matrix is given by  $R(q) = R_1(q_1)R_2(q_2)\dots R_n(q_n)$ .

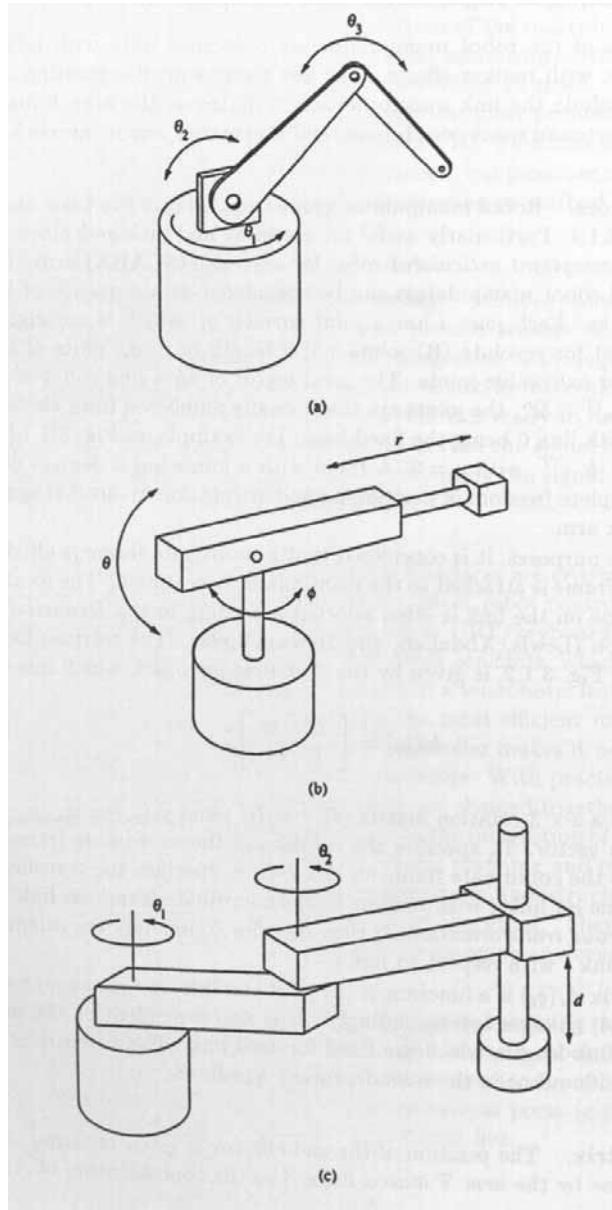


Figure 3.1.1: Basic robot arm geometries. (a) Articulated arm, revolute coordinates (RRR). (b) Spherical coordinates (RRP). (c) SCARA arm (RRP).

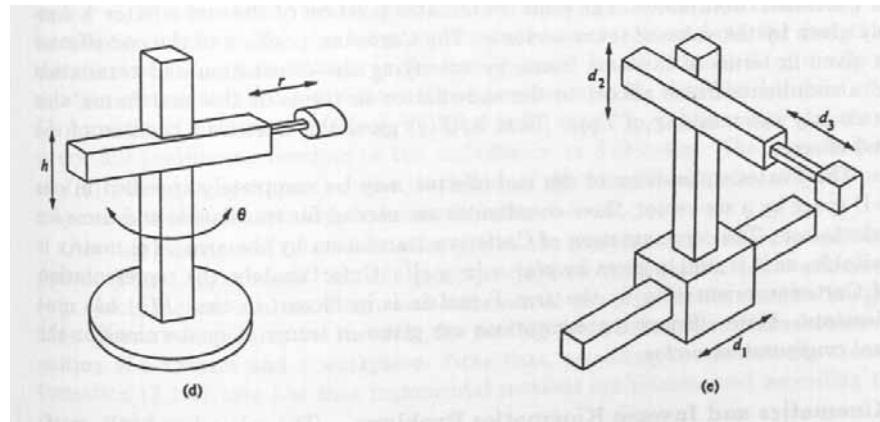
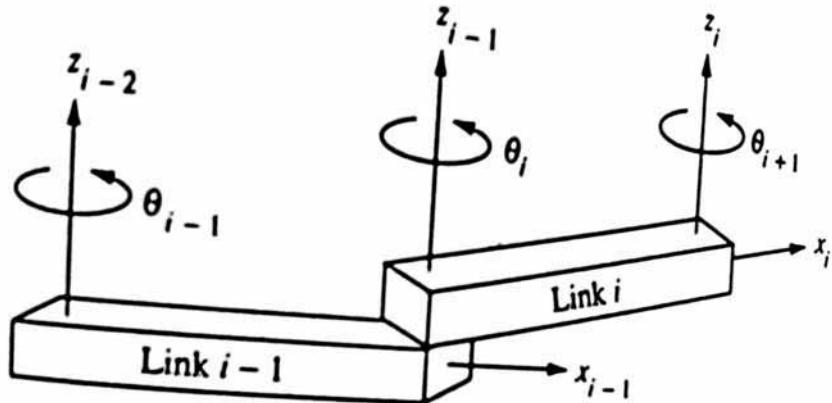


Figure 3.1.1: Basic robot arm geometries (cont'd.). (d) Cylindrical coordinates (RPP). (e) Cartesian arm, rectangular coordinates (PPP).



Scanned by CamScanner

Figure 3.1.2: Denavit-Hartenberg coordinate frames in a serial-link manipulator.

**Joint Space Versus Cartesian Space.** An  $n$ -link manipulator has  $n$  degrees of freedom, and the position of the end-effector is completely fixed once the joint variables  $q_i$  are prescribed. This position may be described either in joint coordinates or in Cartesian coordinates. The joint coordinates position of the end-effector is simply given by the value of the  $n$ -vector  $q$ . The Cartesian position of the end-effector is given in terms of the base frame by specifying the orientation and translation of a coordinate frame affixed to the end-effector in terms of the base frame; this is exactly the meaning of  $T(q)$ . That is,  $T(q)$  gives the Cartesian position of the end-effector.

The Cartesian position of the end-effector may be completely specified in our 3-D space by a six-vector; three coordinates are needed for translation and three for orientation. The representation of Cartesian translation by the arm  $T(q)$  matrix is suitable, as it is simply given by  $p(q) = [x \ y \ z]^T$ . Unfortunately, the representation of Cartesian orientation by the arm  $T$  matrix is inefficient in that  $R(q)$  has nine elements. More efficient representations are given in terms of quaternions or the tool configuration vector.

**Kinematics and Inverse Kinematics Problems.** The robot kinematics problem is to determine the Cartesian position of the end-effector once the joint variables are given. This is accomplished simply by computing  $T(q)$  for a given value of  $q$ .

The inverse kinematics problem is to determine the required joint angles  $q_i$  to position the end-effector at a prescribed Cartesian position. This corresponds to solving (3.1.2) for  $q \in \mathbb{R}^n$  given a desired orientation  $R$  and translation  $p$  of the end-effector. This is not an easy problem, and may have more than one solution (e.g. think of picking up a coffee cup— one may reach with elbow up, elbow down, etc.). There are various efficient techniques for accomplishing this. One should avoid the functions  $\arcsin$ ,  $\arccos$  and use where possible the numerically well-conditioned  $\arctan$  function.

### 3.1.2 Robot Jacobians

Kinematics transformations deal with conversion of positions between various coordinate frames. Jacobians allow the transformation of dynamical quantities including velocities, accelerations, and forces.

**Transformation of Velocity and Acceleration.** When the manipulator moves, the joint variable becomes a function of time  $t$ . Suppose there is prescribed a generally nonlinear transformation from the joint variable  $q(t) \in \mathbb{R}^n$  to another variable  $y(t) \in \mathbb{R}^p$  given by

$$y(t) = h(q(t)). \quad (3.1.3)$$

An example is provided by the equation  $y = T(q)$ , where  $y(t)$  is the Cartesian position. Taking partial derivatives one obtains

$$\dot{y} = \frac{\partial h}{\partial q} \dot{q} \equiv J(q)\dot{q}, \quad (3.1.4)$$

where  $J(q)$  is the Jacobian associated with  $h(q)$ . This equation tells how the joint velocities  $\dot{q}$  are transformed to the velocity  $\dot{y}$ .

If  $y = T(q)$  the Cartesian end-effector position, then the associated Jacobian  $J(q) = \frac{\partial T(q)}{\partial q}$  is known as the manipulator Jacobian. There are several techniques for efficiently computing this particular Jacobian. Note that  $\dot{y} = [v^T \ \omega^T]^T \in \Re^6$ , with  $v \in \Re^3$  the linear velocity and  $\omega \in \Re^3$  the angular velocity. Therefore, in formally computing  $J(q)$  there are some complications arising from the fact that the representation of orientation in the homogeneous transformation  $T(q)$  is a  $3 \times 3$  rotation matrix and not a 3-vector. If the arm has  $n$  links, then the Jacobian is a  $6 \times n$  matrix; if  $n$  is less than 6 (e.g. SCARA arm), then  $J(q)$  is not square and there is not full positioning freedom of the end-effector in 3-D space. The singularities of  $J(q)$  (where it loses rank) define the limits of the robot workspace; singularities may occur within the workspace for some arms.

Another example of interest is when  $y(t)$  is the position in a camera coordinate frame. Then  $J(q)$  reveals the relationships between manipulator joint velocities (e.g. joint incremental motions) and incremental motions in the camera image. This affords a technique, for instance, for moving the arm to cause desired relative motion of a camera and a workpiece. Note that, according to the velocity transformation (3.1.4), one has that incremental motions are transformed according to  $\Delta y = J(q)\Delta q$ .

Differentiating (3.1.4) one obtains the acceleration transformation

$$\ddot{y} = J\ddot{q} + \dot{J}\dot{q}. \quad (3.1.5)$$

**Force Transformation.** Using the notion of virtual work, it can be shown that forces in terms of  $q$  may be transformed to forces in terms of  $y$  using

$$\tau = J^T(q)F, \quad (3.1.6)$$

where  $\tau(t)$  is the force in joint space (given as an  $n$ -vector of torques for a revolute robot), and  $F$  is the force vector in  $y$  space. If  $y$  is the Cartesian position, then  $F$  is a vector of three forces  $[f_x \ f_y \ f_z]^T$  and three torques  $[\tau_x \ \tau_y \ \tau_z]^T$ . When  $J(q)$  loses rank, the arm cannot exert forces in all directions that may be specified.

## 3.2 ROBOT DYNAMICS AND PROPERTIES

Robot dynamics considers motion effects due to the control inputs and inertias, Coriolis forces, gravity, disturbances, and other effects. It reveals the relation between the control inputs and the joint variable motion  $q(t)$ , which is required for the purpose of servo-control system design. A robot manipulator can have either revolute joints or prismatic joints. The values of the angles, for revolute joints, and link lengths, for prismatic joints, are called the link variables and are denoted  $q_1(t), q_2(t), \dots, q_n(t)$  for joints one, two, and so on. The number of links is denoted  $n$ ; for complete freedom of motion in space, six degrees of freedom are needed, three for positioning, and three for orientation. Thus, many commercial robots have 6 links. We discuss here robots which are rigid, that is which have no flexibility in the links or in the gearing of the joints; flexible robots are discussed in Chapter 5.

### 3.2.1 Joint Space Dynamics and Properties

The dynamics of robot manipulators with rigid links can be written as

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = \tau, \quad (3.2.1)$$

where  $M(q)$  is the inertia matrix,  $V_m(q, \dot{q})$  is the Coriolis/centripetal matrix,  $F(\dot{q})$  are the friction terms,  $G(q)$  is the gravity vector, and  $\tau_d(t)$  represents disturbances. The control input vector  $\tau(t)$  has components of torque for revolute joints and force for prismatic joints. It is often convenient to write the robot dynamics as

$$M(q)\ddot{q} + N(q, \dot{q}) + \tau_d = \tau, \quad (3.2.2)$$

where

$$N(q, \dot{q}) \equiv V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) \quad (3.2.3)$$

represents a vector of the nonlinear terms.

The dynamics can be computed for a specific manipulator from the link A matrices and link inertias (Lewis, Abdallah, and Dawson 1993). Alternatively, they can be derived from first principles using Lagrange's equation of motion

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = \tau, \quad (3.2.4)$$

with the Lagrangian defined in terms of the kinetic energy  $K$  and the potential energy  $P$  as

$$L = K - P. \quad (3.2.5)$$

The use of Lagrange's equation in deriving the manipulator dynamics is illustrated in Example 3.2.1.

The objective of robot control is generally to select the control torques  $\tau(t)$  so that the robot follows a prescribed desired motion trajectory or exerts a desired force. Examples include spray painting, grinding, or manufacturing assembly operations. The position control objective can be achieved by first defining a desired trajectory  $q_d(t)$ , which is a vector containing the desired values versus time  $q_{d_i}(t)$  of each joint of the manipulator. This desired trajectory vector  $q_d(t)$  is determined in a higher-level path planner, based on an even higher-level task decomposition, and then fed to the real-time motion control system (Lewis, Fitzgerald, and Liu 1997). This chapter discusses the real-time motion control problem assuming that  $q_d(t)$  is given.

The robot dynamics (3.2.1) satisfy some important physical properties as a consequence of the fact that they are a Lagrangian system. These properties significantly simplify the robot control problem. The main properties of which one should be aware are given in Table 3.2.1. The bounding properties are especially useful in robust control approaches. The skew-symmetry property P3 is vital for Lyapunov control proofs which provide guaranteed tracking motion and often give the structure of the control loops. It essentially allows some very nice linear systems techniques to be used with the time-varying robot dynamics.

The selection of  $V_m(q, \dot{q})$  in (3.2.1) is not unique. In the robot control applications given in this book it is necessary to select it so that property P3 holds. The selection of  $V_m(q, \dot{q})$ , the bounds, and the skew-symmetric matrix  $S(q, \dot{q})$  for a representative robot arm are discussed in Example 3.2.2.

Table 3.2.1: Properties of Robot Arm Dynamics

- 
- P1** The inertia matrix  $M(q)$  is symmetric, positive definite, and bounded so that  $\mu_1 I \leq M(q) \leq \mu_2 I$  for all  $q(t)$ . For revolute joints, the only occurrences of the joint variables  $q_i$  are as  $\sin(q_i), \cos(q_i)$ . For arms with no prismatic joints, the bounds  $\mu_1, \mu_2$  are constants.
- P2** The Coriolis/centripetal vector  $V_m(q, \dot{q})\dot{q}$  is quadratic in  $\dot{q}$ .  $V_m$  is bounded so that  $\|V_m\| \leq v_B\|\dot{q}\|$ , or equivalently  $\|V_m\dot{q}\| \leq v_B\|\dot{q}\|^2$ .
- P3** The Coriolis/centripetal matrix can always be selected so that the matrix  $S(q, \dot{q}) \equiv \dot{M}(q) - 2V_m(q, \dot{q})$  is *skew symmetric*. Therefore,  $x^T S x = 0$  for all vectors  $x$ . This is a statement of the fact that the fictitious forces in the robot system do no work.
- P4** The friction terms have the approximate form

$$F(\dot{q}) = F_v \dot{q} + F_d(\dot{q}),$$

with  $F_v$  a diagonal matrix of constant coefficients representing the viscous friction, and  $F_d(\cdot)$  a vector with entries like  $K_{d_i} \text{sgn}(\dot{q}_i)$ , with  $\text{sgn}(\cdot)$  the signum function and  $K_{d_i}$  the coefficients of dynamic friction. These friction terms are bounded so that  $\|F(\dot{q})\| \leq f_B\|\dot{q}\| + k_B$  for constants  $f_B, k_B$ .

- P5** The gravity vector is bounded so that  $\|G(q)\| \leq g_B$ . For revolute joints, the only occurrences of the joint variables  $q_i$  are as  $\sin(q_i), \cos(q_i)$ . For revolute joint arms the bound  $g_B$  is a constant.
- P6** The disturbances are bounded so that  $\|\tau_d(t)\| \leq d_B$ .
- 

**Example 3.2.1 (Dynamics of 2-Link Planar Elbow Arm) :**

A 2-link planar RR robot arm used extensively for simulation in the literature (Lewis, Abdallah, and Dawson 1993) is shown in Fig. 3.2.1. This arm is simple enough to simulate yet has all the nonlinear effects common to general robot manipulators. This example shows how to derive the dynamics of a robot arm using Lagrange's equation of motion.

To determine the arm dynamics, examine Fig. 3.2.1 where we have assumed that the link masses are concentrated at the ends of the links. The joint variable is  $q = [q_1 \ q_2]^T$  and the generalized force vector is  $\tau = [\tau_1 \ \tau_2]^T$ , with  $\tau_1, \tau_2$  the torques supplied by the actuators.

**a. Kinetic and Potential Energy .**

To determine the dynamics of the 2-link arm using Lagrange's equation (3.2.4), one must first find the kinetic and potential energies. For link 1, the kinetic and potential energies are

$$K_1 = \frac{1}{2}m_1 a_1^2 \dot{q}_1^2$$

$$P_1 = m_1 g a_1 \sin q_1.$$

For link 2, we have the positions and velocities

$$x_2 = a_1 \cos q_1 + a_2 \cos(q_1 + q_2)$$

$$y_2 = a_1 \sin q_1 + a_2 \sin(q_1 + q_2)$$

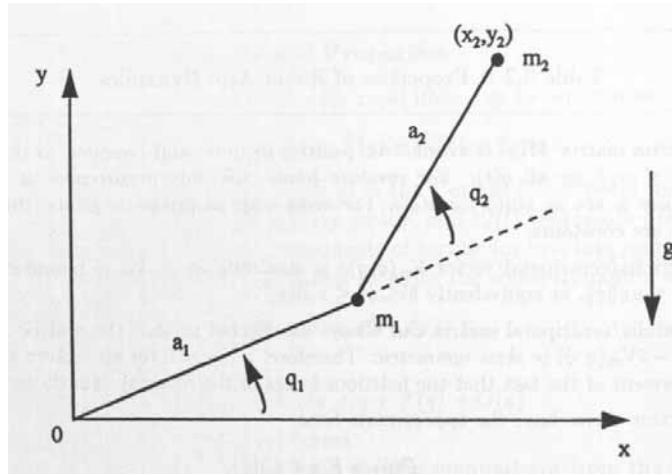


Figure 3.2.1: Two-link planar robot arm.

$$\dot{x}_2 = -a_1 \dot{q}_1 \sin q_1 - a_2(\dot{q}_1 + \dot{q}_2) \sin(q_1 + q_2)$$

$$\dot{y}_2 = a_1 \dot{q}_1 \cos q_1 + a_2(\dot{q}_1 + \dot{q}_2) \cos(q_1 + q_2)$$

so that the velocity squared is

$$v_2^2 = \dot{x}_2^2 + \dot{y}_2^2 = a_1^2 \dot{q}_1^2 + a_2^2(\dot{q}_1 + \dot{q}_2)^2 + 2a_1 a_2 (\dot{q}_1^2 + \dot{q}_1 \dot{q}_2) \cos q_2.$$

Therefore, the kinetic energy for link 2 is

$$K_2 = \frac{1}{2} m_2 v_2^2 = \frac{1}{2} m_2 a_1^2 \dot{q}_1^2 + \frac{1}{2} m_2 a_2^2 (\dot{q}_1 + \dot{q}_2)^2 + m_2 a_1 a_2 (\dot{q}_1^2 + \dot{q}_1 \dot{q}_2) \cos q_2.$$

The potential energy for link 2 is

$$P_2 = m_2 g y_2 = m_2 g [a_1 \sin q_1 + a_2 \sin(q_1 + q_2)].$$

### b. Lagrange's Equation .

The Lagrangian for the entire arm is

$$\begin{aligned} L &= K - P = K_1 + K_2 - P_1 - P_2 \\ &= \frac{1}{2}(m_1 + m_2)a_1^2 \dot{q}_1^2 + \frac{1}{2}m_2 a_2^2 (\dot{q}_1 + \dot{q}_2)^2 + m_2 a_1 a_2 (\dot{q}_1^2 + \dot{q}_1 \dot{q}_2) \cos q_2 \\ &\quad -(m_1 + m_2)g a_1 \sin q_1 - m_2 g a_2 \sin(q_1 + q_2). \end{aligned}$$

Equation (3.2.4) is a vector equation comprised of  $n = 2$  scalar equations. The individual terms needed to write down these  $n$  equations are

$$\begin{aligned} \frac{\partial L}{\partial \dot{q}_1} &= (m_1 + m_2)a_1^2 \dot{q}_1 + m_2 a_2^2 (\dot{q}_1 + \dot{q}_2) + m_2 a_1 a_2 (2\dot{q}_1 + \dot{q}_2) \cos q_2 \\ \frac{d}{dt} \frac{\partial L}{\partial \dot{q}_1} &= (m_1 + m_2)a_1^2 \ddot{q}_1 + m_2 a_2^2 (\ddot{q}_1 + \ddot{q}_2) + m_2 a_1 a_2 (2\ddot{q}_1 + \ddot{q}_2) \cos q_2 \\ &\quad - m_2 a_1 a_2 (2\dot{q}_1 \dot{q}_2 + \dot{q}_2^2) \sin q_2 \end{aligned}$$

$$\begin{aligned}
\frac{\partial L}{\partial q_1} &= -(m_1 + m_2)ga_1 \cos q_1 - m_2ga_2 \cos(q_1 + q_2) \\
\frac{\partial L}{\partial \dot{q}_2} &= m_2a_2^2(\dot{q}_1 + \dot{q}_2) + m_2a_1a_2\dot{q}_1 \cos q_2 \\
\frac{d}{dt} \frac{\partial L}{\partial \dot{q}_2} &= m_2a_2^2(\ddot{q}_1 + \ddot{q}_2) + m_2a_1a_2\ddot{q}_1 \cos q_2 - m_2a_1a_2\dot{q}_1\dot{q}_2 \sin q_2 \\
\frac{\partial L}{\partial q_2} &= -m_2a_1a_2(\dot{q}_1^2 + \dot{q}_1\dot{q}_2) \sin q_2 - m_2ga_2 \cos(q_1 + q_2).
\end{aligned}$$

Finally, according to Lagrange's equation, the arm dynamics are given by the two coupled nonlinear differential equations

$$\begin{aligned}
\tau_1 &= [(m_1 + m_2)a_1^2 + m_2a_2^2 + 2m_2a_1a_2 \cos q_2]\ddot{q}_1 + [m_2a_2^2 + m_2a_1a_2 \cos q_2]\ddot{q}_2 \\
&\quad - m_2a_1a_2(2\dot{q}_1\dot{q}_2 + \dot{q}_2^2) \sin q_2 + (m_1 + m_2)ga_1 \cos q_1 + m_2ga_2 \cos(q_1 + q_2) \\
\tau_2 &= [m_2a_2^2 + m_2a_1a_2 \cos q_2]\ddot{q}_1 + m_2a_2^2\ddot{q}_2 + m_2a_1a_2\dot{q}_1^2 \sin q_2 + m_2ga_2 \cos(q_1 + q_2).
\end{aligned}$$

### c. Manipulator Dynamics .

Writing the arm dynamics in vector form yields

$$\begin{aligned}
&\begin{bmatrix} (m_1 + m_2)a_1^2 + m_2a_2^2 + 2m_2a_1a_2 \cos q_2 & m_2a_2^2 + m_2a_1a_2 \cos q_2 \\ m_2a_2^2 + m_2a_1a_2 \cos q_2 & m_2a_2^2 \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} \\
&+ \begin{bmatrix} -m_2a_1a_2(2\dot{q}_1\dot{q}_2 + \dot{q}_2^2) \sin q_2 \\ m_2a_1a_2\dot{q}_1^2 \sin q_2 \end{bmatrix} + \begin{bmatrix} (m_1 + m_2)ga_1 \cos q_1 + m_2ga_2 \cos(q_1 + q_2) \\ m_2ga_2 \cos(q_1 + q_2) \end{bmatrix} \\
&= \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}.
\end{aligned}$$

These manipulator dynamics are in the standard form

$$M(q)\ddot{q} + V(q, \dot{q}) + G(q) = \tau$$

with  $M(q)$  the inertia matrix,  $V(q, \dot{q})$  the Coriolis/centripetal vector, and  $G(q)$  the gravity vector. Note that  $M(q)$  is symmetric. Friction terms may be added of the form in Property P4 in Table 3.2.1.

One may write the dynamics of the 2-link arm compactly as

$$\begin{aligned}
&\begin{bmatrix} \alpha + \beta + 2\eta \cos q_2 & \beta + \eta \cos q_2 \\ \beta + \eta \cos q_2 & \beta \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} \\
&+ \begin{bmatrix} -\eta(2\dot{q}_1\dot{q}_2 + \dot{q}_2^2) \sin q_2 \\ \eta\dot{q}_1^2 \sin q_2 \end{bmatrix} + \begin{bmatrix} \alpha e_1 \cos q_1 + \eta e_1 \cos(q_1 + q_2) \\ \eta e_1 \cos(q_1 + q_2) \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}
\end{aligned}$$

where  $\alpha = (m_1 + m_2)a_1^2$ ,  $\beta = m_2a_2^2$ ,  $\eta = m_2a_1a_2$ ,  $e_1 = g/a_1$ .

The bounds and the skew-symmetric matrix in properties P1-P6 for the 2-link planar robot arm are given in Example 3.2.2.  $\square$

#### **Example 3.2.2 (Exercise — Bounds and Coriolis Matrices) :**

The choice of the Coriolis/centripetal matrix  $V_m(q, \dot{q})$  in (3.2.1) is not unique. Show that the correct selection for property P3 to hold for the 2-link planar arm is

$$V_m(q, \dot{q}) = \begin{bmatrix} -\dot{q}_2 m_2 a_1 a_2 \sin q_2 & -(\dot{q}_1 + \dot{q}_2) m_2 a_1 a_2 \sin q_2 \\ \dot{q}_1 m_2 a_1 a_2 \sin q_2 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} -\eta\dot{q}_2 \sin q_2 & -\eta(\dot{q}_1 + \dot{q}_2) \sin q_2 \\ \eta\dot{q}_1 \sin q_2 & 0 \end{bmatrix}$$

with  $\eta = m_2 a_1 a_2$ . Show that the skew-symmetric matrix is then given by

$$\begin{aligned} S(q, \dot{q}) &= \begin{bmatrix} 0 & (2\dot{q}_1 + \dot{q}_2)m_2 a_1 a_2 \sin q_2 \\ -(2\dot{q}_1 + \dot{q}_2)m_2 a_1 a_2 \sin q_2 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & \eta(2\dot{q}_1 + \dot{q}_2) \sin q_2 \\ -\eta(2\dot{q}_1 + \dot{q}_2) \sin q_2 & 0 \end{bmatrix}. \end{aligned}$$

The bounds on the various dynamics terms discussed in properties P1-P6 depend on the norm selected (Chapter 2 discusses norms). The correct selection of norm can simplify the computations. Show that the bounds in properties P1-P6 are given for the 2-link planar arm of Example 3.2.1 in terms of the 1-norm by the following.

Inertia matrix lower and upper bounds:

$$\begin{aligned} \mu_1 &= (m_1 + m_2)a_1^2 + 2m_2a_2^2 \equiv \alpha + 2\beta \\ \mu_2 &= (m_1 + m_2)a_1^2 + 2m_2a_2^2 + 3m_2a_1a_2 \equiv \alpha + 2\beta + 3\eta. \end{aligned}$$

Coriolis bound:

$$v_B = m_2 a_1 a_2 \equiv \eta.$$

Gravity bound:

$$g_B = (m_1 + m_2)ga_1 + 2m_2ga_2 \equiv \alpha e_1 + 2\eta e_1,$$

where  $\alpha = (m_1 + m_2)a_1^2$ ,  $\beta = m_2a_2^2$ ,  $\eta = m_2a_1a_2$ ,  $e_1 = g/a_1$ .  $\square$

### 3.2.2 State Variable Representations

*The nonlinear state-variable representation  $\dot{x} = f(x, u)$ , with  $x(t)$  the internal state and  $u(t)$  the control input, is very convenient for many applications, including the derivation of suitable control laws and computer simulation. Once the system has been put into state-space form, it can easily be integrated to obtain simulation time plots using, for instance, a Runge-Kutta integrator; many standard software packages have such integration routines, including MATLAB, MATRIXx, and SIMNON. Computer simulation of nonlinear state-space systems was discussed in Chapter 2. Simulation of robot controllers is discussed in Section 3.3.2.*

*It is supposed for convenience in this section that the disturbance  $\tau_d(t)$  is equal to zero. There are three convenient state-space formulations for the robot dynamics (3.2.1). In the position/velocity state-space form, one defines the state as the 2n-vector  $x \equiv [q^T \quad \dot{q}^T]^T$  and writes*

$$\dot{x} = \begin{bmatrix} \dot{q} \\ -M^{-1}(q)N(q, \dot{q}) \end{bmatrix} + \begin{bmatrix} 0 \\ M^{-1}(q) \end{bmatrix} \tau, \quad (3.2.6)$$

*which is in state-space form with  $u(t) \equiv \tau(t)$ .*

*An alternative linear state-space equation in the form  $\dot{x} = Ax + Bu$  can be defined as*

$$\dot{x} = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ I \end{bmatrix} u, \quad (3.2.7)$$

*with  $u(t) \equiv -M^{-1}(q)N(q, \dot{q}) + M^{-1}(q)\tau$ . This is known as the Brunovsky canonical form.*

The third state-space formulation is the Hamiltonian form, which derives from Hamilton's equations of motion. Here, the state is defined as the  $2n$ -vector  $x = [q^T \ p^T]^T$ , with  $p(t) \equiv M(q)\dot{q}$  the generalized momentum. Then, the state-space equation is

$$\dot{x} = \begin{bmatrix} M^{-1}(q)p \\ -\frac{1}{2}(I_n \otimes p^T) \frac{\partial M^{-1}(q)}{\partial q} p \end{bmatrix} + \begin{bmatrix} 0 \\ I_n \end{bmatrix} u, \quad (3.2.8)$$

with the control input defined by  $u = \tau - G(q)$  and  $\otimes$  the Kronecker product (Lewis, Abdallah, and Dawson, 1993).

### 3.2.3 Cartesian Dynamics and Actuator Dynamics

**Cartesian Dynamics.** The dynamics (3.2.1) are known as the joint-space dynamics as they are expressed in the joint-space coordinates  $q$ . Cartesian coordinates referred to some frame, often the base of the robot manipulator, may be used to describe the position of the end-effector of the robot arm. Denote the Cartesian coordinates of the end of the arm as  $y(t) = h(q)$ , whose first three coordinates represent position and last coordinates represent orientation. The nonlinear function  $h(q)$  gives the end-effector Cartesian coordinates in terms of the current joint positions  $q$  and is called the arm kinematics transformation. The arm Jacobian relates joint and Cartesian velocities and is defined as  $J(q) \equiv \frac{\partial h(q)}{\partial q}$  so that

$$\begin{bmatrix} v \\ \omega \end{bmatrix} \equiv \dot{y} = J(q)\dot{q} \quad (3.2.9)$$

where  $v(t)$  is the linear velocity and  $\omega(t)$  the angular velocity of the end-effector. Both these velocities are 3-vectors. Differentiating this equation gives the acceleration transformation  $\ddot{y} = J\ddot{q} + \dot{J}\dot{q}$ .

By substituting these expressions in (3.2.1) one discovers that the dynamics may be written in Cartesian form as

$$\bar{M}\ddot{y} + \bar{N} + f_d = F, \quad (3.2.10)$$

where  $\bar{M} \equiv J^{-T}M J^{-1}$ ,  $\bar{N} \equiv J^{-T}(N - MJ^{-1}\dot{J}J^{-1}\dot{y})$ , and the disturbance is  $f_d \equiv J^{-T}\tau_d$ . In the Cartesian dynamics, the control input is  $F$ , which has three components of force and three of torque.

The important conclusion of this discussion is that the Cartesian dynamics are of the same form as (3.2.2). Furthermore, it can be shown that the properties in Table 3.2.1 also hold in Cartesian form. Therefore, all the control techniques to be described in this book can be used for either the joint-space or the Cartesian dynamics.

**Actuator Dynamics.** The robot manipulator is driven by actuators which may be electric, hydraulic, pneumatic, and so on. Considering the case of electric motors it is straightforward to show that, if the armature inductance is negligible, the dynamics of the arm plus actuators can be written as

$$(J_M + R^2M)\ddot{q} + (B_M + R^2V_m)\dot{q} + (RF_M + R^2F) + R^2G = RK_Mv, \quad (3.2.11)$$

where the robot arm dynamics are described by  $M(q)$ ,  $V_m(q, \dot{q})$ ,  $F(\dot{q})$ ,  $G(q)$ , and  $J_M$  is the motor inertia,  $B_M$  is given by the rotor damping constant and back emf, and  $R$  has diagonal elements containing the gear ratios of the motor/joint couplings. The control input is the motor voltage  $v(t)$ , with  $K_M$  the diagonal matrix of motor torque constants.

The important conclusion is that the dynamics of the arm-plus-actuators has the same form as the dynamics (3.2.1) and can be shown to enjoy all the properties in Table 3.2.1. Therefore, the control methods to be described in this book apply to this composite system as well. Similar comments hold for other sorts of actuators such as hydraulic. If the armature inductances of the electric motors are not negligible, then the arm-plus-actuators have a coupled form of dynamics such as those discussed in Chapter 5. Special controls techniques must then be used.

### 3.3 COMPUTED-TORQUE (CT) CONTROL AND COMPUTER SIMULATION

In this section we introduce a variety of controllers for robot manipulators; these are collected for easy reference in Table 3.3.1. It is shown how to simulate these control algorithms on a digital computer, and examples are given to examine the sort of performance to be expected from the various controllers. The computed-torque (CT) method provides a unifying point of view for many control schemes, including PD-gravity control, classical independent joint control, and digital control. For many years during the 1960s and 1970s the major techniques for robot dynamics control were based on CT. Recently, advanced mathematical techniques based on feedback linearization have been derived. For the rigid-link arms, these are equivalent.

It is assumed that the desired motion trajectory for the manipulator  $q_d(t)$ , as determined for instance by a path planner, is prescribed. To use feedback linearization controls design, define the tracking error as

$$e(t) = q_d(t) - q(t) \quad (3.3.1)$$

and differentiate twice to see that the Brunovsky canonical form (3.2.7) can be written in terms of the state  $x = [e^T \quad \dot{e}^T]^T$  as

$$\frac{d}{dt} \begin{bmatrix} e \\ \dot{e} \end{bmatrix} = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix} \begin{bmatrix} e \\ \dot{e} \end{bmatrix} + \begin{bmatrix} 0 \\ I \end{bmatrix} u, \quad (3.3.2)$$

with

$$u \equiv \ddot{q}_d + M^{-1}(q)(N(q, \dot{q}) - \tau). \quad (3.3.3)$$

#### 3.3.1 Computed-Torque (CT) Control

A two-step design procedure now suggests itself. First, use linear system design techniques to select a feedback control  $u(t)$  that stabilizes the tracking error system (3.3.2), then compute the required arm torques using the inverse of (3.3.3), namely

$$\tau = M(q)(\ddot{q}_d - u) + N(q, \dot{q}). \quad (3.3.4)$$

Table 3.3.1: Robot Manipulator Control Algorithms

---

*Robot Dynamics:*

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = \tau,$$

*PD Computed Torque (CT) Control:*

$$\tau = M(q)(\ddot{q}_d + K_v\dot{e} + K_p e) + V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q)$$

*PID Computed Torque (CT) Control:*

$$\begin{aligned}\dot{\varepsilon} &= e \\ \tau &= M(q)(\ddot{q}_d + K_v\dot{e} + K_p e + K_i \varepsilon) + V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q)\end{aligned}$$

*PD-Gravity Controller:*

$$\tau = K_v\dot{e} + K_p e + G(q)$$

*Classical Joint Controller:*

$$\begin{aligned}\dot{\varepsilon} &= e \\ \tau &= K_v\dot{e} + K_p e + K_i \varepsilon\end{aligned}$$

*Digital Controller:*

$$\tau_k = M(q_k)(\ddot{q}_{d_k} + K_v\dot{e}_k + K_p e_k) + V_m(q_k, \dot{q}_k)\dot{q}_k + F(\dot{q}_k) + G(q_k)$$


---

This is a nonlinear feedback control law that guarantees tracking of the desired trajectory. It relies on computing the torque  $\tau$  that makes the nonlinear dynamics (3.2.1) equivalent to the linear dynamics (3.3.2), which is termed feedback linearization.

**PD Computed-Torque Control.** Selecting proportional-plus-derivative (PD) feedback for  $u(t)$  results in the PD computed-torque controller

$$\tau = M(q)(\ddot{q}_d + K_v\dot{e} + K_p e) + N(q, \dot{q}) \quad (3.3.5)$$

and yields the tracking error dynamics  $\ddot{e} = -K_v\dot{e} - K_p e$ , which is stable as long as the derivative gain matrix  $K_v$  and the proportional gain matrix  $K_p$  are selected positive definite. It is common to select the gain matrices diagonal, so that stability is ensured as long as all gains are selected positive.

The PD computed-torque controller is shown in Fig. 3.3.1, which has a multiloop structure, with a nonlinear inner feedback linearization loop and an outer unity-gain tracking loop. Note that there are actually  $n$  outer loops, one for each joint. In this figure  $\underline{q} \equiv [q^T \quad \dot{q}^T]^T$ ,  $\underline{e} \equiv [e^T \quad \dot{e}^T]^T$  and  $\underline{q}_d \equiv [q_d^T \quad \dot{q}_d^T]^T$ .

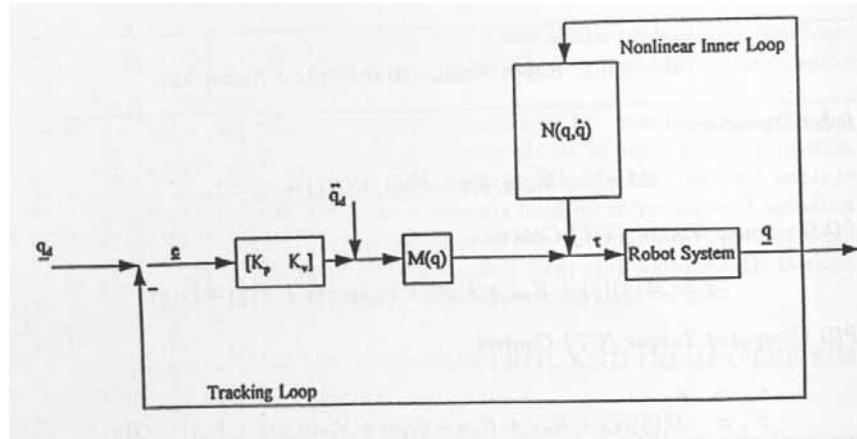


Figure 3.3.1: PD computed-torque controller.

**PID Computed-Torque Control.** To improve steady-state tracking errors,  $n$  integrators can be added, one to each joint controller, to place an integrator in the outer tracking loop in the figure. In fact, selecting  $u(t)$  as a proportional-plus-integral-plus-derivative controller yields the PID computed-torque controller

$$\begin{aligned}\dot{\varepsilon} &= e \\ \tau &= M(q)(\ddot{q}_d + K_v \dot{e} + K_p e + K_i \varepsilon) + N(q, \dot{q}).\end{aligned}\quad (3.3.6)$$

which has its own dynamics, and gives stable tracking as long as the integral gain  $K_i$  is not chosen too large.

### 3.3.2 Computer Simulation of Robot Controllers

The robot dynamics are given by

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = \tau, \quad (3.3.7)$$

where the joint variable vector is  $q \in \mathbb{R}^n$  for an  $n$ -link arm. One may alternatively write

$$M(q)\ddot{q} + N(q, \dot{q}) + \tau_d = \tau, \quad (3.3.8)$$

where the nonlinear vector is given by

$$N(q, \dot{q}) \equiv V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q). \quad (3.3.9)$$

These dynamics may include the actuators, and may be in Cartesian space, where  $q(t)$  should be interpreted as the end-effector Cartesian position.

Robot controllers can be simulated using the dynamical system techniques introduced in Chapter 2. A Runge-Kutta integrator such as ODE23 in MATLAB (1994) may be used. Then, a subroutine may be written that has two parts, the control input computation (e.g. (3.3.5)) and the robot dynamics in state-space form  $\dot{x} = f(x, u)$ .

One state-space form suitable for simulation is the position/velocity form, where the state is the  $2n$ -vector  $x \equiv [q^T \quad \dot{q}^T]^T$  and the dynamics are written as

$$\dot{x} = \begin{bmatrix} \dot{q} \\ -M^{-1}(q)N(q, \dot{q}) \end{bmatrix} + \begin{bmatrix} 0 \\ M^{-1}(q) \end{bmatrix} \tau. \quad (3.3.10)$$

For numerical integration purposes, the matrix inversion  $M^{-1}(q)$  is required at every integration time step to find  $\dot{x}$ . For arms with simple dynamics it is often possible to invert the inertia matrix analytically off-line, reducing the on-line computational burden. Otherwise, it is more suitable to solve (3.3.8) for  $\ddot{q}$ , required by the integration routine, at each time step using least-squares techniques to avoid the inversion of  $M(q)$ .

The next example is provided to illustrate computer simulation of robot controllers as well as to demonstrate the performance of computed-torque controllers.

**Example 3.3.1 (Simulation – Performance of PD/PID CT Controllers) :**

It is desired to design and simulate a PD computed-torque controller for the 2-link robot arm in Example 3.2.1. The dynamics are given in that example, and the PD control law by (3.3.5). The M file to be called by MATLAB routine ode23 is given in Fig. 3.3.2. Note that it has two parts—the controller and the arm dynamics. The controller computes the desired trajectory  $q_d(t)$ , the tracking error, and the arm torques. Here was selected  $q_{d1}(t) = g_1 \sin(2\pi t/T)$ ,  $q_{d2}(t) = g_2 \sin(2\pi t/T)$ , with amplitudes  $g_i = 0.1$  and period  $T = 2$  sec. The arm dynamics is in state-space form and computes the state derivatives for ode23.

To simulate a PID controller one must add additional states  $x(5), x(6)$  corresponding to the integrators. Then, the control torque computation lines in Fig. 3.3.2 are replaced by

```
% PID CT control torques
s1= qdpp(1) + kv*ep(1) + kp*e(1) + ki*x(5) ;
s2= qdpp(2) + kv*ep(2) + kp*e(2) + ki*x(6) ;
tau1= M11*s1 + M12*s2 + N1 ;
tau2= M12*s1 + M22*s2 + N2 ;
xdot(5)= e(1)
xdot(6)= e(2)
```

where the integral gain  $ki$  must be specified as a controller parameter. The integrator states should be initialized at zero on calling ode23.

Due to the idiosyncrasies of ode23, one must compute the outputs  $y = h(x, u)$  of the state equation  $\dot{x} = f(x, u)$  after completing the integration over  $[t_0, t_f]$ . The outputs required for plotting are the desired trajectory  $q_d(t)$  and the tracking error  $e(t)$ . The MATLAB M file for computing them is given by

```
% file robout.m
function [qd,e]= robout(t,x)
% compute desired trajectory
    period= 2 ; amp1= 0.1 ; amp2= 0.1 ;
    fact= 2*pi/period ;
    sinf= sin(fact*t) ;
    cosf= cos(fact*t) ;
    qd= [amp1*sinf amp2*cosf] ;
% tracking errors
    e= qd - x(:,1:2) ;
```

```
% file robctl.m, to be called by MATLAB function ode23
function xdot= robctl(t,x) ;

% -----
% COMPUTE CONTROL INPUT FOR ROBOT ARM

% compute desired trajectory
period= 2 ; amp1= 0.1 ; amp2= 0.1 ;
fact= 2*pi/period ;
sinf= sin(fact*t) ;
cosf= cos(fact*t) ;
qd= [amp1*sinf amp2*cosf]' ;
qdp= fact*[amp1*cosf -amp2*sinf]' ;
qdpp= -fact^2*qd ;

% PD Computed-Torque control input

m1= 1 ; m2= 1 ; a1= 1 ; a2= 1 ; g= 9.8 ; % arm parameters
kp= 100 ; kv= 20 ; % controller parameters

% tracking errors
e= qd - [x(1) x(2)]' ;
ep= qdp - [x(3) x(4)]' ;

% computed inertia M(q) and nonlinear terms N(q,qdot)
M11= (m1 + m2)*a1^2 + m2*a2^2 + 2*m2*a1*a2*cos(x(2)) ;
M12= m2*a2^2 + m2*a1*a2*cos(x(2)) ;
M22= m2*a2^2 ;
N1= -m2*a1*a2*(2*x(3)*x(4) + x(4)^2)*sin(x(2)) ;
N1= N1 + (m1 + m2)*g*a1*cos(x(1)) + m2*g*a2*cos(x(1) + x(2));
N2= m2*a1*a2*x(3)^2*sin(x(2)) + m2*g*a2*cos(x(1) + x(2)) ;
```

Figure 3.3.2: PD computed-torque controller, Part I.

```
% file robctl.m, to be called by MATLAB function ode23, CONTINUED

% PD CT control torques
s1= qdpp(1) + kv*ep(1) + kp*e(1) ;
s2= qdpp(2) + kv*ep(2) + kp*e(2) ;
tau1= M11*s1 + M12*s2 + N1 ;
tau2= M12*s1 + M22*s2 + N2 ;

% -----
% ROBOT ARM DYNAMICS
m1= 1 ; m2= 1 ; a1= 1 ; a2= 1 ; g= 9.8 ; % arm parameters

% inertia M(q) and nonlinear terms N(q,qdot)
M11= (m1 + m2)*a1^2 + m2*a2^2 + 2*m2*a1*a2*cos(x(2)) ;
M12= m2*a2^2 + m2*a1*a2*cos(x(2)) ;
M22= m2*a2^2 ;
N1= -m2*a1*a2*(2*x(3)*x(4) + x(4)^2)*sin(x(2)) ;
N1= N1 + (m1 + m2)*g*a1*cos(x(1)) + m2*g*a2*cos(x(1) + x(2));
N2= m2*a1*a2*x(3)^2*sin(x(2)) + m2*g*a2*cos(x(1) + x(2)) ;

% Inversion of M(q) (for large values of n, use least-squares)
det= M11*M22 - M12*M12 ;
MI11= M22/det ;
MI12= -M12/det ;
MI22= M11/det ;

% state equations
xdot(1)= x(3) ;
xdot(2)= x(4) ;
xdot(3)= MI11*(-N1 + tau1) + MI12*(-N2 + tau2) ;
xdot(4)= MI12*(-N1 + tau1) + MI22*(-N2 + tau2) ;
```

Figure 3.3.2 PD computed-torque controller (cont'd, Part II).

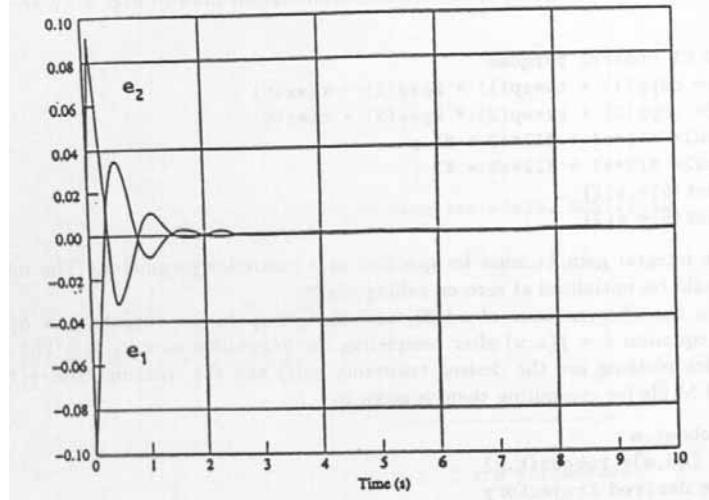


Figure 3.3.3: Joint tracking errors using PD computed-torque controller under ideal conditions.

The MATLAB dialog for running the simulation and plotting the tracking error is

```
t0= 0; tf= 10;
x0= [.1 0 0 0]';
[t,x]= ode23('robctl',t0,tf,x0);
[qd,e]= robout(t,x);
plot(t,e)
```

Three simulations were performed using this code.

**a. Ideal PD CT control.** Since CT is theoretically an exact cancellation of nonlinearities, under ideal circumstances the PD CT controller yields performance like that shown in Fig. 3.3.3, where the initial tracking errors go to zero quickly, so that each joint perfectly tracks its prescribed trajectory. In this figure are shown the plots for joint 1 tracking error  $e_1(t)$  and joint 2 tracking error  $e_2(t)$ .

**b. PD CT control with constant unknown disturbance.** Now, a constant unknown disturbance  $\tau_d$  is added to the robot arm dynamics. As shown in Fig. 3.3.4, the PD controller now exhibits steady-state tracking errors of  $e_1 = -0.01 \text{ rad}$ ,  $e_2 = 0.035 \text{ rad}$ .

**c. PID CT control.** If an integral term is now added to the outer loop to achieve PID CT control, even with a constant unknown disturbance, the simulation results look very much like the original plots in Fig. 3.3.3; that is, the integral term reduces to zero the steady-state tracking errors due to a constant disturbance.  $\square$

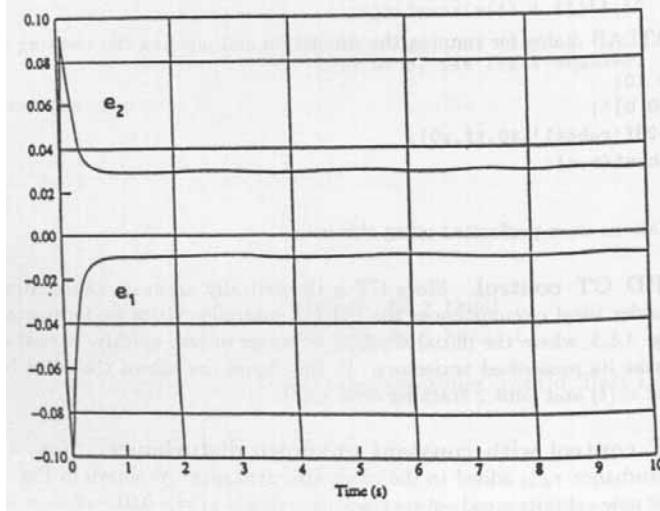


Figure 3.3.4: Joint tracking errors using PD computed-torque controller with constant unknown disturbance.

### 3.3.3 Approximate Computed-Torque Control and Classical Joint Control

*A large class of useful robot controllers can be derived using the notion of approximate computed-torque control.*

**A Family of Approximate Computed-Torque Controllers.** *A class of computed-torque-like controllers is given by selecting*

$$\tau = \hat{M}(\ddot{q}_d - u) + \hat{N}, \quad (3.3.11)$$

where  $\hat{M}, \hat{N}$  are approximations, estimates, or simplified expressions for  $M(q), N(q, \dot{q})$ .

If  $\hat{M}, \hat{N}$  are selected not as the actual inertia matrix and nonlinear terms, but only as approximations or simplified values, it is not always possible to guarantee stable tracking. In fact, the error dynamics (3.3.2) are then driven by modeling mismatch errors which can degrade or even destabilize the closed-loop system. Substituting the controller (3.3.11) into the dynamics (3.3.8) yields (see Problems section) the error dynamics

$$\ddot{e} = u - \Delta u + d, \quad (3.3.12)$$

where the inertia and nonlinear-term model mismatch errors are given by

$$\Delta \equiv M^{-1}(M - \hat{M}) = I - M^{-1}\hat{M} \quad (3.3.13)$$

$$\delta \equiv M^{-1}(N - \hat{N}) \quad (3.3.14)$$

and the disturbance is

$$d \equiv M^{-1}\tau_d + \Delta\ddot{q}_d + \delta(t). \quad (3.3.15)$$

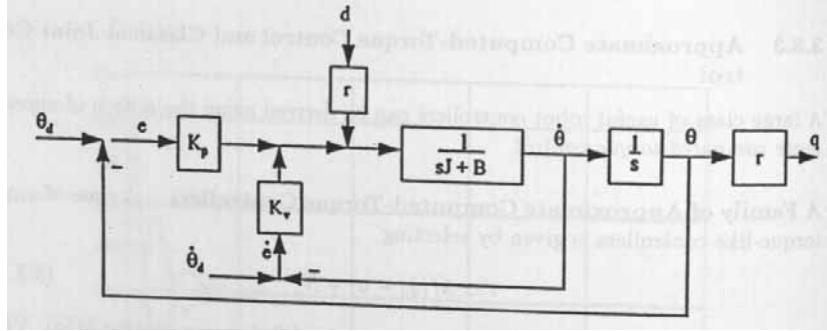


Figure 3.3.5: PD classical joint controller.

Therefore, if approximate CT control is used, the dynamics of the tracking error  $e = q_d - q$  are driven by the model mismatch terms, as well as the desired acceleration  $\ddot{q}_d(t)$ . Thus, as the performance expectations (e.g. maximum required acceleration) of the arm increase, the tracking error increases as well.

**PD-Gravity Control.** One example of the approximate CT family is the PD-gravity controller

$$\tau = K_v \dot{e} + K_p e + G(q) \quad (3.3.16)$$

which selects  $\hat{M} = I$  and only includes the gravity nonlinear terms, so that it is very easy to implement compared to full CT control. This has been used with good results in many applications.

**Classical Joint Control.** Another computed-torque-like controller is PID classical joint control, where all nonlinearities of the robot arm are neglected and one selects simply

$$\begin{aligned} \dot{\varepsilon} &= e \\ \tau &= K_v \dot{e} + K_p e + K_i \varepsilon \end{aligned} \quad (3.3.17)$$

with the gain matrices diagonal, so that all the joints are decoupled. A PD classical joint controller is shown in Fig. 3.3.5, which may seem familiar to many readers. The same figure may be drawn for each joint. In this figure,  $d(t)$  represents the neglected nonlinear coupling effects from the other joints, and  $r$  is the gear ratio. The motor angle is  $\theta(t)$  and  $q(t)$  is the joint angle. The effective joint inertia and damping are  $J$  and  $B$  respectively.

The simplified classical joint controller is very easy to implement as no digital computations are needed to determine nonlinear terms. It has been found suitable in many applications if the PD gains are selected high enough, particularly when the gear ratio  $r$  is small so that the neglected nonlinearities  $d(t)$  are attenuated. Unfortunately, if the gains are selected too high, the control may excite vibratory modes of the links and degrade performance. Moreover, practical applications often benefit by including additional terms such as gravity  $G(q)$ , desired acceleration feedforward  $\ddot{q}_d(t)$ , and various additional nonlinear terms.

**Example 3.3.2 (Performance of PD-Gravity and Classical Joint Controllers) :**

The sort of performance to be expected from PD-gravity and classical joint controllers is shown in this example. It is desired for the 2-link robot arm in Example 3.2.1 to follow, in each of its joints, sinusoidal trajectories  $q_d(t)$  with period of 2 sec.

- a. PD-Gravity controller.** The PD-Gravity controller is simulated using the same sort of MATLAB functions as in Example 3.3.1. The controller code in Fig. 3.3.2 is replaced by the following simplified code that implements the PD-gravity controller (3.3.16).

```
% PD-Gravity control input

m1= 1 ; m2= 1 ; a1= 1 ; a2= 1 ; g= 9.8 ; % arm parameters
kp= 100 ; kv= 20 ; % controller parameters

% tracking errors
e= qd - [x(1) x(2)]' ;
ep= qdp - [x(3) x(4)]' ;

% computed gravity terms
G1= (m1 + m2)*g*a1*cos(x(1)) + m2*g*a2*cos(x(1) + x(2));
G2= m2*g*a2*cos(x(1) + x(2)) ;

% PD CT control torques
s1= kv*ep(1) + kp*e(1) ;
s2= kv*ep(2) + kp*e(2) ;
tau1= s1 + G1
tau2= s2 + G2
```

The resulting joint 1 and 2 tracking errors are shown in Fig. 3.3.6. Note that the errors are small but not exactly zero, a reflection of the fact that the nonlinear Coriolis/centripetal terms are missing in the controller. However, the DC error is equal to zero, since gravity compensation is used. (The gravity terms are effectively the ‘DC terms’ of the robot dynamics.)

- b. Classical PD controller.** The sort of behavior to be expected from classical (independent joint) control is illustrated in Fig. 3.3.7. In this figure, the tracking errors are nonzero, but using large enough PD gains can often make them small enough. Note that the DC error is no longer equal to zero; the offset is due to ignoring the gravity terms.  
□

### 3.3.4 Digital Control

*Another important CT-like controller is the PD digital controller given by*

$$\tau_k = M(q_k)(\ddot{q}_{d_k} + K_v \dot{e}_k + K_p e_k) + N(q_k, \dot{q}_k). \quad (3.3.18)$$

*Digital control amounts to selecting in (3.3.11)  $\hat{M} = M(q_k)$ ,  $\hat{N} = N(q_k, \dot{q}_k)$ , so that the control input can only be computed at the sample times,  $t_k = KT$ , with  $T$  the sample period and  $k$  taking on integer values. Digital control is usually required in modern applications as robot control laws are generally implemented using microprocessors or digital signal processors. Unfortunately, the stability of robot*

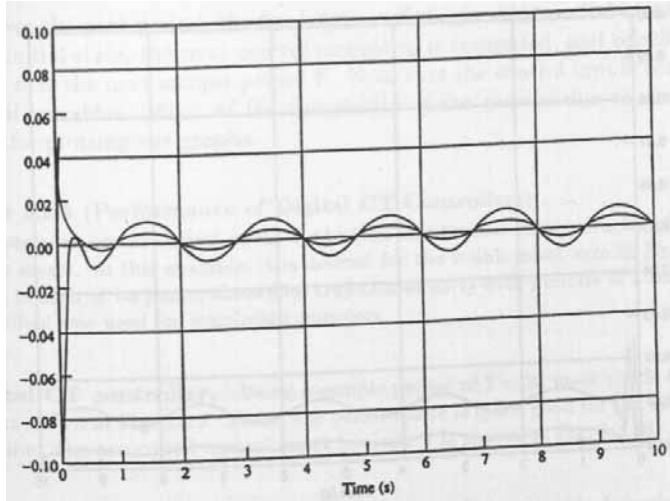


Figure 3.3.6: Joint tracking errors using PD-gravity controller.

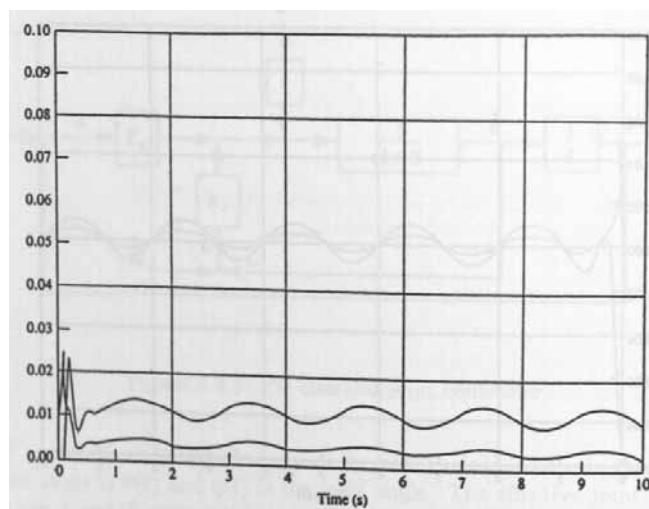


Figure 3.3.7: Joint tracking errors using classical independent joint control.

*digital controllers has not been generally addressed, so that the traditional approach relies on designing continuous-time controllers, meticulously proving stability, then sampling ‘fast enough’ and hoping for the best.*

*In practice there are many other problems to be faced in robot controller implementation, including actuator saturation, antiwindup compensation, and so on. See Lewis, Abdallah, and Dawson, (1993) and Lewis (1992).*

**Simulation of Digital Controllers.** *In digital controllers the control input  $\tau$  may only be updated at times  $kT$ , integer multiples of the sampling period  $T$ . In using fourth-order Runge-Kutta integrators for simulation, for instance, the integrator calls the state-space dynamics routine  $\dot{x} = f(x, u)$  four times during each integration interval, and there are usually many integration intervals during each sampling period. During all these calls the control input should be held at the same value  $\tau_k$ . This requires some trickery since in MATLAB, for instance, ode23 is not built to function in such a manner.*

*A digital control simulator for MATLAB is given for the 2-link robot arm (Example 3.2.1) in Fig. 3.3.8. It has several modules for easy modification, including the robot arm nonlinear term computation arm.m, the robot dynamics staspace.m, the digital controller control.m, and the desired trajectory generator sysinp.m. The driver program robot.m calls param.m (for parameter initialization), the digital controller control.m, and the Runge Kutta integrator ode23. In turn ode23 calls staspace.m, which calls arm.m. The controller control.m calls sysinp.m, and computes the control sample  $\tau_k$  using (3.3.18).*

*The way this program works is as follows. Using the initial conditions for the robot dynamics state  $x$ , the first control sample  $\tau_0$  is computed. Then, ode23 is used to integrate the system over one sample period  $T$  using this constant control value. Over the next period, the final state  $x$  of the previous period is reassigned as the new initial state, the next control sample  $\tau_k$  is computed, and ode23 is used to integrate over the next sample period  $T$ . Note that the control inputs (called  $t_1, t_2$ ) are global variables. Much of the complexity of the code is due to machinations required for plotting the graphs.*

### Example 3.3.3 (Performance of Digital CT Controllers) :

The performance of digital robot controllers has several idiosyncrasies of which one should be aware. In this example, it is desired for the 2-link robot arm in Example 3.2.1 to follow, in each of its joints, sinusoidal trajectories  $q_d(t)$  with periods of 2 sec. The code just described was used for simulation purposes.

**a. Digital CT controller.** Using a sample period of  $T = 20$  msec yields the tracking error plots shown in Fig. 3.3.9. There, the performance is quite good for the specific choice of PD gains. The associated control input for joint 2 is shown in Fig. 3.3.10.

**b. Limit cycle of digital robot controller.** Unacceptable behavior in digital robot controllers can be due to integrator windup problems, selecting too large a sample period, selecting too small a sample period (so that there is not enough time to perform all control calculations in each period), or the occurrence of *limit cycles*. If the sample period is selected as  $T = 100$  msec, everything seems acceptable according to Fig. 3.3.11, where the tracking errors are somewhat increased but still small. However, Fig. 3.3.12 shows the

```
% Digital Robot Controller- J. Campos July 23, 1996

% robot.m, Driver program, Part I
clear all;
global t1 t2
param; % Setting all the parameters
t0=0; % Initial Simulation Time (Sec)
tf=5; % Final Simulation Time (Sec)

x0=[0.1 0 0 0]'; % Initial Conditions
xx=x0;
tt=0:T:tf; % generating the discrete steps between t0 and tf
for i=1:length(tt)-1,
    t1=0;
    t2=0;
    [t1,t2]=control(kv,kp,tt(i),xx);
    clear t
    clear y
    [t,y]=ode23('staspace',tt(i),tt(i+1),x0,0.001);
    x0=y(length(t),:)';
    xx=x0;
    k111(i)=t1;
    k112(i)=t2;
    yyy(i,:)=x0';
end

for i=1:length(tt)-1,
    ttt(i)=tt(i);
end

stairs(ttt,yyy(:,1));
hold on
stairs(ttt,yyy(:,2));
hold off
title('Joint Angles theta1(t) and theta2(t) in rad (T=100 msec)');
xlabel('Time (sec)');
figure;

stairs(ttt,k111);
hold on
stairs(ttt,k112);
hold off
title('Input Torque Tao1(t) and Tao2(t) (T=100 msec)');
xlabel('Time (sec)');
```

Figure 3.3.8: Digital controller, Part I: Routine robot.m Part I.

```
% robot.m, Driver program, Part II

for i=1:length(ttt),
    [qd,qdp,qdpp]=sysinp(g1,g2,T,ttt(i));
    e(:,i)=qd-[yyy(i,1) yyy(i,2)]';
end
figure;
stairs(ttt,e(1,:));
hold on
stairs(ttt,e(2,:));
hold off
title('Tracking Error for Theta1(t) and Theta2(t) (T=100 msec)');
xlabel('Time (sec)');
-----
% control.m, Digital controller for 2-link Planar Arm
function [t1,t2]=control(kv,kp,t,xx);

param;
[qd,qdp,qdpp]=sysinp(g1,g2,T,t);
e=qd-[xx(1) xx(2)]';
ep=qdp-[xx(3) xx(4)]';

[M,N]=arm(m1,m2,a1,a2,g,xx);
s1=qdpp(1)+kv*ep(1)+kp*e(1);
s2=qdpp(2)+kv*ep(2)+kp*e(2);
t1=M(1,1)*s1+M(1,2)*s2+N(1);
t2=M(1,2)*s1+M(2,2)*s2+N(2);
-----
% param.m, arm and controller parameters

g1=0.1; g2=0.1;
T=100e-3; % Sampling period in seconds
m1=1; m2=1;
a1=1; a2=1;
g=9.8;
kp=100; kv=20;
-----
```

Figure 3.3.8 Digital controller, Part II: robot.m Part II and controller routines.

```
% sysinp.m , compute the desired trajectory
function [qd,qdp,qdpp]=sysinp(g1,g2,T,t)
fact=2*pi/2;
sinf=sin(fact*t);
cosf=cos(fact*t);
qd=[g1*sinf g2*cosf]';
qdp=fact*[g1*cosf -g2*sinf]';
qdpp=-fact^2*qd;
-----
%
% staspace.m, robot arm state equation
function xdot=staspace(t,x);
global t1 t2
param;
[M,N]=arm(m1,m2,a1,a2,g,x);
MI=inv(M);
xdot(1)=x(3);
xdot(2)=x(4);
xdot(3)=MI(1,1)*(-N(1)+t1)+MI(1,2)*(-N(2)+t2);
xdot(4)=MI(1,2)*(-N(1)+t1)+MI(2,2)*(-N(2)+t2);

% arm.m, 2-link robot arm inertia matrix and nonlinear term computation
function [M,N]=arm(m1,m2,a1,a2,g,x)

M11=(m1+m2)*a1^2+m2*a2^2+2*m2*a1*a2*cos(x(2));
M12=m2*a2^2+m2*a1*a2*cos(x(2));
M22=m2*a2^2;
N1=-m2*a1*a2*(2*x(3)*x(4)+x(4)^2)*sin(x(2));
N1=N1+(m1+m2)*g*a1*cos(x(1))+m2*g*a2*cos(x(1)+x(2));
N2=m2*a1*a2*x(3)^2*sin(x(2))+m2*g*a2*cos(x(1)+x(2));
M=[M11 M12;M12 M22];
N=[N1 N2]';
```

Figure 3.3.8 Digital controller, Part III: Robot arm routines.

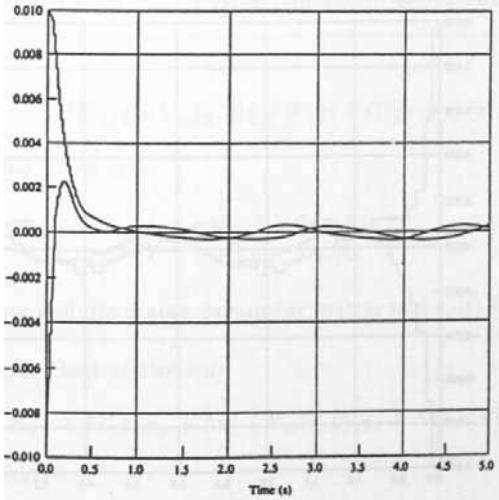


Figure 3.3.9: Joint tracking errors using digital computed-torque controller,  $T = 20$  msec.

control torque for link 2, which has now entered a limit cycle type behavior due to too large a sample period.  $\square$

### 3.4 FILTERED-ERROR APPROXIMATION-BASED CONTROL

*Computed-torque control works very well when all the dynamical terms  $M(q)$ ,  $V_m(q, \dot{q})$ ,  $F(\dot{q})$ ,  $G(q)$  are known. In practice, robot manipulator parameters such as friction coefficients are unknown or change with time, and the masses picked up by the arm are often unknown. Therefore, in many applications simplified CT controllers that do not compute all nonlinear terms are used (e.g. classical joint control). These methods often rely on increasing the PD gains to obtain good performance. However, large control signals may result and stability proofs of such controllers are few and far between. A large class of controllers are the approximation-based controllers which are based on approximating unknown robot functions. In this class are included various sorts of adaptive and robust control techniques, as well as learning control and the neural network techniques discussed later in the book. These controllers provide good performance even in the presence of unknown dynamics and disturbances. Performance and stability can be mathematically proven and so relied upon in applications. Such advanced techniques also extend directly to more complicated control objectives such as force control for grinding, polishing, and so on where straight PD methods are inadequate.*

*In this section we discuss approximation-based control based on the filtered tracking error, deriving several adaptive and robust controllers which are collected for easy reference in Table 3.4.1.*

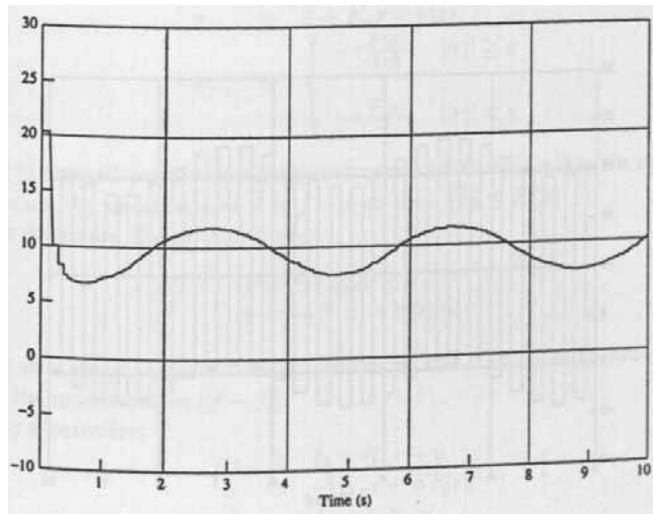


Figure 3.3.10: Joint 2 control torque using digital computed-torque controller,  $T=20$  msec.

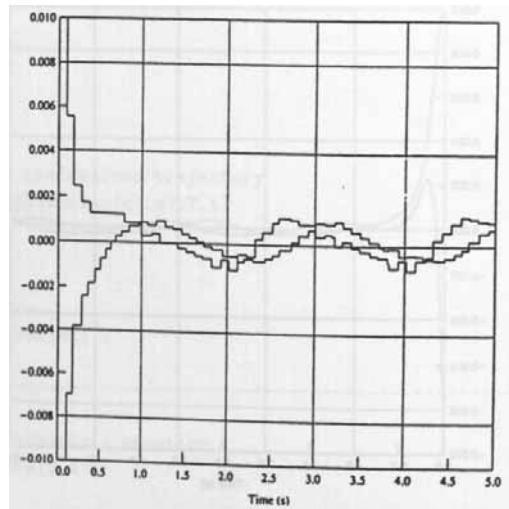


Figure 3.3.11: Joint tracking errors using digital computed-torque controller,  $T=100$  msec.

Table 3.4.1: Filtered-Error Approximation-Based Control Algorithms

*Robot Dynamics:*

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = \tau,$$

*Filtered Tracking Error  $r(t)$ :*

$$\begin{aligned} e &= q_d - q \\ r &= \dot{e} + \Lambda e \end{aligned}$$

with  $\Lambda$  a positive definite design parameter matrix and  $q_d(t)$  the desired trajectory.

*Computed-Torque Control Variant:*

$$\tau = K_v r + M(q)(\ddot{q}_d + \Lambda \dot{e}) + V_m(q, \dot{q})(\dot{q}_d + \Lambda e) + F(\dot{q}) + G(q).$$

*Adaptive Controller:*

$$\begin{aligned} \tau &= W(x)\hat{\phi} + K_v r \\ \dot{\hat{\phi}} &= \Gamma W^T(x)r, \end{aligned}$$

where  $W(x)$  is the regression matrix and  $\Gamma$  is a positive definite tuning rate matrix.

*Saturation-Based Robust Controller:*

$$\begin{aligned} \tau &= \hat{f} + K_v r - v(t) \\ v(t) &= \begin{cases} -r \frac{F(x)}{\|r\|}, & \|r\| \geq \varepsilon \\ -r \frac{F(x)}{\varepsilon}, & \|r\| < \varepsilon \end{cases} \end{aligned}$$

where  $\varepsilon$  is a small positive design parameter and  $F(x)$  is a known scalar function that bounds the uncertainties  $\tilde{f} = f - \hat{f}$  so that  $\|\tilde{f}\| \leq F(x)$ .

*Variable Structure Robust Controller:*

$$\begin{aligned} \tau &= \hat{f} + K_v r - v \\ v &= -(F(x) + \eta) \operatorname{sgn}(r) \end{aligned}$$

where  $\eta$  is a small positive design parameter and  $F(x)$  is a known function that bounds the uncertainties  $\|f - \hat{f}\|$ .

*Learning Controller:*

$$\begin{aligned} \tau_\ell &= \hat{f}_\ell + K_v r - v \\ v &= -(K_p e + K_s \|e\|^2 r) \\ \hat{f}_\ell &= \hat{f}_{\ell-1} + K_L r \end{aligned}$$

where  $\ell$  is the iteration number, the gains  $K_v, K_p, K_s$  are positive diagonal design matrices, and  $K_L$  is a positive diagonal learning gain matrix.

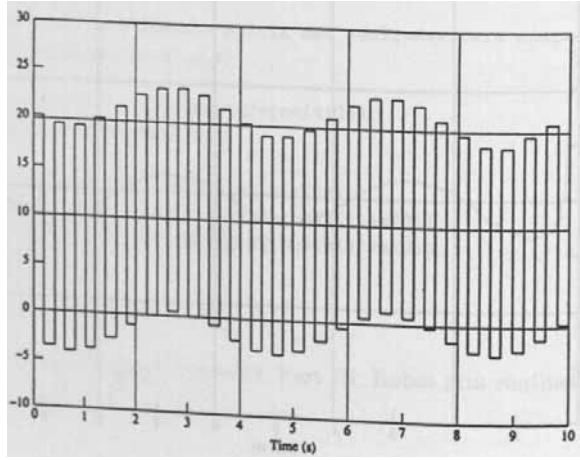


Figure 3.3.12: Joint 2 control torque using digital computed-torque controller,  $T = 100$  msec.

### 3.4.1 A General Controller Design Framework Based on Approximation

*The robot dynamics is given by*

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = \tau. \quad (3.4.1)$$

*In this section is derived a general tracking controller structure for robots that can be used to design adaptive, robust, and learning controllers, as well as neural network controllers.*

**The Tracking Problem.** *In robot control the objective is generally to make the robot manipulator follow a prescribed desired trajectory, often expressed in joint space as  $q_d(t)$ . Finding a control input  $\tau(t)$  that causes this to occur is called the tracking design problem. A general framework for tracking control that includes many adaptive, robust, learning, and neural network techniques is the approximation-based technique now presented. Given the desired trajectory  $q_d(t)$  define the tracking error  $e(t)$  and filtered tracking error  $r(t)$  by*

$$e = q_d - q \quad (3.4.2)$$

$$r = \dot{e} + \Lambda e \quad (3.4.3)$$

*with  $\Lambda$  a positive definite design parameter matrix. Common usage is to select  $\Lambda$  diagonal with large positive entries. Then, (3.4.3) is a stable system so that  $e(t)$  is bounded as long as the controller guarantees that the filtered error  $r(t)$  is bounded. In fact, it is easy to show that one has*

$$\|e\| \leq \frac{\|r\|}{\sigma_{min}(\Lambda)}, \quad \|\dot{e}\| \leq \|r\|, \quad (3.4.4)$$

*where  $\sigma_{min}(\Lambda)$  is the minimum singular value of  $\Lambda$  and the 2-norm is used.*

In practical situations the desired trajectory is specified by the design engineer, so that it always satisfies the following boundedness assumption.

**Bounded Trajectory.** The desired trajectory is bounded so that

$$\begin{aligned} \left\| \begin{array}{c} q_d(t) \\ \dot{q}_d(t) \\ \ddot{q}_d(t) \end{array} \right\| &\leq q_B, \end{aligned} \quad (3.4.5)$$

with  $q_B$  a known scalar bound.

Differentiating (3.4.3) and invoking (3.4.1) it is seen that the robot dynamics are expressed in terms of the filtered error as

$$M\dot{r} = -V_m r + f(x) + \tau_d - \tau \quad (3.4.6)$$

where the nonlinear robot function is defined as

$$f(x) = M(q)(\ddot{q}_d + \Lambda \dot{e}) + V_m(q, \dot{q})(\dot{q}_d + \Lambda e) + F(\dot{q}) + G(q). \quad (3.4.7)$$

Vector  $x$  contains all the time signals needed to compute  $f(\cdot)$  and may be defined for instance as  $x \equiv [e^T \dot{e}^T q_d^T \dot{q}_d^T \ddot{q}_d^T]^T$ . It is important to note that  $f(x)$  contains all the potentially unknown robot arm parameters, except for the  $V_m r$  term in (3.4.6), which cancels out in controller stability Lyapunov proofs.

The definition of the nonlinear Coriolis/centripetal matrix  $V_m(q, \dot{q})$  in the robot dynamics (3.4.1) is not unique (Lewis, Abdallah, and Dawson 1993). In approximation-based control using the approach given here, it is necessary to select the correct version of  $V_m(q, \dot{q})$ ; specifically, one must use the version of  $V_m(q, \dot{q})$  so that the skew-symmetry property P3 holds. See Example 3.2.2 and Problems section.

**Approximation-Based Controllers.** A general sort of approximation-based controller is now derived by setting

$$\tau = \hat{f} + K_v r - v(t), \quad (3.4.8)$$

with  $\hat{f}$  an estimate of  $f(x)$ ,  $K_v r = K_v \dot{e} + K_v \Lambda e$  an outer PD tracking loop, and  $v(t)$  an auxiliary signal to provide robustness in the face of disturbances and modelling errors. The estimate  $\hat{f}$  and robustifying signal  $v(t)$  are defined differently for adaptive control, robust control, neural net control, fuzzy logic control, etc. The multiloop control structure implied by this scheme is shown in Fig. 3.4.1. In adaptive control techniques, most of the effort goes into selecting and updating the estimate  $\hat{f}(x)$ ; in robust control, most of the effort goes into selecting the control term  $v(t)$ .

**The Error Dynamics.** Using nonlinear stability proofs based on Lyapunov or passivity techniques, it can be shown that tracking error stability can be guaranteed by selecting one of a variety of specific controllers. The controllers are derived and proofs of stability are given based on the all-important closed-loop error dynamics. The closed-loop error dynamics are found by substitution of (3.4.8) into (3.4.6) to be

$$M\dot{r} = -V_m r - K_v r + \tilde{f} + \tau_d + v(t), \quad (3.4.9)$$

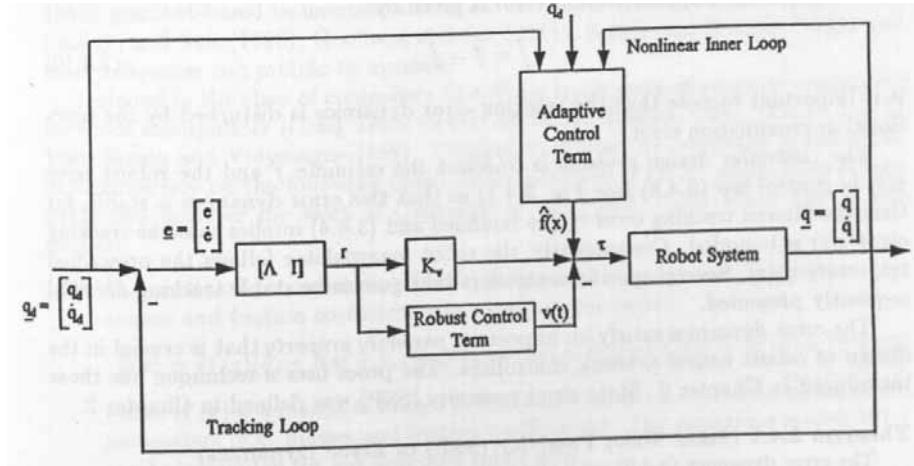


Figure 3.4.1: Filtered error approximation-based controller.

where the function approximation error is given by

$$\tilde{f} = f - \hat{f}. \quad (3.4.10)$$

It is important to note that the tracking error dynamics is disturbed by the functional approximation error.

The controller design problem is to select the estimate  $\hat{f}$  and the robust term  $v(t)$  in control law (3.4.8) (see Fig. 3.4.1) so that this error dynamics is stable, for then the filtered tracking error  $r(t)$  is bounded and (3.4.4) implies that the tracking error  $e(t)$  is bounded. Consequently, the robot manipulator follows the prescribed trajectory  $q_d(t)$ . Several specific controllers that guarantee stable tracking are subsequently presented.

The error dynamics satisfy an important passivity property that is crucial in the design of robust neural network controllers. The proof uses a technique like those introduced in Chapter 2. State strict passivity (SSP) was defined in Chapter 2.

**Theorem 3.4.1 (State Strict Passivity (SSP) of Error Dynamics) :**

The error dynamics (3.4.9) are a state strict passive system from  $\zeta_0(t) \equiv \tilde{f} + \tau_d + v(t)$  to  $r(t)$ .

Proof:

Take the nonnegative function

$$L = \frac{1}{2}r^T M(q)r$$

so that, using (3.4.9), one obtains

$$\dot{L} = r^T M \dot{r} + \frac{1}{2} r^T \dot{M} r = -r^T K_v r + \frac{1}{2} r^T (\dot{M} - 2V_m) r + r^T \zeta_0$$

whence the skew-symmetry property P3 yields the power form

$$\dot{L} = r^T \zeta_0 - r^T K_v r$$

This is the power delivered to the system minus a quadratic term in  $\|r\|$ , verifying state strict passivity.  $\square$

### 3.4.2 Computed-Torque Control Variant

Several approximation-based controllers of the form (3.4.8) are now presented. A variant of computed-torque control can be used if the nonlinear function  $f(x)$  is known. Then, one may select  $\hat{f}(x) = f(x)$  so that the control input is

$$\tau = K_v r + M(q)(\ddot{q}_d + \Lambda \dot{e}) + V_m(q, \dot{q})(\dot{q}_d + \Lambda e) + F(\dot{q}) + G(q). \quad (3.4.11)$$

This may be compared to (3.3.5).

### 3.4.3 Adaptive Control

Adaptive control has proven successful in dealing with modeling uncertainties in general linear and nonlinear systems by on-line tuning of parameters. Variants of adaptive control include the model-reference approach (Landau 1976), hyperstability techniques (Landau 1976), self-tuning regulators (Åström and Wittenmark 1989), gradient-based techniques, and so on. See Narendra and Annaswamy (1989), Ioannou and Sun (1996), Goodwin and Sin (1984), Sastry and Bodson (1989) and other references too prolific to mention.

Included in the class of controllers (3.4.8) are many sorts of adaptive controllers for robot manipulators (Craig 1985; Lewis, Abdallah, Dawson 1993; Slotine and Li 1991; Spong and Vidyasagar 1989). Conventional adaptive controller applications in robotics rely on the following linear-in-the-parameters (LIP) assumption (Craig 1985) (see however the work of Colbaugh et al. (1994, 1995) where LIP is not needed).

**LIP** The nonlinear robot function (3.4.7) is linear in the unknown parameters (e.g. masses and friction coefficients) so that one can write

$$f(x) = M(q)(\ddot{q}_d + \Lambda \dot{e}) + V_m(q, \dot{q})(\dot{q}_d + \Lambda e) + F(\dot{q}) + G(q) = W(x)\phi \quad (3.4.12)$$

where  $W(x)$  is a matrix of known robot functions and  $\phi$  is a vector of unknown parameters (e.g. masses and friction coefficients). The regression matrix  $W(\cdot)$  can be computed for any specified robot arm.

One adaptive controller is given by Slotine (1988),

$$\tau = W(x)\hat{\phi} + K_v r \quad (3.4.13)$$

$$\dot{\hat{\phi}} = \Gamma W^T(x)r \quad (3.4.14)$$

where  $\Gamma$  is a tuning parameter matrix, generally selected diagonal with positive elements. The adaptive controller manufactures an estimate  $\hat{\phi}$  for the unknown parameter vector  $\phi$  by dynamic on-line tuning using Equation (3.4.14), thus the controller has its own dynamics. The estimate of the nonlinear function is given by

$$\hat{f}(x) = W(x)\hat{\phi}. \quad (3.4.15)$$

The adaptive controller has the structure shown in Fig. 3.4.1. It has a multiloop structure with an outer PD tracking loop and an inner nonlinear adaptive loop whose function is to estimate the nonlinear function required for feedback linearization of the robot arm. Its performance is described in the following theorem and illustrated in the next example. Uniform ultimate bounded (UUB) stability was defined in Chapter 2.

**Theorem 3.4.2 (Adaptive Controller) :**

Suppose the disturbance  $\tau_d(t)$  in (3.4.1) is zero and the desired trajectory  $q_d(t)$  is bounded according to (3.4.5). Assume the linear-in-the-parameters assumption (3.4.12) holds and the unknown parameter vector  $\phi$  is constant. Then, using the control (3.4.13) with adaptive parameter tuning given by (3.4.14), the tracking error  $r(t)$  goes to zero and the parameter estimates  $\hat{\phi}(t)$  are UUB.

Proof:

Due to the LIP assumption one has

$$\tilde{f} = f - \hat{f} = W(x)\phi - W(x)\hat{\phi} = W(x)\tilde{\phi},$$

with  $\tilde{\phi} = \phi - \hat{\phi}$  the parameter estimation error. Therefore, using the proposed control yields the closed-loop error dynamics

$$M\dot{r} = -V_m r - K_v r + W(x)\tilde{\phi}.$$

Select the Lyapunov function candidate

$$L = \frac{1}{2}r^T M(q)r + \frac{1}{2}\tilde{\phi}^T \Gamma^{-1}\tilde{\phi},$$

with  $\Gamma$  a symmetric positive definite weighting matrix. Differentiate to discover

$$\dot{L} = \frac{1}{2}r^T \dot{M}r + r^T M\dot{r} + \tilde{\phi}^T \Gamma^{-1}\dot{\tilde{\phi}},$$

whence substitution from the error dynamics yields

$$\begin{aligned} \dot{L} &= \frac{1}{2}r^T \dot{M}r - r^T V_m r - r^T K_v r + r^T W(x)\tilde{\phi} + \tilde{\phi}^T \Gamma^{-1}\dot{\tilde{\phi}} \\ &= \frac{1}{2}r^T (\dot{M} - 2V_m)r - r^T K_v r + \tilde{\phi}^T (\Gamma^{-1}\dot{\tilde{\phi}} + W^T(x)r). \end{aligned}$$

Now use the skew-symmetry property P3 and select

$$\dot{\tilde{\phi}} = -\Gamma W^T(x)r$$

to yield

$$\dot{L} = -r^T K_v r.$$

In view of the definition  $\tilde{\phi} = \phi - \hat{\phi}$ , and the assumption that  $\phi$  is constant, the selection for  $\dot{\tilde{\phi}}$  yields the tuning law (3.4.14).

Since  $L$  is positive definite in the overall state  $[r^T \ \tilde{\phi}^T]^T$  and  $\dot{L}$  is negative semidefinite, both  $r$  and  $\tilde{\phi}$  are bounded according to Lyapunov's theorem. Boundedness of the parameter estimate  $\hat{\phi}$  follow from the fact that  $\phi$  is bounded.

To show that  $r(t)$  goes to zero, one may use Barbalat's Lemma to show that  $\dot{L}$  goes to zero with  $t$ . To accomplish this, differentiate to obtain

$$\ddot{L} = -2r^T K_v \dot{r},$$

and substitute the error dynamics to obtain

$$\ddot{L} = -2r^T K_v M^{-1} [-V_m r - K_v r + W(x)\tilde{\phi}].$$

The right-hand side is bounded due to the bounding assumptions P1-P6 and the demonstrated boundedness of  $r$  and  $\hat{\phi}$ ; for one has

$$M^{-1} \leq \frac{1}{\mu_1} I, \quad \|V_m\| \leq v_B \|\dot{q}\|;$$

but  $\dot{q}$  is bounded since  $r$  is bounded and the desired trajectory  $q_d(t)$  is bounded. Therefore,  $\ddot{L}$  is bounded, implying that  $\dot{L}$  is uniformly continuous, and by the Barbalat's Lemma Lyapunov extension (Chapter 1),  $L$  goes to zero with  $t$ . Therefore,  $r(t)$  vanishes as  $t$  becomes large.  $\square$

*It has only been shown that the parameter error  $\tilde{\phi}$  is bounded, not small. It can be shown that under an additional persistence of excitation condition (Chapter 2), the parameter error goes to zero so that the parameter estimates  $\hat{\phi}$  approach the actual values  $\phi$  (Slotine 1985, 1988). However, for practical controls purposes the result of the theorem is good enough, since the tracking error  $r(t)$  is small so that one has tracking of the prescribed trajectory. Note also that the UUB of the parameter estimate  $\hat{\phi}$  guarantees the boundedness of the control input  $\tau(t)$  in (3.4.13).*

*This proof is typical of adaptive control and recurs in various forms throughout the book—The parameter to be tuned is weighted in the Lyapunov function candidate  $L$ . Then, the derivative of the parameter appears in  $\dot{L}$ , yielding the tuning law.*

#### Example 3.4.1 (Performance of Adaptive Controller) :

This example illustrates the sort of performance to be expected from the adaptive controller. It is also shown that if the regression matrix is not exactly known, the adaptive controller does not perform well. In this example, it is desired for the 2-link robot arm in Example 3.2.1 to follow a prescribed trajectory.

To implement the adaptive controller just described, it is necessary to determine the regression matrix  $W(x)$ . In computing  $W(x)$  one must begin with the version of  $V_m(q, \dot{q})$  that satisfies the skew-symmetry property P3 (see Example 3.2.2). The unknown parameter vector is selected to contain the masses, so that  $\phi = [m_1 \ m_2]^T$ , where  $m_2$  includes the payload mass. The joint tracking errors are  $e_1 = q_{d1} - q_1$ ,  $e_2 = q_{d2} - q_2$ . Referring to the dynamics derived in Example 3.2.1, one sees that the elements of the  $2 \times 2$  matrix  $W(x)$  are given (see Problems section) by

$$\begin{aligned} W_{11} &= a_1^2(\ddot{q}_{d1} + \lambda_1 \dot{e}_1) + a_1 g \cos q_1 \\ W_{12} &= (a_2^2 + 2a_1 a_2 \cos q_2 + a_1^2)(\ddot{q}_{d1} + \lambda_1 \dot{e}_1) + (a_2^2 + a_1 a_2 \cos q_2)(\ddot{q}_{d2} + \lambda_2 \dot{e}_2) \\ &\quad - a_1 a_2 \dot{q}_2 (\dot{q}_{d1} \sin q_2 + \lambda_1 e_1) - a_1 a_2 (\dot{q}_1 + \dot{q}_2) (\dot{q}_{d2} \sin q_2 + \lambda_2 e_2) \\ &\quad + a_2 g \cos(q_1 + q_2) + a_1 g \cos q_1 \\ W_{21} &= 0 \\ W_{22} &= (a_2^2 + a_1 a_2 \cos q_2)(\ddot{q}_{d1} + \lambda_1 \dot{e}_1) + a_2^2(\ddot{q}_{d2} + \lambda_2 \dot{e}_2) \\ &\quad + a_1 a_2 (\dot{q}_{d1} + \lambda_1 e_1) \sin q_2 + a_2 g \cos(q_1 + q_2). \end{aligned}$$

A function M file was written in MATLAB (1994) to simulate the adaptive controller—this procedure is very direct. The adaptive controller is an easy modification of the function routine given in Example 3.3.1. The PD CT controller equations in Fig. 3.3.2 are simply replaced by the adaptive controller equations (3.4.13)-(3.4.14). The lines of code implementing the adaptive controller are given in Fig. 3.4.2.

```
% file robadapt.m, to be called by MATLAB routine ode23

function xdot= robadapt(t,x) ;

% compute desired trajectory as in Fig. 3.3.2

% Use period= 2*pi ; amp1= 1 ; amp2= 1 ;

% Adaptive control input

m1= 0.8 ; m2= 2.3 ; a1= 1 ; a2= 1 ; g= 9.8 ; % arm parameters
Kv= 20*eye(2) ; lam= 5*eye(2); gam= 10*eye(2) ; % controller parameters

% tracking errors
e= qd - [x(1) x(2)]' ;
ep= qdp - [x(3) x(4)]' ;
r= ep + lam*e ;

% compute regression matrix
f= qdpp + lam*ep ;
W(1,1) = a1^2*f(1) + a1*g*cos(x(1)) ;
W(1,2) = (a2^2 + 2*a1*a2*cos(x(2)) + a1^2)*f(1) ...
+ (a2^2 + a1*a2*cos(x(2)))*f(2) ...
- a1*a2*x(4)*(qdp(1)*sin(x(2)) + lam(1,1)*e(1)) ...
- a1*a2*(x(3)+ x(4))*(qdp(2)+sin(x(2)) + lam(2,2)*e(2)) ...
+ a2*g*cos(x(1) + x(2)) + a1*g*cos(x(1)) ;
W(2,1) = 0 ;
W(2,2) = (a2^2 + a1*a2*cos(x(2)))*f(1) + a2^2*f(2) ...
+ a1*a2*(qdp(1) + lam(1,1)*e(1))*sin(x(2)) ...
+ a2*g*cos(x(1) + x(2)) ;

% control torques. Parameter estimates are [x(5) x(6)]'
tau= Kv*r + W*[x(5) x(6)]' ;
tau1= tau(1) ; tau2= tau(2) ;
% parameter updates
phidot= gam*W'*r ;
xdot(5)= phidot(1) ;
xdot(6)= phidot(2) ;

% ROBOT ARM DYNAMICS from Fig. 3.3.2
```

Figure 3.4.2: Adaptive controller.

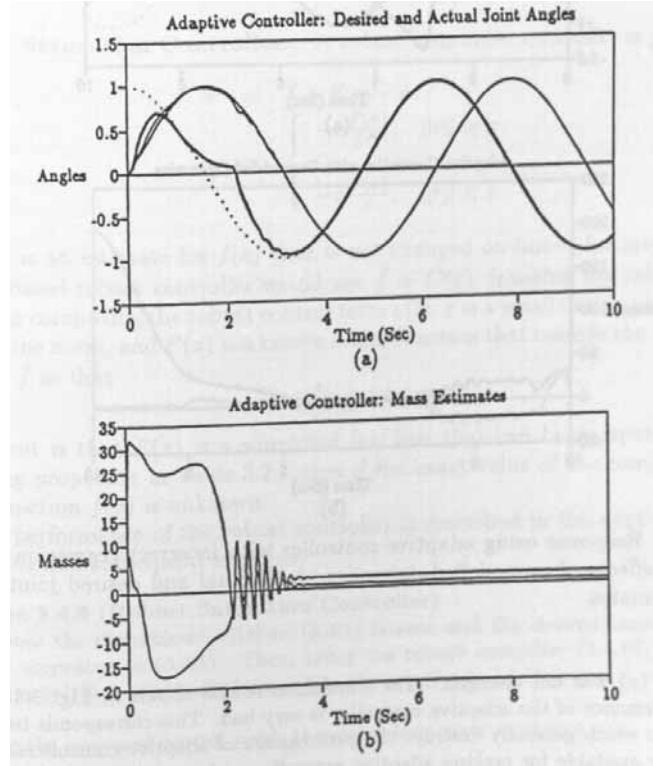


Figure 3.4.3: Response using adaptive controller. (a) Actual and desired joint angles. (b) Mass estimates.

The arm parameters were taken as  $a_1 = a_2 = 1m$ ,  $m_1 = 0.8kg$ ,  $m_2 = 2.3kg$ . The desired trajectory was selected as  $q_{d1}(t) = \sin t$ ,  $q_{d2}(t) = \cos t$ . The controller parameters were selected as  $K_v = \text{diag}\{20, 20\}$ ,  $\Gamma = \text{diag}\{10, 10\}$ ,  $\Lambda = \text{diag}\{\lambda_1, \lambda_2\} = \text{diag}\{5, 5\}$ . The dialog required to run the adaptive controller is the same as in Example 3.3.1. Remember to modify the output function *robout.m* to reflect the new desired trajectory parameters!

**a. Adaptive Control.** The response with the adaptive controller is given in Fig. 3.4.3, which is excellent even though the masses  $m_1, m_2$  are unknown by the controller. It is seen that, after an initial error, the actual joint angles closely match the desired joint angles. In adaptive control, the controller dynamics allow for learning of the unknown parameters, so that the performance improves over time. Note that the parameter (mass) estimates converge to the correct constant values.

**b. Adaptive Control with Unmodelled Dynamics.** To show how important it is to model all the dynamics in the regression matrix, the simulation was now repeated using the incorrect value

$$W_{11} = a_1^2(\ddot{q}_{d1} + \lambda_1 \dot{e}_1);$$

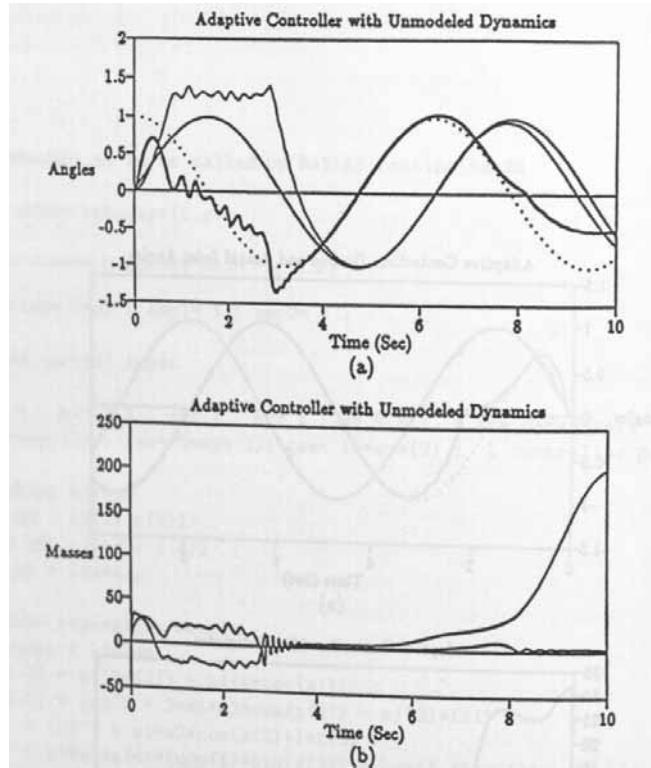


Figure 3.4.4: Response using adaptive controller with incorrect regression matrix, showing the effects of unmodelled dynamics. (a) Actual and desired joint angles. (b) Mass estimates.

the rest of  $W(x)$  was not changed. The simulation results shown in Fig. 3.4.4 reveal that the performance of the adaptive controller is very bad. This corresponds to *unmodelled dynamics* which generally destroys the performance of adaptive controllers. Several techniques are available for making adaptive controllers *robust to unmodelled dynamics*, including the  $e$ -modification (Narendra and Annaswamy 1987), the  $\sigma$ -modification (Ioannou and Kokotovic 1984), and deadzone techniques (Kreisselmeier and Anderson 1986). These all involve adding correction terms to the tuning algorithm (3.4.14).  $\square$

### 3.4.4 Robust Control

*Robust controllers for robot arms (Spong et al. 1987, Corless 1989, Dawson et al. 1990) comprise a large class of controllers, some of which are based on approximation techniques. Referring to Equation (3.4.8) and Fig. 3.4.1, in adaptive controllers the primary design effort goes into selecting a dynamic estimate  $\hat{f}$  for the nonlinear function (3.4.7) that is tuned on-line. By contrast, in robust controllers, the primary design effort goes into selecting the robust term  $v(t)$ . An advantage of standard robust controllers is that they have no dynamics, so they are generally*

simpler to implement. On the other hand, adaptive controllers are somewhat more refined in that the dynamics are learned on-line and less control effort is usually needed. Furthermore, in adaptive control it is necessary to compute a regression matrix  $W(x)$ , while in robust control it is necessary to compute a bounding function  $F(x)$ . In some modern techniques, robust and adaptive techniques are combined to provide the advantages of each class. These are called adaptive-robust controllers, or robust-adaptive controllers. Two popular robust controllers are now described.

**Robust Saturation Controller.** A robust saturation controller is given by

$$\begin{aligned} \tau &= \hat{f} + K_v r - v \\ v &= \begin{cases} -r \frac{F(x)}{\|r\|}, & \|r\| \geq \varepsilon \\ -r \frac{F(x)}{\varepsilon}, & \|r\| < \varepsilon \end{cases} \end{aligned} \quad (3.4.16)$$

where  $\hat{f}$  is an estimate for  $f(x)$  that is not changed on-line—for instance, a PD-gravity-based robust controller would use  $\hat{f} = G(q)$ , ignoring the other nonlinear terms. In computing the robust control term  $v(t)$ ,  $\varepsilon$  is a small design parameter,  $\|\cdot\|$  denotes the norm, and  $F(x)$  is a known scalar function that bounds the uncertainties  $\tilde{f} = f - \hat{f}$  so that

$$\|\tilde{f}\| \leq F(x). \quad (3.4.17)$$

The intent is that  $F(x)$  is a simplified function that can be computed using the bounding properties in Table 3.2.1 even if the exact value of the complicated nonlinear function  $f(x)$  is unknown.

The performance of the robust controller is described in the next theorem and illustrated in a subsequent example.

**Theorem 3.4.3 (Robust Saturation Controller) :**

Suppose the disturbance  $\tau_d(t)$  in (3.4.1) is zero and the desired trajectory  $q_d(t)$  is bounded according to (3.4.5). Then, using the robust controller (3.4.16), the tracking error norm  $\|r(t)\|$  is eventually bounded to the neighborhood of  $\varepsilon$ .

Proof:

Using the proposed control yields the closed-loop error dynamics

$$M\dot{r} = -V_m r - K_v r + \tilde{f} + v,$$

with  $\|\tilde{f}\| < F(x)$  the known function. Select the Lyapunov function candidate

$$L = \frac{1}{2} r^T M(q) r.$$

Differentiate to discover

$$\dot{L} = \frac{1}{2} r^T \dot{M} r + r^T M \dot{r}$$

whence substitution from the error dynamics yields

$$\begin{aligned} \dot{L} &= \frac{1}{2} r^T \dot{M} r - r^T V_m r - r^T K_v r + r^T (\tilde{f} + v) \\ &= \frac{1}{2} r^T (\dot{M} - 2V_m) r - r^T K_v r + r^T (\tilde{f} + v), \end{aligned}$$

so that skew-symmetry property P3 yields

$$\dot{L} = -r^T K_v r + r^T (\tilde{f} + v).$$

Recall from Chapter 2 the norm property  $\sigma_{\min}(K_v) \|r\|^2 \leq r^T K_v r$ , with  $\sigma_{\min}(K_v)$  the smallest singular value of the gain matrix  $K_v$ . (If  $K_v$  is diagonal, this is simply equal to the smallest gain value.) Therefore,

$$\begin{aligned}\dot{L} &\leq -\sigma_{\min}(K_v) \|r\|^2 + \|r\| \cdot \|\tilde{f}\| + r^T v \\ \dot{L} &\leq -\sigma_{\min}(K_v) \|r\|^2 + \|r\| F(x) + r^T v.\end{aligned}$$

There are now two cases to consider.

**Case 1.**  $\|r\| \geq \varepsilon$ . In this case, according to the definition of the robust control term  $v(t)$  one has

$$\begin{aligned}\dot{L} &\leq -\sigma_{\min}(K_v) \|r\|^2 + \|r\| F(x) - \|r\|^2 F(x)/\|r\| \\ &\leq -\sigma_{\min}(K_v) \|r\|^2 + \|r\| F(x) - \|r\| F(x) \\ &\leq -\sigma_{\min}(K_v) \|r\|^2.\end{aligned}$$

Therefore,  $\dot{L}$  is negative definite in terms of  $\|r(t)\|$ . Hence,  $L$  is decreasing in this region and  $\|r\|$  decreases towards  $\varepsilon$ .

**Case 2.**  $\|r\| < \varepsilon$ . In this case, according to the definition of the robust control term  $v(t)$  one has

$$\begin{aligned}\dot{L} &\leq -\sigma_{\min}(K_v) \|r\|^2 + \|r\| F(x) - \|r\|^2 F(x)/\varepsilon \\ &\leq -\sigma_{\min}(K_v) \|r\|^2 + \|r\| F(x) \left(1 - \frac{\|r\|}{\varepsilon}\right).\end{aligned}$$

The last term is generally positive in this region, so that nothing can be said about whether  $L$  is increasing or decreasing. In general,  $L$  may be increasing in this region so that  $\|r\|$  increases towards  $\varepsilon$ .  $\square$

*This proof is typical of robust control and recurs in various forms throughout the book—During the proof, various norm inequalities are used to manipulate  $\dot{L}$ , ending finally in the selection for the robust control term  $v(t)$  overbounding some remaining uncooperative terms. It is often required to consider different disjoint regions. The result is generally that the tracking error is bounded by some ‘small enough’ value, though not equal to zero. Contrast this with the situation in adaptive control (e.g. Theorem 3.4.2)*

**Variable Structure Robust Controller.** Another robust controller is the variable structure robust controller (Slotine 1985)

$$\begin{aligned}\tau &= \hat{f} + K_v r - v \\ v &= -(F(x) + \eta) \operatorname{sgn}(r)\end{aligned}\tag{3.4.18}$$

where  $\operatorname{sgn}(\cdot)$  is the signum function and  $F(x)$  is a known function computed using the properties in Table 3.2.1 to bound the uncertainties  $\|f - \hat{f}\|$ . The design parameter  $\eta$  is selected as a small value. This controller takes advantage of the properties of sliding mode or variable structure controllers to provide its robustness.

**Example 3.4.2 (Design and Performance of Robust Controller) :**

To implement the robust controllers just described, it is necessary to determine a bound  $F(x)$  on the functional estimation error  $\tilde{f}$ . This may be accomplished using the bounding properties in Table 3.2.1. For the 2-link planar arm, for instance, the required bounds are located in Example 3.2.2.

**a. Control Design and Bounding Function.** Assume, for instance, that a robust PD gravity controller is to be implemented. Then one has  $\hat{f} = G(q)$  and the control input (3.4.8) is

$$\tau = K_v r + G(q) - v.$$

Compare this to the computed-torque PD-gravity controller (3.3.16).

For this choice of control, according to (3.4.7) one has

$$\tilde{f} = f(x) - \hat{f} = M(q)(\ddot{q}_d + \Lambda \dot{e}) + V_m(q, \dot{q})(\dot{q}_d + \Lambda e) + F(\dot{q}).$$

Now, some norm inequalities (Chapter 2) and the bounding properties in Table 3.2.1 can be used to determine that

$$\begin{aligned} \|\tilde{f}\| &\leq \|M\| \cdot \|\ddot{q}_d + \Lambda \dot{e}\| + \|V_m(q, \dot{q})\| \cdot \|\dot{q}_d + \Lambda e\| + \|F(\dot{q})\| \\ &\leq \mu_2 \|\ddot{q}_d + \Lambda \dot{e}\| + v_B \|\dot{q}\| \cdot \|\dot{q}_d + \Lambda e\| + f_B \|\dot{q}_d - \dot{e}\| + k_B \\ &\equiv F(x), \end{aligned}$$

where the inertia matrix and Coriolis/centripetal bounds are given for the 2-link arm in Example 3.2.2 as

$$\begin{aligned} \mu_2 &= (m_1 + m_2)a_1^2 + 2m_2a_2^2 + 3m_2a_1a_2. \\ v_B &= m_2a_1a_2 \end{aligned}$$

and the friction bounds  $f_B, k_B$  depend on the arm, gears, and actuators used. In this example friction will be neglected. One recalls that  $x \equiv [e^T \ \dot{e}^T \ q_d^T \ \dot{q}_d^T]^T$ .

**b. Computer Simulation.** A function M file was written in MATLAB (1994) to simulate the robust saturation controller—this procedure is very direct. The robust controller is an easy modification of the function routine given in Example 3.3.1. The CT controller equations in Fig. 3.3.2 are simply replaced by the robust saturation controller equations (3.4.16). The lines of code implementing the robust saturation controller are given in Fig. 3.4.5. Note that this code is significantly simpler than either the CT controller in Example 3.3.1 or the adaptive controller in Example 3.4.1. No dynamics are contained in the robust controller so that additional states for ode23 are not needed.

In typical robust controllers, there are no controller dynamics so that the performance does not improve with time. However, with good designs (and large enough control gains) the errors are bounded so that they are small enough. Typical plots are like those in Fig. 3.4.6, where the errors are always small, though nonzero, but do not become smaller with time.  $\square$

### 3.4.5 Learning Control

*In many industrial applications robot manipulators are used to perform the same task repeatedly, such as in spray painting, short assembly operations, component insertion, and so on. In such repetitive motion situations, information from one iteration can be recorded and used to improve the performance on the next iteration.*

```
% file robrob.m, to be called by MATLAB routine ode23

function xdot= robrob(t,x) ;

% compute desired trajectory as in Fig. 3.3.2

% Use period= 2*pi ; amp1= 1 ; amp2= 1 ;

% Robust control input

m1= 0.8 ; m2= 2.3 ; a1= 1 ; a2= 1 ; g= 9.8 ; % arm parameters
Kv= 20*eye(2) ; lam= 5*eye(2); eps= 0.1 ; % controller parameters

% tracking errors
e= qd - [x(1) x(2)]' ;
ep= qdp - [x(3) x(4)]' ;
r= ep + lam*e ;

% compute bounding function
mu2= (m1+m2)*a1^2 + 2*m2*a2^2 + 3*m2*a1*a2 ;
vB= m2*a1*a2 ;
fsig= qdp + lam*e ;
fsigp= qdpp + lam*ep ;
F= mu2*norm(fsigp) + vB*norm([x(3) x(4)]')*norm(fsig) ;

% control torques.
G= [(m1 + m2)*g*a1*cos(x(1)) + m2*g*a2*cos(x(1) + x(2))
      m2*g*a2*cos(x(1) + x(2))] ;
div= max([norm(r) eps])' ;
v= -r*F/div ;
tau= Kv*r + G - v ;
tau1= tau(1) ; tau2= tau(2) ;

% ROBOT ARM DYNAMICS from Fig. 3.3.2
```

Figure 3.4.5: Robust controller.

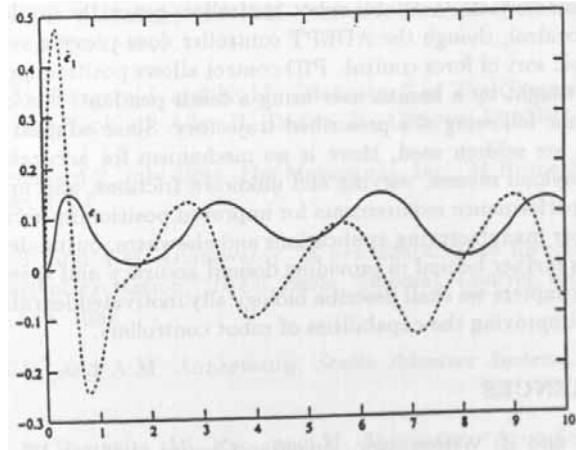


Figure 3.4.6: Typical behavior of robust controller.

This is termed repetitive motion learning control. Using the filtered error approach of section 3.4.1, it is direct to derive the learning controller of Sadegh et al. (1990) for the robot arm (3.4.1).

Let  $\ell = 1, 2, \dots$  denote the iteration number of the trajectory repetition. Then, using information from the  $(\ell - 1)$ -st iteration, the controller for the  $\ell$ -th iteration is given by

$$\begin{aligned}\tau_\ell &= \hat{f}_\ell + K_v r - v \\ v &= -(K_p e + K_s \|e\|^2 r) \\ \hat{f}_\ell &= \hat{f}_{\ell-1} + K_L r\end{aligned}\tag{3.4.19}$$

where the filtered error is  $r = \dot{e} + \Lambda e$  and the tracking error is  $e = q_d - q$ , with  $q_d(t)$  the specified trajectory to be followed repeatedly. The gains  $K_v, K_p, K_s$  are positive diagonal design matrices, and  $K_L$  is a positive diagonal learning gain matrix. The function  $\hat{f}_\ell$  is a learning term that uses its value on the previous iteration to improve on an estimate for the nonlinear function appearing in the error analysis. The functions  $e(t), r(t)$  appearing in the equations may be interpreted as  $e_\ell(t), r_\ell(t)$ , the errors occurring on the  $\ell$ -th repetition.

### 3.5 CONCLUSIONS

In this chapter we have provided the basics for robot servo-controller design by summarizing kinematics, Jacobians, and dynamics for serial-link robot manipulators. Computed-torque control was reviewed; this technique is important because it provides a general framework for unifying a variety of robot control algorithms. It was shown how to simulate robot controllers using a digital computer. We discussed classical joint control, digital control, adaptive control, robust control, and repetitive learning control. Extensive examples were provided to show the sort of behavior that can be expected from each sort of controller.

*Standard commercially available robot controllers generally employ PD or PID classical joint control, though the ADEPT controller does provide velocity feedforward and a basic sort of force control. PID control allows positioning at a sequence of ‘via’ points taught by a human user using a teach pendant, but does not allow accurate dynamic following of a prescribed trajectory. Since adaptation and learning techniques are seldom used, there is no mechanism for accurately correcting for unknown payload masses, varying and unknown frictions, and unknown motor dynamics. As performance requirements for improved positioning accuracy increase in semiconductor manufacturing applications and elsewhere, outmoded PID control technology lags further behind in providing desired accuracy and speed of response. In subsequent chapters we shall describe biologically motivated learning adaptation techniques for improving the capabilities of robot controllers.*

### 3.6 REFERENCES

- Åström, K.J. and B. Wittenmark, *Adaptive Control*, Addison-Wesley, Reading, MA, 1989.
- Craig, J., *Adaptive Control of Mechanical Manipulators*, Addison-Wesley, Reading, MA, 1985.
- Colbaugh, R., H. Seraji, and K. Glass, “A new class of adaptive controllers for robot trajectory tracking,” *J. Robotic Systems*, vol. 11, no. 8, pp. 761-772, 1994.
- Colbaugh, R., K. Glass, and H. Seraji, “Performance-based adaptive tracking control of robot manipulators,” *J. Robotic Systems*, vol. 12, no. 8, pp. 517-530, 1995.
- Corless, M., “Tracking controllers for uncertain systems: applications to a Manutec R3 robot,” *J. Dynam. Sys. Meas. Control*, vol. 111, pp. 609-618, Dec. 1989.
- Dawson, D., Z. Qu, F.L. Lewis, and J.F. Dorsey, “Robust control for the tracking of robot motion,” *Int. J. Control*, vol. 52, no. 3, pp. 581-595, 1990.
- Goodwin, G.C., and K.S. Sin, *Adaptive Filtering, Prediction, and Control*, Prentice-Hall, NJ, 1984.
- Ioannou, P.A., and P.V. Kokotovic, “Instability analysis and improvement of robustness of adaptive control,” *Automatica*, vol. 20, no. 5, pp. 583-594, 1984.
- Ioannou, P.A., and J. Sun, *Robust Adaptive Control*, Prentice-Hall, NJ, 1996.
- Kreisselmeier, G., and B. Anderson, “Robust model reference adaptive control,” *IEEE Trans. Automat. Control*, vol. AC-31, no. 2, pp. 127-133, Feb. 1986.
- Landau, Y.D., *Adaptive Control: The Model Reference Approach*, Marcel Dekker, Inc., 1979.
- Lewis, F.L., *Applied Optimal Control and Estimation*, Prentice Hall, New Jersey, 1992.

*Lewis, F.L., C.T. Abdallah, and D.M. Dawson, Control of Robot Manipulators, Macmillan, New York, 1993.*

*Lewis, F.L., M. Fitzgerald, and K. Liu "Robotics," in The Computer Science and Engineering Handbook, ed. Allen B. Tucker, Jr., Chapter 33, CRC Press, 1997.*

*MATLAB version 4.2, July 1994, The Mathworks, Inc., 24 Prime Park Way, Natick, MA 01760, USA.*

*Narendra, K.S., and A.M. Annaswamy, "A new adaptive law for robust adaptation without persistent excitation," IEEE Trans. Automat. Control, vol. AC-32, no. 2, pp. 134-145, Feb. 1987.*

*Narendra, K.S., and A.M. Annaswamy, Stable Adaptive Systems, Prentice-Hall, NJ, 1989.*

*Sadegh, N., R. Horowitz, W. Kao, and M. Tomizuka, "A unified approach to the design of adaptive and repetitive controllers for robot manipulators," Trans. ASME, vol. 112, pp. 618-629, Dec. 1990.*

*Sastry, S., and M. Bodson, Adaptive Control, Prentice-Hall, NJ, 1989.*

*Slotine, J.-J.E., "The robust control of robot manipulators," Int. J. Robotics Research, vol. 4, no. 4, pp. 49-64, 1985.*

*Slotine, J.-J., "Putting physics in control: the example of robotics," Control Systems Magazine, vol. 8., pp. 12-15, Dec. 1988.*

*Slotine, J.-J.E., and W. Li, Applied Nonlinear Control, Prentice-Hall, New Jersey, 1991.*

*Spong, M.W., J.S. Thorp, and J.M. Kleinwaks, "Robust microprocessor control of robot manipulators," Automatica, vol. 23, no. 3, pp. 373-379, 1987.*

*Spong, M.W., and M. Vidyasagar, Robot Dynamics and Control, Wiley, New York, 1989.*

### 3.7 PROBLEMS

#### Section 3.2

**Problem 3.2-1 : Dynamics of 2-Link Polar Arm.** A 2-link polar RP arm is shown in Fig. 3.7.1. The motion of the arm is in the plane of the drawing. (a) Use Lagrange's equation to derive the dynamics of this robot arm. (b) Find the bounds of the properties in Table 3.2.1 in a suitable norm. (c) Find the correct Coriolis/centripetal matrix  $V_m(q, \dot{q})$  so that Property P3 holds. (d) Find the skewsymmetric matrix  $S(q, \dot{q})$ .

**Problem 3.2-2 : Dynamics of 3-Link Cylindrical Arm.** A 3-link cylindrical RPP arm is shown in Fig. 3.7.2. (a) Use Lagrange's equation to show that the

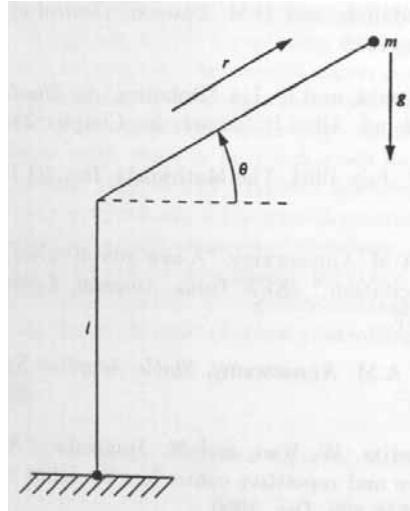


Figure 3.7.1: Two-link polar robot arm.

dynamics of this robot arm are given by

$$\begin{bmatrix} J + m_2 r^2 & 0 & 0 \\ 0 & m_1 + m_2 & 0 \\ 0 & 0 & m_2 \end{bmatrix} \begin{bmatrix} \ddot{\theta} \\ \ddot{h} \\ \ddot{r} \end{bmatrix} + \begin{bmatrix} 2m_2 r \dot{r} \dot{\theta} \\ 0 \\ m_2 r \dot{\theta}^2 \end{bmatrix} + \begin{bmatrix} 0 \\ (m_1 + m_2)gh \\ 0 \end{bmatrix} = \begin{bmatrix} \tau_1 \\ f_2 \\ f_3 \end{bmatrix},$$

where  $J$  is the inertia of the base link,  $\tau_1$  is a torque, and  $f_2, f_3$  are forces. (b) Find the bounds of the properties in Table 3.2.1 in a suitable norm. (c) Find the correct Coriolis/centripetal matrix  $V_m(q, \dot{q})$  so that Property P3 holds. (d) Find the skew-symmetric matrix  $S(q, \dot{q})$ .

**Problem 3.2-3 : Cartesian Dynamics.** Derive the Cartesian dynamics (3.2.10).

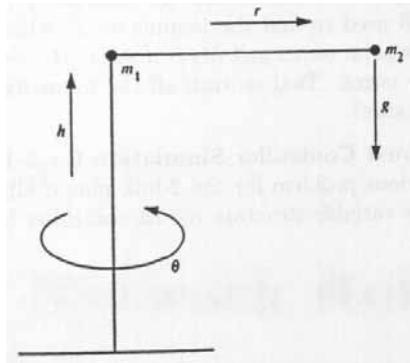


Figure 3.7.2: Three-link cylindrical robot arm.

*Prove that properties P1-P6 in Table 3.2.1 hold for the Cartesian dynamics.*

**Problem 3.2-4 : Dynamics of Robot Arm with Actuators.** *The dynamics of  $n$  electric motors whose electrical time constants are very fast and can be ignored with respect to the mechanical time constants can be written as*

$$J_M \ddot{q}_M + B_M \dot{q}_M + F_M + R\tau = K_M v,$$

*with  $q_M$  the  $n$ -vector of individual motor angles  $q_{M_i}$ . The motor inertias are in the diagonal matrix  $J_M$ , and  $B_M$  is given by the rotor damping constant and back emf. The torque supplied to the  $n$  joints of a robot arm is  $\tau \in \mathbb{R}^n$ , and  $R = \text{diag}\{r_i\}$  with  $r_i$  the gear ratios of the  $n$  motor/robot joint couplings. The control input is the motor voltage vector  $v(t)$  and  $K_M$  is the diagonal matrix of motor torque constants. (a) Derive the dynamics of a robot arm with motor actuators (3.2.11). (Note that  $q = Rq_M$ .) (b) Prove that properties P1-P6 in Table 3.2.1 hold for these dynamics.*

### Section 3.3

**Problem 3.3-1 : Error Dynamics of Approximate CT Controller.** *Derive the error dynamics (3.3.12).*

**Problem 3.3-2 : Controller Simulation for 3-Link Cylindrical Arm.** *For the 3-link cylindrical arm described in the Problems for Section 3.2, run MATLAB simulations for: (a) PD CT control. (b) PID CT control. (c) PD Gravity control. (d) Classical joint control. Select suitable control gains in each case and compare the performance of the different types of controllers. You should be able to write a general purpose MATLAB function routine that simulates all these controllers with suitable minor modification.*

**Problem 3.3-3 : Controller Simulation for 2-Link Planar Elbow Arm.** *For the 2-link planar elbow arm described in Example 3.2.1 run MATLAB simulations for the following cases. Use a sinusoidal desired trajectory with a small period. (a) Classical joint control. (b) PD gravity control. (c) PD gravity control with acceleration feedforward. (That is, include the acceleration term in (3.3.4).) (d) Full PD CT control. Isolate the improvements yielded by each control term added. You should see that major improvements occur on the addition of gravity and acceleration feedforward.*

### Section 3.4

**Problem 3.4-1 : Derivation of Regression Matrix.** *Derive the regression matrix  $W(x)$  used in Example 3.4.1.*

**Problem 3.4-2 : Adaptive Controller Simulation for 3-Link Cylindrical Arm.** *Simulate the adaptive controller (3.4.13)-(3.4.14) for the 3-link cylindrical arm described in the Problems for Section 3.2.*

**Problem 3.4-3 : Robust Controller Simulation for 3-Link Cylindrical Arm.** *(a) Simulate the saturation-based robust controller (3.4.16) for the 3-link cylindrical arm described in the Problems for Section 3.2. Use a sinusoidal desired trajectory with a small period. Assume that the estimate  $\hat{f}$  is taken simply as the*

gravity terms. You will need to find the bounds on  $\tilde{f}$ , which corresponds to the neglected Coriolis/centripetal terms and  $M(q)$  matrix. (b) Now, simulate only the PD control and gravity terms. That is, turn off the robustifying term  $v(t)$ . What happens to the performance?

**Problem 3.4-4 : Robust Controller Simulation for 2-Link Planar Elbow Arm.** Repeat the previous problem for the 2-link planar elbow arm described in Example 3.2.1 using the variable structure robust controller (3.4.18).

## Chapter 4

# Neural Network Robot Control

*Most commercially available robot controllers implement some variety of PID control algorithm, though the ADEPT does have a simple version of velocity feedforward and a basic force control option. PID control allows accuracy acceptable for many applications at a set of 'via' points specified by a human user using the teach pendant, but it does not allow accurate dynamic trajectory following between the via points. As performance requirements on speed and accuracy of motion increase in today's manufacturing environments, PID controllers lag further behind in providing adequate robot manipulator performance. Since most commercial controllers do not use any adaptive or learning capability, control accuracy is lost when unknown frictions change, for force control in surface finishing applications, and elsewhere. In this chapter we show how to use biologically inspired control techniques to remedy these problems.*

*In Chapter 3 were introduced the robot dynamics and an approximation technique for controller design based on the filtered error. Using this technique, we showed how to design adaptive, robust, and learning controllers. A serious problem in using adaptive control in robotics is the requirement for the assumption of linearity in the unknown system parameters (Craig 1985):*

$$f(x) = R(x)\xi \quad (4.0.1)$$

*where  $f(x)$  is a nonlinear robot function,  $R(x)$  is a regression matrix of known robot functions and  $\xi$  is a vector of unknown parameters (e.g. masses and friction coefficients). This is an assumption that restricts the sorts of systems amenable to control. Some forms of friction, for instance, are not linear in the parameters (LIP). Moreover, this LIP assumption requires one to determine the regression matrix for the system; this can involve tedious computations, and a new regression matrix must be computed for each different robot manipulator. Some recent adaptive control techniques for robotics avoid the requirement for LIP (Colbaugh et al. 1994, 1995). Hyperstability and some model-reference techniques in adaptive control (Landau 1979), for instance, do not require LIP, though they have not been applied in a rigorous manner to robot tracking control.*

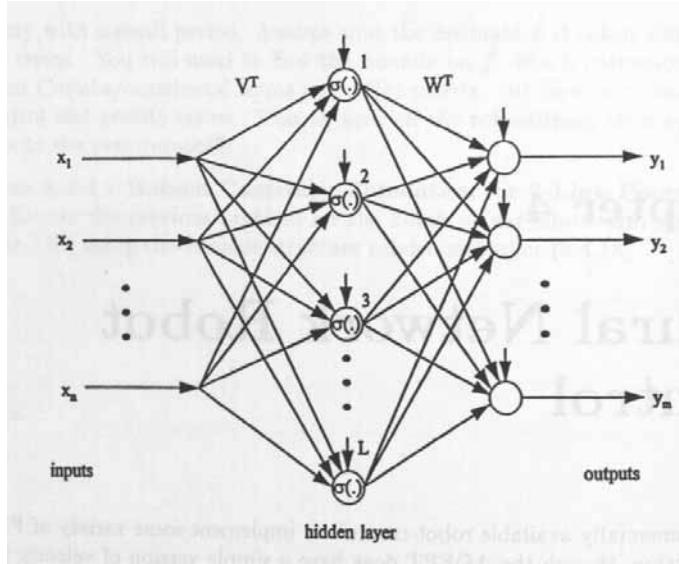


Figure 4.0.1: Two-layer neural net.

In Chapter 1 we saw that neural networks (NN) possess some very important properties, including a universal approximation property (Cybenko 1989, Hornik et al. 1989, Barron 1993) where, for every smooth function  $f(x)$ , there exists a neural network such that

$$f(x) = W^T \sigma(V^T x) + \varepsilon \quad (4.0.2)$$

for some weights  $W, V$ . This approximation holds for all  $x$  in a compact set  $S$ , and the functional estimation error  $\varepsilon$  is bounded so that

$$\|\varepsilon\| < \varepsilon_N, \quad (4.0.3)$$

with  $\varepsilon_N$  a known bound dependent on  $S$ . The approximating weights may be unknown, but the NN approximation property guarantees that they exist. In contrast with the adaptive control LIP requirement, which is an assumption that restricts the sorts of system one can deal with, the result (4.0.2) is a property that holds for all smooth functions  $f(\cdot)$ . The two-layer NN required for approximation is shown in Fig. 4.0.1.

In this chapter we propose to use a filtered-error-based approach, employing a NN to approximate unknown nonlinear functions in the robot arm dynamics, thereby overcoming some limitations of adaptive control (Lewis et al. 1995-1996, Yeşildirek 1994). The main result of this chapter is the controller presented in Section 4.3 and displayed in Table 4.3.2. Instead of requiring knowledge of the system structure, as in adaptive control, NN controls design uses some deep structural properties of the system, including skew symmetry and passivity, to guarantee good performance. The NN will be designed to adapt its weights on-line to learn the unknown portion of the dynamics. The study will be for rigid-link robot arms. In Chapter 5 NN

controllers will be designed for several more complex robotic systems including arms with link flexibility and motor coupling dynamics, and for force control tasks.

In this book we assume all the system states are measurable. If only some states are measurable, corresponding to the case of output feedback, then an additional dynamical or recurrent NN is required to estimate the unmeasured states (Kim and Lewis 1996).

Overcoming requirements for linearity in the tunable parameters has been a major obstacle to continued development of adaptive control techniques. In this chapter we overcome this problem, providing tuning rules for a set of NN weights, some of which appear in a nonlinear fashion. In fact, the two-layer NN required in (4.0.2) is nonlinear in the first-layer weights  $V$ .

The nonlinear dependence of the two-layer NN makes for some difficulties in designing a NN controller that adapts its weights on-line. Therefore, in Section 4.2 we first design a controller based on a simplified one-layer NN. In Section 4.3 are given the main results of this chapter—the general two-layer NN controller. Some implementation considerations are given in Section 4.4, including partitioned NN of simplified structure and signal preconditioning to improve the performance of the NN. The NN controllers have important passivity properties that make them robust to disturbances and unmodeled dynamics; these are detailed in Section 4.5.

In open-loop NN applications such as system identification, prediction, classification, and pattern recognition, boundedness of the NN weights alone implies the stability of the overall system, since the open-loop system is assumed stable. This is why standard gradient methods (e.g. backpropagation) yielding non-increasing weight energy functions are widely applied to these types of systems. Unfortunately, in closed-loop feedback control applications, boundedness of the weights alone demonstrates very little. Thus, standard open-loop weight training algorithms do not suffice in closed-loop control systems. There, it must be guaranteed that the NN weights remain bounded and also that the tracking error remains small and all internal states remain bounded.

A foundation for neural networks in control has been provided in seminal results by Narendra et al. (1987, 1989, 1990, 1991), Werbos (1974, 1989) and others. See the Handbook of Intelligent Control (White and Sofge 1992) and Neural Networks for Control (Miller et al. 1991). Papers employing NN in robot control are too numerous to mention, but for the most part omit stability proofs and rely on ad hoc design and simulation studies. Several researchers have studied NN control and managed to prove stability (F.-C. Chen et al. 1992, 1994, Rovithakis and Christodoulou 1994, Polycarpou and Ioannou 1991, Sanner and Slotine 1991, Sadegh 1993). See as well the collection of papers in (Zbikowski and Hunt 1996). Of key importance is the backpropagation NN tuning algorithm (Werbos 1974, Rumelhart et al. 1986), which is modified in various ways in this chapter to provide stable tracking.

## 4.1 ROBOT ARM DYNAMICS AND TRACKING ERROR DYNAMICS

The dynamics of rigid-link robot arms introduced in Chapter 3 have the form

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = \tau \quad (4.1.1)$$

Table 4.1.1: Properties of Robot Arm Dynamics

- 
- P1** The inertia matrix  $M(q)$  is symmetric, positive definite, and bounded so that  $\mu_1 I \leq M(q) \leq \mu_2 I$  for all  $q(t)$ . For revolute joints, the only occurrences of the joint variables  $q_i$  are as  $\sin(q_i), \cos(q_i)$ . For arms with no prismatic joints, the bounds  $\mu_1, \mu_2$  are constants.
- P2** The Coriolis/centripetal vector  $V_m(q, \dot{q})\dot{q}$  is quadratic in  $\dot{q}$ .  $V_m$  is bounded so that  $\|V_m\| \leq v_B \|\dot{q}\|$ , or equivalently  $\|V_m \dot{q}\| \leq v_B \|\dot{q}\|^2$ .
- P3** The Coriolis/centripetal matrix can always be selected so that the matrix  $S(q, \dot{q}) \equiv \dot{M}(q) - 2V_m(q, \dot{q})$  is *skew symmetric*. Therefore,  $x^T S x = 0$  for all vectors  $x$ . This is a statement of the fact that the fictitious forces in the robot system do no work.
- P4** The friction terms have the approximate form

$$F(\dot{q}) = F_v \dot{q} + F_d(\dot{q}),$$

with  $F_v$  a diagonal matrix of constant coefficients representing the viscous friction, and  $F_d(\cdot)$  a vector with entries like  $K_{d_i} \text{sgn}(\dot{q}_i)$ , with  $\text{sgn}(\cdot)$  the signum function and  $K_{d_i}$  the coefficients of dynamic friction. These friction terms are bounded so that  $\|F(\dot{q})\| \leq f_B \|\dot{q}\| + k_B$  for constants  $f_B, k_B$ .

- P5** The gravity vector is bounded so that  $\|G(q)\| \leq g_B$ . For revolute joints, the only occurrences of the joint variables  $q_i$  are as  $\sin(q_i), \cos(q_i)$ . For revolute joint arms the bound  $g_B$  is a constant.
- P6** The disturbances are bounded so that  $\|\tau_d(t)\| \leq d_B$ .
- 

or

$$M(q)\ddot{q} + N(q, \dot{q})\dot{q} + \tau_d = \tau, \quad (4.1.2)$$

where

$$N(q, \dot{q}) \equiv V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) \quad (4.1.3)$$

is the vector of the nonlinear terms. In these dynamics,  $M(q)$  is the inertia matrix,  $V_m(q, \dot{q})$  is the Coriolis/centripetal matrix,  $F(\dot{q})$  are the friction terms,  $G(q)$  is the gravity vector, and  $\tau_d(t)$  represents disturbances. These dynamics can include actuators, as shown in Chapter 3, and can be referred to Cartesian motion coordinates. Thus, control input  $\tau(t)$  can represent torques or motor currents, etc. The rigid robot dynamics enjoy the properties in Chapter 3, which are reproduced here in Table 4.1.1.

The objective in this chapter is to make the robot manipulator follow a prescribed desired trajectory  $q_d(t)$ . Define the tracking error  $e(t)$  and filtered tracking error  $r(t)$  by

$$e = q_d - q \quad (4.1.4)$$

$$r = \dot{e} + \Lambda e \quad (4.1.5)$$

with  $\Lambda > 0$  a positive definite design parameter matrix. Since (4.1.5) is a stable system, it follows that  $e(t)$  is bounded as long as the controller guarantees that the

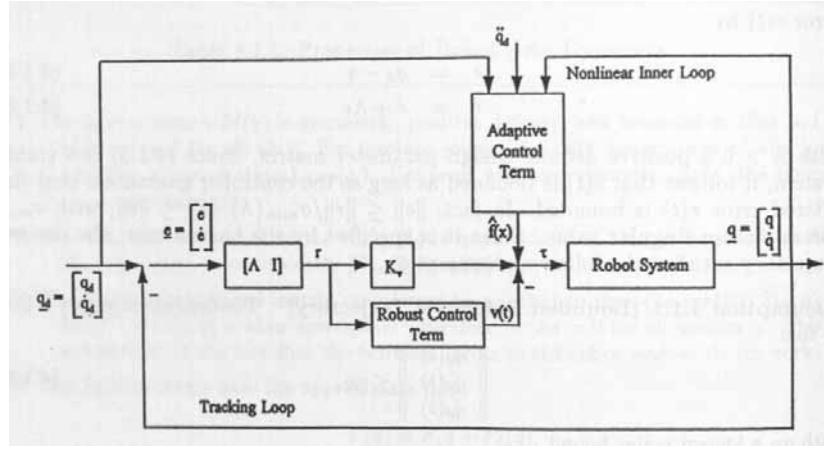


Figure 4.1.1: Filtered error approximation-based controller.

filtered error  $r(t)$  is bounded. In fact,  $\|e\| \leq \|r\|/\sigma_{min}(\Lambda)$ ,  $\|\dot{e}\| \leq \|r\|$ , with  $\sigma_{min}$  the minimum singular value. Since it is specified by the human user, the desired trajectory satisfies the following assumption.

**Assumption 4.1.1 (Bounded Reference Trajectory) :** The desired trajectory is bounded so that

$$\begin{array}{c} q_d(t) \\ \dot{q}_d(t) \\ \ddot{q}_d(t) \end{array} \leq q_B, \quad (4.1.6)$$

with  $q_B$  a known scalar bound.

As in Chapter 3, the robot dynamics are expressed in terms of the filtered error as

$$M\dot{r} = -V_m r + f(x) + \tau_d - \tau \quad (4.1.7)$$

where the unknown nonlinear robot function is defined as

$$f(x) = M(q)(\ddot{q}_d + \Lambda\dot{e}) + V_m(q, \dot{q})(\dot{q}_d + \Lambda e) + F(\dot{q}) + G(q). \quad (4.1.8)$$

One may define, for instance,

$$x \equiv [e^T \ \dot{e}^T \ q_d^T \ \dot{q}_d^T \ \ddot{q}_d^T]^T. \quad (4.1.9)$$

A general sort of approximation-based controller is derived by setting

$$\tau = \hat{f} + K_v r - v(t), \quad (4.1.10)$$

with  $\hat{f}$  an estimate of  $f(x)$ ,  $K_v r = K_v \dot{e} + K_v \Lambda e$  an outer PD tracking loop, and  $v(t)$  an auxiliary signal to provide robustness in the face of disturbances and modeling errors. The multiloop control structure implied by this scheme is shown in Fig. 4.1.1.

Using this controller, the closed-loop error dynamics are

$$M\dot{r} = -V_m r - K_v r + \tilde{f} + \tau_d + v(t), \quad (4.1.11)$$

where the function approximation error is given by

$$\tilde{f} = f - \hat{f}. \quad (4.1.12)$$

The next bound for  $x$  is needed in subsequent work.

**Lemma 4.1.1 (Bound on NN Input  $x$ )** : For each time  $t$ ,  $x(t)$  is bounded by

$$\|x\| \leq c_1 + c_2 \|r\| \leq q_B + c_0 \|r(0)\| + c_2 \|r\| \quad (4.1.13)$$

for computable positive constants  $c_0, c_1, c_2$ .

Proof:

The solution of the LTI system (4.3.4) with the initial value vector  $q(t_0)$  is

$$e(t) = e_0 e^{-\Lambda(t-t_0)} + \int_{t_0}^t e^{-\Lambda(t-\tau)} r(\tau) d\tau, \quad \forall t \geq t_0$$

where  $e_0 = q_d(t_0) - q(t_0)$ . Thus,

$$\|e\| \leq \|e_0\| + \frac{\|r(t)\|}{\sigma_{min}(\Lambda)},$$

with  $\sigma_{min}(\Lambda)$  the minimum singular value of  $\Lambda$ . The NN input can be written as

$$x = \begin{bmatrix} e \\ \dot{e} \\ q_d \\ \dot{q}_d \\ \ddot{q}_d \end{bmatrix} = \begin{bmatrix} e \\ r - \Lambda e \\ q_d \\ \dot{q}_d \\ \ddot{q}_d \end{bmatrix}.$$

Then a bound can be given as

$$\begin{aligned} \|x\| &\leq (1 + \sigma_{max}(\Lambda)) \|e\| + q_B + \|r\| \\ &\leq \{[1 + \sigma_{max}(\Lambda)] \|e_0\| + q_B\} + \left\{1 + \frac{1}{\sigma_{min}(\Lambda)} + \frac{\sigma_{max}(\Lambda)}{\sigma_{min}(\Lambda)}\right\} \|r\| \\ &= c_1 + c_2 \|r\| \end{aligned}$$

with

$$c_1 = [1 + \sigma_{max}(\Lambda)] \|e_0\| + q_B \quad (4.1.14)$$

and

$$c_2 = 1 + \frac{1}{\sigma_{min}(\Lambda)} + \frac{\sigma_{max}(\Lambda)}{\sigma_{min}(\Lambda)}. \quad (4.1.15)$$

Now, from (4.1.5) one has  $\|e\| < \|r\|/\sigma_{min}(\Lambda)$  for all  $t$ , whence one obtains that

$$c_0 = \frac{1 + \sigma_{max}(\Lambda)}{\sigma_{min}(\Lambda)}.$$

□

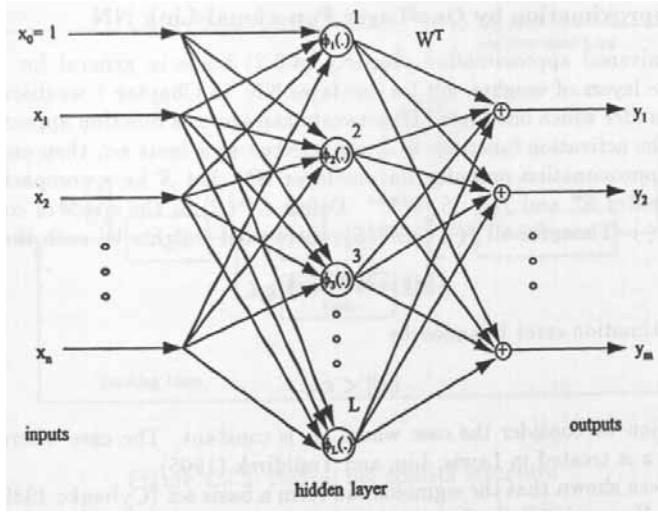


Figure 4.2.1: One-layer functional-link neural net.

## 4.2 ONE-LAYER FUNCTIONAL-LINK NEURAL NETWORK CONTROLLER

*Neural net (NN) controllers for a general serial-link rigid robot arm are developed in this chapter. The main result in this chapter is the two-layer NN controller derived in Section 4.3 and displayed in Table 4.3.2, as the controller that is the most versatile and easy to initialize. The derivation of this controller is somewhat involved since the two-layer NN is nonlinear in the tunable weights. Therefore, we develop here a controller that uses a NN with one layer of tunable weights. Though in practical situations, one would implement the two-layer NN controller if there is enough computing power, the one-layer NN controller sometimes presents an attractive alternative since it is less complex to implement.*

*The one-layer NN is shown in Fig. 4.2.1 and is described by the equation*

$$y = W^T \phi(x), \quad (4.2.1)$$

*with  $x \in \mathbb{R}^n$ ,  $\phi(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}^L$ ,  $y \in \mathbb{R}^m$ . The  $L$ -vector function  $\phi(x)$  consists of  $L$  scalar functions  $[\phi_1(x) \ \phi_2(x) \dots \phi_L(x)]^T$ . Input vector  $x$  is augmented by a first component of  $x_0 = 1$ , so that the thresholds for the first layer are included as the first column of the weight matrix  $W^T$  (see Chapter 1). In Chapter 1 we discussed several such linear-in-the-parameter (LIP) NN, including the radial basis function (RBF) and cerebellar model arithmetic computer (CMAC) nets. The one-layer NN controller case has been treated using radial basis functions by Sanner and Slotine (1991) and Polycarpou and Ioannou (1991) (where a projection algorithm was used for weight tuning), and for discrete-time systems in Sadegh (1993). The material in this section is from Lewis, Liu, and Yeşildirek (1995).*

### 4.2.1 Approximation by One-Layer Functional-Link NN

The NN universal approximation property (4.0.2) holds in general for NN with two or more layers of weights, not for one-layer NN. In Chapter 1 we discussed the conditions under which one-layer LIP networks can serve as function approximators. Indeed, if the activation functions  $\phi(x)$  are selected as a basis set, then one has the following approximation property for one-layer NN. Let  $S$  be a compact, simply connected set of  $\mathbb{R}^n$  and  $f(\cdot) : S \rightarrow \mathbb{R}^m$ . Define  $C^m(S)$  as the space of continuous functions  $f(\cdot)$ . Then, for all  $f(\cdot) \in C^m(S)$ , there exist weights  $W$  such that

$$f(x) = W^T \phi(x) + \varepsilon \quad (4.2.2)$$

with the estimation error bounded by

$$\|\varepsilon\| < \varepsilon_N. \quad (4.2.3)$$

In this section we consider the case where  $\varepsilon_N$  is constant. The case where  $\varepsilon_N$  is a function of  $x$  is treated in Lewis, Liu, and Yeşildirek (1995).

It has been shown that the sigmoids can form a basis set (Cybenko 1989, Hornik et al. 1989, Barron 1993). In Sanner and Slotine (1991) it was shown that the radial basis functions can form a basis. In Commuri and Lewis (1995) it is shown that a basis set is particularly easy to choose for CMAC NN. In Chapter 1 it was shown how to select the parameters of the functions  $\phi(\cdot)$  randomly to obtain a basis. In Section 4.4 appears more discussion on selecting a basis suitable for rigid robot arm control. We shall suppose that a suitable basis has been selected. Then, (4.2.2) holds for any smooth function; that is, the one-layer NN possesses a universal approximation property.

Barron [1993] has shown that for linear-in-the-parameters sums of the form (4.2.2), the approximation error can never be made smaller than order  $1/L^{2/n}$ . This lower bound can be overcome for nonlinear-in-the-parameter approximations such as (4.0.2), which we use in Section 4.3. However, despite the lower bound on  $\varepsilon_N$  for the FLNN, the closed-loop performance of the NN controller presented in this section is good since the tracking error will be found to be bounded by a term like  $\varepsilon_N/K_{v_{min}}$ , with  $K_{v_{min}}$  the smallest PD control gain, which can be made large to improve tracking accuracy.

**Advantage of NN Control Over Adaptive Control.** The contrast between the property (4.2.2) and the adaptive control LIP assumption (4.0.1) should be clearly understood. Both are linear in the tunable parameters, but the former is linear in the tunable NN weights, while the latter is linear in the unknown system parameters. The former holds for all functions  $f(x)$  in  $C^m(S)$ , while the latter holds only for a specific function  $f(x)$ . In the NN property the same basis set  $\phi(x)$  suffices for all  $f(\cdot) \in C^m(S)$ , while in the LIP assumption the regression matrix  $R(x)$  depends on  $f(x)$  and must be recomputed for each different  $f(x)$ . That is, for each different type of robot arm, one must recompute  $R(x)$ . Therefore, the one-layer NN controller is significantly more powerful than adaptive controllers; it provides a universal controller for all rigid-link robot arms.

One must notice as well that (4.2.2) includes a modeling approximation error term; the result will turn out to be a more robust control scheme. Robust adaptive

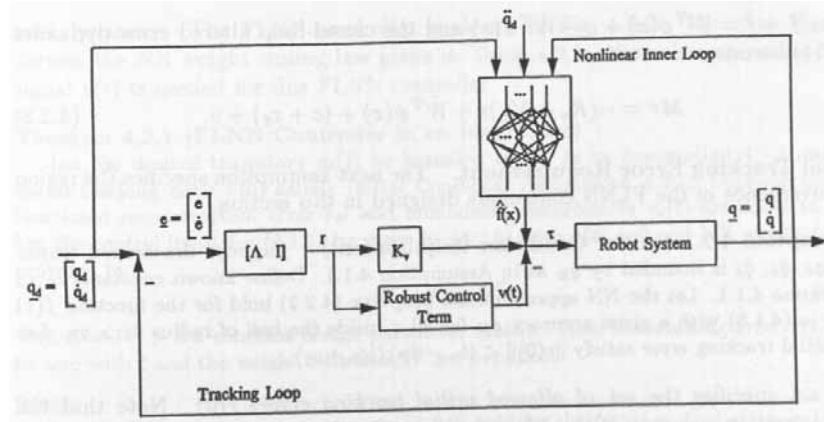


Figure 4.2.2: Neural net control structure.

control techniques are available for handling modeling errors (Narendra and Annaswamy 1989), including the sigma-modification (Ioannou and Kokotovic 1984), the  $e$ -modification (Narendra and Annaswamy 1987), and dead-zone techniques (Peterson and Narendra 1982, Kreisselmeier and Anderson 1986).

#### 4.2.2 NN Controller and Error System Dynamics

Suppose now that a FLNN is used to approximate the nonlinear robot function (4.1.8) according to (4.2.2), with  $W$  the ideal approximating weights. The ideal weights are unknown and may even be nonunique. Assume they are constant and bounded so that

$$\|W\|_F \leq W_B, \quad (4.2.4)$$

with the bound  $W_B$  known. The Frobenius norm  $\|\cdot\|_F$  was defined in Chapter 2. Then, an estimate of  $f(x)$  is given by

$$\hat{f}(x) = \hat{W}^T \phi(x), \quad (4.2.5)$$

with  $\hat{W}$  the current actual values of the FLNN weights as provided by the tuning algorithm to be specified. Then the control law (4.1.10) becomes

$$\tau = \hat{W}^T \phi(x) + K_v r - v. \quad (4.2.6)$$

The proposed NN control structure is shown in Fig. 4.2.2, where  $\underline{q} \equiv [q^T \ \dot{q}^T]^T$ ,  $\underline{e} \equiv [e^T \ \dot{e}^T]^T$ .

It is now necessary to show how to tune the NN weights  $\hat{W}$  on-line so as to guarantee stable tracking. The tuning algorithm found will presumably modify the actual weights  $\hat{W}$  so that they become close to the ideal weights  $W$ , which are unknown. To this end, define the weight deviations or weight estimation errors as

$$\tilde{W} = W - \hat{W}. \quad (4.2.7)$$

Then,  $f - \hat{f} = W^T \phi(x) + \varepsilon - \hat{W}^T \phi(x)$  and the closed-loop filtered error dynamics (4.1.11) becomes

$$M\dot{r} = -(K_v + V_m)r + \tilde{W}^T \phi(x) + (\varepsilon + \tau_d) + v. \quad (4.2.8)$$

**Initial Tracking Error Requirement.** The next assumption specifies the region of convergence of the FLNN controllers designed in this section.

**Assumption 4.2.1 (Initial Condition Requirement)** : Suppose the desired trajectory  $q_d$ ,  $\dot{q}_d$ ,  $\ddot{q}_d$  is bounded by  $q_B$  as in Assumption 4.1.1. Define known constants  $c_0, c_2$  by Lemma 4.1.1. Let the NN approximation property (4.2.2) hold for the function  $f(x)$  given in (4.1.8) with a given accuracy  $\varepsilon_N$  for all  $x$  inside the ball of radius  $b_x > q_B$ . Let the initial tracking error satisfy  $\|r(0)\| < (b_x - q_B)/(c_0 + c_2)$ .

This set specifies the set of allowed initial tracking errors  $r(0)$ . Note that the approximation accuracy of the NN determines the allowed magnitude of the initial tracking error  $r(0)$ . For a larger NN (i.e. more hidden-layer units),  $\varepsilon_N$  is small for a larger radius  $b_x$ . Thus, the allowed initial condition set  $S_r$  is larger. Likewise, a more active desired trajectory (e.g. containing higher frequency components) results in a larger acceleration  $\ddot{q}_d(t)$ , which yields a larger bound  $q_B$ , thereby decreasing  $S_r$ . It is important to note the dependence of  $S_r$  on the PD design ratio  $\Lambda$ — both  $c_0$  and  $c_2$  depend on  $\Lambda$ .

Though the initial condition requirement may seem to be cast in terms of complex inequalities, its key role is in showing the dependence of the allowed initial condition set  $S_r$  on the design parameters. The constants  $q_B, c_0, c_2, b_x$  need not be explicitly determined. In practical situations, the IC requirement merely indicates that the NN should be ‘large enough’ in terms of the number  $L$  of hidden-layer units. Therefore, in design, one would select a value for  $L$ , run a simulation to test the controller, then repeat with a larger value of  $L$ . The value of  $L$  selected for implementation on the actual system is that above which there is no appreciable increase in performance.

A key feature of the initial condition requirement is its independence of the NN initial weights. This is in stark contrast to other techniques in the literature where the proofs of stability depend on selecting some initial stabilizing NN weights, which is very difficult to do.

### 4.2.3 Unsupervised Backpropagation Weight Tuning

In this and the next subsection we give some FLNN weight tuning algorithms that guarantee the tracking stability of the closed-loop system. It is required to demonstrate that the tracking error  $r(t)$  is suitably small and that the FLNN weights  $\hat{W}$  remain bounded, for then the control  $\tau(t)$  is bounded. In this subsection we show that, under some conditions, a modified unsupervised version of the continuous-time backpropagation algorithm in Chapter 1 works.

#### 4.2.3.1 Ideal Case—Unsupervised Backpropagation Tuning of Weights

The first result details the closed-loop behavior in the idealized case of no net functional reconstruction error  $\varepsilon$  and no unmodeled disturbances  $\tau_d(t)$  in the robot arm dynamics. The FLNN controller is shown in Fig. 4.2.2. The next theorem derives

the NN weight tuning law given in Table 4.2.1. Note that no robustifying signal  $v(t)$  is needed for this FLNN controller.

**Theorem 4.2.1 (FLNN Controller in an Ideal Case) :**

Let the desired trajectory  $q_d(t)$  be bounded by  $q_B$  as in Assumption 4.1.1 and the initial tracking error  $r(0)$  satisfy Initial Condition Assumption 4.2.1. Suppose the NN functional reconstruction error  $\varepsilon_N$  and unmodeled disturbances  $\tau_d(t)$  are equal to zero. Let the control input for (4.1.1) be given by (4.2.6) with  $v(t) = 0$  and NN weight tuning provided by

$$\dot{\tilde{W}} = F\phi(x)r^T, \quad (4.2.9)$$

with  $F = F^T > 0$  a constant design parameter matrix. Then the tracking error  $r(t)$  goes to zero with  $t$  and the weight estimates  $\hat{W}$  are bounded.

Proof:

Let the NN approximation property (4.2.2) hold for the function  $f(x)$  given in (4.1.8) with a given accuracy  $\varepsilon_N$  for all  $x$  in the compact set  $S_x \equiv \{x \mid \|x\| < b_x\}$  with  $b_x > q_B$ . Define  $S_r \equiv \{r \mid \|r\| < (b_x - q_B)/(c_0 + c_2)\}$ . Let  $r(0) \in S_r$ . Then the approximation property holds.

Under the ideal case the error system is

$$M\dot{r} = -(K_v + V_m)r + \tilde{W}^T\phi(x). \quad (4.2.10)$$

Select the Lyapunov function candidate

$$L = \frac{1}{2}r^T M r^T + \frac{1}{2}\text{tr}\{\tilde{W}^T F^{-1} \tilde{W}\}. \quad (4.2.11)$$

Differentiating yields

$$\dot{L} = r^T M \dot{r} + \frac{1}{2}r^T \dot{M} r + \text{tr}\{\tilde{W}^T F^{-1} \dot{\tilde{W}}\} \quad (4.2.12)$$

whence substitution from (4.2.10) yields

$$\dot{L} = -r^T K_v r + \frac{1}{2}r^T (\dot{M} - 2V_m)r + \text{tr}\{\tilde{W}^T (F^{-1} \dot{\tilde{W}} + \phi r^T)\}. \quad (4.2.13)$$

The skew symmetry property makes the second term zero and the third term is zero if we select

$$\dot{\tilde{W}} = -F\phi r^T. \quad (4.2.14)$$

Since  $\tilde{W} = W - \hat{W}$  and  $W$  is constant, this yields the weight tuning law. Now,

$$\dot{L} = -r^T K_v r. \quad (4.2.15)$$

Since  $L > 0$  and  $\dot{L} \leq 0$  this shows stability in the sense of Lyapunov so that  $r$  and  $\tilde{W}$  (and hence  $\hat{W}$ ) are bounded. Thus,

$$\int_0^\infty -\dot{L} dt < \infty. \quad (4.2.16)$$

Now  $\ddot{L} = -2r^T K_v \dot{r}$ , and the boundedness of  $M^{-1}(q)$  and of all signals on the right-hand side of (4.2.10) verify the boundedness of  $\dot{r}$  and hence of  $\ddot{L}$ , and therefore the uniform continuity of  $\dot{L}$ . This allows us to invoke Barbalat's Lemma (Chapter 2) in connection with (4.2.16) to conclude that  $\dot{L}$  goes to zero with  $t$ , and hence that  $r(t)$  vanishes.  $\square$

Table 4.2.1: FLNN Controller for Ideal Case, or for Nonideal Case with PE

---

*Control Input:*

$$\tau = \hat{W}^T \phi(x) + K_v r, \text{ with } \phi(x) \text{ a basis}$$

*NN Weight/Threshold Tuning Algorithms:*

$$\dot{\hat{W}} = F \phi(x) r^T, \text{ with } F \text{ a positive definite design matrix}$$


---

**Unsupervised Backprop Through Time Tuning.** Note that algorithm (4.2.9) is nothing but the continuous-time backpropagation algorithm of Chapter 1 for the one-layer case. However, it is an unsupervised version of backprop in that the ideal plant output is not needed; instead the filtered error, which is easily measurable in the closed-loop system, is used in tuning the weights. It should also be realized that this is a version of the backprop through time algorithm, as the weights are continuously tuned as a function of time  $t$ .

**Weight Initialization and On-Line Tuning.** In the NN control schemes derived in this book there is no preliminary off-line learning phase. The weights are simply initialized at zero, for then Fig. 4.2.2 shows that the controller is just a PD controller. Standard results in the robotics literature (Dawson et al. 1990) show that a PD controller gives bounded errors if  $K_v$  is large enough. Therefore, the closed-loop system remains stable until the NN begins to learn. The weights are tuned on-line in real-time as the system tracks the desired trajectory. As the NN learns  $f(x)$ , the tracking performance improves. This is a significant improvement over other NN control techniques where one must find some initial stabilizing weights, generally an impossible feat for complex nonlinear systems.

#### 4.2.3.2 Unsupervised Backprop with PE Condition in Non-Ideal Case

It has just been seen that under the ideal case of no NN functional approximation errors or unmodeled disturbances, an unsupervised version of backprop through time tuning suffices to make the tracking error go to zero. However, in actual systems there are disturbances. Moreover, the NN approximation error decreases as the number of hidden-layer neurons  $L$  increases, therefore, in practical NN of limited size there generally are approximation errors. In this subsection it will be seen that if the NN approximation errors and system disturbances are not zero but bounded (see Property P6 in Table 4.1.1 and (4.2.3)), then backprop still works under an additional assumption of persistence of excitation (PE). However, now the tracking errors do not vanish, but are bounded by small enough values to guarantee good tracking performance. PE conditions are well-known in adaptive control (Sastry and Bodson 1989). PE of signals was defined in Chapter 2. The notion of PE for a one-layer NN is defined in this subsection.

A vector  $w(t) \in \mathbb{R}^p$  is said to be PE if there exist positive constants  $\delta, \alpha_1, \alpha_2$

such that

$$\alpha_1 I \leq \int_{t_0}^{t_0+\delta} w(\tau)w^T(\tau) d\tau \leq \alpha_2 I \quad (4.2.17)$$

for all  $t_0 \geq 0$ . Uniform complete observability (UCO) of time-varying systems was defined in Chapter 2. The following technical lemma is needed (Lewis Liu, and Yeşildirek 1995).

**Lemma 4.2.1 (Technical Lemma for a Special System) :**

Consider the linear time-varying system  $(0, B(t), C(t))$  defined by

$$\begin{aligned}\dot{x} &= B(t)u \\ y &= C(t)x\end{aligned}$$

with  $x \in \mathbb{R}^n, u \in \mathbb{R}^m, y \in \mathbb{R}^p$  and the elements of  $B(t)$  and  $C(t)$  piecewise continuous functions of time. Since the state transition matrix is the identity matrix, the observability gramian is

$$N(t, t_0) = \int_{t_0}^t C^T(\tau)C(\tau)d\tau.$$

Let the system be uniformly completely observable with  $B(t)$  bounded. Then if  $u(t)$  and  $y(t)$  are bounded, the state  $x(t)$  is bounded.  $\square$

The UCO of this system may be compared to a PE condition with  $w(t) = C^T(t)$ . Note that this result holds despite the less-than-desirable stability properties of the system.

The FLNN controller is shown in Fig. 4.2.2. The next result details tuning algorithms for stable tracking under nonideal conditions but with a PE condition. Uniformly ultimately bounded (UUB) stability was defined in Chapter 2. The theorem shows that the FLNN controller in Table 4.2.1 still works as long as PE is satisfied in the NN, in the sense defined in the theorem.

**Theorem 4.2.2 (FLNN Controller with PE Requirement) :**

Let the desired trajectory  $q_d(t)$  be bounded by  $q_B$  as in Assumption 4.1.1 and the initial tracking error  $r(0)$  satisfy Initial Condition Assumption 4.2.1. Let the NN reconstruction error bound  $\varepsilon_N$  and the disturbance bound  $d_B$  be constants. Let the control input for (4.1.1) be given by (4.2.6) with  $v(t) = 0$  and gain satisfying

$$K_{v_{min}} > \frac{(\varepsilon_N + d_B)(c_0 + c_2)}{b_x - q_B}. \quad (4.2.18)$$

Let NN weight tuning be provided by

$$\dot{\hat{W}} = F\phi(x)r^T, \quad (4.2.19)$$

with  $F = F^T > 0$  a constant design parameter matrix. Suppose the hidden-layer output  $\phi(x)$  is persistently exciting. Then the filtered tracking error  $r(t)$  is UUB, with a practical bound given by the right-hand side of (4.2.25), and the NN weight estimates  $\hat{W}$  are bounded. Moreover,  $r(t)$  may be kept as small as desired by increasing the gain  $K_v$ .

#### Proof:

Let the NN approximation property (4.2.2) hold for the function  $f(x)$  given in (4.1.8) with a given accuracy  $\varepsilon_N$  for all  $x$  in the compact set  $S_x \equiv \{x \mid \|x\| < b_x\}$  with  $b_x > q_B$ .

Define  $S_r \equiv \{r \mid \|r\| < (b_x - q_B)/(c_0 + c_2)\}$ . Let  $r(0) \in S_r$ . Then the approximation property holds.

Define the Lyapunov function candidate

$$L = \frac{1}{2}r^T M r^T + \frac{1}{2} \operatorname{tr}\{\tilde{W}^T F^{-1} \tilde{W}\}. \quad (4.2.20)$$

Differentiating yields

$$\dot{L} = r^T M \dot{r} + \frac{1}{2} r^T \dot{M} r + \operatorname{tr}\{\tilde{W}^T F^{-1} \dot{\tilde{W}}\} \quad (4.2.21)$$

whence substitution from (4.2.8) yields

$$\dot{L} = -r^T K_v r + \frac{1}{2} r^T (\dot{M} - 2V_m) r + \operatorname{tr}\{\tilde{W}^T (F^{-1} \dot{\tilde{W}} + \phi r^T)\} + r^T (\varepsilon + \tau_d). \quad (4.2.22)$$

The skew symmetry property makes the second term zero and the third term is zero if we select

$$\dot{\tilde{W}} = -F\phi r^T. \quad (4.2.23)$$

Since  $\tilde{W} = W - \hat{W}$  and  $W$  is constant, this yields the weight tuning law.

Now,

$$\dot{L} = -r^T K_v r + r^T (\varepsilon + \tau_d) \leq -K_{v_{min}} \|r\|^2 + (\varepsilon_N + d_B) \|r\| \quad (4.2.24)$$

with  $K_{v_{min}}$  the minimum singular value of  $K_v$ . Since  $\varepsilon_N + d_B$  is constant,  $\dot{L} \leq 0$  as long as

$$\|r\| > (\varepsilon_N + d_B)/K_{v_{min}} \equiv b_r. \quad (4.2.25)$$

Selecting the gain according to (4.2.18) ensures that the compact set defined by  $\|r\| \leq b_r$  is contained in  $S_r$ , so that the approximation property holds throughout. Therefore, the tracking error  $r(t)$  is bounded and continuity of all functions shows as well the boundedness of  $\dot{r}(t)$ .

It remains to show that  $\hat{W}$ , or equivalently  $\tilde{W}$ , is bounded. Boundedness of  $r(t)$  guarantees the boundedness of  $e(t)$  and  $\dot{e}(t)$ , whence boundedness of the desired trajectory shows  $q$  and  $\dot{q}$  are bounded. Property P2 in Table 4.1.1 then shows boundedness of  $V_m(q, \dot{q})$ . These facts guarantee boundedness of the function

$$y \equiv M\dot{r} + (K_v + V_m)r - (\varepsilon + \tau_d) \quad (4.2.26)$$

since  $M(q)$  is bounded. Therefore, the dynamics relative to  $\tilde{W}$  are given by

$$\begin{aligned} \dot{\tilde{W}} &= -F\phi r^T \\ y^T &= \phi^T \tilde{W} \end{aligned} \quad (4.2.27)$$

with  $y(t)$  and  $r(t)$  bounded. (The second equation is (4.2.8).)

Note that  $\tilde{W}$  is a matrix. Using the Kronecker product  $\otimes$  allows one to write the vector dynamics

$$\begin{aligned} \frac{d}{dt} \operatorname{vec}(\tilde{W}) &= -(I \otimes F\phi)r \\ y &= (I \otimes \phi^T) \operatorname{vec}(\tilde{W}) \end{aligned}$$

where the  $\operatorname{vec}(A)$  operator stacks the columns of a matrix  $A$  to form a vector, and one notes that  $\operatorname{vec}(z^T) = z$  for a vector  $z$ . Now, the PE condition on  $\phi$  is equivalent to PE of  $(I \otimes \phi)$ , and so to the uniform complete observability of this system, so that by Lemma 4.2.1 boundedness of  $y(t)$  and  $r(t)$  assures the boundedness of  $\tilde{W}$ , and hence of  $\hat{W}$ . (Note that boundedness of  $x(t)$  verifies boundedness of  $F\phi(x(t))$ .)  $\square$

The case where the NN estimation error bound  $\varepsilon_N$  depends on  $x$  is covered in (Lewis, Liu, and Yeşildirek 1995).

See the comments following Assumption 4.2.1 regarding the allowed initial tracking error  $r(0)$  and NN design. According to the PD gain condition (4.2.18), the required PD gains increase with  $\varepsilon_N$  and the disturbances, and also as the desired trajectory becomes more active. They decrease as the NN size increases. The following properties of the NN controller are also important.

**Weight Initialization and On-Line Tuning.** As discussed at the end of Theorem 4.2.1, the tuning algorithm is an unsupervised backpropagation through time scheme. There is no preliminary off-line learning phase. The weights are simply initialized at zero, for then Fig. 4.2.2 shows that the controller is just a PD controller, which holds the system stable until the NN begins to learn. The weights are tuned on-line in real-time as the system tracks the desired trajectory; as the NN learns  $f(x)$ , the tracking performance improves.

**Bounds on the Tracking Error and NN Weight Estimation Errors.** In the ideal case of no NN approximation error  $\varepsilon$  or unmodeled disturbances  $\tau_d(t)$ , Theorem 4.2.1 showed that the tracking error  $r(t)$  vanishes with time. In the nonideal case, a PE condition is needed, and then the tracking error does not vanish but is UUB. The right-hand side of (4.2.25) can be taken as a practical bound on the tracking error in the sense that  $r(t)$  will never stray far above it. It is important to note from this equation that the tracking error increases with the NN reconstruction error  $\varepsilon_N$  and robot disturbances  $d_B$ , yet arbitrarily small tracking errors may be achieved by selecting large gains  $K_v$ . (If  $K_v$  is taken as a diagonal matrix,  $K_{v_{\min}}$  is simply the smallest gain element.)

Note that the NN weights  $\hat{W}$  are not guaranteed to approach the ideal unknown weights  $W$  that give good approximation of  $f(x)$ . However, this is of no concern as long as  $W - \hat{W}$  is bounded, as the proof guarantees. This guarantees bounded control inputs  $\tau(t)$  so that the tracking objective can be achieved.

#### 4.2.4 Augmented Unsupervised Backpropagation Tuning— Removing the PE Condition

In adaptive control the possible unboundedness of the weight (e.g. ‘parameter’) estimates when PE fails to hold is known as ‘parameter drift’. This phenomenon has been called ‘weight overtraining’ in the NN literature. The PE condition in Theorem 4.2.2 is meant to ensure that drift does not occur. To correct this problem without requiring the PE condition, one may modify the NN weight tuning algorithm using techniques from adaptive control, including  $\sigma$ -modification (Ioannou and Kokotovic 1984),  $e$ -modification (Narendra and Annaswamy 1987) or dead-zone techniques (Kreisselmeier and Anderson 1986). Lifting of the PE condition results in a more robust NN controller that is stable under a wide variety of unmodeled dynamics and unforeseen situations.

The FLNN controller structure is shown in Fig. 4.2.2. The next theorem derives the tuning law for the FLNN controller in Table 4.2.2 that does not require PE. It is found necessary to augment the tuning law by an extra term. The proof relies on

Table 4.2.2: FLNN Controller with Augmented Tuning to Avoid PE

*Control Input:*

$$\tau = \hat{W}^T \phi(x) + K_v r, \text{ with } \phi(x) \text{ a basis}$$

*NN Weight/Threshold Tuning Algorithms:*

$$\dot{\hat{W}} = F\phi(x)r^T - \kappa F\|r\|\hat{W}$$

*Design parameters:*  $F$  a positive definite matrix and  $\kappa > 0$  a small parameter.

an extension to Lyapunov theory. The disturbance  $\tau_d$  and NN reconstruction error  $\varepsilon$  make it impossible to show that the Lyapunov derivative is nonpositive for all  $r(t)$  and weight values. In fact, it is only possible to show that  $\dot{L}$  is negative outside a compact set in the state space. This, however, allows one to conclude boundedness of the tracking error and the neural net weights. In fact, explicit bounds are discovered during the proof.

**Theorem 4.2.3 (Augmented NN Weight Tuning Algorithm) :**

Given the hypotheses of Theorem 4.2.2, assume the ideal NN target weights are bounded by  $W_B$  as in (4.2.4). Let the control input for the robot arm be given by

$$\tau = \hat{W}^T \phi(x) + K_v r \quad (4.2.28)$$

with gain satisfying

$$K_{v_{min}} > \frac{(\kappa W_B^2/4 + \varepsilon_N + d_B)(c_0 + c_2)}{b_x - q_B}. \quad (4.2.29)$$

Let the weight tuning be modified as

$$\dot{\hat{W}} = F\phi r^T - \kappa F\|r\|\hat{W}, \quad (4.2.30)$$

with  $F = F^T > 0$  and  $\kappa > 0$  a small design parameter. Make no assumptions of any sort of PE requirements on  $\phi(x)$ . Then the filtered tracking error  $r(t)$  and the NN weight estimates  $\hat{W}(t)$  are UUB with practical bounds given respectively by the right-hand sides of (4.2.33) and (4.2.34). Moreover, the tracking error may be made as small as desired by increasing the tracking gain  $K_v$ .

Proof:

Let the NN approximation property (4.2.2) hold for the function  $f(x)$  given in (4.1.8) with a given accuracy  $\varepsilon_N$  for all  $x$  in the compact set  $S_x \equiv \{x \mid \|x\| < b_x\}$  with  $b_x > q_B$ . Define  $S_r \equiv \{r \mid \|r\| < (b_x - q_B)/(c_0 + c_2)\}$ . Let  $r(0) \in S_r$ . Then the approximation property holds.

Select the Lyapunov function candidate (4.2.20) and obtain (4.2.22). Then, using tuning rule (4.2.30) yields

$$\dot{L} = -r^T K_v r + \kappa \|r\| \operatorname{tr}\{\tilde{W}^T (W - \tilde{W})\} + r^T (\varepsilon + \tau_d). \quad (4.2.31)$$

Since

$$\operatorname{tr}\{\tilde{W}^T (W - \tilde{W})\} = \langle \tilde{W}, W \rangle_F - \|\tilde{W}\|_F^2 \leq \|\tilde{W}\|_F \|W\|_F - \|\tilde{W}\|_F^2,$$

with  $\|\cdot\|_F$  the Frobenius norm (Chapter 2), there results

$$\begin{aligned}\dot{L} &\leq -K_{v_{min}}\|r\|^2 + \kappa\|r\|\cdot\|\tilde{W}\|_F(W_B - \|\tilde{W}\|_F) + (\varepsilon_N + d_B)\|r\| \\ &= -\|r\| [K_{v_{min}}\|r\| + \kappa\|\tilde{W}\|_F(\|\tilde{W}\|_F - W_B) - (\varepsilon_N + d_B)],\end{aligned}\quad (4.2.32)$$

which is negative as long as the term in braces is positive. Completing the square yields

$$\begin{aligned}&K_{v_{min}}\|r\| + \kappa\|\tilde{W}\|_F(\|\tilde{W}\|_F - W_B) - (\varepsilon_N + d_B) \\ &= \kappa(\|\tilde{W}\|_F - W_B/2)^2 - \kappa W_B^2/4 + K_{v_{min}}\|r\| - (\varepsilon_N + d_B)\end{aligned}$$

which is guaranteed positive as long as

$$\|r\| > \frac{\kappa W_B^2/4 + (\varepsilon_N + d_B)}{K_{v_{min}}} \equiv b_r \quad (4.2.33)$$

or

$$\|\tilde{W}\|_F > W_B/2 + \sqrt{\kappa W_B^2/4 + (\varepsilon_N + d_B)/\kappa} \equiv b_W. \quad (4.2.34)$$

Thus,  $\dot{L}$  is negative outside a compact set. Selecting the gain according to (4.2.29) ensures that the compact set defined by  $\|r\| \leq b_r$  is contained in  $S_r$ , so that the approximation property holds throughout. This demonstrates the UUB of both  $\|r\|$  and  $\|\tilde{W}\|_F$ .  $\square$

*See the remarks at the end of Theorem 4.2.2 about the region of convergence and the required PD gain magnitudes (4.2.29).*

**Weight Initialization and On-Line Learning.** *The remarks following Theorem 4.2.2 are valid here as well. The NN weights may be initialized at zero, and stability will be provided by the outer tracking loop until the NN learns. This means that there is no off-line learning phase, but NN learning occurs in real-time.*

**Discussion on the NN Tuning Rules.** *Note that PE is not needed to establish the bounds on  $\tilde{W}$  with the modified weight tuning algorithm. The importance of the  $\kappa$  term added to the NN weight tuning algorithm is that it adds a quadratic term in  $\|\tilde{W}\|_F$  in (4.2.32), so that it is possible to establish that  $\dot{L}$  is negative outside a compact set in the  $(\|r\|, \|\tilde{W}\|_F)$  plane (Narendra and Annaswamy 1987). The  $\kappa$  term is known in adaptive control as Narendra's  $e$ -modification. Its function is to make the tuning law robust to unmodelled dynamics so that the PE condition is not needed. In (Polycarpou and Ioannou 1991) a projection algorithm is used to keep the weights bounded. In (Chen et al. 1992, 1994) a deadzone technique is employed.*

*The first term in the augmented tuning algorithm (4.2.30) is an unsupervised backprop through time term.*

**Tracking Error and NN Weight Bounds.** *The right-hand sides of (4.2.33) and (4.2.34) respectively may be taken as practical bounds on the tracking error and NN weight errors in the sense that excursions beyond these bounds will be very small. Note, moreover, from the former that arbitrarily small tracking error bounds may be achieved by selecting large control gains  $K_v$ . On the other hand, the NN weight error is fundamentally bounded by  $W_B$ , the known bound on the ideal weights  $W$ . The tuning parameter  $\kappa$  offers a design tradeoff between the relative eventual magnitudes of  $\|r\|$  and  $\|\tilde{W}\|_F$ ; a smaller  $\kappa$  yields a smaller  $\|r\|$  and a larger  $\|\tilde{W}\|_F$ , and vice versa.*

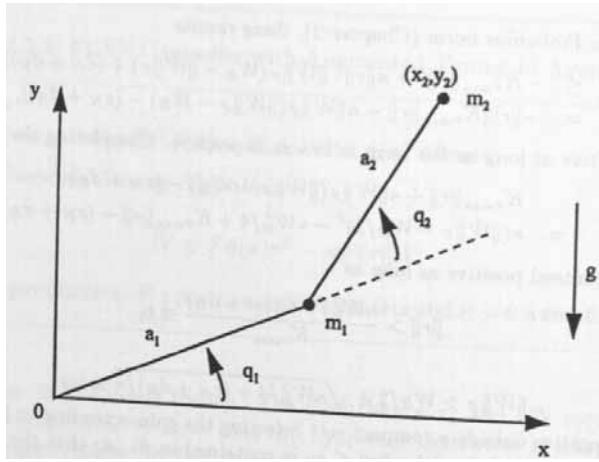


Figure 4.2.3: Two-link planar elbow arm.

#### 4.2.5 Functional-Link NN Controller Design and Simulation Example

The FLNN controller is shown in Fig. 4.2.2. We should like to illustrate this control scheme using the weight tuning laws given in Table 4.2.1 and Table 4.2.2, comparing the performance to a standard adaptive controller and a PD controller. The design ease of the FLNN controllers will be evident; they require no knowledge of the system dynamics, not even their structure which is needed for adaptive control.

##### Example 4.2.1 (FLNN Control of two-Link Robot Arm)

The planar two-link revolute arm in Fig. 4.2.3 is used extensively in the literature for simulation of nonlinear controllers. The dynamics were given in Chapter 3 and are fairly complicated. We took the arm parameters as  $a_1 = a_2 = 1$  m,  $m_1 = 0.8$  kg,  $m_2 = 2.3$  kg, and selected the desired trajectory  $q_{1d}(t) = \sin(t)$ ,  $q_{2d}(t) = \cos(t)$ .

**a. Adaptive Controller—Baseline Design.** In Chapter 3 an adaptive controller was designed for this robot arm. It was found that the adaptive controller required the computation of a complex regression matrix. It was found that the performance of the adaptive controller was very bad if any elements of this regression matrix were not known, corresponding to *unmodeled dynamics*.

**b. Functional-Link NN Controller With Backprop Weight Tuning.** A MATLAB M file was written to simulate the NN controller. It is very similar to the code given in the adaptive control example in Chapter 3. For the NN controller, *all the dynamics are unmodeled* as the controller requires *no knowledge of the system dynamics*. First, the performance of the NN controller in Table 4.2.1 that uses unsupervised backprop tuning was simulated. The controller parameters were taken as  $K_v = \text{diag}\{20, 20\}$ ,  $F = \text{diag}\{50, 50\}$ ,  $\Lambda = \text{diag}\{5, 5\}$ . The basis set  $\phi(x)$  for the FLNN was selected as detailed in Section 4.4. Other techniques for selecting a basis set are given in (Sanner and Slotine 1991, Commuri and Lewis 1995).

The response using the NN controller with backprop weight tuning as in Table 4.2.1, with  $q(0) = 0$ ,  $\dot{q}(0) = 0$ , and initial NN weights of zero, appears in Figs. 4.2.4 and 4.2.5.

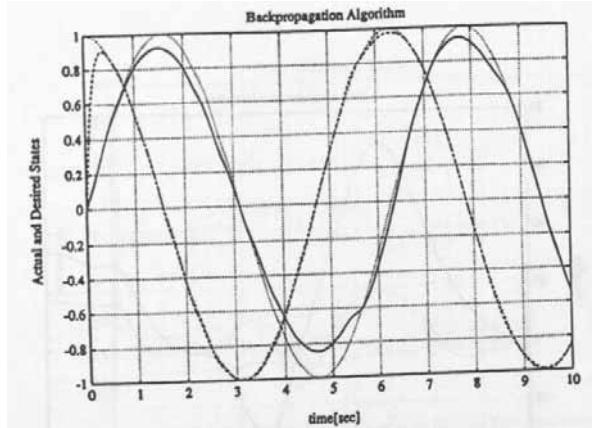


Figure 4.2.4: Response of NN controller with backprop weight tuning: actual and desired joint angles.

The tracking performance is not bad. Note the large values of weights required. In this case they appear to remain bounded, though this cannot in general be guaranteed unless PE holds.

*No initial NN training or learning phase was needed.* The NN weights were simply initialized at zero in this simulation.

**c. FLNN Controller With Augmented Weight Tuning.** The performance of the FLNN controller in Table 4.2.2 that uses augmented backprop tuning was simulated next. The controller parameters were taken as  $K_v = \text{diag}\{20, 20\}$ ,  $F = \text{diag}\{50, 50\}$ ,  $\Lambda = \text{diag}\{5, 5\}$ ,  $\kappa = 0.1$ . The basis set  $\phi(x)$  for the FLNN was selected as detailed in Section 4.4.

No initial NN training or learning phase was needed. The NN weights were simply initialized at zero in this simulation.

The response of the controller with improved weight tuning appears in Figs. 4.2.6 and 4.2.7. The tracking response is much better than that using straight backprop tuning, and the weights are smaller; they are guaranteed to remain bounded even though PE does not hold. The comparison with the performance of the standard adaptive controller in Chapter 3 is impressive, even though neither the dynamics of the arm nor a regression matrix was required to implement the NN controller.

**d. PD Control Without NN.** To study the contribution of the NN in the controller of Fig. 4.2.2, we simulated the PD controller  $\tau = K_v r$ , which has no neural net inner loop. Fig. 4.2.8 shows the result. Standard results in the robotics literature (Dawson *et al.* 1990) indicate that a PD controller should give bounded errors if  $K_v$  is large enough. This is observed in the figure. However, it is now clear that the addition of the NN makes a significant improvement in the tracking performance.  $\square$

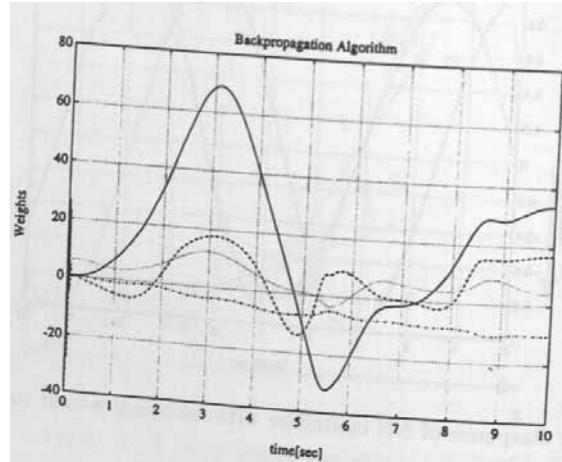


Figure 4.2.5: Response of NN controller with backprop weight tuning: representative weight estimates.

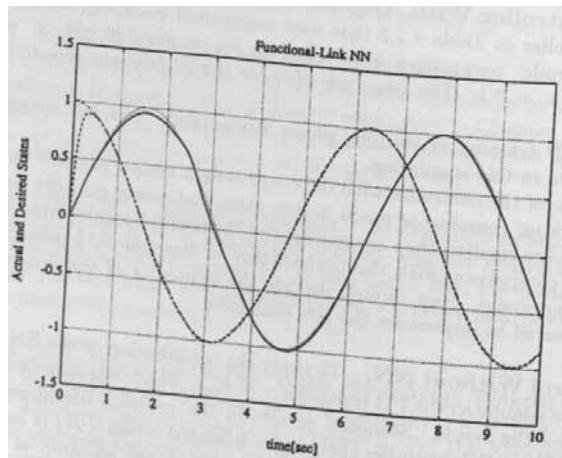


Figure 4.2.6: Response of NN controller with improved weight tuning: actual and desired joint angles.

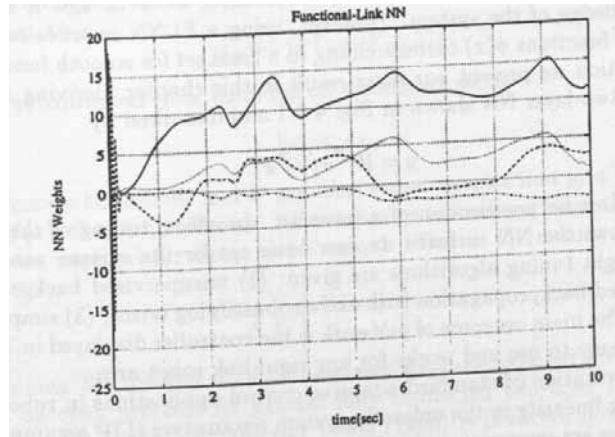


Figure 4.2.7: Response of NN controller with improved weight tuning: representative weight estimates.

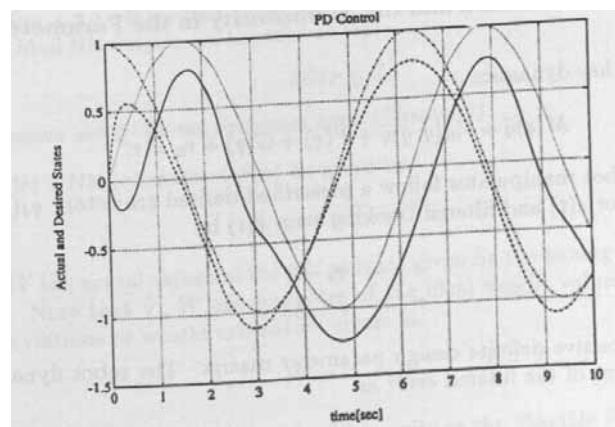


Figure 4.2.8: Response of controller without NN. actual and desired joint angles.

### 4.3 TWO-LAYER NEURAL NETWORK CONTROLLER

In Section 4.2 we presented a rigid-link robot manipulator tracking controller based on the NN  $y = W^T \phi(x)$  with one layer of tunable weights. This functional-link NN (FLNN) controller displayed good performance, even though it required no detailed knowledge of the system. However, using a FLNN requires one to select the activation functions  $\phi(x)$  corresponding to a basis set for smooth functions  $f(x)$ .

In this section we present our main result in this chapter, deriving a controller based on the two-layer NN shown in Fig. 4.0.1 and described by

$$y = W^T \sigma(V^T x). \quad (4.3.1)$$

This NN requires no preselection of a basis set. In effect, tuning of the first layer weights  $V$  allows the NN to learn its own basis set for the system nonlinearities. Three NN weight tuning algorithms are given: (1) unsupervised backpropagation, (2) unsupervised backpropagation with extra robustifying terms, (3) simplified Hebbian tuning. The main outcome of our work is the controller displayed in Table 4.3.2 which is very easy to use and works for any rigid-link robot arm.

A major limitation of standard adaptive control applications in robotics is the requirement for linearity in the unknown system parameters (LIP assumption). Recent approaches are overcoming this limitation (Colbaugh et al. 1994, 1995). The FLNN relaxes this assumption by providing a universal approximation property as long as  $\phi(x)$  is a basis. The FLNN is linear in the tunable weights  $W$ , but this is a less severe restriction than linearity in the system parameters, since the FLNN approximation property holds for all smooth  $f(x)$ . Overcoming requirements for linearity in the tunable parameters has been a major obstacle to continued development of adaptive control techniques. In this section we overcome this problem, providing tuning rules for a set of NN weights, some of which appear in a nonlinear fashion.

#### 4.3.1 NN Approximation and the Nonlinearity in the Parameters Problem

The robot arm has dynamics

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = \tau. \quad (4.3.2)$$

To make the robot manipulator follow a prescribed desired trajectory  $q_d(t)$ , define the tracking error  $e(t)$  and filtered tracking error  $r(t)$  by

$$e = q_d - q \quad (4.3.3)$$

$$r = \dot{e} + \Lambda e \quad (4.3.4)$$

with  $\Lambda > 0$  a positive definite design parameter matrix. The robot dynamics are expressed in terms of the filtered error as

$$M\dot{r} = -V_m r + f(x) + \tau_d - \tau \quad (4.3.5)$$

where the unknown nonlinear robot function is defined as

$$f(x) = M(q)(\ddot{q}_d + \Lambda \dot{e}) + V_m(q, \dot{q})(\dot{q}_d + \Lambda e) + F(\dot{q}) + G(q). \quad (4.3.6)$$

One may define  $x \equiv [e^T \ \dot{e}^T \ q_d^T \ \dot{q}_d^T \ \ddot{q}_d^T]^T$ .

Now, according to the universal approximation property of NN, there is a Two-layer NN such that

$$f(x) = W^T \sigma(V^T x) + \varepsilon \quad (4.3.7)$$

with the approximation error bounded on a compact set by

$$\|\varepsilon\| < \varepsilon_N, \quad (4.3.8)$$

with  $\varepsilon_N$  a known bound.  $W$  and  $V$  are ideal target weights that give good approximation to  $f(x)$ ; they are unknown. All we require is the knowledge that they exist, it is not even required for them to be unique. Define the matrix of all the NN weights as

$$Z = \begin{bmatrix} W & 0 \\ 0 & V \end{bmatrix}. \quad (4.3.9)$$

Determining the number  $L$  of hidden-layer neurons required for good approximation is an open problem for general fully connected two-layer NN of the sort shown in Fig. 4.0.1. In Commuri and Lewis (1995) it is shown how to determine the NN size  $L$  for cerebellar model arithmetic computer (CMAC) NN. In practical situations, one performs computer simulations of the NN controller on the system prior to implementing it. A value of  $L$  can be selected, the controller simulated, and then  $L$  increased for another simulation run. When there is no further improvement, that value of  $L$  is used. It was found that, for the two-link robot arm used in the examples, ten hidden-layer neurons suffices.

We shall require Assumption 4.1.1 that the reference trajectory  $q_d(t)$  is bounded by a known scalar bound  $q_B$ . The next assumption is also true in every practical situation. These assumptions are standard in the existing literature.

**Assumption 4.3.1 (Bounded Ideal Target NN Weights)** : On any compact subset of  $\Re^n$ , the ideal NN weights are bounded so that

$$\|Z\|_F \leq Z_B \quad (4.3.10)$$

with  $Z_B$  known and  $\|\cdot\|_F$  the Frobenius norm (Chapter 2).

Now, let a NN estimate of  $f(x)$  be given by

$$\hat{f}(x) = \hat{W}^T \sigma(\hat{V}^T x) \quad (4.3.11)$$

with  $\hat{V}$ ,  $\hat{W}$  the actual values of the NN weights given by the tuning algorithm to be specified. Note that  $\hat{V}$ ,  $\hat{W}$  are estimates of the ideal weight values and define the weight deviations or weight estimation errors as

$$\tilde{V} = V - \hat{V}, \quad \tilde{W} = W - \hat{W}, \quad \tilde{Z} = Z - \hat{Z}. \quad (4.3.12)$$

#### 4.3.1.1 Overcoming the Restriction of Linearity in the Tunable Parameters

In most standard adaptive robot control schemes, and in Section 4.2, linearity in the tunable parameters is needed to determine tuning algorithms. In this section we

overcome this restriction, providing tuning algorithms for a general set of parameters, some of which (e.g.  $V$ ) appear in a nonlinear fashion. The next steps are crucial in this development.

Define the hidden-layer output error for a given  $x$  as

$$\tilde{\sigma} = \sigma - \hat{\sigma} \equiv \sigma(V^T x) - \sigma(\hat{V}^T x). \quad (4.3.13)$$

The Taylor series expansion of  $\sigma(x)$  for a given  $x$  may be written as

$$\sigma(V^T x) = \sigma(\hat{V}^T x) + \sigma'(\hat{V}^T x)\tilde{V}^T x + O(\tilde{V}^T x)^2, \quad (4.3.14)$$

with

$$\sigma'(\hat{z}) \equiv \left. \frac{d\sigma(z)}{dz} \right|_{z=\hat{z}}$$

the Jacobian matrix and  $O(z)^2$  denoting terms of order two. Denoting  $\hat{\sigma}' = \sigma'(\hat{V}^T x)$ , we have

$$\tilde{\sigma} = \sigma'(\hat{V}^T x)\tilde{V}^T x + O(\tilde{V}^T x)^2 = \hat{\sigma}'\tilde{V}^T x + O(\tilde{V}^T x)^2. \quad (4.3.15)$$

The importance of this equation is that it replaces  $\tilde{\sigma}$ , which is nonlinear in  $\tilde{V}$ , by an expression linear in  $\tilde{V}$  plus higher-order terms. This will allow us to determine tuning algorithms for  $\hat{V}$  in subsequent derivations.

It is important to note that it is very easy to compute the Jacobian  $\hat{\sigma}'$  using the current NN weights and measurements of the signal  $x(t)$ , which appears in the closed-loop system.

Different bounds may be put on the Taylor series higher-order terms depending on the choice for activation functions  $\sigma(\cdot)$ . Noting that

$$O(\tilde{V}^T x)^2 = [\sigma(V^T x) - \sigma(\hat{V}^T x)] - \hat{\sigma}'\tilde{V}^T x \quad (4.3.16)$$

one has the following. The numbering of the constants  $c_i$  takes up after the  $c_0, c_1, c_2$  defined in Lemma 4.1.1.

**Lemma 4.3.1 (Bounds on Taylor Series Higher-Order Terms)** : For sigmoid, RBF, and tanh activation functions, the higher-order terms in the Taylor series are bounded by

$$\|O(\tilde{V}^T x)\| \leq c_3 + c_4 q_B \|\tilde{V}\|_F + c_5 \|\tilde{V}\|_F \|r\| \quad (4.3.17)$$

where  $c_i$  are computable positive constants.

Proof: Direct using (4.1.13), some standard norm inequalities, and the fact that  $\sigma(\cdot)$  and its derivative are bounded by constants for RBF, sigmoid, and tanh.

The extension of these ideas to nets with greater than three layers is not difficult and leads to composite function terms in the Taylor series (giving rise to backpropagation filtered error terms for the multilayer net case).

### 4.3.2 Controller Structure and Error System Dynamics

Select the control input

$$\tau = \hat{W}^T \sigma(\hat{V}^T x) + K_v r - v, \quad (4.3.18)$$

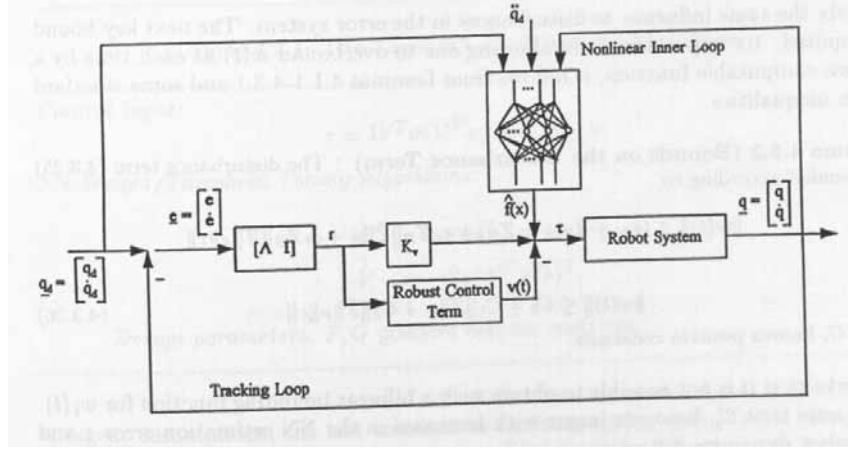


Figure 4.3.1: Multilayer NN controller structure.

with  $v(t)$  a function to be detailed subsequently that provides robustness in the face of higher-order terms in the Taylor series. The proposed NN control structure is shown in Fig. 4.3.1.

Using this controller, the closed-loop filtered error dynamics become

$$M\dot{r} = -(K_v + V_m)r + W^T \sigma(V^T x) - \hat{W}^T \sigma(\hat{V}^T x) + (\varepsilon + \tau_d) + v. \quad (4.3.19)$$

Adding and subtracting  $W^T \hat{\sigma}$  yields

$$M\dot{r} = -(K_v + V_m)r + \tilde{W}^T \hat{\sigma} + W^T \tilde{\sigma} + (\varepsilon + \tau_d) + v \quad (4.3.20)$$

with  $\hat{\sigma}$  and  $\tilde{\sigma}$  defined in (4.3.13). Adding and subtracting now  $\hat{W}^T \tilde{\sigma}$  yields

$$M\dot{r} = -(K_v + V_m)r + \tilde{W}^T \hat{\sigma} + \hat{W}^T \tilde{\sigma} + \tilde{W}^T \tilde{\sigma} + (\varepsilon + \tau_d) + v. \quad (4.3.21)$$

The key step is the use now of the Taylor series approximation (4.3.15) for  $\tilde{\sigma}$ , according to which the closed-loop error system is

$$M\dot{r} = -(K_v + V_m)r + \tilde{W}^T \hat{\sigma} + \hat{W}^T \hat{\sigma}' \tilde{V}^T x + w_1 + v \quad (4.3.22)$$

where the disturbance terms are

$$w_1(t) = \tilde{W}^T \hat{\sigma}' \tilde{V}^T x + W^T O(\tilde{V}^T x)^2 + \varepsilon + \tau_d. \quad (4.3.23)$$

Unfortunately, using this error system does not yield a compact set outside which a certain Lyapunov function derivative is negative. Therefore, finally write the error system

$$M\dot{r} = -(K_v + V_m)r + \tilde{W}^T (\hat{\sigma} - \hat{\sigma}' \hat{V}^T x) + \hat{W}^T \hat{\sigma}' \tilde{V}^T x + w + v \quad (4.3.24)$$

where the disturbance terms are

$$w(t) = \tilde{W}^T \hat{\sigma}' V^T x + W^T O(\tilde{V}^T x)^2 + \varepsilon + \tau_d. \quad (4.3.25)$$

*It is important to note that the NN reconstruction error  $\varepsilon(x)$ , the robot disturbances  $\tau_d$ , and the higher-order terms in the Taylor series expansion of  $f(x)$  all have exactly the same influence as disturbances in the error system. The next key bound is required. Its importance is in allowing one to overbound  $w(t)$  at each time by a known computable function; it follows from Lemmas 4.1.1-4.3.1 and some standard norm inequalities.*

**Lemma 4.3.2 (Bounds on the Disturbance Term)** : The disturbance term (4.3.25) is bounded according to

$$\|w(t)\| \leq (\varepsilon_N + d_B + c_3 Z_B) + c_6 Z_B \|\tilde{Z}\|_F + c_7 Z_B \|\tilde{Z}\|_F \|r\|$$

or

$$\|w(t)\| \leq C_0 + C_1 \|\tilde{Z}\|_F + C_2 \|\tilde{Z}\|_F \|r\| \quad (4.3.26)$$

with  $C_i$  known positive constants.

*Observe that it is not possible to obtain such a bilinear bounding function for  $w_1(t)$ . Also note that  $C_0$  becomes larger with increases in the NN estimation error  $\varepsilon$  and the robot dynamics disturbances  $\tau_d(t)$ .*

### 4.3.3 Weight Updates for Guaranteed Tracking Performance

*We give here some NN weight tuning algorithms that guarantee the tracking stability of the closed-loop system under various assumptions. It is required to demonstrate that the tracking error  $r(t)$  is suitably small and that the NN weights  $\hat{V}, \hat{W}$  remain bounded, for then the control  $\tau(t)$  is bounded. Three tuning algorithms are given: (1) an unsupervised version of backpropagation that works for an ideal case, (2) an improved version of backpropagation augmented by some extra terms, and (3) a simplified Hebbian scheme for computational ease.*

**Initial Tracking Error Requirement.** *The next assumption specifies the region of convergence of the two-layer NN controllers designed in this section.*

**Assumption 4.3.2 (Initial Condition Requirement)** . Suppose the desired trajectory  $q_d$ ,  $\dot{q}_d$ ,  $\ddot{q}_d$  is bounded by  $q_B$  as in Assumption 4.1.1. Define known constants  $c_0, c_2$  by Lemma 4.1.1. Let the NN approximation property (4.3.7) hold for the function  $f(x)$  given in (4.3.6) with a given accuracy  $\varepsilon_N$  for all  $x$  inside the ball of radius  $b_x > q_B$ . Let the initial tracking error satisfy  $\|r(0)\| < (b_x - q_B)/(c_0 + c_2)$ .

*See the remarks following Assumption 4.2.1 about this assumption. The following are key points: The set  $S_r$  specifies the set of allowed initial tracking errors  $r(0)$ , which becomes larger as the number  $L$  of NN hidden-layer neurons increases. The key role of the initial condition requirement is in showing the dependence of the allowed initial condition set  $S_r$  on the design parameters. The constants  $q_B, c_0, c_2, b_x$  need not be explicitly determined. The initial condition requirement is independent of the NN initial weights.*

Table 4.3.1: Two-Layer NN Controller for Ideal Case

*Control Input:*

$$\tau = \hat{W}^T \sigma(\hat{V}^T x) + K_v r - v,$$

*NN Weight/Threshold Tuning Algorithms:*

$$\begin{aligned}\dot{\hat{W}} &= F\hat{\sigma}r^T, \\ \dot{\hat{V}} &= Gx(\hat{\sigma}'^T \hat{W}r)^T,\end{aligned}$$

*Design parameters:*  $F, G$  positive definite matrices.

#### 4.3.3.1 Ideal Case- Unsupervised Backpropagation Tuning of Weights

The next result details the closed-loop behavior in a certain idealized case that demands: (1) no net functional reconstruction error, (2) no unmodeled disturbances in the robot arm dynamics, and (3) no higher-order Taylor series terms. These are stringent assumptions; the last amounts to the assumption that  $f(x)$  is linear! In this case the tuning rules are straightforward and familiar. The resulting controller is given in Table 4.3.1.

**Theorem 4.3.1 (Backprop Tuning for an Ideal Case) :**

Let the desired trajectory  $q_d(t)$  be bounded by  $q_B$  as in Assumption 4.1.1 and the initial tracking error  $r(0)$  satisfy Initial Condition Assumption 4.3.2. Suppose the disturbance term  $w_1(t)$  in (4.3.22) is equal to zero. Let the control input for (4.3.2) be given by (4.3.18) with  $v(t) = 0$  and weight tuning provided by

$$\dot{\hat{W}} = F\hat{\sigma}r^T, \quad (4.3.27)$$

$$\dot{\hat{V}} = Gx(\hat{\sigma}'^T \hat{W}r)^T, \quad (4.3.28)$$

with any constant positive definite design matrices  $F, G$ . Then the tracking error  $r(t)$  goes to zero with  $t$  and the weight estimates  $\hat{V}, \hat{W}$ , are bounded.

Proof:

Let the NN approximation property (4.3.7) hold for the function  $f(x)$  given in (4.3.6) with a given accuracy  $\varepsilon_N$  for all  $x$  in the compact set  $S_x \equiv \{x \mid \|x\| < b_x\}$  with  $b_x > q_B$ . Define  $S_r \equiv \{r \mid \|r\| < (b_x - q_B)/(c_0 + c_2)\}$ . Let  $r(0) \in S_r$ . Then the approximation property holds.

Define the Lyapunov function candidate

$$L(r, \tilde{W}, \tilde{V}) = \frac{1}{2}r^T M(q)r + \frac{1}{2}tr\{\tilde{W}^T F^{-1} \tilde{W}\} + \frac{1}{2}tr\{\tilde{V}^T G^{-1} \tilde{V}\}. \quad (4.3.29)$$

Differentiating yields

$$\dot{L} = r^T M \dot{r} + \frac{1}{2}r^T \dot{M}r + tr\{\tilde{W}^T F^{-1} \dot{\tilde{W}}\} + tr\{\tilde{V}^T G^{-1} \dot{\tilde{V}}\} \quad (4.3.30)$$

whence substitution from (4.3.22) (with  $w_1 = 0, v = 0$ ) yields

$$\dot{L} = -r^T K_v r + \frac{1}{2}r^T (\dot{M} - 2V_m)r + tr\{\tilde{W}^T (F^{-1} \dot{\tilde{W}} + \hat{\sigma}r^T)\} + tr\{\tilde{V}^T (G^{-1} \dot{\tilde{V}} + xr^T \hat{W}^T \hat{\sigma}')\}. \quad (4.3.31)$$

The skew symmetry property makes the second term zero, and since  $\hat{W} = W - \tilde{W}$  with  $W$  constant, so that  $d\tilde{W}/dt = -d\hat{W}/dt$  (and similarly for  $V$ ), the tuning rules yield

$$\dot{L} = -r^T K_v r. \quad (4.3.32)$$

Since  $L > 0$  and  $\dot{L} \leq 0$  this shows stability in the sense of Lyapunov so that  $r$ ,  $\tilde{V}$ , and  $\tilde{W}$  (and hence  $\hat{V}$ ,  $\hat{W}$ ) are bounded.

LaSalle's extension (Chapter 2) is now used to show that  $r(t)$  in fact goes to zero. Note that

$$\int_0^\infty -\dot{L} dt < \infty. \quad (4.3.33)$$

Boundedness of  $r$  guarantees the boundedness of  $e$  and  $\dot{e}$ , whence boundedness of the desired trajectory shows  $q$ ,  $\dot{q}$ ,  $x$  are bounded. Property P2 then shows boundedness of  $V_m(q, \dot{q})$ . Now,  $\dot{L} = -2r^T K_v r$ , and the boundedness of  $M^{-1}(q)$  and of all signals on the right-hand side of (4.3.22) verify the boundedness of  $\dot{r}$  and hence  $\ddot{L}$ , and thus the uniform continuity of  $\dot{L}$ . This allows one to invoke Barbalat's Lemma in connection with (4.3.33) to conclude that  $\dot{L}$  goes to zero with  $t$ , and hence that  $r(t)$  vanishes.  $\square$

**Weight Initialization and On-Line Tuning.** *There is in this scheme no preliminary off-line learning required, and the problem of net weight initialization occurring in other approaches in the literature does not arise. In fact, selecting the initial weights  $\hat{W}(0)$ ,  $\hat{V}(0)$  as zero takes the NN out of the circuit and leaves only the outer PD tracking loop in Fig. 4.3.1. It is well known that the PD term  $K_v r$  in (4.3.18) can then stabilize the plant on an interim basis until the NN begins to learn. A formal proof reveals that  $K_v$  should be large enough and the initial filtered error  $r(0)$  small enough. The exact value of  $K_v$  needed for initial stabilization is given in (Dawson et al. 1990) though for practical purposes it is only necessary to select  $K_v$  large. The NN weights are tuned on-line in real time; as the NN learns, the tracking error decreases.*

**Unsupervised Backprop Through Time Tuning.** *Note next that (4.3.27) and (4.3.28) are nothing but the continuous-time version of the backpropagation algorithm. In the sigmoid case, for instance*

$$\sigma'(z) = \text{diag}\{\sigma(z)\}(I - \text{diag}\{\sigma(z)\})$$

so that

$$\hat{\sigma}'^T \hat{W} r = \text{diag}\{\sigma(\hat{V}^T x)\}[I - \text{diag}\{\sigma(\hat{V}^T x)\}]\hat{W} r$$

which is the filtered error weighted by the current estimate  $\hat{W}$  and multiplied by the usual product involving the hidden-layer outputs. Compare this to the continuous-time backprop algorithm given in Chapter 1.

It is important to note that the tuning algorithms in the theorem are an unsupervised version of backpropagation through time; it is not necessary to know the ‘ideal’ plant output, instead, the filtered error  $r(t)$  is the signal backpropagated. Moreover, the required Jacobian  $\hat{\sigma}'$  is easily computed in terms of signals measured in the closed-loop system. This should be contrasted to other backprop techniques in the literature where direct evaluation of gradients requires the computation of Jacobians of unknown system dynamics.

Table 4.3.2: Two-Layer NN Controller with Augmented Backprop Tuning

---

*Control Input:*

$$\tau = \hat{W}^T \sigma(\hat{V}^T x) + K_v r - v,$$

*Robustifying Signal:*

$$v(t) = -K_z(\|\hat{Z}\|_F + Z_B)r$$

*NN Weight/Threshold Tuning Algorithms:*

$$\begin{aligned}\dot{\hat{W}} &= F\hat{\sigma}'r^T - F\hat{\sigma}'\hat{V}^T x r^T - \kappa F\|r\|\hat{W} \\ \dot{\hat{V}} &= Gx(\hat{\sigma}'^T \hat{W} r)^T - \kappa G\|r\|\hat{V}\end{aligned}$$

*Design parameters:*  $F, G$  positive definite matrices,  $\kappa > 0$  a small design parameter.

---

**Limitations of Backprop Tuning.** Theorem 4.3.1 indicates when backprop alone should suffice, namely, when the disturbance  $w_1(t)$  is equal to zero. According to (4.3.23), this requires no NN estimation errors  $\varepsilon$ , no robot arm disturbances  $\tau_d(t)$ , and no higher-order Taylor series terms. These are serious restrictions that never hold in practical situations.

#### 4.3.3.2 Augmented Backprop Tuning for the General Case

This subsection contains the main result of this chapter. We have just seen that backprop tuning can only be guaranteed to work in an unrealistic ideal case. To confront the stability and tracking performance of a NN robot arm controller in the thorny general case that allows NN estimation errors and system disturbances, we shall require: (1) the modification of the weight tuning rules, and (2) the addition of a robustifying term  $v(t)$ . The problem in this case is that, though it is not difficult to conclude that  $r(t)$  is bounded, it is impossible without these modifications to show that the NN weights are bounded in general. Boundedness of the weights is needed to verify that the control input  $\tau(t)$  remains bounded. The resulting controller is given in Table 4.3.2. In contrast to the ideal case, the tracking error does not go to zero with time, but is bounded by a small enough value.

**Theorem 4.3.2 (Augmented Backprop Weight Tuning) :**

Let the desired trajectory  $q_d(t)$  be bounded by  $q_B$  as in Assumption 4.1.1 and the initial tracking error  $r(0)$  satisfy Initial Condition Assumption 4.3.2. Let the ideal target NN weights be bounded as in Assumption 4.3.1. Take the control input for the robot dynamics (4.3.2) as (4.3.18) with PD gain satisfying

$$K_{v_{min}} > \frac{(C_0 + \kappa C_3^2/4)(c_0 + c_2)}{b_x - q_B}, \quad (4.3.34)$$

where  $C_3$  is defined in the proof and  $C_0$  and  $C_2$  are defined in (4.3.26). Let the robustifying term be

$$v(t) = -K_z(\|\hat{Z}\|_F + Z_B)r \quad (4.3.35)$$

with gain

$$K_z > C_2. \quad (4.3.36)$$

Let NN weight tuning be provided by

$$\dot{\hat{W}} = F\hat{\sigma}r^T - F\hat{\sigma}'\hat{V}^T x r^T - \kappa F\|r\|\hat{W} \quad (4.3.37)$$

$$\dot{\hat{V}} = Gx(\hat{\sigma}'^T \hat{W} r)^T - \kappa G\|r\|\hat{V} \quad (4.3.38)$$

with any constant matrices  $F = F^T > 0, G = G^T > 0$ , and  $\kappa > 0$  a small scalar design parameter. Then the filtered tracking error  $r(t)$  and NN weight estimates  $\hat{V}, \hat{W}$  are UUB, with the bounds given specifically by (4.3.40) and (4.3.41). Moreover, the tracking error may be kept as small as desired by increasing the gains  $K_v$  in (4.3.18).

Proof:

Let the NN approximation property (4.3.7) hold for the function  $f(x)$  given in (4.3.6) with a given accuracy  $\varepsilon_N$  for all  $x$  in the compact set  $S_x \equiv \{x \mid \|x\| < b_x\}$  with  $b_x > q_B$ . Define  $S_r \equiv \{r \mid \|r\| < (b_x - q_B)/(c_0 + c_2)\}$ . Let  $r(0) \in S_r$ . Then the approximation property holds.

Selecting the Lyapunov function (4.3.29), differentiating, and substituting now from the error system (4.3.24) yields

$$\begin{aligned} \dot{L} = & -r^T K_v r + \frac{1}{2} r^T (\dot{M} - 2V_m) r + \text{tr}\{\tilde{W}^T (F^{-1} \dot{\hat{W}} + \hat{\sigma} r^T - \hat{\sigma}' \hat{V}^T x r^T)\} \\ & + \text{tr}\{\tilde{V}^T (G^{-1} \dot{\hat{V}} + x r^T \hat{W}^T \hat{\sigma}')\} \end{aligned} \quad (4.3.39)$$

The tuning rules give

$$\begin{aligned} \dot{L} = & -r^T K_v r + \kappa \|r\| \text{tr}\{\tilde{W}^T (W - \tilde{W})\} + \kappa \|r\| \text{tr}\{\tilde{V}^T (V - \tilde{V})\} + r^T (w + v) \\ = & -r^T K_v r + \kappa \|r\| \text{tr}\{\tilde{Z}^T (Z - \tilde{Z})\} + r^T (w + v). \end{aligned}$$

Since

$$\text{tr}\{\tilde{Z}^T (Z - \tilde{Z})\} = \langle \tilde{Z}, Z \rangle - \|\tilde{Z}\|_F^2 \leq \|\tilde{Z}\|_F \|Z\|_F - \|\tilde{Z}\|_F^2,$$

there results

$$\begin{aligned} \dot{L} \leq & -r^T K_v r + \kappa \|r\| \cdot \|\tilde{Z}\|_F (Z_B - \|\tilde{Z}\|_F) - K_Z (\|\hat{Z}\|_F + Z_B) \|r\|^2 + \|r\| \cdot \|w\| \\ \leq & -K_{v_{min}} \|r\|^2 + \kappa \|r\| \cdot \|\tilde{Z}\|_F (Z_B - \|\tilde{Z}\|_F) - K_Z (\|\hat{Z}\|_F + Z_B) \|r\|^2 \\ & + \|r\| [C_0 + C_1 \|\tilde{Z}\|_F + C_2 \|r\| \cdot \|\tilde{Z}\|_F] \\ \leq & -\|r\| \{K_{v_{min}} \|r\| - \kappa \cdot \|\tilde{Z}\|_F (Z_B - \|\tilde{Z}\|_F) - C_0 - C_1 \|\tilde{Z}\|_F\} \end{aligned}$$

where  $K_{v_{min}}$  is the minimum singular value of  $K_v$ , Lemma 4.3.2 was used, and the last inequality holds due to (4.3.36).

$\dot{L}$  is negative as long as the term in braces is positive. Defining  $C_3 = Z_B + C_1/\kappa$  and completing the square yields

$$\begin{aligned} & K_{v_{min}} \|r\| - \kappa \cdot \|\tilde{Z}\|_F (Z_B - \|\tilde{Z}\|_F) - C_0 - C_1 \|\tilde{Z}\|_F \\ & = \kappa (\|\tilde{Z}\|_F - C_3/2)^2 + K_{v_{min}} \|r\| - C_0 - \kappa C_3^2/4 \end{aligned}$$

which is guaranteed positive as long as either

$$\|r\| > \frac{C_0 + \kappa C_3^2/4}{K_{v_{min}}} \equiv b_r \quad (4.3.40)$$

or

$$\|\tilde{Z}\|_F > C_3/2 + \sqrt{C_0/\kappa + C_3^2/4} \equiv b_Z. \quad (4.3.41)$$

Thus,  $\dot{L}$  is negative outside a compact set. According to the LaSalle extension in Chapter 2, this demonstrates the UUB of both  $\|r\|$  and  $\|\tilde{Z}\|_F$  as long as the control remains valid within this set. However, the PD gain condition (4.3.34) shows that the compact set defined by  $\|r\| \leq b_r$  is contained in  $S_r$ , so that the approximation property holds throughout.  $\square$

*The comments following Theorems 4.2.1, 4.2.2, 4.2.3, and 4.3.1 all apply here. The following remarks are particularly relevant.*

**Weight Initialization and On-Line Tuning.** *As usual the algorithms in this book require no preliminary off-line learning phase. The NN weights are initialized at zero. Weight training occurs on-line in real time.*

**Unsupervised Backpropagation Through Time with Extra Terms.** *The first terms of (4.3.37) and (4.3.38) are modified versions of the standard backpropagation algorithm. The last terms correspond to the  $e$ -modification (Narendra and Annaswamy 1987) in standard use in adaptive control to guarantee bounded parameter estimates; they form a special sort of forgetting term in the weight updates. Their function is to add to  $\dot{L}$  a quadratic term in  $\|\tilde{Z}\|_F$  so it can be shown that  $\dot{L}$  is negative outside a compact set in the  $(\|r\|, \|\tilde{Z}\|_F)$  plane. The second term in (4.3.37) is very interesting and bears discussion. The standard backprop terms can be thought of as backward propagating signals in a nonlinear ‘backprop’ network (Chapter 1) that contains multipliers. The second term in (4.3.37) corresponds to a forward traveling wave in the backprop net that provides a second-order correction to the weight tuning for  $\hat{W}$ .*

**Bounds on the Tracking Error and NN Weight Estimation Errors.** *The right-hand side of (4.3.40) can be taken as a practical bound on the tracking error in the sense that  $r(t)$  will never stray far above it. It is important to note from this equation that the tracking error increases with the NN reconstruction error  $\varepsilon_N$  and robot disturbances  $d_B$  (both appear in  $C_0$ ), yet arbitrarily small tracking errors may be achieved by selecting large gains  $K_v$ . On the other hand, (4.3.41) reveals that the NN weight errors are fundamentally bounded by  $Z_B$  (through  $C_3$ ). The tuning parameter  $\kappa$  offers a design trade-off between the relative eventual magnitudes of  $\|r\|$  and  $\|\tilde{Z}\|_F$ .*

**Design Trade-off of NN Size Versus Tracking Accuracy.** *Note that there is design freedom in the degree of complexity (e.g. size) of the NN. For a more complex NN (e.g. more hidden units), the NN estimation error  $\varepsilon_N$  decreases, so the bounding constant  $C_0$  will decrease, resulting in smaller tracking errors. On the other hand, a simplified NN with fewer hidden units will result in larger error bounds; this degradation can be compensated for by selecting a larger value for the PD gain  $K_v$ .*

#### 4.3.3.3 Simplified Hebbian Tuning

*The Hebbian NN weight tuning rule in Chapter 1 is based on classical conditioning experiments in psychology and associative memory paradigms. It is a simplified*

Table 4.3.3: Two-Layer NN Controller with Augmented Hebbian Tuning

*Control Input:*

$$\tau = \hat{W}^T \sigma(\hat{V}^T x) + K_v r - v,$$

*Robustifying Signal:*

$$v(t) = -K_z (\|\hat{Z}\|_F + Z_B) r$$

*NN Weight/Threshold Tuning Algorithms:*

$$\begin{aligned}\dot{\hat{W}} &= F\hat{\sigma}r^T - \kappa F\|r\|\hat{W} \\ \dot{\hat{V}} &= G\|r\|x\hat{\sigma}^T - \kappa G\|r\|\hat{V}\end{aligned}$$

*Design parameters:*  $F, G$  positive definite matrices,  $\kappa > 0$  a small design parameter.

tuning rule that does not require the computation of Jacobians as needed for back-propagation. Unfortunately, it has not been shown to converge in the literature so its use has been questionable. In this subsection we derive a Hebbian tuning algorithm for the two-layer NN controller in Fig. 4.3.1, proving closed-loop stability and guaranteeing the tracking performance. It is seen that, to prove convergence, an extra term must be added to the standard Hebbian algorithm. This controller is simpler to implement than the NN controller in Table 4.3.2 and often gives comparable performance.

The next theorem details the Hebbian tuning algorithm, which is summarized in Table 4.3.3. In this derivation, the problem of nonlinearity in the tunable weights  $V$  is overcome without using the Taylor series expansion of  $\sigma(V^T x)$  in (4.3.14)—the proof relies on the error system (4.3.20).

**Theorem 4.3.3 (Augmented Hebbian Weight Tuning) :**

Let the desired trajectory  $q_d(t)$  be bounded by  $q_B$  as in Assumption 4.1.1 and the initial tracking error  $r(0)$  satisfy Initial Condition Assumption 4.3.2. Let the ideal target NN weights be bounded as in Assumption 4.3.1. Take the control input for the robot dynamics (4.3.2) as (4.3.18) with PD gain satisfying

$$K_{v_{min}} > \frac{D(c_0 + c_2)}{b_x - q_B}, \quad (4.3.42)$$

with  $D$  defined in the proof. Let the robustifying term be

$$v(t) = -K_z (\|\hat{Z}\|_F + Z_B) r, \quad (4.3.43)$$

with gain

$$K_z > c_2. \quad (4.3.44)$$

Let NN weight tuning be provided by

$$\dot{\hat{W}} = F\hat{\sigma}r^T - \kappa F\|r\|\hat{W} \quad (4.3.45)$$

$$\dot{\hat{V}} = G\|r\|x\hat{\sigma}^T - \kappa G\|r\|\hat{V} \quad (4.3.46)$$

with any constant matrices  $F = F^T > 0, G = G^T > 0$ , and  $\kappa > 0$  a small scalar design parameter. Then the filtered tracking error  $r(t)$  and NN weight estimates  $\tilde{V}, \tilde{W}$  are UUB, with the bounds given specifically by (4.3.49) and (4.3.50). Moreover, the tracking error may be kept as small as desired by increasing the gains  $K_v$  in (4.3.18).

Proof:

Let the NN approximation property (4.3.7) hold for the function  $f(x)$  given in (4.3.6) with a given accuracy  $\varepsilon_N$  for all  $x$  in the compact set  $S_x \equiv \{x \mid \|x\| < b_x\}$  with  $b_x > q_B$ . Define  $S_r \equiv \{r \mid \|r\| < (b_x - q_B)/(c_0 + c_2)\}$ . Let  $r(0) \in S_r$ . Then the approximation property holds.

Let the Lyapunov function candidate be

$$L(r, \tilde{W}, \tilde{V}) = \frac{1}{2}r^T M(q)r + \frac{1}{2}\text{tr}\{\tilde{W}^T P^{-1} \tilde{W}\} + \frac{1}{2}\text{tr}\{\tilde{V}^T R^{-1} \tilde{V}\}. \quad (4.3.47)$$

Differentiating with respect to time along the solution of the error system dynamics (4.3.20) yields

$$\begin{aligned} \dot{L} = & -r^T K_v r + \frac{1}{2}r^T (\dot{M} - 2V_m)r + \text{tr}\left\{\tilde{W}^T (P^{-1} \dot{\tilde{W}} + \hat{\sigma} r^T)\right\} \\ & + \text{tr}\left\{\tilde{V}^T R^{-1} \dot{\tilde{V}}\right\} + r^T W^T \tilde{\sigma} + r^T v + r^T (\tau_d + \varepsilon). \end{aligned}$$

Use the NN weight update laws and skew-symmetry to obtain

$$\begin{aligned} \dot{L} = & -r^T K_v r + \kappa \|r\| \text{tr}\{\tilde{W}^T (W - \tilde{W})\} + \kappa \|r\| \text{tr}\{\tilde{V}^T (V - \tilde{V})\} \\ & + \|r\| \text{tr}\{\tilde{V}^T x \hat{\sigma}^T\} + r^T v + r^T W^T \tilde{\sigma} + r^T (\tau_d + \varepsilon) \\ = & -r^T K_v r + \kappa \|r\| \text{tr}\{\tilde{Z}^T (Z - \tilde{Z})\} + \|r\| \text{tr}\{\tilde{V}^T x \hat{\sigma}^T\} + r^T v + r^T W^T \tilde{\sigma} + r^T (\tau_d + \varepsilon). \end{aligned}$$

Since  $\text{tr}\{A^T B\} = \langle A, B \rangle \leq \|A\|_F \|B\|_F$  and

$$\text{tr}\{\tilde{Z}^T (Z - \tilde{Z})\} = \langle \tilde{Z}, Z \rangle - \|\tilde{Z}\|_F^2 \leq \|\tilde{Z}\|_F \|Z\|_F - \|\tilde{Z}\|_F^2,$$

there results

$$\begin{aligned} \dot{L} \leq & -K_{v_{min}} \|r\|^2 + \kappa \|r\| \cdot \|\tilde{Z}\|_F (Z_B - \|\tilde{Z}\|_F) + \|r\| \cdot \|\tilde{V}\|_F \cdot \|x \hat{\sigma}^T\|_F + r^T v \\ & + \|r\| \cdot \|W\|_F \cdot \|\tilde{\sigma}\| + \|r\| (d_B + \varepsilon_N) \end{aligned}$$

where  $K_{v_{min}}$  is the minimum singular value of  $K_v$ . Using (4.3.43) yields

$$\begin{aligned} \dot{L} \leq & -\|r\| \left\{ K_{v_{min}} \|r\| - \kappa \|\tilde{Z}\|_F (Z_B - \|\tilde{Z}\|_F) - \|\tilde{Z}\|_F (c_1 + c_2 \|r\|) + \right. \\ & \left. K_z (\|\tilde{Z}\|_F + Z_B) \|r\| - (\sqrt{L} Z_B + d_B + \varepsilon_N) \right\}. \end{aligned}$$

Picking  $K_z \geq c_2$  and completing the squares results in

$$\dot{L} \leq -\|r\| \left\{ K_{v_{min}} \|r\| + \kappa (\|\tilde{Z}\|_F - C_3)^2 - D \right\} \quad (4.3.48)$$

where

$$C_3 = \frac{Z_B}{2} + \frac{c_1}{2\kappa}$$

and

$$D = \kappa C_3^2 + \sqrt{L} Z_B + d_B + \varepsilon_N.$$

Thus, whenever

$$\|r\| > \frac{D}{K_{v_{min}}} \equiv b_r \quad (4.3.49)$$

or

$$\|\tilde{Z}\| > C_3 + \sqrt{\frac{D}{\kappa}} \equiv b_Z \quad (4.3.50)$$

$\dot{L}$  becomes negative and  $L$  decreases. Therefore, outside the region defined by the constant bounds (4.3.49) and (4.3.50) the tracking and parameter errors will decrease. Therefore, one can conclude UUB of  $r(t)$  and  $\tilde{Z}(t)$  as long as the control remains valid within this set. However, the PD gain condition (4.3.42) shows that the compact set defined by  $\|r\| \leq b_r$  is contained in  $S_r$ , so that the approximation property holds throughout. By Assumptions 4.1.1 and 4.3.1 one concludes that  $q(t)$  and  $\tilde{Z}(t)$  are UUB.  $\square$

*See the remarks at the end of Theorem 4.3.2. Note that the tuning rules are of Hebbian form, with each layer of weights tuned using the outer product of its input signal and its output signal. To prove convergence, the standard Hebbian rules must be modified by adding the robustifying  $\epsilon$ -mod terms, and also by multiplying the first term in (4.3.46) by  $\|r\|$ .*

#### 4.3.4 Two-Layer NN Controller Design and Simulation Example

The two-layer controller is shown in Fig. 4.3.1. We want to illustrate this control scheme using the augmented weight tuning laws given in Table 4.3.2. The design ease of the two-layer NN controller will be evident; it requires no knowledge of the system dynamics, not even their structure which is needed for adaptive control. Nor does it require the selection of any basis set as for the FLNN controllers.

##### Example 4.3.1 (Two-Layer NN Control of Two-Link Robot Arm) :

The dynamics of the planar two-link revolute arm in Fig. 4.2.3 were given in Chapter 3. We took the arm parameters as  $a_1 = a_2 = 1$  m,  $m_1 = 0.8$  kg,  $m_2 = 2.3$  kg. In Example 4.2.1 we made several points: (1) The standard adaptive controller performs satisfactorily when the regression matrix is exactly known. However, if there are any unmodeled dynamics, it performs badly. (2) The FLNN controller performs well when all the dynamics are unmodeled, but it requires the computation of a NN basis set of activation functions. (3) A PD controller by itself does not perform well.

A MATLAB M file was written to simulate the two-layer NN controller. It is very similar to the code given in the adaptive control example in Chapter 3. To implement the NN controller in Table 4.3.2 we simply selected 10 hidden-layer neurons and sigmoid activation functions. No basis set selection was required. We selected a desired trajectory exponential in  $q_{1d}(t)$  and sinusoidal in  $q_{2d}(t)$ . To test the ability of the NN controller to handle discontinuous commands, the desired trajectory was turned on at 0.1 sec and turned off at 0.9 sec.

The NN controller parameters were taken as  $K_v = \text{diag}\{20, 20\}$ ,  $F = \text{diag}\{50, 50\}$ ,  $\Lambda = \text{diag}\{5, 5\}$ ,  $\kappa = 0.1$ . The response of the NN controller with the augmented backprop weight tuning in Table 4.3.2 appears in Figs. 4.3.2-4.3.3. The tracking response is good and the weights reach bounded values. No initial NN training or learning phase was needed. The NN weights were simply initialized at zero in this simulation.  $\square$

### 4.4 PARTITIONED NN AND SIGNAL PREPROCESSING

In this section we show how NN controller implementation may be streamlined by partitioning the NN into several smaller subnets to obtain more efficient computation. Also discussed in this section is preprocessing of input signals for the NN to improve efficiency and accuracy of the approximation. Some discussion is given on determining a basis set of activation functions for the FLNN controller in Section 4.2, though the NN controllers in section 4.3 would normally be used for implementation—they do not require computation of a basis set.

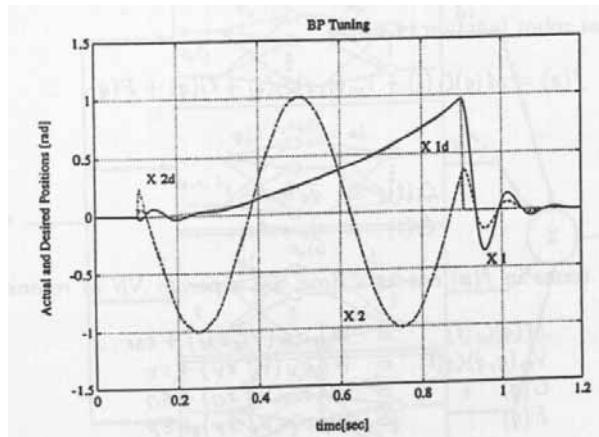


Figure 4.3.2: Response of NN controller with improved weight tuning: actual and desired joint angles.

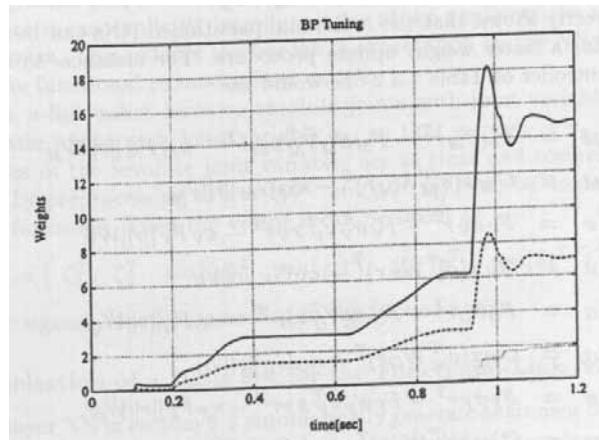


Figure 4.3.3: Response of NN controller with improved weight tuning: representative weight estimates.

#### 4.4.1 Partitioned NN

A major advantage of the NN approach is that it allows one to partition the controller in terms of partitioned NN or neural subnets. This: (1) simplifies the design, (2) gives added controller structure, and (3) makes for faster weight tuning algorithms.

The nonlinear robot function (4.1.8) is

$$f(x) = M(q)\zeta_1(t) + V_m(q, \dot{q})\zeta_2(t) + G(q) + F(\dot{q}), \quad (4.4.1)$$

where

$$\begin{aligned} \zeta_1(t) &\equiv \ddot{q}_d + \Lambda \dot{e} \\ \zeta_2(t) &\equiv \dot{q}_d + \Lambda e. \end{aligned}$$

Taking the four terms in  $f(x)$  one at a time, use separate NN to reconstruct each term so that

$$\begin{aligned} M(q)\zeta_1(t) &= W_M^T \sigma_M(V_M^T x_M) + \varepsilon_M \\ V_m(q, \dot{q})\zeta_2(t) &= W_V^T \sigma_V(V_V^T x_V) + \varepsilon_V \\ G(q) &= W_G^T \sigma_G(V_G^T x_G) + \varepsilon_G \\ F(\dot{q}) &= W_F^T \sigma_F(V_F^T x_F) + \varepsilon_F. \end{aligned} \quad (4.4.2)$$

This procedure results in four neural subnets, which we term a structured or partitioned NN, as shown in Fig. 4.4.1. The nonlinear function is given by

$$f(x) = [W_M^T \ W_V^T \ W_G^T \ W_F^T] \begin{bmatrix} \sigma_M \\ \sigma_V \\ \sigma_G \\ \sigma_F \end{bmatrix} + \varepsilon \quad (4.4.3)$$

with overall estimation error  $\varepsilon = \varepsilon_M + \varepsilon_V + \varepsilon_G + \varepsilon_F$ .

It can be directly shown that the individual partitioned NNs can be separately tuned, making for a faster weight update procedure. For instance, to implement the two-layer controller of Table 4.3.2 one would use

$$\begin{aligned} \dot{\hat{W}}_M &= F_M \hat{\sigma}_M r^T - F_M \hat{\sigma}'_M \hat{V}_M^T x_M r^T - \kappa_M F_M \|r\| \hat{W}_M \\ \dot{\hat{V}}_M &= G_M x (\hat{\sigma}'_M \hat{W}_M r)^T - \kappa_M G_M \|r\| \hat{V}_M \\ \dot{\hat{W}}_V &= F_V \hat{\sigma}_V r^T - F_V \hat{\sigma}'_V \hat{V}_V^T x_V r^T - \kappa_V F_V \|r\| \hat{W}_V \\ \dot{\hat{V}}_V &= G_V x (\hat{\sigma}'_V \hat{W}_V r)^T - \kappa_V G_V \|r\| \hat{V}_V \\ \dot{\hat{W}}_G &= F_G \hat{\sigma}_G r^T - F_G \hat{\sigma}'_G \hat{V}_G^T x_G r^T - \kappa_G F_G \|r\| \hat{W}_G \\ \dot{\hat{V}}_G &= G_G x (\hat{\sigma}'_G \hat{W}_G r)^T - \kappa_G G_G \|r\| \hat{V}_G \\ \dot{\hat{W}}_F &= F_F \hat{\sigma}_F r^T - F_F \hat{\sigma}'_F \hat{V}_F^T x_F r^T - \kappa_F F_F \|r\| \hat{W}_F \\ \dot{\hat{V}}_F &= G_F x (\hat{\sigma}'_F \hat{W}_F r)^T - \kappa_F G_F \|r\| \hat{V}_F \end{aligned} \quad (4.4.4)$$

An advantage of this structured NN is that if some terms in the robot dynamics are well-known (e.g. inertia matrix  $M(q)$  and gravity  $G(q)$ ), then their NNs can be replaced by equations that compute them. NNs can be used to reconstruct only the unknown terms or those too complicated to compute, which will probably include the friction  $F(\dot{q})$  and the Coriolis/centripetal terms  $V_m(q, \dot{q})$ .

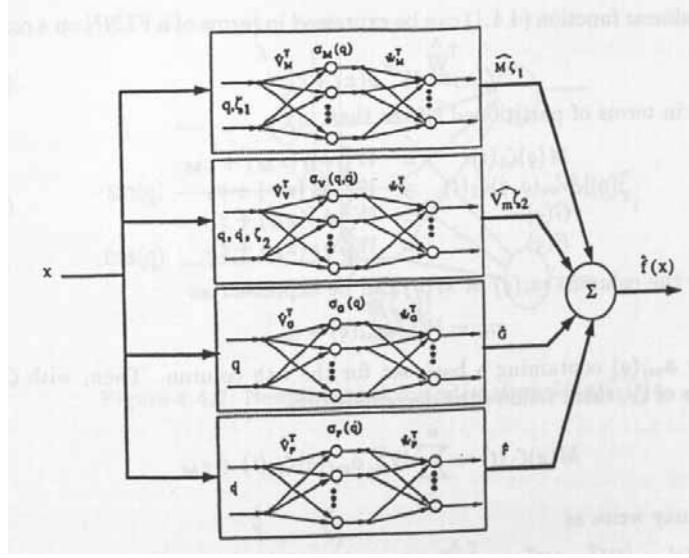


Figure 4.4.1: Partitioned neural net.

#### 4.4.2 Preprocessing of Neural Net Inputs

The selection of a suitable NN input vector  $x(t)$  for computation remains to be addressed; some preprocessing of signals yields a more advantageous choice than (4.1.9) since it can explicitly introduce some of the nonlinearities inherent to robot arm dynamics. This reduces the burden of expectation on the NN and, in fact, also reduces the functional reconstruction error  $\varepsilon$ .

Let an  $n$ -link robot have  $n_r$  revolute joints with joint variable vector  $q_r$ , and  $n_p$  prismatic joints with joint variable  $q_p$ , so that  $n = n_r + n_p$ . Since the only occurrences of the revolute joint variables are as sines and cosines, transform  $q = [q_r^T \ q_p^T]^T$  by preprocessing to  $[\cos(q_r)^T \ \sin(q_r)^T \ q_p^T]^T$  to be used as arguments for the basis functions. Then the vector  $x$  can be taken as

$$x = [\zeta_1^T \ \zeta_2^T \ \cos(q_r)^T \ \sin(q_r)^T \ q_p^T \ \dot{q}^T \ \operatorname{sgn}(\dot{q})^T]^T, \quad (4.4.5)$$

where the signum function is needed in the friction terms.

#### 4.4.3 Selection of a Basis Set for the Functional-Link NN

The two-layer NN in section 4.3 automatically generate their own basis set by tuning the first layer NN weights  $V$ . In practical applications, if there is enough computing power one would use the two-layer NN controller. If computing power is limited, however, the one-layer NN becomes attractive since the weight tuning algorithms are less complex. In Section 4.2 was given a one-layer FLNN controller where it is required to select a basis set of activation functions  $\phi(x)$ . This may be accomplished by choosing random scaling and shift parameters for the functions  $\phi(\cdot)$ , as in the Random Vector Functional Link (RVFL) net (Chapter 1), or as follows.

The nonlinear function (4.4.1) can be expressed in terms of a FLNN on a compact set as

$$f(x) = W^T \phi(x) + \varepsilon(x). \quad (4.4.6)$$

Write  $f(x)$  in terms of partitioned NN so that

$$\begin{aligned} M(q)\zeta_1(t) &= W_M^T \phi_M(x_M) + \varepsilon_M \\ V_m(q, \dot{q})\zeta_2(t) &= W_V^T \phi_V(x_V) + \varepsilon_V \\ G(q) &= W_G^T \phi_G(x_G) + \varepsilon_G \\ F(\dot{q}) &= W_F^T \phi_F(x_F) + \varepsilon_F. \end{aligned} \quad (4.4.7)$$

Assume the columns  $m_i(q)$  of  $M(q)$  can be expressed as

$$m_i = W_{mi}^T \phi_{mi}(q) + \varepsilon_{mi},$$

with vector  $\phi_{mi}(q)$  containing a basis set for the  $i$ -th column. Then, with  $\zeta_{1i}$  the components of  $\zeta_1$ , there follows the decomposition

$$M(q)\zeta_1(t) = \sum_1^n W_{mi}^T \phi_{mi}(q) \zeta_{1i}(t) + \varepsilon_M$$

which one may write as

$$M(q)\zeta_1(t) = [W_{m1}^T \ W_{m2}^T \ \cdots \ W_{mn}^T] [\zeta_1(t) \otimes \phi_m(q)] \equiv W_M^T \phi_M(q) + \varepsilon_M \quad (4.4.8)$$

where  $\otimes$  denotes Kronecker product (Lewis, Abdallah, and Dawson 1993) and it has been assumed that the same basis serves for each column so that  $\phi_{m1}(q) = \phi_{m2}(q) = \dots = \phi_{mn}(q) \equiv \phi_m(q)$ .

Similarly,

$$V_m(q, \dot{q})\zeta_2(t) = W_V^T [\zeta_2(t) \otimes \phi_v(q, \dot{q})] \equiv W_V^T \phi_V(q, \dot{q}) + \varepsilon_V \quad (4.4.9)$$

with  $\phi_v$  a basis for each column of  $V_m$ . It is direct to write

$$G(q) = W_G^T \phi_G(q) + \varepsilon_G \quad (4.4.10)$$

$$F(\dot{q}) = W_F^T \phi_F(\dot{q}) + \varepsilon_F. \quad (4.4.11)$$

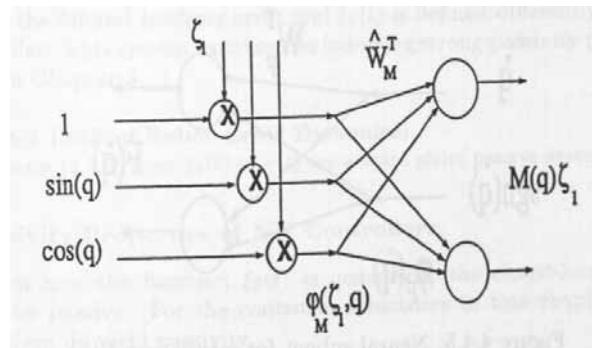
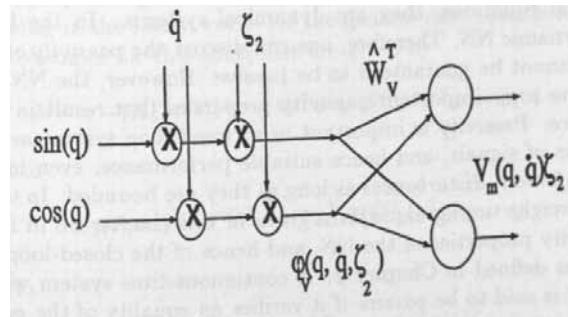
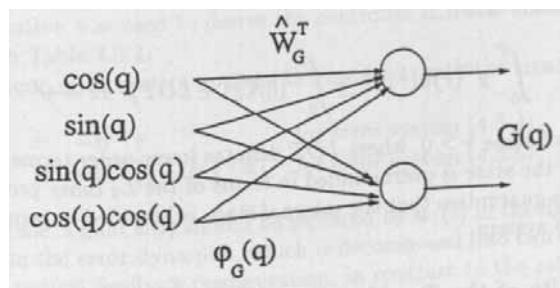
This procedure involves separately determining basis functions for the four terms in the robot dynamics, a much simplified problem. Then, the required basis functions are given in terms of the basis functions for the individual terms as

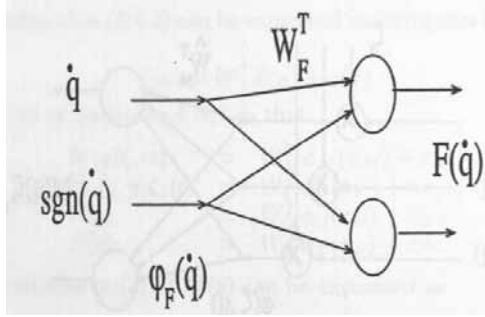
$$\phi(x) = \begin{bmatrix} \zeta_1(t) \otimes \phi_m(q) \\ \zeta_2(t) \otimes \phi_v(q, \dot{q}) \\ \phi_G(q) \\ \phi_F(\dot{q}) \end{bmatrix} \equiv \begin{bmatrix} \phi_M \\ \phi_V \\ \phi_G \\ \phi_F \end{bmatrix}. \quad (4.4.12)$$

#### Example 4.4.1 (Partitioned NN Controller Design) :

For the two link robot arm in Fig. 4.2.3 one proceeds as follows to determine the FLNN basis used in Example 4.2.1.

From well-known properties of the dynamics of any two-link revolute robot, the robot inertia matrix requires terms like  $\sin(q)$ ,  $\cos(q)$ , and constant terms. Therefore, the neural subnet required to construct  $M(q)\zeta_1$  appears in Fig. 4.4.2, where  $\otimes$  denotes Kronecker product. The Coriolis/centripetal matrix needs terms like  $\sin(q)$ ,  $\cos(q)$ , multiplied generally in all possible combinations by  $\dot{q}(t)$ . Therefore, the neural subnet required to estimate  $V_m(q, \dot{q})\zeta_2$  appears in Fig. 4.4.3. Likewise, the sub NN required for the gravity and friction terms appear respectively in Fig. 4.4.4 and Fig. 4.4.5.  $\square$

Figure 4.4.2: Neural subnet for estimating  $M(q)\zeta_1(t)$ .Figure 4.4.3: Neural subnet for estimating  $V_m(q, \dot{q})\zeta_2(t)$ .Figure 4.4.4: Neural subnet for estimating  $G(q)$ .

Figure 4.4.5: Neural subnet for estimating  $F(\dot{q})$ .

#### 4.5 PASSIVITY PROPERTIES OF NN CONTROLLERS

The NN used in this chapter are static feedforward nets, but since they are tuned using differential equations, they are dynamical systems. In the feedback loop, they become dynamic NN. Therefore, one can discuss the passivity of these NN. In general a NN cannot be guaranteed to be passive. However, the NN controllers in this chapter have some important passivity properties that result in robust closed-loop performance. Passivity is important in a closed-loop system as it guarantees the boundedness of signals, and hence suitable performance, even in the presence of additional unforeseen disturbances as long as they are bounded. In this section we show that the weight tuning algorithms given in this chapter do in fact guarantee desirable passivity properties of the NN, and hence of the closed-loop system.

Passivity was defined in Chapter 2. A continuous-time system with input  $u(t)$  and output  $y(t)$  is said to be passive if it verifies an equality of the power form

$$\dot{L}(t) = y^T u - g(t) \quad (4.5.1)$$

for some  $L(t)$  that is lower bounded and some  $g(t) \geq 0$ . An important form of passivity is the stronger notion of state strict passivity (SSP), where  $g(t)$  is a monic quadratic function of  $\|X\|$  with bounded coefficients, where  $X(t)$  is the internal state of the system. Then,

$$\int_0^T y^T(\tau)u(\tau)d\tau \geq \int_0^T (\|X\|^2 + LOT) d\tau - \gamma^2 \quad (4.5.2)$$

for all  $T \geq 0$  and some  $\gamma \geq 0$ , where  $LOT$  denotes lower-order terms in  $\|X\|$ . Then, the  $L_2$  norm of the state is overbounded in terms of the  $L_2$  inner product of output and input. This guarantees that the internal state is bounded in terms of the power delivered to the system.

##### 4.5.1 Passivity of the Tracking Error Dynamics

The error dynamics in this chapter all have the form

$$M\ddot{r} = -(K_v + V_m)r + \xi_0 \quad (4.5.3)$$

where  $r(t)$  is the filtered tracking error and  $\xi_0(t)$  is defined differently for each type of NN controller. This system satisfies the following strong passivity property which was proven in Chapter 3.

**Theorem 4.5.1 (SSP of Robot Error Dynamics) :**

The dynamics (4.5.3) from  $\xi_0(t)$  to  $r(t)$  are a state strict passive system.  $\square$

#### 4.5.2 Passivity Properties of NN Controllers

Depending on how the function  $\xi_0(t)$  is generated, the closed-loop system may or may not be passive. For the controller structures in this chapter, the tuning algorithms given do yield passivity.

##### 4.5.2.1 Passivity Properties of Two-Layer NN Controller

Consider the two-layer NN controller with tuning given by Table 4.3.1 or Table 4.3.2. The backprop tuning in the former only works in an idealized case, while the augmented tuning in the latter works for the general case. Let us see why.

The error dynamics for this controller are given by

$$M\dot{r} = -(K_v + V_m)r + \tilde{W}^T\hat{\sigma} + \hat{W}^T\hat{\sigma}'\tilde{V}^T x + w_1 + v \quad (4.5.4)$$

with disturbance

$$w_1(t) = \tilde{W}^T\hat{\sigma}'\tilde{V}^T x + W^T O(\tilde{V}^T x)^2 + \varepsilon + \tau_d \quad (4.5.5)$$

or, equivalently,

$$M\dot{r} = -(K_v + V_m)r + \tilde{W}^T(\hat{\sigma} - \hat{\sigma}'\hat{V}^T x) + \hat{W}^T\hat{\sigma}'\tilde{V}^T x + w + v \quad (4.5.6)$$

with disturbance

$$w(t) = \tilde{W}^T\hat{\sigma}'V^T x + W^T O(\tilde{V}^T x)^2 + \varepsilon + \tau_d. \quad (4.5.7)$$

The former equation was used to derive the controller in Table 4.3.1 and the latter the controller in Table 4.3.2.

The closed-loop error system appears in Fig. 4.5.1, with the signal  $\xi_1$  defined as

$$\begin{aligned} \xi_1(t) &= -\tilde{W}^T\hat{\sigma}, && \text{for error system (4.5.4)} \\ \xi_1(t) &= -\tilde{W}^T(\hat{\sigma} - \hat{\sigma}'\hat{V}^T x), && \text{for error system (4.5.6).} \end{aligned} \quad (4.5.8)$$

(In the former case, signal  $w(t)$  should be replaced by  $w_1(t)$  in the figure.) Note the role of the NN in the error dynamics, which is decomposed into two effective blocks appearing in a typical feedback configuration, in contrast to the role of the NN in the controller in Fig. 4.3.1.

We now reveal the passivity properties engendered by the two tuning algorithms. The first result is with regard to the backprop tuning algorithm in Table 4.3.1 which is based on error system (4.5.4).

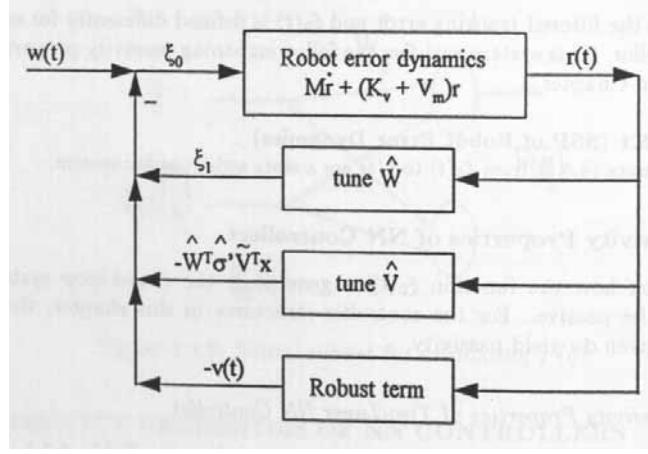


Figure 4.5.1: Two-layer neural net closed-loop error system.

**Theorem 4.5.2 (Passivity of Backprop NN Tuning Algorithm) :**

The backprop weight tuning algorithm in Table 4.3.1 makes the map from  $r(t)$  to  $-\tilde{W}^T \hat{\sigma}$ , and the map from  $r(t)$  to  $-\tilde{W}^T \hat{\sigma}' \tilde{V}^T x$ , both passive maps.

Proof:

The dynamics with respect to  $\tilde{W}, \tilde{V}$  are

$$\dot{\tilde{W}} = -F\hat{\sigma}r^T \quad (4.5.9)$$

$$\dot{\tilde{V}} = -Gx(\hat{\sigma}'^T \hat{W}r)^T \quad (4.5.10)$$

**1.** Selecting the non-negative function

$$L = \frac{1}{2} \text{tr}\{\tilde{W}^T F^{-1} \tilde{W}\}$$

and evaluating  $\dot{L}$  along the trajectories of (4.5.9) yields

$$\dot{L} = \text{tr}\{\tilde{W}^T F^{-1} \dot{\tilde{W}}\} = -\text{tr}\{\tilde{W}^T \hat{\sigma}r^T\} = r^T(-\tilde{W}^T \hat{\sigma}),$$

which is in power form (4.5.1).

**2.** Selecting the non-negative function

$$L = \frac{1}{2} \text{tr}\{\tilde{V}^T G^{-1} \tilde{V}\}$$

and evaluating  $\dot{L}$  along the trajectories of (4.5.10) yields

$$\dot{L} = \text{tr}\{\tilde{V}^T G^{-1} \dot{\tilde{V}}\} = -\text{tr}\{\tilde{V}^T x(\hat{\sigma}'^T \hat{W}r)^T\} = r^T(-\hat{W}^T \hat{\sigma}' \tilde{V}^T x).$$

which is in power form.  $\square$

Thus, the robot error system in Fig. 4.5.1 is state strict passive (SSP) and the weight error blocks are passive; this guarantees the passivity of the closed-loop system. Using the passivity theorem (Slotine and Li 1991) one may now conclude that the input/output signals of each block are bounded as long as the external inputs are bounded.

Unfortunately, though passive, the closed-loop system is not SSP so, when disturbance  $w_1(t)$  is nonzero, this does not yield boundedness of the internal states of the weight blocks (i.e.  $\tilde{W}$ ,  $\tilde{V}$ ) unless those blocks are observable, that is persistently exciting (PE). Unfortunately, this does not yield a convenient method for defining PE in a two-layer NN, as the two weight tuning blocks are coupled, forming in fact a bilinear system. By contrast, PE conditions for the one-layer FLNN case are easy to deduce.

The next result shows why a PE condition is not needed with the modified weight update algorithm of Table 4.3.2; it is in the context of error system (4.5.6).

**Theorem 4.5.3 (SSP of Augmented Backprop NN Tuning Algorithm) :**

The modified weight tuning algorithms in Table 4.3.2 make the map from  $r(t)$  to  $-\tilde{W}^T(\hat{\sigma} - \hat{\sigma}'\hat{V}^T x)$ , and the map from  $r(t)$  to  $-\hat{W}^T\hat{\sigma}'\hat{V}^T x$ , both state strict passive (SSP) maps.

Proof:

The revised dynamics relative to  $\tilde{W}$ ,  $\tilde{V}$  are given by

$$\begin{aligned}\dot{\tilde{W}} &= -F\hat{\sigma}r^T + F\hat{\sigma}'\hat{V}^T x r^T + \kappa F\|r\|\hat{W} \\ \dot{\tilde{V}} &= -Gx(\hat{\sigma}'^T \hat{W} r)^T + \kappa G\|r\|\hat{V}.\end{aligned}$$

**1.** Selecting the non-negative function

$$L = \frac{1}{2} \operatorname{tr}\{\tilde{W}^T F^{-1} \tilde{W}\}$$

and evaluating  $\dot{L}$  yields

$$\dot{L} = \operatorname{tr}\{\tilde{W}^T F^{-1} \dot{\tilde{W}}\} = \operatorname{tr}\{[-\tilde{W}^T(\hat{\sigma} - \hat{\sigma}'^T \hat{V}^T x)] r^T + \kappa \|r\| \tilde{W}^T \hat{W}\}$$

Since

$$\operatorname{tr}\{\tilde{W}^T(W - \tilde{W})\} = \langle \tilde{W}, W \rangle_F - \|\tilde{W}\|_F^2 \leq \|\tilde{W}\|_F \cdot \|W\|_F - \|\tilde{W}\|_F^2,$$

there results

$$\begin{aligned}\dot{L} &\leq r^T [-\tilde{W}^T(\hat{\sigma} - \hat{\sigma}'^T \hat{V}^T x)] - \kappa \|r\| \cdot (\|\tilde{W}\|_F^2 - \|\tilde{W}\|_F \|W\|_F) \\ &\leq r^T [-\tilde{W}^T(\hat{\sigma} - \hat{\sigma}'^T \hat{V}^T x)] - \kappa \|r\| \cdot (\|\tilde{W}\|_F^2 - W_B \|\tilde{W}\|_F)\end{aligned}$$

which is in power form with the last function quadratic in  $\|\tilde{W}\|_F$ .

**2.** Selecting the non-negative function

$$L = \frac{1}{2} \operatorname{tr}\{\tilde{V}^T G^{-1} \tilde{V}\}$$

and evaluating  $\dot{L}$  yields

$$\begin{aligned}\dot{L} = \operatorname{tr}\{\tilde{V}^T G^{-1} \dot{\tilde{V}}\} &= r^T (-\hat{W}^T \hat{\sigma}' \tilde{V}^T x) - \kappa \|r\| (\|\tilde{V}\|_F^2 - \langle \tilde{V}, V \rangle_F) \\ &\leq r^T (-\hat{W}^T \hat{\sigma}' \tilde{V}^T x) - \kappa \|r\| (\|\tilde{V}\|_F^2 - V_B \|\tilde{V}\|)\end{aligned}$$

which is in power form with the last function quadratic in  $\|\tilde{V}\|_F$ .  $\square$

*It is exactly the special forms of  $\dot{L}$  that allow one to show the boundedness of  $\tilde{W}$  and  $\tilde{V}$  when the first terms (power input) are bounded, as in the proof of Theorem 4.3.2.*

*The SSP of both the robot dynamics and the weight tuning blocks does guarantee SSP of the closed-loop system, so that the norms of the internal states are bounded in terms of the power delivered to each block. Then, boundedness of input/output signals assures state boundedness without any sort of observability or PE requirement.*

#### 4.5.2.2 Passivity Properties of One-Layer NN Controller

*In a similar fashion, it is shown that the FLNN controller tuning algorithm in Table 4.2.1 makes the system passive, so that an additional PE condition is needed to verify internal stability of the NN weights. On the other hand, the augmented tuning algorithm in Table 4.2.2 yields SSP, so that no PE is needed.*

#### 4.5.2.3 Definition of Passive and Robust NN

*We define a dynamically tuned NN as passive if, in the error formulation, the tuning guarantees the passivity of the weight tuning subsystems. Then, an extra PE condition is needed to guarantee boundedness of the weights. We define a dynamically tuned NN as robust if, in the error formulation, the tuning guarantees the SSP of the weight tuning subsystem. Then, no extra PE condition is needed for boundedness of the weights. Note that (1) SSP of the open-loop plant error system is needed in addition for tracking stability, and (2) the NN passivity properties are dependent on the weight tuning algorithm used.*

## 4.6 CONCLUSIONS

*In this chapter we have introduced several sorts of biologically motivated control algorithms that can afford improved control of robot manipulators. These NN controllers allow accurate and dynamic following of prescribed trajectories, not simply control using ‘via’ points specified by a teach pendant. They can significantly improve the capabilities of commercial robot controllers for modern-day speed and accuracy requirements by retaining the basic PD/PID loop, but adding an inner adaptive loop that allows the controller to learn unknown parameters such as friction coefficients and payload masses, thereby improving tracking accuracy.*

*Two sorts of NN controller were given. The one-layer linear-in-the-parameters NN controllers are simpler to implement as the weight tuning algorithms are less complex. However, a basis set must be selected for the activation functions. The two-layer NN controllers are more general but are nonlinear in the tunable weights. Thus, a fundamental limitation in standard adaptive control techniques had to be overcome to allow confrontation of this thorny problem. The two-layer NN controllers should be used in practical situations if computing power permits. Some advanced notions were introduced, including partitioned NN and passivity properties of NN controllers.*

*In the next chapter we show how to design NN controllers for more complex industrial applications including force control and control of robots with vibratory and flexible modes.*

#### 4.7 REFERENCES

- Barron, A.R., "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Trans. Info. Theory*, vol. 39, no. 3, pp. 930-945, May 1993.
- Chen, F.-C., and H.K. Khalil, "Adaptive control of nonlinear systems using neural networks," *Int. J. Control*, vol. 55, no. 6, pp. 1299-1317, 1992.
- Chen, F.-C., and C.-C. Liu, "Adaptively controlling nonlinear continuous-time systems using multilayer neural networks," *IEEE Trans. Automat. Control*, vol. 39, no. 6, pp. 1306-1310, June 1994.
- Colbaugh, R., H. Seraji, and K. Glass, "A new class of adaptive controllers for robot trajectory tracking," *J. Robotic Systems*, vol. 11, no. 8, pp. 761-772, 1994.
- Colbaugh, R., K. Glass, and H. Seraji, "Performance-based adaptive tracking control of robot manipulators," *J. Robotic Systems*, vol. 12, no. 8, pp. 517-530, 1995.
- Commuri, S., and F.L. Lewis, "CMAC neural networks for control of nonlinear dynamical systems: structure, stability and passivity," *Proc. IEEE Int. Symposium on Intelligent Control*, pp. 123-129, Monterey, Aug. 1995.
- Dawson, D.M., Z. Qu, F.L. Lewis, and J.F. Dorsey, "Robust control for the tracking of robot motion," *Int. J. Control*, vol. 52, no. 3, pp. 581-595, 1990.
- Craig, J., *Adaptive Control of Mechanical Manipulators*, Addison-Wesley, Reading, MA, 1985.
- Cybenko, G., "Approximation by superpositions of a sigmoidal function," *Mathematics of Control. Signals and Systems*, vol. 2, no. 4, pp. 303-314, 1989.
- Hornik, K., M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359-366, 1989.
- Ioannou, P.A., and P.V. Kokotovic, "Instability analysis and improvement of robustness of adaptive control," *Automatica*, vol. 20, no. 5, pp. 583-594, 1984.
- Kim, Y.H., and F.L. Lewis, "Output feedback control of rigid robots using dynamic neural networks," *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 1923-1928, Minneapolis, April 1996.
- Kreisselmeier, G., and B. Anderson, "Robust model reference adaptive control," *IEEE Trans. Automat. Control*, vol. AC-31, no. 2, pp. 127-133, Feb. 1986.
- Landau, Y.D., *Adaptive Control: The Model Reference Approach*, Marcel Dekker, Inc., Basel, 1979.
- Lewis, F.L., C.T. Abdallah, and D.M. Dawson, *Control of Robot Manipulators*, Macmillan, New York, 1993.

*Lewis, F.L., K. Liu, and A. Yeşildirek, "Neural net robot controller with guaranteed tracking performance," IEEE Trans. Neural Networks, vol. 6, no. 3, pp. 703-715, 1995.*

*Lewis, F.L., A. Yeşildirek, and K. Liu, "Neural net robot controller: structure and stability proofs," J. Intelligent and Robotic Sys., vol. 12, pp. 277-299, 1995.*

*Lewis, F.L., A. Yeşildirek, and K. Liu, "Multilayer neural net robot controller: structure and stability proofs," IEEE Trans. Neural Networks, vol. 7, no. 2, pp. 1-12, Mar. 1996.*

*Lewis, F.L., "Neural network control of robot manipulators", IEEE Expert special track on "Intelligent Control" pp. 64-75, ed. K. Passino and Ü. Özgüner, Jun 1996.*

*MATLAB version 4.2, July 1994, The Mathworks, Inc., 24 Prime Park Way, Natick, MA 01760, USA.*

*Miller, W.T., R.S. Sutton, P.J. Werbos, ed., Neural Networks for Control, MIT Press, Cambridge, 1991.*

*Miyamoto, H., M. Kawato, T. Setoyama, and R. Suzuki, "Feedback-error-learning neural network for trajectory control of a robotic manipulator," Neural Networks, vol. 1, pp. 251-265, 1988.*

*Narendra, K.S., "Adaptive control using neural networks," in Neural Networks for Control, pp 115-142, ed. W.T. Miller, R.S. Sutton, P.J. Werbos, Cambridge: MIT Press, 1991.*

*Narendra, K.S., "Adaptive control of dynamical systems using neural networks," in Handbook of Intelligent Control, pp. 141-183, ed. D.A. White and D.A. Sofge, New York: Van Nostrand Reinhold, 1992.*

*Narendra, K.S., and A.M. Annaswamy, "A new adaptive law for robust adaptation without persistent excitation," IEEE Trans. Automat. Control, vol. AC-32, no. 2, pp. 134-145, Feb. 1987.*

*Narendra, K.S., and A.M. Annaswamy, "Stable Adaptive Systems, Prentice-Hall, New Jersey, 1989.*

*Narendra, K.S., and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," IEEE Trans. Neural Networks, vol. 1, pp. 4-27, Mar. 1990.*

*Narendra, K.S., and K. Parthasarathy, "Gradient methods for the optimization of dynamical systems containing neural networks," IEEE Trans. Neural Networks, vol. 2, no. 2, pp. 252-262, Mar. 1991.*

*Peterson, B.B., and K.S. Narendra, "Bounded error adaptive control," IEEE Trans. Automat. Control, vol. 27, pp. 1161-1168, Dec. 1982.*

- Polycarpou, M.M., and P.A. Ioannou, "Identification and control using neural network models: design and stability analysis," Tech. Report 91-09-01, Dept. Elect. Eng. Sys., Univ. S. Cal., Sept. 1991.*
- Rovithakis, G.A., and M.A. Christodoulou, "Adaptive control of unknown plants using dynamical neural networks," IEEE Trans. Systems, Man, and Cybernetics, vol. 24, no. 3, pp. 400-412, 1994.*
- Rumelhart, D.E., G.E. Hinton, and R.J. Williams, "Learning internal representations by error propagation," in Parallel Distributed Processing, ed. D.E. Rumelhart and J.L. McClelland, Cambridge, MA: MIT Press, 1986.*
- Sadegh, N., "A perceptron network for functional identification and control of nonlinear systems," IEEE Trans. Neural Networks, vol. 4, no. 6, pp. 982-988, Nov. 1993.*
- Sanner, R.M., and J.-J.E. Slotine, "Stable adaptive control and recursive identification using radial gaussian networks," Proc. IEEE Conf. Decision and Control, Brighton, 1991.*
- Sastry, S., and M. Bodson, Adaptive Control, Prentice-Hall, New Jersey, 1989.*
- Slotine, J.-J.E., and W. Li, Applied Nonlinear Control, Prentice-Hall, New Jersey, 1991.*
- Werbos, P.J., Beyond Regression: New Tools for Prediction and Analysis in the Behavior Sciences, Ph.D. Thesis, Committee on Appl. Math. Harvard Univ., 1974.*
- Werbos, P.J., "Back propagation: past and future," Proc. 1988 Int. Conf. Neural Nets, vol. 1, pp. I343-I353, 1989.*
- White, D.A., and D.A. Sofge, ed. Handbook of Intelligent Control, Van Nostrand Reinhold, New York, 1992.*
- Yeşildirek, A., Nonlinear Systems Control Using Neural Networks, Ph.D. Thesis, Dept. of Electrical Engineering, The University of Texas at Arlington, Arlington, Texas 76019, USA, 1994.*
- Zbikowski, R., and K.J. Hunt, Neural Adaptive Control Technology, World Scientific, Singapore, 1996.*

## 4.8 PROBLEMS

### Section 4.2

**Problem 4.2-1 : Observability Properties and Bounds on the State.** *Prove Lemma 4.2.1. Begin by writing the solution  $x(t)$  of the system  $(0, B(t), C(t))$ . Note that the state transition matrix is the identity matrix. Now use integral bounds to obtain a bound on the state in terms of  $\|y\|$  and  $\|u\|$ .*

**Problem 4.2-2 : Simulation of FLNN Controller.** *Write a MATLAB M file to simulate the FLNN controller. Perform the simulations in Example 4.2.1. Add friction to the robot model and see how the performance of the FLNN controller is affected.*

### Section 4.3

**Problem 4.3-1 : Taylor Series and Disturbance Bounds.** *Prove the bounding Lemmas 4.3.1 and 4.3.2.*

**Problem 4.3-2 : Simulation of Two-Layer NN Controller.** *Write a MATLAB M file to simulate the two-layer NN controller. Perform the simulations in Example 4.3.1. Add friction to the robot model and see how the performance of the NN controller is affected.*

### Section 4.5

**Problem 4.5-1 : Proof of SSP Bound.** *Verify the SSP inequality (4.5.2).*

**Problem 4.5-2 : Passivity of FLNN Controller.** *Show that the FLNN controller tuning algorithm in Table 4.2.1 makes the system passive while the augmented tuning algorithm in Table 4.2.2 yields SSP, so that no PE is needed.*

**Problem 4.5-3 : Passivity of Hebbian Controller.** *Investigate the passivity of the Hebbian controller tuning algorithm in Table 4.3.3.*

## Chapter 5

# Neural Network Robot Control: Applications and Extensions

*So far in the book we have focused on control of rigid-link robot arms, which have dynamics of the form*

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = \tau \quad (5.0.1)$$

*or*

$$M(q)\ddot{q} + N(q, \dot{q}) + \tau_d = \tau, \quad (5.0.2)$$

*where*

$$N(q, \dot{q}) \equiv V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) \quad (5.0.3)$$

*is the vector of the nonlinear terms. In these dynamics,  $M(q)$  is the inertia matrix,  $V_m(q, \dot{q})$  is the Coriolis/centripetal matrix,  $F(\dot{q})$  are the friction terms,  $G(q)$  is the gravity vector, and  $\tau_d(t)$  represents disturbances. The rigid robot dynamics enjoy the properties in Chapter 3, which are reproduced here in Table 5.0.1.*

We have so far considered the selection of the control torques  $\tau(t)$  so that the robot joint variables  $q(t)$  follow a prescribed desired position trajectory  $q_d(t)$ . In Chapter 4 we showed how to design neural network (NN) controllers for rigid robot arms. In this chapter we shall first extend our results to force control. Then, we shall consider the control of more complex robotic systems with practical relevance, including flexible-link arms, and systems with actuator dynamics including joint flexibility (e.g. compliant coupling shaft or gearing) and motor electrical dynamics. These systems are characterized by the fact that they have more degrees of freedom than control inputs. We shall see that the NN controllers can be extended to such systems with reduced control effectiveness by two basic approaches: singular perturbations and backstepping. The result in every case is a NN controller of the same basic structure as those in Chapter 4, but with additional feedback loops to handle the more complex dynamics.

All of the controller designs in this book assume full state feedback, that is, all internal states of the plants being controlled are measurable. If there is reduced

Table 5.0.1: Properties of Robot Arm Dynamics

- 
- P1** The inertia matrix  $M(q)$  is symmetric, positive definite, and bounded so that  $\mu_1 I \leq M(q) \leq \mu_2 I$  for all  $q(t)$ . For revolute joints, the only occurrences of the joint variables  $q_i$  are as  $\sin(q_i), \cos(q_i)$ . For arms with no prismatic joints, the bounds  $\mu_1, \mu_2$  are constants.
- P2** The Coriolis/centripetal vector  $V_m(q, \dot{q})\dot{q}$  is quadratic in  $\dot{q}$ .  $V_m$  is bounded so that  $\|V_m\| \leq v_B \|\dot{q}\|$ , or equivalently  $\|V_m \dot{q}\| \leq v_B \|\dot{q}\|^2$ .
- P3** The Coriolis/centripetal matrix can always be selected so that the matrix  $S(q, \dot{q}) \equiv \dot{M}(q) - 2V_m(q, \dot{q})$  is *skew symmetric*. Therefore,  $x^T S x = 0$  for all vectors  $x$ . This is a statement of the fact that the fictitious forces in the robot system do no work.
- P4** The friction terms have the approximate form
- $$F(\dot{q}) = F_v \dot{q} + F_d(\dot{q}),$$
- with  $F_v$  a diagonal matrix of constant coefficients representing the viscous friction, and  $F_d(\cdot)$  a vector with entries like  $K_{d_i} \text{sgn}(\dot{q}_i)$ , with  $\text{sgn}(\cdot)$  the signum function and  $K_{d_i}$  the coefficients of dynamic friction. These friction terms are bounded so that  $\|F(\dot{q})\| \leq f_B \|\dot{q}\| + k_B$  for constants  $f_B, k_B$ .
- P5** The gravity vector is bounded so that  $\|G(q)\| \leq g_B$ . For revolute joints, the only occurrences of the joint variables  $q_i$  are as  $\sin(q_i), \cos(q_i)$ . For revolute joint arms the bound  $g_B$  is a constant.
- P6** The disturbances are bounded so that  $\|\tau_d(t)\| \leq d_B$ .
- 

*measurement information so that some states are not directly measurable, then additional complications arise in feedback controls design. In this case of output feedback control, one may add a dynamic neural net to reconstruct the missing states. This may be accomplished as detailed by Kim (1997).*

## 5.1 FORCE CONTROL USING NEURAL NETWORKS

*In many industrial applications it is desired for the robot to exert a prescribed force normal to a given surface while following a prescribed motion trajectory tangential to the surface. This is the case in surface finishing, grinding, polishing, etc. A hybrid position/force controller can be designed by extension of the neural network design principles in Chapter 4. This section represents the work of Dr. Chiman Kwan (see references).*

*The force control problem is discussed in McClamroch and Wang (1988), Spong and Vidyasagar (1989), and Lewis, Abdallah, and Dawson (1993). The robot dynamics with environmental contact on a prescribed surface can be described by*

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = \tau + J^T(q)\lambda, \quad (5.1.1)$$

*where  $q(t) \in \Re^n$ ,  $J(q)$  is a Jacobian matrix associated with the contact surface geometry, and  $\lambda$  (the so-called ‘Lagrange multiplier’) is a vector of contact forces*

exerted normal to the surface, described in coordinates relative to the surface. Vector  $\tau_d(t)$  represents disturbances. In looking at this equation one should recall the force transformation from Chapter 3,

$$\tau_f = J^T(q)\lambda, \quad (5.1.2)$$

where  $J(q)$  is a Jacobian matrix and  $\tau_f(t)$  is the force in joint coordinates corresponding to  $\lambda(t)$ .

### 5.1.1 Force Constrained Motion and Error Dynamics

The hybrid position/force control problem is to follow a prescribed motion trajectory tangential to the surface while exerting a prescribed contact force normal to the surface. To solve this problem it is necessary to understand end-effector motion constrained to a prescribed surface and then derive the associated error dynamics.

#### 5.1.1.1 Constrained Motion on a Surface

Let

$$y = h(q) \quad (5.1.3)$$

be the Cartesian position of the end of the arm, with  $h(q)$  the kinematics transformation (Chapter 3). Then the prescribed surface can be described by the geometric holonomic constraint equation

$$\phi(y) = 0, \quad (5.1.4)$$

where  $\phi(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}^m$ . The constraint Jacobian matrix

$$J(q) \equiv \frac{\partial(\phi(h(q)))}{\partial q} \quad (5.1.5)$$

describes the joint velocities when the arm moves on the surface; in fact, the normal velocity is  $J(q)\dot{q} = 0$ .

The constraint equation reduces the number of degrees of freedom to

$$n_1 \equiv n - m. \quad (5.1.6)$$

Let the motion in the plane of the surface be described by the reduced position variable  $q_1(t) \in \mathbb{R}^{n_1}$ . Then, according to the implicit function theorem, on the surface  $\phi(q) = 0$  one may find a function  $\gamma(\cdot)$  such that

$$q_2 = \gamma(q_1), \quad (5.1.7)$$

where  $q_2(t) \in \mathbb{R}^m$  represents dependent variables. Note that  $q = [q_1^T \ q_2^T]^T$ .

The robot, constrained for motion along the surface, satisfies a reduced-order dynamics in terms of  $q_1(t) \in \mathbb{R}^{n_1}$ . To find these dynamics, define the extended Jacobian

$$L(q_1) \equiv \left[ \begin{array}{c} I_{n_1} \\ \frac{\partial \gamma}{\partial q_1} \end{array} \right], \quad (5.1.8)$$

where  $I_{n_1}$  is the  $n_1 \times n_1$  identity matrix. Then the relation of the tangential velocity  $\dot{q}_1$  to the full joint velocity  $\dot{q}$  is given via

$$\dot{q} = L(q_1)\dot{q}_1, \quad (5.1.9)$$

and one also has

$$\ddot{q} = L(q_1)\ddot{q}_1 + \dot{L}(q_1)\dot{q}_1. \quad (5.1.10)$$

Substituting these expressions into (5.1.1) yields the reduced-order dynamics on the surface given by

$$M(q_1)L(q_1)\ddot{q}_1 + V_1(q_1, \dot{q}_1)\dot{q}_1 + F(q_1) + G(q_1) + \tau_d = \tau + J^T(q_1)\lambda, \quad (5.1.11)$$

where  $V_1 = V_m L + M \dot{L}$  and one has implicitly used (5.1.7) to compute  $q(t)$  in terms of  $q_1(t)$ , so that the functional dependence is shown in terms of  $q_1(t)$ .

Premultiplying this equation by  $L^T$  yields

$$\bar{M}\ddot{q}_1 + \bar{V}_1\dot{q}_1 + \bar{F} + \bar{g} + \bar{\tau}_d = L^T\tau, \quad (5.1.12)$$

where  $\bar{M} = L^T M L$ ,  $\bar{V}_1 = L^T V_1 = L^T(V_m L + M \dot{L})$ ,  $\bar{F} = L^T F$ ,  $\bar{G} = L^T G$ ,  $\bar{\tau}_d = L^T \tau_d$ . One has used the fact that

$$J(q_1)L(q_1) = 0. \quad (5.1.13)$$

This dynamics describes motion in the plane of the constraint surface, with force information removed. It is not difficult (see Problems section) to show that (5.1.12) satisfies the properties in Table 5.0.1, specifically that  $\bar{M}$  is positive definite symmetric and bounded above and below, and  $\bar{M} - 2\bar{V}_1$  is skew symmetric. This equation may profitably be compared to the Cartesian dynamics in Chapter 3.

### 5.1.1.2 Filtered Error Dynamics

The hybrid position/force control problem is to follow a prescribed motion trajectory  $q_{1_d}(t) \in \mathbb{R}^{n_1}$  tangential to the surface while exerting a prescribed contact force  $\lambda_d(t) \in \mathbb{R}^m$  normal to the surface. To design a position/force controller we may follow the basic filtered error approximation-based approach laid out in Chapter 3. Thus, define the motion error in the plane of the surface

$$e_m = q_{1_d} - q_1 \quad (5.1.14)$$

and the filtered motion error

$$r = \dot{e}_m + \Lambda e_m, \quad (5.1.15)$$

where  $\Lambda$  is a positive diagonal design matrix. Define the force error as

$$\tilde{\lambda} = \lambda_d - \lambda, \quad (5.1.16)$$

where  $\lambda(t)$  is the normal force measured in a coordinate frame attached to the surface.

Now, differentiate the filtered tracking error  $r(t)$  and substitute from (5.1.12) to see that the constrained dynamics may be written in terms of  $r(t)$  as

$$\bar{M}\dot{r} = -\bar{V}_1 r + L^T f(x) + L^T \tau_d - L^T \tau \quad (5.1.17)$$

where the nonlinear robot function is

$$f(x) = M(q_1)L(q_1)(\ddot{q}_{1_d} + \Lambda\dot{e}_m) + V_1(q_1, \dot{q}_1)(\dot{q}_{1_d} + \Lambda e_m) + F(\dot{q}_1) + G(q_1). \quad (5.1.18)$$

This is the sought form of the error dynamics. Equation (5.1.7) has implicitly been used to compute  $q(t)$  in terms of  $q_1(t)$ , so that the functional dependence is shown in terms of  $q_1(t)$ . Vector  $x$  contains all the time signals needed to compute  $f(\cdot)$ , and may be defined for instance as  $x \equiv [e_m^T \dot{e}_m^T q_{1_d}^T \dot{q}_{1_d}^T]^T$ . It is important to note that  $f(x)$  contains all the potentially unknown robot arm parameters, except for the  $\bar{V}_1 r$  term in (5.1.17), which cancels out in controller stability Lyapunov proofs.

The constrained error dynamics is given by (5.1.17). To prove that the force error is small using the proposed control scheme, it will be necessary to work also with (5.1.11), which contains force information. Therefore, differentiate the filtered tracking error  $r(t)$  and substitute from (5.1.11) to see that this equation may be written in terms of  $r(t)$  as

$$ML\dot{r} = -V_1 r + f(x) + \tau_d - \tau - J^T \lambda. \quad (5.1.19)$$

### 5.1.2 Neural Network Hybrid Position/Force Controller

A hybrid position/force controller has the structure

$$\tau = \hat{f} + K_v L(q_1)r - J^T[\lambda_d + K_f \tilde{\lambda}] - v, \quad (5.1.20)$$

where the position gain  $K_v$  and the force gain  $K_f$  are positive definite matrices. The nonlinear function estimate  $\hat{f}$  and the robustifying term  $v(t)$  can be selected using any of the techniques mentioned in Chapters 3 and 4, including adaptive control, robust control, neural network control, and so on. A simplified controller that may work in some applications is obtained by setting  $\hat{f} = 0, v = 0$ , and increasing the PD motion gain  $K_v$  and force gain  $K_f$ .

A neural network (NN) controller is obtained if the functional estimate is manufactured using a NN. Here, we shall use the two-layer NN shown in Fig. 5.1.1, which has first-layer weight matrix  $V$  and second-layer weight matrix  $W$ . The thresholds are included in the weight matrices by augmenting the input signals to the NN layers by the entry ‘1’, as described in Chapter 1. This NN is nonlinear in the tunable NN weights  $V$ . Since the unknown nonlinear function (5.1.18) is smooth, according to the NN universal approximation property in Chapter 1 there exist ideal (target) NN weights  $V, W$  such that  $f(x)$  is given on a compact set  $\mathcal{S}$  by

$$f(x) = W^T \sigma(V^T x) + \varepsilon, \quad (5.1.21)$$

where the functional estimation error  $\varepsilon$  is bounded by

$$\|\varepsilon\| \leq \varepsilon_N \quad (5.1.22)$$

for  $x \in \mathcal{S}$ , with  $\varepsilon_N$  a known positive constant. We call  $\varepsilon_N$  the NN approximation inaccuracy. This NN approximation property replaces the usual linear-in-the-parameters (LIP) assumptions required in adaptive control, and means that no regression matrix need be computed. As the number of hidden-layer neurons increases, the approximation accuracy increases and  $\varepsilon_N$  decreases.

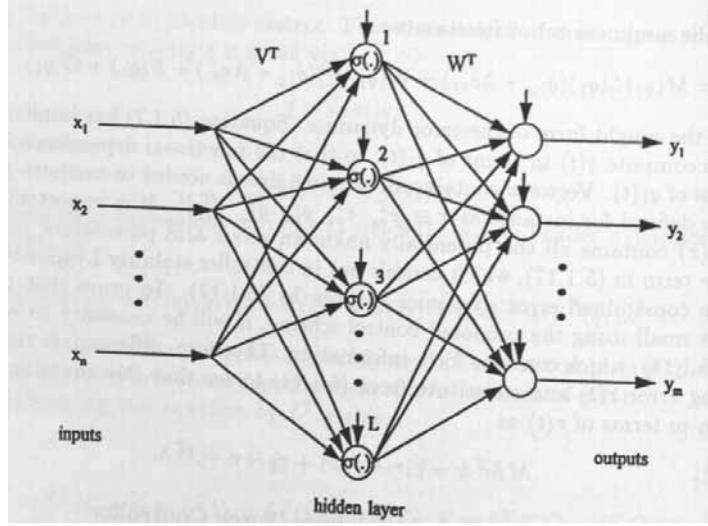


Figure 5.1.1: Two-layer neural net.

The NN controller is given by

$$\tau = \hat{W}^T \sigma(\hat{V}^T x) + K_v L(q_1) r - J^T [\lambda_d + K_f \tilde{\lambda}] - v, \quad (5.1.23)$$

where  $\hat{V}, \hat{W}$  are the current weights in the actual NN as provided by the weight tuning algorithms. The tuning algorithms and the robustifying term  $v(t)$  must be derived to guarantee system stability. The NN controller is shown in Fig. 5.1.2; it has the basic structure of the NN controllers in Chapter 4, but with an additional inner force control loop.

#### 5.1.2.1 Closed-Loop Error Dynamics Using NN Controller

The closed-loop error dynamics using this control algorithm are found by substitution of (5.1.23) into (5.1.17), and subsequent use of (5.1.13), to be

$$\bar{M}\dot{r} = -L^T K_v L r - \bar{V}_1 r + L^T [W^T \sigma(V^T x) - \hat{W}^T \sigma(\hat{V}^T x)] + L^T [\tau_d + \varepsilon + v]. \quad (5.1.24)$$

Now the manipulations closely follow those in Chapter 4. By expanding  $\sigma(V^T x)$  in a Taylor series about  $\sigma(\hat{V}^T x)$  and some other manipulations one expresses the closed-loop error dynamics (see Problems section) as

$$\bar{M}\dot{r} = -L^T K_v L r - \bar{V}_1 r + L^T [\tilde{W}^T (\hat{\sigma} - \hat{\sigma}' \hat{V}^T x) + \hat{W}^T \hat{\sigma}' \tilde{V}^T x] + L^T [w + v] \quad (5.1.25)$$

where the disturbance term is

$$w(t) = \tilde{W}^T \hat{\sigma}' V^T x + W^T O(\tilde{V}^T x)^2 + \tau_d + \varepsilon. \quad (5.1.26)$$

The expression  $O(z)^2$  denotes terms of order two in  $z$ . In these expressions, the NN weight estimation errors are given by

$$\tilde{V} = V - \hat{V} \quad \tilde{W} = W - \hat{W} \quad (5.1.27)$$

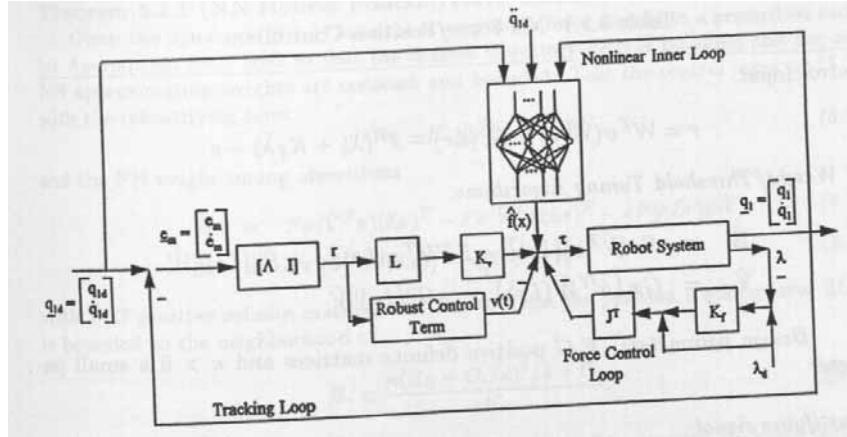


Figure 5.1.2: Neural net hybrid position/force controller.

and

$$\hat{\sigma} \equiv \sigma(\hat{V}^T x) \quad (5.1.28)$$

$$\hat{\sigma}' \equiv \left. \frac{\partial \sigma(z)}{\partial z} \right|_{z=\hat{V}^T x}. \quad (5.1.29)$$

To bound the disturbance term  $w(t)$ , it is now assumed that various bounding assumptions hold. In particular, the bounds in Table 5.0.1 are needed along with the next assumptions, which hold in practical situations. Recall the definition of the Frobenius norm  $\|\cdot\|_F$  from Chapter 2.

**Assumption 5.1.1 (Desired Trajectory and NN Target Weight Bounds) :**

a. The desired motion trajectory is bounded so that

$$\begin{vmatrix} q_{1d}(t) \\ \dot{q}_{1d}(t) \\ \ddot{q}_{1d}(t) \end{vmatrix} \leq q_B, \quad (5.1.30)$$

with  $q_B$  a known scalar bound.

b. Define the matrix of ideal target NN weights as

$$Z \equiv \begin{bmatrix} W & 0 \\ 0 & V \end{bmatrix}. \quad (5.1.31)$$

Then, the ideal NN weights are constant and bounded so that

$$\|Z\|_F \leq Z_B \quad (5.1.32)$$

with  $Z_B$  a known bound.  $\square$

Table 5.1.1: NN Force/Position Controller.

*Control Input:*

$$\tau = \hat{W}^T \sigma(\hat{V}^T x) + K_v(Lr) - J^T(\lambda_d + K_f \tilde{\lambda}) - v$$

*NN Weight/Threshold Tuning Algorithms:*

$$\begin{aligned}\dot{\hat{W}} &= F\sigma(\hat{V}^T x)(Lr)^T - F\hat{\sigma}'\hat{V}^T x(Lr)^T - \kappa F\|(Lr)\|\hat{W} \\ \dot{\hat{V}} &= Gx\left(\hat{\sigma}'^T \hat{W}(Lr)\right)^T - \kappa G\|(Lr)\|\hat{V}\end{aligned}$$

*Design parameters:*  $F, G$  positive definite matrices and  $\kappa > 0$  a small parameter.

*Robustifying signal:*

$$v(t) = -K_z(\|\hat{Z}\|_F + Z_B)r$$

Define also the overall NN weight estimate and error matrices

$$\hat{Z} \equiv \begin{bmatrix} \hat{W} & 0 \\ 0 & \hat{V} \end{bmatrix}. \quad \tilde{Z} \equiv \begin{bmatrix} \tilde{W} & 0 \\ 0 & \tilde{V} \end{bmatrix}. \quad (5.1.33)$$

Now, it can be shown that the disturbance term  $w(t)$  is bounded by

$$\|w(t)\| \leq C_0 + C_1\|\tilde{Z}\|_F + C_2\|\tilde{Z}\|_F \cdot \|r\|, \quad (5.1.34)$$

with  $C_0, C_1, C_2$  computable constants expressed in terms of various bounds. For instance, one can compute that

$$C_0 = \varepsilon_N + d_B + c_0 Z_B. \quad (5.1.35)$$

This interesting expression shows that the disturbances  $w(t)$  driving the error system increase as the disturbance torque  $\tau_d(t)$  increases, or as the NN functional estimation error  $\varepsilon(t)$  increases (which occurs, for instance, if the number of hidden-layer neurons is decreased, e.g. to save in computations).

### 5.1.2.2 NN Weight Tuning Algorithms for Guaranteed Stability

Based on the machinery just developed it is now possible to complete the design of the hybrid position/force controller in Fig. 5.1.2 by providing NN weight tuning algorithms for  $\hat{W}, \hat{V}$  and also a selection for the robustifying signal  $v(t)$  such that tracking performance and internal stability are both guaranteed. The next theorem provides the details; the NN controller it describes is given in Table 5.1.1. The complete proof of the theorem is given in Kwan and Lewis (1994).

**Theorem 5.1.1 (NN Hybrid Position/Force Controller) :**

Given the dynamics (5.1.1) for a robot exerting a force normal to a prescribed surface, let Assumption 5.1.1 hold so that the desired trajectory  $q_d(t)$  is bounded and the target NN approximating weights are constant and bounded. Take the control input as (5.1.23) with the robustifying term

$$v(t) = -K_z(\|\hat{Z}\|_F + Z_B)r \quad (5.1.36)$$

and the NN weight tuning algorithms

$$\dot{\hat{W}} = F\sigma(\hat{V}^T x)(Lr)^T - F\hat{\sigma}'\hat{V}^T x(Lr)^T - \kappa F\|(Lr)\|\hat{W} \quad (5.1.37)$$

$$\dot{\hat{V}} = Gx(\hat{\sigma}'^T \hat{W}(Lr))^T - \kappa G\|(Lr)\|\hat{V} \quad (5.1.38)$$

with  $F, G$  positive definite matrices and  $\kappa > 0$ . Then the position tracking error  $\|Lr(t)\|$  is bounded to the neighborhood of

$$B_r \equiv \frac{\kappa(Z_B + C_1/\kappa)^2/4 + C_0}{K_v}, \quad (5.1.39)$$

the force tracking error  $\tilde{\lambda}(t)$  is bounded, and the NN weight estimates  $\hat{W}, \hat{V}$  are bounded. Moreover, the position tracking error can be made as small as desired by increasing the gain  $K_v$ , and the force tracking error can be made as small as desired by increasing the gain  $K_f$ .

Outline of Proof:

The proof is similar to the proofs that are detailed in Chapter 4.

a. Boundedness of position error and NN weights.

Define a Lyapunov function candidate

$$L = \frac{1}{2}r^T \bar{M}r + \frac{1}{2}tr(\tilde{W}^T F^{-1} \tilde{W}) + \frac{1}{2}tr(\tilde{V}^T G^{-1} \tilde{V})$$

with  $tr(\cdot)$  the matrix trace and  $F, G$  symmetric positive definite matrices. Differentiate, substitute from the error dynamics (5.1.25), and use skew-symmetry to obtain

$$\begin{aligned} \dot{L} = & -r^T L^T K_v L r + tr\left(\tilde{W}^T [F^{-1} \dot{\tilde{W}} + (\hat{\sigma} - \hat{\sigma}' \hat{V}^T x)(Lr)^T]\right) \\ & + tr\left(\tilde{V}^T [G^{-1} \dot{\tilde{V}} + x(\hat{\sigma}'^T \hat{W}(Lr))^T]\right) + r^T L^T [w + v]. \end{aligned}$$

Using the proposed tuning algorithms, recalling (5.1.27), and employing the assumption that  $\dot{V} = 0, \dot{W} = 0$  yields

$$\dot{L} = -r^T L^T K_v L r + \kappa \|Lr\| (tr[\tilde{W}^T (W - \tilde{W})] + tr[\tilde{V}^T (V - \tilde{V})]) + r^T L^T [w + v].$$

At this point one uses various norm inequalities and the bounds (5.1.34), in similar fashion to the proofs detailed in Chapter 4, to obtain the fact that  $\dot{L} \leq 0$  if either

$$\|Lr\| > B_r$$

or

$$\|\tilde{Z}\|_F > B_Z$$

where  $B_Z$  is a bound given in Kwan and Lewis (1994). This proves boundedness of the position error and the NN weights.

b. Boundedness of the force error.

To show boundedness of the force tracking error  $\tilde{\lambda}(t)$  we use an approach that can be compared to Barbalat's extension in Chapter 2. Thus, note first that in part a. of the

proof we have shown that all quantities on the right-hand side of (5.1.25) are bounded. Therefore, from the invertibility of  $\bar{M}$  it follows that  $\dot{r}$  is bounded. Now, substitute the control (5.1.23) into the error dynamics (5.1.19) to obtain

$$\begin{aligned} ML\dot{r} &= -V_1r + f + \tau_d - J^T\lambda - \hat{W}^T\sigma(\hat{V}^Tx) - K_vLr + J^T[\lambda_d + K_f\tilde{\lambda}] + v \\ ML\dot{r} &= -K_vLr - V_1r + \hat{W}^T(\hat{\sigma} - \hat{\sigma}'\hat{V}^Tx) + \hat{W}^T\hat{\sigma}'\hat{V}^Tx + J^T[I + K_f]\tilde{\lambda} + (w + v). \end{aligned}$$

This may be written as

$$J^T[I + K_f]\tilde{\lambda} = ML\dot{r} + K_vLr + V_1r - \hat{W}^T(\hat{\sigma} - \hat{\sigma}'\hat{V}^Tx) - \hat{W}^T\hat{\sigma}'\hat{V}^Tx - (w + v) \equiv B(\dot{r}, x, \tilde{Z}).$$

where all quantities on the right-hand side are bounded. Therefore, premultiply by  $J$  to derive

$$\begin{aligned} JJ^T[I + K_f]\tilde{\lambda} &= JB(\dot{r}, x, \tilde{Z}) \\ \tilde{\lambda} &= [I + K_f]^{-1}(JJ^T)^{-1}JB(\dot{r}, x, \tilde{Z}), \end{aligned}$$

where we have used the fact that  $JJ^T$  is nonsingular. This expression shows that the force tracking error  $\tilde{\lambda}(t)$  is bounded and can be made as small as desired by increasing the force tracking gain  $K_f$ .  $\square$

*This proof has shown that the Lyapunov derivative is negative if  $\|r\|$  or  $\|\tilde{Z}\|_F$  are above some bounds, namely  $B_r$  in (5.1.39) and  $B_Z$  respectively. Thus,  $B_r, B_Z$  may be interpreted as practical bounds for the filtered position tracking error and the NN weights. The form of the bounds shows that  $B_r$  may be made as small as desired by increasing the position feedback gains  $K_v$ . Note that the dependence of  $B_r$  on  $C_0$  in (5.1.35) shows that the tracking errors increase if either the robot arm disturbance torque  $\tau_d(t)$  increases, or the NN approximation inaccuracy  $\varepsilon_N$  increases. The latter occurs, for instance, if a smaller NN is used to avoid computations. However, these effects may be offset by increasing  $K_v$ .*

*As in Chapter 4, a complete analysis shows that this controller is local in the sense that the initial tracking errors must be in a certain set of allowable initial conditions. This initial condition set depends on the speed of the desired trajectory and the size of the compact set over which the approximation property (5.1.21) holds. Approximation accuracy generally increases with the number of hidden-layer neurons in the NN. Technically, this leads also to a minimum requirement on the PD gain (see Chapter 4).*

*Some remarks about the NN hybrid position/force controller in Table 5.1.1 are in order. First, the NN weights can be initialized at zero, so that in Fig. 5.1.2 the PD loop initially stabilizes the system until the NN begins to learn. Therefore, no off-line training phase is needed for this NN controller.*

*Next, note that the NN controller is very much like those in Chapter 4, but using  $Lr$  instead of  $r$  for position tracking. The first terms in tuning algorithms (5.1.37)-(5.1.38) are backpropagation-through-time, with the modified filtered error  $Lr(t)$  the signal being backpropagated. The Jacobian required, namely  $\hat{\sigma}'$  is very easily computed in terms of signals measurable in the feedback system. The last terms in the tuning algorithms correspond to Narendra's e-mod (Narendra and Annaswamy 1987), which is well known to be effective in standard adaptive control algorithms to provide robustness and avoid the need for persistence of excitation. The middle term in the tuning algorithm for  $\hat{W}$  is new. It is required due to the nonlinear dependence of  $\hat{f}$  on the first-layer NN weights  $\hat{V}$ .*

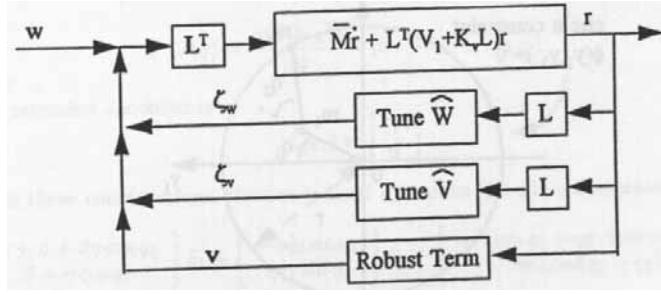


Figure 5.1.3: Closed-loop position error system.

Note finally that boundedness of the tracking position error, the force error, and the NN weights guarantees that the proposed control input (5.1.23) is bounded. More discussion on NN controllers of this form is provided in Chapter 4.

#### 5.1.2.3 Passivity of the NN Hybrid Position/Force Controller

The NN hybrid position/force controller has the same passivity properties as the NN controllers in Chapter 4. The closed-loop error system (5.1.25) has the structure shown in Fig. 5.1.3, where all blocks are interconnected in a feedback configuration and

$$\zeta_W \equiv \tilde{W}^T(\hat{\sigma} - \hat{\sigma}'\tilde{V}^T x) \quad (5.1.40)$$

$$\zeta_V \equiv \hat{W}^T\hat{\sigma}'\tilde{V}^T x. \quad (5.1.41)$$

Passivity and state-strict passivity (SSP) were defined in Chapter 2. It was shown in Chapter 3 that the filtered error dynamics are a state-strict passive system; it is easily shown that the constrained filtered position error system in the top block in the figure is also state-strict passive. The next result details the passivity properties of the position/force controller. The proof is given in Kwan and Lewis (1994) and follows closely similar proofs in Chapter 4.

#### Theorem 5.1.2 (Passivity Properties of NN Force Controller) :

Under the hypotheses of Theorem 5.1.1, the NN weight tuning algorithms in Table 5.1.1 make the maps from  $Lr(t)$  to  $\zeta_W$  and from  $Lr(t)$  to  $\zeta_V$  both SSP maps.  $\square$

Since both the filtered error dynamics and the tuning algorithms are SSP, it follows that the overall closed-loop system is SSP. Therefore, all signals and internal states (e.g. the NN weights) are bounded. The SSP property explains why no persistence of excitation (PE) is needed for this controller, as SSP guarantees weight boundedness without an observability-like condition.

It should be noted that if one uses only backpropagation weight tuning, which basically amounts to using only the first terms in the tuning algorithms in Table 5.1.1, the closed-loop system is not SSP, but only passive. Therefore, an additional PE condition is needed for the controller to perform correctly.

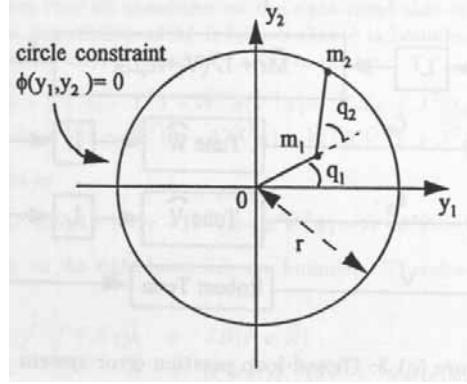


Figure 5.1.4: Two-link planar elbow arm with circle constraint.

### 5.1.3 Design Example for NN Hybrid Position/Force Controller

#### Example 5.1.1 (NN Hybrid Position/Force Controller) :

The two-link planar manipulator in Fig. 5.1.4 has dynamics (see Chapter 3) given by

$$M(q) = \begin{bmatrix} \alpha + \beta + 2\eta \cos q_2 & \beta + \eta \cos q_2 \\ \beta + \eta \cos q_2 & \beta \end{bmatrix}, V_m(q, \dot{q}) = \begin{bmatrix} -\eta \dot{q}_2 \sin q_2 & -\eta(\dot{q}_1 + \dot{q}_2) \sin q_2 \\ \eta \dot{q}_1 \sin q_2 & 0 \end{bmatrix},$$

$$G(q) = \begin{bmatrix} \alpha e_1 \cos q_1 + \eta e_1 \cos(q_1 + q_2) \\ \eta e_1 \cos(q_1 + q_2) \end{bmatrix}.$$

where  $\alpha = (m_1 + m_2)a_1^2$ ,  $\beta = m_2a_2^2$ ,  $\eta = m_2a_1a_2$ ,  $e_1 = g/a_1$ . The specific arm considered in this example has  $\alpha = 0.8$ ,  $\beta = 0.32$ ,  $\eta = 0.4$ ,  $a_1 = 1$ ,  $a_2 = 0.8$ . Friction is not included in this example.

**a. Constraint Surface and Jacobians.** The constraint surface is the circle in Cartesian space  $(y_1, y_2)$  shown in Fig. 5.1.4, which can be expressed as

$$\phi(y) = y_1^2 + y_2^2 - r^2 = 0$$

where  $y \equiv [y_1 \ y_2]^T$ . The transformation from joint space to Cartesian space is given by

$$y = h(q) = \begin{bmatrix} a_1 \cos q_1 + a_2 \cos(q_1 + q_2) \\ a_1 \sin q_1 + a_2 \sin(q_1 + q_2) \end{bmatrix}.$$

Therefore, in joint space the constraint is expressed as

$$\phi(h(q)) = a_1^2 + a_2^2 + 2a_1a_2 \cos q_2 - r^2 = 0,$$

which has a unique constant solution for  $q_2$  given by

$$q_2 = \cos^{-1} \left( \frac{r^2 - a_1^2 - a_2^2}{2a_1a_2} \right) \equiv \gamma(q_1).$$

The Jacobian  $J(q)$  is

$$J(q) = \frac{\partial(\phi(h(q)))}{\partial q} = [0 \quad -2a_1a_2 \sin q_2]$$

and the extended Jacobian is

$$L(q_1) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Using these constructions, the constrained dynamics (5.1.11) are expressed as

$$\begin{aligned} \begin{bmatrix} \alpha + \beta + 2\eta \cos q_2 \\ \beta + \eta \cos q_2 \end{bmatrix} \ddot{q}_1 + \begin{bmatrix} -\eta \dot{q}_2 \sin q_2 \\ \eta \dot{q}_1 \sin q_2 \end{bmatrix} \dot{q}_1 + \begin{bmatrix} \alpha e_1 \cos q_1 + \eta e_1 \cos(q_1 + q_2) \\ \eta e_1 \cos(q_1 + q_2) \end{bmatrix} \\ = \tau + \begin{bmatrix} 0 \\ -2a_1 a_2 \sin q_2 \end{bmatrix} \lambda. \end{aligned}$$

**b. Simulation.** Let the desired motion trajectory be

$$q_{1d} = \begin{cases} -90 + 52.5(1 - \cos 1.26t) \text{ m}, & t \leq 2.5 \text{ sec} \\ 15 \text{ m}, & t > 2.5 \text{ sec} \end{cases}$$

and the desired force be

$$\lambda_d = 10 nt.$$

Take the initial conditions as  $q_1(0) = -70 \text{ deg}$ ,  $q_2(0) = 80 \text{ deg}$ ,  $\dot{q}_1(0) = 0$ ,  $\dot{q}_2(0) = 0$ .

The controller parameters are selected as  $\Lambda = 20$ ,  $K_v = 100$ ,  $K_f = 20$ ,  $F = 10$ ,  $G = 10$ ,  $\kappa = 1$ ,  $K_z = 1$ ,  $Z_B = 4$ . A simulation was performed with MATLAB using the techniques in previous chapters. The motion trajectory  $q_1(t)$  is shown in Fig. 5.1.5a and the force  $\lambda(t)$  exerted in Fig. 5.1.5b. In both cases, after a short initial transient, the desired performance is achieved.  $\square$

## 5.2 ROBOT MANIPULATORS WITH LINK FLEXIBILITY, MOTOR DYNAMICS, AND JOINT FLEXIBILITY

*Some classes of practical robotic systems cannot be controlled using the techniques discussed in Chapters 3 and 4, which were appropriate for rigid-link arms. In this section we introduce some of these systems, including robots with link flexibility (e.g. vibratory modes), and robots with drive train dynamics including motor dynamics and compliant coupling shafts (e.g. joint flexibility). These systems are characterized by the fact that they have more degrees of freedom than control inputs. In subsequent sections we shall show two approaches for controlling such systems with reduced control effectiveness, namely singular perturbations and backstepping.*

### 5.2.1 Flexible-Link Robot Arms

*Flexible-link robotic systems comprise an important class of systems that include lightweight arms for assembly, civil infrastructure bridge/vehicle systems, military tank gun barrel applications, and large-scale space structures. Their key feature is the presence of vibratory modes that tend to disturb or even destabilize the motion. They are described as infinite-dimensional dynamical systems using partial differential equations (PDE); some assumptions make them tractable by allowing one to describe them using an ordinary differential equation (ODE), which is finite-dimensional. Yet, this ODE has a form that is unlike the rigid-arm dynamics (5.0.1) since there are more degrees of freedom than control inputs (i.e. the dimension of*

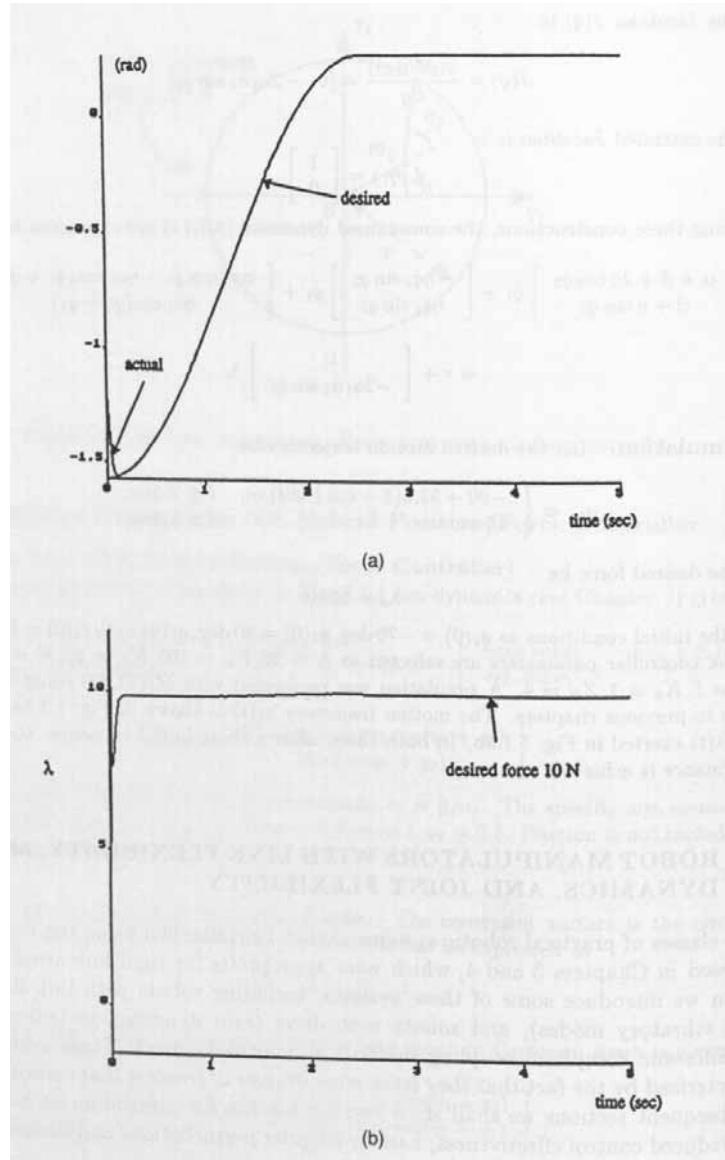


Figure 5.1.5: NN force/position controller simulation results. (a) Desired and actual motion trajectories  $q_{1d}(t)$  and  $q_1(t)$ . (b) Force trajectory  $\lambda(t)$ .

$q(t)$  is greater than the dimension of  $\tau(t)$ , so that special techniques must be used for control systems design. The controller design problem for flexible-link systems is to cause the rigid modes to follow prescribed trajectories while quickly suppressing the vibrations. References for flexible-link dynamics include Asada et al. (1990), Lin and Lewis (1993). References for the control of flexible-link robots include Çetinkunt and Book (1990), Kwon and Book (1990), Madhavan and Singh (1991), Qian and Ma (1992), Siciliano and Book (1988), Spong (1989), Vandegrift et al. (1994), Wang and Vidyasagar (1991), Zhu et al. (1994).

### 5.2.1.1 Derivation of Flexible-Link Robot Dynamics

The elastic deformation  $w(x, t)$  of a flexible beam can be described by the Bernoulli-Euler equation

$$-\frac{\partial^2 w(x, t)}{\partial t^2} = \frac{EI}{\rho A} \frac{\partial^4 w(x, t)}{\partial x^4} \quad (5.2.1)$$

where  $x$  is the position along the beam,  $EI$  is its flexural rigidity,  $\rho$  its mass density, and  $A$  the cross section.

The assumed-mode-shapes method allows one to derive an ODE describing the motion by expressing the deflection as

$$w(x, t) = \sum_{k=1}^N \varphi_k(x) \xi_k(t) \quad (5.2.2)$$

with  $\xi_k(t)$  the mode amplitudes and  $\varphi_k(x)$  the eigenfunctions for mode  $k$ . The eigenfunctions depend on the boundary conditions, and one may assume pinned-pinned modes, clamped-free modes, and so on. The clamped-free modes are said to yield the best approximation for a given value of  $N$ , the number of flexible modes retained. The importance of (5.2.2) is that the two-dimensional function  $w(x, t)$  is decomposed into separate time-dependent parts  $\xi_k(t)$  and position-dependent parts  $\varphi_k(x)$ .

Now, (5.2.2) is substituted into the Bernoulli-Euler PDE, separation of variables is used to solve the PDE, and the eigenfunctions  $\varphi_k(x)$  corresponding to the selected boundary conditions are determined. Finally, Lagrange's equation is used for a robot arm with  $n_r$  flexible links, combining the rigid motion and the flexible motion. Since each link is flexible, one expansion (5.2.2) is used for each link. The result is an ODE in the form

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + Kq + F(\dot{q}) + G(q) + \tau_d = B(q)\tau \quad (5.2.3)$$

where  $q(t) \in \Re^n$  consists of all rigid and flexible mode variables and can be written as

$$q = \begin{bmatrix} q_r \\ q_f \end{bmatrix}, \quad q_f = \begin{bmatrix} q_{f_1} \\ q_{f_2} \\ \vdots \\ q_{f_{n_r}} \end{bmatrix}, \quad (5.2.4)$$

where  $q_r \in \Re^{n_r}$  is the vector of rigid variables, which could be joint angles and extensions exactly as for the rigid-link robot. Vector  $q_f \in \Re^{n_f}$  is the vector of

flexible mode amplitudes  $q_{fi}$ , and  $n = n_r + n_f$ . The deflection vector  $q_{fi}$  for link  $i$  is in  $\mathbb{R}^{N_i}$ , where  $N_i$  is the number of modes retained for the  $i$ -th link in the assumed-mode-shapes expansion (5.2.2). One could write  $q_{fi} = [\xi_1 \ \xi_2 \dots \xi_N]_i^T$  for link  $i$ , where subscript  $i$  denotes that all quantities in the vector should have an additional subscript. One has  $n_f = N_1 + N_2 + \dots + N_{n_r}$ . The link flexibility stiffness matrix is  $K$ . The exact form of all matrices depends on the boundary conditions (e.g. pinned-pinned, pinned-free, clamped-free) chosen by the designer. Vector  $\tau_d(t)$  represents disturbances.

### 5.2.1.2 Properties, Structure, and Non-Minimum Phase Nature of the Flexible-Link Robot Dynamics

The major difference between (5.2.3) and the rigid robot equation (5.0.1) for controls purposes is that the matrix  $B(q)$  has more rows than columns, so that flexible-link arms have reduced control effectiveness and none of the techniques yet discussed can be directly applied. An ameliorating factor is that, like the rigid-link case, it is possible to select the Coriolis/centripetal matrix  $V_m(q, \dot{q})$  so that  $M - 2V_m$  is skew-symmetric, a property that has been seen to be very useful for controls design. Thus, the flexible-link dynamics satisfy the properties in Table 5.0.1.

One may partition the dynamics (5.2.3) according to the rigid and flexible modes as

$$\begin{bmatrix} M_{rr} & M_{rf} \\ M_{fr} & M_{ff} \end{bmatrix} \begin{bmatrix} \ddot{q}_r \\ \ddot{q}_f \end{bmatrix} + \begin{bmatrix} V_{rr} & V_{rf} \\ V_{fr} & V_{ff} \end{bmatrix} \begin{bmatrix} \dot{q}_r \\ \dot{q}_f \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & K_{ff} \end{bmatrix} \begin{bmatrix} q_r \\ q_f \end{bmatrix} + \begin{bmatrix} F_r \\ 0 \end{bmatrix} + \begin{bmatrix} G_r \\ 0 \end{bmatrix} = \begin{bmatrix} B_r \\ B_f \end{bmatrix} \tau. \quad (5.2.5)$$

Note that gravity and friction only act on the rigid modes, while the flexibility effects in  $K$  only effect the flexible modes. The control matrix  $B_r$  is nonsingular regardless of the boundary conditions used in the assumed mode shapes expansion. This equation may also be written as

$$\begin{aligned} M_{rr}\ddot{q}_r + M_{rf}\ddot{q}_f + V_{rr}\dot{q}_r + V_{rf}\dot{q}_f + F_r(\dot{q}_r) + G_r(q_r) &= B_r\tau \\ M_{fr}\ddot{q}_r + M_{ff}\ddot{q}_f + V_{fr}\dot{q}_r + V_{ff}\dot{q}_f + K_{ff}q_f &= B_f\tau. \end{aligned} \quad (5.2.6)$$

In these equations, the flexible-mode matrices have the form

$$V_{ff} = \begin{bmatrix} 2\zeta_1\omega_1 & 0 & 0 & \cdots & 0 \\ 0 & 2\zeta_2\omega_2 & 0 & \cdots & 0 \\ 0 & 0 & 2\zeta_3\omega_3 & \cdots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & \cdots & 2\zeta_{n_r}\omega_{n_r} \end{bmatrix}, \quad K_{ff} = \begin{bmatrix} \omega_1^2 & 0 & 0 & \cdots & 0 \\ 0 & \omega_2^2 & 0 & \cdots & 0 \\ 0 & 0 & \omega_3^2 & \cdots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & \cdots & \omega_{n_r}^2 \end{bmatrix}. \quad (5.2.7)$$

where  $\zeta_i, \omega_i$  are matrices describing the damping ratios and natural frequencies of the flexible modes for link  $i$ . If  $N_i$  modes were retained in modeling link  $i$ , then  $\omega_i, \zeta_i$  are diagonal  $N_i \times N_i$  matrices.

To write a state equation, define

$$\begin{bmatrix} H_{rr} & H_{rf} \\ H_{fr} & H_{ff} \end{bmatrix} = \begin{bmatrix} M_{rr} & M_{rf} \\ M_{fr} & M_{ff} \end{bmatrix}^{-1}, \quad (5.2.8)$$

multiply (5.2.5) by (5.2.8) from the left, rearrange terms, and write

$$\ddot{q}_r = -V_{rr}^1 \dot{q}_r - V_{rf}^1 \dot{q}_f - K_{rf}^1 q_f - F_r^1 - G_r^1 + B_r^1 \tau \quad (5.2.9)$$

$$\ddot{q}_f = -V_{fr}^1 \dot{q}_r - V_{ff}^1 \dot{q}_f - K_{ff}^1 q_f - F_f^1 - G_f^1 + B_f^1 \tau \quad (5.2.10)$$

with:

$$\begin{aligned} V_{rr}^1 &\equiv H_{rr}V_{rr} + H_{rf}V_{fr}, & V_{fr}^1 &\equiv H_{fr}V_{rr} + H_{ff}V_{fr}, \\ V_{rf}^1 &\equiv H_{rr}V_{rf} + H_{rf}V_{ff}, & V_{ff}^1 &\equiv H_{fr}V_{rf} + H_{ff}V_{ff}, \\ K_{rf}^1 &\equiv H_{rf}K_{ff}, & K_{ff}^1 &\equiv H_{ff}K_{ff}, \\ F_r^1 &\equiv H_{rr}F_r, & F_f^1 &\equiv H_{fr}F_r \\ G_r^1 &\equiv H_{rr}G_r, & G_f^1 &\equiv H_{fr}G_r \\ B_r^1 &\equiv H_{rr}B_r + H_{rf}B_f, & B_f^1 &\equiv H_{fr}B_r + H_{ff}B_f. \end{aligned}$$

Now, equations (5.2.9)-(5.2.10) can be placed into the state-space form  $\dot{x} = f(x, u)$  by defining the state, for instance, as  $x = [q \ \dot{q}]^T = [q_r \ q_f \ \dot{q}_r \ \dot{q}_f]^T$ .

**Control Difficulties and Non-Minimum Phase Zero Dynamics.** One immediately sees the major problem in controlling the flexible-link arm. The objective is basically to force the rigid mode variable  $q_r(t)$  to follow a desired trajectory. There are  $n_r$  links, so that  $q_r(t) \in \mathbb{R}^{n_r}$ . The control input available is  $\tau \in \mathbb{R}^{n_r}$ , since there is one actuator per link. However, the extra state  $q_f$  introduces  $n_f$  additional vibratory degrees of freedom that require control to suppress the vibrations. Therefore, the number of inputs is less than the number of degrees of freedom and there is reduced control effectiveness. This is reflected in the fact that the matrix  $B(q)$  in (5.2.3) has more rows than columns. Thus, the techniques so far discussed in Chapters 3 and 4 will not work here.

The situation is even worse than that, for it turns out that by selecting the control input  $\tau(t)$  to achieve practical tracking performance of the rigid variable  $q_r(t)$ , one actually destabilizes the flexible modes  $q_f(t)$ . This is due to the non-minimum phase nature of the zero dynamics of flexible-link robot arms (Madhavan and Singh 1991, Wang and Vidyasagar 1991, Vandegrift et al. 1994). (Zero dynamics and internal dynamics were defined in Chapter 2 under the feedback linearization discussion.)

Two techniques for confronting these problems are given in the next two sections of this chapter. Both these techniques afford approaches for increasing the control effectiveness, and capitalize on the fact that the rigid-mode control matrix  $B_r$  is nonsingular.

#### Example 5.2.1 (Open-Loop Behavior of Flexible-Link Robot Arm) :

The model for a one-link robot arm with pinned-pinned boundary conditions and two retained modes is given by (5.2.3) with

$$M = \begin{bmatrix} 2.2024 & 0.0517 & 0.0410 \\ 0.0517 & 0.0026 & 0.0036 \\ 0.0410 & 0.0036 & 0.0080 \end{bmatrix}, \quad V_m = \begin{bmatrix} 0.0200 & 0.0013 & 0.0027 \\ 0.0013 & 0.0001 & 0.0002 \\ 0.0027 & 0.0002 & 0.0004 \end{bmatrix}$$

$$K = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 14.0733 & 0 \\ 0 & 0 & 225.1734 \end{bmatrix}, \quad G = 0, \quad B = \begin{bmatrix} 1.0000 \\ 0.0668 \\ 0.1337 \end{bmatrix}.$$

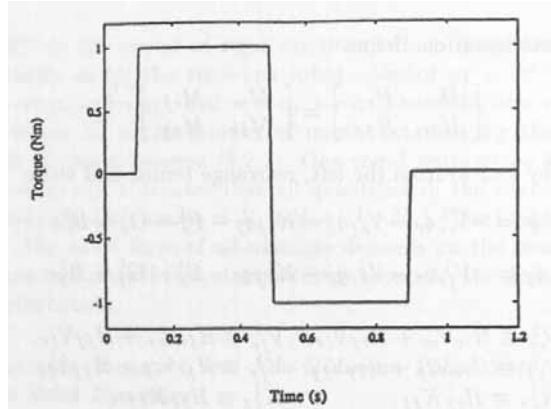


Figure 5.2.1: Acceleration/deceleration torque profile  $\tau(t)$ .

This single-link beam could model, for instance, a tank gun barrel with two vibratory modes. Note that the single-link robot is linear; multi-link flexible arms have nonlinear effects.

Using MATLAB, we find the open-loop poles to be

$$s = \begin{pmatrix} 0 \\ -0.01 \\ -0.0002 \pm 89.97i \\ -0.0000 \pm 342.01i \end{pmatrix}.$$

Note that there are two poles near  $s = 0$  corresponding to the Newton's law-type motion  $F = ma$  of the rigid mode, which is the joint angle  $q_r$ . There are two complex pole pairs, almost completely undamped, one corresponding to the first mode with frequency  $89.97 \text{ rad/sec} = 14.32 \text{ Hz}$  and one corresponding to the second mode with frequency  $342.01 \text{ rad/sec} = 54.43 \text{ Hz}$ .

We simulated the open-loop motion using MATLAB routine `ode23` on the state-space form derived from (5.2.9)-(5.2.10), where the state is  $x = [q_r \ q_{f_1} \ q_{f_2} \ \dot{q}_r \ \dot{q}_{f_1} \ \dot{q}_{f_2}]^T$ . The input  $\tau(t)$  was selected as the test acceleration/deceleration profile given in Fig. 5.2.1. The open-loop rigid mode position and velocity responses  $q_r(t), \dot{q}_r(t)$  are shown in Fig. 5.2.2. Note that in the absence of vibrations one should have a linear velocity profile and a quadratic position profile. The first and second modes  $q_{f_1}(t), q_{f_2}(t)$  are shown in Fig. 5.2.3 and show no detectable attenuation over time. The result, apparent in Fig. 5.2.2, is the significant residual oscillation in the joint velocity  $\dot{q}_r(t)$ .  $\square$

### 5.2.2 Robots with Actuators and Compliant Drive Train Coupling

*Robot manipulators are driven by actuators which may be electric, hydraulic, pneumatic, and so on. The actuators are coupled to the links through coupling mechanisms which may contain gears. Particularly in the case of high-speed performance requirements, the coupling shafts may exhibit appreciable compliance that cannot be disregarded. This effect is known as joint flexibility. Joint flexibility and link flexibility, discussed in the previous subsection, represent two cases of high-frequency*

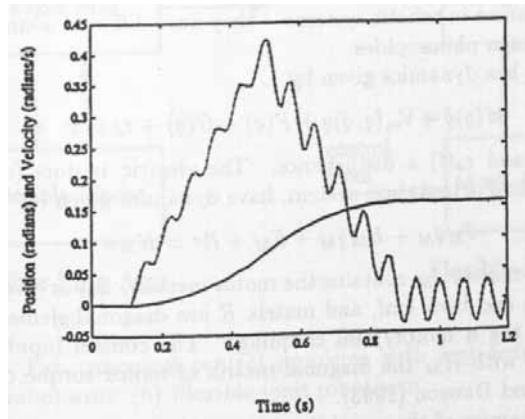


Figure 5.2.2: Open-loop response of flexible arm: tip position  $q_r(t)$  (solid) and velocity (dashed).

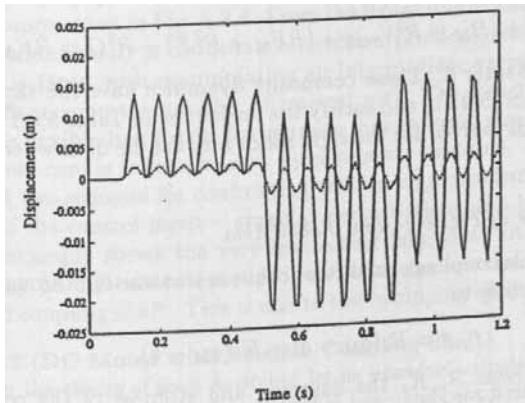


Figure 5.2.3: Open-loop response of flexible arm: flexible modes  $q_{f1}(t), q_{f2}(t)$ .

dynamical disturbances in robotic systems— they have different sources and require distinct control design philosophies.

The robot arm has dynamics given by

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = \tau \quad (5.2.11)$$

where  $q(t) \in \mathbb{R}^n$  and  $\tau_d(t)$  a disturbance. The electric motors (one per joint), exemplifying a typical actuation system, have dynamics given by

$$J_M\ddot{q}_M + B_M\dot{q}_M + F_M + R\tau = K_M v \quad (5.2.12)$$

$q_M(t) \in \mathbb{R}^n$ , where matrix  $J_M$  contains the motor inertias,  $B_M$  is given by the rotor damping constants and back emf, and matrix  $R$  has diagonal elements containing the gear ratios of the  $n$  motor/joint couplings. The control input is the motor voltage  $v(t) \in \mathbb{R}^n$ , with  $K_M$  the diagonal matrix of motor torque constants. See Lewis, Abdallah, and Dawson (1993).

The overall dynamics of the coupled robot arm/actuator depends on the compliance of the coupling shaft and gearing. Two cases will be considered— rigid coupling shaft and flexible coupling shaft.

### 5.2.2.1 Rigid Actuator/Arm Coupling

If the coupling has no flexibility effects, then one may simply use the equation for a rigid gear train with no slippage, where  $q = Rq_M$  with  $R = \text{diag}\{r_i\}$ , and  $\|r_i\| \leq 1$  the gear ratio of link  $i$ . The torque scales conversely, as  $\tau = \tau_M/R$ , which effect is already included in (5.2.12).

Using these relations it is easy to show that the dynamics of the arm plus actuators can be written as

$$(J_M + R^2 M)\ddot{q} + (B_M + R^2 V_m)\dot{q} + (RF_M + R^2 F) + R^2 G = RK_M v, \quad (5.2.13)$$

as described in Chapter 3. These composite dynamics have the same form as the robot arm dynamics (5.2.11) and satisfy the properties in Table 5.0.1. Therefore, all the control methods derived for the rigid robot arm can be used when the actuators are included.

### 5.2.2.2 Dynamics of Flexible-Joint Robot Arm

In the case of flexible coupling shafts, the torque is transmitted through the  $n$  shafts, one per link, according to

$$\tau = B_s(\dot{q}_M - \dot{q}) + K_s(q_M - q), \quad (5.2.14)$$

with diagonal matrices  $B_s, K_s$  the damping and stiffness of the coupling shafts. Using this in (5.2.11)-(5.2.12) yields the composite flexible joint robot arm dynamics

$$\begin{aligned} M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d + B_s(\dot{q} - \dot{q}_M) &+ K_s(q - q_M) \\ &= 0 \end{aligned} \quad (5.2.15)$$

$$\begin{aligned} J_M\ddot{q}_M + B_M\dot{q}_M + F_M + RB_s(\dot{q}_M - \dot{q}) &+ RK_s(q_M - q) \\ &= K_M v \end{aligned} \quad (5.2.16)$$

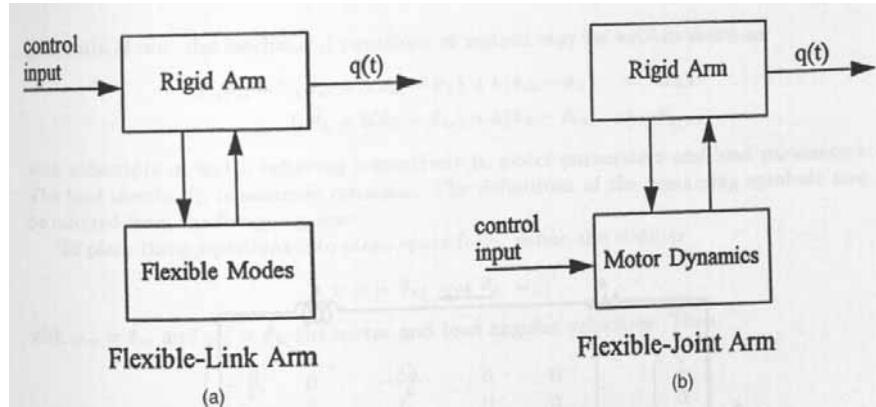


Figure 5.2.4: Two canonical control problems with high-frequency modes. (a) Flexible-link robot arm. (b) Flexible-joint robot arm.

One sees that coupling flexibility has resulted in a higher-order dynamics. This has the form of (5.2.3), with  $B(q) = [0 \ K_M^T]^T$  having more rows than columns, so that the control effectiveness is reduced and modified control techniques must be invoked, as described in the next two sections.

The control objective for the flexible-joint robot arm is to control the arm joint variable  $q(t)$  in the face of the additional dynamics of the variable  $q_M(t)$ , the motor angle. This is similar to the situation for the flexible-link arm in Section 5.2.1, where one must control the arm variable in the face of the additional dynamics of  $q_f(t)$ , the link flexible modes. However, controls design for these two problems must be approached by different philosophies. The flexible-link and flexible-joint dynamics are represented in Fig. 5.2.4. From the figure it is evident that the flexible-link arm is fundamentally a disturbance rejection problem, while for the flexible-joint arm one is faced with manipulating an intermediate variable  $q_M(t)$  that has subsequent influence on the variable of interest  $q(t)$ . The structure of the systems means that the flexible-link robot, for instance, has internal dynamics (see Chapter 2); in fact unless care is taken the zero dynamics are unstable. In the next section are introduced two schemes for confronting these design problems by extending the effectiveness of the control input—singular perturbations and backstepping.

The next example shows the very interesting fact that, although the flexible-joint robot may be difficult to control, it can have faster response than the robot arm with rigid coupling shaft. This is due to the ‘whipping effect’ of a flexible shaft.

#### Example 5.2.2 (DC Motor with Flexible Coupling Shaft) :

To focus on the effects of joint flexibility let us examine a single armature-controlled DC motor coupled to a load through a shaft that has significant flexibility. The electrical and mechanical subsystems are shown in Fig. 5.2.5. The motor electrical equation is

$$\dot{L_i} = -Ri - k'_m \dot{\theta}_m + u$$

with  $i(t)$ ,  $u(t)$  the armature current and voltage respectively. The back emf is  $v_b = k'_m \dot{\theta}_m$ .

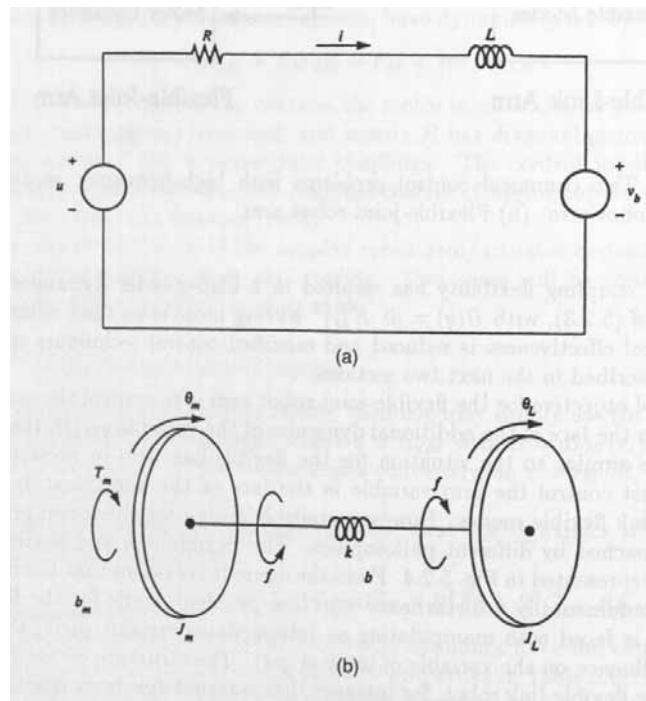


Figure 5.2.5: DC motor with shaft compliance. (a) Electrical subsystem. (b) Mechanical subsystem.

The interaction force exerted by the flexible shaft is given by  $f = b(\dot{\theta}_m - \dot{\theta}_L) + k(\theta_m - \theta_L)$ , where the shaft damping and spring constants are denoted by  $b$  and  $k$ . Thus, with a gear ratio of one, the mechanical equations of motion may be written down as

$$\begin{aligned} J_m \ddot{\theta}_m + b_m \dot{\theta}_m + b(\dot{\theta}_m - \dot{\theta}_L) + k(\theta_m - \theta_L) &= k_m i \\ J_L \ddot{\theta}_L + b(\dot{\theta}_L - \dot{\theta}_m) + k(\theta_L - \theta_m) &= 0, \end{aligned}$$

with subscripts  $m$  and  $L$  referring respectively to motor parameters and load parameters. The load inertia  $J_L$  is assumed constant. The definitions of the remaining symbols may be inferred from the foregoing text.

To place these equations into state-space form, define the state as

$$x = [i \ \theta_m \ \omega_m \ \theta_L \ \omega_L]^T,$$

with  $\omega_m = \dot{\theta}_m$  and  $\omega_L = \dot{\theta}_L$  the motor and load angular velocities. Then,

$$\dot{x} = \begin{bmatrix} -\frac{R}{L} & 0 & -\frac{k'_m}{L} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \frac{k_m}{J_m} & -\frac{k}{J_m} & -\frac{(b+b_m)}{J_m} & \frac{k}{J_m} & \frac{b}{J_m} \\ 0 & 0 & 0 & 0 & 1 \\ 0 & \frac{k}{J_L} & \frac{b}{J_L} & -\frac{k}{J_L} & -\frac{b}{J_L} \end{bmatrix} x + \begin{bmatrix} \frac{1}{L} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} u.$$

**a. Rigid Coupling Shaft.** If there is no compliance in the coupling shaft,  $\omega_m = \omega_L \equiv \omega$  and the state equations reduce to (see Problems section)

$$\dot{x} = \begin{bmatrix} -\frac{R}{L} & -\frac{k'_m}{L} \\ \frac{k_m}{J} & -\frac{b_m}{J} \end{bmatrix} x + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} u \equiv Ax + Bu,$$

where  $x = [i \ \omega]^T$ ,  $J = J_m + J_L$ . Defining the output as the motor speed gives

$$y = [0 \ 1]xCx.$$

The transfer function is computed to be

$$H(s) = C(sI - A)^{-1}B = \frac{k_m}{(Ls + R)(Js + b_m) + k_m k'_m}.$$

Using parameter values of  $J_m = J_L = 0.1 \text{ kg-m}^2$ ,  $k_m = k'_m = 1 \text{ V-s}$ ,  $L = 0.5 \text{ H}$ ,  $b_m = 0.2 \text{ N-m/rad/s}$ , and  $R = 5 \Omega$  yields

$$H(s) = \frac{10}{(s + 2.3)(s + 8.7)}$$

so that there are two real poles at  $s = -2.3$ ,  $s = -8.7$ . Using MATLAB to perform a simulation yields the step response for  $\omega(t)$  shown in Fig. 5.2.6.

**b. Very Flexible Coupling Shaft.** Coupling shaft parameters of  $k = 2 \text{ N-m/rad}$  and  $b = 0.2 \text{ N-m/rad/s}$  correspond to a very flexible shaft. Using these values, MATLAB can be employed to obtain the two transfer functions

$$\begin{aligned} \frac{\omega_m}{u} &= \frac{20s[(s+1)^2 + 4.36^2]}{s(s+3.05)(s+6.14)[(s+3.4)^2 + 5.6^2]} \\ \frac{\omega_L}{u} &= \frac{40s(s+10)}{s(s+3.05)(s+6.14)[(s+3.4)^2 + 5.6^2]}. \end{aligned}$$

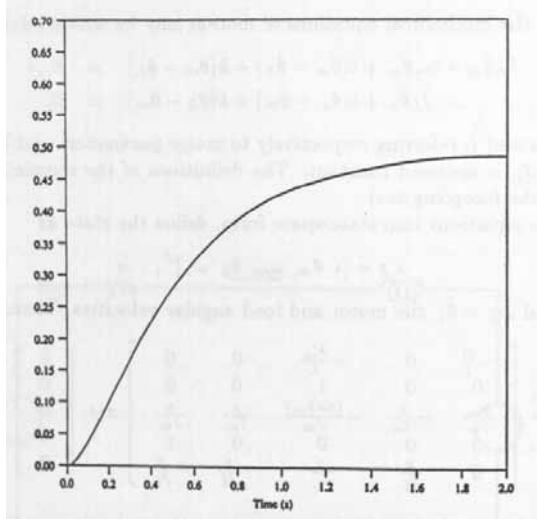


Figure 5.2.6: Step response of DC motor with no shaft flexibility. Motor speed in rad/s.

The shaft flexible mode has the poles  $s = -3.4 \pm j5.6$ , and so has a damping ratio of  $\zeta = 0.52$  and a natural frequency of  $\omega_N = 6.55 \text{ rad/s}$ . Note that the system is marginally stable, with a pole at  $s = 0$ . It is BIBO stable due to pole/zero cancellation.

MATLAB yielded the step response shown in Fig. 5.2.7. Several points are worthy of note. Initially the motor speed  $\omega_m$  rises more quickly than in Fig. 5.2.6 since the shaft flexibility means that only the rotor moment of inertia  $J_m$  initially affects the speed. Then, as the load  $J_L$  is coupled back to the motor through the shaft, the rate of increase of  $\omega_m$  slows. Note also that the load speed  $\omega_L$  exhibits a *delay* of approximately 0.1 sec due to the flexibility in the shaft.

It is extremely interesting to note that the shaft flexibility has the effect of speeding up the slowest real pole (compare the poles), so that  $\omega_L$  approaches its steady-state value more quickly than in the rigid shaft case. This is due to the ‘whipping’ action of the flexible shaft. The shaft dynamics make the control of  $\theta_L$ , which corresponds in a robot arm to the joint angle  $q$ , very difficult without some sort of specially designed controller.  $\square$

### 5.2.3 Rigid-Link Electrically-Driven (RLED) Robot Arms

*Electrical motors have both electrical and mechanical dynamics. If the motor electrical time constants are fast enough, it is adequate to ignore the electrical dynamics and use equations such as those in Subsection 5.2.2. If the electrical dynamics are not negligible, then one must use include them. The dynamics of an n-link rigid robot arm with no actuator shaft compliance and non-negligible motor electrical dynamics are given by*

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = K_T i \quad (5.2.17)$$

$$Li + R(i, \dot{q}) + \tau_e = u_e \quad (5.2.18)$$

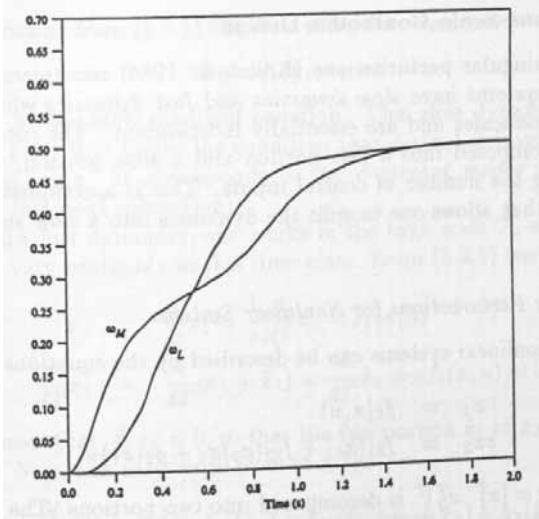


Figure 5.2.7: Step response of DC motor with very flexible shaft.

with  $q(t) \in \mathbb{R}^n$  the joint variable,  $i(t) \in \mathbb{R}^n$  the motor armature currents,  $K_T$  a diagonal electro-mechanical conversion matrix,  $L$  a matrix of electrical inductances,  $R(i, \dot{q})$  representing both electrical resistance and back emf,  $\tau_d(t)$  and  $\tau_e(t)$  the mechanical and electrical disturbances, and motor terminal voltage vector  $u_e(t) \in \mathbb{R}^n$  the control input.

The rigid-link electrically-driven (RLED) equations represent dynamics similar to those of the flexible-joint equations (5.2.15)-(5.2.16), as depicted in Fig. 5.2.4b. The electrical time constants produce high-frequency dynamics that must be confronted in controller design. The approach used in Chapter 4 cannot be used to design neural network controllers, but the techniques in the next section may be used to extend the control effectiveness and solve this problem.

If one desires to include both the motor electrical and mechanical dynamics and there is no shaft compliance, then (5.2.13) should be used instead of (5.2.17). If the robot arm has both joint flexibility and non-negligible actuator electrical dynamics, then the electrical equation (5.2.18) must be appended to (5.2.15)-(5.2.16), as in Example 5.2.2.

### 5.3 SINGULAR PERTURBATION DESIGN

We have seen that some practical robotic systems cannot be controlled using the rigid robot arm techniques described in Chapter 4, including robots with link vibration, joint compliance, and actuators having fast dynamics (e.g. electrical dynamics). Such systems have high-frequency dynamics, resulting in more degrees of freedom than control inputs. Two techniques are now introduced to deal with this problem by extending the control effectiveness of the system. They are singular perturbation design and backstepping design. The former is covered in this section, the latter in

the next section.

### 5.3.1 Two-Time-Scale Controller Design

The method of singular perturbations (Kokotovic 1984) recognizes the fact that a large class of systems have slow dynamics and fast dynamics which operate on very different time-scales and are essentially independent. The control input can therefore be decomposed into a fast portion and a slow portion, which has the effect of doubling the number of control inputs. This is accomplished by a time-scale separation that allows one to split the dynamics into a slow subsystem and a fast subsystem.

#### 5.3.1.1 Singular Perturbations for Nonlinear Systems

A large class of nonlinear systems can be described by the equations

$$\dot{x}_1 = f_1(x, u) \quad (5.3.1)$$

$$\varepsilon \dot{x}_2 = f_{21}(x_1) + f_{22}(x_1)x_2 + g_2(x_1)u \quad (5.3.2)$$

where the state  $x = [x_1^T \ x_2^T]^T$  is decomposed into two portions. The left-hand side of the second equation is premultiplied by  $\varepsilon \ll 1$ , indicating that the dynamics of  $x_2$  are much faster than those of  $x_1$ . In fact, the variable  $x_1$  develops on the standard time-scale  $t$ , while  $\frac{d}{dT}x_2 = \varepsilon \dot{x}_2$  with

$$T = \frac{t}{\varepsilon}, \quad (5.3.3)$$

so that the natural time-scale for  $x_2$  is a faster one defined by  $T$ .

The control variable  $u(t)$  is required to manipulate both  $x_1$  and  $x_2$ . Using the technique of singular perturbations, its effectiveness can be increased under certain conditions as follows. This allows simplified controller design.

**Slow/Fast Subsystem Decomposition.** The system may be decomposed into a slow subsystem and a fast subsystem to increase the control effectiveness. To accomplish this, define all variables to consist of a slow part, denoted by overbar, and a fast part, denoted by tilde. Thus,

$$x_1 = \bar{x}_1 + \tilde{x}_1, \quad x_2 = \bar{x}_2 + \tilde{x}_2, \quad u = \bar{u} + \tilde{u}. \quad (5.3.4)$$

To derive the slow dynamics, set  $\varepsilon = 0$  and replace all variables by their slow portions. From (5.3.2) one obtains

$$0 = f_{21}(\bar{x}_1) + f_{22}(\bar{x}_1)\bar{x}_2 + g_2(\bar{x}_1)\bar{u},$$

which may be solved to obtain

$$\bar{x}_2 = -f_{22}^{-1}(\bar{x}_1)[f_{21}(\bar{x}_1) + g_2(\bar{x}_1)\bar{u}], \quad (5.3.5)$$

under the condition that  $f_{22}(x_1)$  is invertible. This is known as the slow manifold equation, and reveals that the slow portion of  $x_2$  is dependent completely on the slow portion of  $x_1$  and the slow portion of the control input.

Now one obtains from (5.3.1) the slow subsystem equation

$$\dot{\bar{x}}_1 = f_1(\bar{x}_1, \bar{x}_2, u) \quad (5.3.6)$$

with  $\bar{x}_2$  given by the slow manifold equation. This slow dynamics is completely dependent on  $\bar{x}_1(t)$ ,  $\bar{u}(t)$  under the condition that  $f_{22}(x_1)$  is invertible. Therefore, it is of reduced order. It corresponds to the dominant modes in classical linear system theory (see next subsection).

To study the fast dynamics, one works in the time scale  $T$ , assuming that the slow variables vary negligibly in this time scale. From (5.3.1) one has

$$\begin{aligned} \dot{x}_1 &= \frac{d}{dt}x_1 = \frac{1}{\varepsilon}\frac{d}{dT}x_1 = f_1(x, u) \\ \frac{d}{dT}x_1 &= \frac{d}{dT}(\bar{x}_1 + \tilde{x}_1) = \frac{d}{dT}\tilde{x}_1 = \varepsilon f_1(x, u) \approx 0, \end{aligned}$$

since one assumes that  $\frac{d}{dT}\bar{x}_1 = 0$ , so that the fast portion  $\tilde{x}_1$  of  $x_1$  is approximately equal to zero. Now, one may write from (5.3.2)

$$\varepsilon \frac{d}{dt}x_2 = \frac{d}{dT}(\bar{x}_2 + \tilde{x}_2) = f_{21}(\bar{x}_1) + f_{22}(\bar{x}_1)(\bar{x}_2 + \tilde{x}_2) + g_2(\bar{x}_1)(\bar{u} + \tilde{u}),$$

whence substitution from (5.3.5) and the assumption that  $\frac{d}{dT}\bar{x}_2 = 0$  yields

$$\frac{d}{dT}\tilde{x}_2 = f_{22}(\bar{x}_1)\tilde{x}_2 + g_2(\bar{x}_1)\tilde{u}. \quad (5.3.7)$$

This is a fast subsystem, which is linear since  $\bar{x}_1$  is constant in the  $T$  time scale.

**Composite Controls Design and Tikhonov's Theorem.** The singular perturbation slow/fast decomposition suggests the following controls design technique. Design a slow control  $\bar{u}$  for the slow subsystem (5.3.6) using any method. Design a fast control  $\tilde{u}$  for the fast subsystem (5.3.7) using any method. Then, apply the composite control

$$u = \bar{u} + \tilde{u}. \quad (5.3.8)$$

Using this technique, one performs two control designs, and effectively obtains two independent control input components that are simply added to produce  $u(t)$ . This independent design of two control inputs practically increases the control effectiveness.

The composite control approach works due to an extension of Tikhonov's Theorem, which also relates the slow/fast decomposition (5.3.6)-(5.3.7) to the original system description (5.3.1)-(5.3.2). It states that if  $f_{22}(x_1)$  is invertible and the linear system (5.3.7) is stabilizable considering  $\bar{x}_1$  as slowly time-varying (i.e. practically constant), then one has

$$x_1 = \bar{x}_1 + O(\varepsilon) \quad (5.3.9)$$

$$x_2 = \bar{x}_2 + \tilde{x}_2 + O(\varepsilon), \quad (5.3.10)$$

where  $O(\varepsilon)$  denotes terms of order  $\varepsilon$ .

From these notions one obtains the concept of  $\tilde{x}_2$  as a boundary-layer correction term due to the fast dynamics, and of  $\tilde{u}$  as a boundary-layer correction control term that manages the high-frequency (fast) motion.

### 5.3.1.2 Singular Perturbations for Linear Systems

In the linear case, the singular perturbations technique simplifies so that one can obtain more insight. Therefore, consider the linear system

$$\dot{x}_1 = A_{11}x_1 + A_{12}x_2 + B_1u \quad (5.3.11)$$

$$\varepsilon\dot{x}_2 = A_{21}x_1 + A_{22}x_2 + B_2u \quad (5.3.12)$$

$$y = C_1x_1 + C_2x_2 \quad (5.3.13)$$

with  $A_{22}$  invertible. Define slow and fast portions of the variables as in (5.3.4) and

$$y = \bar{y} + \tilde{y}. \quad (5.3.14)$$

The slow dynamics are obtained using  $\varepsilon = 0$  and slow variables, so that from (5.3.12) one obtains the slow manifold equation

$$\begin{aligned} 0 &= \varepsilon\dot{x}_2 = A_{21}\bar{x}_1 + A_{22}\bar{x}_2 + B_2\bar{u} \\ \bar{x}_2 &= -A_{22}^{-1}[A_{21}\bar{x}_1 + B_2\bar{u}], \end{aligned} \quad (5.3.15)$$

whence substitution into (5.3.11) yields the slow subsystem

$$\dot{\bar{x}}_1 = (A_{11} - A_{12}A_{22}^{-1}A_{21})\bar{x}_1 + (B_1 - A_{12}A_{22}^{-1}B_2)\bar{u}. \quad (5.3.16)$$

The slow subsystem output equation is found by substituting (5.3.15) into (5.3.13) to be

$$\bar{y} = (C_1 - C_2A_{22}^{-1}A_{21})\bar{x}_1 - C_2A_{22}^{-1}B_2\bar{u}. \quad (5.3.17)$$

Note that there is now a direct feed term from the input to the output.

The fast dynamics are obtained by working in the fast time scale  $T = \frac{t}{\varepsilon}$ , starting from (5.3.11) to write

$$\frac{d}{dT}x_1 = \varepsilon\dot{x}_1 = \varepsilon(A_{11}x_1 + A_{12}x_2 + B_1u) \approx 0,$$

so that  $\tilde{x}_1 \approx 0$ . Thus, one may write from (5.3.12) that

$$\varepsilon\frac{d}{dt}x_2 = \frac{d}{dT}(\bar{x}_2 + \tilde{x}_2) = A_{21}\bar{x}_1 + A_{22}(\bar{x}_2 + \tilde{x}_2) + B_2(\bar{u} + \tilde{u}),$$

so that substitution from (5.3.15) and the assumption that  $\frac{d}{dT}\bar{x}_2 = 0$  yields

$$\frac{d}{dT}\tilde{x}_2 = A_{22}\tilde{x}_2 + B_2\tilde{u}. \quad (5.3.18)$$

The fast subsystem output is given as

$$\tilde{y} = C_2\tilde{x}_2. \quad (5.3.19)$$

The slow subsystem (5.3.16) has poles corresponding to the dominant modes in classical controls analysis, namely, the modes when the fast dynamics are neglected.

To design a controller and apply Tikhonov's theorem to guarantee the closed-loop performance, it is necessary that the pair  $(A_{22}, B_2)$  be stabilizable.

### 5.3.2 NN Controller for Flexible-Link Robot Using Singular Perturbations

*Singular perturbations can be used to design controllers for flexible-link arms (Siciliano and Book 1988), flexible-joint arms (Spong 1989), and electrically-driven arms where the electrical time constants are faster than the mechanical ones. In this subsection we show how to use the singular perturbations approach to design a neural network (NN) controller for the flexible-link robot arm discussed in Section 5.2.1 (Yeşildirek et al. 1996), thereby extending the results of Chapter 4 to systems with vibratory high-frequency dynamics in the links.*

*Exploiting the natural time-scale separation between the faster flexible mode dynamics and the slower rigid mode dynamics, we use singular perturbation theory to formulate a boundary-layer correction that stabilizes the non-minimum phase internal dynamics of the flexible-link arm. First, singular perturbation theory is applied to decompose the flexible robot into a slow subsystem and a fast subsystem; this is a streamlined version of the development in Siciliano and Book (1988). Next, a modified practical tracking problem is defined. Finally, the NN tracking controller is presented.*

#### 5.3.2.1 Time-Scale Decomposition of Flexible-Link Robot Arm

We start from the flexible-link dynamics in the form (5.2.9)-(5.2.10). Introduce a small scale factor  $\varepsilon$  and define  $\xi$  and  $\tilde{K}_{ff}$  by

$$\varepsilon^2 \xi = q_f, \quad \tilde{K}_{ff} \equiv \varepsilon^2 K_{ff} \quad (5.3.20)$$

where  $1/\varepsilon^2$  is equal to the smallest stiffness in  $K_{ff}$ . In most practical vibratory systems the parameter  $\varepsilon$  is small, that is, the link stiffness is large. Substitution of (5.3.20) into (5.2.9) and (5.2.10) gives the system in the form,

$$\begin{aligned} \ddot{q}_r &= -V_{rr}^1(q_r, \dot{q}_r, \varepsilon^2 \xi, \varepsilon^2 \dot{\xi}) \dot{q}_r - V_{rf}^1(q_r, \dot{q}_r, \varepsilon^2 \xi, \varepsilon^2 \dot{\xi}) \varepsilon^2 \dot{\xi} \\ &\quad - (1/\varepsilon^2) H_{rf} \tilde{K}_{ff}(q_r, \varepsilon^2 \xi) \varepsilon^2 \xi - F_r^1(q_r, \varepsilon^2 \xi) \\ &\quad - G_r^1(q_r, \varepsilon^2 \xi) + B_r^1(q_r, \varepsilon^2 \xi) \tau \end{aligned} \quad (5.3.21)$$

$$\begin{aligned} \varepsilon^2 \ddot{\xi} &= -V_{fr}^1(q_r, \dot{q}_r, \varepsilon^2 \xi, \varepsilon^2 \dot{\xi}) \dot{q}_r - V_{ff}^1(q_r, \dot{q}_r, \varepsilon^2 \xi, \varepsilon^2 \dot{\xi}) \varepsilon^2 \dot{\xi} \\ &\quad - (1/\varepsilon^2) H_{ff} \tilde{K}_{ff}(q_r, \varepsilon^2 \xi) \varepsilon^2 \xi - F_f^1(q_r, \varepsilon^2 \xi) \\ &\quad - G_f^1(q_r, \varepsilon^2 \xi) + B_f^1(q_r, \varepsilon^2 \xi) \tau \end{aligned} \quad (5.3.22)$$

where  $K_{ff}$  is invertible because it is diagonal.

Define now the control

$$\tau = \bar{\tau} + \tau_F \quad (5.3.23)$$

with  $\bar{\tau}$  the slow control component and  $\tau_F$  a fast component.

Setting  $\varepsilon = 0$  yields the slow equation

$$\ddot{\bar{q}}_r = -\bar{V}_{rr}^1 \dot{\bar{q}}_r - \bar{H}_{rf} \tilde{K}_{ff} \bar{\xi} - \bar{F}_r^1 - \bar{G}_r^1 + \bar{B}_r^1 \bar{\tau} \quad (5.3.24)$$

and an algebraic relation in  $\dot{\bar{q}}_r$ ,  $\bar{\xi}$ , and,  $\bar{\tau}$

$$0 = -\bar{V}_{fr}^1 \dot{\bar{q}}_r - \bar{H}_{ff} \tilde{K}_{ff} \bar{\xi} - \bar{F}_f^1 - \bar{G}_f^1 + \bar{B}_f^1 \bar{\tau} \quad (5.3.25)$$

Note that we use an overbar to denote evaluation of nonlinear functions with  $\varepsilon = 0$ . The slow manifold is found by solving for  $\bar{\xi}$  using (5.3.25), to be

$$\bar{\xi} = \tilde{K}_{ff}^{-1} \bar{H}_{ff}^{-1} (-\bar{V}_{fr}^1 \dot{\bar{q}}_r - \bar{F}_f^1 - \bar{G}_f^1 + \bar{B}_f^1 \bar{\tau}). \quad (5.3.26)$$

One formally acquires the slow subsystem, which is the rigid model, by substituting (5.3.26) into (5.3.24) to obtain (see Problems section)

$$\begin{aligned} \ddot{\bar{q}}_r &= (\bar{H}_{rr} - \bar{H}_{rf} \bar{H}_{ff}^{-1} \bar{H}_{fr}) [-\bar{V}_{rr} \dot{\bar{q}}_r - \bar{F}_r - \bar{G}_r + \bar{B}_r \bar{\tau}] \\ &= \bar{M}_{rr}^{-1} [-\bar{V}_{rr} \dot{\bar{q}}_r - \bar{F}_r - \bar{G}_r + \bar{B}_r \bar{\tau}]. \end{aligned} \quad (5.3.27)$$

To derive the fast subsystem, we now select states  $\varsigma_1 \equiv \xi - \bar{\xi}$ ,  $\varsigma_2 \equiv \varepsilon \dot{\xi}$  and write (5.3.22) as

$$\begin{aligned} \varepsilon \dot{\varsigma}_1 &= \varsigma_2 \\ \varepsilon \dot{\varsigma}_2 &= -\bar{V}_{fr}^1 \dot{\bar{q}}_r - V_{ff}^1 \varepsilon \varsigma_2 - \bar{H}_{ff} \tilde{K}_{ff} (\varsigma_1 + \bar{\xi}) - \bar{F}_f^1 - \bar{G}_f^1 + \bar{B}_f^1 \tau, \end{aligned} \quad (5.3.28)$$

whence a time-scale change of  $T = t/\varepsilon$  results in

$$\begin{aligned} \frac{d\varsigma_1}{dT} &= \varsigma_2 \\ \frac{d\varsigma_2}{dT} &= -\bar{V}_{fr}^1 \dot{\bar{q}}_r - V_{ff}^1 \varepsilon \varsigma_2 - \bar{H}_{ff} \tilde{K}_{ff} (\varsigma_1 + \bar{\xi}) - \bar{F}_f^1 - \bar{G}_f^1 + \bar{B}_f^1 (\bar{\tau} + \tau_F) \end{aligned}$$

since  $d\bar{\xi}/dT \approx 0$ . Setting now  $\varepsilon = 0$  and substituting for  $\bar{\xi}$  from (5.3.26) gives the fast dynamics

$$\begin{aligned} \frac{d\varsigma_1}{dT} &= \varsigma_2 \\ \frac{d\varsigma_2}{dT} &= -\bar{H}_{ff} (\bar{q}_r, 0) \tilde{K}_{ff} \varsigma_1 + \bar{B}_f^1 \tau_F. \end{aligned} \quad (5.3.29)$$

It is important to note that this is a linear system, with the slow variables as parameters, that describes the vibratory modes. The stiffness matrix has the form (5.2.7).

In summary, we have obtained the description of the full-order model given by the slow subsystem

$$\bar{M}_{rr} \ddot{\bar{q}}_r + \bar{V}_{rr} \dot{\bar{q}}_r + \bar{F}_r + \bar{G}_r = \bar{B}_r \bar{\tau} \quad (5.3.30)$$

and the fast subsystem

$$\frac{d}{dT} \begin{bmatrix} \varsigma_1 \\ \varsigma_2 \end{bmatrix} = \begin{bmatrix} 0 & I \\ -\bar{H}_{ff} \tilde{K}_{ff} & 0 \end{bmatrix} \begin{bmatrix} \varsigma_1 \\ \varsigma_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \bar{B}_f^1 \end{bmatrix} \tau_F \quad (5.3.31)$$

which we write as

$$\frac{d\varsigma}{dT} = A_F \varsigma + B_F \tau_F. \quad (5.3.32)$$

It is important to realize at this point that the slow subsystem, describing the rigid motion of the robot arm, is in the standard form of robot dynamics, and satisfies the properties in Table 5.0.1.

### 5.3.2.2 A Modified Tracking Problem

According to a theorem of Tikhonov, the net result is that the flexible-link robot (5.2.9)-(5.2.10) can be described to order  $\varepsilon$  using (5.3.30)-(5.3.32). In fact, as long as we apply the control

$$\tau = \bar{\tau} + \tau_F, \quad (5.3.33)$$

the trajectories in the original system are given by

$$q_r = \bar{q}_r + O(\varepsilon), \quad (5.3.34)$$

$$q_f = \varepsilon^2(\bar{\xi} + \varsigma_1) + O(\varepsilon), \quad (5.3.35)$$

$$(5.3.36)$$

with  $\bar{\xi}(t)$  given by (5.3.26) and  $O(\varepsilon)$  denoting terms of order  $\varepsilon$ . The fast control  $\tau_F$  is a boundary-layer correction that suppresses the vibrations.

We now formulate a tracking problem to allow selection of the slow control  $\bar{\tau}$  and the fast, vibration suppression, control  $\tau_F$ . The main objective of our control design is to cause the rigid variable  $q_r(t)$  to track a desired trajectory  $q_d(t)$ . In view of this objective, one could select the output of the system as  $y = [q_r^T \dot{q}_r^T]^T$ . Unfortunately, this is not a suitable choice, as it has been shown that it leads to unstable zero dynamics (Madhavan and Singh 1991, Wang and Vidyasagar 1991, Vandegrift et al. 1994). (Internal dynamics were defined in Chapter 2 in the discussion on feedback linearization.)

The non-minimum phase internal dynamics problems of vibratory systems are not difficult to solve. The key point is to relax the tracking requirement so that the slow part,  $\bar{q}_r(t)$  and  $\dot{\bar{q}}_r(t)$ , of the link positions and velocities track  $q_d(t)$  and  $\dot{q}_d(t)$ . Define, therefore, the output as

$$y = \begin{bmatrix} \bar{q}_r \\ \dot{\bar{q}}_r \end{bmatrix}. \quad (5.3.37)$$

This modified tracking output corresponds to a practically useful performance objective, allowing the actual link-tip motions  $q_r(t), \dot{q}_r(t)$  to track the desired trajectories within order  $\varepsilon$ .

The reason this tracking output is a good one is that the internal dynamics relative to  $y(t)$  are given by the fast system (5.3.32). Thus,  $\tau_F(t)$  can be chosen to give stable internal dynamics if the pair  $(A_F, B_F)$  is stabilizable. This is generally the case for flexible-link arms. Under this assumption there are many approaches in robust linear systems theory that yield a stabilizing control  $\tau_F(t)$ , including the linear quadratic regulator,  $H_\infty$  design, and techniques that avoid the measurement of the strain rates  $\dot{q}_f(t)$  (Zhu et al. 1994). The fast control is of the form

$$\tau_F = -[K_{pF} \ K_{dF}] \begin{bmatrix} \varsigma_1 \\ \varsigma_2 \end{bmatrix} = -\frac{K_{pF}}{\varepsilon^2} q_f - \frac{K_{dF}}{\varepsilon} \dot{q}_f + K_{pF} \bar{\xi} \quad (5.3.38)$$

with  $\bar{\xi}$  given by (5.3.26). Note that this requires measurements or computation of the slow manifold variable  $\bar{\xi}$ .

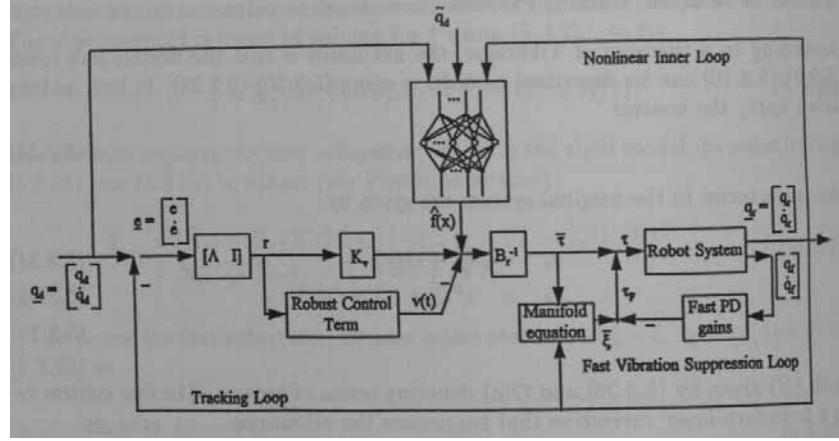


Figure 5.3.1: Neural net controller for flexible-link robot arm.

### 5.3.2.3 Neural Network Controller

We have just seen how to select a fast control input  $\tau_F$  to stabilize the vibratory modes. In this subsection it is shown that a NN controller like those in Chapter 4 can be used to design the slow control component  $\bar{\tau}(t)$  to achieve tracking of the desired motion  $q_d(t), \dot{q}_d(t)$  by the output  $y(t)$  in (5.3.37). The entire composite controller is shown in Fig. 5.3.1. This figure is like the rigid robot NN controllers in Chapter 4, but it has an additional inner fast control loop to suppress vibrations. Compare it to the force controller in Fig. 5.1.2.

**Tracking and Error System Dynamics.** The following developments parallel those in Chapter 4. The slow dynamics (5.3.30) is

$$\bar{M}_{rr}\ddot{\bar{q}}_r + \bar{V}_{rr}\dot{\bar{q}}_r + \bar{F}_r + \bar{G}_r = \bar{B}_r\bar{\tau} \quad (5.3.39)$$

which is exactly the Lagrange form of the dynamics of an  $n$ -link rigid robot arm with  $\bar{q}_r(t) \in \mathbb{R}^{n_r}$  the slow joint variable vector,  $\bar{M}_{rr}(\bar{q}_r)$  the inertia matrix,  $\bar{V}_{rr}(\bar{q}_r, \dot{\bar{q}}_r)$  the Coriolis/centripetal matrix,  $\bar{G}_r(\bar{q}_r)$  the gravity vector, and  $\bar{F}_r(\bar{q}_r)$  the friction. The control input torque is  $\bar{\tau}(t)$ . These dynamics satisfy the standard robot properties in Table 5.0.1, including boundedness of  $\bar{M}_{rr}$  and skew symmetry (i.e.  $\dot{\bar{M}}_{rr} - 2\bar{V}_{rr}$  is skew symmetric).

A neural net (NN) controller was designed for rigid robot arms in Chapter 4. This puts us in position to design by the same technique a NN tracking controller for the slow dynamics of the flexible arm without knowledge of friction, gravity, or Coriolis/centripetal terms. Thus, given a desired trajectory  $q_d(t) \in \mathbb{R}^{n_r}$  for the slow part of the flexible-link dynamics, define the tracking error

$$e(t) = q_d - \bar{q}_r \quad (5.3.40)$$

and the filtered tracking error

$$r = \dot{e} + \Lambda e \quad (5.3.41)$$

where  $\Lambda = \Lambda^T > 0$  is a design parameter matrix, usually selected diagonal. Differentiating  $r(t)$  and using (5.3.39), the arm dynamics may be written in terms of the filtered tracking error as

$$\bar{M}_{rr}\dot{r} = -\bar{V}_{rr}r - \bar{B}_r\bar{\tau} + f, \quad (5.3.42)$$

where the unknown nonlinear robot function is

$$f(X) = \bar{M}_{rr}(\bar{q}_r)(\ddot{q}_d + \Lambda\dot{e}) + \bar{V}_{rr}(\bar{q}_r, \dot{\bar{q}}_r)(\dot{q}_d + \Lambda e) + \bar{F}_r(\dot{\bar{q}}_r) + \bar{G}_r(\bar{q}_r) \quad (5.3.43)$$

and, for instance, one may select

$$X = \begin{bmatrix} e \\ \dot{e} \\ q_d \\ \dot{q}_d \\ \ddot{q}_d \end{bmatrix}.$$

**Stable NN Controller.** As in Chapter 4, define a control input torque for the slow subsystem as

$$\bar{\tau} = \bar{B}_r^{-1}[\hat{f} + K_v r - v] \quad (5.3.44)$$

with  $\hat{f}(X)$  an estimate of  $f(X)$ , a gain matrix  $K_v = K_v^T > 0$ , and  $v(t)$  a function to be detailed subsequently that provides robustness. The closed-loop system for the rigid dynamics can now be written as

$$\bar{M}_{rr}\dot{r} = -(K_v + \bar{V}_{rr})r + \tilde{f} + v$$

where the functional estimation error is given by

$$\tilde{f} = f - \hat{f}.$$

Since the nonlinear robot function  $f(X)$  is continuous, it can be approximated by a NN (Chapter 1). Therefore, let the estimate  $\hat{f}(X)$  required in (5.3.44) be provided by a NN so that

$$\hat{f}(X) = \hat{W}^T \sigma(\hat{V}^T X),$$

with  $\hat{V}, \hat{W}$  the current estimated values of the ideal NN weights  $V, W$  as provided by the tuning algorithm. Then, the control signal becomes

$$\bar{\tau} = \bar{B}_r^{-1}[\hat{W}^T \sigma(\hat{V}^T X) + K_v r - v]. \quad (5.3.45)$$

The NN control structure is shown in Fig. 5.3.1, which shows the NN loops as well as the inner fast loop (5.3.38) needed to stabilize the flexible (internal) dynamics.

We make the following reasonable assumption which holds in all practical situations.

**Assumption 5.3.1 (Desired Trajectory and NN Target Weight Bounds) :**

a. The desired motion trajectory is bounded so that

$$\left\| \begin{array}{c} q_d(t) \\ \dot{q}_d(t) \\ \ddot{q}_d(t) \end{array} \right\| \leq q_B, \quad (5.3.46)$$

with  $q_B$  a known scalar bound.

b. Define the matrix of ideal target NN weights as

$$Z \equiv \begin{bmatrix} W & 0 \\ 0 & V \end{bmatrix}. \quad (5.3.47)$$

Then, the ideal NN weights are constant and bounded so that

$$\|Z\|_F \leq Z_B \quad (5.3.48)$$

with  $Z_B$  a known bound.  $\square$

*Now, exactly as in Chapter 4, one may derive NN weight tuning algorithms that guarantee stable tracking. The resulting complete controller for the flexible-link robot arm is given in Table 5.3.1. It can be shown as in Chapter 4 that using this controller, the rigid motion tracking error  $r(t)$  is bounded by a small value that decreases as the gain  $K_v$  increases. See the discussion about the advantages and properties of this sort of NN controller in Chapter 4 or towards the end of Section 5.1.2.*

*As in Chapter 4, this controller is local in the sense that the initial tracking errors must be in a certain set of allowable initial conditions. This initial condition set depends on the speed of the desired trajectory and the size of the compact set over which the NN can suitably approximate the unknown nonlinearity (5.3.43). Approximation accuracy generally increases with the number of hidden-layer neurons in the NN.*

*There is an extra caveat in singular perturbation control—the fast modes must be significantly faster than the slow modes, reflected in a small value of  $\varepsilon$ , otherwise the Tikhonov's Theorem extension does not hold and stability is not guaranteed.*

**Example 5.3.1 (Flexible-Link NN Controller Design Example) :**

The NN control strategy in Table 5.3.1 was tested by means of a simulation of the one-link flexible-arm with pinned-pinned boundary conditions from Example 5.2.1. We choose the trapezoidal velocity profile in Fig. 5.3.2 as the desired trajectory  $q_d(t)$ . The open-loop behavior is detailed in Example 5.2.1 and is unacceptable. In this example the NN controller presented in Table 5.3.1 and shown in Fig. 5.3.1 is used to provide tracking.

**Slow and Fast Subsystems.** According to the matrices in Example 5.2.1, the slow subsystem for  $q_r$  is

$$\begin{aligned} \bar{M}_{rr}\ddot{q}_r + \bar{V}_{rr}\dot{q}_r + \bar{F}_r + \bar{G}_r &= \bar{B}_r\bar{\tau} \\ 2.2024\ddot{q}_r + 0.02\dot{q}_r &= \bar{\tau}. \end{aligned}$$

To write down the fast subsystem corresponding to the flexible modes  $q_{f1}, q_{f2}$  one inverts  $M(q)$  to find that

$$H_{ff} = \begin{bmatrix} 2352.5 & -857.4 \\ -857.4 & 450.7 \end{bmatrix}$$

and notes that

$$K_{ff} = \begin{bmatrix} 14.0733 & 0 \\ 0 & 225.1734 \end{bmatrix}.$$

One may now compute that the fast subsystem (5.3.31) is given by

$$\frac{d}{dT} \begin{bmatrix} \varsigma_1 \\ \varsigma_2 \end{bmatrix} = \begin{bmatrix} 0 & I \\ -\bar{H}_{ff}\tilde{K}_{ff} & 0 \end{bmatrix} \begin{bmatrix} \varsigma_1 \\ \varsigma_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \bar{B}_f^1 \end{bmatrix} \tau_F$$

Table 5.3.1: NN Controller for Flexible-Link Robot Arm

*Control Input:*

$$\tau = \bar{B}_r^{-1}[\hat{W}^T \sigma(\hat{V}^T X) + K_v r - v] + \tau_F$$

*Fast Control:*

$$\tau_F = -\frac{K_{pF}}{\varepsilon^2} q_f - \frac{K_{dF}}{\varepsilon} \dot{q}_f + K_{pF} \bar{\xi}$$

*NN Weight/Threshold Tuning Algorithms:*

$$\begin{aligned}\dot{\hat{W}} &= F\sigma(\hat{V}^T X)r^T - F\hat{\sigma}'\hat{V}^T Xr^T - \kappa F\|r\|\hat{W} \\ \dot{\hat{V}} &= GX\left(\hat{\sigma}'^T \hat{W}r\right)^T - \kappa G\|r\|\hat{V}\end{aligned}$$

*Design parameters:*  $F, G$  positive definite matrices and  $\kappa > 0$  a small parameter.

*Slow Tracking Error:*

$$e(t) = q_d - \bar{q}_r, \quad r = \dot{e} + \Lambda e, \quad \Lambda > 0$$

*Robustifying signal:*

$$v(t) = -K_z(\|\hat{Z}\| + Z_B)r$$

*Slow Manifold Equation:*

$$\bar{\xi} = \tilde{K}_{ff}^{-1}\bar{H}_{ff}^{-1}(-\bar{V}_{fr}^1\dot{\bar{q}}_r - \bar{F}_f^1 - \bar{G}_f^1 + \bar{B}_f^1\bar{\tau}).$$

$$\frac{d}{dT} \begin{bmatrix} \varsigma_1 \\ \varsigma_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -33,108\varepsilon^2 & 193,067\varepsilon^2 & 0 & 0 \\ 12,067\varepsilon^2 & -101,482\varepsilon^2 & 0 & 0 \end{bmatrix} \begin{bmatrix} \varsigma_1 \\ \varsigma_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 4.25 \\ 15.72 \end{bmatrix} \tau_F$$

where  $\varepsilon$  is the selected value of the time-scaling parameter. In this subsystem, the states are

$$\begin{aligned}\varsigma_1 &= \xi - \bar{\xi} = \frac{q_f}{\varepsilon^2} - \frac{\bar{q}_f}{\varepsilon^2} \\ \varsigma_2 &= \varepsilon \dot{\xi} = \frac{\dot{q}_f}{\varepsilon}\end{aligned}$$

To study the open-loop flexible modes, setting  $\varepsilon = 1$  one finds the poles of the fast subsystem to be

$$s = \begin{pmatrix} 0.0 \pm 90.26i \\ 0.0 \pm 355.59i \end{pmatrix}$$

which should be compared with the exact open-loop poles in Example 5.2.1.

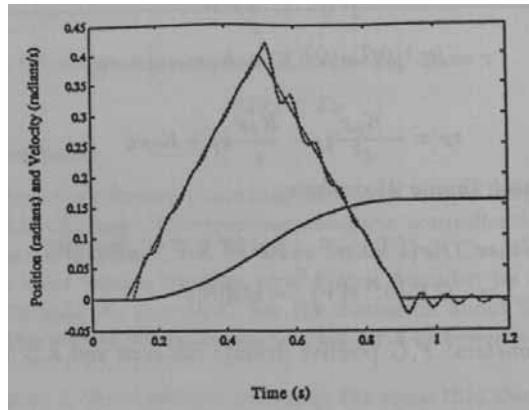


Figure 5.3.2: Response of flexible arm with NN and boundary layer correction.  
Actual and desired tip positions and velocities,  $\varepsilon = 0.26$ .

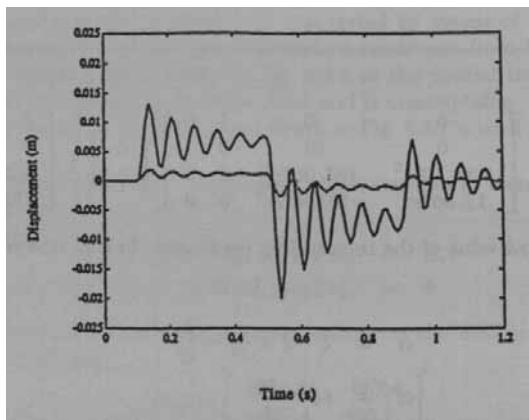


Figure 5.3.3: Response of flexible arm with NN and boundary layer correction.  
Flexible modes,  $\varepsilon = 0.26$ .

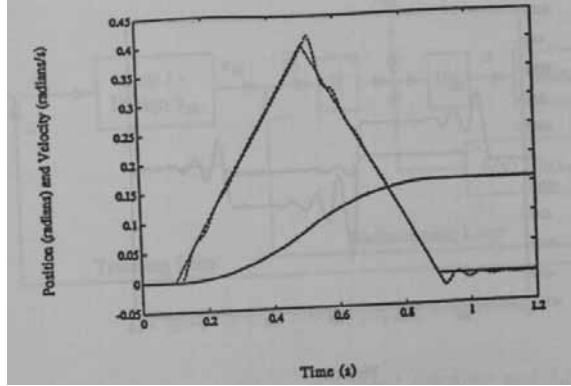


Figure 5.3.4: Response of flexible arm with NN and boundary layer correction. Actual and desired tip positions and velocities,  $\varepsilon = 0.1$ .

**Fast Controls Design.** An LQR design was performed via MATLAB on the fast system to select the PD gains in the fast boundary-layer correction control

$$\tau_F = -K_{pF}\xi_1 - K_{dF}\xi_2. \quad (5.3.49)$$

We used  $Q = \text{diag}(1, 10, 1, 10)$  and  $R = 100$  for two different values of  $\varepsilon$ . First,  $\varepsilon$  was selected to be equal to the square root of the reciprocal of the smallest stiffness constant 14.0733. This value of  $\varepsilon = 0.26$  yields a gain of

$$K_F = [K_{pF} \ K_{dF}] = [-0.1696 \ 1.0229 \ 0.0479 \ 0.4726].$$

A smaller value of  $\varepsilon = 0.1$  gives a gain of

$$K_F = [-0.1674 \ 1.0197 \ 0.0485 \ 0.4721].$$

Note that the boundary-layer correction control must be implemented using

$$\tau_F = -\frac{K_{pF}}{\varepsilon^2}q_f - \frac{K_{dF}}{\varepsilon}\dot{q}_f + K_{pF}\bar{\xi}$$

so that the slow manifold variable  $\bar{\xi}$  must be computed.

**Neural Net Controller and Simulation.** The NN design parameters in Table 5.3.1 were selected as  $K_v = 20, \Lambda = 5, F = G = 20, \kappa = 1, K_z = 20, Z_B = 50$ . The controller was first simulated using the fast subsystem gains for  $\varepsilon = 0.26$ . In Fig. 5.3.2 the closed-loop performance for position and velocity can be seen. The flexible modes are shown in Fig. 5.3.3.

Next, we used the fast gains corresponding to  $\varepsilon = 0.1$ . In Fig. 5.3.4 we see that the tracking performance is considerably improved. The flexible modes shown in Fig. 5.3.5 are also improved. Therefore, the proposed NN-based controller has excellent performance that improves as  $\varepsilon$  decreases. This is in spite of the fact that the controller requires no knowledge of the actual arm matrices; the controller learns the dynamics by automatically tuning the NN weights on-line.

It should be realized that if  $\varepsilon$  is taken too small, the result will be control chattering and deteriorated performance.  $\square$

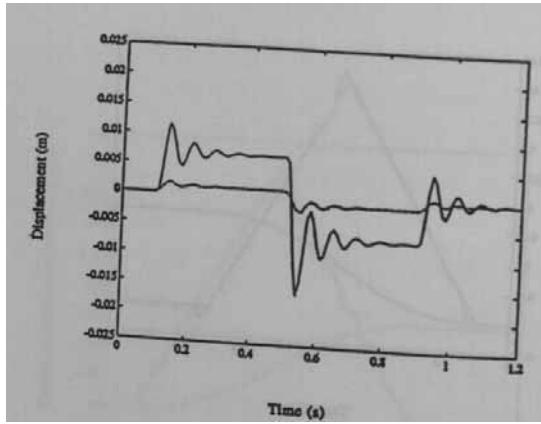


Figure 5.3.5: Response of flexible arm with NN and boundary layer correction. Flexible modes,  $\varepsilon = 0.1$ .

## 5.4 BACKSTEPPING DESIGN

We have seen that some practical robotic systems cannot be controlled using the rigid robot arm techniques described in Chapter 4, including robots with link vibration, joint compliance, and actuators having fast dynamics (e.g. electrical dynamics). In the previous section, singular perturbation theory was used to extend the control effectiveness in the system to deal with this problem. Here, we introduce the technique of backstepping to increase the control effectiveness.

### 5.4.1 Backstepping Design

Backstepping design (Kanellakopoulos et al. 1991, Kokotovic 1992) is a method for extending various controller design techniques to a wider class of systems than originally possible using that technique. It is conveniently studied using Lyapunov proof techniques. Also relevant are the notions of feedback linearization and zero dynamics. These concepts were discussed in Chapter 2.

Consider the class of systems of the form

$$\dot{x}_1 = f_1(x) = f_1(x_1, x_2) \quad (5.4.1)$$

$$\dot{x}_2 = f_2(x) + g_2(x)u \quad (5.4.2)$$

with state  $x = [x_1^T \ x_2^T]^T$  and control input  $u(t)$ . Note that this system is of the form in Fig. 5.2.4a if the output is selected as  $x_2(t)$ , and of the form in Fig. 5.2.4b if the output is selected as  $x_1(t)$ .

If  $x_2(t)$  is selected as the output, then the zero dynamics are defined as the dynamics when  $u(t)$  is chosen to make  $x_2(t) = 0$ . Thus, the zero dynamics are given by

$$\dot{x}_1 = f_1(x_1, 0). \quad (5.4.3)$$

If these dynamics are unstable, the system is of non-minimum phase. Backstepping notions can be applied to control such systems as long as the internal dynamics

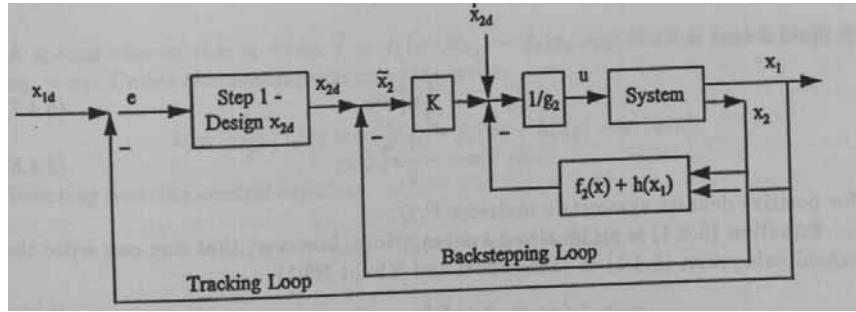


Figure 5.4.1: Backstepping controller.

are stabilizable considering  $x_2(t)$  as the input (Slotine and Li 1991). This class of systems includes the flexible-link arms, which were stabilized using a different technique in the previous section.

In traditional backstepping, the output is selected as  $x_1(t)$ , which might be required to follow a prescribed trajectory  $x_{1_d}(t)$ . This corresponds to the cascaded dynamics in Fig. 5.2.4b of the sort representing the flexible-joint robot and RLED robot. In the upcoming presentation we take this point of view.

To select the control input, let us proceed in two steps. First, select a desirable value of  $x_2$ , possibly a function of  $x_1$ , denoted  $x_{2_d}(x_1, t)$ , such that in the ideal system

$$\dot{x}_1 = f_1(x_1, x_{2_d}) \quad (5.4.4)$$

one has stable tracking by  $x_1(t)$  of  $x_{1_d}(t)$ . Then, in a second step, select the control input  $u(t)$  so that the actual value of  $x_2(t)$  becomes  $x_{2_d}$ . This two-step design procedure is known as backstepping, since one starts at the desired output, and backsteps through the system selecting desirable values of the state components until the actual control input  $u(t)$  is reached. Under conditions discussed below, a stable tracking controller can be designed in this manner. The result of this two-step design procedure is the two-loop controller shown in Fig. 5.4.1, which is subsequently derived in more detail. When it works, the technique extends to more than two systems in cascade.

To formalize this technique, one requires

**Assumption 5.4.1 (Stabilizability Assumption)** : Let (5.4.1) be stabilizable considering  $x_2(t)$  as the control input. Let (5.4.2) be stabilizable using input  $u(t)$ .  $\square$

Suppose for simplicity that the desired output  $x_{1_d}$  is equal to zero. (The extension to  $x_{1_d}(t) \neq 0$  is discussed in the Problems section). Then, under the assumption one is able to select in the first design step a desired value  $x_{2_d}$  so that (5.4.4) is stable. If one is talking about asymptotic stability (AS) and (5.4.4) is autonomous (not an explicit function of time), this assumption implies there exists a Lyapunov function  $L_1(x_1)$  such that

$$L_1 > 0 \quad (5.4.5)$$

$$\dot{L}_1 = \frac{\partial L_1}{\partial x_1} \dot{x}_1 = \frac{\partial L_1}{\partial x_1} f_1(x_1, x_{2_d}) < 0. \quad (5.4.6)$$

A special case is

$$L_1 = \frac{1}{2} x_1^T P x_1 \quad (5.4.7)$$

$$\dot{L}_1 = -\frac{1}{2} x_1^T Q x_1 \quad (5.4.8)$$

for positive definite symmetric matrices  $P, Q$ .

Equation (5.4.4) is an idealized system. Note, however, that one can write the actual subsystem (5.4.1) as (Esfandiari and Khalil 1991)

$$\dot{x}_1 = f_1(x_1, x_{2_d}) + f_1(x_1, x_2) - f_1(x_1, x_{2_d})$$

or

$$\dot{x}_1 = f_1(x_1, x_{2_d}) - \tilde{f}_1 \quad (5.4.9)$$

which is the ideal system driven by a mismatch error term  $\tilde{f}_1 \equiv f_1(x_1, x_{2_d}) - f_1(x_1, x_2)$ .

Now, define a new error signal as

$$\tilde{x}_2 = x_{2_d} - x_2 \quad (5.4.10)$$

and find the dynamics

$$\begin{aligned} \dot{\tilde{x}}_2 &= \dot{x}_{2_d} - \dot{x}_2 \\ \dot{\tilde{x}}_2 &= \dot{x}_{2_d} - f_2(x) - g_2(x)u. \end{aligned} \quad (5.4.11)$$

The complete dynamics (5.4.1)-(5.4.2) can therefore be represented as (5.4.9), (5.4.11).

For the second design step, one must select the input  $u(t)$  in (5.4.11) to stabilize (5.4.9)/(5.4.11), which is possible under Assumption 5.4.1. To accomplish this, define the overall Lyapunov function

$$L = L_1 + \frac{1}{2} \tilde{x}_2^T \tilde{x}_2. \quad (5.4.12)$$

Differentiating, and using the actual dynamics (5.4.9)/(5.4.11) one obtains

$$\begin{aligned} \dot{L} &= \dot{L}_1 + \tilde{x}_2^T \dot{\tilde{x}}_2 \\ &= \frac{\partial L_1}{\partial x_1} [f_1(x_1, x_{2_d}) - \tilde{f}_1] + \tilde{x}_2^T [\dot{x}_{2_d} - f_2(x) - g_2(x)u]. \end{aligned}$$

Assume for simplicity that (5.4.8) holds (this is not required for the technique to work, but brings out some intuition in this development). Then,

$$\dot{L} = -\frac{1}{2} x_1^T Q x_1 - \frac{\partial L_1}{\partial x_1} \tilde{f}_1 + \tilde{x}_2^T [\dot{x}_{2_d} - f_2(x) - g_2(x)u].$$

To proceed, one needs some more structure in the term  $\frac{\partial L_1}{\partial x_1} \tilde{f}_1$ . Suppose, for instance, that one has the following

**Assumption 5.4.2 (Linearity Assumption)** : Let

$$\frac{\partial L_1}{\partial x_1} \tilde{f} = h^T(x_1) \tilde{x}_2 \quad (5.4.13)$$

for some function  $h(x_1)$ .  $\square$

A special case of this is when  $\tilde{f} = f_1(x_1, x_{2d}) - f_1(x_1, x_2)$  itself is linear in  $\tilde{x}_2 = x_{2d} - x_2$ . Under this assumption one may write

$$\dot{L} = -\frac{1}{2}x_1^T Q x_1 + \tilde{x}_2^T [\dot{x}_{2d} - f_2(x) - h(x_1) - g_2(x)u].$$

Selecting now the control input as

$$u = \frac{1}{g_2} [\dot{x}_{2d} - f_2(x) - h(x_1) + K\tilde{x}_2] \quad (5.4.14)$$

yields

$$\dot{L} = -\frac{1}{2}x_1^T Q x_1 - \tilde{x}_2^T K \tilde{x}_2 < 0, \quad (5.4.15)$$

so the overall system is stable. For success using this control strategy one must make

**Assumption 5.4.3 (Bounded Below Assumption)** : Let  $g_2(x)$  be positive and bounded below so that

$$\gamma_B < g_2(x) \quad (5.4.16)$$

for all  $x$  and some bound  $\gamma_B$ .  $\square$

Note that then  $\frac{1}{g_2(x)} < \frac{1}{\gamma_B}$  for all  $x$ , which is finite so that the control is well-defined.

The closed-loop system designed using backstepping is depicted in Fig. 5.4.1. Note that the selection of  $x_{2d}(t)$  produces a first control loop, and the selection of  $u(t)$  produces a second loop generating the final control input.

One notes that the linearity assumption 5.4.2 is not required. A milder assumption is the following.

**Assumption 5.4.4 (Lipschitz-Like Assumption)** : Let

$$\left\| \frac{\partial L_1}{\partial x_1} \tilde{f} \right\| \leq h(x_1) \|\tilde{x}_2\| \quad (5.4.17)$$

for some function  $h(x_1)$ .  $\square$

Using this assumption one may also select a stabilizing control  $u(t)$ . See Esfandiari and Khalil (1991) and the Problem section.

**Example 5.4.1 (Backstepping Control Design)** :

This example is motivated by some examples in Slotine and Li (1991). Let the system be

$$\begin{aligned} \dot{x}_1 &= x_1^2 x_2 \\ \dot{x}_2 &= x_2 - x_1^2 + u. \end{aligned}$$

**Step 1.** To stabilize the first subsystem, one may select  $x_{2d} = -x_1$ , for then the ideal system (5.4.4) is

$$\dot{x}_1 = x_1^2 x_{2d} = -x_1^3$$

which has a Lyapunov function given by

$$L_1 \equiv \frac{1}{2} x_1^2$$

for

$$\dot{L}_1 = x_1 \dot{x}_1 = -x_1^4.$$

**Step 2.** The actual  $x_1$  subsystem is

$$\begin{aligned}\dot{x}_1 &= x_1^2 x_2 = x_1^2 x_{2d} - x_1^2 (x_{2d} - x_2) \\ &= -x_1^3 - x_1^2 \tilde{x}_2\end{aligned}$$

with

$$\tilde{x}_2 \equiv x_{2d} - x_2 = -x_1 - x_2.$$

The dynamics of  $\tilde{x}_2$  are given by

$$\begin{aligned}\dot{\tilde{x}}_2 &= -\dot{x}_1 - \dot{x}_2 \\ &= -x_1^2 x_2 - x_2 + x_1^2 - u.\end{aligned}$$

An overall Lyapunov function candidate is

$$L = L_1 + \frac{1}{2} \tilde{x}_2^2 = \frac{1}{2} x_1^2 + \frac{1}{2} \tilde{x}_2^2.$$

Evaluating along the actual system trajectories one has

$$\begin{aligned}\dot{L} &= \dot{L}_1 + \tilde{x}_2 \dot{\tilde{x}}_2 = x_1 \dot{x}_1 + \tilde{x}_2 \dot{\tilde{x}}_2 \\ &= x_1 (-x_1^3 - x_1^2 \tilde{x}_2) + \tilde{x}_2 (-x_1^2 x_2 - x_2 + x_1^2 - u) \\ &= -x_1^4 + \tilde{x}_2 (-x_1^2 x_2 - x_2 + x_1^2 - x_1^3 - u).\end{aligned}$$

Selecting now the control input

$$u = -x_1^2 x_2 - x_2 + x_1^2 - x_1^3 + k \tilde{x}_2$$

yields the negative definite function

$$\dot{L} = -x_1^4 - k \tilde{x}_2^2.$$

One notes that the function of the control input  $u(t)$  is to stabilize the second subsystem about  $x_{2d} = -x_1$ . It has some components from the dynamics of the second subsystem (e.g.  $-x_1^2 x_2 - x_2 + x_1^2 + k \tilde{x}_2$ ) and some components from the dynamics of the first subsystem (e.g.  $-x_1^3$ ).  $\square$

#### 5.4.2 NN Controller for Rigid-Link Electrically-Driven Robot Using Backstepping

*In the previous subsection we introduced backstepping design for the case where all the system dynamics are fully known. When the parameters are not completely known, backstepping can be applied in conjunction with techniques such as adaptive control (Kokotovic 1992) or robust control. A disadvantage of standard adaptive control backstepping is that a stringent linear-in-the-parameters assumption is*

needed on all nonlinearities (stronger than Assumption 5.4.2), so that the form of the nonlinearities must be known. This leads to requirements to determine two regression matrices (basically one for  $f_1$  and one for  $f_2$ ), and to differentiate the first of them. This can be very tedious.

In this subsection we apply backstepping design to the rigid-link electrically-driven (RLED) manipulator introduced in Subsection 5.2.3. It is shown that backstepping can be applied in conjunction with the neural net control techniques of Chapter 4, so that no regression matrices are needed and no linearity-in-the-parameters assumptions are needed. The key to this is the NN universal approximation property which is not an assumption, but holds for all smooth functions. Two NN will be required in this controller, one basically to estimate the nonlinearities  $f_1$  and another basically for  $f_2$ . This is the work of C. Kwan (see References), who has also applied backstepping to NN control of flexible-joint robot arms, induction motors, and a large class of nonlinear systems.

The dynamics of the RLED robot are

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = K_T i \quad (5.4.18)$$

$$Li + R(i, \dot{q}) + \tau_e = u_e \quad (5.4.19)$$

with  $q(t) \in \Re^n$  the joint variable,  $i(t) \in \Re^n$  the motor armature currents, and  $u_e(t) \in \Re^n$  the control input voltage.

#### 5.4.2.1 Trajectory Tracking and Backstepping Error Dynamics

This derivation combines some techniques in Chapter 4 with backstepping techniques. Define the tracking error

$$e = q_d - q, \quad (5.4.20)$$

with  $q_d(t)$  the desired robot arm trajectory, and the filtered tracking error

$$r = \dot{e} + \Lambda e \quad (5.4.21)$$

with  $\Lambda > 0$  a diagonal design matrix. Differentiating  $r(t)$  and using (5.4.18) one finds the dynamics in terms of  $r(t)$  as

$$M(q)\ddot{r} = -V_m r + F_1 + \tau_d - K_T i \quad (5.4.22)$$

where the unknown nonlinear robot function is

$$F_1(X_1) = M(q)(\ddot{q}_d + \Lambda \dot{e}) + V_m(q, \dot{q})(\dot{q}_d + \Lambda e) + F(\dot{q}) + G(q) \quad (5.4.23)$$

and, for instance, one may select

$$X_1 = \begin{bmatrix} e \\ \dot{e} \\ q_d \\ \dot{q}_d \\ \ddot{q}_d \end{bmatrix}.$$

The dynamics (5.4.22)/(5.4.19) are of the form (5.4.1)/(5.4.2) with  $x_1 = r$  and  $x_2 = i$ , and satisfy the assumptions in the previous subsection.

Now let  $i_d(t)$  be a value of  $i(t)$  that stabilizes the dynamics (5.4.22) written in the form

$$M\dot{r} = -V_m r + F_1 + \tau_d - K_T i_d + K_T \eta, \quad (5.4.24)$$

with

$$\eta \equiv i_d - i \quad (5.4.25)$$

an error term. Signal  $i_d(t)$  will be selected later. To find the complete error dynamics, differentiate  $L\eta$  and substitute from (5.4.19) to find

$$L\dot{\eta} = F_2(X_2) + \tau_e - u_e \quad (5.4.26)$$

where the unknown nonlinear motor function is

$$F_2(X_2) = L\dot{i}_d + R(i, \dot{q}), \quad (5.4.27)$$

with  $R(i, \dot{q})$  the motor electrical resistance and back emf. Since  $i_d(t)$  will turn out to be a complex function of  $F_1(X_1)$  and  $r(t)$ , the function  $F_2(X_2)$  is complicated. Note moreover that  $F_2$  requires the derivative of  $i_d(t)$ . For  $X_2$ , one may select, for instance,

$$X_2 = \begin{bmatrix} i \\ e \\ \dot{e} \\ q_d \\ \dot{q}_d \\ \ddot{q}_d \end{bmatrix}.$$

#### 5.4.2.2 Neural Network Backstepping Controller

The system dynamics has now been written in terms of the error subsystems (5.4.24) and (5.4.26). Here we shall show how to select  $i_d(t)$  and  $u_e(t)$  so that these dynamics are stable, yielding tracking of the desired robot arm trajectory  $q_d(t)$ . The resulting NN controller is shown in Fig. 5.4.2. This figure is similar to the NN controllers in Chapter 4, but has an additional feedback loop as required for backstepping from  $i_d$  to  $u_e$ .

Note that matrix  $K_T$  is unknown. Assume that

$$K_{B1} < \|K_T\| < K_{B2} \quad (5.4.28)$$

with  $K_{B1}, K_{B2}$  known scalar bounds.

Assume next that there exists two networks that can approximate the unknown nonlinear functions  $F_1$  and  $F_2$  on a compact set. Thus,

$$F_1(X_1) = W_1^T \phi_1(X_1) + \varepsilon_1 \quad (5.4.29)$$

$$F_2(X_2) = W_2^T \phi_2(X_2) + \varepsilon_2, \quad (5.4.30)$$

with  $W_i$  the tunable NN weights,  $\phi_i(\cdot)$  the activation functions, and  $\varepsilon_i$  the NN functional reconstruction errors that are assumed bounded so that

$$\|\varepsilon_i\| < \varepsilon_{iN} \quad (5.4.31)$$

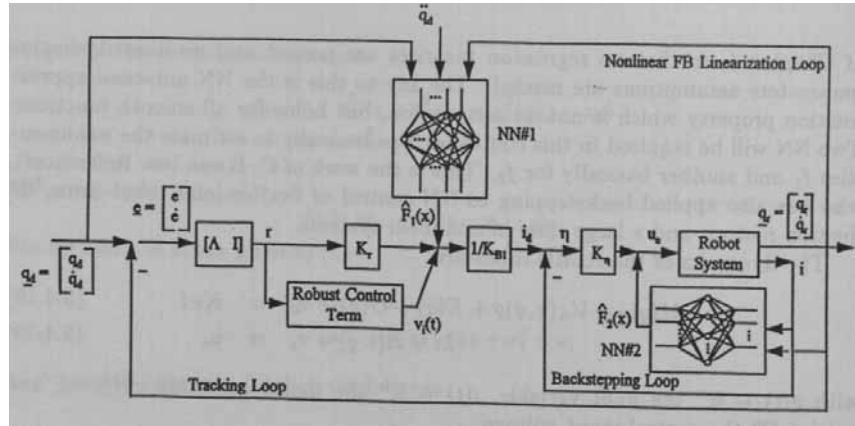


Figure 5.4.2: Backstepping neural network controller.

with  $\varepsilon_{1N}, \varepsilon_{2N}$  known bounds. For simplicity, we are using one-layer functional-link neural networks (FLNN— see Chapter 4), although two-layer NN could be used. In the case of one-layer NN the functions  $\phi_i(\cdot)$  must be selected to provide a basis. One could use radial basis functions, or the CMAC neural net (Commuri and Lewis 1995).

Estimates for  $F_1, F_2$  are given by

$$\hat{F}_1(X_1) = \hat{W}_1^T \phi_1(X_1) \quad (5.4.32)$$

$$\hat{F}_2(X_2) = \hat{W}_2^T \phi_2(X_2), \quad (5.4.33)$$

with  $\hat{W}_i$  the current values of the NN weights as provided by the tuning algorithms.

Now select the desirable value of armature current as

$$i_d = \frac{1}{K_{B1}} (\hat{F}_1 + K_r r + v_i) \quad (5.4.34)$$

$$= \frac{1}{K_{B1}} (\hat{W}_1^T \phi_1 + K_r r + v_i) \quad (5.4.35)$$

with  $K_r > 0$  a gain matrix and  $v_i$  a robustifying term to be defined. This is a feedback-linearization motivated design based on (5.4.24). Select the control input as

$$u_e = \hat{F}_2 + K_\eta \eta \quad (5.4.36)$$

$$= \hat{W}_2^T \phi_2 + K_\eta \eta. \quad (5.4.37)$$

with  $K_\eta > 0$  a gain matrix. This is a feedback-linearization motivated design based on (5.4.26).

Now, using techniques very much like those in Chapter 4, one may show that the NN backstepping control given in its entirety in Table 5.4.1 gives stable tracking performance. The NN backstepping controller is depicted in Fig. 5.4.2. The stability

Table 5.4.1: NN Backstepping Controller for RLED Robot Arm

---

*Control Input:*

$$u_e = \hat{W}_2^T \phi_2 + K_\eta \eta$$

*Auxiliary Signal:*

$$i_d(t) = \frac{1}{K_{B1}} (\hat{W}_1^T \phi_1 + K_r r + v_i)$$

*NN Weight/Threshold Tuning Algorithms:*

$$\begin{aligned} \dot{\hat{W}}_1 &= \Gamma_1 \phi_1 r^T - \kappa_1 \Gamma_1 \|\xi\| \hat{W}_1 \\ \dot{\hat{W}}_2 &= \Gamma_2 \phi_2 r^T - \kappa_2 \Gamma_2 \|\xi\| \hat{W}_2 \end{aligned}$$

where  $\xi = [r^T \ \eta^T]^T$ .

*Design parameters:*  $\Gamma_1, \Gamma_2$  positive definite matrices and  $\kappa_1, \kappa_2 > 0$  small parameters.

*Tracking Error:*

$$\begin{aligned} e(t) &= q_d - q \\ r(t) &= \dot{e} + \Lambda e, \quad \Lambda > 0 \\ \eta(t) &= i_d(t) - i(t) \end{aligned}$$

*Robustifying signal:*

$$v_i(t) = \frac{r \rho^2}{\|r\| \rho + \varepsilon_v}, \quad \rho = \|\hat{W}_1^T \phi_1\|, \quad \varepsilon_v > 0 \text{ and small.}$$


---

proof hinges on using the Lyapunov function

$$L = \frac{1}{2} \xi^T P \xi + \frac{1}{2} \tilde{Z}^T \Gamma^{-1} \tilde{Z} \quad (5.4.38)$$

where  $\xi = [r^T \ \eta^T]^T$ ,  $Z = \text{diag}\{W_1, W_2\}$ ,  $\Gamma = \text{diag}\{\Gamma_1, \Gamma_2\} > 0$ ,  $P > 0$ . This Lyapunov function weights both the tracking errors and the NN weight estimation errors. The details are given in Kwan and Lewis (July 1995). It can be shown that the tracking error  $r(t)$  (and  $\eta(t)$ ) can be made arbitrarily small by increasing the gains  $K_r, K_\eta$ .

The NN backstepping controller enjoys the properties and advantages discussed in Chapter 4 and towards the end of Section 5.1.2. Included are fast on-line weight tuning with no preliminary off-line learning, no persistence of excitation, and no need to compute any regression matrices. One notes that the nonlinear function  $F_2$  depends on  $i_d$ ; however,  $i_d(t)$  is a function of  $\hat{F}_1$ . In backstepping using standard adaptive control techniques, one must determine a regression matrix for  $F_1$ , and

then differentiate it to find a regression matrix for  $F_2$ . All this is unnecessary using the NN approach.

Initialization of the NN weights at zero amounts to setting  $\hat{F}_1 = 0, \hat{F}_2 = 0$  in Fig. 5.4.2. The reason this works is that, setting  $\hat{F}_1 = 0, \hat{F}_2 = 0$  in the expressions for  $u_e(t)$  and  $i_d(t)$  in Table 5.4.1 gives

$$\begin{aligned} u_e &= K_\eta(i_d - i) = K_\eta\left[\frac{1}{K_{B_1}}(K_r r + v_i) - i\right] \\ &= \frac{K_\eta K_r}{K_{B_1}}r + \frac{K_\eta}{K_{B_1}}v_i - K_\eta i, \end{aligned} \quad (5.4.39)$$

which is a PD tracking controller plus extra terms in the robust term  $v_i(t)$  and the armature current  $i(t)$ . This holds the system in stable tracking until the NN begin to learn, improving the closed-loop performance.

As detailed in Chapter 4, this controller is local in the sense that the initial tracking errors must be in a certain set of allowable initial conditions. This initial condition set depends on the speed of the desired trajectory and the size of the compact sets over which the approximation properties (5.4.30) hold. Approximation accuracy generally increases with the number of hidden-layer neurons in the NN.

**Example 5.4.2 (RLED Backstepping NN Controller Design Example) :**

The backstepping NN controller in Table 5.4.1 was applied to control the two-link planar elbow arm in Example 5.1.1. The arm parameters were selected as  $a_1 = a_2 = 1\text{ m}$ ,  $m_1 = 0.8\text{ kg}$ ,  $m_2 = 2.3\text{ kg}$ ,  $g = 9.8\text{ m/sec}^2$ , which yields  $\alpha = 3.1$ ,  $\beta = 2.3$ ,  $\eta = 2.3$ . The two actuator motors were taken as permanent magnet DC motors with  $L_i = 0.01\text{ H}$ ,  $K_{T_i} = 2\text{ N-m/A}$ , back emf constant of one, and armature resistance  $R_{a_i} = 1\Omega$ , for  $i = 1, 2$ , which gives dynamics (5.4.19) of

$$\begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix} \dot{i} + i + \dot{q} = u_e.$$

with variables in  $\mathbb{R}^2$ . The EM conversion matrix is  $K_T = 2I$ , with  $I$  the identity matrix. We selected the lower bound  $K_{B_1} = 1$ .

The desired trajectories were  $q_{d1}(t) = \sin t$ ,  $q_{d2}(t) = \cos t$ . The inputs to both NN were taken as

$$X_1 = X_2 = [\zeta_1^T \ \zeta_2^T \ \cos(q)^T \ \sin(q)^T \ \dot{q}^T \ i^T \ 1]^T$$

with  $\zeta_1 = \ddot{q}_d + \Lambda \dot{e}, \zeta_2 = \dot{q}_d + \Lambda e$ . We selected  $\Lambda = 20I$ . This choice of  $X_1, X_2$  contains preprocessing of signals to give the NN more information and improve its performance, as discussed in Chapter 4. We selected the controller gains as  $K_r = 20I, K_\eta = 50I$ . In each NN we used ten hidden-layer neurons. The weight tuning parameters were selected as  $\Gamma_1 = \Gamma_2 = 20I, \kappa_1 = \kappa_2 = 0.5$ .

First, only PD control was used, which amounts to setting all the NN weights to zero; this disables the two nonlinear feedback loops in Fig. 5.4.2 and gives the PD controller (5.4.39). In this simulation, we also left out the terms there in  $v_i(t)$  and  $i(t)$  so that  $u_e = (K_\eta K_r / K_{B_1})r$ . The result is shown in Fig. 5.4.3. The tracking error is bounded, but does not decrease with time. By increasing the gains one can make  $r(t)$  smaller, but at the expense of increased control magnitude.

Next, we enabled the two NN loops to provide estimates  $\hat{F}_1, \hat{F}_2$ . The resulting performance shown in Fig. 5.4.4 is excellent. Note that the NN controller does not require any of the parameters of the robot arm or motor dynamics. It learns all this information on line by NN weight tuning.  $\square$

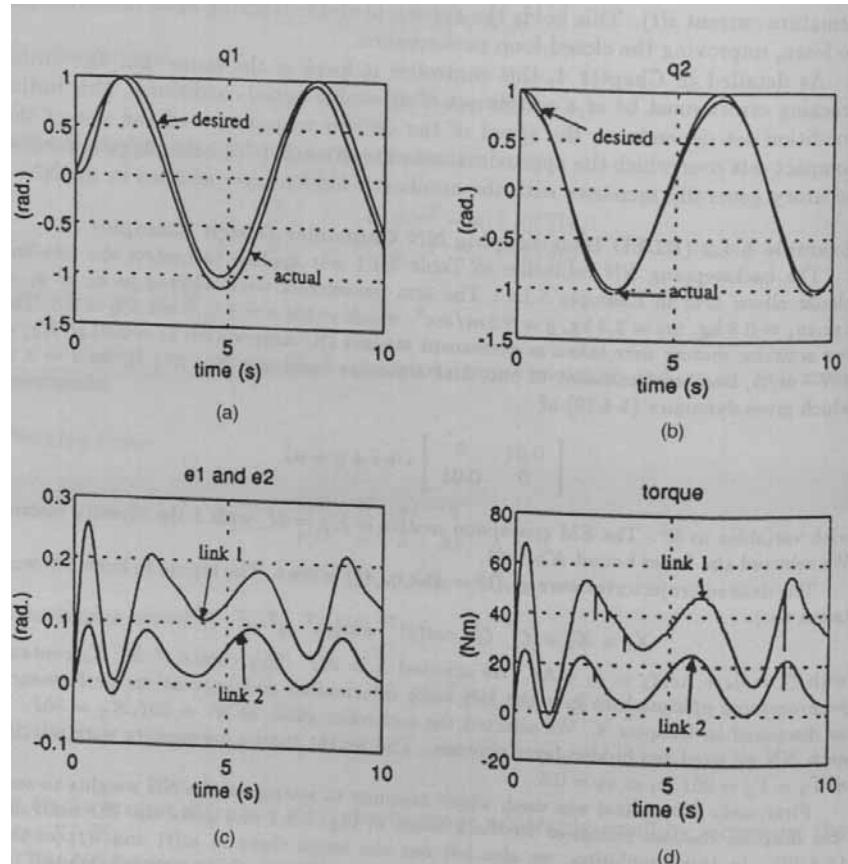


Figure 5.4.3: Response of RLED controller with only PD control. (a) Actual and desired joint angle  $q_1(t)$ . (b) Actual and desired joint angle  $q_2(t)$ . (c) Tracking errors  $e_1(t), e_2(t)$ . (d) Control torques  $K_{Ti}(t)$ .

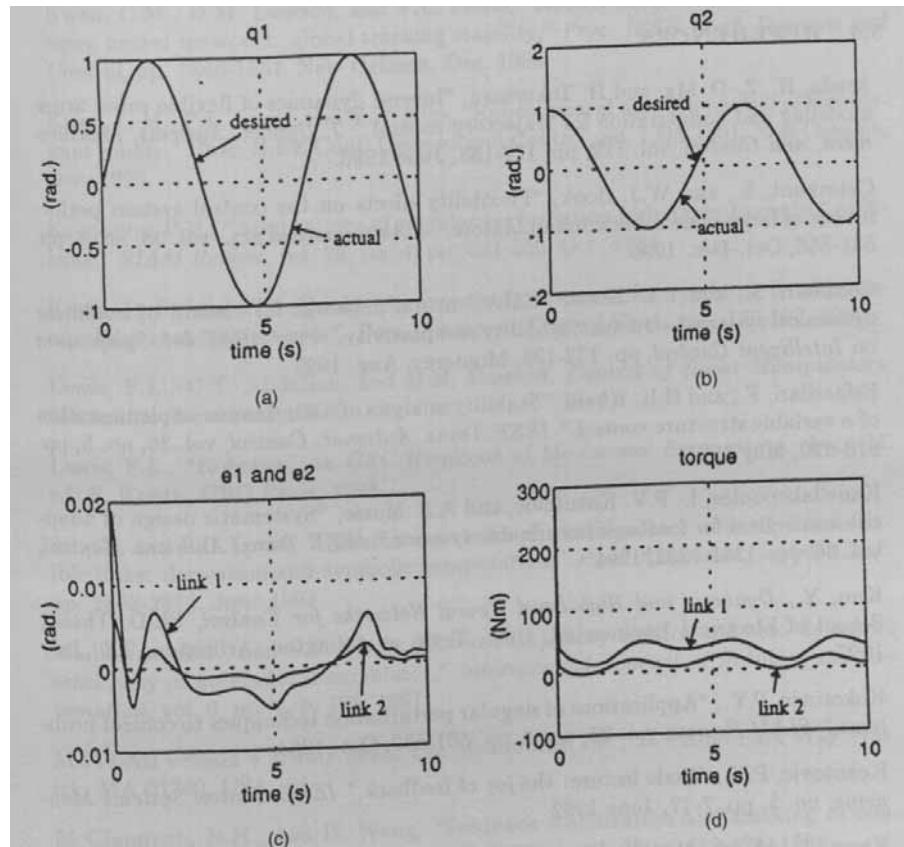


Figure 5.4.4: Response of RLED backstepping NN controller. (a) Actual and desired joint angle  $q_1(t)$ . (b) Actual and desired joint angle  $q_2(t)$ . (c) Tracking errors  $e_1(t), e_2(t)$ . (d) Control torques  $K_T i(t)$ .

## 5.5 CONCLUSIONS

*In this chapter we explored some extensions of the basic neural network control techniques developed in Chapter 4. We discussed some practical industrial applications such as force control, systems with vibratory and flexible modes, and systems with drive train and actuator high-frequency dynamics. It was shown how to use singular perturbations and backstepping techniques to extend the class of systems amenable to our NN design approach. The controllers in this chapter contained the basic PD loop and NN inner feedback loop of Chapter 4, but also some extra feedback loops to deal with the more complex structures of systems dealt with here.*

## 5.6 REFERENCES

- Asada, H., Z.-D. Ma, and H. Tokumaru, "Inverse dynamics of flexible robot arms: modeling and computation for trajectory control," *J. Dynam. Systems, Measurement, and Control*, vol. 112, pp. 177-185, June 1990.
- Cetinkunt, S., and W.J. Book, "Flexibility effects on the control system performance of large-scale robotic manipulators," *J. Astronautical Sci.*, vol. 38, no. 4, pp. 531-556, Oct.-Dec. 1990.
- Commuri, S., and F.L. Lewis, "CMAC neural networks for control of nonlinear dynamical systems: structure, stability and passivity," *Proc. IEEE Int. Symposium on Intelligent Control*, pp. 123-129, Monterey, Aug. 1995.
- Esfandiari, F., and H.K. Khalil, "Stability analysis of a continuous implementation of a variable structure control," *IEEE Trans. Automat. Control*, vol. 36, no. 5, pp. 616-620, May 1991.
- Kanellakopoulos, I., P.V. Kokotovic, and A.S. Morse, "Systematic design of adaptive controllers for feedback linearizable systems," *IEEE Trans. Automat. Control*, vol. 36, pp. 1241-1253, 1991.
- Kim, Y., *Dynamic and High-Level Neural Networks for Control, Ph.D. Thesis, School of Electrical Engineering, Univ. Texas at Arlington, Arlington, TX, July 1997*.
- Kokotovic, P.V., "Applications of singular perturbation techniques to control problems," *SIAM Review*, vol. 26, no. 4, pp. 501-550, Oct. 1984.
- Kokotovic, P.V., "Bode lecture: the joy of feedback," *IEEE Control Systems Magazine*, no. 3, pp. 7-17, June 1992.
- Kwan, C.-M., A. Yeşildirek, and F.L. Lewis, "Robust force/motion control of constrained robots using neural network," *Proc. IEEE Conf. Decision and Control*, pp. 1862-1867, Dec. 1994.
- Kwan, C.-M., A. Yeşildirek, and F.L. Lewis, "Robust force/motion control of flexible-joint robots using neural networks," *Proc. American Control Conf.*, pp. 4460-4465, Seattle, June 1995.

- Kwan, C.M., and F.L. Lewis, "Robust backstepping control of induction motors using neural networks," Proc. IEEE Mediterranean Symp. on New Directions in Control and Automation, pp. 323-330, Cyprus, July 1995.*
- Kwan, C.M., F.L. Lewis, and D.M. Dawson, "Robust neural network control of rigid-link electrically-driven robots," Proc. IEEE Int. Symposium on Intelligent Control, pp. 117-122, Monterey, Aug. 1995.*
- Kwan, C.M., and F.L. Lewis, "Robust backstepping control of nonlinear systems using neural networks," Proc. European Control Conf., pp. 2772-2777, Rome, Sept 1995.*
- Kwan, C.M., D.M. Dawson, and F.L. Lewis, "Robust adaptive control of robots using neural networks: global tracking stability," Proc. IEEE Conf. Decision and Control, pp. 1846-1851, New Orleans, Dec. 1995.*
- Kwan, C.M., F.L. Lewis, and Y.H. Kim, "Robust neural network control of flexible-joint robots," Proc. IEEE Conf. Decision and Control, pp. 1296-1301, New Orleans, Dec. 1995.*
- Kokotovic, P.V., "Applications of singular perturbation techniques to control problems," SIAM Review, vol. 26, no. 4, pp. 501-550, Oct. 1984.*
- Kwon, D.-S., and W.J. Book, "An inverse dynamic method yielding flexible manipulator state trajectories," Proc. American Control Conf., pp., 186-193, 1990.*
- Lewis, F.L., C.T. Abdallah, and D.M. Dawson, Control of Robot Manipulators, Macmillan, New York, 1993.*
- Lewis, F.L., "Robotics", in CRC Handbook of Mechanical Engineering, chap. 14, ed. F. Kreith, CRC Press, 1998.*
- Lin, J., and F.L. Lewis, "Dynamic equations of a manipulator with rigid and flexible links: derivation and symbolic computation," Proc. American Control Conf., pp. 2868-2872, June 1993.*
- Madhavan, S.K., and S.N. Singh, "Inverse trajectory control and zero dynamics sensitivity of an elastic manipulator," International Journal of Robotics and Automation, vol. 6, no. 4, p. 179, 1991.*
- MATLAB version 4.2, July 1994, The Mathworks, Inc., 24 Prime Park Way, Natick, MA 01760, USA.*
- McClamroch, N.H., and D. Wang, "Feedback stabilization and tracking of constrained robots," IEEE Trans. Automat. Control, vol. 33, p. 419-426, 1988.*
- Narendra, K.S., and A.M. Annaswamy, "A new adaptive law for robust adaptation without persistent excitation," IEEE Trans. Automat. Control, vol. AC-32, no. 2, pp. 134-145, Feb. 1987.*
- Qian, W.T., and C.C.H. Ma, "A new controller design for a flexible one-link manipulator," IEEE Trans. Automat. Control, vol. 37, no. 1., pp. 132-137, Jan. 1992.*

- Siciliano, B., and W. Book, "A singular perturbation approach to control of lightweight manipulators," Int. J. Robotics Research, vol. 7, no. 4, pp. 79-90, Aug. 1988.*
- Slotine, J.-J.E., and W. Li, Applied Nonlinear Control, Prentice-Hall, New Jersey, 1991.*
- Spong, M.W., "Adaptive control of flexible joint manipulators," Systems and Control Letters, vol. 13, pp. 15-21, 1989.*
- Spong, M.W., and M. Vidyasagar, Robot Dynamics and Control, Wiley, New York, 1989.*
- Vandegrift, M., F.L. Lewis, and S. Zhu, "Flexible-link robot arm control by a feedback linearization/singular perturbation approach," J. Robotic Systems, vol. 11, no. 7, pp. 591-603, 1994.*
- Wang, D., and M. Vidyasagar, "Transfer functions for a single link flexible link," Int. J. Robotics Research, vol. 10, no. 5, Oct. 1991.*
- Wang, D., and M. Vidyasagar, "Control of a class of manipulators with a single flexible link - Part I: feedback linearization," Trans. ASME, J. Dynam. Sys., Meas., and Control, vol. 113, pp. 655-661, Dec. 1991.*
- Yeşildirek, A., M.W. Vandegrift, and F.L. Lewis, "A neural net controller for flexible-link robots," J. Intelligent and Robotic Systems, vol. 17, pp. 327-349, 1996.*
- Zhu, S.Q., F.L. Lewis, and L.R. Hunt, "Robust stabilization of the internal dynamics of flexible robots without measuring the velocity of the deflection," Proc. IEEE Conf. Decision and Control, pp. 1811-1816, Dec. 1994.*

## 5.7 PROBLEMS

### Section 5.1

**Problem 5.1-1 : Reduced-Order Dynamics on Constraint Surface.** Derive the reduced-order dynamics (5.1.12) and show that it satisfies the properties in Table 5.0.1.

**Problem 5.1-2 : Constraint Surface Jacobian Multiplication Property.** Prove the Jacobian multiplication property (5.1.13).

**Problem 5.1-3 : Closed-Loop Error Dynamics.** (a) Derive the closed-loop error dynamics (5.1.17) and function  $f(x)$ . (b) Derive the form of the closed-loop error dynamics given in (5.1.25).

**Problem 5.1-4 : Simulation of NN Hybrid Position/Force Controller.** Perform the simulation of Example 5.1.1 using MATLAB. Simulation is discussed in Chapter 3. Include friction terms of the form given in Chapter 3.

**Problem 5.1-5 : Constraint Formulation.** One notes from Example 5.1.1 that, technically, the constrained motion variable  $q_1(t)$  is not tangential to the surface  $\phi(y) = 0$ . Reformulate the constraint dynamics in Section 5.1.1 so that  $q_1(t)$

*describes true motion tangential to the surface. You will need to introduce an additional transformation  $q_1 = \psi(q)$  that transforms to the tangent plane. Redo Example 5.1.1 using the new formulation.*

## Section 5.2

**Problem 5.2-1 : Simulation of Flexible-Link Arm.** *Perform the MATLAB simulation in Example 5.2.1.*

**Problem 5.2-2 : Rigid Joint Actuator/Arm Dynamics.** *Derive the dynamics in equation (5.2.13).*

**Problem 5.2-3 : Simulation of DC Motor with Flexible Coupling Shaft.** *Perform the MATLAB simulation in Example 5.2.2.*

**Problem 5.2-4 : Dynamics of Robot Arm with Electrical and Mechanical Actuator Dynamics and Joint Flexibility.** *Derive the full dynamics of a robot arm including electrical and mechanical actuator dynamics as well as joint flexibility. Draw a block diagram like the one in Fig. 5.2.4.*

**Problem 5.2-5 : Dynamics of Robot Arm with Both Link and Joint Flexibility.** *(a) Derive the dynamics of a robot arm having both link flexibility and joint flexibility. Draw a block diagram like the one in Fig. 5.2.4. (b) Perform a MATLAB simulation of the system. Use parameters taken from Examples 5.2.1 and 5.2.2.*

## Section 5.3

**Problem 5.3-1 : Linear System Singular Perturbation Control.** *The longitudinal dynamics of an F-16 aircraft in straight and level flight at 502 ft/sec are given by*

$$\begin{aligned} \dot{x} = Ax + Bu &= \begin{bmatrix} -2.0244E-2 & 7.8761 & -3.2169E+1 & -6.502E-1 \\ -2.5373E-4 & -1.0189 & 0 & 9.0484E-1 \\ 0 & 0 & 0 & 1 \\ 7.9472E-11 & -2.498 & 0 & -1.3861 \end{bmatrix} x \\ &+ \begin{bmatrix} -1.E-2 \\ -2.09E-3 \\ 0 \\ -1.99E-1 \end{bmatrix} u. \end{aligned}$$

The state is  $x = [v_T \ \alpha \ \theta \ q]^T$ , with  $v_T$  the forward velocity,  $\alpha$  the angle of attack,  $\theta$  the pitch angle, and  $q$  the pitch rate. The control input  $u(t)$  is the elevator deflection  $\delta_e$ . (a) Find the open-loop poles. The slow pole pair is known as the phugoid mode and the fast pair as the short period mode. (b) Select slow variables  $v_T, \theta$  and fast variables  $\alpha, q$  and determine the slow/fast decomposition (5.3.16)-(5.3.18). What is a good value for  $\varepsilon$ ? Find the poles of the slow and fast subsystems. (c) Plot the response of the open-loop system to initial conditions of  $x_0 = [0 \ 0.1 \ 0 \ 1]^T$  (note—the angular units are in radians). (d) Try MATLAB function `lqr` to perform two linear quadratic regulator designs, one for the slow subsystem and one for the fast subsystem. Use  $Q = I, R = 0.1$ . Simulate the closed-loop response for  $x_0$  using composite control. Use two values of  $\varepsilon$ , one significantly smaller than the other.

**Problem 5.3-2 : Derivation of Flexible-Link Slow Dynamics.** *Derive the slow dynamics (5.3.27).*

**Problem 5.3-3 : Simulation of Flexible-Link NN Controller.** Perform the simulation in Example 5.3.1. Add friction. Compare the performance using  $\varepsilon$  values of 0.26, 0.1, and 0.01.

**Problem 5.3-4 : Singular Perturbations Design for Flexible-Joint Robot.** Perform a singular perturbations control design for the flexible-joint robot (5.2.15)-(5.2.16). Define the fast variable as  $\xi = q - q_M$  and assume that the coupling stiffness matrix  $K_s$  is large.

**Problem 5.3-5 : Singular Perturbations Design for Electrically-Driven Manipulator.** Perform a singular perturbations control design for the flexible-joint robot (5.2.17)-(5.2.18). Define the fast variable as the armature constant  $i$ .

#### Section 5.4

**Problem 5.4-1 : Backstepping Tracker Design for Flexible-Joint Type Systems.** Derive the backstepping controller in Subsection 5.4.1 if  $x_1(t)$  is required not to be zero, but to track a desired trajectory  $x_{1_d}(t)$ .

**Problem 5.4-2 : Backstepping Tracker Design for Flexible-Link Type Systems.** Now suppose that the output is selected as  $x_2(t)$  in (5.4.1)-(5.4.2), and that it is required to follow a desired trajectory  $x_{2_d}(t)$ . Can you derive a backstepping controller that stabilizes the internal dynamics  $x_1(t)$  and ensures tracking by  $x_2(t)$ ?

**Problem 5.4-3 : Relaxation of Linear-in- $\dot{x}_2$  Assumption.** Instead of Assumption 5.4.2, derive a backstepping control  $u(t)$  using the milder Assumption 5.4.4. You will need some robust control derivation techniques using norms (See the discussion on uniform ultimate boundedness Chapter 2 and that on robust control in Chapter 3).

**Problem 5.4-4 : Backstepping Design Examples.** Use backstepping to stabilize the following systems.

a. 
$$\begin{aligned}\dot{x}_1 &= \cos x_1 + (2 + \sin x_1)x_2 \\ \dot{x}_2 &= \sin x_1 + \cos^2 x_2 + (2 + \sin x_1)u\end{aligned}$$

b. 
$$\begin{aligned}\ddot{x}_1 + \dot{x}_1 &= 5x_2 \\ \dot{x}_2 &= x_1x_2 + (1 + x_1^2)u\end{aligned}$$

c. 
$$\begin{aligned}\dot{x}_1 &= x_1^2 + x_2 \\ \dot{x}_2 &= x_1x_2 + x_3 \\ \dot{x}_3 &= x_1x_2 + (1 + x_1^2)u\end{aligned}$$

**Problem 5.4-5 : Backstepping Tracker Design.** Use backstepping to stabilize the systems in the previous example with tracking of  $x_{1_d} = 2 \sin(2\pi/T)$ .

**Problem 5.4-6 : Backstepping Design for Flexible-Joint Robots.** Use backstepping to stabilize the flexible-joint robot arm of equations (5.2.15)-(5.2.16). You will effectively be doing two rigid-robot designs in steps one and two.



## Chapter 6

# Neural Network Control of Nonlinear Systems

*In Chapter 4 were given several techniques for the design of neural network (NN) controllers for rigid-link robotic systems. In Chapter 5 were given NN controllers for complex practical robotic systems including force control, flexible-link robots, flexible-joint systems, and systems with high-frequency actuator dynamics. The NN controllers relied on a filtered-error approximation-based approach, and the weight tuning algorithms included a simple backpropagation-type method that works in an ideal case, and modified tuning algorithms that work in general cases that include disturbances. Two sorts of NN were considered—functional-link NN (FLNN) that are linear in the tunable weights, and two-layer NN that are nonlinear in the first-layer weights  $V$ . It was shown how to overcome the linear-in-the-parameters (LIP) restriction of standard adaptive control approaches.*

*In this chapter are designed NN controllers for a large class of nonlinear systems that include robot arms and other sorts of dynamical systems, namely, those in the Brunovsky canonical form described in Chapter 2. Robot arms have special properties including passivity, skew-symmetry, and bounded nonlinearities that make them convenient to analyze. In this chapter we show how to design intelligent controllers for the larger class of Brunovsky form systems.*

*The work in this chapter was done in Yeşildirek (1994) and Yeşildirek and Lewis (1985). The approach is based on feedback linearization, discussed in Chapter 2. Feedback linearization techniques require the system to be controllable, and the nonlinearities must satisfy some conditions to guarantee that a solution to the closed-loop system exists. Feedback linearization is focused around geometric techniques. However applicability of these approaches to feedback control of actual systems is quite limited because they rely on exact knowledge of system nonlinearities. In order to relax some of the exact model-matching restrictions, several adaptive schemes have been introduced that tolerate some linear parametric uncertainties (Campion and Bastin 1990, Nam and Arapostathis 1988, Taylor et al. 1989, Teel et al. 1991). In this chapter linearity in the system parameters is not required as NN are used to learn the unknown nonlinearities based on the universal approximation property.*

*Even if the closed-loop system is stable, it must be shown that all inputs, outputs,*

and states remain bounded. For example, if the controller is given in the form of  $u = \frac{N(x)}{D(x)}$  then  $D(x)$  must be non-zero for all time—we will call this type of controller a well-defined controller. Unfortunately, for feedback linearization this type of controller structure is usually needed. If an adaptive scheme is employed to approximate the denominator part of the controller using  $\hat{D}(x)$  an estimate for  $D(x)$ , then extra precautions are required to guarantee that  $\hat{D} \neq 0$  for all time. This problem is far from trivial and because of it existing solutions are usually given locally and/or assume additional prior knowledge about the system, such as LIP. As expected, the same difficulties appear in NN control systems, which can be categorized as nonlinear-in-the-parameter adaptive systems. We confront the full problem of feedback linearization for a class of nonlinear systems in Section 6.3. First, we solve a simpler design problem in Section 6.2 to provide insight.

In this chapter we assume all the system states are measurable. If only some states are measurable, corresponding to the case of output feedback, then an additional dynamical NN is required to estimate the unmeasured states (Kim and Lewis 1996).

## 6.1 SYSTEM AND TRACKING ERROR DYNAMICS

Consider any state-feedback linearizable system having a state-space representation in the Brunovsky canonical form (BCF)

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= x_3 \\ &\vdots \\ \dot{x}_n &= f(\mathbf{x}) + g(\mathbf{x})u + d \\ y &= x_1\end{aligned}\tag{6.1.1}$$

with state  $\mathbf{x}(t) = [x_1 \ x_2 \dots x_n]^T$ ,  $d(t)$  an unknown disturbance, and  $f, g : R^n \rightarrow R$  unknown smooth functions. The following assumption is made.

**Assumption 6.1.1 (Bounds on Disturbance and Unknown Function  $g(\mathbf{x})$ ) :**

- a. There is a known upper bound  $b_d$  so that

$$\|d(t)\| \leq b_d, \text{ for all } t.\tag{6.1.2}$$

- b. The unknown function  $g(\mathbf{x})$  is bounded away from zero so that

$$|g(\mathbf{x})| \geq \underline{g} > 0, \quad \forall \mathbf{x},\tag{6.1.3}$$

with  $\underline{g}$  a known lower bound.

The assumption (6.1.3) on the smooth function  $g$  implies that  $g$  is strictly either positive or negative for all  $\mathbf{x}(t)$ . Thus, the sign of  $g(\mathbf{x})$  must be known. Note that at this point there is no general approach to analyze this class of unknown nonlinear systems. Adaptive control, for instance, needs an additional linear-in-the-parameters assumption.

Although we treat only single-input-single-output (SISO) systems here, the approach in this chapter extends to the multivariable case. The multivariable BCF

was discussed in Chapter 2. According to the discussion on state-space forms for robot arms in Chapter 3,  $N$ -link rigid robots are in this class of systems, with the exception that scalar  $x_i$  is replaced by vector  $x_i \in R^N$ , and  $g(\mathbf{x})$  is an invertible matrix for all  $\mathbf{x}$ . Subsequent developments easily accommodate this extension.

In this chapter we show how to develop NN controllers for systems in Brunovsky form. General nonlinear systems occur in the form

$$\begin{aligned}\dot{x} &= F(x, u) \\ y &= H(x, u).\end{aligned}\quad (6.1.4)$$

To transform this to the Brunovsky form, the system can be feedback linearized if it satisfies a reachability condition and an involutivity condition (Slotine and Li 1991, Isidori 1989). If these conditions are satisfied, then there exist a state-space transformation that puts the system in Brunovsky form. It is not always easy to find this transformation. In recent work by Zhang et al. (1998) it is shown how to use the NN approximation properties to effectively estimate this transformation, so that NN controllers can be designed for a large class of nonlinear systems that are more general than Brunovsky form.

The reachability condition is usually satisfied for practical systems, but some classes of systems fail to be involutive. If the involutivity condition fails to hold, then it is still possible to perform partial feedback linearization, or input-output feedback linearization. In this case, there may exist some zero dynamics that must be dealt with (Chapter 2). A certain pathological condition may also hold with regard to the relative degree of the system. In this ‘ill-defined relative degree’ situation, even greater care is needed and ‘almost’ feedback linearization techniques are needed (Hauser et al. 1992).

### 6.1.1 Tracking Controller and Error Dynamics

Feedback linearization will be used to perform output tracking, whose objective can be described as the following: given a desired output,  $y_d(t)$ , find a control action,  $u(t)$ , so that  $y(t) = x_1(t)$  follows the desired trajectory with an acceptable accuracy (i.e. bounded-error tracking) while all the states and controls remain bounded. To design a tracking controller we will make some mild assumptions which are widely used. Define the desired trajectory vector as

$$\mathbf{x}_d(t) \equiv [y_d \ \dot{y}_d \dots y_d^{(n-1)}]^T. \quad (6.1.5)$$

**Assumption 6.1.2 (Bounded Desired Trajectory)** : The desired trajectory vector  $\mathbf{x}_d(t)$  is continuous, available for measurement, and

$$\|\mathbf{x}_d(t)\| \leq Q \quad (6.1.6)$$

with  $Q$  a known bound.

Define a state error vector as

$$\mathbf{e} = \mathbf{x} - \mathbf{x}_d \quad (6.1.7)$$

and a filtered tracking error as

$$r = \Lambda^T \mathbf{e} \quad (6.1.8)$$

where  $\Lambda = [\lambda_1 \ \lambda_2 \cdots \lambda_{n-1} \ 1]^T$  is an appropriately chosen coefficient vector so that  $\mathbf{e} \rightarrow \mathbf{0}$  exponentially as  $r \rightarrow 0$ , (i.e.  $s^{n-1} + \lambda_{n-1}s^{n-2} + \cdots + \lambda_1$  is asymptotically stable). Note that the tracking error is defined differently than in previous chapters where it was equal to  $e(t) = q_d(t) - q(t)$ . This makes for some sign changes in subsequent work as compared to previous chapters.

The time derivative of the filtered error can be written as

$$\dot{r} = f(\mathbf{x}) + g(\mathbf{x})u + d + Y_d \quad (6.1.9)$$

where

$$Y_d \equiv -x_d^{(n)} + \sum_{i=1}^{n-1} \lambda_i e_{i+1} = -x_d^{(n)} + [0 \ \bar{\Lambda}^T] \mathbf{e} \quad (6.1.10)$$

is a known signal. We have defined  $e_{i+1} = y^{(i)} - y_d^{(i)}$  for  $i = 1, 2, \dots, n-1$  and  $\bar{\Lambda} = [\lambda_1 \ \lambda_2 \cdots \lambda_{n-1}]^T$ .

We will use the dynamics in the form of (6.1.9) to construct a controller using NN that keeps  $r(t)$  bounded. Then, since (6.1.8) is a stable system,  $e(t)$  is bounded, thus, our tracking performance is achieved.

If we knew the exact form of the nonlinear functions then the feedback linearization control action

$$u_{exact} = \frac{1}{g(\mathbf{x})} [-f(\mathbf{x}) - K_v r - Y_d] \quad (6.1.11)$$

would be appropriate for any positive  $K_v$ . Since we assume that  $f(x)$  and  $g(x)$  are not exactly known we will define a control action

$$u_c = \frac{1}{\hat{g}(\hat{\Theta}_g, \mathbf{x})} [-\hat{f}(\hat{\Theta}_f, \mathbf{x}) + v] \quad (6.1.12)$$

where the estimates  $\hat{f}(\hat{\Theta}_f, \mathbf{x})$  and  $\hat{g}(\hat{\Theta}_g, \mathbf{x})$  will be constructed by two neural networks with respective weights  $\Theta_f, \Theta_g$ , and the auxiliary term is

$$v = -K_v r - Y_d. \quad (6.1.13)$$

It is well known, even in adaptive control of linear systems, that guaranteeing boundedness of  $\hat{g}$  away from zero becomes an important issue in this type of controller, as discussed subsequently.

To find the closed-loop error dynamics, add zero to (6.1.9) to get

$$\begin{aligned} \dot{r} &= f + gu + d + Y_d + \hat{g}u_c - \hat{g}u_c + gu_c - gu_c \\ \dot{r} &= f + d + Y_d - \hat{f} - K_v r - Y_d + (g - \hat{g})u_c + g(u - u_c) \\ \dot{r} &= -K_v r + (f - \hat{f}) + (g - \hat{g})u_c + d + gu_d \\ \dot{r} &= -K_v r + \tilde{f} + \tilde{g}u_c + d + gu_d \end{aligned} \quad (6.1.14)$$

where  $u_d \equiv u - u_c$  and the functional approximation errors are

$$\tilde{f} = f - \hat{f} \quad \tilde{g} = g - \hat{g}. \quad (6.1.15)$$

### 6.1.2 Well-Defined Control Problem

In general, boundedness of  $\mathbf{x}$ ,  $\hat{\Theta}_f$ , and  $\hat{\Theta}_g$  does not indicate the stability of the closed-loop system, because control law (6.1.12) is not well-defined when  $\hat{g}(\hat{\Theta}_g, \mathbf{x}) = 0$ . Therefore some attention must be taken to guarantee the boundedness of the controller as well. There are some techniques in the literature which assure local stability or global stability in the presence of additional knowledge. First, if the bounds on  $g$  are known then  $\hat{g}$  may be set to a constant and a robust-adaptive controller bypasses the  $\hat{g} = 0$  problem. This is not an accurate approximation and may not give controllers that perform well.

If  $g$  is reconstructed by an adaptive scheme, most of which require linearity-in-the-parameters, then a local solution to this problem can be given by assuming that initial estimates are close to the actual values and they do not leave a feasible invariant set in which  $\hat{g} \neq 0$  (Liu and Chen 1993), or they lie inside a region of attraction of a stable equilibrium point which forms a feasible set as well (Kanellakopoulos et al. 1991). Unfortunately, even with very good knowledge of the system it is not easy to pick initial weights so that the NN approximates the nonlinearity  $g(\cdot)$ .

Another way to keep  $\hat{g}(\hat{\Theta}_g, \mathbf{x})$  away from zero is to project  $\hat{\Theta}_g$  inside an estimated feasible region through the weight adaptation law (Polycarpou and Ioannou 1991). Assume that there exists an open set  $B_\theta$  such that  $\hat{g}(\hat{\Theta}_g, \mathbf{x}) \neq 0$  for all  $\hat{\Theta}_g \in B_\theta$ ,  $\mathbf{x} \in R^n$ . In addition, it is known that the actual target weights  $\Theta_g$  (that yield good approximation of  $g(\cdot)$ ) belong to  $B_\theta$ . Then a global solution may be found as follows. Whenever  $\hat{\Theta}_g$  is on the boundary of  $B_\theta$ , if it is projected inside the set by changing the adaptation rule then we obtain a well-defined control action. Since the number of parameters of NN may be big, the difficulty of finding such an estimate region becomes obvious. A candidate set for this reason is shown in Polycarpou and Ioannou (1991) as  $B_\theta = \left\{ \hat{\Theta}_g : \hat{\theta}_{ij} > 0, \forall i, j \right\}$  with gaussian or sigmoid type activation function. This results in a feasible region in which  $\hat{g} \neq 0$ . A shortcoming of this estimate region is that the actual  $\Theta_g$  does not necessarily belong to such a set, i.e. this is a suboptimal solution, in general.

In this chapter we will suggest a new controller structure which takes into account this problem without such tight assumptions and allows a simple method of initializing the NN weights.

## 6.2 CASE OF KNOWN FUNCTION $g(\mathbf{x})$

In this section we assume that  $f(\mathbf{x})$  is unknown and  $g(\mathbf{x})$  is known and design a tracking controller; things are considerably simplified when  $g(\mathbf{x})$  is known. The main result of this chapter appears in Section 6.3 where we present a tracking controller for the case of unknown  $g(\mathbf{x})$  and  $f(\mathbf{x})$ . In this section one neural net is required for approximation of the unknown function  $f(\mathbf{x})$ . In Section 6.3 two NN are required—for approximation of the unknown functions  $f(\mathbf{x})$  and  $g(\mathbf{x})$ .

### 6.2.1 Proposed NN Controller

#### 6.2.1.1 Neural Network for Approximating $f$

Let a neural network be used to approximate the unknown continuous function  $f(\cdot)$ . Therefore, on a compact set of  $\Re^n$  there exist ideal target weights so that

$$f(\mathbf{x}) = W^T \sigma(V^T \mathbf{x}) + \varepsilon \quad (6.2.1)$$

where the functional estimation error is bounded so that

$$\|\varepsilon\| < \varepsilon_N \quad (6.2.2)$$

for a known constant  $\varepsilon_N$ . Define the weight matrix

$$\Theta = \begin{bmatrix} V & 0 \\ 0 & W \end{bmatrix}. \quad (6.2.3)$$

The ideal weights are unknown and possibly nonunique, but they satisfy the following assumption.

**Assumption 6.2.1 (Bounded NN Target Weights)** : The ideal NN weights for the continuous functions  $f(x)$  are bounded in any compact subset of  $\Re^n$  according to

$$\|\Theta\| \leq \Theta_m \quad (6.2.4)$$

with  $\theta_m$  known.

Estimates for the nonlinear function  $f(\mathbf{x})$  are now given by the NN as

$$\hat{f}(\mathbf{x}) = \hat{W}^T \sigma(\hat{V}^T \mathbf{x}) \quad (6.2.5)$$

with ‘hat’ denoting the actual values of the NN weights as provided by the tuning algorithms. Using a Taylor Series expansion exactly as in Chapter 4 one may write the functional estimation error as

$$\tilde{f}(\mathbf{x}) = \tilde{W}^T (\hat{\sigma} - \hat{\sigma}' \hat{V}^T \mathbf{x}) + \hat{W}^T \hat{\sigma}' \tilde{V}^T \mathbf{x} + w \quad (6.2.6)$$

where the higher-order terms are bounded according to

$$\|w(t)\| \leq C_0 + C_1 \|\tilde{\Theta}\|_F + C_2 \|r\| \cdot \|\tilde{\Theta}\|_F \quad (6.2.7)$$

for computable positive constants  $C_0, C_1, C_2$ .

#### 6.2.1.2 Controller Structure

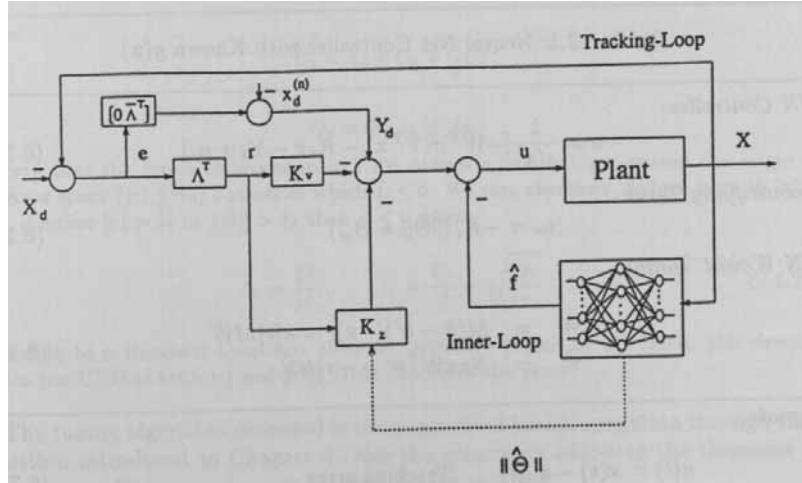
Since  $g(\mathbf{x})$  is known, based on (6.1.12) we select the control as

$$\begin{aligned} u = u_c &= \frac{1}{g(\mathbf{x})} [-\hat{f}(\mathbf{x}) - K_v r - Y_d + u_r] \\ &= \frac{1}{g(\mathbf{x})} [-\hat{W}^T \sigma(\hat{V}^T \mathbf{x}) - K_v r - Y_d + u_r] \end{aligned} \quad (6.2.8)$$

for any  $K_v > 0$ . The auxiliary robustifying term  $u_r(t)$  will be detailed later. This controller is shown in Fig. 6.2.1.

Substituting this control into the filtered error dynamics (6.1.14) yields

$$\begin{aligned} \dot{r} &= -K_v r + \tilde{f} + u_r + d \\ \dot{r} &= -K_v r + \tilde{W}^T (\hat{\sigma} - \hat{\sigma}' \hat{V}^T \mathbf{x}) + \hat{W}^T \hat{\sigma}' \tilde{V}^T \mathbf{x} + d + w + u_r. \end{aligned} \quad (6.2.9)$$

Figure 6.2.1: Neural network controller with known  $g(\mathbf{x})$ .

### 6.2.2 NN Weight Tuning for Tracking Stability

The next theorem shows how to tune the weights in the NN so that tracking performance and internal stability are guaranteed. The resulting controller is shown in Table 6.2.1.

**Theorem 6.2.1 (NN Tracking Controller for Unknown  $f$  and Known  $g$ ) :**

Assume that the system has a representation in the Brunovsky canonical form with  $g(\mathbf{x})$  known. Let the control input be given by (6.2.8) with robustifying term

$$u_r = -K_z(\|\hat{\Theta}\| + \Theta_m) \quad (6.2.16)$$

with  $K_z > C_2 > 0$ . Let the neural net weight update laws be provided by

$$\begin{aligned} \dot{\hat{W}} &= M(\hat{\sigma} - \hat{\sigma}'\hat{V}^T\mathbf{x})r - \kappa|r|M\hat{W} \\ \dot{\hat{V}} &= N\mathbf{r}\mathbf{x}\hat{W}^T\hat{\sigma}' - \kappa|r|N\hat{V} \end{aligned} \quad (6.2.17)$$

where  $M$  and  $N$  are positive definite matrices and design parameter  $\kappa > 0$ . Then the filtered tracking error  $r(t)$  and neural net weight error  $\hat{\Theta}$  are UUB with specific bounds giving by (6.2.20). Moreover, the tracking error may be kept as small as desired by increasing the PD gains  $K_v$ .

Proof:

Let the Lyapunov function candidate be

$$L = \frac{1}{2}r^2 + \frac{1}{2}tr\{\tilde{W}^T M^{-1} \tilde{W}\} + \frac{1}{2}tr\{\tilde{V}^T N^{-1} \tilde{V}\}. \quad (6.2.18)$$

Differentiate, substitute for  $\dot{r}$  from (6.2.9) and perform a simple manipulation, (i.e. using the equality  $\mathbf{x}^T \mathbf{y} = tr\{\mathbf{x}^T \mathbf{y}\} = tr\{\mathbf{y} \mathbf{x}^T\}$ , one can place weight matrices inside a trace operator) to obtain

$$\begin{aligned} \dot{L} &= -K_v r^2 + tr\left\{\tilde{W}^T (\hat{\sigma} - \hat{\sigma}'\hat{V}^T\mathbf{x})r + M^{-1}\dot{\tilde{W}}\right\} \\ &\quad + tr\left\{\tilde{V}^T (\mathbf{x}r\hat{W}^T\hat{\sigma}' + N^{-1}\dot{\tilde{V}})\right\} + r(d + w + u_r). \end{aligned}$$

Table 6.2.1: Neural Net Controller with Known  $g(\mathbf{x})$ *NN Controller:*

$$u = \frac{1}{g(\mathbf{x})} [-\hat{W}^T \sigma(\hat{V}^T \mathbf{x}) - K_v r - Y_d + u_r] \quad (6.2.10)$$

*Robustifying Term:*

$$u_r = -K_z (\|\hat{\Theta}\| + \Theta_m) \quad (6.2.11)$$

*NN Weight Tuning:*

$$\begin{aligned} \dot{\hat{W}} &= M(\hat{\sigma} - \hat{\sigma}' \hat{V}^T \mathbf{x}) r - \kappa |r| M \hat{W} \\ \dot{\hat{V}} &= N r \mathbf{x} \hat{W}^T \hat{\sigma}' - \kappa |r| N \hat{V}. \end{aligned} \quad (6.2.12)$$

*Signals:*

$$\mathbf{e}(t) = \mathbf{x}(\mathbf{t}) - \mathbf{x}_d(t) \quad \text{Tracking error} \quad (6.2.13)$$

$$r(t) = \Lambda^T \mathbf{e}(t) \quad \text{Filtered tracking error} \quad (6.2.14)$$

$$Y_d = -x_d^{(n)} + [0 \ \bar{\Lambda}^T] \mathbf{e} \quad \text{Desired trajectory feedforward signal} \quad (6.2.15)$$

*Design Parameters:**Gains  $K_v, K_z$  positive* *$\Lambda$  a coefficient vector of a Hurwitz function.* *$\Theta_m$  a bound on the unknown target weight norms.**Tuning matrices  $M, N$  symmetric and positive definite.**Scalar  $\kappa > 0$ .*

With the update rules given in (6.2.17) one has

$$\dot{L} = -K_v r^2 + r(d + w + u_r) + \kappa |r| \operatorname{tr}\{\tilde{\Theta}^T \hat{\Theta}\}.$$

From the inequality

$$\operatorname{tr}\{\tilde{\Theta}^T \hat{\Theta}\} = \langle \tilde{\Theta}^T, \hat{\Theta} \rangle - \operatorname{tr}\{\tilde{\Theta}^T \tilde{\Theta}\} \leq \|\tilde{\Theta}\|(\Theta_m - \|\tilde{\Theta}\|),$$

it follows that

$$\dot{L} \leq -K_v r^2 + r(d + w + u_r) + \kappa |r| \|\tilde{\Theta}\|(\Theta_m - \|\tilde{\Theta}\|).$$

Substitute the upper bound of  $w$  according to (6.2.7), bound  $b_d$  for disturbances, and  $u_r$  from (6.2.16) to yield

$$\begin{aligned} \dot{L} &\leq -K_v r^2 - K_z (\|\hat{\Theta}\| + \Theta_m) r^2 + \kappa |r| \|\tilde{\Theta}\| (\Theta_m - \|\tilde{\Theta}\|) \\ &\quad + [C_2 \|\tilde{\Theta}\| |r| + C_1 \|\tilde{\Theta}\| + (b_d + C_0)] |r|. \end{aligned}$$

Picking  $K_z > C_2$  and completing the squares yields

$$\dot{L} \leq -|r| \{ K_v |r| + \kappa (\|\tilde{\Theta}\| - C_3/2)^2 - D_1 \} \quad (6.2.19)$$

where

$$D_1 = b_d + C_0 + \frac{\kappa}{4} C_3^2,$$

and

$$C_3 = \Theta_m + C_1/\kappa.$$

Observe that the terms in braces in (6.2.19) defines a compact set around the origin of the error space  $(|r|, \|\tilde{\Theta}\|)$  outside of which  $\dot{L} \leq 0$ . We can, therefore, deduce from (6.2.19) that, if either  $|r| > \delta_r$  or  $\|\tilde{\Theta}\| > \delta_f$  then  $\dot{L} \leq 0$  where

$$\delta_r = \frac{D_1}{K_v}, \quad \delta_f = \frac{C_3}{2} + \sqrt{\frac{D_1}{\kappa}}. \quad (6.2.20)$$

According to a standard Lyapunov theorem extension (Lewis *et al.* 1993), this demonstrates the UUB of both  $|r|$  and  $\|\tilde{\Theta}\|$ . This concludes the proof.  $\square$

*The tuning algorithm proposed is the augmented backpropagation through time algorithm introduced in Chapter 4. See the comments following the theorems in that chapter. The next remarks are particularly relevant.*

*Remarks:*

1. *For practical purposes, (6.2.20) can be considered as bounds on  $|r|$  and  $\|\tilde{\Theta}\|$  in the sense that excursions above these bounds will be small.*
2. *The NN reconstruction construction error bound  $\varepsilon_N$  (which depends on the number of hidden-layer neurons), the disturbance bound  $b_d$ , and the bound  $Q$  on the desired trajectory are all involved in the constants  $C_i$  which contribute to definition of  $\delta_r$ ,  $\delta_f$ . Note from the definitions of  $\delta_r$  that the bound on the tracking error may be kept arbitrarily small by selecting the control gain  $K_v$  large enough.*
3. *Note the role of the design parameter  $\kappa$ ; the larger  $\kappa$ , the smaller the bound on parameter errors and the larger the bound on the tracking error.*
4. *The adaptation laws are derived from the Lyapunov approach. It turns out that the first terms in the adaptation laws have the same structure as backpropagation through time terms, but show that the quantity to be backpropagated is the filtered tracking error  $r(t)$ . Moreover, the required Jacobian  $\dot{\sigma}'$  is easily computed in terms of measurable signals in the closed-loop system. In the absence of persistence of excitation (see Chapter 4) and with noise and/or unmodeled dynamics we suggest an extra  $e$ -modification term (Narendra and Annaswamy 1987) to allow proof of UUB (i.e. the last terms). Moreover, we introduce a novel feedforward propagating term in the tuning algorithm for  $\hat{W}$  to cancel out some of the higher-order terms coming from the Taylor series expansion.*
5. *As in Chapter 4, the initial NN weights may be set to zero, for then according to Fig. 6.2.1 the controller consists of a PD tracking loop which holds the system bounded stable until the NN begins to learn, at which point the tracking performance will improve.*

### 6.2.3 Illustrative Simulation Example

**Example 6.2.1 (Van der Pol's System) :**

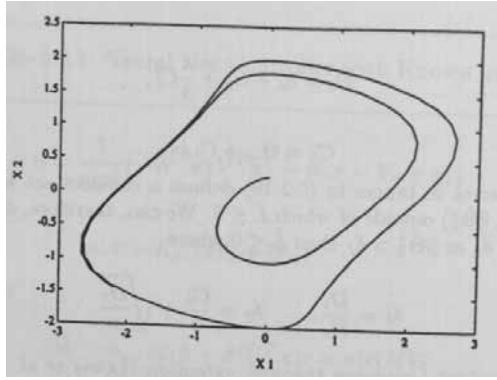
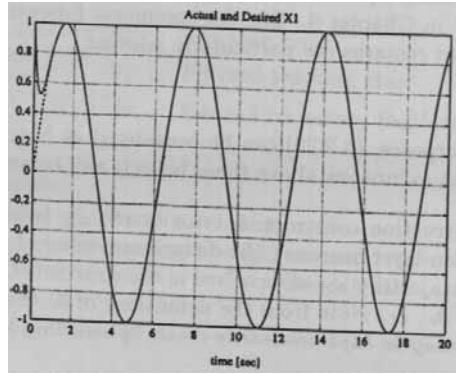


Figure 6.2.2: Open-loop state trajectory of the Van der Pol's system.

Figure 6.2.3: Actual and desired state  $x_1$ .

Let us illustrate the NN controller design on a Van der Pol's system

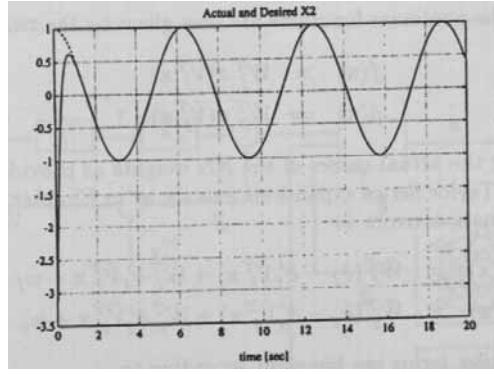
$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= (1 - x_1^2)x_2 - x_1 + u\end{aligned}\tag{6.2.21}$$

which is in the Brunovsky canonical form. Note that this system has an unstable equilibrium point at the origin  $\mathbf{x} = (0, 0)$  and a stable limit cycle. A typical open-loop trajectory for this system is illustrated in Fig. 6.2.2.

The neural net which is used for estimation of  $f(x_1, x_2) = (1 - x_1^2)x_2 - x_1$  consists of 10 neurons. Design parameters are set to  $K_v = 20$ ,  $\Lambda = 5$ ,  $K_z = 10$ ,  $\Theta_m = 1$ ,  $M = N = 20$ , and  $\kappa = 1$ . Initial conditions are  $\hat{\Theta}(0) = 0$  and  $x_1 = x_2 = 1$ . The desired trajectory is defined as  $y_d(t) = \sin t$ . Actual and desired outputs are shown in Figs. 6.2.3 and 6.2.4. The closed-loop tracking performance is very good even though  $f(\mathbf{x})$  is unknown.  $\square$

### 6.3 CASE OF UNKNOWN FUNCTION $g(\mathbf{x})$

*In this section we assume that both  $f(\mathbf{x})$  and  $g(\mathbf{x})$  are unknown and provide the main result of this chapter. In this section two NN are required, one for approximation*

Figure 6.2.4: Actual and desired state  $x_2$ .

of  $f(\mathbf{x})$  and one for approximation of  $g(\mathbf{x})$ .

### 6.3.1 Proposed NN Controller

#### 6.3.1.1 Neural Networks For Approximating $f$ and $g$

Let two neural networks be used to approximate the unknown continuous functions. Therefore, on a compact set of  $\Re^n$  there exist ideal target weights so that

$$\begin{aligned} f(\mathbf{x}) &= W_f^T \sigma(V_f^T \mathbf{x}) + \varepsilon_f \\ g(\mathbf{x}) &= W_g^T \sigma(V_g^T \mathbf{x}) + \varepsilon_g, \end{aligned} \quad (6.3.1)$$

where the functional estimation errors are bounded so that

$$\|\varepsilon_f\| < \varepsilon_{fN}, \quad \|\varepsilon_g\| < \varepsilon_{gN} \quad (6.3.2)$$

for known constants  $\varepsilon_{fN}, \varepsilon_{gN}$ . Define the weight matrices

$$\Theta_f = \begin{bmatrix} V_f & 0 \\ 0 & W_f \end{bmatrix}, \quad \Theta_g = \begin{bmatrix} V_g & 0 \\ 0 & W_g \end{bmatrix}. \quad (6.3.3)$$

The ideal weights are unknown and possibly nonunique, but they satisfy the following assumption.

**Assumption 6.3.1 (Bounded NN Target Weights)** : The ideal NN weights for the continuous functions  $f(x)$  and  $g(x)$  are bounded in any compact subset of  $\Re^n$  according to

$$\|\Theta_f\| \leq \Theta_{fm} \quad (6.3.4)$$

and

$$\|\Theta_g\| \leq \Theta_{gm} \quad (6.3.5)$$

with  $\theta_{fm}, \theta_{gm}$  known.

Estimates for the nonlinear functions are now given by the two NN as

$$\hat{f}(\mathbf{x}) = \hat{W}_f^T \sigma(\hat{V}_f^T \mathbf{x}) \quad (6.3.6)$$

$$\hat{g}(\mathbf{x}) = \hat{W}_g^T \sigma(\hat{V}_g^T \mathbf{x}), \quad (6.3.7)$$

with ‘hat’ denoting the actual values of the NN weights as provided by the tuning algorithms. Using Taylor Series expansions exactly as in Chapter 4 one may write the functional estimation errors as

$$\begin{aligned} \tilde{f}(\mathbf{x}) &= \tilde{W}_f^T (\hat{\sigma}_f - \hat{\sigma}'_f \hat{V}_f^T \mathbf{x}) + \hat{W}_f^T \hat{\sigma}'_f \tilde{V}_f^T \mathbf{x} + w_f \\ \tilde{g}(\mathbf{x}) &= \tilde{W}_g^T (\hat{\sigma}_g - \hat{\sigma}'_g \hat{V}_g^T \mathbf{x}) + \hat{W}_g^T \hat{\sigma}'_g \tilde{V}_g^T \mathbf{x} + w_g \end{aligned} \quad (6.3.8)$$

where the higher-order terms are bounded according to

$$\begin{aligned} \|w_f(t)\| &\leq C_0 + C_1 \|\tilde{\Theta}_f\|_F + C_2 \|r\| \cdot \|\tilde{\Theta}_f\|_F \\ \|w_g(t)\| &\leq C_0 + C_1 \|\tilde{\Theta}_g\|_F + C_2 \|r\| \cdot \|\tilde{\Theta}_g\|_F \end{aligned} \quad (6.3.9)$$

for computable positive constants  $C_0, C_1, C_2$ .

The next bounds will be needed. The first is easy to prove and the second relies on the fact that the standard activation functions are bounded with bounded derivatives.

**Lemma 6.3.1 (Technical Bounding Lemma) :**

a.  $\|x\| \leq d_0 + d_1 \|r\|$  for computable constants  $d_0, d_1$ .

b. In any compact set, there exist constants  $C_3, C_4$  so that

$$\begin{aligned} \|f(\mathbf{x})\| &= \|W_f^T \sigma(V_f^T \mathbf{x}) + \varepsilon_f\| \leq C_3 + C_4 \|r\| \\ \|g(\mathbf{x})\| &= \|W_g^T \sigma(V_g^T \mathbf{x}) + \varepsilon_g\| \leq C_3 + C_4 \|r\|. \end{aligned}$$

### 6.3.1.2 Controller Structure

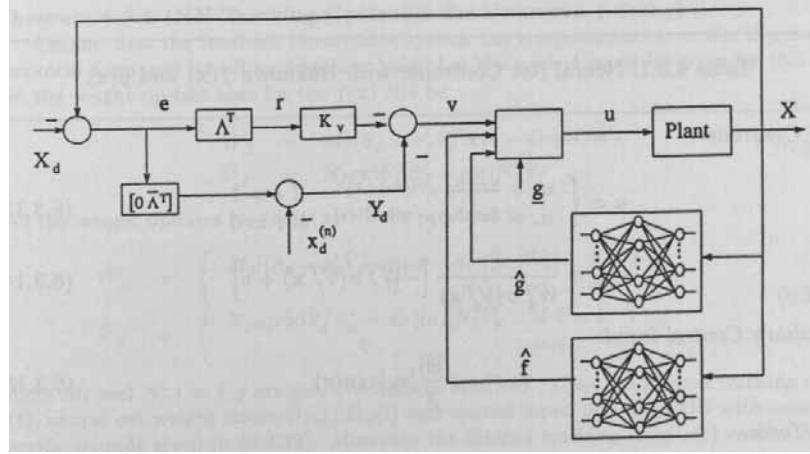
The design of adaptive tracking controllers for systems with unknown  $g(\mathbf{x})$  is extremely difficult due to the fact that the error dynamics (6.1.14) has a term  $\tilde{g}u_c$  where estimation uncertainty is multiplied by the control input. This arises from the well-definedness problem discussed in Section 6.1.2. We now introduce a controller which is well-defined at any values of  $\mathbf{x}$ ,  $\hat{\Theta}_f$ , and  $\hat{\Theta}_g$  and does not require restrictions on the system other than the assumptions already made. Particularly, no linearity in an unknown parameter set is needed.

To ensure the stability of the closed-loop system with a well-defined control input we propose the following action

$$u = \begin{cases} u_c + \frac{u_r - u_c}{2} \epsilon^{\gamma(|u_c| - s)} & \text{If } I = 1 \\ u_r - \frac{u_r - u_c}{2} \epsilon^{-\gamma(|u_c| - s)} & \text{If } I = 0 \end{cases} \quad (6.3.10)$$

where  $s > 0$  is a design parameter,  $\gamma < \ln 2/s$ , and  $u_c$  is as defined in (6.1.12) with the gain in (6.1.13) time-varying and given by

$$K_v = K_N + K_z \left[ (\|\hat{\Theta}_f\| + \Theta_{fm}) + s(\|\hat{\Theta}_g\| + \Theta_{gm}) \right] \quad (6.3.11)$$

Figure 6.3.1: NN controller with unknown  $f(\mathbf{x})$  and  $g(\mathbf{x})$ .

with the constant portion  $K_N > 0$  and the multiplier  $K_z > \max\{C_2, C_4/s\gamma\epsilon\Theta_{gm}\}$  design parameters. The robustifying control term is

$$u_r = -\mu \frac{|\hat{g}|}{g} |u_c| \operatorname{sgn}(r), \quad \mu \geq 2, \quad (6.3.12)$$

and the indicator function  $I$  is defined as

$$I = \begin{cases} 1 & \text{If } \hat{g} \geq g \text{ and } |u_c| \leq s \\ 0 & \text{o.w.} \end{cases}$$

and  $\operatorname{sgn}(\cdot)$  is the signum function. Note that  $I = 0$  when the estimate  $\hat{g}$  is too small or  $u_c(t)$  is too large, either one corresponding to an undesirable situation. It is important to note that  $u$  is well-defined for all  $\hat{g}$ , even when  $I = 0$ . Therefore,  $\hat{g} \rightarrow 0$  does not generate any unbounded control signal.

The intuition behind this controller is as follows. When  $\hat{g} \geq g$  and  $|u_c| < s$  then the total control action is set to  $u_c$ , otherwise control is switched to the auxiliary input  $u_r$ . This controller structure is shown in Fig. 6.3.1. Thus the resulting control action is well-defined everywhere and the uniform ultimate boundedness (UUB) of the closed-loop system can be shown with suitable NN weight tuning algorithms, as in the next subsection. Our proposed controller in (6.3.10) is formed from this idea with extra terms added for a smooth transition between  $u_c$  and  $u_r$  so that existence of solutions is guaranteed.

### 6.3.2 NN Weight Tuning for Tracking Stability

The next theorem shows how to tune the weights in the two NNs so that tracking performance and internal stability are guaranteed. The resulting controller is shown in Table 6.3.1. It is important for the existence of solutions to the closed-loop system to note that the trajectory generated for  $\hat{\Theta}_g(t)$  is continuous.

Table 6.3.1: Neural Net Controller with Unknown  $f(\mathbf{x})$  and  $g(\mathbf{x})$ *NN Controller:*

$$u = \begin{cases} u_c + \frac{u_r - u_c}{2} \epsilon^{\gamma(|u_c| - s)} & \text{If } I = 1 \\ u_r - \frac{u_r - u_c}{2} \epsilon^{-\gamma(|u_c| - s)} & \text{If } I = 0 \end{cases} \quad (6.3.13)$$

$$u_c = \frac{1}{\hat{W}_g^T \sigma(\hat{V}_g^T \mathbf{x})} \left[ -\hat{W}_f^T \sigma(\hat{V}_f^T \mathbf{x}) + v \right] \quad (6.3.14)$$

*Auxiliary Control Input:*

$$u_r = -\mu \frac{|\hat{g}|}{\underline{g}} |u_c| \operatorname{sgn}(r) \quad (6.3.15)$$

*PD Term*

$$v = -K_v r - Y_d \quad (6.3.16)$$

*Time-Varying Gain:*

$$K_v(t) = K_N + K_z \left[ (\|\hat{\Theta}_f(t)\| + \Theta_{fm}) + s(\|\hat{\Theta}_g(t)\| + \Theta_{gm}) \right] \quad (6.3.17)$$

*Indicator Function:*

$$I = \begin{cases} 1 & \text{If } \hat{g} \geq \underline{g} \text{ and } |u_c| \leq s \\ 0 & \text{o.w.} \end{cases} \quad (6.3.18)$$

*NN Weight Tunings:*

$$\begin{aligned} \dot{\hat{W}}_f &= M_f(\hat{\sigma}_f - \hat{\sigma}'_f \hat{V}_f^T \mathbf{x})r - \kappa|r|M_f \hat{W}_f \\ \dot{\hat{V}}_f &= N_f \mathbf{x} r \hat{W}_f^T \hat{\sigma}'_f - \kappa|r|N_f \hat{V}_f \end{aligned} \quad (6.3.19)$$

$$\begin{aligned} \dot{\hat{W}}_g &= \begin{cases} M_g[(\hat{\sigma}_g - \hat{\sigma}'_g \hat{V}_g^T \mathbf{x})u_c r - \kappa|r||u_c|\hat{W}_g] & \text{if } I = 1 \\ 0 & \text{o.w.} \end{cases} \\ \dot{\hat{V}}_g &= \begin{cases} N_g u_c r \mathbf{x} \hat{W}_g^T \hat{\sigma}'_g - \kappa|r||u_c|N_g \hat{V}_g & \text{if } I = 1 \\ 0 & \text{o.w.} \end{cases} \end{aligned} \quad (6.3.20)$$

*Signals:*

$$\mathbf{e}(t) = \mathbf{x}(\mathbf{t}) - \mathbf{x}_d(t) \quad \text{Tracking error} \quad (6.3.21)$$

$$r(t) = \Lambda^T \mathbf{e}(t) \quad \text{Filtered tracking error} \quad (6.3.22)$$

$$Y_d = -x_d^{(n)} + [0 \ \bar{\Lambda}^T] \mathbf{e} \quad \text{Desired trajectory feedforward signal} \quad (6.3.23)$$

*Design Parameters:*Constant gains  $\Lambda > 0$ ,  $K_N > 0$ ,  $K_z > \max\{C_2, C_4/s\gamma\epsilon\Theta_{gm}\}$ Tuning matrices  $M_i$ ,  $N_i$  symmetric and positive definite.Scalar  $\kappa > 0$ ,  $s > 0$ ,  $\gamma < \ln 2/s$ ,  $\mu \geq 2$ .

**Theorem 6.3.1 (NN Tracking Controller for Unknown  $f$  and  $g$ ) :**

Assume that the feedback linearizable system has a representation in the Brunovsky canonical form and let all assumptions hold. Let the control input be given by (6.3.10). Let the weight update laws for the  $f(\mathbf{x})$  NN be

$$\begin{aligned}\dot{\hat{W}}_f &= M_f(\hat{\sigma}_f - \hat{\sigma}'_f \hat{V}_f^T \mathbf{x})r - \kappa|r|M_f \hat{W}_f \\ \dot{\hat{V}}_f &= N_f r \mathbf{x} \hat{W}_f^T \hat{\sigma}'_f - \kappa|r|N_f \hat{V}_f\end{aligned}\quad (6.3.24)$$

and the weight updates for the  $g(\mathbf{x})$  NN be provided by

$$\begin{aligned}\dot{\hat{W}}_g &= \begin{cases} M_g[(\hat{\sigma}_g - \hat{\sigma}'_g \hat{V}_g^T \mathbf{x})u_c r - \kappa|r||u_c|\hat{W}_g] & \text{if } I = 1 \\ 0 & \text{o.w.} \end{cases} \\ \dot{\hat{V}}_g &= \begin{cases} N_g u_c r \mathbf{x} \hat{W}_g^T \hat{\sigma}'_g - \kappa|r||u_c|N_g \hat{V}_g & \text{if } I = 1 \\ 0 & \text{o.w.} \end{cases}\end{aligned}\quad (6.3.25)$$

where  $M_i$  and  $N_i$   $i = f, g$  are positive definite matrices. Then the filtered tracking error  $r(t)$ , neural net weight errors  $\tilde{\Theta}_f(t), \tilde{\Theta}_g(t)$  and control input  $u(t)$  are UUB with constant specific bounds given in (6.3.32). Moreover the filtered tracking error  $r(t)$  can be made arbitrarily small by increasing the gain  $K_N$ .

Proof:

Let the Lyapunov function candidate be

$$L = \frac{1}{2}r^2 + \frac{1}{2}tr \left\{ \sum_{i=f,g} \tilde{W}_i^T M_i^{-1} \tilde{W}_i + \tilde{V}_i^T N_i^{-1} \tilde{V}_i \right\}. \quad (6.3.26)$$

We will study the derivative of (6.3.26) in two mutually exclusive and exhaustive regions.

Region 1:  $|\hat{g}| \geq g$  and  $|u_c| \leq s$ .

Substitution of the functional approximation errors (6.3.8) into the error system dynamics (6.1.14) yields

$$\begin{aligned}\dot{r} &= -K_v r + \tilde{W}_f^T (\hat{\sigma}_f - \hat{\sigma}'_f \hat{V}_f^T \mathbf{x}) + \hat{W}_f^T \hat{\sigma}'_f \hat{V}_f^T \mathbf{x} \\ &\quad + [\tilde{W}_g^T (\hat{\sigma}_g - \hat{\sigma}'_g \hat{V}_g^T \mathbf{x}) + \hat{W}_g^T \hat{\sigma}'_g \hat{V}_g^T \mathbf{x}] u_c \\ &\quad + d + w_f + w_g u_c + g u_d.\end{aligned}$$

The time derivative of (6.3.26) is

$$\dot{L} = rr + tr \left\{ \sum_{i=f,g} \tilde{W}_i^T M_i^{-1} \dot{\tilde{W}}_i + \tilde{V}_i^T N_i^{-1} \dot{\tilde{V}}_i \right\}. \quad (6.3.27)$$

Substitute now  $\dot{r}$  into (6.3.27) and perform a simple manipulation, (i.e. using  $\mathbf{x}^T \mathbf{y} = tr\{\mathbf{x}^T \mathbf{y}\} = tr\{\mathbf{y} \mathbf{x}^T\}$ , one can place weight matrices inside a trace operator). Then

$$\begin{aligned}\dot{L} &= -K_v r^2 + r(d + w_f) + r g u_d + r w_g u_c + \\ &\quad tr\{\tilde{W}_f^T (\hat{\sigma}_f - \hat{\sigma}'_f \hat{V}_f^T \mathbf{x})r + M_f^{-1} \dot{\tilde{W}}_f\} + tr\{\tilde{V}_f^T (\mathbf{x} r \hat{W}_f^T \hat{\sigma}'_f + N_f^{-1} \dot{\tilde{V}}_f)\} + \\ &\quad tr\{\tilde{W}_g^T (\hat{\sigma}_g - \hat{\sigma}'_g \hat{V}_g^T \mathbf{x})u_c r + M_g^{-1} \dot{\tilde{W}}_g\} + tr\{\tilde{V}_g^T (\mathbf{x} u_c r \hat{W}_g^T \hat{\sigma}'_g + N_g^{-1} \dot{\tilde{V}}_g)\}.\end{aligned}$$

With the update rules give in (6.3.24) and (6.3.25)

$$\begin{aligned}\dot{L} &= -K_v r^2 + r(d + w_f) + r w_g u_c + r g u_d + \\ &\quad \kappa|r|tr\{\tilde{\Theta}_f^T \hat{\Theta}_f\} + \kappa|r||u_c|tr\{\tilde{\Theta}_g^T \hat{\Theta}_g\}\end{aligned}$$

and from the inequality

$$\text{tr}\{\tilde{\Theta}^T \hat{\Theta}\} = <\tilde{\Theta}^T, \Theta> - \text{tr}\{\tilde{\Theta}^T \tilde{\Theta}\} \leq \|\tilde{\Theta}\|(\Theta_m - \|\tilde{\Theta}\|),$$

it follows that

$$\begin{aligned} \dot{L} \leq & -K_v r^2 + r(d + w_f) + rw_g u_c + rgu_d + \\ & \kappa|r|\|\tilde{\Theta}_f\|(\Theta_{fm} - \|\tilde{\Theta}_f\|) + \kappa|r|\|\tilde{\Theta}_g\|(\Theta_{gm} - \|\tilde{\Theta}_g\|)|u_c|. \end{aligned}$$

Substitute the upper bound of  $w_f$  and  $w_g$  according to (6.3.9) and  $K_v$  from (6.3.11) to yield

$$\begin{aligned} \dot{L} \leq & -\{K_N + K_z [\|\hat{\Theta}_f\| + \Theta_{fm} + s(\|\hat{\Theta}_g\| + \Theta_{gm})]\}r^2 + \kappa|r|\|\tilde{\Theta}_f\|(\Theta_{fm} - \|\tilde{\Theta}_f\|) + \\ & \kappa|r||u_c|\|\tilde{\Theta}_g\|(\Theta_{gm} - \|\tilde{\Theta}_g\|) + (C_2\|\tilde{\Theta}_f\||r|^2 + C_1\|\tilde{\Theta}_f\||r| + (b_d + C_0)|r|) \\ & + (C_2\|\tilde{\Theta}_g\||r|^2 + C_1\|\tilde{\Theta}_g\||r| + C_0|r|)|u_c| - rg \left[ \mu \frac{|\hat{g}|}{g} |u_c| \text{sgn}(r) + u_c \right] \frac{\epsilon^{-\gamma(s-|u_c|)}}{2}. \end{aligned}$$

Picking  $K_z > C_2$ ,

$$\begin{aligned} \dot{L} \leq & -|r| [K_N|r| + \kappa\|\tilde{\Theta}_f\|(\|\tilde{\Theta}_f\| - \Theta_{fm}) - b_d - C_0 - C_1\|\tilde{\Theta}_f\|] \\ & -|r|\cdot|u_c| [\kappa\|\tilde{\Theta}_g\|(\|\tilde{\Theta}_g\| - \Theta_{gm}) - C_0 - C_1\|\tilde{\Theta}_g\|] - \\ & rg \left[ \mu \frac{|\hat{g}|}{g} |u_c| \text{sgn}(r) + u_c \right] \frac{\epsilon^{-\gamma(s-|u_c|)}}{2}. \end{aligned}$$

Since  $|\hat{g}| \geq g$  and  $\mu \geq 2$  the last term in this inequality is always negative. Now we can write the final form by completing the squares

$$\dot{L} \leq -|r| \{ K_N|r| + \kappa(\|\tilde{\Theta}_f\| - C_f)^2 + \kappa|u_c|(\|\tilde{\Theta}_g\| - C_g)^2 - D_1 \} \quad (6.3.28)$$

where

$$D_1 \equiv b_d + (1+s)C_0 + \kappa(C_f^2 + sC_g^2),$$

and

$$C_f \equiv \frac{\Theta_{fm}}{2} + \frac{C_1}{2\kappa}, \quad C_g \equiv \frac{\Theta_{gm}}{2} + \frac{C_1}{2\kappa}.$$

Observe that the terms in braces in (6.3.28) define a conic ellipsoid, a compact set around the origin of  $(r, \|\tilde{\Theta}_f\|, \|\tilde{\Theta}_g\|)$ . We can, therefore, deduce from (6.3.28) that, if

$$|r| > \delta_{r_1}$$

then  $\dot{L} \leq 0$  for all  $\|\tilde{\Theta}_f\|$  and  $\|\tilde{\Theta}_g\|$  where

$$\delta_{r_1} = \frac{D_1}{K_N}, \quad (6.3.29)$$

or, if

$$\|\tilde{\Theta}_f\| > \delta_{f_1}$$

then  $\dot{L} \leq 0$  for all  $|r|$  and  $\|\tilde{\Theta}_f\|$  where

$$\delta_{f_1} = \frac{C_f}{2} + \sqrt{\frac{D_1}{\kappa}}$$

(since the quadratic terms dominate the linear terms after the certain points  $\delta_{r_1}$  and  $\delta_{f_1}$ ).

For the weights of  $\hat{g}(\mathbf{x})$  we can only claim an upper bound when  $|u_c| \geq \varepsilon_u > 0$  for any positive  $\varepsilon_u$  as

$$\delta_{g_1} = \frac{C_g}{2} + \sqrt{\frac{\frac{C_g^2}{4} + \frac{C_f^2}{4\varepsilon_u} + (1 + 1/\varepsilon_u)C_0 + (\ell\Theta_{gm} + \varepsilon)/2}{\kappa}}$$

$$\delta_{g_1} = \frac{C_g}{2} + \sqrt{\frac{D_1}{\varepsilon_u \kappa}}.$$

However it is not straightforward to show a bound on  $\hat{g}$ , through (6.3.28) when  $|u_c| < \varepsilon_u$ . This difficulty appears due to the multiplication by  $|u_c|$  in  $\hat{L}$ . In order to obtain a bound on  $\|\tilde{\Theta}_g\|$  when  $|u_c| < \varepsilon_u$  we will investigate its time derivative. Fortunately, the update laws for  $\hat{\Theta}_g$  can be rewritten in the form

$$\dot{\hat{\Theta}}_g = B_1(r)u_c\hat{\Theta}_g + B_2(r)u_c$$

where  $B_i$  matrices are functions of  $r$  only. Integration of this equation in the interval of  $[t_0, t_0 + T]$  yields

$$\hat{\Theta}_g = \int_{t_0}^{t_0+T} B_1(r, \tau)u_c(\tau)\hat{\Theta}_g d\tau + \int_{t_0}^{t_0+T} B_2(r, \tau)u_c(\tau)d\tau + \hat{\Theta}_g(t_0).$$

Thus

$$\|\hat{\Theta}_g\| \leq \int_{t_0}^{t_0+T} \|B_1(r, \tau)\| |u_c(\tau)| \|\hat{\Theta}_g\| d\tau +$$

$$\int_{t_0}^{t_0+T} \|B_2(r, \tau)\| |u_c(\tau)| d\tau + \|\hat{\Theta}_g(t_0)\|$$

is satisfied. We have already shown that  $r \in L_\infty$ , which implies that for a finite  $T$ , the bounds

$$\int_{t_0}^{t_0+T} \|B_1(r, \tau)\| d\tau \leq \beta_1(T)$$

and

$$\int_{t_0}^{t_0+T} \|B_2(r, \tau)\| d\tau \leq \beta_2(T)$$

hold. This allows us to write

$$\|\hat{\Theta}_g\| \leq \int_{t_0}^{t_0+T} \varepsilon_u \beta_1 \|\hat{\Theta}_g\| d\tau + \varepsilon_u \beta_2 T + \|\hat{\Theta}_g(t_0)\|.$$

From the Bellman-Gronwall Lemma (Chapter 2) we can now infer a constant upper bound for  $\|\hat{\Theta}_g\|$  as

$$\|\hat{\Theta}_g\| \leq (\varepsilon_u \beta_2 T + \|\hat{\Theta}_g(t_0)\|) e^{\varepsilon_u \beta_2 T}.$$

Since

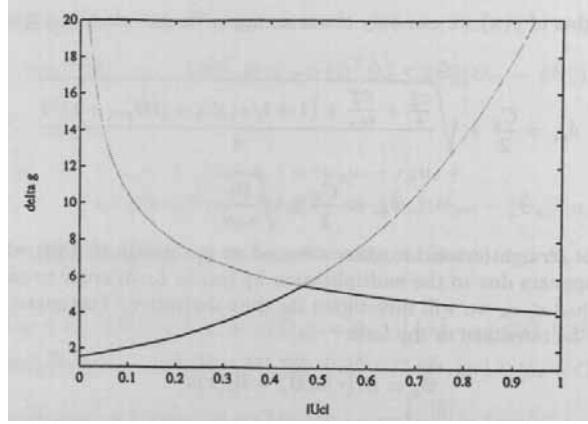
$$\|\tilde{\Theta}_g\| = \|\hat{\Theta}_g - \Theta_g\| \leq \|\hat{\Theta}_g\| + \Theta_{gm}.$$

This implies another upper bound on  $\|\tilde{\Theta}_g\|$  as

$$\|\tilde{\Theta}_g\| \leq \delta_{g2}$$

where

$$\delta_{g2} = (\varepsilon_u \beta_2 T + \|\hat{\Theta}_g(t_0)\|) e^{\varepsilon_u \beta_2 T} + \Theta_{gm}.$$

Figure 6.3.2: Illustration of the upper bound on  $\|\tilde{\Theta}_g\|$ .

We have shown two upper bounds on  $\|\tilde{\Theta}_g\|$  then one can establish a finite upper bound on  $\|\tilde{\Theta}_g\|$  for all  $|u_c| < s$  as

$$\delta_g = \min\{\delta_{g_1}, \delta_{g_2}\}.$$

The symbolic variation of these bounds is illustrated in Fig. 6.3.2. This shows the boundedness of  $r, \tilde{\Theta}_f, \tilde{\Theta}_g, u_r$ , and since  $|u_c| \leq s$ , this implies that  $u \in L_\infty$ .

Region 2:  $\hat{g} < \underline{g}$  or  $|u_c| > s$ .

With the update rules corresponding to this region  $\dot{L}$  becomes

$$\dot{L} \leq -K_v r^2 + r(d + w_f) + r\tilde{g}u_c + rgu_d + \kappa|r|\|\tilde{\Theta}_f\|(\Theta_f - \|\tilde{\Theta}_f\|). \quad (6.3.30)$$

Observe that  $u_c$  may not be defined in this region though, because of notational simplicity we will use it in the  $\hat{g}u_c$  or  $u_c\epsilon^{-\gamma(|u_c|-s)}$  forms which are bounded when  $\hat{g} = 0$ . Now define

$$\dot{L}_g \equiv r\tilde{g}u_c + rgu_d = -r\hat{g}u_c + rgu.$$

Substitution of the controller corresponding to this region yields

$$\dot{L}_g = -r\hat{g}u_c - rgu_r(1 - \frac{1}{2}\epsilon^{-\gamma(|u_c|-s)}) + \frac{1}{2}rgu_c\epsilon^{-\gamma(|u_c|-s)}.$$

If  $|u_c| > s$ ,

$$\dot{L}_g \leq |r|\|\hat{g}\||u_c|(1 - \frac{\mu}{2}\frac{g}{\underline{g}}) + \frac{1}{2}|r|g|u_c|\epsilon^{-\gamma(|u_c|-s)}$$

Assume at time  $t_0$  that  $g$  is in a compact set, invoke Assumption 6.3.1 to yield

$$|g| \leq C_3 + C_4|r|.$$

Then using  $\epsilon^{\gamma s} \leq 2$

$$\dot{L}_g \leq |r|(C_3 + C_4|r|)\frac{1}{\gamma\epsilon}.$$

The other case occurs when  $|u_c| < s$  and  $|\hat{g}| < \underline{g}$ . This affects  $\dot{L}_g$  as follows,

$$\begin{aligned}\dot{L}_g &\leq |r||\hat{g}||u_c| + \frac{1}{2}rgu_c\epsilon^{-\gamma(|u_c|-s)} \\ &\leq |r|\left[\underline{g}s + \frac{(C_3 + C_4|r|)}{\gamma\epsilon}\right].\end{aligned}$$

Therefore

$$\dot{L}_g \leq \frac{|r|}{\gamma\epsilon} [\gamma\epsilon\underline{g}s + C_3 + C_4|r|]$$

in this region. Now pick  $K_z > \frac{C_4}{\gamma\epsilon s\Theta_{gm}}$  and substitute  $\dot{L}_g$  into  $L$  to obtain

$$\dot{L} \leq -|r|\{K_N|r| + \kappa(\|\tilde{\Theta}_f\| - C_f)^2 - D_2\}$$

where the constant  $D_2$  is

$$D_2 \equiv b_d + C_0 + \kappa C_f^2 + \frac{C_3}{\gamma\epsilon} + \underline{g}s.$$

Bounds for  $|r|$  and  $\|\Theta_f\|$  are

$$\delta_{r_2} = \frac{D_2}{K_N} \quad (6.3.31)$$

and

$$\delta_{f_2} = \frac{C_f}{2} + \sqrt{\frac{D_2}{\kappa}}.$$

Whenever  $|r| > \delta_{r_2}$  or  $\|\tilde{\Theta}_f\| > \delta_{f_2}$ ,  $\dot{L} \leq 0$ . This implies that  $\mathbf{x}$  and  $\tilde{\Theta}_f$  stay in a compact set so does  $g(\mathbf{x})$ . This shows the boundedness of  $r$ ,  $\tilde{\Theta}_f$  together with bounded  $\tilde{\Theta}_g$  implies that  $u_r \in L_\infty$ , hence  $u \in L_\infty$ .

Reprise: Combining the results from region one and two, one can readily set

$$\begin{aligned}\delta_r &= \max\{\delta_{r_1}, \delta_{r_2}\}, \quad \delta_f = \max\{\delta_{f_1}, \delta_{f_2}\} \\ \delta_g &= \min\{\delta_{g_1}, \delta_{g_2}\}.\end{aligned} \quad (6.3.32)$$

Thus for both regions, if  $|r| > \delta_r$  or  $\|\tilde{\Theta}_f\| > \delta_f$  or  $\|\tilde{\Theta}_g\| > \delta_g$ , then  $\dot{L} \leq 0$  and  $u \in L_\infty$ . Let us denote  $(|r|, \|\tilde{\Theta}_f\|, \|\tilde{\Theta}_g\|)$  by new coordinate variables  $(\xi_1, \xi_2, \xi_3)$ . Define the region

$$\mathcal{D} : \{\xi \mid \xi_1 < \delta_r, \xi_2 < \delta_f, \xi_3 < \delta_g\},$$

then there exists an open set

$$\Omega : \{\xi \mid \xi_1 < \bar{\delta}_r, \xi_2 < \bar{\delta}_f, \xi_3 < \bar{\delta}_g\},$$

where  $\bar{\delta}_i > \delta_i$  implies that  $\mathcal{D} \subset \Omega$ . These sets are shown in Fig. 6.3.3. We have proved that whenever  $\xi_i > \delta_i$  then  $L(\xi)$  will not increase. This implies that  $\xi$  will decrease, in other words it will stay in the region  $\Omega$  which is an invariant set. Therefore all the signals in the closed-loop system remain bounded. This concludes the proof.  $\square$

*See the remarks following Theorem 6.2.1. The tuning algorithms presented in the theorem are augmented versions of backpropagation through time. Simplified Hebbian tuning algorithms are given in [Yeşildirek 1994] (c.f. Chapter 4). Also note the following.*

*Remarks:*

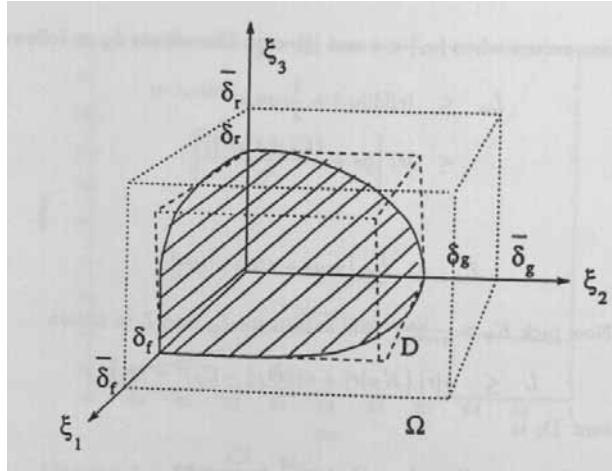


Figure 6.3.3: Illustration of the invariant set.

1. For practical purposes, (6.3.32) can be considered as bounds on  $|r|$ ,  $\|\tilde{\Theta}_f\|$ , and  $\|\tilde{\Theta}_g\|$  in the sense that excursions above these bounds will be small.
2. Note from the definitions of  $\delta_{r_1}, \delta_{r_2}$  that the bound on the tracking error may be kept arbitrarily small by selecting the control gain  $K_N$  large enough.
3. Note that the tuning of the NN that estimates  $g(\mathbf{x})$  is interrupted if  $\hat{g}$  becomes too small. If the switching parameter  $s$  is chosen too small, it will limit the control input and result in a large tracking error which gives undesirable closed-loop performance. If it is too large, the control actuator may saturate as  $u(t)$  increases in magnitude.
4. Stability of the closed-loop system is shown without making any assumptions on the initial NN weight values. The NNs can easily be initialized as  $\hat{\Theta}_f(0) = 0$  and  $\hat{\Theta}_g(0) > \hat{g}^{-1}(g)$ . It is crucial to note that the NN need not to be trained off-line before use in closed-loop. No assumptions of the initial weights being in an invariant set, or a region of attraction, or a feasible region are needed.

### 6.3.3 Illustrative Simulation Examples

#### Example 6.3.1 (Van der Pol System) :

As an example consider a Van der Pol's system

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= (1 - x_1^2)x_2 - x_1 + (1 + x_1^2 + x_2^2)u\end{aligned}\tag{6.3.33}$$

which is in the controllability canonical form and has  $g(\mathbf{x}) \geq 1 \equiv g \forall \mathbf{x}$ . The neural nets which are used for  $\hat{f}$  and  $\hat{g}$  consist of 10 neurons. The function  $sgn(r)$  is approximated by a hyperbolic tangent function in simulations. Design parameters are set to  $s = 10$ ,  $\gamma = 0.05$ ,  $K_N = 20$ ,  $\lambda_1 = 5$ ,  $M_i = N_i = 20$ ,  $\mu = 4$  and the rest are set equal to 1. Initial

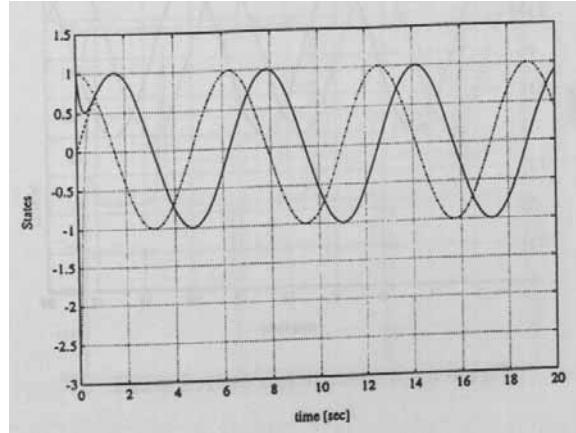


Figure 6.3.4: Actual and desired states.

conditions are  $\hat{\Theta}_f(0) = 0$  and  $\hat{\Theta}_g(0) = 0.4$  so that  $\hat{g}(0) > 1$  and  $x_1(0) = x_2(0) = 1$ . The desired trajectory is defined as  $y_d(t) = \sin t$ .

The actual and desired outputs obtained by simulation are shown in Fig. 6.3.4 and the control action is shown in Fig. 6.3.5. Note that almost perfect tracking is obtained in less than one second.  $\square$

**Example 6.3.2 (Ill-Defined Relative Degree System) :**

Let us change the above Van der Pol's systems so that it is ill-defined when  $x_1(t) = 0$ :

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= (1 - x_1^2)x_2 - x_1 + x_1^2 u.\end{aligned}\tag{6.3.34}$$

We took the same NN controller parameters as in Example 1. Although  $g(\mathbf{x})$  is not lower bounded, we treated  $g$  as a design parameter and set  $\underline{g} = 0.1$ . The objective is to show that our NN controller can give good performance even with systems that are *not well-defined in relative degree*.

Simulation results showing the performance of the NN controller are given in Figs. 6.3.6 and 6.3.7. Observe that around the singularity points ( $t = n\pi$  for  $n = \dots, -1, 0, 1, \dots$  after tracking is satisfied) the controller needed to linearize the system reaches its peak which is set by the design parameters  $\underline{g}$  and  $s$ . That is, when  $u \gg s$ ,  $u \rightarrow u_r$  which is proportional to  $\underline{g}^{-1}$ . There is a trade-off between the amount of control signal which can be applied and the bound on the tracking error in the neighborhood of those singular points. By choosing a lower bound on  $g$ , the amount of the control and tracking error are decided.

Further discussion on control of systems with ill-defined relative degree is given in (Commuri and Lewis 1994).  $\square$

**Example 6.3.3 (Chemical Stirred-Tank Reactor) :**

Chemical systems, in general, can be very difficult to control because of their strong nonlinearities which are difficult to model, even though they may have few variables. They offer a good application for NN-based control.

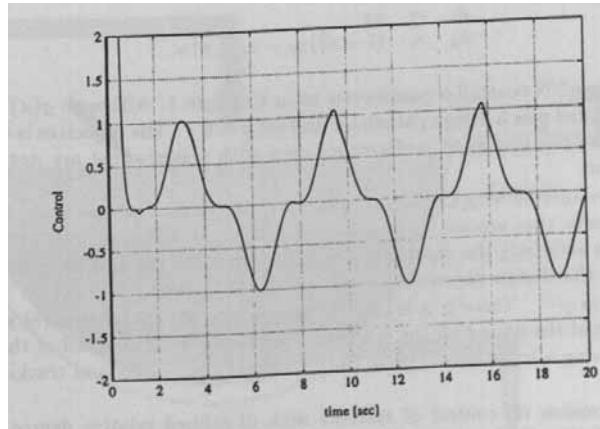


Figure 6.3.5: Control input.

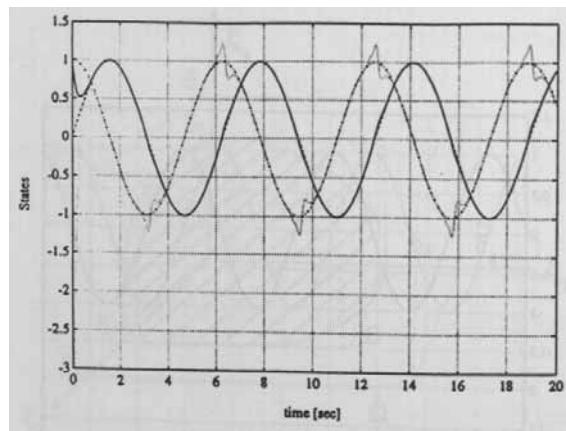


Figure 6.3.6: Actual and desired states.

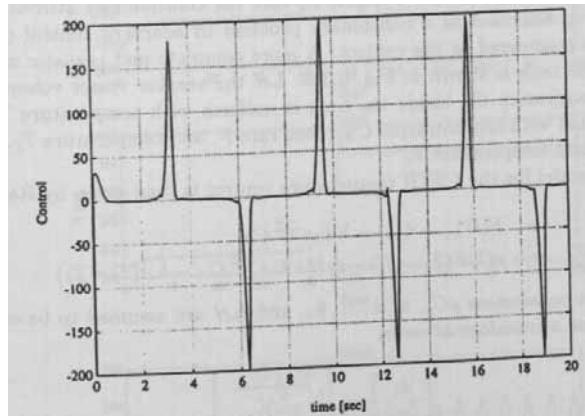


Figure 6.3.7: Control input.

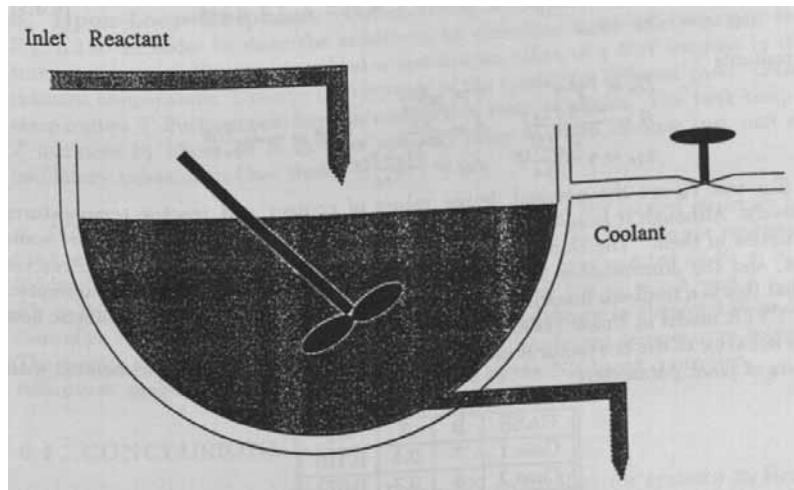


Figure 6.3.8: Chemical stirred-tank reactor (CSTR) process.

**a. CSTR Dynamical Model.** There are many different models for bioreactors that contain liquid in a tank and cells that consume substrates and yield desired products and undesired byproducts. As an example, we take the continuously stirred tank reactor (CSTR), which is described as a benchmark problem in adaptive control (Ungar 1990) when the flow is considered as the control. A more complete and realistic model using a cooler around the tank is shown in Fig. 6.3.8. Let the reactor vessel volume be  $V$ . We assume the concentration  $C_A$  inside the tank is uniform with temperature  $T$ . The inlet reactant is supplied with concentration  $C_{Af}$ , feed rate  $F$ , and temperature  $T_f$ . The control input is the coolant temperature  $T_c$ .

A dynamic model for the CSTR temperature control is then given by Ray (1975)

$$\begin{aligned} V \frac{dC_A}{dt} &= F(C_{Af} - C_A) - V k_o e^{-\frac{E}{RT}} C_A \\ V \rho C_p \frac{dT}{dt} &= \rho C_p F(T_f - T) - \Delta H V k_o e^{-\frac{E}{RT}} C_A - hA(T - T_c) \end{aligned} \quad (6.3.35)$$

where the system parameters  $\rho C_p$ ,  $h$ ,  $A$ ,  $E$ ,  $k_o$ , and  $\Delta H$  are assumed to be constant. Let  $t = \frac{F}{V} t'$  and define a transformation as

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \frac{C_{Af} - C_A}{C_{Af}} \\ \frac{T - T_{fd}}{T_{fd}} \end{bmatrix} \quad (6.3.36)$$

with  $T_{fd}$  the nominal inlet temperature. Note that the normalized  $x_1(t)$  is the difference between inlet and reactor concentrations and it is always less than or equal to 1 (since  $C_{Af} \geq C_A \geq 0$ ).

Suppose the objective is to control the temperature  $T$  of the reactor. Then, (6.3.35) can be rewritten in terms of dimensionless variables as

$$\begin{aligned} \dot{x}_1 &= -x_1 + D_a(1 - x_1)e^{\frac{\gamma x_2}{\gamma + x_2}} \\ \dot{x}_2 &= -x_2 + BD_a(1 - x_1)e^{\frac{\gamma x_2}{\gamma + x_2}} - \beta(x_2 - x_{2c}) + d + \beta u \\ y &= x_2 - x_{2d} \end{aligned} \quad (6.3.37)$$

with constants

$$\begin{aligned} D_a &= \frac{k_o V}{F} e^{-\gamma} & \gamma &= \frac{E}{RT_{fd}} \\ B &= \frac{-\Delta H C_{Af} \gamma}{\rho C_p T_{fd}} & \beta &= \frac{hA}{\rho C_p F} & d &= \gamma \frac{T_f - T_{fd}}{T_{fd}} \\ x_{2c} &= \gamma \frac{T_{cd} - T_{fd}}{T_{fd}} & x_{2d} &= \gamma \frac{T_d - T_{fd}}{T_{fd}} \end{aligned}$$

where  $T_{cd}$  and  $T_d$  are the nominal design values of coolant and reactor temperatures respectively. Although it is assumed that  $D_a$ ,  $\gamma$ ,  $B$ , and  $\beta$  are constants, there are some uncertainties in them. The Damkohler constant  $D_a$  is, in fact, a function of the reactor catalyst, and the dimensionless heat transfer constant  $\beta$  is a slowly-varying parameter. Note that this is a feedback linearizable model, though the less complete and realistic flow control CSTR model in Ungar (1990) is not.

The behavior of the bioreactor is usually studied for three typical cases associated with three sets of plant parameters:

CASE	B	$\beta$	$D_a$
Case 1	7	0.5	0.110
Case 2	8	0.3	0.072
Case 3	11	1.5	0.135

with  $\gamma = 20$ . The corresponding regions have been studied previously (Liu and Lewis 1994, Ray 1975). The desired inlet reactant temperature,  $T_{fd}$  is here selected as 300° K, and the control objective is to keep the reactor temperature  $T$  at this level. Then, the tank concentration will converge to some constant steady-state value.

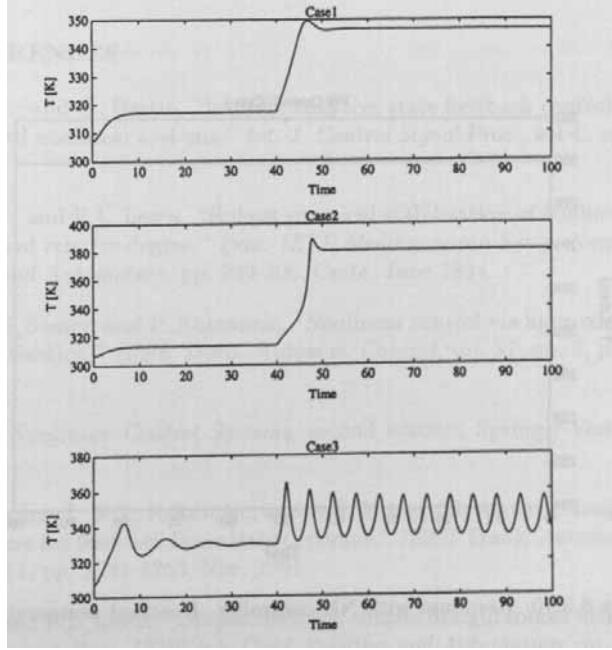


Figure 6.3.9: CSTR open-loop response to a disturbance.

**b. Open-Loop Response.** The open-loop response of the bioreactor is shown in Fig. 6.3.9. In order to show the sensitivity to variations in the inlet reactant temperature, at time  $t = 40$  unit, we added a disturbance effect of a  $5^\circ K$  increase in the inlet reactant temperature. Observe the response of the reactor for different cases. Clearly, the temperature  $T$  fluctuates widely, an undesirable state of affairs. The tank temperature  $T$  increases by about  $30^\circ K$  in Case one and twice as much in Case two, and exhibits oscillatory behavior in Case three.

**c. NN Controller Design and Simulation.** The NN control structure in Table 6.3.1 was now used to regulate the CSTR temperature  $T$ . We use six neurons in the hidden layers for both  $\hat{f}$  and  $\hat{g}$ . The design parameters were selected as  $K_v = 20$ ,  $\Lambda = 5$ ,  $K_z = 0.01$ ,  $\Theta_{fm,gm} = 0$ ,  $M_i = N_i = 50$ ,  $\kappa = 10$ ,  $s = 100$ ,  $g = 0.2$ , and  $\mu = 4$ . The controller is simulated for Case 1 and its response is shown in Figs. 6.3.10 and 6.3.11. The results show that fast on-line convergence to the desired temperature setpoint and robustness against disturbances can be achieved by the NN-based controller.  $\square$

## 6.4 CONCLUSIONS

*In this chapter we showed how to design NN controllers for systems in Brunovsky form. Two cases were considered—the case of known input influence function and the case of unknown input influence function. In the latter case, great care had to be taken to ensure that the control remained bounded. Several examples were considered, including a continuously stirred tank chemical reactor, which was presented*

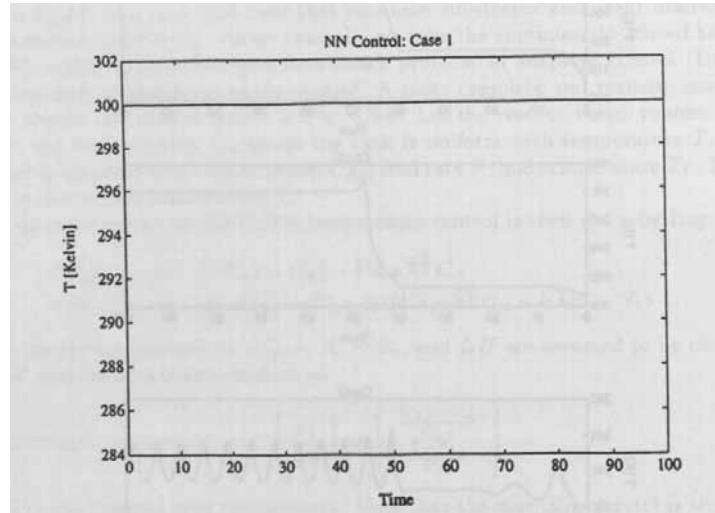


Figure 6.3.10: Response with NN controller. Reactant temperature,  $T$ .

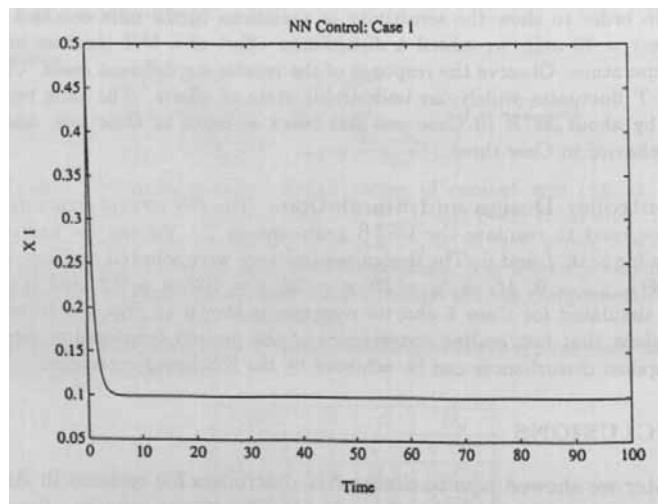


Figure 6.3.11: Response with NN controller. The state  $x_1(t)$ .

as a benchmark problem for NN control in Ungar (1990) due to its complexity.

## 6.5 REFERENCES

- Campion, G., and G. Bastin, "Indirect adaptive state feedback control of linearly parameterized nonlinear systems," *Int. J. Control Signal Proc.*, vol 4., pp. 345-358, 1990.
- Commuri, S., and F.L. Lewis, "Robust practical stabilization of nonlinear systems with ill-defined relative degree," *Proc. IEEE Mediterranean Symp. New Directions in Control and Automation*, pp. 299-306, Crete, June 1994.
- Hauser, J., S. Sastry, and P. Kokotovic, "Nonlinear control via approximate input-output linearization," *IEEE Trans. Automat. Control*, vol. 37, no. 3, pp. 392-398, 1992.
- Isidori, A., *Nonlinear Control Systems, second edition*, Springer-Verlag, Berlin, 1989
- Kanellakopoulos, I., P.V. Kokotovic, and A.S. Morse, "Systematic design of adaptive controllers for feedback linearizable systems," *IEEE Trans. Automat. Control*, vol. 36, no. 11, pp. 1241-1253, Nov. 1991.
- Kim, Y.H., and F.L. Lewis, "Output feedback control of rigid robots using dynamic neural networks," *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 1923-1928, Minneapolis, April 1996.
- Kim, Y., and F.L. Lewis, "Nonlinear observer design using dynamic recurrent neural networks, *Proc. IEEE Conf. Decision and Control*, pp. 949-954, Kobe, Dec. 1996.
- Lewis, F.L., C.T. Abdallah, and D.M. Dawson, *Control of Robot Manipulators*, Macmillan, New York, 1993.
- Liu, C.-C., and F.-C. Chen, "Adaptive control of non-linear continuous systems using neural networks-general relative degree and MIMO cases," *Int. J. Control*, vol. 58, no. 2, pp. 317-335, 1993.
- Liu, K., and F.L. Lewis, "Robust control of a continuous stirred-tank reactor," *Proc. American Control Conf.*, pp. 2350-2354, Baltimore, Maryland 1994.
- Nam, K., and A. Arapostathis, "A model-reference adaptive control scheme for pure-feedback nonlinear systems," *IEEE Trans. Automat. Control*, vol. 33, pp 803-811, Sept 1988.
- Narendra, K.S., and A.M. Annaswamy, "A new adaptive law for robust adaptation without persistent excitation," *IEEE Trans. Automat. Control*, vol. AC-32, no. 2, pp. 134-145, Feb. 1987.
- Polycarpou, M.M., and P.A. Ioannou, "Identification and control using neural network models: design and stability analysis," *Tech. Report 91-09-01, Dept. Elect. Eng. Sys., Univ. S. Cal.*, Sept. 1991.

*Ray, W.H., "New approaches to the dynamics of nonlinear systems with implications of process and control system design," Chemical Process Control II, pp. 245-267, 1975.*

*Rovithakis, G.A., and M.A. Christodoulou, "Adaptive control of unknown plants using dynamical neural networks," IEEE Trans. Systems, Man, and Cybernetics, vol. 24, no. 3, pp. 400-412, 1994.*

*Slotine, J.-J.E., and W. Li, Applied Nonlinear Control, Prentice-Hall, New Jersey, 1991.*

*Taylor, D.G., P.V. Kokotovic, R. Marino, and I. Kanellakopoulos, "Adaptive regulation of nonlinear systems with unmodeled dynamics," IEEE Trans. Automat. Control, vol.34, pp. 405-412, Apr. 1989.*

*Teel, A., R. Kadiyala, P.V. Kokotovic, and S.S. Sastry, "Indirect techniques for adaptive input output linearization of nonlinear systems," Int. J. Control, vol. 53, pp. 193-222, Jan. 1991.*

*Ungar, L.H., "A bioreactor benchmark for adaptive network-based process control," in Neural Networks for Control, Chap. 16, ed. W.T. Miller, R.S. Sutton, and P.J. Werbos, MIT press, Boston, 1990.*

*Yeşildirek, A., Nonlinear Systems Control Using Neural Networks, Ph.D. Thesis, Dept. of Electrical Engineering, The University of Texas at Arlington, Arlington, Texas 76019, USA, 1994.*

*Yeşildirek, A., and F.L. Lewis, "Feedback linearization using neural networks," Automatica, vol. 31., no. 11, pp. 1659-1664, Nov. 1995.*

*Zhang, T., C.C. Hang, and S.S. Ge, "Robust adaptive control for general nonlinear systems using multilayer neural networks," preprint, 1998.*

## Chapter 7

# NN Control with Discrete-Time Tuning

*Adaptive control is an important area of research that has been pursued by many across the world (Åström and Wittenmark 1989, Landau 1979, Ljung and Söderström 1983, Narendra and Annaswamy 1989). The progress of adaptive control theory and the availability of microprocessors have led to a series of successful applications in the last several decades in the areas of robotics, aircraft control, process control and estimation, etc. One of the first industrial adaptive controllers was installed in the Port Arthur oil refinery in the 1960s by Åström.*

*Controllers are usually implemented on actual systems using microprocessors. To implement a controller on a digital microprocessor, it is necessary to express it in terms of difference equations. This is accomplished using digital or discrete-time design, discussed in this chapter and the next two. Unfortunately, most rigorous adaptive control results are available for continuous-time systems, where approaches such as the direct model-reference approach (Åström and Wittenmark 1989, Landau 1993) allow simultaneous proofs of tracking error stability and parameter error stability. Discrete-time adaptive control design is far more complex than continuous-time design, due primarily to the fact that discrete-time Lyapunov derivatives are quadratic in the state first difference, while for continuous-time systems the Lyapunov derivative is linear in the state derivative. This has led to traditional techniques where the parameter identification problem is decoupled from the control problem using the so-called certainty equivalence assumption. In this approach, two separate proofs are essentially given, one for identification and one for control. Various elegant techniques for providing a posteriori proofs of overall convergence have been offered by Ljung and others (Ljung and Söderström 1983). Even recently, several authors (e.g. Kanellakopoulos 1994)) state that for discrete-time nonlinear adaptive systems very few rigorous results exist, and one has to impose linear growth conditions on the nonlinearities to provide global stability.*

*Moreover, most adaptive control design algorithms are restricted to systems that are linear in the unknown parameters, and an often complex regression matrix must be computed for each plant. Uncertainty in the regression matrix and unmodeled disturbances may cause the performance of the adaptive controllers to deteriorate*

considerably (Åström and Wittenmark 1989).

In recent years, learning-based control using neural networks (NN) and fuzzy logic systems has emerged as an alternative to adaptive control. These systems are nonlinear in the tunable parameters, and open-loop systems can be tuned using the essential backpropagation algorithm and its variants (see Chapter 1). For closed-loop feedback systems, however, it has not been fully understood until recently how to use the learning phenomenon for training. Research in NN for control applications is now being pursued by several groups (Narendra and Parthasarathy 1990, Polycarpou and Ioannou 1991, Sanner and Slotine 1992, Sadegh 1993, Chen and Khalil, 1995). Results for continuous-time NN controllers are presented in Chapters 4 through 6 of this book. While some work in the design of discrete-time NN controllers has been performed (Sadegh 1993), until recently (Chen and Khalil 1995, Jagannathan and Lewis 1996c) there have been no results for closed-loop control of nonlinear systems with multilayer NN in the discrete-time domain that employ direct techniques to estimate the controller parameters while guaranteeing boundedness of both the tracking error and the parameter estimation error.

In this chapter a family of novel learning schemes is given for discrete-time neural networks. The traditional problems with discrete-time adaptive control are overcome by using a single Lyapunov-like proof for both the parameter identification and the control error stability. This guarantees at once both stable identification and stable tracking with no certainty equivalence assumption. In these proofs, complex manipulations are required where it is necessary to complete the square with respect to several different variables in the same proof. Along the way various other standard assumptions in discrete-time adaptive control are also overcome, including persistence of excitation, linearity-in-the-parameters, and the need for tedious computation of a regression matrix.

First we discuss design for one-layer neural nets (Jagannathan and Lewis 1996c) where the NN weights enter linearly, then design for the more complex multilayer discrete-time NN (Jagannathan and Lewis 1996b) which are nonlinear in the adjustable weights. In each case we discuss the controller structure, various weight update algorithms, and persistence of excitation definitions for NN. In keeping with the approach in Chapters 4-6, the weights are tuned on-line with no off-line learning phase needed. Finally, passivity properties of discrete-time NN controllers are covered.

## 7.1 BACKGROUND AND ERROR DYNAMICS

### 7.1.1 Neural Network Approximation Property

As discussed in Chapter 1, a general function  $f(x) \in C^{(S)}$  can be approximated using the  $n$ -layer neural network shown in Fig. 7.1.1 as

$$f(x) = W_n^T \varphi_n [W_{n-1}^T \varphi_{n-1} [\cdots \varphi_1(x(k))]] + \epsilon(k) \quad (7.1.1)$$

where  $W_n^T, W_{n-1}^T, \dots, W_2^T, W_1^T$  are constant weights and  $\varphi_i(k)$  denotes the vectors of activation functions at the instant  $k$ , with  $\epsilon(k)$  a NN functional reconstruction error vector. If there exist  $N_2$  and constant constant ideal ('target') weights such that  $\epsilon = 0$  for all  $x \in S$ , then  $f(x)$  is said to be in the functional range of the NN.

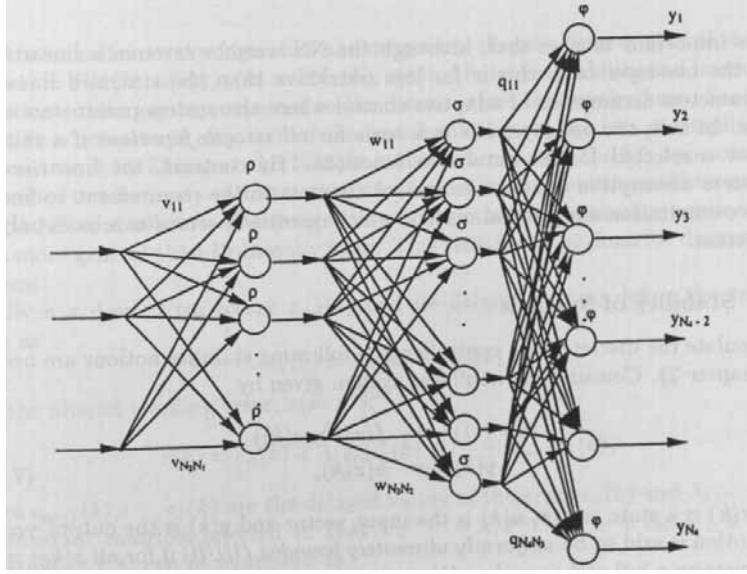


Figure 7.1.1: A multilayer neural network.

In general, given a constant real number  $\epsilon_N \geq 0$ ,  $f(x)$  is within  $\epsilon_N$  of the NN range if there exist constant weights so that for all  $x \in \mathbb{R}^n$ , (7.1.1) holds, for a sufficiently large number of hidden-layer neurons, with  $\|\epsilon\| \leq \epsilon_N$ .

In (7.1.1) for notational convenience, the vector of activation functions of the input layer at the instant  $k$  is denoted as  $\varphi_1(k) = \varphi(x(k))$ . Then, the vectors of hidden and output layer activation functions are denoted by

$$\varphi_{m+1}(k) = W_m^T \varphi_m(k); \quad \forall m = 1, \dots, n-1. \quad (7.1.2)$$

Define  $\varphi(x(k)) = \varphi_n[\dots \varphi_1(x(k))]$  and  $\hat{W}_n^T(k) = \hat{W}^T(k)$  so that the net output is defined as

$$\hat{y}(k) = \hat{W}^T(k) \varphi(x(k)). \quad (7.1.3)$$

By selecting  $n=1$  one obtains a one-layer NN. Then, for suitable approximation properties,  $\varphi(x(k))$  must be a basis (Sadegh 1993).

**Definition (Sadegh 1993):** Let  $S$  be a compact simply-connected set of  $\mathbb{R}^n$ , and  $\varphi(x(k)) : S \rightarrow \mathbb{R}^{N_2}$  be integrable and bounded. Then  $\varphi(x(k))$  is said to provide a basis for  $C^k(S)$  if

a) A constant function on  $S$  can be expressed as (7.1.3) for finite  $N_2$ .

b) The functional range of NN (7.1.3) is dense in  $C^k(S)$  for countable  $N_2$ .  $\square$

In Chapter 1 we discussed the selection of a basis for one-layer functional-link NN; for instance it is well-known in the literature that radial basis functions (Sanner and Slotine 1992) and sigmoidal functions form a basis (Sadegh 1993). In Section 7.2 we use a one-layer net (7.1.3) for controls purposes. Multilayer NN are used in Section 7.3.

*It is important to note that, although the NN weights enter in a linear fashion in the one-layer case, this is far less restrictive than the standard linear-in-the-parameters assumption of adaptive control where the system parameters enter linearly. In fact, the one-layer NN is a basis for all smooth functions if a suitable basis set is selected for the activation functions. By contrast, the linear-in-the-parameters assumption of adaptive control amounts to the requirement to find by tedious computation a regression matrix, which essentially serves as a basis only for that system.*

### 7.1.2 Stability of Systems

*To formulate the discrete-time controller, the following stability notions are needed (see Chapter 2). Consider the nonlinear system given by*

$$\begin{aligned} x(k+1) &= f(x(k), u(k)) \\ y(k) &= h(x(k)), \end{aligned} \quad (7.1.4)$$

*where  $x(k)$  is a state vector,  $u(k)$  is the input vector and  $y(k)$  is the output vector. The solution is said to be uniformly ultimately bounded (UUB) if for all  $x(k_0) = x_0$ , there exists an  $\epsilon \geq 0$  and a number  $N(\epsilon, x_0)$  such that  $\|x(k)\| \leq \epsilon$  for all  $k \geq k_0 + N$ .*

*Consider now the linear discrete time-varying system given by*

$$x(k+1) = A(k)x(k) + B(k)u(k), y(k) = C(k)x(k) \quad (7.1.5)$$

**Lemma 7.1.1 :** Define  $\psi(k_1, k_0)$  as the state-transition matrix corresponding to  $A(k)$  for system (7.1.5), i.e.,  $\psi(k_1, k_0) = \prod_{k=k_0}^{k_1-1} A(k)$ . Then if  $\|\psi(k_1, k_0)\| \leq 1, \forall k_1, k_0 \geq 0$ , system (7.1.5) is exponentially stable.

Proof: See (Sadegh 1993). □

### 7.1.3 Tracking Error Dynamics for a Class of Nonlinear Systems

*Consider an  $mn$ -th order multi-input and multi-output (MIMO) discrete-time nonlinear system, to be controlled, given in multivariable Brunovsky form (see Chapter 2) by*

$$\begin{aligned} x_1(k+1) &= x_2(k) \\ &\vdots \\ x_{n-1}(k+1) &= x_n(k) \\ x_n(k+1) &= f(x(k)) + u(k) + d(k) \end{aligned} \quad (7.1.6)$$

*with state  $x(k) = [x_1^T(k), \dots, x_n^T(k)]^T$  with  $x_i(k) \in \Re^m; i = 1, \dots, n$ , and control  $u(k) \in \Re^m$ . The nonlinear function  $f(\cdot)$  is assumed unknown. The disturbance vector acting on the system at the instant  $k$  is  $d(k) \in \Re^m$ , which we assume unknown but bounded so that  $\|d(k)\| \leq d_M$  a known constant. In this chapter we consider the class of systems where  $u(k)$  directly enters the last equation in (7.1.6). In Chapter 8 we consider the more complex case where  $x_n(k+1)$  depends on  $g(x(k))u(k)$  with the control influence function  $g(\cdot)$  unknown.*

*Many systems occur naturally in the continuous-time Brunovsky form. Unfortunately, the exact discretization of the continuous Brunovsky form does not yield the*

discrete-time Brunovsky form, but a more general discrete-time system of the form  $x(k+1) = F(x(k), u(k))$ ,  $y(k) = H(x(k), u(k))$ . Under reachability and involutivity conditions, this may be converted to the discrete-time Brunovsky form. See Chapter 2. It is not always easy to find the transformation required to accomplish this. In recent work by Zhang et al. (1998) it is shown how to use the NN approximation properties to effectively estimate this transformation for continuous-time systems, so that NN controllers can be designed for a large class of continuous nonlinear systems more general than Brunovsky form. This remains to be done for discrete-time systems.

Given a desired trajectory  $x_{nd}(k)$  and its delayed values, define the tracking error as

$$e_n(k) = x_n(k) - x_{nd}(k), \quad (7.1.7)$$

and the filtered tracking error,  $r(k) \in \mathbb{R}^m$ ,

$$r(k) = e_n(k) + \lambda_1 e_{n-1}(k) + \cdots + \lambda_{n-1} e_1(k), \quad (7.1.8)$$

where  $e_{n-1}(k), \dots, e_1(k)$  are the delayed values of the error  $e_n(k)$  and  $\lambda_1, \dots, \lambda_{n-1}$  are constant matrices selected so that  $|z^{n-1} + \lambda_1 z^{n-2} + \cdots + \lambda_{n-1}|$  is stable. Equation (7.1.8) can be expressed as

$$r(k+1) = e_n(k+1) + \lambda_1 e_{n-1}(k+1) + \cdots + \lambda_{n-1} e_1(k+1). \quad (7.1.9)$$

Using (7.1.6) in (7.1.9), the dynamics of the MIMO system (7.1.6) can be written in terms of the filtered tracking error as

$$r(k+1) = f(x(k)) - x_{nd}(k+1) + \lambda_1 e_n(k) + \cdots + \lambda_{n-1} e_2(k) + u(k) + d(k) \quad (7.1.10)$$

Define the control input  $u(k)$  as

$$u(k) = x_{nd}(k+1) - \hat{f}(x(k)) + k_v r(k) - \lambda_1 e_n(k) - \cdots - \lambda_{n-1} e_2(k) \quad (7.1.11)$$

with a diagonal gain matrix  $k_v$ , and  $\hat{f}(x(k))$  an estimate of the unknown nonlinear function  $f(x(k))$ . Then, the closed-loop error system becomes

$$r(k+1) = k_v r(k) + \tilde{f}(x(k)) + d(k) \quad (7.1.12)$$

where the functional estimation error is given by

$$\tilde{f}(x(k)) = f(x(k)) - \hat{f}(x(k)). \quad (7.1.13)$$

Note that the error system is driven by the functional estimation error and the unknown disturbances.

In this chapter, a discrete-time NN is used to provide the estimate  $\hat{f}(\cdot)$ . The error system (7.1.12) is used to focus on selecting discrete-time NN tuning algorithms that guarantee the stability of the filtered tracking error  $r(k)$ . Then, since (7.1.8), with the input considered as  $r(k)$  and the output  $e(k)$  describes a stable system, using the notion of operator gain (Slotine and Li 1991) one can guarantee that  $e(k)$  exhibits stable behavior. In fact, (7.1.8) can be rewritten as

$$\bar{x}(k+1) = A\bar{x}(k) + Br(k) \quad (7.1.14)$$

where  $\bar{x}(k) = [e_1(k), \dots, e_{n-1}]^T$ ,

$$A \equiv \begin{bmatrix} 0 & 1 & 0 \\ \cdot & \cdot & \cdot \\ -\lambda_{n-1} & \cdot & -\lambda_1 \end{bmatrix}$$

$$B \equiv \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Then one may show using the notion of operator gain that

$$\|e_1(k)\| \leq \frac{\|r(k)\|}{\Lambda_{\min}(A)}, \dots, \|e_n(k)\| \leq \frac{\|r(k)\|}{\Lambda_{\min}(A)}, \quad (7.1.15)$$

with  $\Lambda_{\min}(A)$  the minimum singular value of matrix  $A$ .

## 7.2 ONE-LAYER NEURAL NETWORK CONTROLLER DESIGN

In this section the one-layer NN is considered as a first step to bridging the gap between discrete-time adaptive control and NN control. In the next section we cover multilayer discrete-time NN for control and present our main results. In the one-layer case the tunable NN weights enter in a linear fashion. The one-layer case is treated for radial basis functions in Sanner and Slotine (1992), using a projection algorithm in Polycarpou and Ioannou (1991), and employing a delta rule (Sira-Ramirez and Zak 1991) for weight tuning for discrete-time systems in Sadegh (1993). Even though discrete-time controller design is presented in Sira-Ramirez and Zak (1991), Lyapunov stability analysis is not discussed.

In this section stability analysis by Lyapunov's direct method is performed for a family of weight tuning algorithms for a one-layer neural network developed based on the delta rule. These weight tuning paradigms yield a passive neural net, yet persistency of excitation (PE) is generally needed for suitable performance. Specifically this holds as well as for standard backpropagation in continuous-time case (Lewis et al. 1995). Unfortunately, PE cannot generally be tested for or guaranteed in a NN. Therefore, modified tuning paradigms are proposed in subsequent sections to make the NN robust so that the PE is not needed. Finally, for guaranteed stability, it is shown that the delta-rule-based-weight tuning algorithms must slow down as the NN becomes larger. By employing a projection algorithm as shown in the next section the tuning rate can be made independent of the NN size.

Assume, therefore, that there exist some constant ideal weights  $W$  for a one-layer NN so that the nonlinear function in (7.1.6) can be written as

$$f(x) = W^T \varphi(x(k)) + \epsilon(k) \quad (7.2.1)$$

where  $\varphi(x(k))$  provides a suitable basis and  $\|\epsilon(k)\| < \epsilon_N$ , with the bounding constant  $\epsilon_N$  known. Unless the net is 'minimal', the 'ideal' weights may not be unique (Sontag 1992, Sussmann 1992). The best weights may then be defined as those which minimize the supremum norm over  $S$  of  $\epsilon(k)$ . This issue is not a major concern here, as it is needed to know only the existence of such ideal weights; their

actual values are not required. This assumption is similar to Erzberger's assumptions in the linear-in-the-parameters adaptive control. The major difference is that, while Erzberger's assumptions often do not hold, the approximation properties of NN guarantee that the ideal weights always exist if  $f(x)$  is continuous over a compact set.

For suitable approximation properties, it is necessary to select a 'large enough number' of hidden-layer neurons. It is not known how to compute this number for general fully-connected NN; however, for CMAC NN the required number of hidden-layer neurons for approximation to a desired degree of accuracy is given in *Commuri and Lewis (1996)*.

### 7.2.1 Structure of the One-layer NN Controller and Error System Dynamics

Defining the NN functional estimate in the controller (7.1.11) by

$$\hat{f}(x(k)) = \hat{W}^T(k)\varphi(x(k)) \quad (7.2.2)$$

with  $\hat{W}(k)$  the current value of the weights, yields the controller structure shown in Fig. 7.2.1. The output of the plant is processed through a series of delays to obtain the past values of the output, and fed as inputs to the NN so that the nonlinear function in (7.1.6) can be suitably approximated. Thus, the NN controller derived in a straightforward manner using filtered error notions naturally provides a dynamical NN structure. Note that neither the input  $u(k)$  or its past values are needed by the NN. The next step is to determine the weight updates so that the tracking performance of the closed-loop filtered error dynamics is guaranteed.

Let  $W$  be the unknown ideal weights required for the approximation to hold in (7.2.2) and assume they are bounded by known values so that

$$\| W \| \leq W_{max}. \quad (7.2.3)$$

Then the error in the weights during estimation is given by

$$\tilde{W}(k) = W - \hat{W}(k). \quad (7.2.4)$$

**Fact 1:** The activation functions are bounded by known positive values so that  $\| \varphi(x(k)) \| \leq \varphi_{max}$  and  $\| \tilde{\varphi}(x(k)) \| \leq \tilde{\varphi}_{max}$ .

The control input  $u(k)$  is

$$u(k) = x_{nd}(k+1) - \hat{W}^T(k)\varphi(x(k)) - \lambda_1 e_n(k) - \cdots - \lambda_{n-1} e_2(k) + k_v r(k), \quad (7.2.5)$$

and the closed-loop filtered dynamics become

$$r(k+1) = k_v r(k) + \bar{e}_i(k) + \epsilon(k) + d(k), \quad (7.2.6)$$

where the identification error is defined by

$$\bar{e}_i(k) = \tilde{W}^T(k)\varphi(x(k)). \quad (7.2.7)$$

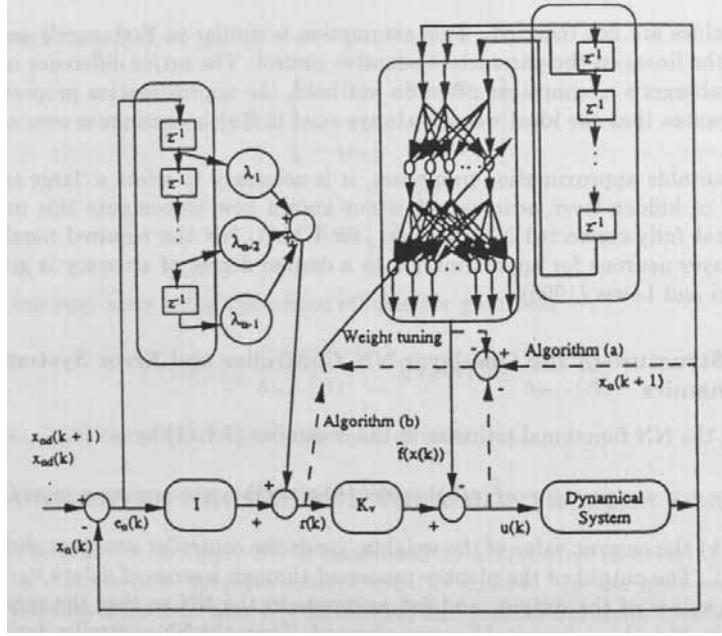


Figure 7.2.1: One-layer discrete-time neural network controller structure.

### 7.2.2 One-layer Neural Network Weight Updates

A family of NN weight tuning paradigms that guarantee the stability of the closed-loop system (7.2.6) is presented in this section. It is required to demonstrate that the tracking error  $r(k)$  is suitably small and that the NN weights  $\hat{W}(k)$  remain bounded, for then the control  $u(k)$  is bounded. To proceed further, the following machinery is needed.

**Lemma 7.2.1 :** If  $A(k) = I - \alpha\varphi(x(k))\varphi^T(x(k))$  in (7.1.5), where  $0 < \alpha < 2$  and  $\varphi(x(k))$  is a vector of basis functions, then  $\|\psi(k_1, k_0)\| < 1$  is guaranteed if there is an  $L > 0$  such that  $\sum_{k=k_0}^{k_1+L-1} \varphi(x(k))\varphi^T(x(k)) > 0$  for all  $k$ . Then, Lemma 7.1.1 guarantees the exponential stability of the system (7.1.5).

Proof: See Sadegh (1993). □

**Definition 7.2.1 :** An input sequence  $x(k)$  is said to be *persistency exciting (PE)* (Chapter 2) if there are  $\lambda > 0$  and an integer  $k_1 \geq 1$  such that

$$\lambda_{min} \left[ \sum_{k=k_0}^{k_1} \varphi(x(k))\varphi^T(x(k)) \right] > \lambda, \forall k_0 \geq 0 \quad (7.2.8)$$

where  $\lambda_{min}(P)$  represents the smallest eigenvalue of  $P$ . □

Note that PE is exactly the stability condition needed in Lemma 7.2.1.

Table 7.2.1: Discrete-Time Controller Using One-Layer Neural Net: PE Required

---

The control input is

$$u(k) = x_{nd}(k+1) - \hat{W}^T(k)\varphi(x(k)) - \lambda_1 e_n(k) - \dots - \lambda_{n-1} e_2(k) + k_v r(k),$$

The NN weight tuning is given by either

$$(a) \quad \hat{W}(k+1) = \hat{W}(k) + \alpha\varphi(x(k))\bar{f}^T(k)$$

where  $\bar{f}(k)$  is defined as the functional augmented error given by

$$\bar{f}(k) = x_n(k+1) - u(k) - \hat{f}(x(k))$$

or

$$(b) \quad \hat{W}(k+1) = \hat{W}(k) + \alpha(k)\varphi(x(k))r^T(k+1)$$

with  $\alpha > 0$  denoting constant learning rate parameters or adaptation gains.

---

In the following theorem we present the two alternative discrete-time weight tuning algorithms given in Table 7.2.1, one based on a modified functional estimation error and the other based on the filtered tracking error. Both algorithms guarantee that both the tracking error and the error in the weight estimates are bounded if a PE condition holds. (This PE requirement is relaxed in Theorem 7.2.3).

**Theorem 7.2.1 (One-Layer Discrete-Time NN Controller Requiring PE) :**

Let the desired trajectory  $x_{nd}(k)$  be bounded and the NN functional reconstruction error bound  $\epsilon_N$  and the disturbance bound  $d_M$  be known constants. Take the control input for (7.2.6) as (7.2.5) with weight tuning provided by either

$$\text{Algorithm (a)} \quad \hat{W}(k) = \hat{W}(k) + \alpha\varphi(x(k))\bar{f}^T(k) \quad (7.2.9)$$

where  $\bar{f}(k)$  is defined as the functional augmented error computed using

$$\bar{f}(k) = x_n(k+1) - u(k) - \hat{f}(x(k)) \quad (7.2.10)$$

or

$$\text{Algorithm (b)} \quad \hat{W}(k+1) = \hat{W}(k) + \alpha\varphi(x(k))r^T(k+1) \quad (7.2.11)$$

with  $\alpha > 0$  denoting constant learning rate parameters or adaptation gains.

Let the hidden-layer output vector,  $\varphi(x(k))$  be persistently exciting. Then the filtered tracking error  $r(k)$  and the error in weight estimates,  $\tilde{W}(k)$ , are UUB, with the bounds specifically given by (7.2.20) and (7.2.21) for the case of Algorithm (a), and (7.2.23) and (7.2.24) for the case of Algorithm (b), provided the following conditions hold

$$(1) \quad \alpha \|\varphi(x(k))\|^2 < 1, \quad (7.2.12)$$

$$(2) \quad k_{vmax} < \frac{1}{\sqrt{\eta}}, \quad (7.2.13)$$

where  $\eta$  is given for Algorithm (a) as

$$\eta = 1 + \frac{1}{(1 - \alpha \|\varphi(x(k))\|^2)} \quad (7.2.14)$$

and for the Algorithm (b) as

$$\eta = \frac{1}{(1 - \alpha \|\varphi(x(k))\|^2)}. \quad (7.2.15)$$

Proof:

Note: Given a trajectory and a specific activation function, the bound on the activation function,  $\|\varphi(x(k))\|$ , and the corresponding  $\eta$  can be calculated. Note that  $\eta$  is dependent upon the trajectory.

Algorithm(a): Define the Lyapunov function candidate

$$J = r^T(k)r(k) + \frac{1}{\alpha} \text{tr}(\tilde{W}^T(k)\tilde{W}(k)). \quad (7.2.16)$$

The first difference is given by

$$\begin{aligned} \Delta J &= r^T(k+1)r(k+1) - r^T(k)r(k) \\ &\quad + \frac{1}{\alpha} \text{tr}(\tilde{W}^T(k+1)\tilde{W}(k+1) - \tilde{W}^T(k)\tilde{W}(k)) \end{aligned} \quad (7.2.17)$$

Substituting (7.2.9) and (7.2.6) in (7.2.17), collecting terms together, and completing the square yields

$$\begin{aligned} \Delta J &\leq -r^T(k)[I - k_v^T k_v]r(k) + \bar{e}_i^T(k)\bar{e}_i(k) + (\epsilon(k) + d(k))^T(\epsilon(k) + d(k)) \\ &\quad + 2(k_v r(k))^T \bar{e}_i(k) + 2(k_v r(k))^T(\epsilon(k) + d(k)) + 2(\epsilon(k) + d(k))^T \bar{e}_i(k) \\ &\quad - (2 - \alpha \varphi^T(x(k))\varphi(x(k)))\bar{e}_i^T(k)\bar{e}_i(k) \\ &\quad + \alpha \varphi^T(x(k))\varphi(x(k))(\epsilon(k) + d(k))^T(\epsilon(k) + d(k)) \\ &\quad - 2(1 - \alpha \varphi^T(x(k))\varphi(x(k)))\bar{e}_i^T(k)(\epsilon(k) + d(k)). \end{aligned} \quad (7.2.18)$$

Reorganizing (7.2.18) and completing the squares for  $\bar{e}_i(k)$  yields

$$\begin{aligned} \Delta J &\leq -(1 - \eta k_{vmax}^2)[\|r(k)\|^2 - 2\frac{(\eta - 1)k_{vmax}}{(1 - \eta k_{vmax}^2)}(\epsilon_N + d_M)\|r(k)\| \\ &\quad - \frac{(\eta - 1)}{(1 - \eta k_{vmax}^2)}(\epsilon_N - d_M)^2] - (1 - \alpha \|\varphi(x(k))\|^2)\|\bar{e}_i(k)\| \\ &\quad - \frac{k_v r(k) + \alpha \|\varphi(x(k))\|^2 \|(\epsilon(k) + d(k))\|}{(1 - \alpha \|\varphi(x(k))\|^2)} \|^2 \end{aligned} \quad (7.2.19)$$

where  $\eta$  is given in (7.2.14) with  $k_{vmax}$  the maximum singular value of  $k_v$ . Since  $(\epsilon_N + d_M)$  is constant,  $\Delta J \leq 0$  as long as

$$\|r(k)\| > \frac{(\eta - 1)}{(1 - \eta k_{vmax}^2)}[k_{vmax} + \sqrt{\frac{(1 - k_{vmax}^2)}{(\eta - 1)}}]. \quad (7.2.20)$$

Note that  $|\sum_{k=k_0}^{\infty} \Delta J(k)| = |J(\infty) - J(0)| < \infty$  since  $\Delta J \leq 0$  as long as (7.2.12), (7.2.13), and (7.2.20) hold. This demonstrates that the tracking error  $r(k)$  is bounded for all  $k \geq 0$  and it remains to show that the weight estimates,  $\tilde{W}(k)$ , are bounded.

The dynamics relative to error in weight estimates using (7.2.9) are given by

$$\tilde{W}(k+1) = [I - \alpha \varphi(x(k))\varphi^T(x(k))] \tilde{W}(k) - \alpha \varphi(x(k))[\epsilon(k) + d(k)]^T, \quad (7.2.21)$$

where the functional reconstruction error  $\epsilon(k)$  and the disturbance  $d(k)$  are considered to be bounded. Applying the PE condition (7.2.8), (7.2.20), and Lemma 7.2.1, the boundedness of  $\tilde{W}(k)$  in (7.2.21), and hence of  $\hat{W}(k)$  are assured.

**Algorithm(b):** Define the Lyapunov function candidate (7.2.16). Substituting (7.2.6) and (7.2.11) in (7.2.17), collecting terms together, and completing the square for  $\bar{e}_i(k)$  yields

$$\begin{aligned}\Delta J \leq & -(1 - \eta k_{vmax}^2) [\| r(k) \|^2 - 2 \frac{\eta k_{vmax}}{(1 - \eta k_{vmax})} (\epsilon_N + d_M) \| r(k) \| \\ & - \frac{\eta}{(1 - \eta k_{vmax})} (\epsilon_N + d_M)^2] - (1 - \alpha \| \varphi(x(k)) \|^2) \| \bar{e}_i(k) \\ & - \frac{\alpha \| \varphi(x(k)) \|^2}{(1 - \eta \| \varphi(x(k)) \|^2)} [k_v r(k) + \epsilon(k) + d(k)] \|^2\end{aligned}\quad (7.2.22)$$

with  $\eta$  is given by (7.2.15).  $\Delta J \leq 0$  as long as (7.2.12) through (7.2.13) hold and this results in

$$\| r(k) \| > \frac{1}{(1 - \eta k_{vmax}^2)} (\epsilon_N + d_M) [\eta k_{vmax} + \sqrt{\eta}]. \quad (7.2.23)$$

The dynamics relative to error in weight estimates using (7.2.11) are given by

$$\tilde{W}(k+1) = [I - \alpha \varphi(x(k)) \varphi^T(x(k))] \tilde{W}(k) - \alpha \varphi(x(k)) [k_v r(k) + \epsilon(k) + d(k)]^T \quad (7.2.24)$$

where the filtered tracking error,  $r(k)$ , functional reconstruction error  $\epsilon(k)$  and the disturbance  $d(k)$  are considered to be bounded. Applying the PE condition (7.2.8), and Lemma 7.2.1 the boundedness of  $\tilde{W}(k)$  in (7.2.24), respectively, and hence of  $\hat{W}(k)$  are assured.  $\square$

*In applications, the right-hand sides of (7.2.20) and (7.2.21) for the case of Algorithm (a), or (7.2.23) and (7.2.24) for the case of Algorithm (b), may be taken as practical bounds on the norms of the error  $r(k)$  and the weight errors  $\tilde{W}(k)$ . Since the target weight values are bounded, it follows that the NN weights,  $\hat{W}(k)$  provided by the tuning algorithms are bounded; hence the control input is bounded.*

*Note from (7.2.20) and (7.2.23) that the tracking error increases with the NN reconstruction error bound  $\epsilon_N$  and the disturbance bound  $d_M$ , yet small tracking errors (but not arbitrary small) may be achieved by selecting small gains  $k_v$ . In other words, placing the closed-loop poles closer to the origin inside the unit circle forces smaller tracking errors. Selecting  $k_{vmax} = 0$  results in a deadbeat controller, but it should be avoided as it is not robust.*

*It is important to note that the problem of initializing the net weights (referred to as symmetric breaking (Rumelhart et al. 1990)) occurring in other techniques in the literature does not arise, since when  $\hat{W}(0)$  is taken as zero the PD term  $k_v r(k)$  stabilizes the plant on an interim basis for a restricted class of nonlinear systems such as robotic systems. Thus, the NN controller requires no off-line learning phase.*

*It is technically necessary to include in the proof the compact set on which the NN approximation property holds. This has not been done in this chapter since the discrete-time proofs are exceedingly complicated, and including the compact set makes them more difficult to follow. Techniques like those in Chapter 4 can be used to include the compact approximation region.*

#### **Example 7.2.1 (NN Control of Continuous-Time Nonlinear System) :**

To illustrate the performance of the NN controller, a continuous-time nonlinear system is considered and the objective is to control this MIMO system by using a one-layer NN

controller. Note that it is extremely difficult to discretize a nonlinear system and therefore offer stability proofs. Note that the NN controllers derived herein require no *a priori* knowledge of the dynamics of the nonlinear systems, unlike conventional adaptive control, nor is any initial learning phase needed.

Consider the nonlinear system described by

$$\begin{aligned}\dot{X}_1 &= X_2 \\ \dot{X}_2 &= F(X_1, X_2) + U\end{aligned}\quad (7.2.25)$$

where  $X_1 = [x_1, x_2]^T$ ,  $X_2 = [x_3, x_4]^T$ ,  $U = [u_1, u_2]^T$  and the nonlinear function in (7.2.25) is described by  $F(X_1, X_2) = [M(X_1)]^{-1}G(X_1, X_2)$ , with

$$M(X_1) = \begin{bmatrix} (b_1 + b_2)a_1^2 + b_2d_2^2 + 2b_2a_1a_2\cos(x_2) & b_2a_2^2 + b_2a_1a_2\cos(x_2) \\ b_2a_2^2 + b_2a_1a_2\cos(x_2) & b_2a_2^2 \end{bmatrix} \quad (7.2.26)$$

and

$$G(X_1, X_2) = \begin{bmatrix} -b_2a_1a_2(2x_3x_4 + x_4^2)\sin(x_2) + 9.8(b_1 + b_2)a_1\cos(x_1) + 9.8b_2a_2\cos(x_1 + x_2) \\ b_2a_1a_2x_1^2\sin(x_2) + 9.8b_2a_2\cos(x_1 + x_2) \end{bmatrix} \quad (7.2.27)$$

The parameters for the nonlinear system were selected as  $a_1 = a_2 = 1$ ,  $b_1 = b_2 = 1$ . Desired sinusoidal,  $\sin(\frac{2\pi t}{25})$ , and cosine inputs,  $\cos(\frac{2\pi t}{25})$ , were preselected for the axis 1 and 2 respectively. The continuous-time gains of the PD controller were chosen as  $k_v = \text{dia}(20, 20)$  with  $\Lambda = \text{diag}(5, 5)$  and a sampling interval of 10msec was considered. A one-layer NN was selected with 25 hidden-layer nodes. Sigmoidal activation functions were employed in all the nodes in the hidden layer. The initial conditions for  $X_1$  were chosen to be  $[0.5, 0.1]^T$ , and the weights were initialized to zero. No off-line learning is performed initially to train the networks. Fig. 7.2.2 presents the tracking response of the neural network controller with delta-rule weight tuning (7.2.11) and small  $\alpha = 0.1$ . From the figure it can be seen that the delta rule-based weight tuning performs impressively.  $\square$

### 7.2.3 Projection Algorithm

*The adaptation gains  $\alpha > 0$  are constant parameters in the update laws presented in (7.2.9) and (7.2.11). These update laws correspond to the delta rule (Rumelhart et al. 1990, Sadegh 1993), also referred to as the Widrow-Hoff rule (Widrow and Lehr 1990). This reveals that the update tuning mechanisms employing the delta rule have a major drawback. In fact, using (7.2.12), the upper bound on the adaptation gain can be obtained as*

$$\alpha < \frac{1}{\|\varphi(x(k))\|^2}, \quad (7.2.28)$$

since  $\varphi(x(k)) \in \Re^{N_2}$ , with  $N_2$  the number of hidden-layer neurons. It is evident that the upper bound on the adaptation gain depends upon the number of hidden-layer neurons. Specifically, if there  $N_2$  hidden-layer neurons and the maximum value of the each hidden-node output is taken as unity (as for the sigmoid), then the bounds on the adaptation gain to assure stability of the closed-loop system are given by

$$0 < \alpha < \frac{1}{N_2}. \quad (7.2.29)$$

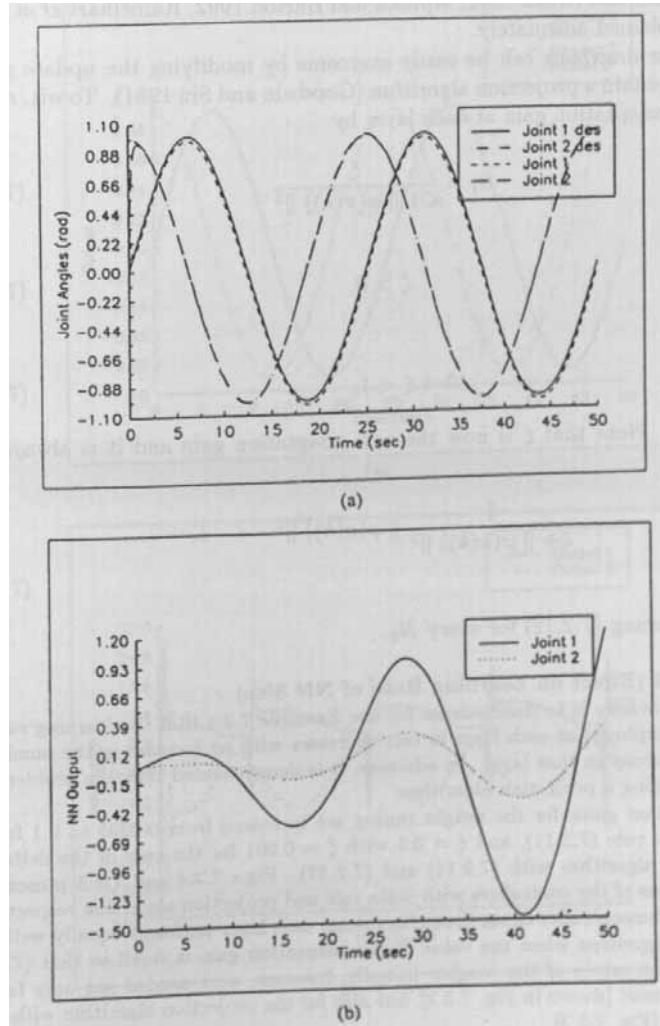


Figure 7.2.2: Response of neural network controller with delta-rule weight tuning and small  $\alpha$ . (a) Actual and desired joint angles. (b) Neural network outputs

In other words, the upper bound on the adaptation gain for the case of delta rule decreases with an increase in the number of hidden-layer nodes, so that learning must slow down for guaranteed performance. The phenomenon of large NN requiring very slow learning rates has often been encountered in the practical NN literature (Chen and Khalil 1992, Mpitsos and Burton 1992, Rumelhart et al. 1990) but never explained adequately.

This major drawback can be easily overcome by modifying the update rule at each layer to obtain a projection algorithm (Goodwin and Sin 1984). To wit, replace the constant adaptation gain at each layer by

$$\alpha_i = \frac{\xi}{\zeta + \|\varphi(x(k))\|^2}, \quad (7.2.30)$$

where

$$\zeta > 0, \quad (7.2.31)$$

and

$$0 < \xi < 1, \quad (7.2.32)$$

are constants. Note that  $\xi$  is now the new adaptation gain and it is always true that

$$\frac{\xi}{\zeta + \|\varphi(x(k))\|^2} \|\varphi(x(k))\|^2 < 1, \quad (7.2.33)$$

hence guaranteeing (7.2.12) for every  $N_2$ .

#### **Example 7.2.2 (Effect on Learning Rate of NN Size) :**

The objective here is to demonstrate for the Example 7.2.1 that the learning rate for the delta rule employed at each layer in fact decreases with an increase in the number of hidden-layer neurons in that layer. In addition, it is demonstrated that the problem can be avoided by using a projection algorithm.

The adaptation gains for the weight tuning are increased from 0.0005 to 0.1 for the case of the delta rule (7.2.11), and  $\xi = 0.5$  with  $\zeta = 0.001$  for the case of the delta rule with projection algorithm with (7.2.11) and (7.2.33). Fig.s 7.2.4 and 7.2.3 present the tracking responses of the controllers with delta rule and projection algorithm respectively. It is clear that the controller using the delta rule at each layer performs equally well with the projection algorithm when the value of the adaptation gain is small so that (7.2.12) is satisfied. Large values of the weights initially, however, were needed not only for the delta rule with small [shown in Fig. 7.2.2], but also for the projection algorithm with large adaptation gain [Fig. 7.2.3].

Note from Fig. 7.2.3 the tracking performance is extremely impressive. Fig. 7.2.4 illustrates the response of the NN controller when the delta rule is employed with the adaptation gain 0.1 to 0.51. From Fig. 7.2.4 it is evident that the weight tuning using the delta rule becomes unstable. Note that the representative weight estimates, as illustrated in Fig. 7.2.4(b), of the NN are unbounded in this case. This demonstrates that the adaptation gain in the case of delta rule must decrease with an increase in the hidden-layer neurons.  $\square$

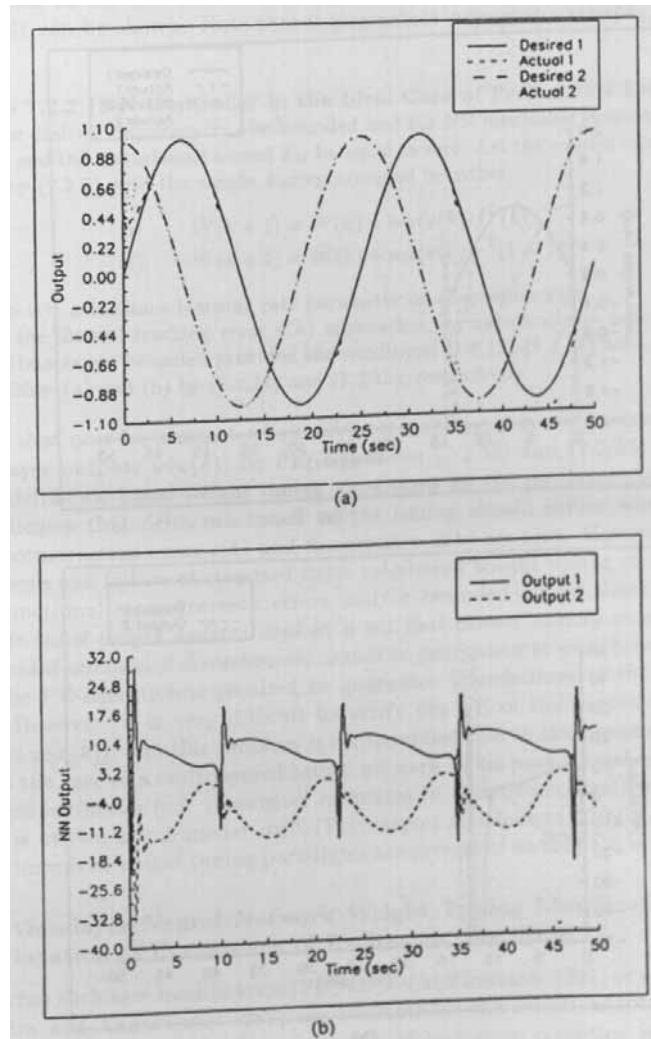


Figure 7.2.3: Response of neural network controller with delta-rule weight tuning and projection algorithm. (a) Actual and desired joint angles. (b) Neural network outputs.

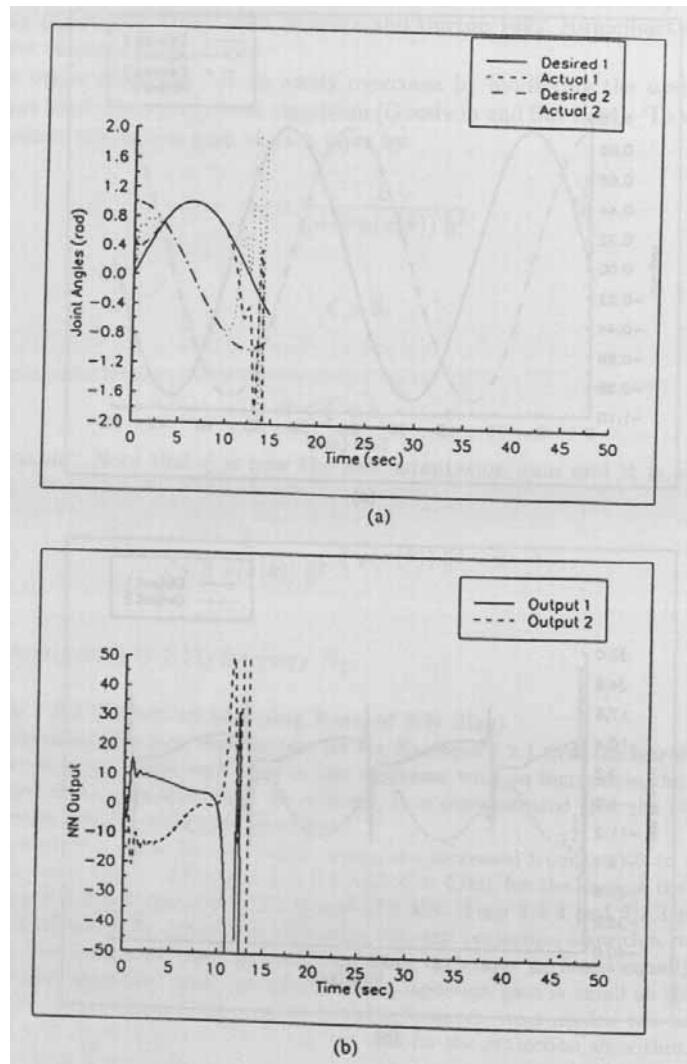


Figure 7.2.4: Response of neural network controller with delta-rule weight tuning and large  $\alpha$ . (a) Actual and desired joint angles. (b) Neural network outputs.

### 7.2.4 Ideal Case: No Disturbances or NN Reconstruction Errors

*In the idealized case of no net functional reconstruction errors with no unmodeled disturbances in the dynamics of the system, the PE condition is not needed. The next result can be shown. Note that it guarantees asymptotic stability, not simply UUB.*

**Theorem 7.2.2 (NN Controller in the Ideal Case of Perfect NN Estimation) :**

Let the desired trajectory  $x_{nd}$  be bounded and the NN functional reconstruction error bound  $\epsilon_N$  and the disturbance bound  $d_M$  be equal to zero. Let the control input for (7.1.6) be given by (7.2.5) with the weight tuning provided by either:

$$a) \quad \hat{W}(k+1) = \hat{W}(k) + \alpha\varphi(x(k))\bar{f}^T(k), \quad (7.2.34)$$

$$b) \quad \hat{W}(k+1) = \hat{W}(k) + \alpha\varphi(x(k))r^T(k+1), \quad (7.2.35)$$

where  $\alpha > 0$  is a constant learning rate parameter or adaptation gain.

Then the filtered tracking error  $r(k)$  approaches asymptotically to zero and the NN weight estimates are bounded provided the conditions (7.2.12)-(7.2.13) hold, with  $\eta$  given for Algorithm (a) and (b) by (7.2.14) and (7.2.15), respectively.  $\square$

*Note that now for guaranteed closed-loop stability, it is not necessary that the hidden-layer outputs  $\varphi(x(k))$  be PE. Equations (7.2.34) and (7.2.35) are nothing but the delta-rule-based weight tuning algorithms for the one-layer case. Theorem 7.2.2 indicates that delta-rule-based weight tuning should suffice when the functional reconstruction error  $\epsilon(k)$  and disturbance  $d(k)$  are zero. However, Theorem 7.2.1 reveals the failure of standard delta rule-based weight tuning in the presence of net functional reconstruction errors and/or bounded disturbances. Therefore, delta-rule-based weight updates used in a net that cannot exactly reconstruct  $f(x)$  with bounded unmodeled disturbances, cannot be guaranteed to yield bounded weights. Then, the PE condition is required to guarantee boundedness of the weight estimates. However, it is very difficult to verify the PE of the hidden-layer output functions  $\varphi(x(k))$ , and this problem is compounded due to the presence of hidden-layers in the case of a multilayered neural network. This possible unboundedness of the weight estimates (c.f. parameter estimates in adaptive control) when PE fails to hold is known as parameter drift (Polycarpou and Ioannou 1991). In the next section, improved weight tuning paradigms are presented so that PE is not required.*

### 7.2.5 One-layer Neural Network Weight Tuning Modification for Relaxation of Persistency of Excitation Condition

*Approaches such as  $\sigma$ -modification (Polycarpou and Ioannou 1991) or  $\epsilon$ -modification (Narendra and Annaswamy 1987) are available for the robust adaptive control of continuous systems wherein the persistency of excitation condition is not needed. A one-layer network with continuous weight update laws and  $\epsilon$ -modification was developed (Lewis et al. 1995) and the UUB of both the tracking error and the error in weight estimates was demonstrated. On the other hand, modification to the standard weight tuning mechanisms in discrete-time to avoid the necessity of PE is also investigated in (Jagannathan and Lewis 1996a).*

*In (Jagannathan and Lewis 1996c) an approach similar to  $\epsilon$ -modification was derived for discrete-time NN control. The following theorem from that paper shows*

---

Table 7.2.2: Discrete-Time Controller Using One-Layer Neural Net: PE not Required

---

*The control input is*

$$u(k) = x_{nd}(k+1) - \hat{W}^T(k)\varphi(x(k)) - \lambda_1 e_n(k) - \dots - \lambda_{n-1} e_2(k) + k_v r(k),$$

*The weight tuning is given by either*

$$(a) \quad \hat{W}(k+1) = \hat{W}(k) + \alpha\varphi(x(k))\bar{f}^T(k) - \Gamma \| I - \alpha\varphi(x(k))\varphi^T(x(k)) \| \hat{W}(k)$$

*where  $\bar{f}(k)$  is defined as the functional augmented error given by*

$$\bar{f}(k) = x_n(k+1) - u(k) - \hat{f}(x(k))$$

*or*

$$(b) \quad \hat{W}(k+1) = \hat{W}(k) + \alpha\varphi(x(k))r^T(k+1) - \Gamma \| I - \alpha\varphi(x(k))\varphi^T(x(k)) \| \hat{W}(k)$$

*with  $\alpha = \frac{\xi}{\zeta + \|\varphi(x(k))\|}$ , where  $\zeta > 0$  and  $0 < \xi < 1$  denoting learning rate parameters or adaptation gains.*

---

*two tuning algorithms that do not require persistence of excitation. The controllers derived are presented in Table 7.2.2.*

**Theorem 7.2.3 (One-Layer Discrete-Time NN Controller With No PE) :**

Assume the hypotheses presented in Theorem 7.2.1, and consider the modified weight tuning algorithms provided by either:

$$(a) \quad \hat{W}(k+1) = \hat{W}(k) + \alpha\varphi(x(k))\bar{f}^T(k) - \Gamma \| I - \alpha\varphi(x(k))\varphi^T(x(k)) \| \hat{W}(k) \quad (7.2.36)$$

$$(b) \quad \hat{W}(k+1) = \hat{W}(k) + \alpha\varphi(x(k))r^T(k+1) - \Gamma \| I - \alpha\varphi(x(k))\varphi^T(x(k)) \| \hat{W}(k) \quad (7.2.37)$$

with  $\Gamma > 0$  a design parameter. Then the filtered tracking error  $r(k)$  and the NN weight estimates  $\hat{W}(k)$  are UUB, with the bounds specifically given by (7.2.46) and (7.2.49) for the case of Algorithm (a) and (7.2.53) and (7.2.55) for the case of Algorithm (b) provided the following conditions hold:

$$(1) \quad \alpha \| \varphi(x(k)) \|^2 < 1, \quad (7.2.38)$$

$$(2) \quad 0 < \Gamma < 1, \quad (7.2.39)$$

with

$$(3) \quad k_{vmax} < \frac{1}{\sqrt{\eta}} \quad (7.2.40)$$

for the case of Algorithm (a), and

$$(3) \quad k_{vmax} < \frac{1}{\sqrt{\sigma}} \quad (7.2.41)$$

for the case of Algorithm (b), where  $\eta$  is given in (7.2.14) and  $\bar{\sigma}$  is given by

$$\bar{\sigma} = \eta + \frac{1}{(1 - \alpha \| \varphi(x(k)) \|^2)} [\Gamma^2 (1 - \alpha \| \varphi(x(k)) \|^2)^2 + 2\alpha\Gamma \| \varphi(x(k)) \|^2 (1 - \alpha \| \varphi(x(k)) \|^2)]. \quad (7.2.42)$$

Note: The parameters  $\alpha, \eta, \bar{\sigma}, \kappa$  and  $\rho$  are dependent upon the trajectory.

Proof:

Algorithm (a): Select the Lyapunov function candidate (7.2.16). Use the tracking error dynamics (7.2.6) and tuning mechanism (7.2.36) in (7.2.17) and completing the squares for  $\| \tilde{W}(k) \|$  to obtain (7.2.43)

$$\begin{aligned} \Delta J \leq & -[1 - \bar{\eta}k_{vmax}^2] [\| r(k) \|^2 - \frac{2\kappa k_{vmax}}{[1 - \eta k_{vmax}^2]} \| r(k) \| (\epsilon_N + d_M) - \frac{\rho}{[1 - \eta k_{vmax}^2]}] \\ & - [1 - \alpha \varphi^T(x(k)) \varphi(x(k))] \| \bar{e}_i(k) - \frac{1}{[1 - \alpha \varphi^T(x(k)) \varphi(x(k))]} [k_v r(k) + \\ & (\alpha \varphi^T(x(k)) \varphi(x(k)) + \Gamma \| I - \alpha \varphi(x(k)) \varphi^T(x(k)) \|) (\epsilon(k) + d(k))] \|^2 - \\ & \frac{1}{\alpha} \| I - \alpha \varphi(x(k)) \varphi^T(x(k)) \|^2 \Gamma (2 - \Gamma) [\| \tilde{W}(k) - \frac{(1 - \Gamma)}{(2 - \Gamma)} W_{max} \|^2], \end{aligned} \quad (7.2.43)$$

where

$$\kappa = [1 + \frac{1}{(1 - \alpha \| \varphi(x(k)) \|^2)} (\Gamma (1 - \alpha \| \varphi(x(k)) \|^2) + \alpha \| \varphi(x(k)) \|^2)] \quad (7.2.44)$$

$$\begin{aligned} \rho = & [1 + \alpha \| \varphi(x(k)) \|^2 + \frac{1}{(1 - \alpha \| \varphi(x(k)) \|^2)} (\Gamma (1 - \alpha \| \varphi(x(k)) \|^2) + \\ & \alpha \| \varphi(x(k)) \|^2)^2 (\epsilon_N + d_M)^2 + 2\Gamma (1 - \alpha \| \varphi(x(k)) \|^2) \| \varphi(x(k)) \|^2 \\ & W_{max} (\epsilon_N + d_M) + \frac{1}{\alpha} \frac{\Gamma}{(2 - \Gamma)} (1 - \alpha \| \varphi(x(k)) \|^2)^2 W_{max}^2] \end{aligned} \quad (7.2.45)$$

Then  $\Delta J \leq 0$  as long as (7.2.38) through (7.2.40) hold and the quadratic term of  $r(k)$  in (7.2.43) is positive, which is guaranteed when

$$\| r(k) \| > \frac{1}{(1 - \eta k_{vmax}^2)} [\kappa k_{vmax} (\epsilon_N + d_M) + \sqrt{\kappa^2 k_{vmax}^2 (\epsilon_N + d_M)^2 + \rho (1 - \eta k_{vmax}^2)}] \quad (7.2.46)$$

Similarly, completing the squares  $\| r(k) \|$  in (7.2.43) yields

$$\begin{aligned} \Delta J \leq & -[1 - \eta k_{vmax}^2] [\| r(k) \|^2 - \frac{\kappa k_{vmax}}{[1 - \eta k_{vmax}^2]} (\epsilon_N + d_M)^2 - [1 - \alpha \varphi^T(x(k)) \varphi(x(k))] \\ & \| \bar{e}_i(k) - \frac{1}{[1 - \alpha \varphi^T(x(k)) \varphi(x(k))]} [k_v r(k) + (\alpha \varphi^T(x(k)) \varphi(x(k)) \\ & + \Gamma \| I - \alpha \varphi(x(k)) \varphi^T(x(k)) \|) (\epsilon(k) + d(k))] \|^2 \\ & - \frac{1}{\alpha} \| I - \alpha \varphi(x(k)) \varphi^T(x(k)) \|^2 \Gamma (2 - \Gamma) \\ & \| \tilde{W}(k) \|^2 - 2\Gamma (1 - \Gamma) W_{max} \| \tilde{W}(k) \| - \rho] \end{aligned} \quad (7.2.47)$$

where  $\kappa$  is given in (7.2.44) and  $\rho$  is given by

$$\begin{aligned} \rho = & [\frac{k_{vmax}^2 \kappa^2}{(1 - \eta k_{vmax}^2)} (\epsilon_N + d_M)^2 + 2\Gamma (1 - \alpha \| \varphi(x(k)) \|^2) \| \varphi(x(k)) \| W_{max} (\epsilon_N + d_M) + \\ & [1 + \alpha \| \varphi(x(k)) \|^2 + \frac{1}{(1 - \alpha \| \varphi(x(k)) \|^2)} (\Gamma (1 - \alpha \| \varphi(x(k)) \|^2) + \\ & \alpha \| \varphi(x(k)) \|^2)^2 (\epsilon_N + d_M)^2] \frac{\alpha}{(1 - \alpha \| \varphi(x(k)) \|^2)^2} + \Gamma^2 W_{max}^2]. \end{aligned} \quad (7.2.48)$$

Then  $\Delta J \leq 0$  as long as (7.2.38)-(7.2.40) hold and the quadratic term for  $\|\tilde{W}(k)\|$  is positive, which is guaranteed when

$$\|\tilde{W}(k)\| > \frac{\Gamma(1-\Gamma)W_{max} + \sqrt{\Gamma^2(1-\Gamma)^2W_{max}^2 + \Gamma(2-\Gamma)\rho}}{\Gamma(2-\Gamma)}. \quad (7.2.49)$$

From (7.2.46) and (7.2.49),  $\Delta J$  is negative outside a compact set. According to a standard Lyapunov extension, it can be concluded that the tracking error  $r(k)$  and the error in weight estimates  $\tilde{W}(k)$  are UUB.

Algorithm (b): Select the Lyapunov function candidate (7.2.16). Use the tracking error dynamics (7.2.6) and tuning mechanism (7.2.37) to obtain

$$\begin{aligned} \Delta J \leq & -[1 - \bar{\sigma}k_{vmax}^2] \|r(k)\|^2 - [1 - \alpha\varphi^T(x(k))\varphi(x(k))] \|\bar{e}_i(k) \\ & - \frac{1}{(1 - \alpha\varphi^T(x(k))\varphi(x(k)))} \\ & (\alpha\varphi^T(x(k))\varphi(x(k)) + 2\Gamma \|I - \alpha\varphi(x(k))\varphi^T(x(k))\|)(k_v r(k) + \epsilon(k) + d(k))\|^2 \\ & + 2\gamma k_{vmax} \|r(k)\| + \rho - \\ & \frac{1}{\alpha} \|I - \alpha\varphi(x(k))\varphi^T(x(k))\|^2 [\Gamma(2-\Gamma) \|\tilde{W}(k)\| W_{max} - \Gamma^2 W_{max}^2], \end{aligned} \quad (7.2.50)$$

where

$$\gamma = [\eta(\epsilon_N + d_M) + \Gamma(1 - \alpha \|\varphi(x(k))\|^2) \|\varphi(x(k))\| W_{max}] \quad (7.2.51)$$

and

$$\rho = [\eta(\epsilon_N + d_M)^2 + 2\Gamma(1 - \alpha \|\varphi(x(k))\|^2) \|\varphi(x(k))\| W_{max}(\epsilon_N + d_M)]. \quad (7.2.52)$$

Completing the squares for  $\|\tilde{W}(k)\|$  in (7.2.50) similar to Algorithm (a) results in  $\Delta J \leq 0$  as long as the conditions in (7.2.38) through (7.2.41) are satisfied and with the upper bound on the tracking error given by

$$\|r(k)\| > \frac{1}{(1 - \bar{\sigma}k_{vmax}^2)} [\gamma k_{vmax} + \sqrt{\rho_1(1 - \bar{\sigma}k_{vmax}^2)}] \quad (7.2.53)$$

where

$$\rho_1 = \rho + \frac{1}{\alpha} \frac{\Gamma}{(2-\Gamma)} (1 - \alpha \|\varphi(x(k))\|^2)^2 W_{max}^2. \quad (7.2.54)$$

On the other hand, completing the squares for  $\|r(k)\|$  in (7.2.50) results in  $\Delta J \leq 0$  as long as the conditions (7.2.38)-(7.2.41) are satisfied and

$$\|\tilde{W}(k)\| > \frac{\Gamma(1-\Gamma)W_{max} + \sqrt{\Gamma^2(1-\Gamma)^2W_{max}^2 + \Gamma(2-\Gamma)\theta}}{\Gamma(2-\Gamma)}, \quad (7.2.55)$$

where

$$\theta = [\Gamma^2 W_{max}^2 + \frac{\alpha\rho_1}{(1 - \alpha \|\varphi(x(k))\|^2)^2}], \quad (7.2.56)$$

and

$$\rho_1 = \rho + \frac{\gamma^2 k_{vmax}^2}{(1 - \bar{\sigma}k_{vmax}^2)}. \quad (7.2.57)$$

In general  $\Delta J \leq 0$  in a compact set as long as (7.2.38) and (7.2.41) are satisfied and either (7.2.53) or (7.2.55) holds. According to a standard Lyapunov extension theorem (Lewis *et al.* 1993), this demonstrates that the tracking error and the error in weight estimates are UUB.  $\square$

Note that for practical purposes, (7.2.46) and (7.2.49) for the case of Algorithms (a) and (7.2.53) with (7.2.55) for the case of Algorithm (b) can be considered as bounds for  $\| r(k) \|$  and  $\| \tilde{W}(k) \|$ .

Note that the NN reconstruction error bound  $\epsilon_N$  and the bounded disturbances  $d_M$  increase the bounds on  $\| r(k) \|$  and  $\| \tilde{W}(k) \|$  in a very interesting way. Note that small tracking error bounds, but not arbitrarily small, may be achieved by placing the closed-loop poles inside the unit circle and near the origin through the selection of the largest eigenvalue,  $k_{vmax}$ . On the other hand, the NN weight error estimates are fundamentally bounded by  $W_{max}$ , the known bound on ideal weights  $W$ . The parameter  $\Gamma$  offers a design trade-off between the relative eventual magnitudes of  $\| r(k) \|$  and  $\| \tilde{W}(k) \|$ ; a smaller  $\Gamma$  yields a smaller  $\| r(k) \|$  and a larger  $\| \tilde{W}(k) \|$ , and vice versa.

The effect of adaptation gains  $\alpha$  at each layer on the weight estimation error,  $\tilde{W}(k)$ , and tracking error,  $r(k)$ , can be easily observed by using the bounds presented in (7.2.46) and (7.2.53). Large values of  $\alpha$  forces smaller tracking error but larger weight estimation error. In contrast, a small value of  $\alpha$  forces larger tracking and smaller weight estimation errors.

**Example 7.2.3 :** For Example 7.2.1, the response of the neural network controller with the improved weight tuning (7.2.37) and projection algorithm (7.2.33) is presented in Fig. 7.2.5. The design parameter  $\Gamma$  is selected as 0.01. Note that with the improved weight tuning, the output of the neural network remains bounded because the weights are guaranteed to remain bounded without the necessity of persistency of excitation. To study the contribution of the neural network, Fig. 7.2.6 shows the response of the PD controller with no neural network. It is clear that the addition of the neural network makes a significant improvement in the tracking performance.  $\square$

**Example 7.2.4 (NN Control of Discrete-Time Nonlinear System) :** Consider the first-order multi-input/multi-output discrete-time nonlinear system described by

$$X(k+1) = F(X) + U(k), \quad (7.2.58)$$

where  $X(k) = [x_1(k), x_2(k)]^T$ ,  $F(X) = \begin{bmatrix} \frac{x_2(k)}{1+x_2^2(k)} \\ \frac{x_1(k)}{1+x_1^2(k)} \end{bmatrix}$ , and  $U(k) = [u_1(k), u_2(k)]^T$ . The objective is to track a periodic step input of magnitude two units with a period of 30s. The elements of the diagonal matrix were chosen as  $k_v = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}$  and a sampling interval of 10 ms was considered. A one-layer NN was selected with twelve hidden-layer nodes. Sigmoidal activation functions were employed in all the nodes in the hidden layer. The initial conditions for the plant were chosen to be  $[1, -1]^T$ . The weights were initialized to zero with an initial threshold value of 3.0. The design parameter  $\Gamma$  is selected to be 0.01. No learning is performed initially to train the networks. The design parameters for the projection algorithm (7.2.33) were selected to be  $\xi = 0.5$  with  $\zeta = 0.001$ .

It is also found (not shown) in this example that the delta rule based weight tuning performs very well when the learning rate is small. In addition, it was also observed during simulation studies that the learning rate for delta rule based weight tuning should decrease with an increase in the number of hidden-layer neurons. As expected, however, this problem is solved by employing a projection algorithm. In this example, only results using the improved weight tuning is presented. The response of the controller with the

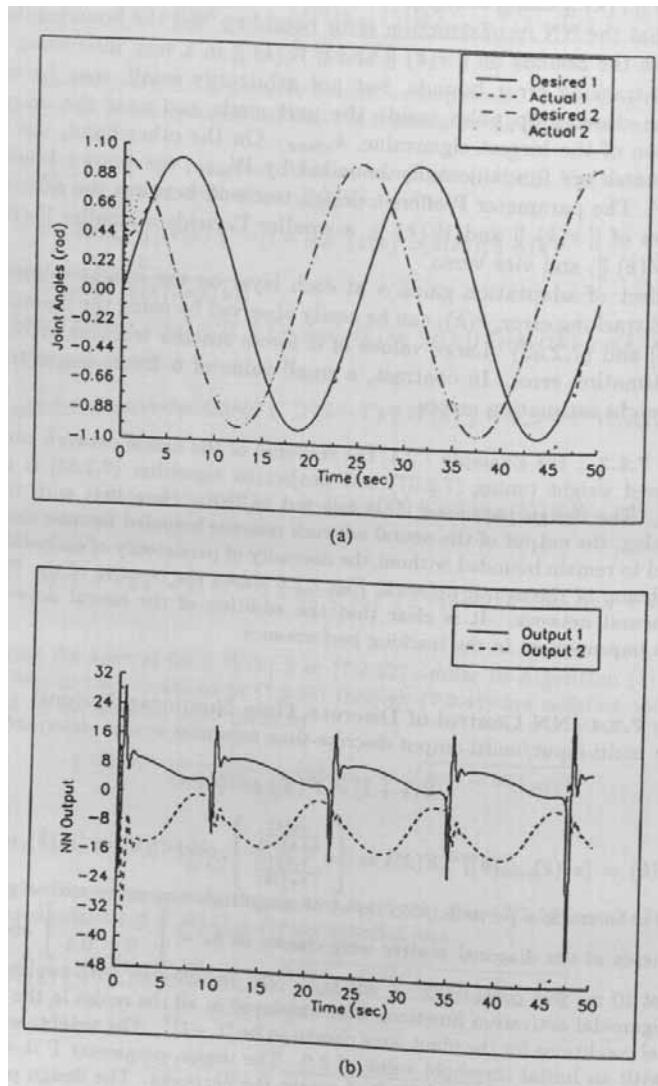


Figure 7.2.5: Response of neural network controller with improved weight tuning and projection algorithm. (a) Actual and desired joint angles. (b) Neural network outputs.

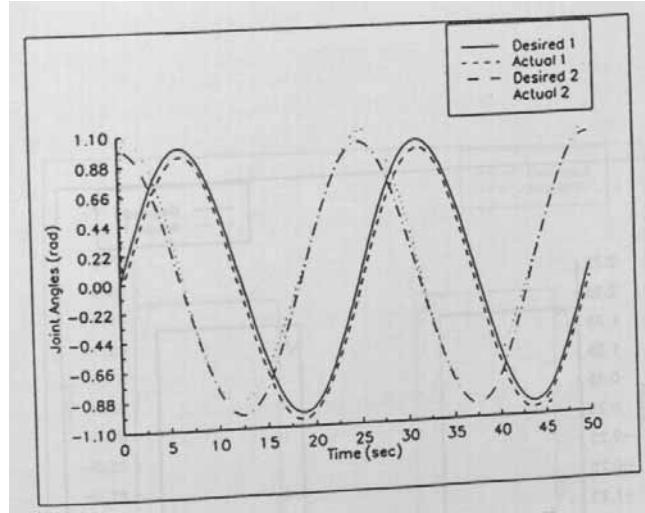


Figure 7.2.6: Response of the PD controller.

improved weight tuning (7.2.37) with (7.2.33) is shown in Fig. 7.2.7. Note from Fig. 7.2.7, as expected, the performance of the controller is extremely impressive.

Let us consider the case when a bounded disturbance given by

$$w(k) = \begin{cases} 0.0 & 0 \leq kT_m < 12 \\ 0.1 & kT_m \geq 12 \end{cases} \quad (7.2.59)$$

is acting on the plant at the time instant  $t$ . Fig. 7.2.8 presents the tracking response of NN controllers with the improved weight tuning and projection algorithm. The magnitude of the disturbance can be increased, however, the value should be bounded. The value shown in (7.2.59) is employed for simulation purposes only. It can be seen from the figure that the bounded disturbance induces bounded tracking errors at the output of the plant. From the results, it can be inferred that the bounds presented and the theoretical claims were justified through simulation studies both in continuous and discrete-time.  $\square$

### 7.3 MULTILAYER NEURAL NETWORK CONTROLLER DESIGN

*In this section a three-layer NN is considered initially and stability analysis is carried out for the closed-loop system (7.1.12). Thereafter, all the stability analysis presented for a three-layer NN is shown to be easily extended for a multilayer NN having an arbitrary number of hidden layers. In this section, stability analysis by Lyapunov's direct method is performed for a family of multilayer NN weight tuning algorithms using a delta rule in each layer. These weight tuning paradigms yield a passive NN, yet persistency of excitation (PE) is generally needed for suitable performance. Specifically, this holds as well as for standard backpropagation in the continuous-time case (Lewis et al. 1995) and the one-layer discrete-time case (Jagannathan and Lewis 1996c). Unfortunately, PE cannot generally be tested for*

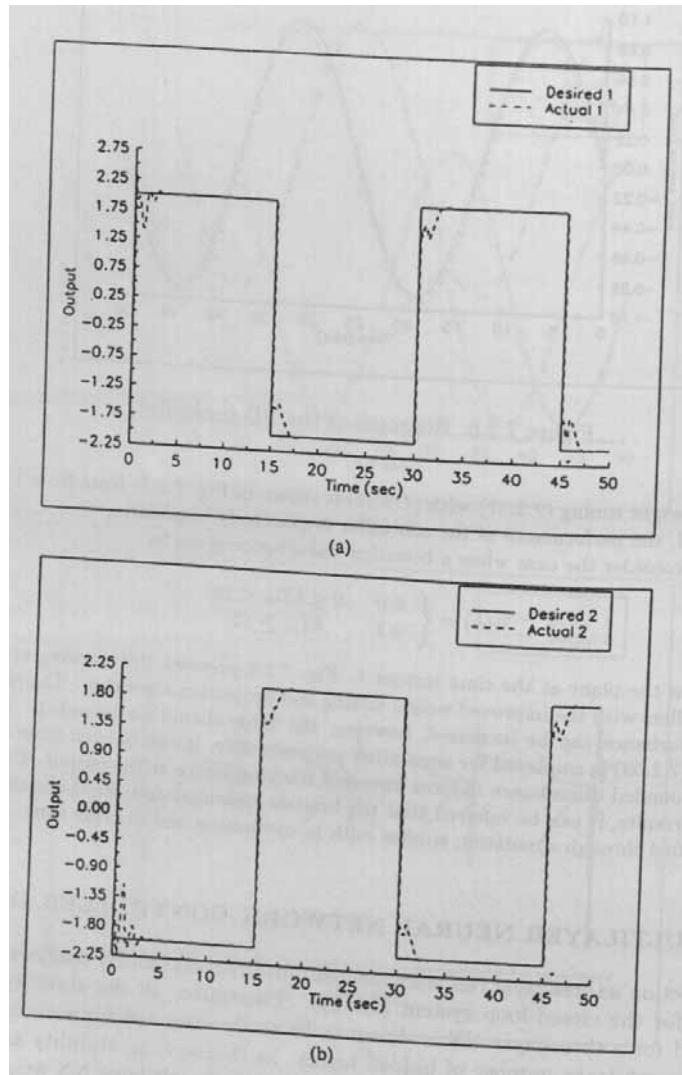


Figure 7.2.7: Response of neural network controller with improved weight tuning and projection algorithm. (a) Desired and actual state 1. (b) Desired and actual state 2.

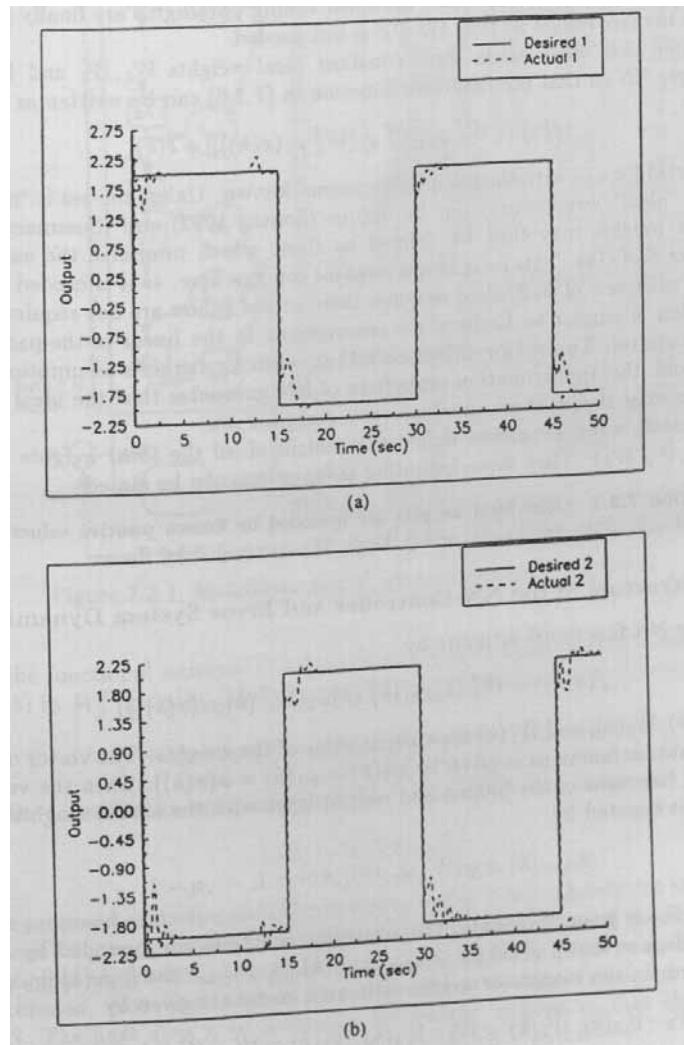


Figure 7.2.8: Response of neural network controller with improved weight tuning in the presence of bounded disturbances. (a) Desired and actual state 1. (b) Desired and actual state 2.

or guaranteed in a NN. In addition, for guaranteed stability, the weight tuning using the delta rule at each layer must slow down as the NN becomes larger. This is a problem often noted in NN literature (Mpitsos and Burton 1992).

By employing a projection algorithm, it is shown that the tuning rate can be made independent of the NN size. Modified tuning paradigms are finally proposed to make the NN robust so that the PE is not needed.

Assume that there exists some constant ideal weights  $W_1, W_2$  and  $W_3$  for a three-layer NN so that the nonlinear function in (7.1.6) can be written as

$$f(x) = W_3^T \varphi_3[W_2^T \varphi_2[W_1^T \varphi_1(x(k))]] + \epsilon(k) \quad (7.3.1)$$

where  $\|\epsilon(k)\| < \epsilon_N$ , with the bounding constant known. Unless the net is “minimal” suitable “ideal” weights may not be unique (Sontag 1992) and (Sussmann 1992). The best weights may then be defined as those which minimize the supremum norm over  $S$  of  $\epsilon(k)$ . This issue is not a major concern here, as it is needed to know only the existence of such ideal weights; their actual values are not required. This assumption is similar to Erzberger’s assumptions in the linear-in-the-parameters adaptive control. This major difference is that, while Erzberger’s assumptions often do not hold, the approximation properties of NN guarantee that the ideal weights do always exist if  $f(x)$  is continuous over a compact set.

For notational convenience define the matrix of all the ideal weights as  $Z = \text{diag}\{W_1, W_2, W_3\}$ . Then some bounding assumptions can be stated.

**Assumption 7.3.1 :** The ideal weights are bounded by known positive values so that  $\|W_1\| \leq W_{1max}$ ,  $\|W_2\| \leq W_{2max}$ , and  $\|W_3\| \leq W_{3max}$ , or  $\|Z\| \leq Z_{max}$ .

### 7.3.1 Structure of the NN Controller and Error System Dynamics

Define the NN functional estimate by

$$\hat{f}(x(k)) = \hat{W}_3^T(k) \varphi_3(\hat{W}_2^T(k) \varphi_2(\hat{W}_1^T(k) \varphi_1(x(k)))) \quad (7.3.2)$$

with  $\hat{W}_3(k)$ ,  $\hat{W}_2(k)$ , and  $\hat{W}_1(k)$  the current value of the weights. The vector of input layer activation functions is given by  $\hat{\varphi}_1(k) = \varphi_1(k) = \varphi(x(k))$ . Then the vector of activation functions of the hidden and output layer with the actual weights at the instant  $k$  is denoted by

$$\hat{\varphi}_{m+1}(k) = \varphi(\hat{W}_m^T \hat{\varphi}_m(k)); \forall m = 1, \dots, n-1. \quad (7.3.3)$$

**Fact 2:** For a given trajectory, the activation functions are bounded by known positive values so that  $\|\varphi_1(k)\| \leq \varphi_{1max}$ ,  $\|\varphi_2(k)\| \leq \varphi_{2max}$ , and  $\|\varphi_3(k)\| \leq \varphi_{3max}$ .

The error in the weights or weight estimation errors are given by

$$\tilde{W}_3(k) = W_3 - \hat{W}_3(k), \tilde{W}_2(k) = W_2 - \hat{W}_2(k), \tilde{W}_1(k) = W_1 - \hat{W}_1(k), \tilde{Z}(k) = Z - \hat{Z}(k), \quad (7.3.4)$$

where  $Z(k) = \text{diag}\{\hat{W}_1(k), \hat{W}_2(k), \hat{W}_3(k)\}$ , and the hidden-layer output errors are defined as

$$\tilde{\varphi}_2(k) = \varphi_2 - \hat{\varphi}_2(k), \tilde{\varphi}_3(k) = \varphi_3 - \hat{\varphi}_3(k). \quad (7.3.5)$$

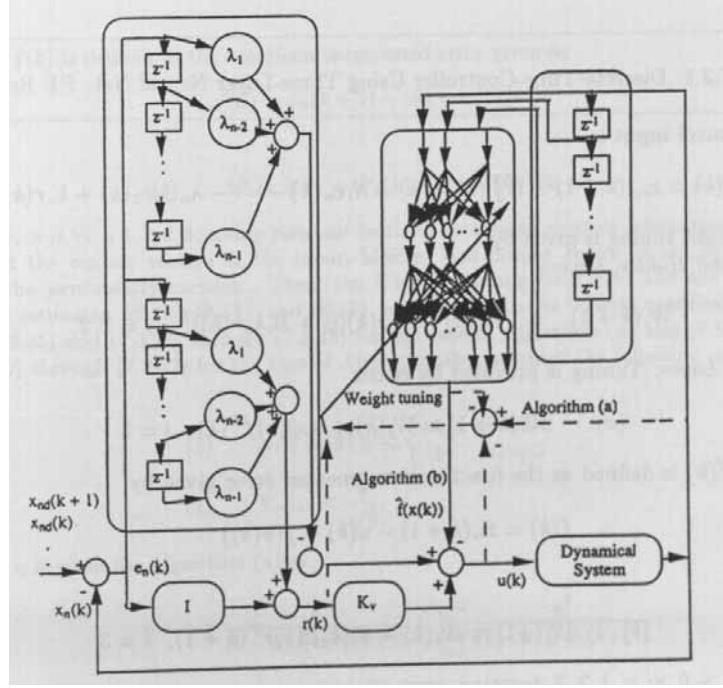


Figure 7.3.1: Multilayer neural network controller structure.

Select the control input  $u(k)$  by

$$u(k) = x_{nd}(k+1) - \hat{W}_3^T(k)\hat{\varphi}_3(k) - \lambda_1 e_n(k) - \dots - \lambda_{n-1} e_2(k) + k_v r(k) \quad (7.3.6)$$

where the functional estimate (7.3.2) is provided by a three-layer NN and denoted in (7.3.6) by  $\hat{W}_3^T(k)\hat{\varphi}_3(k)$ . Then, the closed-loop filtered dynamics become

$$r(k+1) = k_v r(k) + \bar{e}_i(k) + W_3^T \tilde{\varphi}_3(k) + \epsilon(k) + d(k) \quad (7.3.7)$$

where the identification error is defined by

$$\bar{e}_i(k) = \tilde{W}_3^T(k)\hat{\varphi}_3(k). \quad (7.3.8)$$

The proposed controller structure is shown in Fig. 7.3.1. The output of the plant is processed through a series of delays to obtain the past values of the output, and fed as inputs to the NN so that the nonlinear function in (7.1.6) can be suitably approximated. Note that neither the input  $u(k)$  or its past values are needed by the NN. The next step is to determine the weight updates so that the tracking performance of the closed-loop filtered error dynamics is guaranteed.

### 7.3.2 Multilayer Neural Network Weight Updates

A family of NN weight tuning paradigms that guarantee the stability of the closed-loop system (7.3.7) is presented in this section. It is required to demonstrate that the

Table 7.3.1: Discrete-Time Controller Using Three-Layer Neural Net: PE Required

The control input is

$$u(k) = x_{nd}(k+1) - \hat{W}_3^T(k)\hat{\varphi}_3(k) - \lambda_1 e_n(k) - \dots - \lambda_{n-1} e_2(k) + k_v r(k)$$

The weight tuning is given by:

Input and Hidden Layers:

$$\hat{W}_i(k+1) = \hat{W}_i(k) - \alpha_i \hat{\varphi}(k)[\hat{y}_i + B_i k_v r(k)]^T, \quad i = 1, 2,$$

Output Layer: Tuning is provided by either

$$(a) \quad \hat{W}_i(k+1) = \hat{W}_i(k) + \alpha_i \hat{\varphi}(k) \bar{f}^T(k), \quad i = 3$$

where  $\bar{f}(k)$  is defined as the functional augmented error given by

$$\bar{f}(k) = x_n(k+1) - u(k) - \hat{f}(x(k))$$

or

$$(b) \quad \hat{W}_i(+1) = \hat{W}_i(k) + \alpha_i(k) \hat{\varphi}(k) r^T(k+1), \quad i = 3$$

with  $\alpha_i > 0, \forall i = 1, 2, 3$  denoting constant learning rate parameters or adaptation gains and  $\|B_i\| \leq \kappa_i, \forall i = 1, 2$ .

tracking error  $r(k)$  is suitably small and that the NN weights  $\hat{W}_3^T(k), \hat{W}_2^T(k), \hat{W}_1^T(k)$ , remain bounded, for then the control  $u(k)$  is bounded.

### 7.3.2.1 NN Controller with Three Layers

Here we consider the three-layer NN case. The two tuning algorithms in Table 7.3.1 are derived in the next Theorem. One algorithm is based on a modified functional estimation error and the other on the filtered tracking error. These algorithms require PE, which is defined for a multilayer NN during the proof (Jagannathan and Lewis 1996b).

#### Theorem 7.3.1 (Three-Layer NN Controller Requiring PE) :

Let the desired trajectory  $x_{nd}(k)$  be bounded and the NN functional reconstruction error and the disturbance bounds,  $\epsilon_N, d_M$ , respectively, be known constants. Take the control input for (7.1.6) as (7.3.6) and the weight tuning provided for the input and hidden layers as

$$\hat{W}_1(k+1) = \hat{W}_1(k) - \alpha_1 \hat{\varphi}_1(k)[\hat{y}_1(k) + B_1 k_v r(k)]^T, \quad (7.3.9)$$

$$\hat{W}_2(k+1) = \hat{W}_2(k) - \alpha_2 \hat{\varphi}_2(k)[\hat{y}_2(k) + B_2 k_v r(k)]^T, \quad (7.3.10)$$

where  $\hat{y}_i(k) = \hat{W}_i^T(k)\hat{\varphi}_i(k)$ , and

$$\|B_i\| \leq \kappa_i, i = 1, 2. \quad (7.3.11)$$

Take the weight tuning update for the output layer as either

$$(a) \quad \hat{W}_3(k+1) = \hat{W}_3(k) + \alpha_3 \hat{\varphi}_3(k) \bar{f}^T(k), \quad (7.3.12)$$

where  $\bar{f}(k)$  is defined as the functional augmented error given by

$$\bar{f}(k) = x_n(k+1) - u(k) - \hat{f}(x(k)) \quad (7.3.13)$$

or as

$$(b) \quad \hat{W}_3(k+1) = \hat{W}_3(k) + \alpha_3 \hat{\varphi}_3(k) r^T(k+1) \quad (7.3.14)$$

with  $\alpha_i > 0, \forall i = 1, 2, 3$  denoting constant learning rate parameters or adaptation gains.

Let the output vectors of the input, hidden, and output layers,  $\hat{\varphi}_1(k)$ ,  $\hat{\varphi}_2(k)$ , and  $\hat{\varphi}_3(k)$  be persistently exciting. Then, the filtered tracking error  $r(k)$  and the error in weight estimates,  $\tilde{W}_1(k)$ ,  $\tilde{W}_2(k)$ , and  $\tilde{W}_3(k)$ , are UUB, with the bounds specifically given by (7.3.25) and (7.3.26) through (7.3.28) for the case of Algorithm (a) and (7.3.25) and (7.3.30) through (7.3.32) for the case of Algorithm (b), provided the following conditions hold:

$$(1) \quad \alpha_i \| \hat{\varphi}_i(k) \|^2 < \begin{cases} 2 & \forall i = 1, 2 \\ 1 & \forall i = 3 \end{cases} \quad (7.3.15)$$

$$(2) \quad k_{vmax} < \frac{1}{\sqrt{\eta}}, \quad (7.3.16)$$

where  $\eta$  is given for Algorithm (a) as

$$\eta = 1 + \frac{1}{(1 - \alpha_3 \| \hat{\varphi}_3(k) \|^2)} + \sum_{i=1}^2 \frac{\kappa_i^2}{(2 - \alpha_i \| \hat{\varphi}_i(k) \|^2)}, \quad (7.3.17)$$

and for the Algorithm (b) as

$$\eta = \frac{1}{(1 - \alpha_3 \| \hat{\varphi}_3(k) \|^2)} + \sum_{i=1}^2 \frac{\kappa_i^2}{(2 - \alpha_i \| \hat{\varphi}_i(k) \|^2)}. \quad (7.3.18)$$

Note: The design parameters  $\eta, \gamma, \alpha_i; \forall i = 1, 2, 3$  and  $\rho$  are dependent upon the trajectory.

Proof:

Algorithm(a): Define the Lyapunov function candidate

$$J = r^T(k)r(k) + \frac{1}{\alpha_1} \text{tr}(\tilde{W}_1^T(k)\tilde{W}_1(k)) + \frac{1}{\alpha_2} \text{tr}(\tilde{W}_2^T(k)\tilde{W}_2(k)) + \frac{1}{\alpha_3} \text{tr}(\tilde{W}_3^T(k)\tilde{W}_3(k)). \quad (7.3.19)$$

The first difference is given by

$$\begin{aligned} \Delta J &= r^T(k+1)r(k+1) - r^T(k)r(k) + \frac{1}{\alpha_1} \text{tr}(\tilde{W}_1^T(k+1)\tilde{W}_1(k+1) - \tilde{W}_1^T(k)\tilde{W}_1(k)) + \\ &\quad \frac{1}{\alpha_2} \text{tr}(\tilde{W}_2^T(k+1)\tilde{W}_2(k+1) - \tilde{W}_2^T(k)\tilde{W}_2(k)) + \\ &\quad \frac{1}{\alpha_3} \text{tr}(\tilde{W}_3^T(k+1)\tilde{W}_3(k+1) - \tilde{W}_3^T(k)\tilde{W}_3(k)) \end{aligned} \quad (7.3.20)$$

Substituting (7.3.7) and (7.3.9) - (7.3.13) in (7.3.20), collecting terms together, and completing the square yields

$$\begin{aligned}
\Delta J \leq & -r^T(k)[I - k_v^T k_v]r(k) + 2(k_v r(k))^T [W_3^T \tilde{\varphi}_3(k) + \epsilon(k) + d(k)] + (1 + \alpha_3 \hat{\varphi}_3^T \hat{\varphi}_3) \\
& [W_3^T \tilde{\varphi}_3(k) + \epsilon(k) + d(k)] + \frac{W_{1max}^2 \|\hat{\varphi}_1(k)\|^2}{(2 - \alpha_1 \|\hat{\varphi}_1(k)\|^2)} + \frac{W_{2max}^2 \|\hat{\varphi}_2(k)\|^2}{(2 - \alpha_2 \|\hat{\varphi}_2(k)\|^2)} \\
& - [1 - \alpha_3 \hat{\varphi}_3^T(k) \hat{\varphi}_3(k)] [\bar{e}_i(k) - \frac{k_v r(k) + \alpha_3 \hat{\varphi}_3^T(k) \hat{\varphi}_3(k) (W_3^T \tilde{\varphi}_3(k) + \epsilon(k) + d(k))}{(1 - \alpha_3 \hat{\varphi}_3^T(k) \hat{\varphi}_3(k))}]^T \\
& [\bar{e}_i(k) - \frac{k_v r(k) + \alpha_3 \hat{\varphi}_3^T(k) \hat{\varphi}_3(k) (W_3^T \tilde{\varphi}_3(k) + \epsilon(k) + d(k))}{(1 - \alpha_3 \hat{\varphi}_3^T(k) \hat{\varphi}_3(k))}] + \\
& \frac{1}{(1 - \alpha_3 \hat{\varphi}_3^T(k) \hat{\varphi}_3(k))} [k_v r(k) + \alpha_3 \hat{\varphi}_3^T(k) \hat{\varphi}_3(k) (W_3^T \tilde{\varphi}_3(k) + \epsilon(k) + d(k))]^T \\
& [k_v r(k) + \alpha_3 \hat{\varphi}_3^T(k) \hat{\varphi}_3(k) (W_3^T \tilde{\varphi}_3(k) + \epsilon(k) + d(k))] \\
& - (2 - \alpha_1 \hat{\varphi}_1^T(k) \hat{\varphi}_1(k)) \|\hat{W}_1^T(k) \hat{\varphi}_1(k) - \frac{(1 - \alpha_1 \hat{\varphi}_1^T(k) \hat{\varphi}_1(k))}{(2 - \alpha_1 \hat{\varphi}_1^T(k) \hat{\varphi}_1(k))} (W_1^T \hat{\varphi}_1(k) + k_v r(k))\|^2 \\
& - (2 - \alpha_2 \hat{\varphi}_2^T(k) \hat{\varphi}_2(k)) \|\hat{W}_2^T(k) \hat{\varphi}_2(k) - \frac{(1 - \alpha_2 \hat{\varphi}_2^T(k) \hat{\varphi}_2(k))}{(2 - \alpha_2 \hat{\varphi}_2^T(k) \hat{\varphi}_2(k))} (W_2^T \hat{\varphi}_2(k) + k_v r(k))\|^2 \\
& + 2k_{vmax} \|r(k)\| \sum_{i=1}^2 \frac{\kappa_i \|\varphi_i(k)\| W_{imax}}{(2 - \alpha_i \hat{\varphi}_i^T(k) \hat{\varphi}_i(k))} + k_{vmax}^2 \|r(k)\|^2 \sum_{i=1}^2 \frac{\kappa_i^2}{(2 - \alpha_i \hat{\varphi}_i^T(k) \hat{\varphi}_i(k))} \\
\end{aligned} \tag{7.3.21}$$

$$\begin{aligned}
\Delta J \leq & -(1 - \eta k_{vmax}^2) [\|r(k)\|^2 - 2 \frac{\gamma k_{vmax}}{(1 - \eta k_{vmax}^2)} - \frac{\rho}{(1 - \eta k_{vmax}^2)}] - [1 - \alpha_3 \hat{\varphi}_3^T(k) \hat{\varphi}_3(k)] \\
& \|\bar{e}_i(k) - \frac{k_v r(k) + \alpha_3 \hat{\varphi}_3^T(k) \hat{\varphi}_3(k) (W_3^T \tilde{\varphi}_3(k) + \epsilon(k) + d(k))}{(1 - \alpha_3 \hat{\varphi}_3^T(k) \hat{\varphi}_3(k))}\|^2 \\
& - (2 - \alpha_1 \hat{\varphi}_1^T(k) \hat{\varphi}_1(k)) \|\hat{W}_1^T(k) \hat{\varphi}_1(k) - \frac{(1 - \alpha_1 \hat{\varphi}_1^T(k) \hat{\varphi}_1(k))}{(2 - \alpha_1 \hat{\varphi}_1^T(k) \hat{\varphi}_1(k))} (W_1^T \hat{\varphi}_1(k) + k_v r(k))\|^2 \\
& - (2 - \alpha_2 \hat{\varphi}_2^T(k) \hat{\varphi}_2(k)) \|\hat{W}_2^T(k) \hat{\varphi}_2(k) - \frac{(1 - \alpha_2 \hat{\varphi}_2^T(k) \hat{\varphi}_2(k))}{(2 - \alpha_2 \hat{\varphi}_2^T(k) \hat{\varphi}_2(k))} (W_2^T \hat{\varphi}_2(k) + k_v r(k))\|^2
\end{aligned} \tag{7.3.22}$$

where  $\eta$  is given in (7.3.17) with  $k_{vmax}$  the maximum singular value of  $k_v$  and

$$\gamma = \frac{1}{(1 - \alpha_3 \|\hat{\varphi}_3(k)\|^2)} [W_{3max} \tilde{\varphi}_{3max} + \epsilon_N + d_M] + \sum_{i=1}^2 \frac{\kappa_i \|\hat{\varphi}_i(k)\| W_{imax}}{(2 - \alpha_i \|\hat{\varphi}_i(k)\|^2)} \tag{7.3.23}$$

and

$$\rho = [\frac{(W_{3max} \|\hat{\varphi}_3(k)\| + \epsilon_N + d_M)^2}{(1 - \alpha_3 \|\hat{\varphi}_3(k)\|^2)} + \sum_{i=1}^2 \frac{\|\hat{\varphi}_i(k)\|^2 W_{imax}^2}{(2 - \alpha_i \|\hat{\varphi}_i(k)\|^2)}]. \tag{7.3.24}$$

Since  $(\epsilon_N + d_M)$  is constant,  $\Delta J \leq 0$  as long as

$$\|r(k)\| > \frac{1}{(1 - \eta k_{vmax}^2)} [\gamma k_{vmax} + \sqrt{\gamma^2 k_{vmax}^2 + \rho(1 - \eta k_{vmax}^2)}]. \tag{7.3.25}$$

Note that  $|\sum_{k=k_0}^{\infty} \Delta J(k)| = |J(\infty) - J(0)| < \infty$  since  $\Delta J \leq 0$  as long as (7.3.15), (7.3.16), and (7.3.25) hold. This demonstrates that the tracking error  $r(k)$  is bounded for all  $k \geq 0$  and it remains to show that the weight estimates  $\tilde{W}_1(k)$ ,  $\tilde{W}_2(k)$  and  $\tilde{W}_3(k)$ , are bounded.

The dynamics relative to error in weight estimates using (7.3.9), (7.3.10) and (7.3.12) are given by

$$\tilde{W}_1(k+1) = [I - \alpha_1 \hat{\varphi}_1(k) \hat{\varphi}_1^T(k)] \tilde{W}_1(k) + \alpha_1 \hat{\varphi}_1(k) [W_1^T \hat{\varphi}_1(k) + B_1 k_v r(k)]^T, \quad (7.3.26)$$

$$\tilde{W}_2(k+1) = [I - \alpha_2 \hat{\varphi}_2(k) \hat{\varphi}_2^T(k)] \tilde{W}_2(k) + \alpha_2 \hat{\varphi}_2(k) [W_2^T \hat{\varphi}_2(k) + B_2 k_v r(k)]^T, \quad (7.3.27)$$

$$\tilde{W}_3(k+1) = [I - \alpha_3 \hat{\varphi}_3(k) \hat{\varphi}_3^T(k)] \tilde{W}_3(k) - \alpha_3 \hat{\varphi}_3(k) [W_3^T \hat{\varphi}_3(k) + \epsilon(k) + d(k)]^T \quad (7.3.28)$$

where the functional reconstruction error  $\epsilon(k)$  and the disturbance  $d(k)$  are considered to be bounded. Applying the PE condition (7.2.8), and using tracking error bound (7.3.25), and Lemma 7.2.1 for the cases  $\varphi(k) = \hat{\varphi}_i(k); \forall i = 1, \dots, 3$ , the boundedness of  $\tilde{W}_1(k)$ ,  $\tilde{W}_2(k)$ , and  $\tilde{W}_3(k)$  in (7.3.26)-(7.3.28), respectively, and hence of  $\hat{W}_1(k)$ ,  $\hat{W}_2(k)$ , and  $\hat{W}_3(k)$  are assured.

**Algorithm(b):** Define a Lyapunov function candidate as in (7.3.19). Substituting (7.3.7), (7.3.9)-(7.3.10), and (7.3.14) in (7.3.20), collecting terms together, and completing the square yields

$$\begin{aligned} \Delta J \leq & -(1 - \eta k_{vmax}^2) [\| r(k) \|^2 - 2 \frac{\gamma k_{vmax}}{(1 - \eta k_{vmax})} \| r(k) \| - \frac{\rho}{(1 - \eta k_{vmax})}] - \\ & [1 - \alpha_3 \hat{\varphi}_3^T(k) \hat{\varphi}_3(k)] \| \bar{e}_i(k) - \frac{\alpha_3 \hat{\varphi}_3^T(k) \hat{\varphi}_3(k) (k_v r(k) + W_3^T \hat{\varphi}_3(k) + \epsilon(k) + d(k))}{(1 - \alpha_3 \hat{\varphi}_3^T(k) \hat{\varphi}_3(k))} \|^2 \\ & - (2 - \alpha_1 \hat{\varphi}_1^T(k) \hat{\varphi}_1(k)) \| \hat{W}_1^T(k) \hat{\varphi}_1(k) - \frac{(1 - \alpha_1 \hat{\varphi}_1^T(k) \hat{\varphi}_1(k))}{(2 - \alpha_1 \hat{\varphi}_1^T(k) \hat{\varphi}_1(k))} (W_1^T \hat{\varphi}_1(k) + k_v r(k)) \|^2 \\ & - (2 - \alpha_2 \hat{\varphi}_2^T(k) \hat{\varphi}_2(k)) \| \hat{W}_2^T(k) \hat{\varphi}_2(k) - \frac{(1 - \alpha_2 \hat{\varphi}_2^T(k) \hat{\varphi}_2(k))}{(2 - \alpha_2 \hat{\varphi}_2^T(k) \hat{\varphi}_2(k))} (W_2^T \hat{\varphi}_2(k) + k_v r(k)) \|^2 \end{aligned} \quad (7.3.29)$$

with  $\eta$  is given by (7.3.18) and  $\rho$  is given in (7.3.24).  $\Delta J \leq 0$  as long as (7.3.15) and (7.3.16) hold and this results in (7.3.25).

The dynamics relative to error in weight estimates using (7.3.9), (7.3.10), and (7.3.14) are given by

$$\tilde{W}_1(k+1) = [I - \alpha_1 \hat{\varphi}_1(k) \hat{\varphi}_1^T(k)] \tilde{W}_1(k) + \alpha_1 \hat{\varphi}_1(k) [W_1^T \hat{\varphi}_1(k) + B_1 k_v r(k)]^T, \quad (7.3.30)$$

$$\tilde{W}_2(k+1) = [I - \alpha_2 \hat{\varphi}_2(k) \hat{\varphi}_2^T(k)] \tilde{W}_2(k) + \alpha_2 \hat{\varphi}_2(k) [W_2^T \hat{\varphi}_2(k) + B_2 k_v r(k)]^T, \quad (7.3.31)$$

$$\tilde{W}_3(k+1) = [I - \alpha_3 \hat{\varphi}_3(k) \hat{\varphi}_3^T(k)] \tilde{W}_3(k) - \alpha_3 \hat{\varphi}_3(k) [W_3^T \hat{\varphi}_3(k) + \epsilon(k) + d(k)]^T \quad (7.3.32)$$

where the functional reconstruction error  $\epsilon(k)$  and the disturbance  $d(k)$  are considered to be bounded. Applying the PE condition (7.2.8), tracking error bound (7.3.25), and Lemma 7.2.1 for the cases  $\varphi(k) = \hat{\varphi}_i(k); \forall i = 1, \dots, 3$ , the boundedness of  $\tilde{W}_1(k)$ ,  $\tilde{W}_2(k)$ , and  $\tilde{W}_3(k)$  in (7.3.30)-(7.3.32), respectively, and hence of  $\hat{W}_1(k)$ ,  $\hat{W}_2(k)$ , and  $\hat{W}_3(k)$  are assured.  $\square$

In applications, the right-hand sides of (7.3.25) and (7.3.26)-(7.3.28) for the case of Algorithm (a) or (7.3.25) and (7.3.30)-(7.3.32) for the case of Algorithm (b) may be taken as practical bounds on the norms of the error  $r(k)$  and the weight errors  $\tilde{W}_1(k)$ ,  $\tilde{W}_2(k)$ , and  $\tilde{W}_3(k)$ . Since the target values are bounded, it follows that the NN weights,  $\hat{W}_1(k)$ ,  $\hat{W}_2(k)$ , and  $\hat{W}_3(k)$  provided by the tuning algorithms are bounded; hence the control input is bounded.

One of the drawbacks of the available methodologies that guarantee the tracking and bounded weights (Chen and Khalil 1995, Lewis et al. 1995) is the lack of generalization of stability analysis to NN having an arbitrary number of hidden layers.

The reason is partly due to the problem of defining and verifying the persistency of excitation for a multilayered NN. For instance, in the case of a three-layered continuous-time NN (Lewis et al. 1995), the PE conditions are not easy to derive as one is faced with the observability properties of a certain bilinear system. According to the proof presented above, however, the PE for a multilayer NN is defined as the PE (in the sense of Definition) of all the hidden layer inputs  $\hat{\varphi}_i(k); \forall i = 1, \dots, n$ .

The effect of the tracking error on the hidden layer weights and the position of the closed-loop poles can be observed through the design parameters,  $\kappa_i, i = 1, \dots, n$ . These parameters weight the tracking error which in turn drive the hidden layer weight updates. A large value of  $\kappa_i$  will increase the hidden layer weights and the bounding constants  $\gamma$  and  $\rho$ . This in turn causes the position of the poles to move closer to the origin resulting in an increase in the input thereby forcing the tracking error to converge to the compact set as fast as possible.

It is important to note that the problem of initializing the net weights (referred to as symmetric breaking (Rumelhart et al. 1990)) occurring in other techniques in the literature does not arise, since when  $\hat{Z}(0)$  is taken as zero the PD term of  $k_v r(k)$  stabilizes the plant, on an interim basis, for instance in certain restricted class of nonlinear systems such as robotic systems. Thus, the NN controller requires no learning phase.

#### **Example 7.3.1 (Nonlinear System Using Multilayer Neural Network) :**

Consider the nonlinear system described by Example 7.2.1 with the parameters for the nonlinear system selected as  $a_1 = a_2 = 1, b_1 = b_2 = 1$ . Desired sinusoidal,  $\sin(\frac{2\pi t}{25})$ , and cosine inputs,  $\cos(\frac{2\pi t}{25})$ , were preselected for the axis 1 and 2 respectively. The continuous-time gains of the PD controller were chosen as  $k_v = \text{dia}(5, 5)$  and a sampling interval of 10 msec was considered. A three-layer NN was selected with four input, six hidden and two output nodes. Sigmoidal activation functions were employed in all the nodes in the hidden layer. The initial conditions for  $X_1$  were chosen to be  $[0.5, 0.1]^T$ , and the weights were initialized to zero. No off-line learning is performed initially to train the networks. The elements of the  $B_i, \forall i = 1, 2$  are chosen to be 0.1. Fig. 7.3.2 presents the tracking response of the neural net controller with  $\alpha_1 = 0.2, \alpha_2 = 0.01$ , and  $\alpha_3 = 0.1$  using (7.3.9), (7.3.10) and (7.3.14). From the figure, the tracking response is impressive.  $\square$

#### 7.3.2.2 NN Controller with Multiple Hidden Layers

The following presents the generalization of the above to NN having an arbitrary number of hidden layers when the NN functional reconstruction error and the unmodeled disturbances are nonzero but bounded by known constants. Only the updates are presented here but the proof is left as an exercise for the reader.

#### **Theorem 7.3.2 (Multilayer NN Controller Requiring PE) :**

Assume the hypotheses presented before and take the weight tuning provided for the input and hidden layers as

$$\hat{W}_i(k+1) = \hat{W}_i(k) - \alpha_i \hat{\varphi}_i(k) [\hat{g}_i(k) + B_i k_v r(k)]^T, \forall i = 1, \dots, n-1 \quad (7.3.33)$$

where

$$\hat{y}_i(k) = \hat{W}_i^T(k) \hat{\varphi}_i(k); \forall i = 1, \dots, n-1 \quad (7.3.34)$$

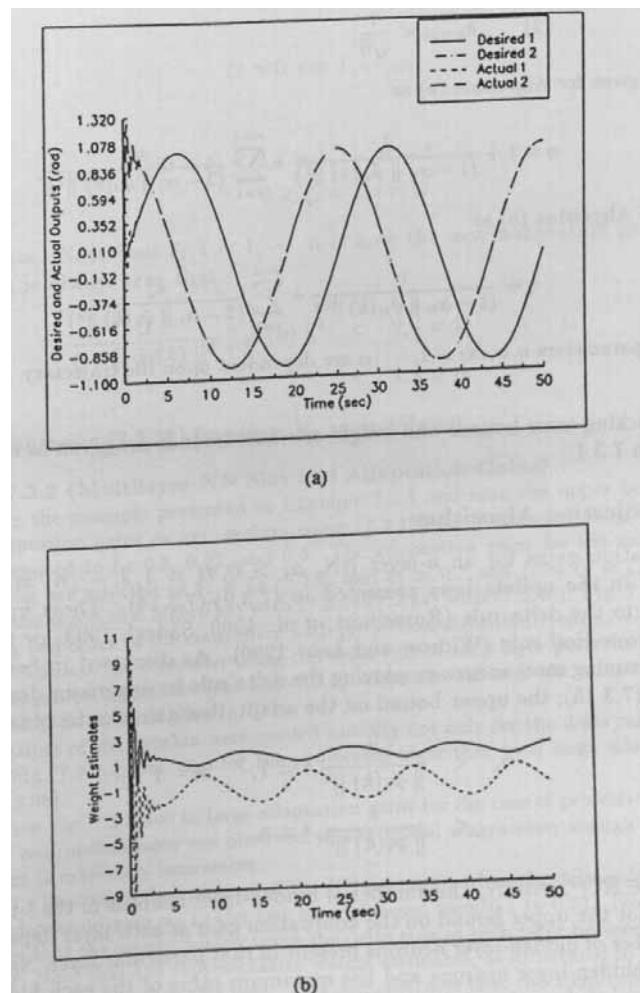


Figure 7.3.2: Response of multilayer neural network controller with delta-rule weight tuning and small  $\alpha_3$ . (a) Desired and actual trajectory. (b) Representative weight estimates.

and the weight tuning update for the output layer is given either by

$$(a) \quad \hat{W}_n(k+1) = \hat{W}_n(k) + \alpha_n \hat{\varphi}_n(k) \bar{f}^T(k) \quad (7.3.35)$$

$$(b) \quad \hat{W}_n(k+1) = \hat{W}_n(k) + \alpha_n \hat{\varphi}_n(k) r^T(k+1) \quad (7.3.36)$$

with  $\alpha_i > 0, \forall i = 1, 2, \dots, n$  denoting constant learning rate parameters or adaptation gains.

Let the output vectors of the input, hidden, and output layers,  $\hat{\varphi}_i(k); \forall i = 1, 2, \dots, n$ , be persistently exciting, the filtered tracking error  $r(k)$  and the error in weight estimates,  $\tilde{W}_i(k); \forall i = 1, 2, \dots, n$ , are UUB, provided the following conditions hold:

$$(1) \quad \alpha_i \|\hat{\varphi}_i(k)\|^2 < \begin{cases} 2 & \forall i = 1, \dots, n-1 \\ 1 & \forall i = n \end{cases} \quad (7.3.37)$$

$$(2) \quad k_{vmax} < \frac{1}{\sqrt{\eta}}, \quad (7.3.38)$$

where  $\eta$  is given for Algorithm (a) as

$$\eta = 1 + \frac{1}{(1 - \alpha_n \|\hat{\varphi}_n(k)\|^2)} + \sum_{i=1}^{n-1} \frac{\kappa_i^2}{(2 - \alpha_i \|\hat{\varphi}_i(k)\|^2)}, \quad (7.3.39)$$

and for the Algorithm (b) as

$$\eta = \frac{1}{(1 - \alpha_n \|\hat{\varphi}_n(k)\|^2)} + \sum_{i=1}^{n-1} \frac{\kappa_i^2}{(2 - \alpha_i \|\hat{\varphi}_i(k)\|^2)}. \quad (7.3.40)$$

Note: The parameters  $\eta, \alpha_i; \forall i = 1, \dots, n$  are dependent upon the trajectory.  $\square$

*The tracking error bounds and weight estimate bounds are given as in the proof of Theorem 7.3.1.*

### 7.3.3 Projection Algorithm

*The adaptation gains for an  $n$ -layer NN,  $\alpha_i > 0, \forall i = 1, 2, \dots, n$ , are constant parameters in the update laws presented in (7.3.9)-(7.3.14). These update laws correspond to the delta rule (Rumelhart et al. 1990, Sadegh 1993) or referred to as the Widrow-Hoff rule (Widrow and Lehr 1990). As discussed in Section 7.2.3 the update tuning mechanisms employing the delta rule have a major drawback. In fact, using (7.3.15), the upper bound on the adaptation gain can be obtained as*

$$\begin{aligned} \alpha_i &< \frac{2}{\|\hat{\varphi}_i(k)\|^2}, i = 1, \dots, n-1 \\ &< \frac{1}{\|\hat{\varphi}_i(k)\|^2}, i = n. \end{aligned} \quad (7.3.41)$$

*Since  $\hat{\varphi}_i(k) \in \Re^{N_{pi}}$ , with  $N_p$  the number of hidden-layer neurons in the  $i$ -th layer, it is evident that the upper bound on the adaptation gain at each layer depends upon the the number of hidden-layer neurons present in that particular layer. Specifically, if there  $N_p$  hidden-layer neurons and the maximum value of the each hidden-node*

output in the  $i$ -th layer is taken as unity (as for the sigmoid), then the bounds on the adaptation gain to assure stability of the closed-loop system are given by

$$\begin{aligned} 0 < \alpha_i &< \frac{2}{N_p}, \forall i = 1, \dots, n-1 \\ 0 < \alpha_i &< \frac{1}{N_p}, i = n. \end{aligned} \quad (7.3.42)$$

This major drawback can be easily overcome by modifying the update rule at each layer to obtain a projection algorithm (Goodwin and Sin 1984). To wit, replace the constant adaptation gain at each layer by

$$\alpha_i = \frac{\xi_i}{\zeta_i + \|\hat{\varphi}_i(k)\|^2}, i = 1, \dots, n \quad (7.3.43)$$

where

$$\zeta_i > 0, i = 1, \dots, n \quad (7.3.44)$$

and

$$\begin{aligned} 0 < \xi_i &< 2, i = 1, \dots, n-1 \\ 0 < \xi_i &< 1, i = n. \end{aligned} \quad (7.3.45)$$

are constants. Note that  $\xi_i, i = 1, \dots, n$  is now the new adaptation gain at each layer and it is always true that

$$\begin{aligned} \frac{\xi_i}{\zeta_i + \|\hat{\varphi}_i(k)\|^2} \|\hat{\varphi}_i(k)\|^2 &< 2, i = 1, \dots, n-1, \\ &< 1, i = n. \end{aligned} \quad (7.3.46)$$

hence guaranteeing (7.3.37) for every  $N_p$  at each layer.

**Example 7.3.2 (Multilayer NN Size and Adaptation Gains) :**

Consider the example presented in Example 7.3.1 and note the upper bound on the allowed adaptation gains  $\alpha_1, \alpha_2$ , and  $\alpha_3$  using (7.3.15) for the case of delta rule at each layer is computed to be 0.5, 0.32, and 0.5. The adaptation gains for the multilayer NN weight tuning are selected as  $\xi_1 = \xi_2 = 1.0$ , and  $\xi_3 = 0.7$  and  $\zeta_1 = \zeta_2 = \zeta_3 = 0.01$  for the case of the projection algorithm with (7.3.9)-(7.3.14) with (7.3.46). Fig. 7.3.3 presents the tracking responses of the controllers with projection algorithm respectively. It is clear from Fig. 7.3.2 that the controller using the delta rule at each layer performs equally well with the projection algorithm (see Fig. 7.3.3) when the value of the adaptation gain is small so that (7.3.15) is satisfied.

Large values of the weights were needed initially not only for the delta rule with small [shown in Fig. 7.3.2b], but also for the projection algorithm with large adaptation gain [see Fig. 7.3.3b].

Note from Fig. 7.3.3 due to large adaptation gains for the case of projection algorithm, overshoots and undershoots are observed in the initial stages even though the tracking performance is extremely impressive.

Fig. 7.3.4 illustrates the response of the NN controller when the delta rule is employed with the adaptation gain  $\alpha_3$  in the last layer changed from 0.1 to 0.51. From Fig. 7.3.4, it is evident that the weight tuning using the delta rule at each layer becomes unstable at

$t = 1.08\text{sec}$ . Note that the representative weight estimates, as illustrated in Fig. 7.3.4(b), of the NN are unbounded in this case. This demonstrates that the adaptation gain in the case of delta rule at each layer must decrease with an increase in the hidden-layer neurons. In fact, the theoretical limit implied by (7.3.15) in this case for this sampling interval is, so that this bound appears to be a tight bound in general.

The performance of the NN controller was investigated while varying the adaptation gains at the output layer for the case of projection algorithm. Fig. 7.3.5(a) and 7.3.5(b) show the tracking response and some NN representative weight estimates of the NN controller with  $\xi_3 = 0.1$ . As expected, the overshoots and undershoots have been totally eliminated but there appears to be a slight degradation in the performance. In other words at very low adaptation gains, overshoots and undershoots are not seen but there appears a slight degradation in the tracking performance with a slow and smooth convergence. On the other hand, at large adaptation gains overshoots and undershoots are observed with a good tracking performance. As the adaptation gains are further increased, the oscillatory behavior continues to increase and finally the system becomes unstable. In other words, from the bounds presented in (7.3.15), as the adaptation gains are increased the margin of stability continues to decrease and at large adaptation gains (close to one for instance for the last layer) the system becomes unstable. Thus, the simulation results conducted corroborate with the bounds presented in the previous sections.  $\square$

### 7.3.4 Multilayer Neural Network Weight Tuning Modification for Relaxation of Persistency of Excitation Condition

An approach similar to  $\epsilon$ -modification is derived for one-layer (i.e. linear) discrete-time NN (Jagannathan and Lewis 1996b). In this section, the modified weight tuning algorithms discussed for a one-layer discrete-time NN in Section 7.2.5 are extended to a multilayer discrete-time NN so that PE is not needed. The results of this section present tuning algorithms that overcome the need for PE in the case of multilayered NN. First we consider three-layer NN, then multiple-layer NN.

#### 7.3.4.1 Three-Layer NN Controller Not Requiring PE

The next theorem proves the stability of the NN controllers in Table 7.3.2, which have tuning algorithms augmented to avoid PE.

**Theorem 7.3.3 (Three-Layer NN Controller Not Requiring PE) :**

Assume the hypotheses presented above and now consider the modified weight tuning algorithms provided for the input and hidden layers as

$$\hat{W}_1(k+1) = \hat{W}_1(k) - \alpha_1 \hat{\varphi}_1(k) [\hat{y}_1(k) + B_1 k_v r(k)]^T - \Gamma \| I - \alpha_1 \hat{\varphi}_1(k) \hat{\varphi}_1^T(k) \| \hat{W}_1(k) \quad (7.3.47)$$

$$\hat{W}_2(k+1) = \hat{W}_2(k) - \alpha_2 \hat{\varphi}_2(k) [\hat{y}_2(k) + B_2 k_v r(k)]^T - \Gamma \| I - \alpha_2 \hat{\varphi}_2(k) \hat{\varphi}_2^T(k) \| \hat{W}_2(k). \quad (7.3.48)$$

Let the weight update for the output layer given by either

$$(a) \quad \hat{W}_3(k+1) = \hat{W}_3(k) + \alpha_3 \hat{\varphi}_3(k) \tilde{f}^T(k) - \Gamma \| I - \alpha_3 \hat{\varphi}_3(k) \hat{\varphi}_3^T(k) \| \hat{W}_3(k) \quad (7.3.49)$$

$$(b) \quad \hat{W}_3(k+1) = \hat{W}_3(k) + \alpha_3 \hat{\varphi}_3(k) r^T(k+1) - \Gamma \| I - \alpha_3 \hat{\varphi}_3(k) \hat{\varphi}_3^T(k) \| \hat{W}_3(k) \quad (7.3.50)$$

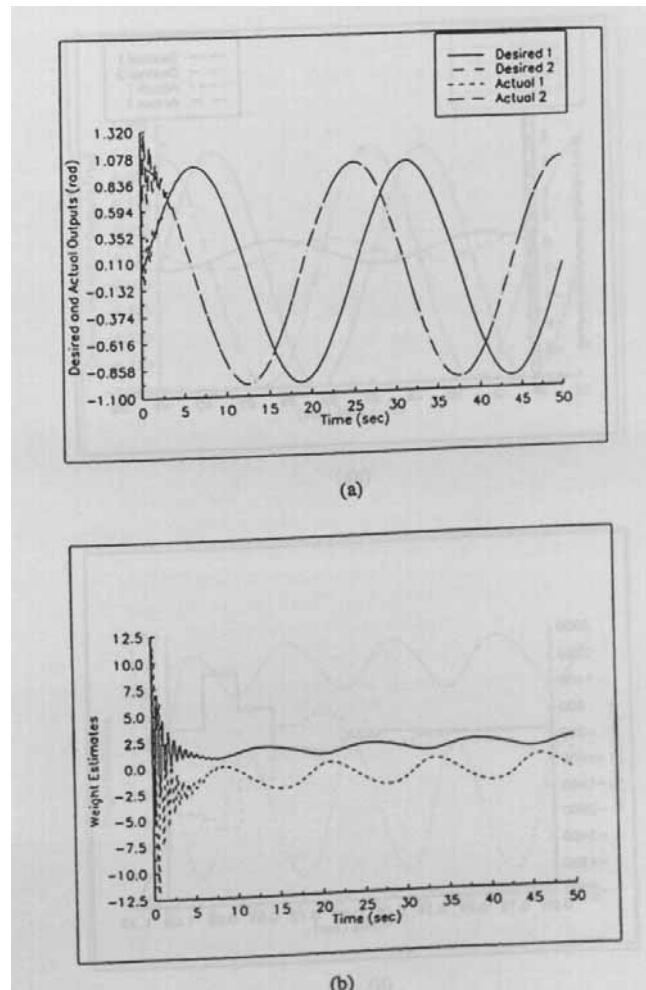


Figure 7.3.3: Response of multilayer neural network controller with delta-rule weight tuning and projection algorithm with large  $\alpha_3$ . (a) Desired and actual trajectory. (b) Representative weight estimates.

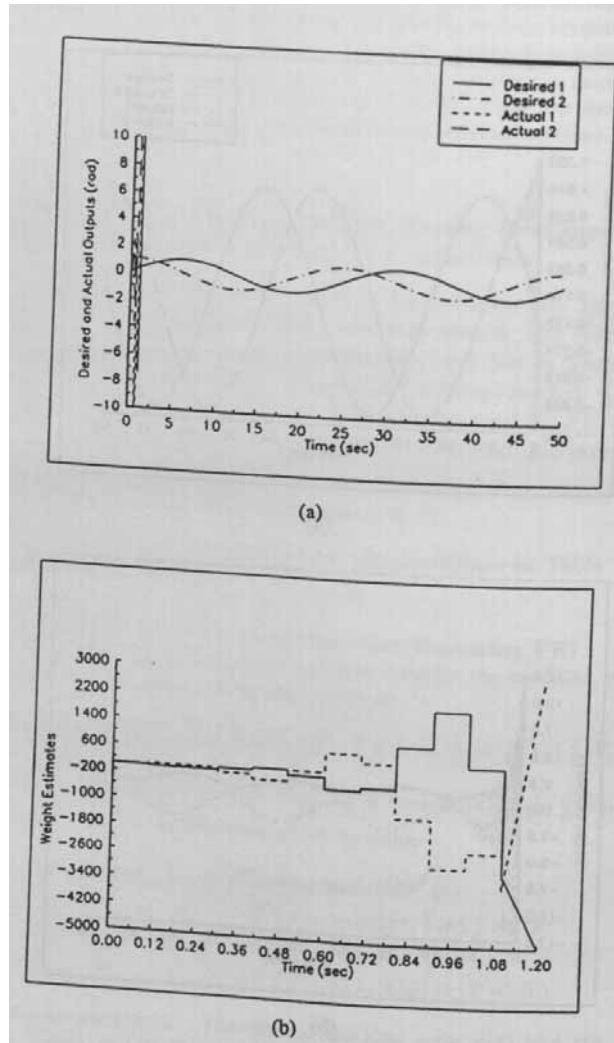


Figure 7.3.4: Response of multilayer neural network controller with delta-rule weight tuning and large  $\alpha_3$ . (a) Desired and actual trajectory. (b) Representative weight estimates.

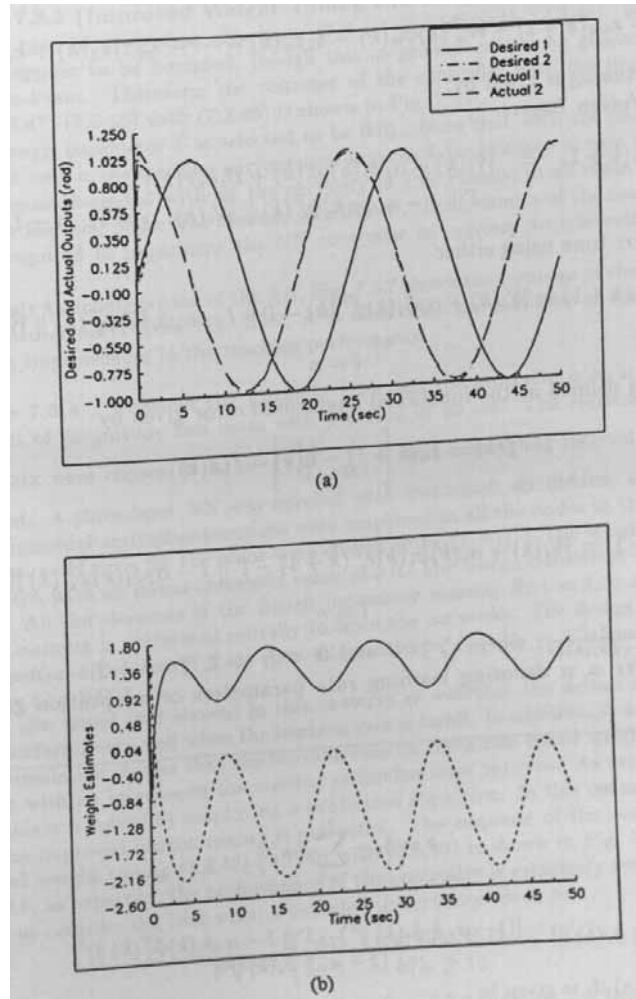


Figure 7.3.5: Response of multilayer neural network controller with delta-rule weight tuning and projection algorithm with small  $\alpha_3$ . (a) Desired and actual trajectory. (b) Representative weight estimates.

---

Table 7.3.2: Discrete-Time Controller Using Three-Layer Neural Net: PE not Required

---

*The control input is*

$$u(k) = x_{nd}(k+1) - \hat{W}_n^T(k)\hat{\varphi}_n(k) - \lambda_1 e_n(k) - \dots - \lambda_{n-1} e_2(k) + k_v r(k)$$

*The weight tuning is given by:*

*Input and Hidden Layers:*

$$\begin{aligned} \hat{W}_i(k+1) &= \hat{W}_i(k) - \alpha_i \hat{\varphi}_i(k)[\hat{y}_i(k) + B_i k_v r(k)]^T \\ &\quad - \Gamma \| I - \alpha_i \hat{\varphi}_i(k) \hat{\varphi}_i^T(k) \| \hat{W}_i(k), \quad i = 1, \dots, n-1, \end{aligned}$$

*Output Layer: tune using either*

$$(a) \quad \hat{W}_i(k+1) = \hat{W}_i(k) + \alpha_i \hat{\varphi}_i(k) \bar{f}^T(k) - \Gamma \| I - \alpha_i \hat{\varphi}_i(k) \hat{\varphi}_i^T(k) \| \hat{W}_i(k),$$

$$i = n$$

where  $\bar{f}(k)$  is defined as the functional augmented error given by

$$\bar{f}(k) = x_n(k+1) - u(k) - \hat{f}(x(k))$$

or

$$(b) \quad \hat{W}_i(k+1) = \hat{W}_i(k) + \alpha_i(k) \hat{\varphi}_i(k) r^T(k+1) - \Gamma \| I - \alpha_i \hat{\varphi}_i(k) \hat{\varphi}_i^T(k) \| \hat{W}_i(k),$$

$$i = n$$

with  $\alpha_i = \frac{\xi_i}{\zeta_i + \|\hat{\varphi}_i(k)\|^2}$ , where  $\zeta_i > 0$  and  $0 < \xi_i < 2$ ,  $\forall i = 1, 2, \dots, n-1$ , and  $0 < \xi_i < 1$ ,  $\forall i = n$  denoting learning rate parameters or adaptation gains and  $\|B_i\| \leq \kappa_i$ .

---

with  $\Gamma > 0$  a design parameter. The filtered tracking error  $r(k)$  and the NN weight estimates  $\hat{W}_1(k)$ ,  $\hat{W}_2(k)$ , and  $\hat{W}_3(k)$  are UUB provided the following conditions hold:

$$(1) \quad \alpha_i \|\hat{\varphi}_i(k)\|^2 < \begin{cases} 2 & \forall i = 1, 2, \\ 1 & \forall i = 3, \end{cases} \quad (7.3.51)$$

$$(2) \quad 0 < \Gamma < 1, \quad (7.3.52)$$

$$(3) \quad k_{vmax} < \frac{1}{\sqrt{\sigma}} \quad (7.3.53)$$

where  $\bar{\sigma}$  is given by

$$\bar{\sigma} = \beta_3 + \sum_{i=1}^2 \beta_i \kappa_i^2 \quad (7.3.54)$$

with

$$\beta_i = \alpha_i \|\hat{\varphi}_i(k)\|^2 + \frac{[(1 - \alpha_i \|\hat{\varphi}_i(k)\|^2) - \Gamma \|I - \alpha_i \hat{\varphi}_i(k) \hat{\varphi}_i^T(k)\|]^2}{(2 - \alpha_i \|\hat{\varphi}_i(k)\|^2)}. \quad (7.3.55)$$

For Algorithm (a)  $\beta_3$  is given by

$$\beta_3 = 1 + \frac{1}{(1 - \alpha_3 \|\hat{\varphi}_3(k)\|^2)} \quad (7.3.56)$$

whereas for Algorithm (b) it is given by

$$\beta_3 = 1 + \alpha_3 \|\hat{\varphi}_3(k)\|^2 + \frac{1}{(1 - \alpha_3 \|\hat{\varphi}_3(k)\|^2)} [\alpha_3 \|\hat{\varphi}_3(k)\|^2 + \Gamma(1 - \alpha_3 \|\hat{\varphi}_3(k)\|^2)]^2. \quad (7.3.57)$$

Note: The parameters  $\beta_i, \alpha_i, \forall i = 1, 2, 3$  and  $\bar{\sigma}$  are dependent upon the trajectory.  $\square$

*The proof and bounds on tracking error and NN weights are similar to those in the other theorems of this chapter.*

**Example 7.3.3 (Improved Weight Tuning and Projection Algorithm) :**

In the case of projection algorithm (7.3.47)-(7.3.50) with (7.3.46), the weights in Fig. 7.3.5 appear to be bounded, though this in general cannot be guaranteed without the PE condition. Therefore, the response of the controller with the improved weight tuning (7.3.47)-(7.3.50) with (7.3.46) is shown in Fig. 7.3.6.

The design parameter  $\Gamma$  is selected to be 0.01. Note that with the improved weight tuning, not only is the tracking performance improved, for instance in axis 2, but also the weights remain bounded without the necessity of PE. Finally, in all cases no initial NN training or learning phase was needed. In addition, the dynamics of the nonlinear system was not required to implement the NN controller as opposed to conventional adaptive control.

To study the contribution of the NN, Fig. 7.3.7 shows the response of the PD controller with no neural net. From Fig. 7.3.7, it is clear that the addition of the NN makes a significant improvement in the tracking performance.  $\square$

**Example 7.3.4 :** Consider the Example 7.2.4 and the objective is to track a periodic step input of magnitude two units with a period of 30 sec. The elements of the diagonal matrix were chosen as  $k_v = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}$  and a sampling interval of 10 ms was considered. A three-layer NN was selected with two input, six hidden, and two output nodes. Sigmoidal activation functions were employed in all the nodes in the hidden layer. The initial conditions for the plant were chosen to be  $[1, -1]^T$ . The weights were initialized to zero with an initial threshold value of 3.0. The design parameter  $\Gamma$  is selected to be 0.01. All the elements of the design parameter matrix,  $B_i; i = 1, 2$ , are taken to be 0.1. No learning is performed initially to train the networks. The design parameters for the projection algorithm (7.3.46) were selected to be  $\xi_1 = \xi_2 = 1.0$ , and  $\xi_3 = 0.7$  with  $\zeta_1 = \zeta_2 = \zeta_3 = 0.001$ .

It is also found (not shown) in this example as well that the delta rule based weight tuning performs very well when the learning rate is small. In addition, it was also observed during simulation studies that the learning rate for delta rule based weight tuning should decrease with an increase in the number of hidden-layer neurons. As expected, however, this problem is solved by employing a projection algorithm. In this example, only results using the improved weight tuning is presented. The response of the controller with the improved weight tuning (7.3.47)-(7.3.50) with (7.3.46) is shown in Fig. 7.3.8. Note from Fig. 7.3.8, as expected, the performance of the controller is extremely impressive.

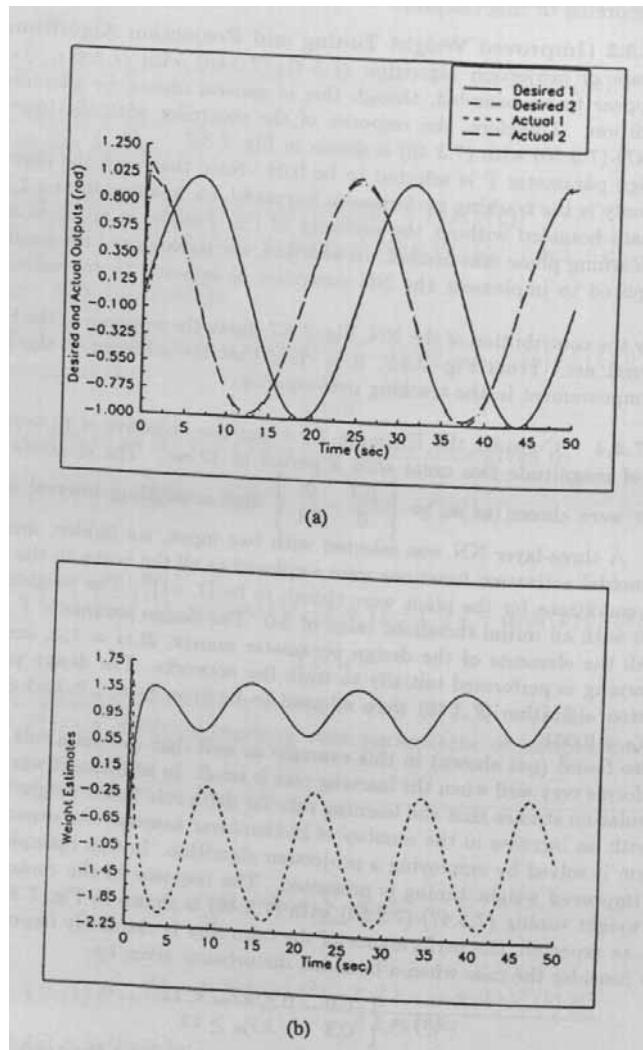
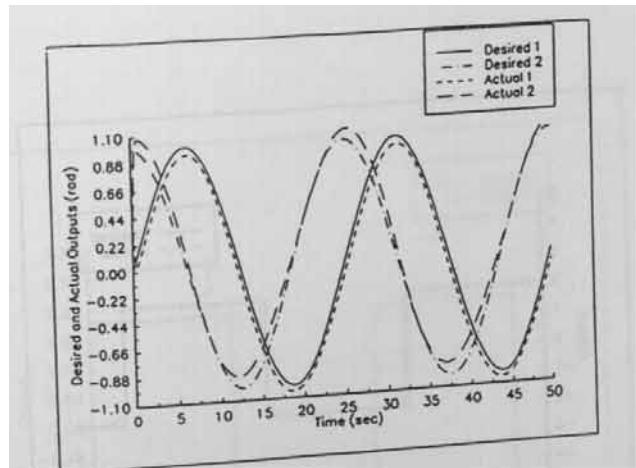


Figure 7.3.6: Response of multilayer neural network controller with improved weight tuning and projection algorithm with small  $\alpha_3$ . (a) Desired and actual trajectory. (b) Representative weight estimates.



Let us consider the case when a bounded disturbance given by

$$w(k) = \begin{cases} 0.0 & 0 \leq kT_m < 12 \\ 0.5 & kT_m \geq 12 \end{cases} \quad (7.3.58)$$

is acting on the plant at the time instant  $t$ . Fig. 7.3.9 presents the tracking response of NN controllers with the improved weight tuning and projection algorithm. The magnitude of the disturbance can be increased, however, the value should be bounded. The value shown in (7.3.58) is employed for simulation purposes only. It can be seen from the figure that the bounded disturbance induces bounded tracking errors at the output of the plant. From the results, it can be inferred that the bounds presented and the theoretical claims were justified through simulation studies both in continuous and discrete-time.  $\square$

#### 7.3.4.2 Multilayer NN Controller Not Requiring PE

*The next result uses the improved weight tuning updates to avoid PE, extending the stability analysis just presented for a three-layer NN to an  $n$ -layer NN. The proof is omitted and left for the reader as an excercise.*

**Theorem 7.3.4 (Multilayer NN Controller Not Requiring PE) :**

Assume the hypotheses presented above and now consider the modified weight tuning algorithms provided for the input and hidden layers as

$$\begin{aligned} \hat{W}_i(k+1) &= \hat{W}_i(k) - \alpha_i \hat{\varphi}_i(k) [\hat{y}_i(k) + B_i k_v r(k)]^T \\ &\quad - \Gamma \| I - \alpha_i \hat{\varphi}_i(k) \hat{\varphi}_i^T(k) \| \hat{W}_i(k), i = 1, \dots, n-1 \end{aligned} \quad (7.3.59)$$

Let the weight update for the output layer be given by either

$$\begin{aligned} a) \quad \hat{W}_n(k+1) &= \hat{W}_n(k) + \alpha_n \hat{\varphi}_n(k) \bar{f}^T \\ &\quad - \Gamma \| I - \alpha_n \hat{\varphi}_n(k) \hat{\varphi}_n^T(k) \| \hat{W}_n(k) \end{aligned} \quad (7.3.60)$$

or

$$\begin{aligned} b) \quad \hat{W}_n(k+1) &= \hat{W}_n(k) + \alpha_n \hat{\varphi}_n(k) r^T(k+1) \\ &\quad - \Gamma \| I - \alpha_n \hat{\varphi}_n(k) \hat{\varphi}_n^T(k) \| \hat{W}_n(k) \end{aligned} \quad (7.3.61)$$

with  $\Gamma > 0$  a design parameter. Then, the filtered tracking error  $r(k)$  and the NN weight estimates  $\hat{W}_i(k); \forall i = 1, \dots, n$  are UUB, provided the following conditions hold:

$$(1) \quad \alpha_i \| \hat{\varphi}_i(k) \|^2 < \begin{cases} 2 & \forall i = 1, \dots, n-1 \\ 1 & \forall i = n \end{cases} \quad (7.3.62)$$

$$(2) \quad 0 < \Gamma < 1, \quad (7.3.63)$$

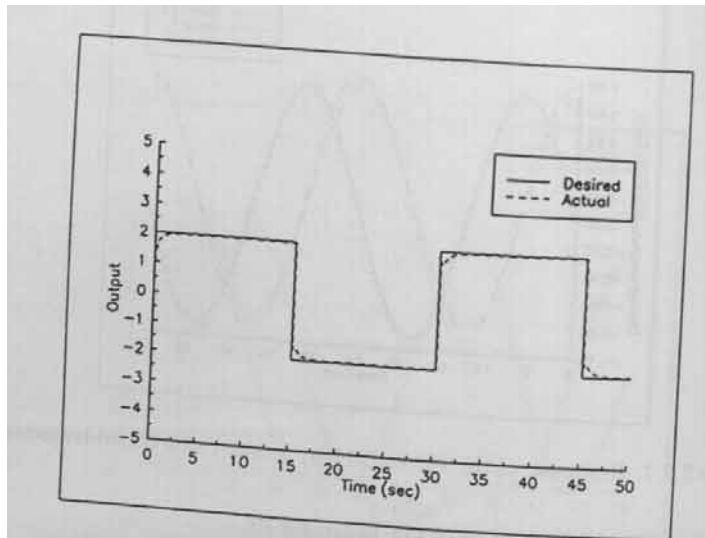
$$(3) \quad k_{vmax} < \frac{1}{\sqrt{\bar{\sigma}}} \quad (7.3.64)$$

where  $\bar{\sigma}$  is given by

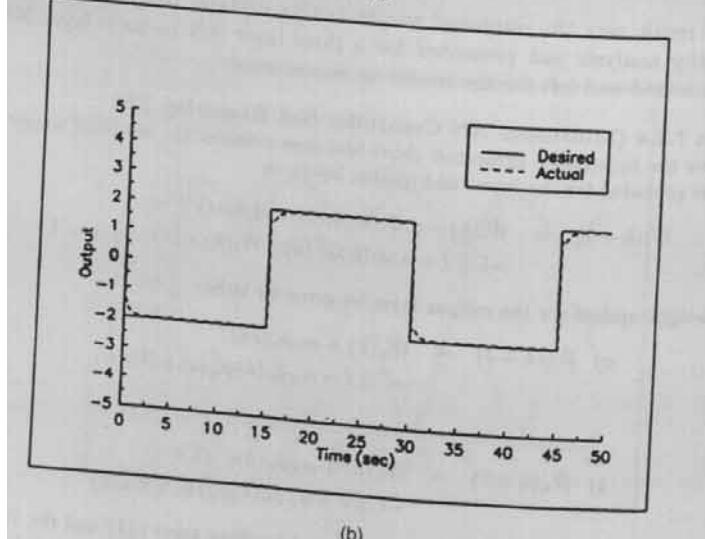
$$\bar{\sigma} = \beta_n + \sum_{i=1}^{n-1} \beta_i \kappa_i^2 \quad (7.3.65)$$

with

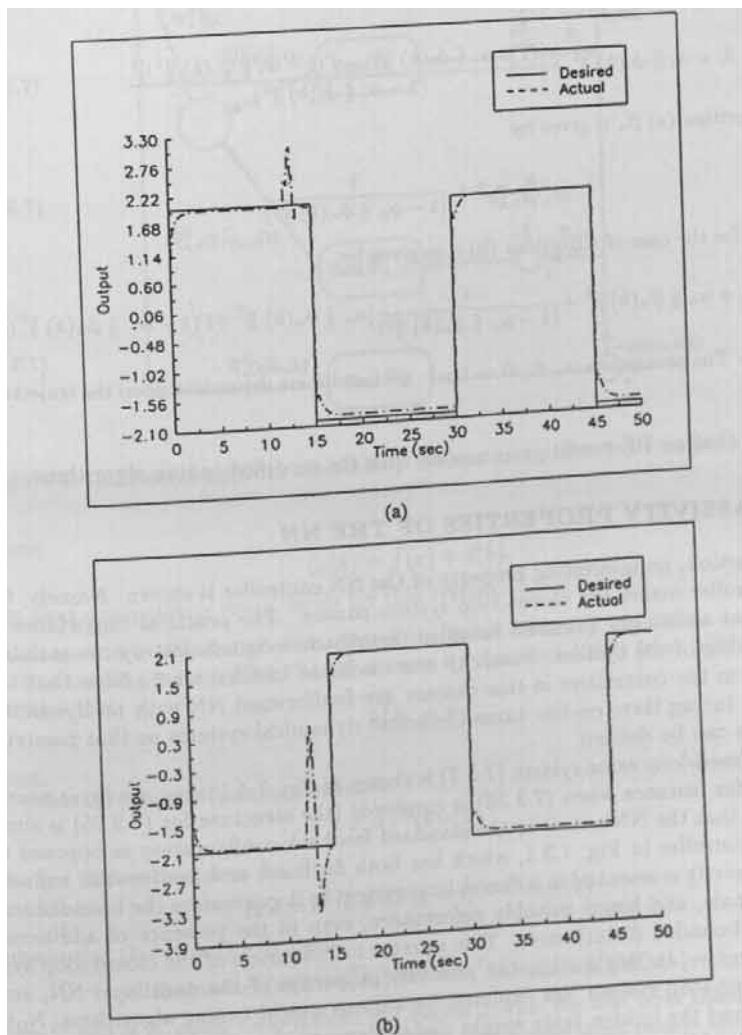
$$\beta_i = \alpha_i \| \hat{\varphi}_i(k) \|^2 + \frac{[(1 - \alpha_i \| \hat{\varphi}_i(k) \|^2) - \Gamma(1 - \alpha_i \| \hat{\varphi}_i(k) \|^2)]^2}{(2 - \alpha_i \| \hat{\varphi}_i(k) \|^2)}. \quad (7.3.66)$$



(a)



(b)



For Algorithm (a)  $\beta_n$  is given by

$$\beta_n = 1 + \frac{1}{(1 - \alpha_n \|\hat{\varphi}_n(k)\|^2)} \quad (7.3.67)$$

whereas for the case of Algorithm (b) it is given by

$$\beta_n = 1 + \alpha_n \|\hat{\varphi}_n(k)\|^2 + \frac{1}{(1 - \alpha_n \|\hat{\varphi}_n(k)\|^2)} [\alpha_n \|\hat{\varphi}_n(k)\|^2 + \Gamma(1 - \alpha_n \|\hat{\varphi}_n(k)\|^2)]^2 \quad (7.3.68)$$

Note: The parameters  $\alpha_i, \beta_i, \forall i = 1, \dots, n-1$  and  $\bar{\sigma}$  are dependent upon the trajectory.  $\square$

*Note that no PE condition is needed with the modified tuning algorithms.*

## 7.4 PASSIVITY PROPERTIES OF THE NN

*In this section, an interesting property of the NN controller is shown. Namely, the NN controller makes the closed-loop system passive. The practical importance of this is that additional unknown bounded disturbances do not destroy the stability and tracking of the system. Passivity was discussed in Chapter 2. Note that the NN used in the controllers in this chapter are feedforward NN with no dynamics. However, tuning them on-line turns them into dynamical systems so that passivity properties can be defined.*

*The closed-loop error system (7.3.7) is shown in Fig. 7.4.1 using a n-layer neural network, for instance when (7.3.36) is employed (the structure for (7.3.35) is similar); note that the NN now is in the standard feedback configuration as opposed to the NN controller in Fig. 7.3.1, which has both feedback and feedforward connections. Passivity is essential in a closed-loop system as it guarantees the boundedness of the signals, and hence suitable performance, even in the presence of additional unforeseen bounded disturbances. This equates to robustness of the closed-loop system. Therefore, in this section the passivity properties of the multilayer NN, and of the closed-loop system, are explored for various weight tuning algorithms. Note the input and the hidden layer weight update laws employed in multilayer nets are the same for Algorithms (a) and (b).*

### 7.4.1 Passivity Properties of the Tracking Error System

*In general, the closed-loop tracking system (7.1.12) can also be expressed as*

$$r(k+1) = k_v r(k) + \zeta_0(k) \quad (7.4.1)$$

*where*

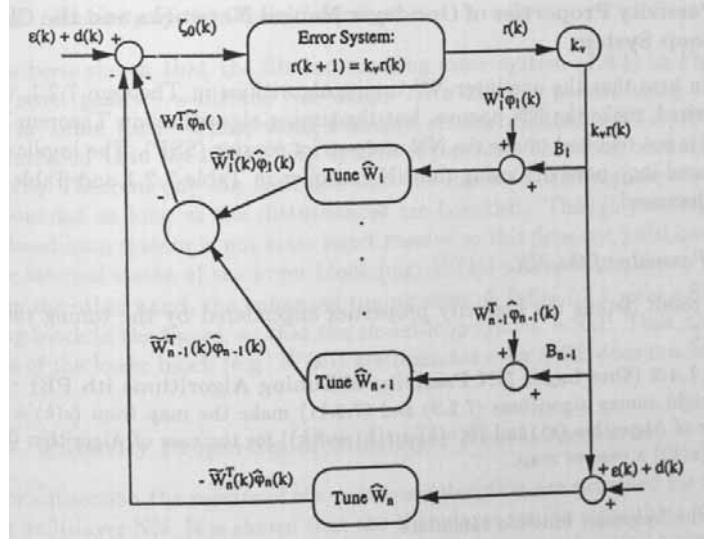
$$\zeta_0(k) = \tilde{f}(x) + d(k). \quad (7.4.2)$$

*The next dissipativity result holds for this system.*

#### Theorem 7.4.1 (Passivity of the Tracking Error System) :

The tracking error system (7.1.12) is state strict passive (SSP) from  $\zeta_0(k)$  to  $k_v r(k)$  provided that

$$k_v^T k_v < I. \quad (7.4.3)$$

Figure 7.4.1: Neural network closed-loop system using an  $n$ -layer neural network.Proof:

Select a Lyapunov function candidate

$$J = r^T(k)r(k). \quad (7.4.4)$$

The first difference is given by

$$\Delta J = r^T(k+1)r(k+1) - r^T(k)r(k). \quad (7.4.5)$$

Substituting (7.4.1) in (7.4.5) yields

$$\Delta J = -r^T(k)[I - k_v^T k_v]r(k) + 2r^T(k)k_v \zeta_0 + \zeta_0^T(k)\zeta_0. \quad (7.4.6)$$

Note (7.4.6) is in power form defined in Chapter 2 with the first term taken as  $g(k)$ , a quadratic function of the state  $r(k)$ . Hence (7.4.1) is a state strict passive system.  $\square$ 

*Even though the closed-loop error system (7.4.1) is state strict passive, the closed-loop system is not passive unless the weight update laws guarantee the passivity of the lower block in Fig. 7.4.1. It is usually difficult to demonstrate that the error in weight updates are passive. However, in the next subsection it is shown that the delta-rule-based tuning algorithm (7.2.9) and (7.2.11) for a one-layer neural network yield a passive net.*

#### 7.4.2 Passivity Properties of One-layer Neural Networks and the Closed-Loop System

*It is shown here that the one-layer NN tuning algorithms in Theorem 7.2.1, where PE is required, make the NN passive, but the tuning algorithms in Theorem 7.2.3, where PE is not required, make the NN state strict passive (SSP). The implications for the closed-loop passivity using the NN controller in Table 7.2.1 and Table 7.2.2 are then discussed.*

#### 7.4.2.1 Passivity of the NN

The next result details the passivity properties engendered by the tuning rules in Table 7.2.1.

**Theorem 7.4.2 (One-Layer NN Passivity of Tuning Algorithms ith PE) :**

The weight tuning algorithms (7.2.9) and (7.2.11) make the map from  $(\epsilon(k) + d(k))$  for the case of Algorithm (a), and  $(k_v r(k) + \epsilon(k) + d(k))$  for the case of Algorithm (b), to  $-\tilde{W}^T(k)\varphi(x(k))$  a passive map.

Proof:

Define the Lyapunov function candidate

$$J = \frac{1}{\alpha} \text{tr}[\tilde{W}^T(k)\tilde{W}(k)], \quad (7.4.7)$$

whose first difference is given by

$$\Delta J = \frac{1}{\alpha} \text{tr}[\tilde{W}^T(k+1)\tilde{W}(k+1) - \tilde{W}^T(k)\tilde{W}(k)]. \quad (7.4.8)$$

Algorithm (a): Substituting the weight update law (7.2.9) in (7.4.8) yields

$$\begin{aligned} \Delta J = & -(2 - \alpha\varphi^T(x(k)))(-\tilde{W}^T(k)\varphi(x(k)))^T(-\tilde{W}^T(k)\varphi(x(k))) + \\ & 2(1 - \alpha\varphi^T(x(k))\varphi(x(k)))(-\tilde{W}^T(k)\varphi(x(k)))^T(\epsilon(k) + d(k)) + \\ & \alpha\varphi^T(x(k))\varphi(x(k))(\epsilon(k) + d(k))^T(\epsilon(k) + d(k)). \end{aligned} \quad (7.4.9)$$

Note (7.4.9) is in power form defined in Chapter 2 as long as the condition (7.2.12) holds. This in turn guarantees the passivity of the weight tuning mechanism (7.2.9).

Algorithm (b): Select the Lyapunov function candidate (7.4.7). Use (7.2.11) in (7.4.8) to obtain

$$\begin{aligned} \Delta J = & -(2 - \alpha^T(x(k))\varphi(x(k)))(-\tilde{W}^T(k)\varphi(x(k)))^T(-\tilde{W}^T(k)\varphi(x(k))) + \\ & 2(1 - \alpha\varphi^T(x(k))\varphi(x(k)))(-\tilde{W}^T(k)\varphi(x(k)))^T(k_v r(k) + \epsilon(k) + d(k)) + \\ & \alpha\varphi^T(x(k))\varphi(x(k))(k_v r(k) + \epsilon(k) + d(k))^T(k_v r(k) + \epsilon(k) + d(k)) \end{aligned} \quad (7.4.10)$$

which is in power form defined in Chapter 2 for discrete-time systems as long as the condition (7.2.12) holds.  $\square$

*The next result shows that the modified tuning algorithms in Table 7.2.2 yield a stronger passivity property for the NN. The proof is an extension of the previous one.*

**Theorem 7.4.3 (One-Layer NN Passivity of Tuning Algorithms without PE) :**

The modified weight tuning algorithms (7.2.36) and (7.2.37) make the map from,  $(\epsilon(k) + d(k))$  for the case of Algorithm (a), and  $(k_v r(k) + \epsilon(k) + d(k))$  for the case of Algorithm (b), to  $-\tilde{W}^T(k)\varphi(x(k))$  a state strict passive map.  $\square$

#### 7.4.2.2 Passivity of the Closed-Loop System

*It has been shown that the filtered tracking error system (7.4.1) in Fig. 7.4.1 is state strict passive, while the NN weight error block is passive using the tuning rules in Table 7.2.1. Thus, using standard results (Slotine and Li 1991), it can be concluded that the closed-loop system is passive. Therefore, according to the Passivity Theorem one can conclude that the inputs/output signals of each block are bounded as long as the disturbances are bounded. Though passive, however, the closed-loop system is not state strict passive so this does not yield boundedness of the internal states of the lower block (e.g.  $\tilde{W}(k)$ ) unless PE holds.*

*On the other hand, the enhanced tuning rules of Table 7.2.2 yield a SSP weight tuning block in the figure, so that the closed-loop system is SSP. Thus, the internal states of the lower block (e.g.  $\tilde{W}(k)$ ) are bounded even if PE does not hold.*

#### 7.4.3 Passivity Properties of Multilayer Neural Networks

*In this subsection the results of the previous subsection are extended for controllers using multilayer NN. It is shown that the three-layer tuning algorithms in Theorem 7.3.1, where PE is required, make the NN passive, whereas the tuning algorithms in Theorem 7.3.3, where PE is not required, make the NN state strict passive. Similar results can be shown for the general case of multiple hidden layers. The implications for closed-loop passivity are detailed.*

##### Theorem 7.4.4 (Three-Layer NN Passivity of Tuning Algorithms with PE) :

The weight tuning algorithms (7.3.9) and (7.3.10) make the map from  $W_i^T \varphi_i(k) + B_i k_v r(k)$  to  $\tilde{W}_i^T(k) \hat{\varphi}_i(k)$ ;  $i = 1, 2$ , both passive maps.  $\square$

*Thus, the weight error block is passive and the closed-loop filtered tracking error system (7.3.7) in Fig. 7.4.1 is dissipative; this guarantees the dissipativity of the closed-loop system [Slotine and Li 1991]. By employing the passivity theorem (Slotine and Li 1991), one can conclude that the inputs/output signals of each block are bounded as long as the external inputs are bounded. Although dissipative, the closed-loop system is not state strict passive so this does not yield boundedness of the internal states of the lower block ( $\tilde{W}_i(k)$ ;  $\forall i = 1, \dots, n$ ) unless PE holds.*

*The next proof shows why PE is not needed with the modified update algorithms.*

##### Theorem 7.4.5 (Three-Layer NN Passivity without PE) :

The modified weight tuning algorithms (7.3.47) and (7.3.48) make the map from  $W_i^T \varphi_i(k) + B_i k_v r(k)$  to  $\tilde{W}_i^T(k) \hat{\varphi}_i(k)$ ;  $i = 1, 2$ , both state strict passive maps. Also, the weight tuning mechanisms (7.3.49) and (7.3.50) for a three-layer NN make the map from,  $(W_3^T \tilde{\varphi}_3(k) + \epsilon(k) + d(k))$  for the case of Algorithm (a), and  $(k_v r(k) + W_3^T \tilde{\varphi}_3(k) + \epsilon(k) + d(k))$  for the case of Algorithm (b), to  $-\tilde{W}_3^T(k) \hat{\varphi}_3(k)$  a state strict passive map.  $\square$

*Thus, the modified tuning algorithms guarantee SSP of the weight tuning blocks, so that the closed-loop system is SSP. Therefore, internal stability can be guaranteed even in the absence of PE.*

## 7.5 CONCLUSIONS

*In this chapter and the next two are given neural network controllers and identifiers that use discrete-time tuning. Discrete-time tuning is important because the complexity of NN controllers, and indeed modern control algorithms in general, requires their implementation as digital controllers using microprocessors. Rigorous proofs of closed-loop stability and performance for discrete-time learning/adaptive controllers are exceedingly complex, since discrete-time Lyapunov functions are quadratic in the state first difference. This has traditionally been approached by proving convergence of the parameter identification algorithm and then making a certainty equivalence assumption. Under this assumption, tracking error stability is then proved separately. However, in our approach we select a Lyapunov function that includes both the tracking error and the parameter estimation error. This makes for complicated proofs but allows one to avoid the certainty equivalence assumption. We also showed how to avoid other restrictions such as linearity in the unknown system parameters, the need for computing a regression matrix, and persistence of excitation.*

*First, a class of NN was considered that has only one layer of tunable weights. Then the full nonlinear in the parameters control problem was confronted for multilayer NN controllers. In each case, several NN controllers were derived. Finally, passivity properties of NN controllers were studied.*

## 7.6 REFERENCES

- Åström, K.J. , and B. Wittenmark, *Adaptive Control*, Addison-Wesley Company, Reading, Massachusetts, 1989.
- Chen, F.-C., and H.K. Khalil, "Adaptive control of nonlinear systems using neural networks," *International Journal of Control*, vol. 55, pp. 1299-1317, 1992.
- Chen, F.-C., and H.K. Khalil, "Adaptive control of nonlinear discrete-time systems using neural networks," *IEEE Trans. on Automatic Control*, vol. 40, no. 5, pp. 791-801, May 1995.
- Commuri, S., and F.L. Lewis, "CMAC neural networks for control of nonlinear dynamical systems: structure, stability and passivity," *Proc. IEEE Int. Symposium on Intelligent Control*, pp. 123-129, Monterey, August 1995.
- Goodwin, G.C., and K.S. Sin, *Adaptive Filtering, Prediction, and Control*, Prentice-Hall, Englewood Cliffs, NJ, 1984.
- Kanellakopoulos, I., "A discrete-time adaptive nonlinear system," *IEEE Trans. on Automatic Control*, vol. 39, no. 11, pp. 2362-2365, November 1994.
- Jagannathan, S., and F.L. Lewis, "Robust implicit self-tuning regulator: convergence and stability," *Automatica*, vol. 32, no. 12, pp. 1629-1644, 1996a.
- Jagannathan, S., and F. L.Lewis, "Multilayer discrete-time neural net controller with guaranteed performance," *IEEE Trans. on Neural Networks*, vol.7, no.1, pp. 107-130, January 1996b.

- Jagannathan, S., and F.L. Lewis, "Discrete-time neural net controller for a class of nonlinear dynamical systems," IEEE Trans. Automat. Control, vol. 41, no. 11, pp. 1693-1699, Nov. 1996c.*
- Landau, I.D., Adaptive Control: The Model Reference Approach, Marcel Dekker, New York, 1979.*
- Landau, I.D., "Evolution of adaptive control," ASME J. Dynamic Syst. Measurements, Contr., vol. 115, pp. 381-391, June 1993.*
- Lewis, F.L., K. Liu and A. Yesildierik, "Multilayer neural robot controller with guaranteed performance," IEEE Trans. on Neural Networks, vol. 6, no. 3, pp. 703-715, May 1995.*
- Lewis, F.L., C.T. Abdallah, and D.M. Dawson, Control of Robot Manipulators, MacMillan, New York, 1993.*
- Ljung, L., and T. Söderström, Theory and Practice of Recursive Identification, MIT Press Cambridge, MA, 1983.*
- Mpitsos, G.J., and R.M. Burton Jr, "Convergence and divergence in neural networks: Processing of chaos and biological analogy," Neural Networks, vol. 5, pp. 605-625, 1992.*
- Narendra, K.S., and A.M. Annaswamy, "A new adaptive law for robust adaptation without persistent excitation," IEEE Trans. on Automatic Control, vol. AC-32, no. 2, pp. 134-145, February 1987.*
- Narendra, K.S., and A.M. Annaswamy, Stable Adaptive Systems, Prentice-Hall, Englewood Cliffs, NJ, 1989.*
- Narendra, K.S., and K.S. Parthasarathy, "Identification and control of dynamical systems using neural networks," IEEE Trans. on Neural Networks, vol. 1, no. 1, pp. 4-27, March 1990.*
- Polycarpou, M.M., and P.A. Ioannou, "Identification and control using neural network models: design and stability analysis," Dept. of Elec. Eng., Tech Report 91-09-01, September 1991.*
- Rumelhart, D.E., G.E. Hinton, and R.J. Williams, "Learning internal representations by error propagation," in Readings in Machine Learning, pp. 115-137, ed. J. W. Shavlik, Morgan Kaufmann, San Mateo, CA, 1990.*
- Sadegh, N., "A perceptron network for functional identification and control of non-linear systems," IEEE Trans. on Neural Networks, vol. 4, no. 6, pp. 982-988, November 1993.*
- Sanner, R.M., and J.-J. Slotine, "Gaussian networks for direct adaptive control", IEEE Trans. on Neural Networks, vol. 3, no. 6, pp. 837-863, November 1992.*
- Sira-Ramirez, H.J., and S.H. Zak, "The adaptation of perceptrons with applications to inverse dynamics identification of unknown dynamic systems," IEEE Trans. Syst., Man, Cybernetics, vol. 21, no. 3, pp. 534-543, May/June 1991.*

*Slotine, J.-J. E, and W. Li, Applied Nonlinear Control, Prentice-Hall, Englewood Cliffs, NJ, 1991.*

*Sontag, E., "Feedback stabilization using two-hidden-layer nets," IEEE Trans. on Neural Networks, vol. 3, no. 6, pp.981-990, November 1992.*

*Sussmann, H. J., "Uniqueness of the weights for minimal feedforward nets with given input-output map," Neural Networks, vol.5, pp.589-593, 1992.*

*Widrow, B., and M. Lehr, "Thirty years of adaptive neural networks: Perceptrons, madaline and backpropagation," Proc. of the IEEE, vol. 78, no. 9, pp.1415-1442, September 1990.*

*Zhang, T., C.C. Hang, and S.S. Ge, "Robust adaptive control for general nonlinear systems using multilayer neural networks," preprint, 1998.*

## 7.7 PROBLEMS

### Section 7.2

**Problem 7.2-1 : One-Layer Neural Network.** For the system described by

$$x(k+1) = f(x(k), x(k-1)) + u(k), \quad (7.7.1)$$

where  $f(x(k), x(k-1)) = \frac{x(k)x(k-1)[x(k)+1.0]}{1+x^2(k)+x^2(k-1)}$ .

Design a one-layer neural network controller with and/or without learning phase and by using the developed delta rule-based weight tuning algorithm and appropriately choosing the adaptation gains. Repeat the problem by using the modified update weight tuning method.

**Problem 7.2-2 : One-layer Neural Network.** For the system described by

$$x(k+1) = f(x(k), x(k-1)) + u(k), \quad (7.7.2)$$

where  $f(x(k), x(k-1)) = \frac{x(k)}{1+x(k)} + u^3(k)$ . Design a one-layer neural network controller with and/or without learning phase and by using the developed delta rule-based weight tuning algorithm and appropriately choosing the adaptation gains. Repeat the problem by using the modified update weight tuning method.

### Section 7.3

**Problem 7.3-1 : Stability and Convergence for a  $n$ -layer NN Using Algorithm (a).** Assume the hypotheses presented for three-layer NN and use the weight updates presented in (7.3.33)-(7.3.35) and show the stability and boundedness of tracking error and error in weight updates.

**Problem 7.3-2 : Stability and Convergence for a  $n$ -layer NN Using Algorithm (b).** Assume the hypotheses presented for three-layer NN and use the weight updates presented in (7.3.33)-(7.3.34) with (7.3.35) and show the stability and boundedness of tracking error and error in weight updates.

**Problem 7.3-3 : Three-layer NN Continuous-Time Simulation Example Using Algorithm (a).** Perform a MATLAB simulation for Example 7.2.1 using a multilayer neural network with delta rule-based weight tuning.

**Problem 7.3-4 : Three-layer NN Using Algorithm (b).** Perform a MATLAB simulation for systems described by (7.7.1) and (7.7.2) using a multilayer neural network with delta rule-based weight tuning.

**Problem 7.3-5 : Three-layer NN Discrete-time Simulation Example Using Algorithm (a).** Perform a MATLAB simulation for Example 7.2.4 using a multilayer neural network with delta rule-based weight tuning.

**Problem 7.3-6 : Delta Rule Slows Down Using Algorithm (a).** Perform a MATLAB Simulation using a large value of adaptation gains for Example 7.2.1.

**Problem 7.3-7 :  $n$ -Layer NN for Control.** Perform a MATLAB simulation using a  $n$ -layer NN and with Algorithms (a) and (b) for the Example 7.2.1. Show the advantage of adding more layers by picking less number of hidden-layer neurons and layers more than three. Use both delta-rule and projection algorithm.

**Problem 7.3-8 : Stability and Convergence of a  $n$ -Layer NN with Modified Weight Tuning.** Show for a  $n$ -layer NN and use the modified weight tuning (use both Algorithm (a) and Algorithm (b)) to show the boundedness of both tracking error and weight estimates.

**Problem 7.3-9 : Example (7.2.1) Using Modified Weight Tuning.** Perform a MATLAB simulation for the Example 7.2.1 using a three-layer NN and with Algorithm (a).

**Problem 7.3-10 : Discrete-Time Simulation Example Using Modified Weight Tuning.** Perform a MATLAB simulation for the Example 7.2.4 using a three-layer NN and with Algorithm (a).

**Problem 7.3-11 : Three-layer NN Using Algorithm (b).** Perform a MATLAB simulation for systems described by (7.7.1) and (7.7.2) using a multilayer neural network with improved weight tuning.

**Problem 7.3-12 :  $n$ -layer NN and Modified Tuning Methods.** Repeat Examples 7.2.1 and 7.2.4 using a  $n$ -layer (choose more than three layers) with fewer number of hidden-layer neurons and with more number of layers.

## Section 7.4

**Problem 7.4-1 : Passivity Properties for a  $n$ -layer NN.** Show the passivity properties of the input and hidden layers for a  $n$ -layer neural network using delta rule-based weight tuning and with Algorithms (a) and (b).

**Problem 7.4-2 : Passivity Properties for a  $n$ -layer NN Using Modified Weight Tuning.** Show the passivity properties of the input and hidden layers for a  $n$ -layer neural network using improved weight tuning and with Algorithms (a) and (b).

## Chapter 8

# Discrete-Time Feedback Linearization by Neural Networks

*In the previous chapter, adaptive control of a class of nonlinear systems in discrete-time was presented using neural networks. Although Lyapunov stability analysis and passivity properties were detailed, the analysis was limited to a specific class of nonlinear systems of the form  $x(k+1) = f(x(k)) + u(k)$ , where there are no uncertainties in the coefficient of the control input  $u(k)$ . However, if a system in continuous-time of the form  $\dot{x} = f_1(x) + u(k)$  is discretized (Chen and Khalil 1995), the system in discrete-time will be of the form  $x(k+1) = f(x(k)) + g(x(k))u(k)$  for some functions  $f(\cdot)$  and  $g(\cdot)$ . Therefore, in this chapter the results in the previous chapter are extended to the more general class of nonlinear discrete-time systems of the form  $x(k+1) = f(x(k)) + g(x(k))u(k)$ .*

*Note that for control purposes even if the open-loop system is stable, it must be shown that inputs, outputs, and states remain bounded when a feedback loop is designed. In addition, if the controller is given in the form  $u(k) = \frac{N(x)}{D(x)}$ , then  $D(x)$  must be non zero for all time—we call this a well-defined controller. For feedback linearization, this type of controller is usually needed. If any adaptation scheme is implemented to provide an estimate  $\hat{D}(x)$  of  $D(x)$ , then extra precaution is required to guarantee that  $\hat{D}(x) \neq 0$  for all time.*

*In Jagannathan and Lewis (1996a, 1996c) it has been shown that NN can effectively control in discrete-time a complex nonlinear system without the necessity of a regression matrix. There, the nonlinear systems under consideration are of the form  $x(k+1) = f(x(k)) + u(k)$ , with the coefficient of the input matrix being identity. Even though in Chen and Khalil (1995) a multilayer NN controller designed in discrete-time is employed to control a nonlinear system of the form  $x(k+1) = f(x(k)) + g((k))u(k)$ , an initial off-line learning phase is needed. There,  $g(x(k))$  is reconstructed by an adaptive scheme as  $\hat{g}(x)$ , and a local solution is given by assuming that initial estimates are close to the actual values and they do not leave a feasible invariant set in which  $\hat{g}(x) \neq 0$ . Unfortunately, even with very good*

knowledge of the system it is not easy to choose the initial weights so that the NN approximates it. Therefore off-line learning phase is used to identify the system in order to choose the initial weight values.

In Chapter 6 a multilayer NN controller (Yeşildirek and Lewis 1995) is designed in continuous-time to perform feedback linearization of Brunovsky form systems that takes into account all these problems. The motivation of this chapter is to provide like results for discrete-time. A family of novel learning schemes is presented here that do not require preliminary off-line training. The traditional problems with discrete-time adaptive control are overcome by using a single Lyapunov function containing both the parameter identification errors and the control errors. This guarantees at once both stable identification and stable tracking. However, it leads to complex proofs where it is necessary to complete the square with respect to several different variables. The use of a single Lyapunov function for tracking and estimation avoids the need for the certainty equivalence assumption. Along the way various other standard assumptions in discrete-time adaptive control are also overcome, including persistence of excitation, linearity-in-the-parameters, and the need for tedious computation of a regression matrix. The problem of  $\hat{g}(x) \neq 0$  is confronted by appropriately selecting the weight updates as well as the control input.

First we treat design for one-layer neural nets (Jagannathan and Lewis 1996b) where the weights enter linearly. In this case, we discuss the controller structure, various weight update algorithms, and persistence of excitation definitions. Note that linearity in the NN weights is far milder than the usual adaptive control restriction of linearity in the unknown system parameters, since the universal approximation property of NN means any smooth nonlinear function can be reconstructed. Next, multilayer NN are employed for feedback linearization, with rigorous stability analyses presented. Finally, passivity properties of discrete-time NN controllers are covered.

## 8.1 SYSTEM DYNAMICS AND THE TRACKING PROBLEM

In this section we describe the class of systems to be dealt with in this chapter and study the error dynamics using a specific feedback linearization controller.

### 8.1.1 Tracking Error Dynamics for a Class of Nonlinear Systems

Consider an  $mn$ -th-order multi-input/multi-output (MIMO) discrete-time state feedback linearizable minimum phase nonlinear system (Chen and Khalil 1995), to be controlled, given in the multivariable Brunovsky form (see Chapter 2) as

$$\begin{aligned} x_1(k+1) &= x_2(k) \\ &\vdots \\ x_{n-1}(k+1) &= x_n(k) \\ x_n(k+1) &= f(x(k)) + g(x(k))u(k) + d(k) \end{aligned} \tag{8.1.1}$$

with state  $x(k) = [x_1^T(k), \dots, x_n^T(k)]^T$  having  $x_i(k) \in \mathbb{R}^m$ ;  $i = 1, \dots, n$ , and control  $u(k) \in \mathbb{R}^m$ . The nonlinear functions  $f(\cdot)$  and  $g(\cdot)$  are assumed unknown. The disturbance vector acting on the system at the instant  $k$  is  $d(k) \in \mathbb{R}^m$ , which we

assume unknown but bounded so that  $\| d(k) \| \leq d_M$  a known constant. Further, the unknown smooth function satisfies the mild assumption

$$| g(x(k)) | \geq g > 0 \quad (8.1.2)$$

with  $g$  a known lower bound. The assumption given above on the smooth function  $g(x)$  implies that  $g(x)$  is strictly either positive or negative for all  $x$ . From now on, without loss of generality, we will assume that  $g(x)$  is strictly positive. Note that at this point there is no general approach to analyze this class of unknown nonlinear systems. Adaptive control, for instance, needs an additional linear in the parameters assumption (Åström and Wittenmark 1989, Goodwin and Sin 1984).

Feedback linearization will be used to perform output tracking, whose objective can be described as: given a desired trajectory in terms of output,  $x_{nd}(k)$ , and its delayed values, find a control input  $u(k)$  so that the system tracks the desired trajectory with an acceptable bounded error in the presence of disturbances while all the states and controls remain bounded. In order to continue, the following assumptions are required.

**Assumption 8.1.1 (Bounds for System and Desired Trajectory) :**

1. The sign of  $g(x)$  is known.
2. The desired trajectory vector with its delayed values is assumed to be available for measurement and bounded by an upper bound.

□

Given a desired trajectory  $x_{nd}(k)$  and its delayed values, define the tracking error as

$$e_n(k) = x_n(k) - x_{nd}(k), \quad (8.1.3)$$

and the filtered tracking error,  $r(k) \in \mathbb{R}^m$ ,

$$r(k) = e_n(k) + \lambda_1 e_{n-1}(k) + \cdots + \lambda_{n-1} e_1(k), \quad (8.1.4)$$

where  $e_{n-1}(k), \dots, e_1(k)$  are the delayed values of the error  $e_n(k)$  and  $\lambda_1, \dots, \lambda_{n-1}$  are constant matrices selected so that  $| z^{n-1} + \lambda_1 z^{n-2} + \cdots + \lambda_{n-1} |$  is stable. Equation (8.1.4) can be expressed as

$$r(k+1) = e_n(k+1) + \lambda_1 e_{n-1}(k+1) + \cdots + \lambda_{n-1} e_1(k+1). \quad (8.1.5)$$

Using (8.1.1) in (8.1.5), the dynamics of the filtered tracking error system (8.1.5) can be written in terms of the tracking error as

$$r(k+1) = f(x(k)) - x_{nd}(k+1) + \lambda_1 e_n(k) + \cdots + \lambda_{n-1} e_2(k) + g(x(k)) u(k) + d(k) \quad (8.1.6)$$

Equation (8.1.6) can be expressed as

$$r(k+1) = f(x(k)) + g(x(k)) u(k) + d(k) + Y_d, \quad (8.1.7)$$

where

$$Y_d = -x_{nd}(k+1) + \sum_{i=0}^{n-2} \lambda_{i+1} e_{n-i}. \quad (8.1.8)$$

If we knew the functions  $f(x(k))$  and  $g(x(k))$  and when no disturbances are present, the control input  $u(k)$  could be selected as the feedback linearization controller

$$u(k) = \frac{1}{g(x(k))}(-f(x(k)) + v(k)) \quad (8.1.9)$$

with  $v(k)$  taken as an auxiliary input given by

$$v(k) = k_v r(k) - Y_d. \quad (8.1.10)$$

Then the filtered tracking error  $r(k)$  goes to zero exponentially by properly selecting the gain matrix  $k_v$ . Since the system functions are not known *a priori*, the control input  $u(k)$  can be selected as

$$u(k) = \frac{1}{\hat{g}(x(k))}(-\hat{f}(x(k)) + v(k)), \quad (8.1.11)$$

with  $\hat{f}(x(k))$  and  $\hat{g}(x(k))$  being the estimates of  $f(x(k))$  and  $g(x(k))$  respectively. Note that it is well-known, even in adaptive control of linear systems, that guaranteeing the boundedness of  $\hat{g}(x(k))$  away from zero becomes an important issue in this type of controller.

Equation (8.1.7) can be rewritten as

$$r(k+1) = v(k) - v(k) + f(x(k)) + g(x(k))u(k) + d(k) + Y_d. \quad (8.1.12)$$

Substituting (8.1.10) and (8.1.11) for  $v(k)$  in (8.1.12), Equation (8.1.12) can be rewritten as

$$r(k+1) = k_v r(k) + \tilde{f}(x(k)) + \tilde{g}(x(k))u(k) + d(k), \quad (8.1.13)$$

where the functional estimation errors are given by

$$\tilde{f}(x(k)) = f(x(k)) - \hat{f}(x(k)) \quad (8.1.14)$$

and

$$\tilde{g}(x(k)) = g(x(k)) - \hat{g}(x(k)). \quad (8.1.15)$$

This is an error system wherein the filtered tracking error is driven by the functional estimation errors and unknown disturbances.

In this chapter, discrete-time NN are used to provide the estimate  $\hat{f}(\cdot)$  and  $\hat{g}(\cdot)$ . The error system (8.1.13) is used to focus on selecting discrete-time NN tuning algorithms that guarantee the stability of the filtered tracking error  $r(k)$ . Then, since (8.1.4), with the input considered as  $r(k)$  and the output  $e(k)$ , describes a stable system, using the notion of operator gain (Jagannathan and Lewis 1996b) one can guarantee that  $e(k)$  exhibits stable behavior.

## 8.2 NN CONTROLLER DESIGN FOR FEEDBACK LINEARIZATION

In this section we derive the error system dynamics and present the NN controller structure. NN weight tuning algorithms are given for the one-layer case in Section 8.3 and for the multilayer case in Section 8.4.

### 8.2.1 NN Approximation of Unknown Functions

*It will be necessary to review the notation given in Chapter 7 for multilayer NN. Assume that there exist some constant ideal weights  $W_f$  and  $W_g$  for two one-layer NN and  $W_{1f}, W_{2f}, W_{3f}$  and  $W_{1g}, W_{2g}, W_{3g}$  for the case of three-layer NN so that the nonlinear functions in (8.1.1) can be written as*

$$f(x) = W_f^T \varphi_f(x(k)) + \epsilon_f(k) \quad (8.2.1)$$

*and*

$$g(x) = W_g^T \varphi_g(x(k)) + \epsilon_g(k) \quad (8.2.2)$$

*for the case of two one-layer NN and*

$$f(x) = W_{3f}^T \varphi_{3f}(k) + \epsilon_f(k) \quad (8.2.3)$$

*and*

$$g(x) = W_{3g}^T \varphi_{3g}(k) + \epsilon_g(k) \quad (8.2.4)$$

*for the case of two three-layer NN. The notation  $\varphi_3(k)$  was defined in Chapter 7. We assume that  $\|\epsilon_f(k)\| < \epsilon_{Nf}$ ,  $\|\epsilon_g(k)\| < \epsilon_{Ng}$  with the bounding constants  $\epsilon_{Nf}$  and  $\epsilon_{Ng}$  known. The activation functions  $\varphi_f(x(k))$  and  $\varphi_g(x(k))$  must be selected to provide suitable basis sets for  $f(\cdot)$  and  $g(\cdot)$  respectively for the case of one-layer NN. Note that the activations functions in the case of multilayer NN do not need to form a basis, unlike the case of one-layer NN, due to the universal approximation property of multilayer NN (Cybenko 1989).*

*Unless the net is ‘minimal’, the ‘ideal’ weights may not be unique (Sontag 1992, Sussmann 1992). The best weights may then be defined as those which minimize the supremum norm over  $S$  of  $\epsilon(k)$ . This issue is not a major concern here as it is needed to know only the existence of such ideal weights; their actual values are not required. This assumption is similar to Erzberger’s assumptions in the linear-in-the-parameters adaptive control. This major difference is that, while Erzberger’s assumptions often do not hold, the approximation properties of NN guarantee that the ideal weights do always exist if  $f(x(k))$  and  $g(x(k))$  are continuous over a compact set.*

*Let  $x \in U$  a compact subset of  $R^n$ . Assume that  $h(x(k)) \in C^\infty[U]$ , i.e. a smooth function  $U \rightarrow R$ , so that the Taylor series expansion of  $h(x(k))$  exists. One can derive that  $\|x(k)\| \leq d_{01} + d_{11} |r(k)|$ . Then using the bound on  $x(k)$  and expressing  $h(x(k))$  as (8.2.5), yields an upper bound on  $h(x(k))$  as*

$$|h(x(k))| = |W_h^T \varphi_h(k) + \epsilon_h(k)| \leq C_{01} + C_{11} \|r(k)\|, \quad (8.2.5)$$

*with  $C_0$  and  $C_{11}$  computable constants. In addition, the hidden-layer activation functions, such as radial basis functions, sigmoids, etc. are bounded by a known upper bound*

$$\begin{aligned} \|\varphi_f(k)\| &\leq \varphi_{fmax} \\ \|\varphi_g(k)\| &\leq \varphi_{gmax} \end{aligned} \quad (8.2.6)$$

*and*

$$\begin{aligned} \|\varphi_{if}(k)\| &\leq \varphi_{ifmax}, i = 1, 2, 3 \\ \|\varphi_{ig}(k)\| &\leq \varphi_{igmax}, i = 1, 2, 3. \end{aligned} \quad (8.2.7)$$

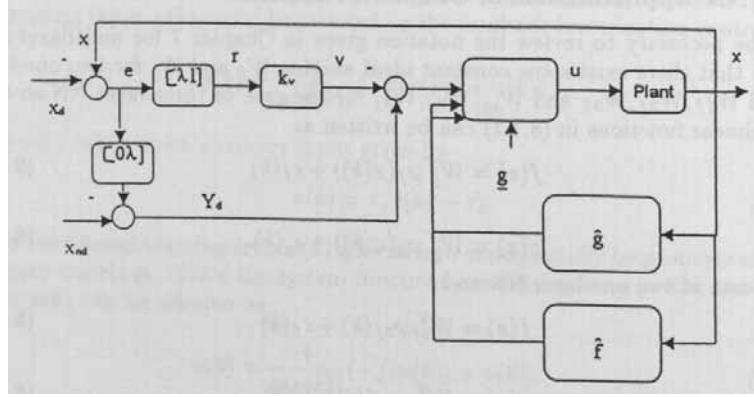


Figure 8.2.1: Discrete-time neural network controller structure for feedback linearization.

### 8.2.2 Error System Dynamics

Defining the NN functional estimates, employed to select the control input presented in (8.1.11), as

$$\hat{f}(x(k)) = \hat{W}_f^T(k)\varphi_f(x(k)) \quad (8.2.8)$$

and

$$\hat{g}(x(k)) = \hat{W}_g^T(k)\varphi_g(x(k)) \quad (8.2.9)$$

with  $\hat{W}_f(k)$  and  $\hat{W}_g(k)$  the current value of the weights. Similarly for the case of three-layer NN,

$$\hat{f}(x(k)) = \hat{W}_{3f}^T(k)\hat{\varphi}_{3f}(k) \quad (8.2.10)$$

and

$$\hat{f}(x(k)) = \hat{W}_{3g}^T(k)\hat{\varphi}_{3g}(k) \quad (8.2.11)$$

with  $\hat{W}_{3f}(k), \hat{W}_{2f}(k), \hat{W}_{1f}(k)$  and  $\hat{W}_{3g}(k), \hat{W}_{2g}(k), \hat{W}_{1g}(k)$  the current values of the weights. This yields the controller structure shown in Fig. 8.2.1. The controller structure is very similar for the multilayer case also. The output of the plant is processed through a series of delays to obtain the past values of the output, and fed as inputs to the NN so that the nonlinear function in (8.1.1) can be suitably approximated. Thus, the NN controller derived in a straightforward manner using filtered error notions naturally provides a dynamical NN structure. Note that neither the input  $u(k)$  nor its past values are needed by the NN. The next step is to determine the weight updates so that the tracking performance of the closed-loop filtered error dynamics is guaranteed.

Let  $W_f$  and  $W_g$  be the unknown ideal weights required for the approximation to hold in (8.2.8) and (8.2.9) for the case of one-layer NN and  $W_{1f}, W_{2f}, W_{3f}, W_{1g}, W_{2g}, W_{3g}$  be the ideal weights for multilayer NN. The weight matrices for the case of multilayered NN are rewritten as

$$Z_f = \text{diag}(Z_{1f}, Z_{2f}, Z_{3f}) \quad (8.2.12)$$

and

$$Z_g = \text{diag}(Z_{1g}, Z_{2g}, Z_{3g}). \quad (8.2.13)$$

Assume they are bounded by known values so that

$$\| W_f(k) \| \leq W_{fmax} \quad (8.2.14)$$

and

$$\| W_g(k) \| \leq W_{gmax} \quad (8.2.15)$$

for the case of one-layer NN and

$$\begin{aligned} \| W_{3f}(k) \| &\leq W_{3fmax}, \\ \| W_{2f}(k) \| &\leq W_{2fmax}, \\ \| W_{1f}(k) \| &\leq W_{1fmax}, \end{aligned} \quad (8.2.16)$$

with

$$\begin{aligned} \| W_{3g}(k) \| &\leq W_{3gmax}, \\ \| W_{2g}(k) \| &\leq W_{2gmax}, \\ \| W_{1g}(k) \| &\leq W_{1gmax}, \end{aligned} \quad (8.2.17)$$

for the multilayer case. Similarly, the ideal weights for the case of multilayered NN are bounded by a known bound

$$\| Z_f(k) \| \leq Z_{fmax} \quad (8.2.18)$$

$$\| Z_g(k) \| \leq Z_{gmax}. \quad (8.2.19)$$

Similarly, one can also define the matrix of current weights for the case of a multilayered NN as

$$\hat{Z}_f(k) = \text{diag}(\hat{Z}_{1f}(k), \hat{Z}_{2f}(k), \hat{Z}_{3f}(k)) \quad (8.2.20)$$

and

$$\hat{Z}_g(k) = \text{diag}(\hat{Z}_{1g}(k), \hat{Z}_{2g}(k), \hat{Z}_{3g}(k)). \quad (8.2.21)$$

Then the error in the weights during estimation is given by

$$\tilde{W}_f(k) = W_f - \hat{W}_f(k) \quad (8.2.22)$$

and

$$\tilde{W}_g(k) = W_g - \hat{W}_g(k), \quad (8.2.23)$$

for the case of one-layer NN and

$$\begin{aligned} \tilde{W}_{3f}(k) &= W_{3f} - \hat{W}_{3f}(k), \\ \tilde{W}_{2f}(k) &= W_{2f} - \hat{W}_{2f}(k), \\ \tilde{W}_{1f}(k) &= W_{1f} - \hat{W}_{1f}(k), \\ \tilde{Z}_f(k) &= Z_f - \hat{Z}_f(k), \end{aligned} \quad (8.2.24)$$

with

$$\begin{aligned}\tilde{W}_{3g}(k) &= W_{3g} - \hat{W}_{3g}(k), \\ \tilde{W}_{2g}(k) &= W_{2g} - \hat{W}_{2g}(k), \\ \tilde{W}_{1g}(k) &= W_{1g} - \hat{W}_{1g}(k), \\ \tilde{Z}_g(k) &= Z_g - \hat{Z}_g(k),\end{aligned}\tag{8.2.25}$$

for the case of multilayered NN. The error vector in activation function is given by

$$\begin{aligned}\tilde{\varphi}_{1f}(k) &= \varphi_{1f}(k) - \hat{\varphi}_{1f}(k), \\ \tilde{\varphi}_{2f}(k) &= \varphi_{2f}(k) - \hat{\varphi}_{2f}(k), \\ \tilde{\varphi}_{3f}(k) &= \varphi_{3f}(k) - \hat{\varphi}_{3f}(k), \\ \tilde{\varphi}_{1g}(k) &= \varphi_{1g}(k) - \hat{\varphi}_{1g}(k), \\ \tilde{\varphi}_{2g}(k) &= \varphi_{2g}(k) - \hat{\varphi}_{2g}(k), \\ \tilde{\varphi}_{3g}(k) &= \varphi_{3g}(k) - \hat{\varphi}_{3g}(k).\end{aligned}\tag{8.2.26}$$

The closed-loop filtered dynamics (8.1.13) become

$$r(k+1) = k_v r(k) + \tilde{W}_f^T(k) \varphi_f(k) + \tilde{W}_g^T(k) \varphi_g(k) u(k) + \epsilon_f(k) + \epsilon_g(k) u(k) + d(k),\tag{8.2.27}$$

using one-layer NN and

$$\begin{aligned}r(k+1) &= k_v r(k) + \tilde{W}_{3f}^T(k) \hat{\varphi}_{3f}(k) + W_{3f}^T(k) \tilde{\varphi}_{3f}(k) + \tilde{W}_{3g}^T(k) \hat{\varphi}_g(k) u(k) + \epsilon_f(k) + \\ &\quad \epsilon_g(k) u(k) + d(k) + W_{3g}^T(k) \tilde{\varphi}_{3g}(k) u(k),\end{aligned}\tag{8.2.28}$$

for the case of multilayered NN.

### 8.2.3 Well-Defined Control Problem

In general boundedness of  $x(k), \hat{W}_f(k)$  and  $\hat{W}_g(k)$  does not indicate the stability of the closed-loop system, because control law (8.1.11) is not well defined when  $\hat{g}(\hat{W}_g, x) = 0$ . Therefore, some attention must be taken to guarantee the boundedness of the controller as well. To overcome the problem, several techniques exist in the literature that assure local or global stability with an additional knowledge. First if the bounds on the function  $g(x)$  are known, then  $\hat{g}(\hat{W}_g, x)$  may be set to a constant and a robust-adaptive controller bypasses this problem. This is not an accurate approach because the bounds on the function are not known *a priori*. If  $g(x)$  is reconstructed by an adaptive scheme then a local solution can be generated by assuming that the initial estimates are close to the actual values and these values do not leave a feasible invariant set in which the  $\hat{g}(\hat{W}_g, x)$  is not equal to zero (Liu and Chen 1991), or lie inside a region of attraction of a stable equilibrium point which forms a feasible set (Kanellakopoulos et al. 1991). Unfortunately with a very good knowledge of the system, it is not easy to pick initial weight values such that NN approximates it. The most popular way to avoid the problem is to project  $\hat{W}_g(k)$  inside an estimated feasible region by properly selecting the weight values (Polycarpou and Ioannou 1991). A shortcoming of this approach is that the actual  $\hat{W}_g(k)$  does not necessarily belong to this set, which then renders a sub-optimal solution.

### 8.2.4 Proposed Controller

In order to guarantee the boundedness of  $\hat{g}(x)$  away from zero for all well-defined values of  $x(k)$ ,  $\hat{W}_f(k)$ , and  $\hat{W}_g(k)$ , the control input in (8.1.11) is selected in terms of another control input,  $u_c(k)$ , and a robustifying term,  $u_r(k)$  as

$$\begin{aligned} u(k) &= u_c(k) + \frac{u_r(k) - u_c(k)}{2} e^{\gamma(|u_c(k)|-s)}, \quad I = 0, \\ &= u_r(k) - \frac{u_r(k) - u_c(k)}{2} e^{-\gamma(|u_c(k)|-s)}, \quad I = 1, \end{aligned} \quad (8.2.29)$$

where

$$u_c(k) = \frac{1}{\hat{g}(x)} (-\hat{f}(x) + v(k)), \quad (8.2.30)$$

and

$$u_r(k) = -\mu \frac{|u_c(k)|}{g} \operatorname{sgn}(r(k)). \quad (8.2.31)$$

The indicator  $I$  in (8.2.29) is

$$\begin{aligned} I &= 1, \text{ if } \hat{g}(x) \geq g \text{ and } |u_c(k)| \leq s \\ &= 0, \text{ otherwise} \end{aligned} \quad (8.2.32)$$

with  $\gamma < \ln \frac{2}{s}$ ,  $\mu > 0$ , and  $s > 0$  design parameters. These modifications in the control input are necessary in order to ensure that the functional estimate  $\hat{g}(x)$  is bounded away from zero.

The intuition behind this controller is as follows. When  $\hat{g}(x) \geq g$  and  $|u_c(k)| \leq s$ , then the total control input is set to  $u_c(k)$ , otherwise the control is smoothly switched to the auxiliary input  $u_r(k)$  due to the additional term in (8.2.29). This results in well-defined control everywhere and the uniform ultimate boundedness of the closed-loop system can be shown by appropriately selecting the NN weight tuning algorithms.

## 8.3 SINGLE-LAYER NN FOR FEEDBACK LINEARIZATION

In this section the one-layer NN is considered as a first step to bridging the gap between discrete-time adaptive control and NN control. As mentioned in the previous chapter, in the one-layer case the tunable NN weights enter in a linear fashion. The one-layer case is treated for radial basis functions in Sanner and Slotine (1992), using a projection algorithm in Polycarpou and Ioannou (1991). This case is treated in Chen and Khalil (1995) using the discrete-time update laws for NN weights and a certainty equivalence assumption. The assumptions made in this chapter are milder than in these works, while rigorous Lyapunov stability analysis is presented similar to the case of continuous-time systems in Chapter 6. In the next section the analysis is extended to the case of general multilayer NN with discrete-time tuning.

A family of one-layer NN weight tuning paradigms that guarantee the stability of the closed-loop system (8.2.27) is presented in this section. It is required to demonstrate that the tracking error  $r(k)$  is suitably small and that the NN weights  $\hat{W}_f(k)$ , and  $\hat{W}_g(k)$  remain bounded. To proceed further, the machinery presented

Table 8.3.1: Discrete-Time Controller Using One-Layer Neural Net: PE Required

---

*The control input is*

$$\begin{aligned} u(k) &= u_c(k) + \frac{u_r(k) - u_c(k)}{2} e^{\gamma(|u_c(k)|-s)}, \quad I = 1, \\ &= u_r(k) - \frac{u_r(k) - u_c(k)}{2} e^{-\gamma(|u_c(k)|-s)}. \quad I = 0 \end{aligned}$$

*The NN weight tuning for  $f(x(k))$  is given by*

$$\hat{W}_f(k+1) = \hat{W}_f(k) + \alpha \varphi_f(k) r^T(k+1)$$

*and the NN weight tuning for  $g(x(k))$  is provided by*

$$\begin{aligned} \hat{W}_g(k+1) &= \hat{W}_g(k) + \beta \varphi_g(k) r^T(k+1), \quad I = 1, \\ &= \hat{W}_g(k), \quad I = 0, \end{aligned} \tag{8.3.1}$$

*with  $\alpha > 0$  and  $\beta > 0$  denoting constant learning rate parameters or adaptation gains.*

---

*in the Lemma 7.2.1 and definition of the PE condition (see Definition 7.2.1) in Chapter 7 should be reviewed. Recall that for one-layer NN the activation functions must provide a basis. See the discussion on functional-link NN in Chapter 4.*

*Stability analysis by Lyapunov's direct method is performed using a novel weight tuning algorithm for a one-layer neural network developed based on the delta rule. These weight tuning paradigms yield a passive neural net, yet persistency of excitation (PE) is generally needed for suitable performance. Specifically, this holds as well as for standard backpropagation in continuous-time case (see Chapter 7). Unfortunately, PE cannot generally be tested for, or guaranteed, in a NN. Therefore, modified tuning paradigms are proposed in subsequent subsections to make the NN robust so that the PE is not needed. For guaranteed stability, it is shown for the case of feedback linearization that the delta-rule-based-weight tuning algorithms must slow down as the NN becomes larger. By employing a projection algorithm it is shown that the tuning rate can be made independent of the NN size.*

### 8.3.1 Weight Updates Requiring Persistence of Excitation

*In the following theorem we present a discrete-time weight tuning algorithm given in Table 8.3.1, based on the filtered tracking error. The algorithm guarantees that both the tracking error and the error in the weight estimates are bounded if a PE condition holds. (This PE requirement is relaxed in Theorem 8.3.2.)*

**Theorem 8.3.1 (One-Layer Discrete-Time NN Controller Requiring PE) :**

Let the desired trajectory  $x_{nd}(k)$  be bounded and the NN functional reconstruction error bound  $\epsilon_{Nf}$  and  $\epsilon_{Ng}$  with the disturbance bound  $d_M$  be known constants. Take the

control input for (8.1.1) as (8.2.29) with weight tuning for  $f(x(k))$  provided by

$$\hat{W}_f(k+1) = \hat{W}_f(k) + \alpha \varphi_f(k) r^T(k+1) \quad (8.3.2)$$

and the weight tuning for  $g(x(k))$  is expressed as

$$\begin{aligned} \hat{W}_g(k+1) &= \hat{W}_g(k) + \beta \varphi_g(k) r^T(k+1), \quad I = 1, \\ &= \hat{W}_g(k), \quad I = 0 \end{aligned} \quad (8.3.3)$$

with  $\alpha > 0$  and  $\beta > 0$  denoting constant learning rate parameters or adaptation gains.

Assume that the initial error in weight estimates for both NN are bounded and let the hidden-layer output vectors,  $\varphi_f(k)$  and  $\varphi_g(k)u_c(k)$ , be persistently exciting. Then the filtered tracking error  $r(k)$  and the error in weight estimates  $\tilde{W}_f(k)$ , and  $\tilde{W}_g(k)$  are UUB, with the bounds specifically given by (8.3.36) or (8.3.56) with (8.3.18) or (8.3.58) and (8.3.19) provided the following conditions hold:

$$(1) \quad \beta \|\varphi_g(k)u_c(k)\|^2 = \beta \|\varphi_g(k)\|^2 < 1, \quad (8.3.4)$$

$$(2) \quad \alpha \|\varphi_f(k)\|^2 < 1, \quad (8.3.5)$$

$$(3) \quad \eta < 1, \quad (8.3.6)$$

$$(4) \quad \max(a_4, b_0) < 1, \quad (8.3.7)$$

where  $\eta$  is given as

$$\eta = \alpha \|\varphi_f(k)\|^2 + \beta \|\varphi_g(k)u_c(k)\|^2 \quad (8.3.8)$$

for  $I = 1$ , and for  $I = 0$ , the parameter  $\eta$  is defined as

$$\eta = \alpha \|\varphi_f(k)\|^2 \quad (8.3.9)$$

and with  $a_4, b_0$  design parameters chosen using the gain matrix  $k_{vmax}$  and the relationship is presented during the proof.

Note: The parameters  $\alpha, \beta$  and  $\eta$  are dependent upon the trajectory.

Proof:

(Note, in the proof  $g(x(k))$  is also referred to as  $g(x)$ .) Define the Lyapunov function candidate

$$J = r^T(k)r(k) + \frac{1}{\alpha} \text{tr}(\tilde{W}_f^T(k)\tilde{W}_f(k)) + \frac{1}{\beta} \text{tr}(\tilde{W}_g^T(k)\tilde{W}_g(k)). \quad (8.3.10)$$

The first difference is given by

$$\begin{aligned} \Delta J &= r^T(k+1)r(k+1) - r^T(k)r(k) + \frac{1}{\alpha} \text{tr}(\tilde{W}_f^T(k+1)\tilde{W}_f(k+1) - \tilde{W}_f^T(k)\tilde{W}_f(k)) \\ &\quad + \frac{1}{\beta} \text{tr}(\tilde{W}_g^T(k+1)\tilde{W}_g(k+1) - \tilde{W}_g^T(k)\tilde{W}_g(k)). \end{aligned} \quad (8.3.11)$$

Region I:  $|\hat{g}| \geq g$  and  $|u_c| \leq s$ .

The filtered error dynamics (8.2.27) can be rewritten as

$$\begin{aligned} r(k+1) &= k_v r(k) + (f(x(k)) - \hat{f}(x(k))) + (g(x(k)) - \hat{g}(x(k)))u_c(k) \\ &\quad + d(k) + g(x(k))u_d(k) \end{aligned} \quad (8.3.12)$$

where  $u_d(k) = u(k) - u_c(k)$ . Substituting (8.3.2) and (8.3.3) in (8.3.12), one obtains

$$\begin{aligned} r(k+1) &= k_v r(k) + \tilde{W}_f^T(k)\varphi_f(k) + \tilde{W}_g^T(k)\varphi_g(k)u_c(k) + \epsilon(k) \\ &\quad + d(k) + g(x(k))u_d(k), \end{aligned} \quad (8.3.13)$$

where

$$\epsilon(k) = \epsilon_f(k) + \epsilon_g(k)u_c(k). \quad (8.3.14)$$

Equation (8.3.13) can be rewritten

$$r(k+1) = k_v r(k) + \bar{e}_f^T(k) + \bar{e}_g^T(k) + \epsilon(k) + d(k) + g(x(k))u_d(k) \quad (8.3.15)$$

where

$$\bar{e}_f(k) = \tilde{W}_f^T(k)\varphi_f(k), \quad (8.3.16)$$

$$\bar{e}_g(k) = \tilde{W}_g^T(k)\varphi_g(k)u_c(k). \quad (8.3.17)$$

Note that for the system defined in (8.3.15), the input  $u_c(k) \in R^{m \times 1}$ ,  $\varphi_f(k) \in R^{mn \times 1}$  where  $\varphi_i(k) \in R^{n \times 1}; i = 1, \dots, m$ , and  $\varphi_g(k) \in R^{mn \times n}$ , in which each  $\varphi_i(k) \in R^{n \times n}; i = 1, \dots, m$ . The error in dynamics for the weight update laws are given for this region as

$$\begin{aligned} \tilde{W}_f(k+1) &= (I - \alpha\varphi_f(k)\varphi_f^T(k))\tilde{W}_f(k) - \alpha\varphi_f(k)(k_v r(k) + \bar{e}_g(k) \\ &\quad + g(x(k))u_d(k) + \epsilon(k) + d(k))^T \end{aligned} \quad (8.3.18)$$

and

$$\begin{aligned} \tilde{W}_g(k+1) &= (I - \beta\varphi_g(k)\varphi_g^T(k))\tilde{W}_g(k) - \beta\varphi_g(k)(k_v r(k) + \bar{e}_f(k) \\ &\quad + g(x(k))u_d(k) + \epsilon(k) + d(k))^T. \end{aligned} \quad (8.3.19)$$

Substituting (8.3.15), (8.3.18), and (8.3.19) in (8.3.11) and simplifying one obtains

$$\begin{aligned} \Delta J &= -r^T(k)[I - k_v^T k_v]r(k) + 2\eta(k_v r(k))^T(g(x)u_d(k) + \epsilon(k) + d(k)) \\ &\quad (1 + \eta)(g(x)u_d(k) + \epsilon(k) + d(k))^T(g(x)u_d(k) + \epsilon(k) + d(k)) \\ &\quad - (1 - \eta) \|(\bar{e}_f(k) + \bar{e}_g(k)) - \frac{\eta}{(1 - \eta)}(k_v r(k) + g(x)u_d(k) + \epsilon(k) + d(k))\|^2 \\ &\quad + \frac{\eta}{(1 - \eta)}(k_v r(k) + g(x)u_d(k) + \epsilon(k) \\ &\quad + d(k))^T(k_v r(k) + g(x)u_d(k) + \epsilon(k) + d(k)) \end{aligned} \quad (8.3.20)$$

where

$$\eta = \alpha \| \varphi_f(k) \|^2 + \beta \| \varphi_g(k) \|^2. \quad (8.3.21)$$

Equation (8.3.20) can be rewritten as

$$\begin{aligned} \Delta J &= -(1 - a_1 k_{vmax}^2) \| r(k) \|^2 + 2a_2 k_{vmax} \| r(k) \| (g(x)u_d(k) + \epsilon(k) + d(k)) + \\ &\quad a_3(g(x)u_d(k) + \epsilon(k) + d(k))^T(g(x)u_d(k) + \epsilon(k) + d(k)) \\ &\quad - (1 - \eta) \|(\bar{e}_f(k) + \bar{e}_g(k)) - \frac{\eta}{(1 - \eta)}(k_v r(k) + g(x)u_d(k) + \epsilon(k) + d(k))\|^2, \end{aligned} \quad (8.3.22)$$

where

$$a_1 = 1 + \eta + \frac{\eta}{(1 - \eta)}, \quad (8.3.23)$$

$$a_2 = \eta + \frac{\eta}{(1 - \eta)}, \quad (8.3.24)$$

$$a_3 = 1 + \eta + \frac{\eta}{(1 - \eta)}. \quad (8.3.25)$$

Now applying the condition on the function  $g(x)$  on a compact set, one can conclude that

$$\| g(x) \| \leq C_{01} + C_{12} \| r(k) \| \quad (8.3.26)$$

with  $C_{01}, C_{12}$  computable constants. Now in this region, the bound on  $u_d(k)$  can be obtained as

$$\begin{aligned} \| u_d(k) \| &\leq \| u(k) - u_c(k) \| \\ &\leq \left\| \frac{(u_r(k) - u_c(k))}{2} e^{\gamma(|u_c(k)|-s)} \right\|. \end{aligned} \quad (8.3.27)$$

In this region since  $|u_c(k)| \leq s$ , and the other input  $u_r(k)$  is given by (8.2.31), the bound in (8.3.27) can be obtained as a constant since all the terms on the right side are bounded and this bound is denoted by

$$\| u_d(k) \| \leq C_2. \quad (8.3.28)$$

Now the bound for  $g(x)u_d(k)$  is obtained as

$$\begin{aligned} \| g(x)u_d(k) \| &\leq C_2(C_{01} + C_{12} \| r(k) \|) \\ &\leq C_0 + C_1 \| r(k) \|. \end{aligned} \quad (8.3.29)$$

Using the bound presented in (8.3.29) for  $g(x)u_d(k)$ , the first difference of the Lyapunov function (8.3.22) is rewritten as

$$\begin{aligned} \Delta J &= -(1 - a_1 k_{vmax}^2) \| r(k) \|^2 + 2a_2 k_{vmax} \| r(k) \| (C_0 + C_1 \| r(k) \| + \epsilon_N + d_M) \\ &\quad + a_3 (C_0 + C_1 \| r(k) \| + \epsilon_N + d_M)^T (C_0 + C_1 \| r(k) \| + \epsilon_N + d_M) \\ &\quad - (1 - \eta) \| (\bar{e}_f(k) + \bar{e}_g(k)) \\ &\quad - \frac{\eta}{(1 - \eta)} (k_v r(k) + g(x)u_d(k) + \epsilon(k) + d(k)) \|^2, \end{aligned} \quad (8.3.30)$$

with the bound for  $\epsilon(k)$  obtained as

$$\begin{aligned} \| \epsilon(k) \| &\leq \| \epsilon_f \| + \| \epsilon_g u_c(k) \| \\ &\leq (\epsilon_{Nf} + s\epsilon_{Ng}) \\ &\leq \epsilon_N. \end{aligned} \quad (8.3.31)$$

Simplifying (8.3.30), one obtains

$$\begin{aligned} \Delta J &= -(1 - a_4) \| r(k) \|^2 + 2a_5 \| r(k) \| + a_6 \\ &\quad - (1 - \eta) \| (\bar{e}_f(k) + \bar{e}_g(k)) \\ &\quad - \frac{\eta}{(1 - \eta)} (k_v r(k) + g(x)u_d(k) + \epsilon(k) + d(k)) \|^2, \end{aligned} \quad (8.3.32)$$

where

$$a_4 = a_1 k_{vmax}^2 + 2a_2 C_1 k_{vmax} + a_3 C_1, \quad (8.3.33)$$

$$a_5 = a_2 k_{vmax} (\epsilon_N + d_M + C_0) + a_3 C_1 (\epsilon_N + d_M) + a_3 C_0 C_1, \quad (8.3.34)$$

and

$$a_6 = a_3 C_0^2 + 2a_3 C_0 (\epsilon_N + d_M) + (\epsilon_N + d_M)^2. \quad (8.3.35)$$

The second term in (8.3.32) is always negative as long as the condition (8.3.4) through (8.3.7) hold. Since  $a_4, a_5$  and  $a_6$  are positive constants,  $\Delta J \leq 0$  as long as (8.3.4) through (8.3.7) hold and

$$\| r(k) \| > \delta_{r1} \quad (8.3.36)$$

where

$$\delta_{r1} > \frac{1}{(1-a_4)} [a_3 + \sqrt{a_5^2 + a_6(1-a_4)}]. \quad (8.3.37)$$

$|\sum_{k=k_0}^{\infty} \Delta J(k)| = |J(\infty) - J(0)| < \infty$  since  $\Delta J \leq 0$  as long as (8.3.4) through (8.3.7) hold. The definition of  $J$  and inequality (8.3.36) imply that every initial condition in the set  $\chi$  and will evolve entirely within  $\chi$ . In other words, whenever the tracking error  $\| r(k) \|$  is outside the region defined by (8.3.36),  $J(r(k), \tilde{W}_f(k), \tilde{W}_g(k))$  will decrease. This further implies that the tracking error  $r(k)$  is UUB for all  $k \geq 0$  and it remains to show that the weight estimation errors,  $\tilde{W}_f(k)$  and  $\tilde{W}_g(k)$  or equivalently  $\hat{W}_f(k)$  and  $\hat{W}_g(k)$  are bounded.

Generally in order to show the boundedness of the weight estimation errors, one uses the error in weight updates (8.3.18) and (8.3.19), tracking error bound (8.3.36), the PE condition and Lemma presented in Chapter 7. Using (8.3.18) and (8.3.19) it can be realized that the output of each neural network is driving the other. Therefore, the boundedness of the tracking error, the PE condition and Lemma in Chapter 7 are necessary but not sufficient. If the initial weight estimation errors for both NN are considered to be bounded, then applying the bound for the tracking error (8.3.36), the PE condition and Lemma in Chapter 7, one can show that the weight estimation errors  $\tilde{W}_f(k)$  and  $\tilde{W}_g(k)$  or equivalently  $\hat{W}_f(k)$  and  $\hat{W}_g(k)$  are bounded. This concludes the boundedness of both tracking error and weight estimates for both NN in this region. On the other hand, a similar and elegant way to show the boundedness of tracking error and weight estimates is to apply passivity theory. The proof using passivity theory is shown in Section 8.5.

Region II:  $|\hat{g}(x)| \leq g$  and  $|u_c(k)| > s$ .

Since the input  $u_c(k)$  may not be defined in this region, because of notational simplicity, we will use it in the form of either  $\hat{g}(x(k))u_c(k)$  or  $u_c(k)e^{-\gamma(|u_c(k)|-s)}$ . Therefore, in this region, the tracking error system given in (8.3.13) is rewritten as

$$r(k+1) = k_v r(k) + \bar{e}_f^T(k) + \overline{g(x)u_d(k)} + \epsilon_f(k) + d(k), \quad (8.3.38)$$

where

$$\overline{g(x)u_d(k)} = g(x)u(k) - \hat{g}(x(k))u_c(k). \quad (8.3.39)$$

Note the extremum of the function  $ye^{-\gamma y}$  for  $\forall y > 0$  can be found as a solution to the following equation

$$\frac{\partial(y^{-\gamma y})}{\partial y} = (1-\gamma y)e^{-\gamma y} = 0, \quad (8.3.40)$$

which is  $y = \frac{1}{\gamma}$ , and it is a maximum. Evaluating the function  $u_c(k)e^{-\gamma u_c(k)}$  yields an upper bound for  $u_c(k) = \frac{1}{\gamma e}$  and this bound is used in the forthcoming set of equations.

Let us compute the bound for  $g(x)u_c(k)$  and  $\hat{g}(x)u_c(k)$ . Consider the following cases in this region when  $|u_c(k)| \leq s$  and  $|u_c(k)| > s$ . The bound on  $u(k)$  from (8.2.27) can be written for this region as

$$|u(k)| \leq \frac{u_r(k) - u_c(k)}{2} e^{-\gamma(|u_c(k)|-s)}. \quad (8.3.41)$$

Using (8.2.31) for  $u_r(k)$ , Equation (8.3.41) can be rewritten as

$$|u(k)| \leq \frac{1}{2} \left( \frac{\mu}{g} |u_c(k)| + |u_c(k)| \right) e^{-\gamma(|u_c(k)| - s)}. \quad (8.3.42)$$

If  $|u_c(k)| \leq s$ , then  $e^{\gamma s} \leq 2$ , Equation (8.3.42) can be written as

$$|u(k)| \leq d_1, \quad (8.3.43)$$

where

$$d_1 = \left( \frac{\mu}{g} s + d_0 s \right), \quad (8.3.44)$$

bounded above by some positive constant. On the other hand, if  $|u_c(k)| > s$ , Equation (8.3.42) can be expressed as

$$|u(k)| \leq d_1, \quad (8.3.45)$$

where

$$d_1 = \frac{1}{2} \left( \frac{\mu}{g} \frac{1}{\gamma e} + d_0 \frac{1}{\gamma e} \right). \quad (8.3.46)$$

Note here for simplicity the upper bound for  $|u(k)|$  is denoted as  $d_1$  for both cases. Now the bound for  $g(x)u(k)$  can be obtained as

$$\|g(x)u(k)\| \leq C_0 + C_1 \|r(k)\|, \quad (8.3.47)$$

where  $C_0 = d_1 C_{01}$  and  $C_1 = d_1 C_{12}$ . Similarly the bound for  $\hat{g}(x)u_c(k)$  can be deduced as

$$\begin{aligned} \|\hat{g}(x)u_c(k)\| &\leq g s \text{ If } |u_c(k)| \leq s, \\ &\leq \frac{g}{\gamma e} \text{ If } |u_c(k)| > s, \end{aligned} \quad (8.3.48)$$

which is denoted as  $C_2$ . Using the individual upper bounds of  $g(x)u(k)$  and  $\hat{g}(x)u_c(k)$ , the upper bound for  $|g(x)u_d(k)|$  can be obtained as

$$|\overline{g(x)u_d(k)}| = |g(x)u(k) - \hat{g}u_c(k)| \leq C_3 + C_4 \|r(k)\|, \quad (8.3.49)$$

where  $C_3 = C_0 + C_2$  and  $C_4 = C_1$ . Now using the Lyapunov function (8.3.10), the first difference (8.3.11) after manipulation can be obtained for this region as

$$\begin{aligned} \Delta J &= -r(k)^T (I - k_v^T k_v) r(k) + 2(k_v r(k) + \overline{g(x)u_d(k)} + \epsilon_f(k) + d(k))^T (\overline{g(x)u_d(k)} \\ &\quad + \epsilon_f(k) + d(k)) \\ &\quad + \frac{1}{(1 - \alpha \varphi_f^T(k) \varphi_f(k))} (k_v r(k) + \overline{g(x)u_d(k)} + \epsilon_f(k) + d(k))^T (\overline{g(x)u_d(k)} \\ &\quad + \epsilon_f(k) + d(k)) \\ &\quad - (1 - \eta) \|\bar{e}_f(k) - \frac{\eta}{(1 - \eta)} (k_v r(k) + \overline{g(x)u_d(k)} + \epsilon_f(k) + d(k))\|^2, \end{aligned} \quad (8.3.50)$$

where  $\eta$  is given in (8.3.9). Substituting for  $\overline{g(x)u_d(k)}$  from (8.3.49) in (8.3.50) and rearranging terms in (8.3.50) results in

$$\begin{aligned} \Delta J &= -(1 - b_0) \|r(k)\|^2 + 2b_1 \|r(k)\| + b_2 \\ &\quad - (1 - \eta) \|\bar{e}_f(k) - \frac{\eta}{(1 - \eta)} (k_v r(k) + \overline{g(x)u_d(k)} + \epsilon_f(k) + d(k))\|^2 \end{aligned} \quad (8.3.51)$$

where

$$b_0 = k_{vmax}^2 + 2C_4(C_4 + k_{vmax}) + \frac{(C_4 + k_{vmax})^2}{(1 - \alpha \|\varphi_f(k)\|)^2}, \quad (8.3.52)$$

$$\begin{aligned} b_1 &= C_3(C_4 + k_{vmax}) + C_3C_4 + (C_4 + k_{vmax})(\epsilon_{Nf} + d_M) + \\ &\quad \frac{C_3(C_4 + k_{vmax})}{(1 - \alpha \|\varphi_f(k)\|^2)} + \frac{(C_4 + k_{vmax})(\epsilon_{Nf} + d_M)}{(1 - \alpha \|\varphi_f(k)\|^2)}, \end{aligned} \quad (8.3.53)$$

$$\begin{aligned} b_2 &= 2C_3^2 + 2C_3(\epsilon_{Nf} + d_M) + (\epsilon_{Nf} + d_M)^2 \\ &\quad \frac{C_3^2 + 2C_3(\epsilon_{Nf} + d_M) + (\epsilon_{Nf} + d_M)^2}{(1 - \alpha \|\varphi_f(k)\|^2)} \end{aligned} \quad (8.3.54)$$

and

$$\|\epsilon_f(k)\| \leq \epsilon_{Nf}. \quad (8.3.55)$$

The second term in (8.3.51) is always negative as long as the conditions (8.3.4) through (8.3.7) hold. Since  $b_0, b_1$  and  $b_2$  are positive constants,  $\Delta J \leq 0$  as long as

$$\|r(k)\| > \delta_{r2}, \quad (8.3.56)$$

with

$$\delta_{r2} = \frac{1}{(1 - b_0)} [b_1 + \sqrt{b_1^2 + b_2(1 - b_0)}]. \quad (8.3.57)$$

One has  $|\sum_{k=k_0}^{\infty} \Delta J(k)| = |J(\infty) - J(0)| < \infty$  since  $\Delta J \leq 0$  as long as (8.3.4) through (8.3.7) hold. The definition of  $J$  and inequality (8.3.56) imply that every initial condition in the set  $\chi$  will evolve entirely within  $\chi$ . In other words, whenever the tracking error  $\|r(k)\|$  is outside the region defined by (8.3.56),  $J(r(k), \tilde{W}_f(k), \tilde{W}_g(k))$  will decrease. This further implies that the tracking error  $r(k)$  is UUB for all  $k \geq 0$  and it remains to show that the weight estimation errors  $\tilde{W}_f(k)$ , or equivalently  $\hat{W}_f(k)$ , are bounded.

In order to show the boundedness of the weight estimation errors, one uses the error in weight updates (8.3.18) for  $f(\cdot)$ , the tracking error bound (8.3.56), the PE condition and Lemma in Chapter 7. Since the weight estimates for  $\hat{g}(x)$  is not updated in this region, the boundedness of the weight estimates for  $\hat{g}(x)$  need not to be shown. However, to show the boundedness of the weight estimates for  $\hat{f}(x)$ , the dynamics relative to the error in weight estimates using (8.3.18) for this region are given by

$$\begin{aligned} \tilde{W}_f(k+1) &= (I - \alpha \varphi_f(k) \varphi_f^T(k)) \tilde{W}_f(k) - \alpha \varphi_f(k) (k_v r(k) + C_3 + C_4 \|r(k)\| \\ &\quad + \epsilon_f(k) + d(k))^T, \end{aligned} \quad (8.3.58)$$

where the tracking error  $r(k)$  is shown to be bounded. Applying the PE condition and Lemma described in Chapter 7, the boundedness of weight estimation errors  $\tilde{W}_f(k)$  or equivalently  $\hat{W}_f(k)$  are guaranteed. Let us denote the bound by  $\delta_{f2}$ . This concludes the boundedness of both tracking error and weight estimates for both NN in this region.

Reprise:

Combining the results from region I and II, one can readily set  $\delta_r = \max(\delta_{r1}, \delta_{r2})$ ,  $\delta_f = \max(\delta_{f1}, \delta_{f2})$ , and  $\delta_g$ . Thus for both regions, if  $\|r(k)\| > \delta_r$ , then  $\Delta J \leq 0$  and  $u(k)$  is bounded. Let us denote  $(\|r(k)\|, \|\tilde{W}_f(k)\|, \|\tilde{W}_g(k)\|)$  by a new coordinate variables  $(\xi_1, \xi_2, \xi_3)$ . Define the region

$$\Xi : \xi | \xi_1 < \delta_r, \xi_2 < \delta_f, \xi_3 < \delta_g,$$

then there exists an open set

$$\Omega : \xi | \xi_1 < \bar{\delta}_r, \xi_2 < \bar{\delta}_f, \xi_3 < \bar{\delta}_g,$$

where  $\bar{\delta}_i > \delta_i$  implies that  $\Xi \subset \Omega$ . In other words, we have proved that whenever  $\xi_i > \delta_i$ , then  $J(\xi)$  will not increase and will remain in the region  $\Omega$  which is an invariant set. Therefore all the signals in the closed-loop system remain bounded. This concludes the proof.  $\square$

*In applications, the right-hand sides of (8.3.36) or (8.3.56), (8.3.18) or (8.3.58) and (8.3.19) may be taken as practical bounds on the norms of the error  $r(k)$  and the weight estimation errors  $\tilde{W}_f(k)$  and  $\tilde{W}_g(k)$ . Since the target weight values are bounded, it follows that the NN weights,  $\hat{W}_f(k)$  and  $\hat{W}_g(k)$ , provided by the tuning algorithms are bounded; hence the control input is bounded.*

*Note from (8.3.36) or (8.3.56) that the tracking error increases with the NN reconstruction error bound  $\epsilon_N$  and the disturbance bound  $d_M$ , yet small tracking errors (but not arbitrary small) may be achieved by selecting small gains  $k_v$ . In other words, placing the closed-loop poles closer to the origin inside the unit circle forces smaller tracking errors. Again as in Chapter 7, selecting  $k_{v\max} = 0$  results in a deadbeat controller, but it should be avoided as it is not robust.*

*It is important to note that the problem of initializing the net weights (referred to as symmetric breaking (Mpitsos and Burton 1992)) occurring in other techniques in the literature does not arise, since when  $\hat{W}_f(0)$  and  $\hat{W}_g(0)$  are taken as zero the PD term  $k_v r(k)$  stabilizes the plant on an interim basis for a restricted class of nonlinear systems such as robotic systems. Thus, the NN controller requires no off-line learning phase.*

### 8.3.2 Projection Algorithm

*The adaptation gains  $\alpha > 0$  and  $\beta > 0$ , are constant parameters in the update laws presented in (8.3.18) and (8.3.19). These update laws correspond to the delta rule, also referred to as the Widrow-Hoff rule (Mpitsos and Burton 1992). This reveals that the update tuning mechanisms employing the delta rule have a major drawback. In fact, using (8.3.4), the upper bound on the adaptation gain for  $g(x(k))$  can be obtained as*

$$\beta < \frac{1}{\| \varphi_g(k) \|^2}, \quad (8.3.59)$$

*since  $\varphi_g(k) \in \Re^{N_2}$ , with  $N_2$  the number of hidden-layer neurons. It is evident that the upper bound on the adaptation gain  $\beta$  depends upon the the number of hidden-layers neurons. Specifically, if there are  $N_2$  hidden-layer neurons and the maximum value of the each hidden-node output is taken as unity (as for the sigmoid), then the bounds on the adaptation gain to assure stability of the closed-loop system are given by*

$$0 < \beta < \frac{1}{N_2}. \quad (8.3.60)$$

*In other words, the upper bound on the adaptation gain for the case of delta rule decreases with an increase in the number of hidden-layer nodes, so that learning*

must slow down for guaranteed performance. The phenomenon of large NN requiring very slow learning rates has often been encountered in the practical NN literature (Mpitsos and Burton 1992) but never explained adequately (Jagannathan and Lewis 1996a).

This major drawback can be easily overcome by modifying the update rule at each layer to obtain a projection algorithm (Goodwin and Sin 1984). To wit, replace the constant adaptation gain at each layer by

$$\beta = \frac{\xi}{\zeta + \|\varphi_g(k)\|^2}, \quad (8.3.61)$$

where

$$\zeta > 0, \quad (8.3.62)$$

and

$$0 < \xi < 1, \quad (8.3.63)$$

are constants. Note that  $\xi$  is now the new adaptation gain and it is always true that

$$\frac{\xi}{\zeta + \|\varphi_g(k)\|^2} \|\varphi_g(k)\|^2 < 1, \quad (8.3.64)$$

hence guaranteeing (8.3.4) for every  $N_2$ . Similarly, using (8.3.4) and (8.3.6), it can be shown that the adaptation gain  $\alpha$  should satisfy  $\alpha < \frac{1}{\|\varphi_f(k)\|^2}$  as in (8.3.5).

Note that now for guaranteed closed-loop stability, it is necessary that the hidden-layer outputs  $\varphi_f(k)$  and  $\varphi_g(k)$  be PE. Equations (8.3.18) and (8.3.19) are nothing but the delta-rule-based weight tuning algorithms for the one-layer case. Then, the PE condition is required to guarantee boundedness of the weight estimates. However, it is very difficult to verify the PE of the hidden-layer output functions  $\varphi_f(k)$  and  $\varphi_g(k)$ , and this problem is compounded due to the presence of hidden-layers in the case of multilayered neural network. In the next section, improved weight tuning paradigms are presented so that PE is not required.

### 8.3.3 Weight Updates not Requiring Persistence of Excitation

Approaches such as  $\sigma$ -modification (Polycarpou and Ioannou 1991) or  $\epsilon$ -modification (Narendra and Annaswamy 1987) are available for the robust adaptive control of continuous systems wherein the persistency of excitation condition is not needed. On the other hand, modification to the standard weight tuning mechanisms in discrete-time to avoid the necessity of PE is also investigated in Jagannathan and Lewis (1996b) and Jagannathan (1996a).

In Jagannathan (1996b) an approach similar to  $\epsilon$ -modification was derived for discrete-time NN for feedback linearization. The following theorem from that paper shows the tuning algorithms that do not require persistency of excitation. The controller derived therein is given in Table 8.3.2.

Table 8.3.2: Discrete-Time Controller Using One-layer Neural Net: PE not Required

$$\begin{aligned} u(k) &= u_c(k) + \frac{u_r(k) - u_c(k)}{2} e^{\gamma(|u_c(k)|-s)}, \quad I = 1, \\ &= u_r(k) - \frac{u_r(k) - u_c(k)}{2} e^{-\gamma(|u_c(k)|-s)}, \quad I = 0. \end{aligned}$$

The NN weight tuning for  $f(x(k))$  is given by

$$\hat{W}_f(k+1) = \hat{W}_f(k) + \alpha \varphi_f(k) r^T(k+1) - \delta \|I - \alpha \varphi_f(k) \varphi_f^T(k)\| \hat{W}_f(k)$$

and the NN weight tuning for  $g(x(k))$  is provided by

$$\begin{aligned} \hat{W}_g(k+1) &= \hat{W}_g(k) + \beta \varphi_g(k) r^T(k+1) - \rho \|I - \beta \varphi_g(k) \varphi_g^T(k)\|, \quad I = 1, \\ &= \hat{W}_g(k), \quad I = 0 \end{aligned}$$

with  $\alpha = \frac{\xi_f}{\zeta_f + \|\varphi_f(k)\|^2}$  and  $\beta = \frac{\xi_g}{\zeta_g + \|\varphi_g(k)\|^2}$  where  $\zeta_f > 0, \zeta_g > 0$  and  $0 < \xi_f < 1, 0 < \xi_g < 1$  denoting learning rate parameters or adaptation gains.

**Theorem 8.3.2 (One-Layer Discrete-Time NN Feedback Lin. without PE) :**

Assume the hypotheses presented in Theorem 8.3.1, and consider the modified weight tuning algorithms provided for  $f(x(k))$  by :

$$\begin{aligned} \hat{W}_f(k+1) &= \hat{W}_f(k) + \alpha \varphi_f(k) r^T(k+1) \\ &\quad - \delta \|I - \alpha \varphi_f(k) \varphi_f^T(k)\| \hat{W}_f(k) \end{aligned} \quad (8.3.65)$$

and the NN weight updates for  $g(x(k))$  are provided by

$$\begin{aligned} \hat{W}_g(k+1) &= \hat{W}_g(k) + \beta \varphi_g(k) r^T(k+1) - \rho \|I - \beta \varphi_g(k) \varphi_g^T(k)\| \hat{W}_g(k), \quad I = 1, \\ &= \hat{W}_g(k), \quad I = 0, \end{aligned} \quad (8.3.66)$$

with  $\alpha > 0, \beta > 0, \delta > 0$  and  $\rho > 0$  design parameters. Then the filtered tracking error  $r(k)$  and the NN weight estimates  $\hat{W}_f(k)$  and  $\hat{W}_g(k)$  are UUB, with the bounds specifically given by Equations (8.3.89) or (8.3.110), (8.3.93) or (8.3.114) and (8.3.97) provided the following conditions hold:

$$(1) \quad \beta \|\varphi_g(k) u_c(k)\| = \beta \|\varphi_g(k)\|^2 < 1, \quad (8.3.67)$$

$$(2) \quad \alpha \|\varphi_f(k)\|^2 < 1, \quad (8.3.68)$$

$$(3) \quad \eta + \max(P_1, P_3, P_4) < 1 \quad (8.3.69)$$

$$(4) \quad 0 < \delta < 1, \quad (8.3.70)$$

$$(5) \quad 0 < \rho < 1, \quad (8.3.71)$$

$$(6) \quad \max(a_2, b_0) < 1, \quad (8.3.72)$$

with  $P_1, P_3$ , and  $P_4$  constants which depend upon  $\eta, \delta$  and  $\rho$  where

$$\begin{aligned} \eta &= \alpha \|\varphi_f(k)\|^2 + \beta \|\varphi_g(k) u_c(k)\|^2 = \alpha \|\varphi_f(k)\|^2 + \beta \|\varphi_g(k)\|^2, \quad I = 1, \\ &= \alpha \|\varphi_f(k)\|^2, \quad I = 0, \end{aligned} \quad (8.3.73)$$

and  $a_2, b_0$  design parameters chosen using the gain matrix  $k_v$ .

Note: The parameters  $\alpha, \beta$  and  $\eta$  depend upon the trajectory.

Proof:

Region I:  $|\hat{g}(x(k))| \geq g$  and  $|u_c(k)| \leq s$ .

Select the Lyapunov function candidate (8.3.10) whose first difference is given by (8.3.11). The error in dynamics for the weight update laws are given for this region as

$$\begin{aligned}\tilde{W}_f(k+1) &= (I - \alpha\varphi_f(k)\varphi_f^T(k))\tilde{W}_f(k) - \alpha\varphi_f(k)(k_v r(k) + \bar{e}_g(k) + g(x)u_d(k) \\ &\quad - \epsilon(k) + d(k))^T + \delta \|I - \alpha\varphi_f(k)\varphi_f^T(k)\| \hat{W}_f(k)\end{aligned}\quad (8.3.74)$$

and

$$\begin{aligned}\tilde{W}_g(k+1) &= (I - \beta\varphi_g(k)\varphi_g^T(k))\tilde{W}_g(k) - \beta\varphi_g(k)(k_v r(k) + \bar{e}_f(k) + g(x)u_d(k) \\ &\quad - \epsilon(k) + d(k))^T + \rho \|I - \beta\varphi_g(k)\varphi_g^T(k)\| \hat{W}_g(k)\end{aligned}\quad (8.3.75)$$

Substituting (8.3.74) and (8.3.75), in (8.3.11) and combining, rewriting and completing the squares and simplifying we get

$$\begin{aligned}\Delta J &= -r(k)^T [I - (2 + \eta)k_v^T k_v] r(k) + 2(2 + \eta)(k_v r(k))^T (g(x)u_d(k) + \epsilon(k) + d(k)) \\ &\quad + (2 + \eta)(g(x)u_d(k) + \epsilon(k) + d(k))^T (g(x)u_d(k) + \epsilon(k) + d(k)) \\ &\quad + 2P_2 \|k_v r(k) + g(x)u_d(k) + \epsilon(k) + d(k)\| + 2\eta(g(x)u_d(k))^T (\epsilon(k) + d(k)) \\ &\quad + 2\eta\epsilon(k)d(k) - (1 - \eta - P_3) \|\bar{e}_f(k)\|^2 - (1 - \eta - P_4) \|\bar{e}_g(k)\|^2 \\ &\quad - 2(1 - \eta - P_1) \|\bar{e}_f(k)\| \|\bar{e}_g(k)\| - \frac{1}{\eta} \|I - \alpha\varphi_f(k)\varphi_f^T(k)\|^2 [\delta(2 - \delta) \|\tilde{W}_f(k)\|^2 \\ &\quad - 2\delta(1 - \delta) \|\tilde{W}_f(k)\| W_{fmax} - \delta^2 W_{fmax}^2] \\ &\quad - \frac{1}{\beta} \|I - \beta\varphi_g(k)\varphi_g^T(k)\|^2 [\rho(2 - \rho) \|\tilde{W}_g(k)\|^2 \\ &\quad - 2\rho(1 - \rho) \|\tilde{W}_g(k)\| W_{gmax} - \rho^2 W_{gmax}^2],\end{aligned}\quad (8.3.76)$$

where  $\eta$  is given in (8.3.69) and

$$P_1 = 2(\delta \|I - \alpha\varphi_f(k)\varphi_f(k)^T\| + \rho \|I - \beta\varphi_g(k)\varphi_g(k)^T\|).\quad (8.3.77)$$

$$P_2 = 2(\delta \|I - \alpha\varphi_f(k)\varphi_f(k)^T\| W_{fmax} \varphi_{fmax} + \rho \|I - \beta\varphi_g(k)\varphi_g(k)^T\| W_{gmax} \varphi_{gmax}).\quad (8.3.78)$$

$$P_3 = (\eta + \delta \|I - \alpha\varphi_f(k)\varphi_f(k)^T\|)^2,\quad (8.3.79)$$

and

$$P_4 = (\eta + \rho \|I - \beta\varphi_g(k)\varphi_g(k)^T\|)^2.\quad (8.3.80)$$

Now in this region, the bound on  $u_d(k)$  can be obtained as

$$\begin{aligned}\|u_d(k)\| &\leq \|u(k) - u_c(k)\| \\ &\leq \left\| \frac{u_r(k) - u_c(k)}{2} e^{\gamma(|u_c(k)| - s)} \right\|.\end{aligned}\quad (8.3.81)$$

In this region, since  $|u_c(k)| \leq s$ , and the auxiliary input  $u_r(k)$  is given by (8.2.31), the bound in (8.3.81) can be taken as a constant since all the terms on the right side are bounded and this bound is denoted as

$$\|u_d(k)\| \leq C_2.\quad (8.3.82)$$

Then the bound for  $g(x)u_d(k)$  can be written as (8.3.29). Using the bound for  $g(x)u_d(k)$  and substituting in (8.3.76), and completing the squares for  $\|\tilde{W}_f(k)\|$  and  $\|\tilde{W}_g(k)\|$  we obtain

$$\begin{aligned}\Delta J \leq & -(1-a_2) \|r(k)\|^2 + 2a_3 \|r(k)\| + a_4 \\ & -(1-\eta-P_3) \|\bar{e}_f(k)\|^2 - (1-\eta-P_4) \|\bar{e}_g(k)\|^2 \\ & -2(1-\eta-P_1) \|\bar{e}_f(k)\| \|\bar{e}_g(k)\| \\ & - \|(\sqrt{P_3}\bar{e}_f(k) + \sqrt{P_4}\bar{e}_g(k)) - (k_v r(k) + g(x)u_d(k) + \epsilon(k) + d(k))\|^2 \\ & - \frac{1}{\alpha} \|I - \alpha\varphi_f(k)\varphi_f^T(k)\|^2 \delta(2-\delta)[\|\tilde{W}_f(k)\| - \frac{(1-\delta)}{(2-\delta)} W_{fmax}^2]^2 \\ & - \frac{1}{\beta} \|I - \beta\varphi_g(k)\varphi_g^T(k)\|^2 \rho(2-\rho)[\|\tilde{W}_g(k)\| - \frac{(1-\rho)}{(2-\rho)} W_{gmax}^2]^2\end{aligned}\quad (8.3.83)$$

where

$$a_2 = (2+\eta)k_{vmax}^2 + 2(1+\eta)C_1k_{vmax} + (2+\eta)C_1^2 + 2k_{vmax}C_1 \quad (8.3.84)$$

$$a_3 = (1+\eta)k_{vmax}(\epsilon_N + d_M + C_0) + P_2k_{vmax} + P_2C_1 + \eta C_1(\epsilon_N + d_M) \quad (8.3.85)$$

$$+ \frac{1}{2}(2+\eta)C_1(\epsilon_N + d_M + C_0) + 2k_{vmax}(\epsilon_N + d_M + C_0) \quad (8.3.86)$$

$$\begin{aligned}a_{44} = & 2P_2(\epsilon_N + d_M + C_0) + 2\eta C_0(\epsilon_N + d_M) \\ & +(2+\eta)(\epsilon_N + d_M + C_0)^2 + 2\eta\epsilon_N d_M\end{aligned}\quad (8.3.87)$$

and

$$\begin{aligned}a_4 = & a_{44} + \frac{1}{\alpha} \|I - \alpha\varphi_f(k)\varphi_f^T(k)\|^2 \frac{\delta^2}{(2-\delta)} W_{fmax}^2 \\ & + \frac{1}{\beta} \|I - \beta\varphi_g(k)\varphi_g^T(k)\|^2 \frac{\rho^2}{(2-\rho)} W_{gmax}^2\end{aligned}\quad (8.3.88)$$

All the terms in (8.3.83) are always negative except the first term as long as the condition (8.3.67) through (8.3.72) hold. Since  $a_2, a_3$  and  $a_4$  are positive constants,  $\Delta J \leq 0$  as long as (8.3.67) through (8.3.72) hold with

$$\|r(k)\| > \delta_{r1} \quad (8.3.89)$$

where

$$\delta_{r1} = \frac{1}{(1-a_2)} [a_3 + \sqrt{a_3^2 + a_4(1-a_2)}]. \quad (8.3.90)$$

Similarly, completing the squares for  $\|r(k)\|, \|\tilde{W}_g(k)\|$  using (8.3.76) yields

$$\begin{aligned}\Delta J = & -(1-a_2)[\|r(k)\| - \frac{a_3}{(1-a_2)}]^2 \\ & -(1-\eta-P_3) \|\bar{e}_f(k)\|^2 \\ & -(1-\eta-P_4) \|\bar{e}_g(k)\|^2 - 2(1-\eta-P_1) \|\bar{e}_f(k)\| \|\bar{e}_g(k)\| \\ & - \|(\sqrt{P_3}\bar{e}_f(k) + \sqrt{P_4}\bar{e}_g(k)) - (k_v r(k) + g(x)u_d(k) + \epsilon(k) + d(k))\|^2 \\ & - \frac{1}{\alpha} \|I - \alpha\varphi_f(k)\varphi_f^T(k)\|^2 \delta(2-\delta)[\|\tilde{W}_f(k)\| - \frac{(1-\delta)}{(2-\delta)} W_{fmax} - a_4]^2 \\ & - \frac{1}{\beta} \|I - \beta\varphi_g(k)\varphi_g^T(k)\|^2 \rho(2-\rho)[\|\tilde{W}_g(k)\| - \frac{(1-\rho)}{(2-\rho)} W_{gmax}]^2\end{aligned}\quad (8.3.91)$$

where

$$\begin{aligned} a_4 &= \delta^2 W_{fmax}^2 + \frac{\alpha}{\|I - \alpha\varphi_f(k)\varphi_f^T(k)\|^2} [a_{44} + \frac{a_3^2}{(1-a_2)} \\ &\quad + \frac{1}{\beta} \|I - \beta\varphi_g(k)\varphi_g^T(k)\|^2 \frac{\rho^2}{(2-\rho)} W_{gmax}^2]. \end{aligned} \quad (8.3.92)$$

Then  $\Delta J \leq 0$  as long as (8.3.67) through (8.3.72) hold and the quadratic term for  $\tilde{W}_f(k)$  in (8.3.91) is positive, which is guaranteed when

$$\|\tilde{W}_f(k)\| > \delta_{f1} \quad (8.3.93)$$

where

$$\delta_{f1} = \frac{1}{(2-\delta)} [(1-\delta) + \sqrt{(1-\delta)^2 + a_4(2-\delta)}]. \quad (8.3.94)$$

Similarly, completing the squares for  $\|r(k)\|, \|\tilde{W}_f(k)\|$  using (8.3.76) yields

$$\begin{aligned} \Delta J &= -(1-a_2)[\|r(k)\| - \frac{a_3}{(1-a_2)}]^2 \\ &\quad - (1-\eta-P_3) \|\bar{e}_f(k)\|^2 - (1-\eta-P_4) \|\bar{e}_g(k)\|^2 \\ &\quad - 2(1-\eta-P_1) \|\bar{e}_f(k)\| \|\bar{e}_g(k)\| \\ &\quad - \|(\sqrt{P_3}\bar{e}_f(k) + \sqrt{P_4}\bar{e}_g(k)) - (k_v r(k) + g(x)u_d(k) + \epsilon(k) + d(k))\|^2 \\ &\quad - \frac{1}{\alpha} \|I - \alpha\varphi_f(k)\varphi_f^T(k)\|^2 \delta(2-\delta) [\|\tilde{W}_f(k)\| - \frac{(1-\delta)}{(2-\delta)} W_{fmax}]^2 \\ &\quad - \frac{1}{\beta} \|I - \beta\varphi_g(k)\varphi_g^T(k)\|^2 \rho(2-\rho) [\|\tilde{W}_g(k)\| - \frac{(1-\rho)}{(2-\rho)} W_{gmax} - a_4] \end{aligned} \quad (8.3.95)$$

where

$$\begin{aligned} a_4 &= \rho^2 W_{gmax}^2 + \frac{\beta}{\|I - \beta\varphi_g(k)\varphi_g^T(k)\|^2} [a_{44} + \frac{a_3^2}{(1-a_2)} + \frac{1}{\alpha} \|I - \alpha\varphi_f(k)\varphi_f^T(k)\|^2 \\ &\quad + \frac{\delta^2}{(2-\delta)} W_{fmax}^2] \end{aligned} \quad (8.3.96)$$

Then  $\Delta J \leq 0$  as long as (8.3.67) through (8.3.72) hold and the quadratic term for  $\tilde{W}_g(k)$  in (8.3.95) is positive, which is guaranteed when

$$\|\tilde{W}_g(k)\| > \delta_g \quad (8.3.97)$$

where

$$\delta_g = \frac{1}{(2-\rho)} [(1-\rho) + \sqrt{(1-\rho)^2 + a_4(2-\rho)}]. \quad (8.3.98)$$

We have shown upper bounds for the tracking error and the NN weight estimation errors for this region for all  $|u_c(k)| \leq s$ .

Region II:  $|\hat{g}(x(k))| < \underline{g}$  and  $|u_c(k)| > s$ .

Select the Lyapunov function candidate (8.3.10) whose first difference is given by (8.3.11). The tracking error system in (8.2.27) can be rewritten as

$$r(k+1) = k_v r(k) + \bar{e}_f^T(k) + \overline{g(x)u_d(k)} + \epsilon_f(k) + d(k) \quad (8.3.99)$$

where

$$\overline{g(x)u_d(k)} = g(x)u(k) - \hat{g}(x)u_c(k) \quad (8.3.100)$$

For the case of modified weight tuning (8.3.74) through (8.3.75) in this region, let us denote the bound given in (8.3.29) as  $d_1$ . The bound for  $\hat{g}u_c(k)$  can be obtained as

$$\begin{aligned} |\hat{g}(x)u_c(k)| &\leq gs, |u_c(k)| \leq s \\ &\leq \frac{g}{\gamma e}, |u_c(k)| > s, \end{aligned} \quad (8.3.101)$$

whose upper bound in either case is denoted by  $C_2$ . Using the individual upper bounds, the upper bound for  $g(x)u_d(k)$  can be obtained as (8.3.49).

Consider the first difference of the Lyapunov function, substitute the bounds for  $g(x)u_d(k)$ , complete the squares and rearrange terms to obtain

$$\begin{aligned} \Delta J &= -(1 - b_0) \|r(k)\|^2 + 2b_1 \|r(k)\| + b_2 \\ &\quad -(1 - \eta) \|\bar{e}_f(k)\| \\ &\quad - \frac{(\eta + \delta \|I - \alpha\varphi_f(k)\varphi_f^T(k)\|)}{(1 - \eta)} (k_v r(k) + \overline{g(x)u_d(k)} + \epsilon_f(k) + d(k)) \| \\ &\quad - \frac{1}{\alpha} \|I - \alpha\varphi_f(k)\varphi_f^T(k)\|^2 [\delta(2 - \delta) \|\tilde{W}_f(k)\|^2 \\ &\quad - 2\delta(1 - \delta) \|\tilde{W}_f(k)\| W_{fmax} - \delta^2 W_{fmax}^2], \end{aligned} \quad (8.3.102)$$

where

$$b_0 = a_0 k_{vmax}^2 + 2a_0 C_4 K_{vmax} + a_0 C_4^2, \quad (8.3.103)$$

$$b_1 = a_0 k_{vmax} (C_3 + \epsilon_{Nf} + d_M) + a_0 (C_3 + \epsilon_{Nf} + d_M) C_4 (C_3 + \epsilon_{Nf} + d_M) \quad (8.3.104)$$

$$\begin{aligned} b_2 &= a_0 (C_3 + \epsilon_{Nf} + d_M)^2 \\ &\quad + 2\delta \|I - \alpha\varphi_f(k)\varphi_f^T(k)\| (C_3 + \epsilon_{Nf} + d_M) W_{fmax} \varphi_{fmax}, \end{aligned} \quad (8.3.105)$$

$$a_0 = \frac{(\alpha\varphi_f(k)\varphi_f^T(k) + 2\delta \|I - \alpha\varphi_f(k)\varphi_f^T(k)\|)^2}{1 + \alpha\varphi_f^T(k)\varphi_f(k)} \quad (8.3.106)$$

and

$$\|\epsilon_f(k)\| \leq \epsilon_{Nf}. \quad (8.3.107)$$

The second term in (8.3.102) is always negative as long as the conditions (8.3.67) through (8.3.72) hold. Completing the squares for  $\|\tilde{W}_f(k)\|$  results in

$$\begin{aligned} \Delta J &= -(1 - b_0) \|r(k)\|^2 + 2b_1 \|r(k)\| + b_3 \\ &\quad -(1 - \eta) \|\bar{e}_f(k) - \frac{(\eta + \delta \|I - \alpha\varphi_f(k)\varphi_f^T(k)\|)}{(1 - \eta)} (k_v r(k) + \overline{g(x)u_d(k)} \\ &\quad + \epsilon_f(k) + d(k))\| \\ &\quad - \frac{1}{\alpha} \|I - \alpha\varphi_f(k)\varphi_f^T(k)\|^2 \delta(2 - \delta) [\|\tilde{W}_f(k)\| \\ &\quad - \frac{(1 - \delta)}{(2 - \delta)} W_{fmax}]^2, \end{aligned} \quad (8.3.108)$$

with

$$b_3 = b_2 + \frac{1}{\alpha} \| I - \alpha \varphi_f(k) \varphi_f^T(k) \|^2 \frac{\delta}{(2-\delta)} W_{fmax}^2. \quad (8.3.109)$$

Since  $b_0, b_1$  and  $b_3$  are positive constants in (8.3.108) and the second and third terms are always negative,  $\Delta J \leq 0$  as long as

$$\| r(k) \| > \delta_{r2} \quad (8.3.110)$$

where

$$\delta_{r2} = \frac{1}{(1-b_0)} [b_1 + \sqrt{b_1^2 + b_3(1-b_0)}]. \quad (8.3.111)$$

Similarly completing the squares for  $\| r(k) \|$  using (8.3.102) yields

$$\begin{aligned} \Delta J = & -(1-b_0)[\| r(k) \| - \frac{b_1}{(1-b_0)}]^2 \\ & -(1-\eta) \| \bar{e}_f(k) - \frac{(\eta+\delta \| I - \alpha \varphi_f(k) \varphi_f^T(k) \|)}{(1-\eta)} (k_v r(k) + \bar{g}(x) u_d(k) \\ & + \epsilon_f(k) + d(k)) \|^2 \\ & - \frac{1}{\alpha} \| I - \alpha \varphi_f(k) \varphi_f^T(k) \|^2 \delta(2-\delta) [\| \tilde{W}_f(k) \| \\ & - 2 \frac{(1-\delta)}{(2-\delta)} \| \tilde{W}_f(k) \| W_{fmax} - b_4], \end{aligned} \quad (8.3.112)$$

with

$$b_4 = \frac{1}{(2-\delta)} \left[ \frac{\alpha}{\| I - \alpha \varphi_f(k) \varphi_f^T(k) \|^2} \frac{b_1^2}{(1-b_0)} + \delta^2 W_{fmax} \right]. \quad (8.3.113)$$

Since  $b_0, b_1$  and  $b_4$  are positive constants in (8.3.112) and the second and third terms are always negative,  $\Delta J \leq 0$  as long as

$$\| \tilde{W}_f(k) \| > \delta_{f2}, \quad (8.3.114)$$

where

$$\delta_{f2} = \frac{1}{(2-\delta)} [(1-\delta) + \sqrt{(1-\delta)^2 + b_4(2-\delta)}]. \quad (8.3.115)$$

$| \sum_{k=k_0}^{\infty} \Delta J(k) | = | J(\infty) - J(0) | < \infty$  since  $\Delta J \leq 0$  as long as (8.3.67) through (8.3.72) hold. The definition of  $J$  and inequalities (8.3.110) and (8.3.114) imply that every intial conditon in the set  $\chi$  will evolve entirely within  $\chi$ . Thus according to the standard Lyapunov extension, it can be concluded that the tracking error  $r(k)$  and the error in weight updates are UUB.

Reprise:

Combining the results from region I and II, one can readily set  $\delta_r = \max(\delta_{r1}, \delta_{r2})$ ,  $\delta_f = \max(\delta_{f1}, \delta_{f2})$ , and  $\delta_g$ .

Thus for both regions, if  $\| r(k) \| > \delta_r$ , then  $\Delta J \leq 0$  and  $u(k)$  is bounded. Let us denote  $(\| r(k) \|, \| \tilde{W}_f(k) \|, \| \tilde{W}_g(k) \|)$  by new coordinate variables  $(\xi_1, \xi_2, \xi_3)$ . Define the region

$$\Xi : \xi | x_i < \delta_r, \xi_2 < \delta_f, \xi_3 < \delta_g,$$

then there exists an open set

$$\Omega : \xi | \xi_1 < \bar{\delta}_r, \xi < \bar{\delta}_f, \xi_3 < \bar{\delta}_g,$$

where  $\bar{\delta}_i > \delta_i$  implies that  $\Xi \subset \Omega$ . In other words, it was shown that whenever  $\xi_i > \delta_i$  then  $V(\xi)$  will not increase and will remain in the region  $\Omega$  which is a invariant set. Therefore all the signals in the closed-loop system remain uniformly ultimately bounded. This concludes the proof.  $\square$

*Remarks:*

1. For practical purposes Equations (8.3.89) or (8.3.110), (8.3.93) or (8.3.114) and (8.3.97) can be considered as bounds for  $r(k)$ ,  $\tilde{W}_f(k)$  and  $\tilde{W}_g(k)$  in both regions.
2. The NN reconstruction errors and the bounded disturbances are all embodied in the constants given by  $\delta_r$ ,  $\delta_f$  and  $\delta_g$ . Note that the bound on the tracking error may be kept small if the closed-loop poles are placed closer to the origin.
3. If the switching parameter  $s$  is chosen small, it will limit the control input and results in a large tracking error which results in undesirable closed-loop performance. Large value of  $s$  results in the saturation of the control input  $u(k)$ .
4. Uniform ultimate boundedness of the closed-loop system is shown without making assumptions on the initial weights. Persistency of excitation condition on the input signals is not required and the certainty equivalence principle is not used. The NN can be easily initialized as  $\tilde{W}_f(k) = 0$ , and  $\tilde{W}_g(k) > g^{-1}(g)$ . In addition, the NN presented here do not need off-line learning phase. No assumptions such as the existence of an invariant set, region of attraction, or a feasible region is needed.

Note that the NN reconstruction error bound  $\epsilon_N$  and the bounded disturbances  $d_M$  increase the bounds on  $\| r(k) \|$  and  $\| \tilde{W}_f(k) \|$  and  $\| W_g(k) \|$  in a very interesting way. Note that small tracking error bounds, but not arbitrarily small, may be achieved by placing the closed-loop poles inside the unit circle and near the origin through the selection of the largest eigenvalue,  $k_{vmax}$ . On the other hand, the NN weight error estimates are fundamentally bounded by  $W_{fmax}$ , and  $W_{gmax}$  the known bound on ideal weights  $W_f$  and  $W_g$ . The parameter  $\delta$  and  $\rho$  offers a design trade-off between the relative eventual magnitudes of  $\| r(k) \|$  and  $\| \tilde{W}_f(k) \|$  and  $\| \tilde{W}_g(k) \|$ ; a smaller  $\delta$  yields a smaller  $\| r(k) \|$  and a larger  $\| \tilde{W}_f(k) \|$ , and vice versa. For the tuning weights  $\| \tilde{W}_g(k) \|$ , similar effects are observed.

The effect of adaptation gains  $\alpha$  and  $\beta$  at each layer on the weight estimation errors,  $\tilde{W}_f(k)$  and  $\tilde{W}_g(k)$ , and tracking error,  $r(k)$ , can be easily observed by using the bounds presented in (8.3.93) or (8.3.114) and (8.3.97). Large values of  $\alpha$  and  $\beta$  force smaller tracking error but larger weight estimation errors.

In contrast, a small value of  $\alpha$  and  $\beta$  force larger tracking and smaller weight estimation errors.

## 8.4 MULTILAYER NEURAL NETWORKS FOR FEEDBACK LINEARIZATION

A family of multilayer NN weight tuning paradigms that guarantee the stability of the closed-loop system (8.2.28) is presented in this section. It is required to demonstrate that the tracking error  $r(k)$  is suitably small and that the NN weights  $\hat{W}_{1f}(k)$ ,  $\hat{W}_{2f}(k)$ ,  $\hat{W}_{3f}(k)$ , and  $\hat{W}_{1g}(k)$ ,  $\hat{W}_{2g}(k)$ ,  $\hat{W}_{3g}(k)$  remain bounded. To proceed

Table 8.4.1: Discrete-Time Controller Using Multilayer Neural Net: PE Required

---

The control input is

$$\begin{aligned} u(k) &= u_c(k) + \frac{u_r(k) - u_c(k)}{2} e^{\gamma(|u_c(k)|-s)}, \quad I = 1, \\ &= u_r(k) - \frac{u_r(k) - u_c(k)}{2} e^{-\gamma(|u_c(k)|-s)}, \quad I = 0. \end{aligned}$$

The NN weight tuning for  $f(x(k))$  is given by

$$\begin{aligned} \hat{W}_{if}(k+1) &= \hat{W}_{if}(k) + \alpha_{if}\varphi_{if}(k)(\hat{y}_{if}(k) + B_{if}k_v r(k))^T, \quad i = 1, 2, \\ \hat{W}_{3f}(k+1) &= \hat{W}_{3f}(k) + \alpha_{3f}\varphi_{3f}(k)r^T(k+1) \end{aligned}$$

and the NN weight tuning for  $g(x(k))$  is provided by

$$\begin{aligned} \hat{W}_{ig}(k+1) &= \hat{W}_{ig}(k) + \beta_{ig}\varphi_{ig}(k)(\hat{y}_{ig}(k) + B_{ig}k_v r(k))^T, \quad i = 1, 2, \\ \hat{W}_{3g}(k+1) &= \hat{W}_{3g}(k) + \beta_{3g}\varphi_{3g}(k)r^T(k+1), \quad I = 1, \\ &= \hat{W}_{3g}(k), \quad I = 0, \end{aligned}$$

with  $\alpha_{if} > 0; i = 1, 2, 3$  and  $\beta_{ig} > 0; i = 1, 2, 3$  denoting constant learning rate parameters or adaptation gains.

---

further, the machinery presented in the Lemma and definition of Chapter 7 is needed.

#### 8.4.1 Weight Updates Requiring Persistence of Excitation

In the following theorem we present a discrete-time weight tuning algorithm given in Table 8.4.1, based on the filtered tracking error. The algorithm guarantee that both the tracking error and the error in the weight estimates are bounded if a PE condition holds. (This PE requirement is relaxed in Theorem 8.4.2).

**Theorem 8.4.1 (Multilayer Discrete-Time NN Controller Requiring PE) :**

Let the desired trajectory  $x_{nd}(k)$  be bounded and the NN functional reconstruction error bound  $\epsilon_{Nf}$  and  $\epsilon_{Ng}$  with the disturbance bound  $d_M$  be known constants. Take the control input for (8.1.1) as (8.2.29) with weight tuning for  $f(x(k))$  provided by

$$\hat{W}_{if}(k+1) = \hat{W}_{if}(k) + \alpha_{if}\hat{\varphi}_{if}(k)(\hat{y}_{if}(k) + B_{if}k_v r(k))^T \quad (8.4.1)$$

$$\hat{W}_{3f}(k+1) = \hat{W}_{3f}(k) + \alpha_{3f}\hat{\varphi}_{3f}(k)r^T(k+1) \quad (8.4.2)$$

and the weight tuning for  $g(x(k))$  is expressed as

$$\hat{W}_{ig}(k+1) = \hat{W}_{ig}(k) + \beta_{ig}\hat{\varphi}_{ig}(k)(\hat{y}_{ig}(k) + B_{ig}k_v r(k))^T \quad (8.4.3)$$

$$\hat{W}_{3g}(k+1) = \hat{W}_{3g}(k) + \beta_{3g}\hat{\varphi}_{3g}(k)r^T(k+1) \quad (8.4.4)$$

where

$$\hat{y}_{if}(k) = \hat{W}_{if}^T(k)\hat{\varphi}_f(k), i = 1, 2, \quad (8.4.5)$$

$$\hat{y}_{ig}(k) = \hat{W}_{ig}^T(k)\hat{\varphi}_g(k), i = 1, 2, \quad (8.4.6)$$

$$\|B_{if}\| \leq \kappa_{if}; i = 1, 2$$

and

$$\|B_{ig}\| \leq \kappa_{ig}; i = 1, 2 \quad (8.4.7)$$

with  $\alpha_{if} > 0; \forall i = 1, 2, 3$  and  $\beta_{ig} > 0; \forall i = 1, 2, 3$  denoting constant learning rate parameters or adaptation gains.

Assume that the initial error in weight estimates for both NN are bounded and let the hidden-layer output vector,  $\hat{\varphi}_{1f}(k), \hat{\varphi}_{2f}(k), \hat{\varphi}_{3f}(k)$  and  $\bar{\hat{\varphi}}_{3g}(k)u_c(k), \hat{\varphi}_{1g}(k), \hat{\varphi}_{2g}(k)$  be persistently exciting. Then the filtered tracking error  $r(k)$  and the error in weight estimates,  $\tilde{W}_{1f}(k), \tilde{W}_{2f}(k), \tilde{W}_{3f}(k)$ , and  $\tilde{W}_{1g}(k), \tilde{W}_{2g}(k), \tilde{W}_{3g}(k)$  are UUB, with the bounds specifically given by (8.4.34) or (8.4.46) with (8.4.24) and (8.4.26) or (8.4.48), (8.4.25) with (8.4.26) and (8.4.27) provided the following conditions hold:

$$(1) \quad \alpha_{if} \|\hat{\varphi}_{if}(k)\|^2 < 2, \quad i = 1, 2, \quad (8.4.8)$$

$$(2) \quad \alpha_{ig} \|\hat{\varphi}_{ig}(k)\|^2 < 2, \quad i = 1, 2, \quad (8.4.9)$$

$$(3) \quad \beta_{3g} \|\varphi_{3g}(k)u_c(k)\|^2 = \beta_{3g} \|\hat{\varphi}_{3g}(k)\|^2 < 1, \quad (8.4.10)$$

$$(4) \quad \alpha_{3f} \|\hat{\varphi}_{3f}(k)\| < 1, \quad (8.4.11)$$

$$(5) \quad \eta < 1, \quad (8.4.12)$$

$$(6) \quad \max(a_{10}, b_6) < 1, \quad (8.4.13)$$

where  $\eta$  is given as

$$\eta = \alpha_{3f} \|\hat{\varphi}_{3f}(k)\|^2 + \beta_{3g} \|\bar{\hat{\varphi}}_{3g}(k)u_c(k)\|^2 \quad (8.4.14)$$

for  $I = 1$ , and for  $I = 0$ , the parameter  $\eta$  is defined as

$$\eta = \alpha_{3f} \|\hat{\varphi}_{3f}(k)\|^2 \quad (8.4.15)$$

with  $a_{10}$  and  $b_6$  design parameters chosen using the gain matrix  $k_{vmax}$  and the relationship presented during the proof.

Note: The design parameters  $\alpha_{if}, \beta_{ig}; \forall i = 1, 2, 3$  and  $\eta$  depend upon the trajectory.

Proof:

(Note, in the proof  $g(x(k))$  is also referred to as  $g(x)$ .) Define the Lyapunov function candidate

$$J = r^T(k)r(k) + \sum_{i=1}^{i=3} [\frac{1}{\alpha_{if}} \text{tr}(\tilde{W}_{if}^T(k)\tilde{W}_{if}(k)) + \frac{1}{\beta_{ig}} \text{tr}(\tilde{W}_{ig}^T(k)\tilde{W}_{ig}(k))]. \quad (8.4.16)$$

The first difference is given by

$$\begin{aligned} \Delta J &= r^T(k+1)r(k+1) - r^T(k)r(k) + \sum_{i=1}^{i=3} (\frac{1}{\alpha_{if}} \text{tr}(\tilde{W}_{if}^T(k+1)\tilde{W}_{if}(k+1) \\ &\quad - \tilde{W}_{if}^T(k)\tilde{W}_{if}(k)) \\ &\quad - \frac{1}{\beta_{ig}} \text{tr}(\tilde{W}_{ig}^T(k+1)\tilde{W}_{ig}(k+1) - \tilde{W}_{ig}^T(k)\tilde{W}_{ig}(k))). \end{aligned} \quad (8.4.17)$$

Region I:  $|\hat{g}| \geq \underline{g}$  and  $|u_c| \leq s$ .

The filtered error dynamics (8.2.28) can be rewritten as

$$\begin{aligned} r(k+1) &= k_v r(k) + (f(x(k)) - \hat{f}(x(k))) + (g(x(k)) - \hat{g}(x(k))) u_c(k) \\ &\quad + d(k) + g(x(k)) u_d(k) \end{aligned} \quad (8.4.18)$$

where  $u_d(k) = u(k) - u_c(k)$ . Substituting (8.2.10) and (8.2.11) in (8.4.18), one obtains

$$\begin{aligned} r(k+1) &= k_v r(k) + \tilde{W}_{3f}^T(k) \hat{\varphi}_{3f}(k) + \tilde{W}_{3g}^T(k) \bar{\hat{\varphi}}_{3g}(k) u_c(k) + \epsilon(k) \\ &\quad + d(k) + g(x(k)) u_d(k), \end{aligned} \quad (8.4.19)$$

where

$$\epsilon(k) = W_{3f}^T \tilde{\varphi}_{3f}(k) + \epsilon_f(k) + W_{3g}^T \tilde{\varphi}_{3g} u_c(k) + \epsilon_g(k) u_c(k). \quad (8.4.20)$$

Equation (8.4.18) can be rewritten as

$$r(k+1) = k_v r(k) + \bar{e}_f^T(k) + \bar{e}_g^T(k) + \epsilon(k) + d(k) + g(x(k)) u_d(k) \quad (8.4.21)$$

where

$$\bar{e}_f(k) = \tilde{W}_{3f}^T(k) \hat{\varphi}_{3f}(k), \quad (8.4.22)$$

$$\bar{e}_g(k) = \tilde{W}_{3g}^T(k) \bar{\hat{\varphi}}_{3g}(k) u_c(k). \quad (8.4.23)$$

The error in dynamics for the weight update laws are given for this region as

$$\begin{aligned} \tilde{W}_{if}(k+1) &= (I - \alpha_{if} \hat{\varphi}_{if}(k) \hat{\varphi}_{if}^T(k)) \tilde{W}_{if}(k) \\ &\quad - \alpha_{if} \hat{\varphi}_{if}(k) (y_{if} + B_{if} k_v r(k))^T; i = 1, 2 \end{aligned} \quad (8.4.24)$$

$$\begin{aligned} \tilde{W}_{ig}(k+1) &= (I - \beta_{ig} \hat{\varphi}_{ig}(k) \hat{\varphi}_{ig}^T(k)) \tilde{W}_{ig}(k) \\ &\quad - \beta_{ig} \hat{\varphi}_{ig}(k) (y_{ig} + B_{ig} k_v r(k))^T; i = 1, 2 \end{aligned} \quad (8.4.25)$$

$$\begin{aligned} \tilde{W}_{3f}(k+1) &= (I - \alpha_{3f} \hat{\varphi}_{3f}(k) \hat{\varphi}_{3f}^T(k)) \tilde{W}_{3f}(k) - \alpha_{3f} \hat{\varphi}_{3f}(k) \\ &\quad (k_v r(k) + \bar{e}_g(k) + g(x(k)) u_d(k) + \epsilon(k) + d(k))^T \end{aligned} \quad (8.4.26)$$

and

$$\begin{aligned} \tilde{W}_{3g}(k+1) &= (I - \beta_{3g} \hat{\varphi}_{3g}(k) \hat{\varphi}_{3g}^T(k)) \tilde{W}_{3g}(k) - \beta_{3g} \hat{\varphi}_{3g}(k) (k_v r(k) + \bar{e}_f(k) \\ &\quad + g(x(k)) u_d(k) + \epsilon(k) + d(k))^T. \end{aligned} \quad (8.4.27)$$

Substituting (8.4.21) and (8.4.24) through (8.4.27) in (8.4.17), and simplifying one obtains

$$\begin{aligned} \Delta J &= -(1 - a_4) \| r(k) \|^2 + 2a_5 \| r(k) \| + a_6 \\ &\quad - (1 - \eta) \| (\bar{e}_f(k) + \bar{e}_g(k)) - \frac{\eta}{(1 - \eta)} (k_v r(k) + g(x) u_d(k) + \epsilon(k) + d(k)) \|^2, \\ &\quad - \sum_{i=1}^{i=2} (2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k)) \| \hat{y}_{if}(k) - \frac{(1 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k))}{(2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k))} (y_{if} + B_{if} k_v r(k)) \|^2 \\ &\quad - \sum_{i=1}^{i=2} (2 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k)) \| \hat{y}_{ig}(k) - \frac{(1 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k))}{(2 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k))} (y_{ig} + B_{ig} k_v r(k)) \|^2 \\ &\quad + a_7 \| r(k) \|^2 + a_8 \| r(k) \| + a_9 \end{aligned} \quad (8.4.28)$$

where

$$a_7 = \sum_{i=1}^{i=2} \left[ \frac{1}{(2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k))} k_{vmax}^2 \kappa_{if}^2 + \frac{1}{(2 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k))} k_{vmax}^2 \kappa_{ig}^2 \right] \quad (8.4.29)$$

$$\begin{aligned} a_8 &= \sum_{i=1}^{i=2} \left[ \frac{1}{(2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k))} k_{vmax} W_{ifmax} \varphi_{ifmax} \kappa_{if} + \right. \\ &\quad \left. \frac{1}{(2 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k))} k_{vmax} W_{igmax} \varphi_{igmax} \kappa_{ig} \right] \end{aligned} \quad (8.4.30)$$

$$\begin{aligned} a_9 &= \sum_{i=1}^{i=2} \left[ \frac{1}{(2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k))} \varphi_{ifmax}^2 W_{ifmax}^2 \right. \\ &\quad \left. + \frac{1}{(2 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k))} \varphi_{igmax}^2 W_{igmax}^2 \right] \end{aligned} \quad (8.4.31)$$

with  $a_1, a_2, a_3, a_4, a_5, a_6$  presented in (8.3.23), (8.3.24), (8.3.25), (8.3.33), (8.3.34), and (8.3.35) respectively. The parameter  $\eta$  is given by

$$\eta = \alpha_{3f} \| \hat{\varphi}_{3f}(x(k)) \|^2 + \beta_{3g} \| \hat{\varphi}_{3g}(x(k)) \|^2 \quad (8.4.32)$$

and

$$\epsilon_N = 2W_{3fmax} \varphi_{3fmax} + \epsilon_{fmax} + s(2W_{3gmax} \varphi_{3gmax} + \epsilon_{gmax}). \quad (8.4.33)$$

The terms in the Equation (8.4.28) are always negative as long as the conditions (8.4.8) through (8.4.13) hold. Since  $a_4, a_5$  and  $a_6$  are positive constants,  $\Delta J \leq 0$  as long as (8.4.8) through (8.4.13) hold and

$$\| r(k) \| > \delta_{r1} \quad (8.4.34)$$

where

$$\delta_{r1} > \frac{1}{(1 - a_{10})} [a_{11} + \sqrt{a_{11}^2 + a_{12}(1 - a_{10})}] \quad (8.4.35)$$

with  $a_{10} = a_4 + a_7, a_{11} = a_5 + a_8, a_{12} = a_6 + a_8$ . Then,  $| \sum_{k=k_0}^{\infty} \Delta J(k) | = | J(\infty) - J(0) | < \infty$  since  $\Delta J \leq 0$  as long as (8.4.8) through (8.4.13) hold. The definition of  $J$  and inequality (8.4.34) imply that every initial condition in the set  $\chi$  will evolve entirely within  $\chi$ . In other words, whenever the tracking error  $\| r(k) \|$  is outside the region defined by (8.4.34),  $J(r(k), \tilde{W}_{if}(k), \tilde{W}_{ig}(k); \forall i = 1, 2, 3)$  will decrease. This further implies that the tracking error  $r(k)$  is UUB for all  $k \geq 0$  and it remains to show that the weight estimation errors,  $\tilde{W}_{if}(k); \forall i = 1, 2, 3$  and  $\tilde{W}_{ig}(k); \forall i = 1, 2, 3$  or equivalently  $\hat{W}_{if}(k); \forall i = 1, 2, 3$  and  $\hat{W}_{ig}(k); \forall i = 1, 2, 3$  are bounded.

Generally in order to show the boundedness of the weight estimation errors, one uses the error in weight updates (8.4.24) through (8.4.27), tracking error bound (8.4.34), the PE condition and Lemma presented in Chapter 7. Using (8.4.24) and (8.4.25), applying the PE condition, the tracking error bound, and Lemma in Chapter 7, the boundedness of the error in weight estimates of the hidden layers for both NN  $\tilde{W}_{if}(k), \tilde{W}_{ig}(k); i = 1, 2$  or equivalently, weight estimates of the hidden layers,  $\hat{W}_{if}(k), \hat{W}_{ig}(k); i = 1, 2$ .

On the other hand, using (8.4.26) and (8.4.27) it can be realized that the output of each neural network is driving the other. Therefore, the boundedness of the tracking error,

the PE condition and Lemma in Chapter 7 are necessary but not sufficient to prove the error in weight estimates of the output layers for both NN  $\tilde{W}_{if}(k), \tilde{W}_{ig}(k); i = 3$ .

If the initial weight estimation errors for both NN are considered to be bounded, then applying the bound for the tracking error (8.4.34), the PE condition and Lemma in Chapter 7, one can show that the weight estimation errors  $\tilde{W}_{3f}(k)$  and  $\tilde{W}_{3g}(k)$  or equivalently  $\hat{W}_{3f}(k)$  and  $\hat{W}_{3g}(k)$  are bounded. This concludes the boundedness of both tracking error and weight estimates for both NN in this region. On the other hand, a similar and elegant way to show the boundedness of tracking error and weight estimates is to apply passivity theory. The proof using passivity theory is shown in Section 8.5.

Region II:  $|\hat{g}(x)| \leq g$  and  $|u_c(k)| > s$ .

The filtered tracking error dynamics can be written as

$$r(k+1) = k_v r(k) + \bar{e}_f(k) + \overline{g(x)u_d(k)} + \epsilon(k) + d(k) \quad (8.4.36)$$

where

$$\epsilon(k) = W_{3f}^T(k)\hat{\varphi}_{3f}(k) + \epsilon_f(k). \quad (8.4.37)$$

Now using the Lyapunov function (8.4.16), the first difference (8.4.17), after substituting for  $\overline{g(x)u_d(k)}$  from (8.3.49) in (8.4.17) and manipulating accordingly, one can obtain

$$\begin{aligned} \Delta J = & -(1 - b_0) \| r(k) \|^2 + 2b_1 \| r(k) \| + b_2 \\ & - (1 - \eta) \| \bar{e}_f(k) - \frac{\eta}{(1 - \eta)} (k_v r(k) + \overline{g(x)u_d(k)} + \epsilon(k) + d(k)) \|^2, \\ & - \sum_{i=1}^{i=2} (2 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k)) \| \hat{y}_{if}(k) - \frac{(1 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))}{(1 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))} (y_{if} + B_{if}k_v r(k)) \|^2 \\ & + b_3 \| r(k) \|^2 + b_4 \| r(k) \| + b_5 \end{aligned} \quad (8.4.38)$$

where

$$b_0 = k_{vmax}^2 + 2C_4(C_4 + k_{vmax}) + \frac{(C_4 + k_{vmax})^2}{(1 - \alpha_{3f} \| \varphi_{3f}(k) \ |)^2}, \quad (8.4.39)$$

$$\begin{aligned} b_1 = & C_3(C_4 + k_{vmax}) + C_3C_4 + (C_4 + k_{vmax})(\epsilon_N + d_M) + \frac{C_3(C_4 + k_{vmax})}{(1 - \alpha_{3f} \| \varphi_{3f}(k) \ |)^2} + \\ & \frac{(C_4 + k_{vmax})(\epsilon_N + d_M)}{(1 - \alpha_{3f} \| \varphi_{3f}(k) \ |)^2} \end{aligned} \quad (8.4.40)$$

$$\begin{aligned} b_2 = & 2C_3^2 + 2C_3(\epsilon_N + d_M) + (\epsilon_N + d_M)^2 \\ & + \frac{C_3^2 + 2C_3(\epsilon_N + d_M) + (\epsilon_N + d_M)^2}{(1 - \alpha_{3f} \| \varphi_{3f}(k) \ |)^2} \end{aligned} \quad (8.4.41)$$

$$\| \epsilon(k) \| \leq \epsilon_N \quad (8.4.42)$$

$$b_3 = \sum_{i=1}^{i=2} \frac{1}{(2 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))} k_{vmax}^2 \kappa_{if}^2 \quad (8.4.43)$$

$$b_4 = \sum_{i=1}^{i=2} \frac{1}{(2 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))} (k_{vmax} W_{ifmax} \varphi_{ifmax} \kappa_{if}) \quad (8.4.44)$$

$$b_5 = \sum_{i=1}^{i=2} \frac{1}{(2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k))} (\varphi_{if}^2 W_{if}^2) \quad (8.4.45)$$

with  $\eta$  given by (8.4.32).

The terms in (8.4.38) are always negative as long as the conditions (8.4.8) through (8.4.13) hold. Since  $b_0, b_1$  and  $b_2$  are positive constants,  $\Delta J \leq 0$  as long as

$$\| r(k) \| > \delta_{r2}, \quad (8.4.46)$$

with

$$\delta_{r2} = \frac{1}{(1 - b_6)} [b_7 + \sqrt{b_7^2 + b_8(1 - b_6)}]. \quad (8.4.47)$$

with  $b_6 = b_0 + b_3, b_7 = b_1 + b_4, b_8 = b_2 + b_5$ . Then  $|\sum_{k=k_0}^{\infty} \Delta J(k)| = |J(\infty) - J(0)| < \infty$  since  $\Delta J \leq 0$  as long as (8.4.8) through (8.4.13) hold. The definition of  $J$  and inequality (8.4.46) imply that every initial condition in the set  $\chi$  will evolve entirely within  $\chi$ . In other words, whenever the tracking error  $\| r(k) \|$  is outside the region defined by (8.4.46),  $J(r(k), \tilde{W}_{if}(k), \tilde{W}_{ig}(k); \forall i = 1, 2, 3)$  will decrease. This further implies that the tracking error  $r(k)$  is UUB for all  $k \geq 0$  and it remains to show that the weight estimation errors,  $\tilde{W}_{1f}(k), \tilde{W}_{2f}(k), \tilde{W}_{3ff}(k)$  or equivalently  $\hat{W}_{1f}(k), \hat{W}_{2f}(k), \hat{W}_{3f}(k)$  are bounded.

In order to show the boundedness of the weight estimation errors, one uses the error in weight updates (8.4.24) and (8.4.26) for  $f(\cdot)$ , the tracking error bound (8.4.46), the PE condition and Lemma in Chapter 7. Since the weight estimates for  $\hat{g}(x)$  is not updated in this region, the boundedness of the weight estimates for  $\hat{g}(x)$  need not to be shown. However, to show the boundedness of the weight estimates for  $\hat{f}(x)$ , the dynamics relative to the error in weight estimates using (8.4.24) and (8.4.26) for this region are given by

$$\begin{aligned} \tilde{W}_{if}(k+1) &= (I - \alpha_{if} \hat{\varphi}_{if}(k) \hat{\varphi}_{if}(k)^T) \tilde{W}_{if}(k) \\ &\quad - \alpha_{if} \hat{\varphi}_{if}(k) (y_{if} + B_{if} k_v r(k))^T, \quad i = 1, 2, \end{aligned} \quad (8.4.48)$$

$$\begin{aligned} \tilde{W}_{3f}(k+1) &= (I - \alpha_{3f} \hat{\varphi}_{3f}(k) \hat{\varphi}_{3f}(k)^T) \tilde{W}_{3f}(k) - \alpha_{3f} \hat{\varphi}_{3f}(k) (k_v r(k) \\ &\quad + C_3 + C_4 \| r(k) \| + \epsilon(k) + d(k))^T, \end{aligned} \quad (8.4.49)$$

where the tracking error  $r(k)$  is shown to be bounded. Applying the PE condition and Lemma described in Chapter 7, the boundedness of weight estimation errors  $\tilde{W}_{1f}(k), \tilde{W}_{2f}(k), \tilde{W}_{3f}(k)$  or equivalently  $\hat{W}_{1f}(k), \hat{W}_{2f}(k), \hat{W}_{3f}(k)$  are guaranteed. Let us denote the bound by  $\delta_{f2}$ . This concludes the boundedness of both tracking error and weight estimates for both NN in this region.

Reprise:

Combining the results from region I and II, one can readily set  $\delta_r = \max(\delta_{r1}, \delta_{r2}), \delta_f = \max(\delta_{f1}, \delta_{f2})$ , and  $\delta_g$ . Thus for both regions, if  $\| r(k) \| > \delta_r$ , then  $\Delta J \leq 0$  and  $u(k)$  is bounded. Let us denote  $(\| r(k) \|, \| \tilde{W}_{if}(k) \|, \| \tilde{W}_{ig}(k) \|)$  by new coordinate variables  $(\xi_1, \xi_2, \xi_3)$ . Define the region

$$\Xi : \xi | \xi_1 < \delta_r, \xi_2 < \delta_f, \xi_3 < \delta_g,$$

then there exists an open set

$$\Omega : \xi | \xi_1 < \bar{\delta}_r, \xi_2 < \bar{\delta}_f, \xi_3 < \bar{\delta}_g,$$

where  $\bar{\delta}_i > \delta_i$  implies that  $\Xi \subset \Omega$ . In other words, we have proved that whenever  $\xi_i > \delta_i$ , then  $J(\xi)$  will not increase and will remain in the region  $\Omega$  which is an invariant set. Therefore all the signals in the closed-loop system remain bounded. This concludes the proof.  $\square$

**Example 8.4.1 (NN Control of Continuous-Time Nonlinear System) :**

To illustrate the performance of the NN controller, a continuous-time nonlinear system is considered and the objective is to control this feedback linearizable MIMO system by using a three-layer NN controller. Note that it is extremely difficult to discretize a nonlinear system and therefore offer stability proofs. Note that the NN controllers derived herein require no *a priori* knowledge of the dynamics of the nonlinear systems, unlike conventional adaptive control nor is any initial learning phase needed.

Consider the nonlinear system described by

$$\begin{aligned}\dot{X}_1 &= X_2 \\ \dot{X}_2 &= F(X_1, X_2) + G_1(X_1, X_2)U\end{aligned}\quad (8.4.50)$$

where  $X_1 = [x_1, x_2]^T$ ,  $X_2 = [x_3, x_4]^T$ , and the input vector is given by  $U = [u_1, u_2]^T$  and the nonlinear function in (8.4.50) is described by  $F(X_1, X_2) = [M(X_1)]^{-1}G(X_1, X_2)$ , with

$$M(X_1) = \begin{bmatrix} (b_1 + b_2)a_1^2 + b_2a_2^2 + 2b_2a_1a_2\cos(x_2) & b_2a_2^2 + b_2a_1a_2\cos(x_2) \\ b_2a_2^2 + b_2a_1a_2\cos(x_2) & b_2a_2^2 \end{bmatrix} \quad (8.4.51)$$

$$G(X_1, X_2) = \begin{bmatrix} -b_2a_1a_2(2x_3x_4 + x_4^2)\sin(x_2) + 9.8(b_1 + b_2)a_1\cos(x_1) + 9.8b_2a_2\cos(x_1 + x_2) \\ b_2a_1a_2x_1^2\sin(x_2) + 9.8b_2a_2\cos(x_1 + x_2) \end{bmatrix} \quad (8.4.52)$$

and

$$G_1(X_1, X_2) = M^{-1}(X_1, X_2). \quad (8.4.53)$$

The parameters for the nonlinear system were selected as  $a_1 = a_2 = 1, b_1 = b_2 = 1$ . Desired sinusoidal,  $\sin(\frac{2\pi t}{25})$ , and cosine inputs,  $\cos(\frac{2\pi t}{25})$ , were preselected for the axes 1 and 2 respectively. The continuous-time gains of the PD controller were chosen as  $k_v = \text{diag}(20, 20)$  with  $\Lambda = \text{diag}(5, 5)$  and a sampling interval of 10 ms was considered. Three-layer NN were selected with 10 hidden-layer nodes. Sigmoidal activation functions were employed in all the nodes in the hidden layer. The initial conditions for  $X_1$  were chosen to be  $[0.5, 0.1]^T$ , and the weights for  $F(\cdot)$  were initialized to zero whereas the weights for  $G(\cdot)$  were initialized to identity matrix. No off-line learning is performed initially to train the networks. Fig. 8.4.1 presents the tracking response of the neural network controller with delta-rule weight tuning (8.4.1) through (8.4.4), with  $\alpha_3 = 0.1, \alpha_i = 1.0; \forall i = 1, 2$  and  $\beta_3 = 0.1, \beta_i = 1.0; \forall i = 1, 2$ . From the figure, it can be seen that the delta rule-based weight tuning performs impressively.  $\square$

**8.4.2 Weight Updates not Requiring Persistence of Excitation**

*Approaches such as  $\sigma$ -modification (Polycarpou and Ioannou 1991) or  $\epsilon$ -modification (Narendra and Annaswamy 1987) are available for the robust adaptive control of continuous systems wherein the persistency of excitation condition is not needed. On the other hand, modification to the standard weight tuning mechanisms in discrete-time to avoid the necessity of PE is also investigated in Jagannathan and Lewis (1996a) using multilayered NN to a specific class of nonlinear systems.*

*In Jagannathan (1996b) an approach similar to  $\epsilon$ -modification was derived for discrete-time NN for feedback linearization. The following theorem from that paper shows the tuning algorithms that do not require persistence of excitation. The controller derived is detailed in Table 8.4.2.*

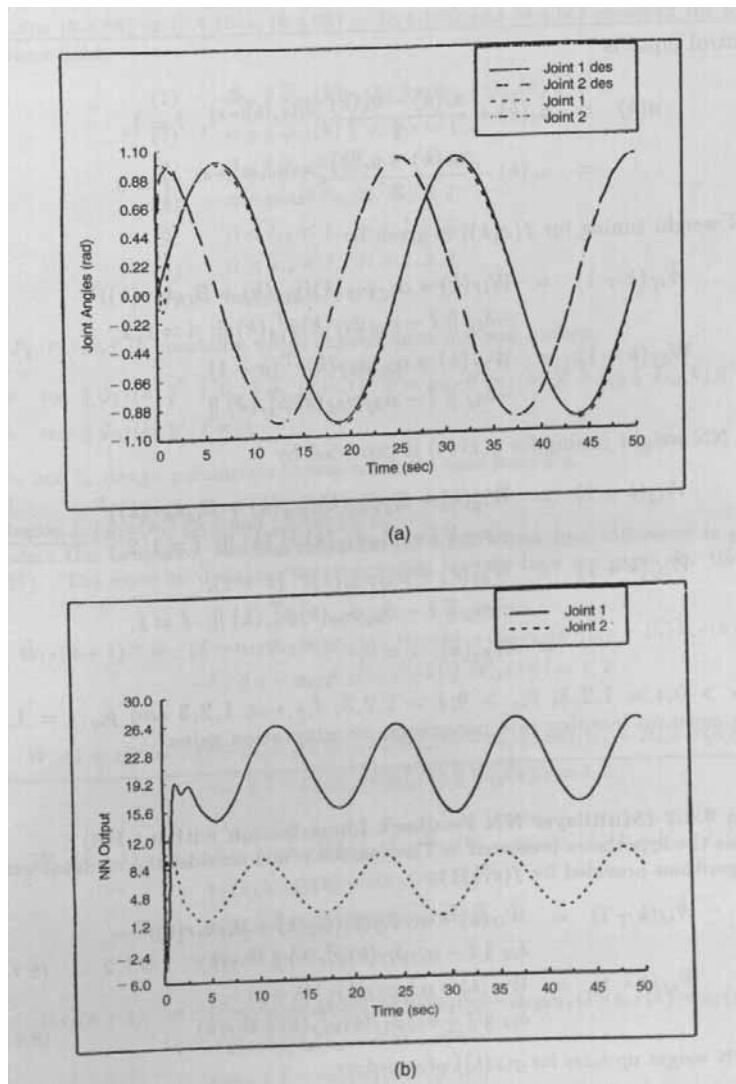


Figure 8.4.1: Response of NN controller with delta-rule based weight tuning. (a) Actual and desired joint angles. (b) Neural network outputs

---

Table 8.4.2: Discrete-Time Controller Using Multilayer Neural Net: PE not Required

---

*The control input is*

$$\begin{aligned} u(k) &= u_c(k) + \frac{u_r(k) - u_c(k)}{2} e^{\gamma(|u_c(k)|-s)}, \quad I = 1, \\ &= u_r(k) - \frac{u_r(k) - u_c(k)}{2} e^{-\gamma(|u_c(k)|-s)}, \quad I = 0. \end{aligned}$$

*The NN weight tuning for  $f(x(k))$  is given by*

$$\begin{aligned} \hat{W}_{if}(k+1) &= \hat{W}_{if}(k) + \alpha_{if}\hat{\varphi}_{if}(k)(\hat{y}_{if}(k) + B_{if}k_v r(k))^T \\ &\quad - \delta_{if} \| I - \alpha_{if}\hat{\varphi}_{if}(k)\hat{\varphi}_{if}^T(k) \|; \quad i = 1, 2, \\ \hat{W}_{3f}(k+1) &= \hat{W}_{3f}(k) + \alpha_{3f}\hat{\varphi}_{3f}(k)r^T(k+1) \\ &\quad - \delta_{3f} \| I - \alpha_{3f}\hat{\varphi}_{3f}(k)\hat{\varphi}_{3f}^T(k) \| \end{aligned}$$

*and the NN weight tuning for  $g(x(k))$  is provided by*

$$\begin{aligned} \hat{W}_{ig}(k+1) &= \hat{W}_{ig}(k) + \beta_{ig}\hat{\varphi}_{ig}(k)(\hat{y}_{ig}(k) + B_{ig}k_v r(k))^T \\ &\quad - \rho_{ig} \| I - \beta_{ig}\hat{\varphi}_{ig}(k)\hat{\varphi}_{ig}^T(k) \|; \quad i = 1, 2, \\ \hat{W}_{3g}(k+1) &= \hat{W}_{3g}(k) + \beta_{3g}\hat{\varphi}_{3g}(k)r^T(k+1), \\ &\quad - \rho_{3g} \| I - \beta_{3g}\hat{\varphi}_{3g}(k)\hat{\varphi}_{3g}^T(k) \|; \quad I = 1, \\ &= \hat{W}_{3g}(k), \quad I = 0, \end{aligned}$$

*with  $\alpha_{if} > 0; i = 1, 2, 3$ ,  $\beta_{ig} > 0; i = 1, 2, 3$ ,  $\delta_{if}; i = 1, 2, 3$  and  $\rho_{ig}; i = 1, 2, 3$  denoting constant learning rate parameters or adaptation gains.*

---

**Theorem 8.4.2 (Multilayer NN Feedback Linearization without PE) :**

Assume the hypotheses presented in Theorem 8.4.1, and consider the modified weight tuning algorithms provided for  $f(x(k))$  by :

$$\begin{aligned} \hat{W}_{if}(k+1) &= \hat{W}_{if}(k) + \alpha_{if}\hat{\varphi}_{if}(k)(\hat{y}_{if}(k) + B_{if}k_v r(k))^T - \\ &\quad \delta_{if} \| I - \alpha_{if}\hat{\varphi}_{if}(k)\hat{\varphi}_{if}^T(k) \| \hat{W}_{if}(k); \quad i = 1, 2 \end{aligned} \quad (8.4.54)$$

$$\begin{aligned} \hat{W}_{3f}(k+1) &= \hat{W}_{3f}(k) + \alpha_{3f}\hat{\varphi}_{3f}(k)r^T(k+1) - \\ &\quad \delta_{3f} \| I - \alpha_{3f}\hat{\varphi}_{3f}(k)\hat{\varphi}_{3f}^T(k) \| \hat{W}_f(k) \end{aligned} \quad (8.4.55)$$

and the NN weight updates for  $g(x(k))$  provided by

$$\begin{aligned} \hat{W}_{ig}(k+1) &= \hat{W}_{ig}(k) + \alpha_{ig}\hat{\varphi}_{ig}(k)(\hat{y}_{ig}(k) + B_{ig}k_v r(k))^T - \\ &\quad \rho_{ig} \| I - \alpha_{ig}\hat{\varphi}_{ig}(k)\hat{\varphi}_{ig}^T(k) \| \hat{W}_{ig}(k); \quad i = 1, 2 \end{aligned} \quad (8.4.56)$$

$$\begin{aligned} \hat{W}_{3g}(k+1) &= \hat{W}_{3g}(k) + \beta_{3g}\hat{\varphi}_{3g}(k)r^T(k+1) \\ &\quad - \rho_{3g} \| I - \beta_{3g}\hat{\varphi}_{3g}(k)\hat{\varphi}_{3g}^T(k) \| \hat{W}_{3g}(k), \quad I = 1, \\ &= \hat{W}_{3g}(k), \quad I = 0, \end{aligned} \quad (8.4.57)$$

with  $\alpha_{ig} > 0; i = 1, 2, 3$ ,  $\beta_{ig} > 0; i = 1, 2, 3$ ,  $\delta_{ig} > 0; i = 1, 2, 3$  and  $\rho_{ig} > 0; i = 1, 2, 3$  design parameters. Then the filtered tracking error  $r(k)$  and the NN weight estimates  $\hat{W}_{if}(k); i = 1, 2, 3$  and  $\hat{W}_{ig}(k); i = 1, 2, 3$  are UUB, with the bounds specifically given by Equations (8.4.86) or (8.4.101), (8.4.92) or (8.4.105) and (8.4.96) provided the following conditions hold:

$$(1) \quad \beta_{3g} \parallel \bar{\varphi}_{3g}(k)u_c(k) \parallel = \beta_{3g} \parallel \hat{\varphi}_{3g}(k) \parallel^2 < 1, \quad (8.4.58)$$

$$(2) \quad \alpha_{if} \parallel \hat{\varphi}_{if}(k) \parallel^2 < 2; i = 1, 2, \quad (8.4.59)$$

$$(3) \quad \beta_{ig} \parallel \hat{\varphi}_{ig}(k) \parallel^2 < 2; i = 1, 2, \quad (8.4.60)$$

$$(4) \quad \eta + \max(P_1, P_3, P_4) < 1 \quad (8.4.61)$$

$$(5) \quad 0 < \delta_{ig} < 1, \forall i = 1, 2, 3, \quad (8.4.62)$$

$$(6) \quad 0 < \rho_{ig} < 1, \forall i = 1, 2, 3, \quad (8.4.63)$$

$$(7) \quad \max(a_5, b_6) < 1, \quad (8.4.64)$$

with  $P_1, P_3$ , and  $P_4$  constants which depend upon  $\eta, \delta$  and  $\rho$  where

$$\begin{aligned} \eta &= \alpha_{3f} \parallel \hat{\varphi}_{3f}(k) \parallel^2 + \beta_{3g} \parallel \bar{\varphi}_{3g}(k)u_c(k) \parallel^2 = \alpha_{3f} \parallel \hat{\varphi}_{3f}(k) \parallel^2 + \beta_{3g} \parallel \hat{\varphi}_{3g}(k) \parallel^2, I = 1, \\ &= \alpha_{3f} \parallel \hat{\varphi}_{3f}(k) \parallel^2, I = 0, \end{aligned} \quad (8.4.65)$$

and  $a_5$  and  $b_6$  design parameters chosen using the gain matrix  $k_v$ .

Proof:

Region I:  $|\hat{g}(x(k))| \geq g$  and  $|u_c(k)| \leq s$ .

Select the Lyapunov function candidate (8.4.16) whose first difference is given by (8.4.17). The error in dynamics for the weight update laws are given for this region as

$$\begin{aligned} \tilde{W}_{if}(k+1) &= (I - \alpha_{if}\hat{\varphi}_{if}(k)\hat{\varphi}_{if}^T(k))\tilde{W}_{if}(k) - \alpha_{if}\hat{\varphi}_{if}(k)(y_{if} + B_{if}k_v r(k))^T \\ &\quad - \delta_{if} \parallel I - \alpha_{if}\hat{\varphi}_{if}(k)\hat{\varphi}_{if}^T(k) \parallel \hat{W}_{if}(k); i = 1, 2 \end{aligned} \quad (8.4.66)$$

$$\begin{aligned} \tilde{W}_{ig}(k+1) &= (I - \beta_{ig}\hat{\varphi}_{ig}(k)\hat{\varphi}_{ig}^T(k))\tilde{W}_{ig}(k) - \beta_{ig}\hat{\varphi}_{ig}(k)(y_{ig} + B_{ig}k_v r(k))^T \\ &\quad - \rho_{ig} \parallel I - \beta_{ig}\hat{\varphi}_{ig}(k)\hat{\varphi}_{ig}^T(k) \parallel \hat{W}_{ig}(k); i = 1, 2 \end{aligned} \quad (8.4.67)$$

$$\begin{aligned} \tilde{W}_{3f}(k+1) &= (I - \alpha_{3f}\hat{\varphi}_{3f}(k)\hat{\varphi}_{3f}^T(k))\tilde{W}_{3f}(k) - \alpha_{3f}\hat{\varphi}_{3f}(k)(k_v r(k) + \bar{e}_g(k) \\ &\quad + g(x(k))u_d(k) + \epsilon(k) + d(k))^T \\ &\quad - \delta_{3f} \parallel I - \alpha_{3f}\hat{\varphi}_{3f}(k)\hat{\varphi}_{3f}^T(k) \parallel \end{aligned} \quad (8.4.68)$$

and

$$\begin{aligned} \tilde{W}_{3g}(k+1) &= (I - \beta_{3g}\hat{\varphi}_{3g}(k)\hat{\varphi}_{3g}^T(k))\tilde{W}_{3g}(k) - \beta_{3g}\hat{\varphi}_{3g}(k)(k_v r(k) + \bar{e}_f(k) \\ &\quad + g(x(k))u_d(k) + \epsilon(k) + d(k))^T \\ &\quad - \rho_{3g} \parallel I - \alpha_{3g}\hat{\varphi}_{3g}(k)\hat{\varphi}_{3g}^T(k) \parallel. \end{aligned} \quad (8.4.69)$$

Substituting (8.4.66) through (8.4.69) in (8.4.17), combining, substituting for  $g(x)u_d(k)$  from (8.3.29), rewriting and completing the squares for  $\parallel \tilde{W}_{if}(k) \parallel; i = 1, 2, 3$ , and  $\parallel \tilde{W}_{ig}(k) \parallel; i = 1, 2, 3$  one obtains

$$\begin{aligned} \Delta J &\leq -(1 - a_2) \parallel r(k) \parallel^2 + 2a_3 \parallel r(k) \parallel + a_4 \\ &\quad - (1 - \eta - P_3) \parallel \bar{e}_f(k) \parallel^2 - (1 - \eta - P_4) \parallel \bar{e}_g(k) \parallel^2 \end{aligned}$$

$$\begin{aligned}
& -2(1-\eta-P_1) \|\bar{e}_f(k)\| \|\bar{e}_g(k)\| \\
& - \|(\sqrt{P_3}\bar{e}_f(k) + \sqrt{P_4}\bar{e}_g(k)) - (k_v r(k) + g(x)u_d(k) + \epsilon(k) + d(k))\|^2 \\
& + \sum_{i=1}^{i=3} \frac{1}{\alpha_{if}} \|I - \alpha_{if}\hat{\varphi}_{if}(k)\hat{\varphi}_{if}^T(k)\|^2 \operatorname{tr}[\delta_{if}^2 \hat{W}_{if}^T(k)\hat{W}_{if}(k) + 2\delta_{if}\tilde{W}_{if}^T(k)\hat{W}_{if}(k)] \\
& + \sum_{i=1}^{i=3} \frac{1}{\beta_{ig}} \|I - \beta_{ig}\hat{\varphi}_{ig}(k)\hat{\varphi}_{ig}^T(k)\|^2 \operatorname{tr}[\rho_{ig}^2 \hat{W}_{ig}^T(k)\hat{W}_{ig}(k) + 2\delta_{ig}\tilde{W}_{ig}^T(k)\hat{W}_{ig}(k)] \\
& - \sum_{i=1}^{i=2} (2 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k)) \|\hat{W}_{if}^T(k)\hat{\varphi}_{if}(k) - \frac{1}{(2 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))} \\
& ((1 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k)) - \delta_{if}) \|I - \alpha_{if}\hat{\varphi}_{if}(k)\hat{\varphi}_{if}^T(k)\| (y_{if} + B_{if}k_v r(k))\|^2 \\
& + P_5 \|r(k)\|^2 + 2P_6 \|r(k)\| + P_7
\end{aligned} \tag{8.4.70}$$

where

$$a_2 = (2+\eta)k_{vmax}^2 + 2(1+\eta)C_1k_{vmax} + (2+\eta)C_1^2 + 2k_{vmax}C_1 \tag{8.4.71}$$

$$\begin{aligned}
a_3 &= (1+\eta)k_{vmax}(\epsilon_N + d_M + C_0) + P_2k_{vmax} + P_2C_1 + \eta C_1(\epsilon_N + d_M) \\
&+ \frac{1}{2}(2+\eta)C_1(\epsilon_N + d_M + C_0) + 2k_{vmax}(\epsilon_N + d_M + C_0)
\end{aligned} \tag{8.4.72}$$

$$\begin{aligned}
a_{44} &= 2P_2(\epsilon_N + d_M + C_0) + 2\eta C_0(\epsilon_N + d_M) \\
&+ (2+\eta)(\epsilon_N + d_M + C_0)^2 + 2\eta\epsilon_N d_M
\end{aligned} \tag{8.4.73}$$

and

$$\begin{aligned}
a_4 &= a_{44} + \frac{1}{\alpha_{3f}} \|I - \alpha_{3f}\hat{\varphi}_{3f}(k)\hat{\varphi}_{3f}(k)^T\|^2 \frac{\delta_{3f}^2}{(2 - \delta_{3f})} W_{3fmax}^2 \\
&+ \frac{1}{\beta_{3g}} \|I - \beta_{3g}\hat{\varphi}_{3g}(k)\hat{\varphi}_{3g}(k)^T\|^2 \\
&+ \frac{\rho_{3g}^2}{(2 - \rho_{3g})} W_{3gmax}^2
\end{aligned} \tag{8.4.74}$$

$$P_1 = 2(\delta_{3f} \|I - \alpha_{3f}\hat{\varphi}_{3f}(k)\hat{\varphi}_{3f}(k)^T\| + \rho_{3g} \|I - \beta_{3g}\hat{\varphi}_{3g}(k)\hat{\varphi}_{3g}(k)^T\|). \tag{8.4.75}$$

$$\begin{aligned}
P_2 &= 2(\delta_{3f} \|I - \alpha_{3f}\hat{\varphi}_{3f}(k)\hat{\varphi}_{3f}^T(k)\| W_{3fmax}\tilde{\varphi}_{3fmax} + \rho_{3g} \|I - \beta_{3g}\hat{\varphi}_{3g}(k)\hat{\varphi}_{3g}^T(k)\| \\
&W_{3gmax}\tilde{\varphi}_{3gmax}).
\end{aligned} \tag{8.4.76}$$

$$P_3 = (\eta + \delta_{3f} \|I - \alpha_{3f}\hat{\varphi}_{3f}(k)\hat{\varphi}_{3f}^T(k)\|)^2, \tag{8.4.77}$$

and

$$P_4 = (\eta + \rho_{3g} \|I - \beta_{3g}\hat{\varphi}_{3g}(k)\hat{\varphi}_{3g}^T(k)\|)^2. \tag{8.4.78}$$

$$\begin{aligned}
P_5 &= (2\delta_{if} \|I - \alpha_{if}\hat{\varphi}_{if}(k)\hat{\varphi}_{if}^T(k)\| + (\alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k) + \\
&\frac{(1 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k)) - \delta_{if}}{(2 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))} \|I - \alpha_{if}\hat{\varphi}_{if}(k)\hat{\varphi}_{if}^T(k)\|)^2) \|\hat{\varphi}_{if}(k)\|^2 \|W_{if}\|^2 \\
&+ (2\delta_{ig} \|I - \beta_{ig}\hat{\varphi}_{ig}(k)\hat{\varphi}_{ig}^T(k)\| + [\beta_{ig}\hat{\varphi}_{ig}^T(k)\hat{\varphi}_{ig}(k) + \\
&\frac{((1 - \beta_{ig}\hat{\varphi}_{ig}^T(k)\hat{\varphi}_{ig}(k)) - \rho_{ig}}{(2 - \beta_{ig}\hat{\varphi}_{ig}^T(k)\hat{\varphi}_{ig}(k))} \|I - \beta_{ig}\hat{\varphi}_{ig}(k)\hat{\varphi}_{ig}^T(k)\|)^2) \|\hat{\varphi}_{ig}(k)\|^2 \|W_{ig}\|^2
\end{aligned} \tag{8.4.79}$$

$$\begin{aligned}
P_6 &= (\alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k) + \frac{((1 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k)) - \delta_{if} \| I - \alpha_{if} \hat{\varphi}_{if}(k) \hat{\varphi}_{if}^T(k) \|)^2}{(2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k))} + \\
&\quad \delta_{if} \| I - \alpha_{if} \hat{\varphi}_{if}(k) \hat{\varphi}_{if}^T(k) \|) \kappa_{if} k_{vmax} \\
&\quad (\beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k) + \frac{((1 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k)) - \rho_{ig} \| I - \beta_{ig} \hat{\varphi}_{ig}(k) \hat{\varphi}_{ig}^T(k) \|)^2}{(2 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k))} + \\
&\quad \rho_{if} \| I - \beta_{ig} \hat{\varphi}_g(k) \hat{\varphi}_{ig}^T(k) \|) \kappa_{ig} k_{vmax}
\end{aligned} \tag{8.4.80}$$

$$\begin{aligned}
P_7 &= \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k) + \frac{((1 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k)) - \delta_{if} \| I - \alpha_{if} \hat{\varphi}_{if}(k) \hat{\varphi}_{if}^T(k) \|)^2}{(2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k))} \kappa_{if}^2 k_{vmax}^2 \\
&\quad \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k) + \frac{((1 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k)) - \rho_{ig} \| I - \beta_{ig} \hat{\varphi}_{ig}(k) \hat{\varphi}_{ig}^T(k) \|)^2}{(2 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k))} \kappa_{ig}^2 k_{vmax}^2
\end{aligned} \tag{8.4.81}$$

with  $\eta$  is given in (8.4.65). Equation (8.4.70) can be rewritten as

$$\begin{aligned}
\Delta J &\leq -(1 - a_5) \| r(k) \|^2 + 2a_6 \| r(k) \| + a_7 \\
&\quad -(1 - \eta - P_3) \| \bar{e}_f(k) \|^2 - (1 - \eta - P_4) \| \bar{e}_g(k) \|^2 \\
&\quad - 2(1 - \eta - P_1) \| \bar{e}_f(k) \| \| \bar{e}_g(k) \| \\
&\quad - \| (\sqrt{P_3} \bar{e}_f(k) + \sqrt{P_4} \bar{e}_g(k)) - (k_v r(k) + g(x) u_d(k) + \epsilon(k) + d(k)) \|^2 \\
&\quad - \sum_{i=1}^{i=2} (2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k)) \| \tilde{W}_{if}^T(k) \hat{\varphi}_{if}(k) - \frac{1}{(2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k))} \\
&\quad ((1 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k)) - \delta_{if} \| I - \alpha_{if} \hat{\varphi}_{if}(k) \hat{\varphi}_{if}^T(k) \|) (y_{if} + B_{if} k_v r(k)) \|^2 \\
&\quad - \sum_{i=1}^{i=2} (2 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k)) \| \tilde{W}_{ig}^T(k) \hat{\varphi}_{ig}(k) - \frac{1}{(2 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k))} \\
&\quad ((1 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k)) - \rho_{ig} \| I - \beta_{ig} \hat{\varphi}_{ig}(k) \hat{\varphi}_{ig}^T(k) \|) (y_{if} + B_{if} k_v r(k)) \|^2 \\
&\quad - tr(\hat{Z}_f^T(k) C_{1f} \hat{Z}_f(k) - 2 \hat{Z}_f(k) C_{2f} \hat{Z}_f) \\
&\quad - tr(\hat{Z}_g^T(k) C_{1g} \hat{Z}_g(k) - 2 \hat{Z}_g(k) C_{2g} \hat{Z}_g)
\end{aligned} \tag{8.4.82}$$

where  $a_5 = a_2 + P_5$ ,  $a_6 = a_3 + P_6$ ,  $a_7 = P_7 + a_4$ .

Rewriting Equation (8.4.82) one obtains

$$\begin{aligned}
\Delta J &\leq -(1 - a_5) \| r(k) \|^2 + 2a_6 \| r(k) \| + a_7 \\
&\quad -(1 - \eta - P_3) \| \bar{e}_f(k) \|^2 - (1 - \eta - P_4) \| \bar{e}_g(k) \|^2 \\
&\quad - 2(1 - \eta - P_1) \| \bar{e}_f(k) \| \| \bar{e}_g(k) \| \\
&\quad - \| (\sqrt{P_3} \bar{e}_f(k) + \sqrt{P_4} \bar{e}_g(k)) - (k_v r(k) + g(x) u_d(k) + \epsilon(k) + d(k)) \|^2 \\
&\quad - \sum_{i=1}^{i=2} (2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k)) \| \tilde{W}_{if}^T(k) \hat{\varphi}_{if}(k) - \frac{1}{(2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k))} \\
&\quad ((1 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k)) - \delta_{if} \| I - \alpha_{if} \hat{\varphi}_{if}(k) \hat{\varphi}_{if}^T(k) \|) (y_{if} + B_{if} k_v r(k)) \|^2 \\
&\quad - \sum_{i=1}^{i=2} (2 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k)) \| \tilde{W}_{ig}^T(k) \hat{\varphi}_{ig}(k) - \frac{1}{(2 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k))} \\
&\quad ((1 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k)) - \rho_{ig} \| I - \beta_{ig} \hat{\varphi}_{ig}(k) \hat{\varphi}_{ig}^T(k) \|) (y_{if} + B_{if} k_v r(k)) \|^2
\end{aligned}$$

$$\begin{aligned}
& -(2C_{2fmin} - C_{1fmax})[\|\tilde{Z}_f(k)\|^2 - 2\frac{(C_{1fmin} + C_{2fmin})}{(2C_{2fmin} - C_{1fmax})}\|\tilde{Z}_f(k)\|Z_{Mf} - \\
& \frac{C_{1fmax}}{(2C_{2fmin} - C_{1fmax})}Z_{Mf}^2] - (2C_{2gmin} - C_{1gmax})[\|\tilde{Z}_g(k)\|^2 - \\
& 2\frac{(C_{1gmin} + C_{2gmin})}{(2C_{2gmin} - C_{1gmax})}\|\tilde{Z}_g(k)\|Z_{Mg} - \frac{C_{1gmax}}{(2C_{2gmin} - C_{1gmax})}Z_{Mg}^2]
\end{aligned} \quad (8.4.83)$$

where  $C_{1f} = \text{diag}(\delta_{if}^2 \frac{1}{\alpha_{if}} \|I - \alpha_{if}\hat{\varphi}_{if}(k)\hat{\varphi}_{if}^T(k)\|^2)$  and  $C_{2f} = \text{diag}(\delta_{if} \frac{1}{\alpha_{if}} \|I - \alpha_{if}\hat{\varphi}_{if}(k)\hat{\varphi}_{if}^T(k)\|^2)$ . Similarly,  $C_{1g} = \text{diag}(\delta_{ig}^2 \frac{1}{\alpha_{ig}} \|I - \alpha_{ig}\hat{\varphi}_{ig}(k)\hat{\varphi}_{ig}^T(k)\|^2)$  and  $C_{2g} = \text{diag}(\delta_{ig} \frac{1}{\alpha_{ig}} \|I - \alpha_{ig}\hat{\varphi}_{ig}(k)\hat{\varphi}_{ig}^T(k)\|^2)$ . Completing the squares for  $\tilde{Z}_f(k)$  and  $\tilde{Z}_g(k)$  in (8.4.83) one obtains

$$\begin{aligned}
\Delta J &\leq -(1-a_5)\|r(k)\|^2 + 2a_6\|r(k)\| + a_8 \\
&- (1-\eta-P_3)\|\bar{e}_f(k)\|^2 - (1-\eta-P_4)\|\bar{e}_g(k)\|^2 \\
&- 2(1-\eta-P_1)\|\bar{e}_f(k)\|\|\bar{e}_g(k)\| \\
&- \|(\sqrt{P_3}\bar{e}_f(k) + \sqrt{P_4}\bar{e}_g(k)) - (k_v r(k) + g(x)u_d(k) + \epsilon(k) + d(k))\|^2 \\
&- \sum_{i=1}^{i=2} (2 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))\|\tilde{W}_{if}^T(k)\hat{\varphi}_{if}(k) - \frac{1}{(2 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))} \\
&((1 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k)) - \delta_{if}\|I - \alpha_{if}\hat{\varphi}_{if}(k)\hat{\varphi}_{if}^T(k)\|)(y_{if} + B_{if}k_v r(k))\|^2 \\
&- \sum_{i=1}^{i=2} (2 - \beta_{ig}\hat{\varphi}_{ig}^T(k)\hat{\varphi}_{ig}(k))\|\tilde{W}_{ig}^T(k)\hat{\varphi}_{ig}(k) - \frac{1}{(2 - \beta_{ig}\hat{\varphi}_{ig}^T(k)\hat{\varphi}_{ig}(k))} \\
&((1 - \beta_{ig}\hat{\varphi}_{ig}^T(k)\hat{\varphi}_{ig}(k)) - \rho_{ig}\|I - \beta_{ig}\hat{\varphi}_{ig}(k)\hat{\varphi}_{ig}^T(k)\|)(y_{ig} + B_{ig}k_v r(k))\|^2 \\
&- (2C_{2fmin} - C_{1fmax})[\|\tilde{Z}_f(k)\| - \frac{(C_{1fmin} + C_{2fmin})}{(2C_{2fmin} - C_{1fmax})}Z_{Mf}]^2 \\
&- (2C_{2gmin} - C_{1gmax})[\|\tilde{Z}_g(k)\| - \frac{(C_{1gmin} + C_{2gmin})}{(2C_{2gmin} - C_{1gmax})}Z_{Mg}]^2
\end{aligned} \quad (8.4.84)$$

where

$$\begin{aligned}
a_8 &= a_7 + \frac{C_{1fmax}}{(2C_{2fmin} - C_{1fmax})}Z_{Mf}^2 + \frac{(C_{1fmin} + C_{2fmin})}{(2C_{2fmin} - C_{1fmax})}Z_{Mf}^2 \\
&+ \frac{C_{1gmax}}{(2C_{2gmin} - C_{1gmax})}Z_{Mg}^2 + \frac{(C_{1gmin} + C_{2gmin})}{(2C_{2gmin} - C_{1gmax})}Z_{Mg}^2.
\end{aligned} \quad (8.4.85)$$

All the terms in (8.4.84) are always negative except the first term as long as the conditions (8.4.58) through (8.4.64) hold. Since  $a_5, a_6$  and  $a_8$  are positive constants,  $\Delta J \leq 0$  as long as (8.4.58) through (8.4.64) hold with

$$\|r(k)\| > \delta_{r1} \quad (8.4.86)$$

where

$$\delta_{r1} = \frac{1}{(1-a_5)}[a_6 + \sqrt{a_6^2 + a_8(1-a_5)}]. \quad (8.4.87)$$

Similarly, completing the squares for  $\|r(k)\|, \|\tilde{Z}_g(k)\|$  using (8.4.83) yields

$$\Delta J \leq -(1-a_5)[\|r(k)\| - \frac{a_6}{(1-a_5)}]^2$$

$$\begin{aligned}
& -(1 - \eta - P_3) \| \bar{e}_f(k) \|^2 - (1 - \eta - P_4) \| \bar{e}_g(k) \|^2 \\
& - 2(1 - \eta - P_1) \| \bar{e}_f(k) \| \| \bar{e}_g(k) \| \\
& - \| (\sqrt{P_3} \bar{e}_f(k) + \sqrt{P_4} \bar{e}_g(k)) - (k_v r(k) + g(x) u_d(k) + \epsilon(k) + d(k)) \|^2 \\
& - \sum_{i=1}^{i=2} (2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k)) \| \tilde{W}_{if}^T(k) \hat{\varphi}_{if}(k) - \frac{1}{(2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k))} \\
& ((1 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k)) - \delta_{if} \| I - \alpha_{if} \hat{\varphi}_{if}(k) \hat{\varphi}_{if}^T(k) \|) (y_{if} + B_{if} k_v r(k)) \|^2 \\
& - \sum_{i=1}^{i=2} (2 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k)) \| \tilde{W}_{ig}^T(k) \hat{\varphi}_{ig}(k) - \frac{1}{(2 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k))} \\
& ((1 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k)) - \rho_{ig} \| I - \beta_{ig} \hat{\varphi}_{ig}(k) \hat{\varphi}_{ig}^T(k) \|) (y_{ig} + B_{ig} k_v r(k)) \|^2 \\
& - (2C_{2fmin} - C_{1fmax}) [\| \tilde{Z}_f(k) \| - \frac{(C_{1fmin} + C_{2fmin})}{(2C_{2fmin} - C_{1fmax})} \| \tilde{Z}_f(k) \| Z_{Mf} - a_{10}] \\
& - (2C_{2gmin} - C_{1gmax}) [\| \tilde{Z}_g(k) \| - \frac{(C_{1gmin} + C_{2gmin})}{(2C_{2gmin} - C_{1gmax})} Z_{Mg}]^2
\end{aligned} \tag{8.4.88}$$

where

$$a_{10} = \frac{(C_{1fmin} + C_{2fmin})^2}{(2C_{2fmin} - C_{1fmax})} Z_{Mf} \tag{8.4.89}$$

$$a_{11} = \frac{C_{1fmax}}{(2C_{2fmin} - C_{1fmax})} Z_{Mf}^2 + \frac{(C_{1gmin} + C_{2gmin})^2}{(2C_{2gmin} - C_{1gmax})} Z_{Mg}^2. \tag{8.4.90}$$

Then  $\Delta J \leq 0$  as long as (8.4.58) through (8.4.64) hold and the quadratic term for  $\tilde{Z}_f(k)$  in (8.4.88) is positive, which is guaranteed when

$$\| \tilde{Z}_f(k) \| > \delta_{f1} \tag{8.4.91}$$

where

$$\delta_{f1} = a_{10} + \sqrt{a_{10}^2 + a_{11}}. \tag{8.4.92}$$

Similarly, completing the squares for  $\| r(k) \|$ ,  $\| \tilde{Z}_f(k) \|$  using (8.4.83) yields

$$\begin{aligned}
\Delta J &\leq -(1 - a_5) [\| r(k) \| - \frac{a_6}{(1 - a_5)}]^2 \\
&- (1 - \eta - P_3) \| \bar{e}_f(k) \|^2 - (1 - \eta - P_4) \| \bar{e}_g(k) \|^2 \\
&- 2(1 - \eta - P_1) \| \bar{e}_f(k) \| \| \bar{e}_g(k) \| \\
&- \| (\sqrt{P_3} \bar{e}_f(k) + \sqrt{P_4} \bar{e}_g(k)) - (k_v r(k) + g(x) u_d(k) + \epsilon(k) + d(k)) \|^2 \\
&- \sum_{i=1}^{i=2} (2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k)) \| \tilde{W}_{if}^T(k) \hat{\varphi}_{if}(k) - \frac{1}{(2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k))} \\
&((1 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k)) - \delta_{if} \| I - \alpha_{if} \hat{\varphi}_{if}(k) \hat{\varphi}_{if}^T(k) \|) (y_{if} + B_{if} k_v r(k)) \|^2 \\
&- \sum_{i=1}^{i=2} (2 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k)) \| \tilde{W}_{ig}^T(k) \hat{\varphi}_{ig}(k) - \frac{1}{(2 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k))} \\
&((1 - \beta_{ig} \hat{\varphi}_{ig}^T(k) \hat{\varphi}_{ig}(k)) - \rho_{ig} \| I - \beta_{ig} \hat{\varphi}_{ig}(k) \hat{\varphi}_{ig}^T(k) \|) (y_{ig} + B_{ig} k_v r(k)) \|^2 \\
&- (2C_{2gmin} - C_{1gmax}) [\| \tilde{Z}_g(k) \| - \frac{(C_{gmin} + C_{2gmin})}{(2C_{2gmin} - C_{1gmax})} \| \tilde{Z}_g(k) \| Z_{Mg} - a_{10}] \\
&- (2C_{2fmin} - C_{1fmax}) [\| \tilde{Z}_f(k) \| - \frac{(C_{1fmin} + C_{2fmin})}{(2C_{2fmin} - C_{1fmax})} Z_{Mf}]^2
\end{aligned} \tag{8.4.93}$$

where

$$a_{10} = \frac{(C_{1gmin} + C_{2gmin})^2}{(2C_{2gmin} - C_{1gmax})} Z_{Mg} \quad (8.4.94)$$

$$a_{11} = \frac{C_{1gmax}}{(2C_{2gmin} - C_{1gmax})} Z_{Mg}^2 + \frac{(C_{1fmin} + C_{2fmin})^2}{(2C_{2fmin} - C_{1fmax})} Z_{Mf}^2. \quad (8.4.95)$$

Then  $\Delta J \leq 0$  as long as (8.4.58) through (8.4.64) hold and the quadratic term for  $\tilde{Z}_g(k)$  in (8.4.93) is positive, which is guaranteed when

$$\|\tilde{Z}_g(k)\| > \delta_{g1} \quad (8.4.96)$$

where

$$\delta_{g1} = a_{10} + \sqrt{a_{10}^2 + a_{11}}. \quad (8.4.97)$$

We have shown upper bounds for the tracking error and the NN weight estimation errors for this region for all  $|u_c(k)| \leq s$ .

Region II:  $|\hat{g}(x)| \leq g$  and  $|u_c(k)| > s$ .

The filtered tracking error dynamics can be written for this region as (8.4.36). Now using the Lyapunov function (8.4.16), the first difference (8.4.17), after substituting for  $g(x)u_d(k)$  from (8.3.49) in (8.4.17) and manipulating accordingly, one can obtain

$$\begin{aligned} \Delta J = & -(1 - b_0) \|r(k)\|^2 + 2b_1 \|r(k)\| + b_2 \\ & -(1 - \eta) \|\bar{e}_f(k) - \frac{\eta}{(1 - \eta)}(k_v r(k) + \overline{g(x)u_d(k)} + \epsilon(k) + d(k))\|^2, \\ & - \sum_{i=1}^{i=2} (2 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k)) \|\hat{y}_{if}(k) - \frac{(1 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))}{(1 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))} (y_{if} + B_{if}k_v r(k))\|^2 \\ & + b_3 \|r(k)\|^2 + b_4 \|r(k)\| + b_5 \\ & + \sum_{i=1}^{i=3} \frac{1}{\alpha_{if}} \|I - \alpha_{if}\hat{\varphi}_{if}(k)\hat{\varphi}_{if}^T(k)\|^2 [\delta_{if}^2 \hat{W}_{if}^T(k) \hat{W}_{if}(k) + 2\delta_{if} \tilde{W}_{if} \hat{W}_{if}(k)] \end{aligned} \quad (8.4.98)$$

where  $b_0, b_1, b_2, b_3, b_4, b_5$  are presented in (8.4.39) through (8.4.45), with  $\eta$  given by (8.4.65).

$$\begin{aligned} \Delta J = & -(1 - b_6) \|r(k)\|^2 + 2b_7 \|r(k)\| + b_8 \\ & -(1 - \eta) \|\bar{e}_f(k) - \frac{\eta}{(1 - \eta)}(k_v r(k) + \overline{g(x)u_d(k)} + \epsilon(k) + d(k))\|^2 \\ & - \sum_{i=1}^{i=2} (2 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k)) \|\hat{y}_{if}(k) - \frac{(1 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))}{(1 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))} (y_{if} + B_{if}k_v r(k))\|^2 \\ & - (2C_{2fmin} - C_{1fmax}) [\|\tilde{Z}_f(k)\|^2 - 2 \frac{(C_{1fmin} + C_{2fmin})}{(2C_{2fmin} - C_{1fmax})} \|\tilde{Z}_f(k)\| Z_{Mf} - \\ & \quad \frac{C_{1fmax}}{(2C_{2fmin} - C_{1fmax})} Z_{Mf}^2] \end{aligned} \quad (8.4.99)$$

with  $b_6 = b_0 + b_3, b_7 = b_2 + b_4, b_8 = b_2 + b_5$ .

Complete the squares for  $\|\tilde{Z}_f(k)\|$  using (8.4.99) to obtain

$$\begin{aligned} \Delta J = & -(1 - b_6) \|r(k)\|^2 + 2b_7 \|r(k)\| + b_9 \\ & -(1 - \eta) \|\bar{e}_f(k) - \frac{\eta}{(1 - \eta)}(k_v r(k) + \overline{g(x)u_d(k)} + \epsilon(k) + d(k))\|^2, \end{aligned}$$

$$\begin{aligned}
& - \sum_{i=1}^{i=2} (2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k)) \| \hat{y}_{if}(k) - \frac{(1 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k))}{(1 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k))} (y_{if} + B_{if} k_v r(k)) \|^2 \\
& - (2C_{2fmin} - C_{1fmax}) [\| \tilde{Z}_f(k) \| - \frac{(C_{1fmin} + C_{2fmin})}{(2C_{2fmin} - C_{1fmax})} Z_{Mf}]^2
\end{aligned} \quad (8.4.100)$$

The terms in (8.4.100) are always negative as long as the conditions (8.4.58) through (8.4.64) hold. Since  $b_6, b_7$  and  $b_9$  are positive constants,  $\Delta J \leq 0$  as long as

$$\| r(k) \| > \delta_{r2}, \quad (8.4.101)$$

where

$$\delta_{r2} = \frac{1}{(1 - b_6)} [b_7 + \sqrt{b_7^2 + b_9(1 - b_6)}]. \quad (8.4.102)$$

with

$$b_9 = b_8 + \frac{(2C_{1fmax} + C_{2fmax})}{(2C_{2fmin} - C_{1fmax})} Z_{Mf}^2. \quad (8.4.103)$$

Similarly, completing the squares for  $\| r(k) \|$  in (8.4.99) one obtains

$$\begin{aligned}
\Delta J = & -(1 - b_6) [\| r(k) \| - \frac{b_7}{(1 - b_6)}]^2 \\
& - (1 - \eta) \| \bar{e}_f(k) - \frac{\eta}{(1 - \eta)} (k_v r(k) + \overline{g(x) u_d(k)} + \epsilon(k) + d(k)) \|^2, \\
& - \sum_{i=1}^{i=2} (2 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k)) \| \hat{y}_{if}(k) - \frac{(1 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k))}{(1 - \alpha_{if} \hat{\varphi}_{if}^T(k) \hat{\varphi}_{if}(k))} (y_{if} + B_{if} k_v r(k)) \| \\
& - (2C_{2fmin} - C_{1fmax}) [\| \tilde{Z}_f(k) \|^2 - 2 \frac{(C_{1fmin} + C_{2fmin})}{(2C_{2fmin} - C_{1fmax})} \| \tilde{Z}_f(k) \| Z_{Mf} \\
& - \frac{C_{1fmax}}{(2C_{2fmin} - (C_{1fmax})} Z_{Mf}^2 \\
& - \frac{1}{(2C_{2fmin} - C_{1fmax})} (Z_M^2 + b_8 + \frac{b_7^2}{(1 - b_6)})].
\end{aligned} \quad (8.4.104)$$

The terms in (8.4.104) are always negative as long as the conditions (8.4.58) through (8.4.64) hold except the last term.  $\Delta J \leq 0$  as long as

$$\| \tilde{Z}_f \| > \delta_{f2}, \quad (8.4.105)$$

with

$$\delta_{f2} = b_9 + \sqrt{b_9^2 + b_{10}} \quad (8.4.106)$$

with

$$b_9 = \frac{(2C_{1fmin} + C_{2fmin})}{(2C_{2fmin} - C_{1fmax})} Z_{Mf}^2 \quad (8.4.107)$$

and

$$\begin{aligned}
b_{10} = & \frac{C_{1fmax}}{(2C_{2fmin} - (C_{1fmax})} Z_{Mf}^2 - \frac{1}{(2C_{2fmin} - C_{1fmax})} (Z_M^2 + b_8 + \frac{b_7^2}{(1 - b_6)})
\end{aligned} \quad (8.4.108)$$

One has  $|\sum_{k=k_0}^{\infty} \Delta J(k)| = |J(\infty) - J(0)| < \infty$  since  $\Delta J \leq 0$  as long as (8.4.58) through (8.4.64) hold. The definition of  $J$  and inequalities (8.4.101) and (8.4.105) imply that every initial condition in the set  $\chi$  will evolve entirely within  $\chi$ . Thus according to the standard Lyapunov extension, it can be concluded that the tracking error  $r(k)$  and the error in weight updates are UUB.

Reprise:

Combining the results from region I and II, one can readily set  $\delta_r = \max(\delta_{r1}, \delta_{r2})$ ,  $\delta_f = \max(\delta_{f1}, \delta_{f2})$ , and  $\delta_g$ . Thus for both regions, if  $\|r(k)\| > \delta_r$ , then  $\Delta J \leq 0$  and  $u(k)$  is bounded. Let us denote  $(\|r(k)\|, \|\tilde{W}_f\|, \|\tilde{W}_g\|)$  by new coordinate variables  $(\xi_1, \xi_2, \xi_3)$ . Define the region

$$\Xi : \xi_1 < \delta_r, \xi_2 < \delta_f, \xi_3 < \delta_g,$$

then there exists an open set

$$\Omega : \xi_1 < \bar{\delta}_r, \xi_2 < \bar{\delta}_f, \xi_3 < \bar{\delta}_g,$$

where  $\bar{\delta}_i > \delta_i$  implies that  $\Xi \subset \Omega$ . In other words, we have proved that whenever  $\xi_i > \delta_i$ , then  $J(\xi)$  will not increase and will remain in the region  $\Omega$  which is an invariant set. Therefore all the signals in the closed-loop system remain bounded. This concludes the proof.  $\square$

#### Example 8.4.2 (Control Using NN Tuning not Requiring PE) :

For Example 8.4.1, the response of the neural network controller with the improved weight tuning (8.4.54) through (8.4.57) and projection algorithm is presented in Fig. 8.4.2. The design parameters  $\Gamma_i; i = 1, 2, 3$  and  $\rho_i; i = 1, 2, 3$  are selected as 0.01. Note that with the improved weight tuning, the output of the neural network remains bounded because the weights are guaranteed to remain bounded without the necessity of persistency of excitation. To study the contribution of the neural network, Fig. 8.4.3 shows the response of the PD controller with no neural network. It is clear that the addition of the neural network makes a significant improvement in the tracking performance.  $\square$

#### Example 8.4.3 (NN Control of Discrete-Time Nonlinear System) :

Consider the first-order multi-input/multi-output discrete-time nonlinear system described by

$$X(k+1) = F(X) + G(X)U(k), \quad (8.4.109)$$

where  $X(k) = [x_1(k), x_2(k)]^T$ ,  $F(X) = \begin{bmatrix} \frac{x_2(k)}{1+x_2^2(k)} \\ \frac{x_1(k)}{1+x_1^2(k)} \end{bmatrix}$ ,  $G(X) = \begin{bmatrix} \frac{1}{1+x_1^2(k)} & 0 \\ 0 & \frac{1}{1+x_2^2(k)} \end{bmatrix}$  and

the input is given by  $U(k) = [u_1(k), u_2(k)]^T$ . The objective is to track a periodic step input of magnitude two units with a period of 30 s. The elements of the diagonal matrix

were chosen as  $k_v = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}$  and a sampling interval of 10 ms was considered.

Multi-layer NN were selected with 12 hidden-layer nodes. Sigmoidal activation functions were employed in all the nodes in the hidden layer. The initial conditions for the plant were chosen to be  $[1, -1]^T$ . The weights were initialized to zero for  $F(\cdot)$  and identity matrix for  $G(\cdot)$  with an initial threshold value of 3.0. The design parameters  $\Gamma_i; i = 1, 2, 3$  and  $\rho_i; i = 1, 2, 3$  were selected to be 0.01. No learning is performed initially to train the networks. The design parameters for the projection algorithm were selected to be  $\xi_3 = 0.5, \xi_i = 1.0; i = 1, 2$  with  $\zeta_i = 0.001; \forall i = 1, 2, 3$  for both NN.

In this example, only results using the improved weight tuning are presented. The response of the controller with the improved weight tuning (8.4.54) through (8.4.57) is

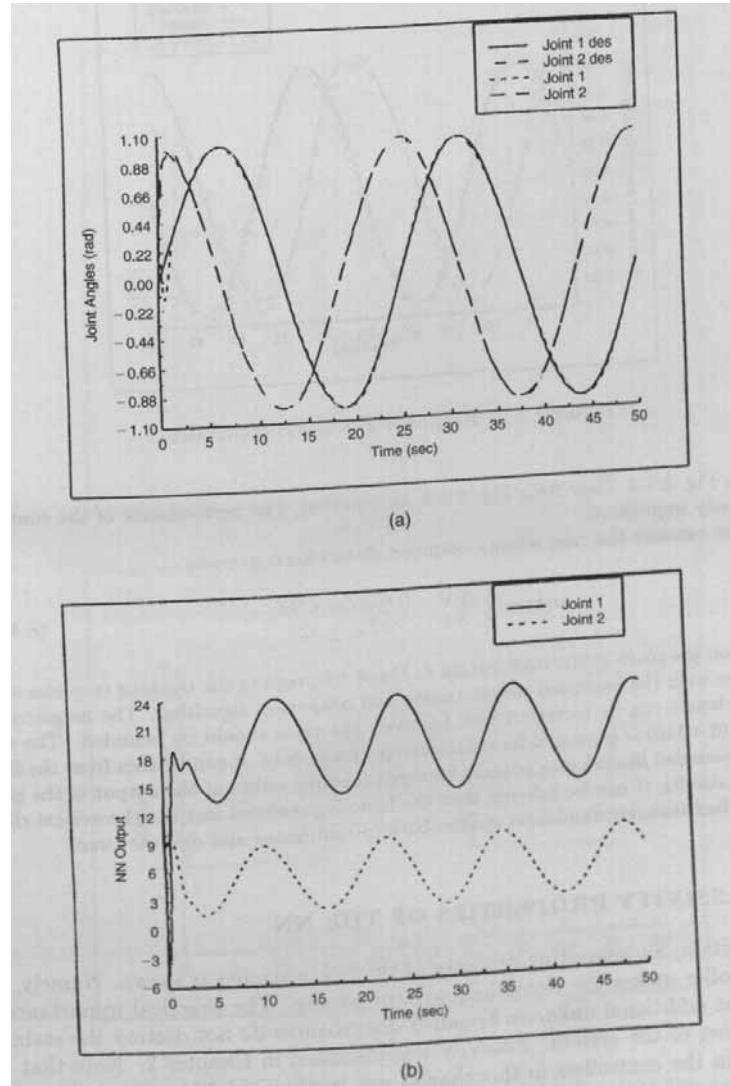


Figure 8.4.2: Response of the NN controller with improved weight tuning and projection algorithm. (a) Actual and desired joint angles. (b) Neural network outputs.

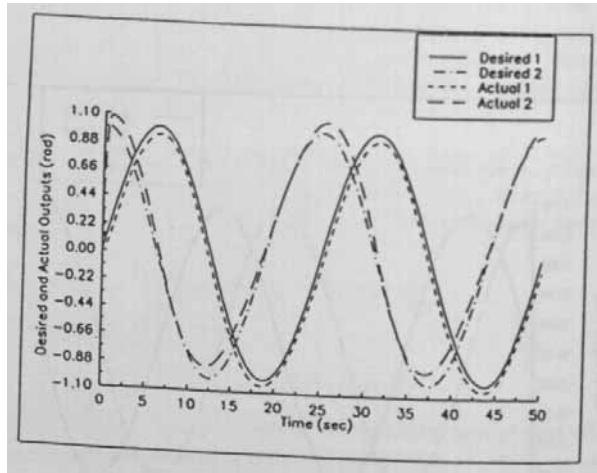


Figure 8.4.3: Response of the PD controller.

shown in Fig. 8.4.4. Note from Fig. 8.4.4, as expected, The performance of the controller is extremely impressive.

Let us consider the case when a bounded disturbance given by

$$w(k) = \begin{cases} 0.0 & 0 \leq kT_m < 12 \\ 0.1 & kT_m \geq 12 \end{cases} \quad (8.4.110)$$

is acting on the plant at the time instant  $t$ . Fig. 8.4.5 presents the tracking response of NN controllers with the improved weight tuning and projection algorithm. The magnitude of the disturbance can be increased and, however, the value should be bounded. The value shown in (8.4.110) is employed for simulation purposes only. It can be seen from the figure that the bounded disturbance induces bounded tracking errors at the output of the plant. From the results, it can be inferred that the bounds presented and the theoretical claims were justified through simulation studies both in continuous and discrete-time.  $\square$

## 8.5 PASSIVITY PROPERTIES OF THE NN

*In this section, an interesting property of the NN controller is shown. Namely, the NN controller makes the closed-loop system passive. The practical importance of this is that additional unknown bounded disturbances do not destroy the stability and tracking of the system. Passivity was discussed in Chapter 2. Note that the NN used in the controllers in this chapter are feedforward NN with no dynamics. However, tuning them on-line turns them into dynamical systems, so that passivity properties can be defined.*

*The closed-loop error system (8.2.27) is shown in Fig. 8.5.1 using a one-layer neural network; note that the NN now is in the standard feedback configuration as opposed to the NN controller in Fig. 8.2.1, which has both feedback and feedforward connections. Passivity is essential in a closed-loop system as it guarantees the boundedness of the signals, and hence suitable performance, even in the presence of*

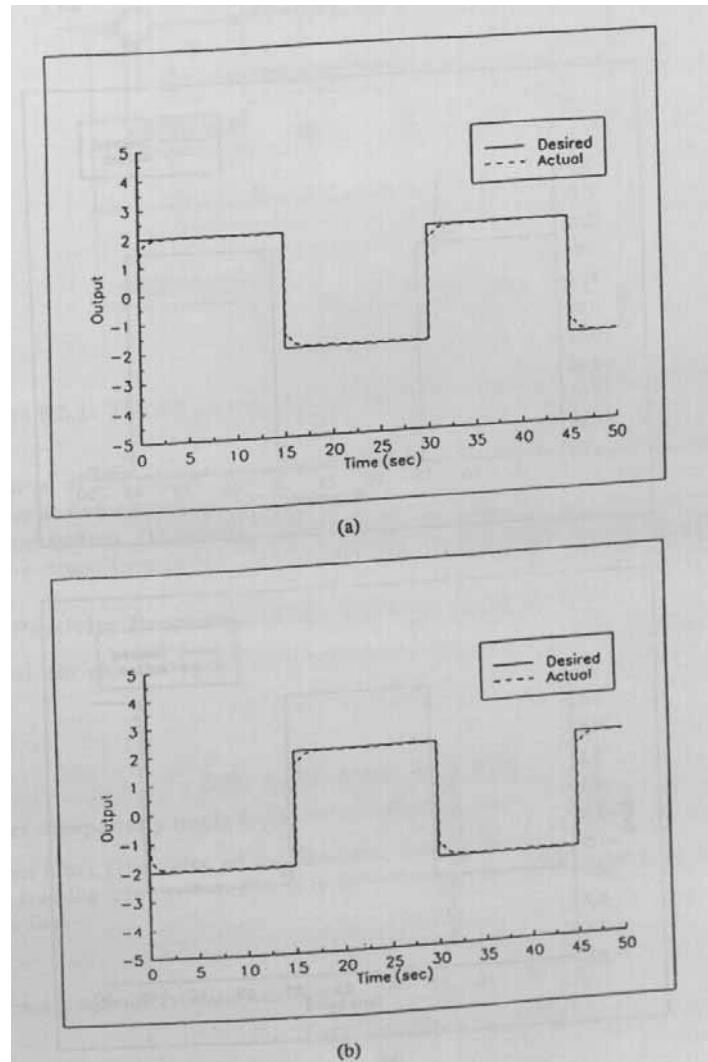


Figure 8.4.4: Response of the NN controller with improved weight tuning and projection algorithm. (a) Desired and actual state 1. (b) Desired and actual state 2.

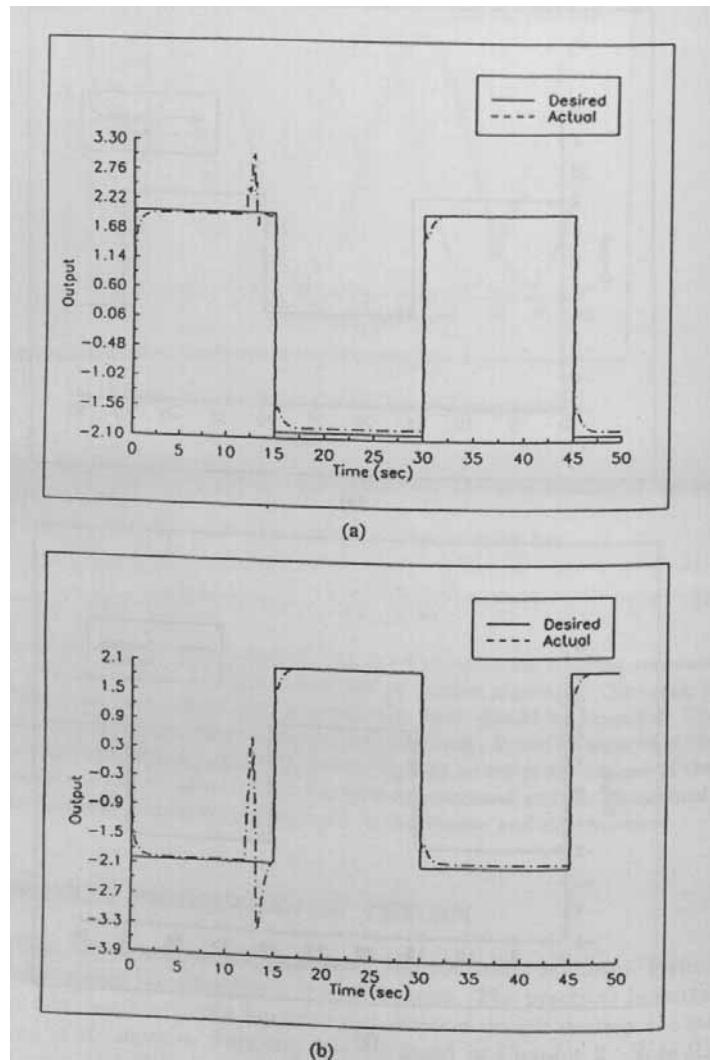


Figure 8.4.5: Response of NN controller with improved weight tuning in the presence of bounded disturbances. (a) Desired and actual state 1. (b) Desired and actual state 2.

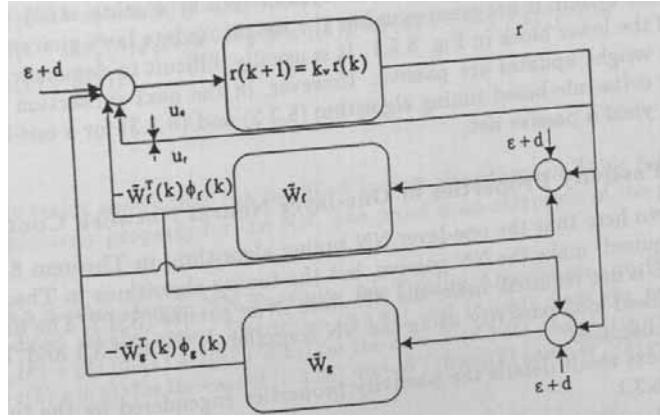


Figure 8.5.1: The NN closed-loop system using a one-layer neural network.

additional unforeseen bounded disturbances. This equates to robustness of the closed-loop system. Therefore, in this section the passivity properties of the NN, and of the closed-loop system, are explored for various weight tuning algorithms.

### 8.5.1 Passivity Properties of the Tracking Error System

In general, the closed-loop tracking system (8.1.8) can also be expressed as

$$r(k+1) = k_v r(k) + \zeta_0(k) \quad (8.5.1)$$

where

$$\zeta_0(k) = \tilde{f}(x) + \tilde{g}(x)u(k) + d(k). \quad (8.5.2)$$

The next dissipativity result holds for this system.

#### Theorem 8.5.1 (Passivity of the Tracking Error System) :

The tracking error system (8.5.1) is state strict passive (SSP) from  $\zeta_0(k)$  to  $k_v r(k)$  provided that

$$k_v^T k_v < I. \quad (8.5.3)$$

#### Proof:

Select a Lyapunov function candidate

$$J = r^T(k)r(k). \quad (8.5.4)$$

The first difference is given by

$$\Delta J = r^T(k+1)r(k+1) - r^T(k)r(k). \quad (8.5.5)$$

Substituting (8.5.1) in (8.5.5) yields

$$\Delta J = -r^T(k)[I - k_v^T k_v]r(k) + 2r^T(k)k_v \zeta_0 + \zeta_0^T(k)\zeta_0. \quad (8.5.6)$$

Note (8.5.6) is in power form defined in Chapter 2 with the first term taken as  $h(k)$ , a quadratic function of the state  $r(k)$ . Hence (8.5.1) is a state strict passive system.  $\square$

Even though the closed-loop error system (8.5.1) is state strict passive, the closed-loop system is not passive unless the weight update laws guarantee the passivity of the lower block in Fig. 8.5.1. It is usually difficult to demonstrate that the error in weight updates are passive. However, in the next subsection it is shown that the delta-rule-based tuning algorithm (8.3.2) and (8.3.3) for a one-layer neural network yield a passive net.

### 8.5.2 Passivity Properties of One-layer Neural Network Controllers

It is shown here that the one-layer NN tuning algorithms in Theorem 8.3.1, where PE is required, make the NN passive, but the tuning algorithms in Theorem 8.3.2, where PE is not required, make the NN state strict passive (SSP). The implications for the closed-loop passivity using the NN controller in Table 8.3.1 and Table 8.3.2 are then discussed.

The next result details the passivity properties engendered by the tuning rules in Table 8.3.1.

**Theorem 8.5.2 (One-Layer NN Passivity for Tuning with PE) :**

The weight tuning algorithms (8.3.18) and (8.3.19) make the map from  $k_r(k) + \bar{e}_g(k) + g(x)u_d(k) + \epsilon(k) + d(k)$  for the case of (8.3.18) and  $k_v r(k) + \bar{e}_f(k) + g(x)u_d(k) + \epsilon(k) + d(k)$  for the case of (8.3.19), to  $-\tilde{W}_f^T(k)\varphi_f(k)$  and  $-\tilde{W}_g^T(k)\varphi_g(k)$  passive maps.

**Proof:**

Define the Lyapunov function candidate

$$J = \frac{1}{\alpha} \text{tr}[\tilde{W}_f^T(k)\tilde{W}_f(k)], \quad (8.5.7)$$

whose first difference is given by

$$\Delta J = \frac{1}{\alpha} \text{tr}[\tilde{W}_f^T(k+1)\tilde{W}_f(k+1) - \tilde{W}_f^T(k)\tilde{W}_f(k)]. \quad (8.5.8)$$

Substituting the weight update law (8.3.18) in (8.5.8) yields

$$\begin{aligned} \Delta J = & -(2 - \alpha\varphi_f^T(k)\varphi_f(k))(-\tilde{W}_f^T(k)\varphi_f(k))^T(-\tilde{W}_f^T(k)\varphi_f(k)) + \\ & \alpha\varphi_f^T(k)\varphi_f(k)(k_v r(k) + \bar{e}_g(k) + g(x)u_d(k) + \epsilon(k) + d(k))^T(k_v r(k) \\ & + \bar{e}_g(k) + g(x)u_d(k) + \epsilon(k) + d(k)) \\ & + 2(1 - \alpha\varphi_f^T(k)\varphi_f(k))(-\tilde{W}_f^T(k)\varphi_f(k))(k_v r(k) + \bar{e}_g(k) + g(x)u_d(k) \\ & + \epsilon(k) + d(k)). \end{aligned} \quad (8.5.9)$$

Note (8.5.9) is in power form defined in Chapter 2 as long as the condition (8.3.4) holds. This in turn guarantees the passivity of the weight tuning mechanism (8.3.18).

Similarly one can also prove that the error in weight updates presented in (8.3.19) is passive as long as the the PE condition is satisfied. In fact, if one chooses the first difference as

$$J = \frac{1}{\beta} \text{tr}[\tilde{W}_g^T(k+1)\tilde{W}_g(k+1) - \tilde{W}_g^T(k)\tilde{W}_g(k)]. \quad (8.5.10)$$

Using the error in update law (8.3.19) and simplifying one obtains

$$\begin{aligned} \Delta J = & -(2 - \beta\varphi_g^T(k)\varphi_g(k))(-\tilde{W}_g^T(k)\varphi_g(k))^T(-\tilde{W}_g^T(k)\varphi_g(k)) + \\ & \alpha\varphi_g^T(k)\varphi_g(k)(k_v r(k) \end{aligned}$$

$$\begin{aligned}
& + \bar{e}_f(k) + g(x)u_d(k) + \epsilon(k) + d(k))^T(k_v r(k) + \bar{e}_f(k) + g(x)u_d(k) + \epsilon(k) + d(k)) \\
& + 2(1 - \beta\varphi_g^T(k)\varphi_g(k))(-\tilde{W}_g^T(k)\varphi_g(k))(k_v r(k) + \bar{e}_f(k) + g(x)u_d(k) \\
& + \epsilon(k) + d(k)).
\end{aligned} \tag{8.5.11}$$

□

The next result shows that the modified tuning algorithms in Table 8.3.2 yield a stronger passivity property for the NN. The proof is an extension of the previous one.

**Theorem 8.5.3 (One-Layer NN Passivity for Tuning Algorithms, no PE) :**

The modified weight tuning algorithms (8.3.65) and (8.3.66) make the map from,  $(k_v r(k) + \bar{e}_g(k) + g(x)u_d(k) + \epsilon(k) + d(k))$  for the case of (8.3.65) and  $(k_v r(k) + \bar{e}_f(k) + g(x)u_d(k) + \epsilon(k) + d(k))$  for the case of (8.3.66), to  $-\tilde{W}_f^T(k)\varphi_f(k)$  and  $-\tilde{W}_g^T(k)\varphi_g(k)$  state strict passive maps. □

It has been shown that the filtered tracking error system (8.2.27) in Fig. 8.5.1 is state strict passive, while the NN weight error block is passive using the tuning rules in Table 8.3.1. Thus, using standard results (Slotine and Li 1991), it can be concluded that the closed-loop system is passive. Therefore, according to the Passivity Theorem one can conclude that the inputs/output signals of each block are bounded as long as the disturbances are bounded. Though passive, however, the closed-loop system is not state strict passive so this does not yield boundedness of the internal states of the lower blocks (e.g.  $\tilde{W}_f(k)$  and  $\tilde{W}_g(k)$ ) unless PE holds for the case of one-layer NN case.

On the other hand, the enhanced tuning rules of Table 8.3.2 yield a SSP weight tuning block in the figure, so that the closed-loop system is SSP. Thus, the internal states of the lower blocks (e.g.  $\tilde{W}_f(k)$  and  $\tilde{W}_g(k)$ ) are bounded even if PE does not hold. Thus, the modified tuning algorithms guarantee SSP of the weight tuning blocks, so that the closed-loop system is SSP. Therefore, internal stability can be guaranteed even in the absence of PE.

### 8.5.3 Passivity Properties of Multilayer Neural Network Controllers

It is shown here that the multilayer NN tuning algorithms in Theorem 8.4.1, where PE is required, make the NN passive, but the tuning algorithms in Theorem 8.4.2, where PE is not required, make the NN state strict passive (SSP). The implications for the closed-loop passivity using the NN controller in Table 8.4.1 and Table 8.4.2 are then discussed.

The next result details the passivity properties engendered by the tuning rules in Table 8.4.1.

**Theorem 8.5.4 (Multilayer NN Passivity, Tuning Algorithms with PE) :**

The weight tuning algorithms (8.4.2) and (8.4.4) make the map from  $k_r(k) + \bar{e}_g(k) + g(x)u_d(k) + \epsilon(k) + d(k)$  for the case of (8.4.2) and  $k_v r(k) + \bar{e}_f(k) + g(x)u_d(k) + \epsilon(k) + d(k)$  for the case of (8.4.4), to  $-\tilde{W}_{3f}^T(k)\hat{\varphi}_{3f}(x(k))$  and  $-\tilde{W}_{3g}^T(k)\hat{\varphi}_{3g}(x(k))$  passive maps.

The weight tuning algorithms for the hidden layers (8.4.1) and (8.4.3) make the map from  $y_{if}(k) + B_{if}k_v r(k)$  for the case of (8.4.1) and  $y_{ig}(k) + B_{ig}k_v r(k)$  for the case of (8.4.3), to  $\tilde{W}_{if}^T(k)\hat{\varphi}_{if}(x(k))$  and  $\tilde{W}_{ig}^T(k)\hat{\varphi}_{ig}(x(k))$  passive maps.

Proof:

Define the Lyapunov function candidate

$$J = \frac{1}{\alpha_{3f}} \text{tr}[\tilde{W}_{3f}^T(k) \tilde{W}_{3f}(k)], \quad (8.5.12)$$

whose first difference is given by

$$\Delta J = \frac{1}{\alpha_{3f}} \text{tr}[\tilde{W}_{3f}^T(k+1) \tilde{W}_{3f}(k+1) - \tilde{W}_{3f}^T(k) \tilde{W}_{3f}(k)]. \quad (8.5.13)$$

Substituting the weight update law (8.4.2) in (8.5.13) yields

$$\begin{aligned} \Delta J = & -(2 - \alpha_{3f} \hat{\varphi}_{3f}^T(k) \hat{\varphi}_{3f}(k))(-\tilde{W}_{3f}^T(k) \hat{\varphi}_{3f}(k))^T (-\tilde{W}_{3f}^T(k) \hat{\varphi}_{3f}(k)) \\ & + \alpha_{3f} \hat{\varphi}_{3f}^T(k) \hat{\varphi}_{3f}(k) (k_v r(k) + \bar{e}_g(k) + g(x) u_d(k) + \epsilon(k) \\ & + d(k))^T (k_v r(k) + \bar{e}_g(k) + g(x) u_d(k) + \epsilon(k) + d(k)) \\ & + 2(1 - \alpha_{3f} \hat{\varphi}_{3f}^T(k) \hat{\varphi}_{3f}(k))(-\tilde{W}_{3f}^T(k) \hat{\varphi}_{3f}(k)) (k_v r(k) + \bar{e}_g(k) \\ & + g(x) u_d(k) + \epsilon(k) + d(k)). \end{aligned} \quad (8.5.14)$$

Note (8.5.14) is in power form defined in Chapter 2 as long as conditions (8.4.8) through (8.4.13) holds. This in turn guarantees the passivity of the weight tuning mechanism (8.4.2).

Similarly one can also prove that the error in weight updates presented in (8.4.4) is passive as long as the PE condition is satisfied. In fact, if one chooses the first difference as

$$J = \frac{1}{\beta_{3g}} \text{tr}[\tilde{W}_{3g}^T(k+1) \tilde{W}_{3g}(k+1) - \tilde{W}_{3g}^T(k) \tilde{W}_{3g}(k)]. \quad (8.5.15)$$

Using the error in update law (8.4.4) and simplifying one obtains

$$\begin{aligned} \Delta J = & -(2 - \beta_{3g} \hat{\varphi}_{3g}^T(k) \hat{\varphi}_{3g}(k))(-\tilde{W}_{3g}^T(k) \hat{\varphi}_{3g}(k))^T (-\tilde{W}_{3g}^T(k) \hat{\varphi}_{3g}(k)) \\ & + \alpha_{3g} \hat{\varphi}_{3g}^T(k) \hat{\varphi}_{3g}(k) (k_v r(k) + \bar{e}_f(k) + g(x) u_d(k) + \epsilon(k) \\ & + d(k))^T (k_v r(k) + \bar{e}_f(k) + g(x) u_d(k) + \epsilon(k) + d(k)) \\ & + 2(1 - \beta_{3g} \hat{\varphi}_{3g}^T(k) \hat{\varphi}_{3g}(k))(-\tilde{W}_{3g}^T(k) \hat{\varphi}_{3g}(k)) (k_v r(k) + \bar{e}_f(k) \\ & + g(x) u_d(k) + \epsilon(k) + d(k)). \end{aligned} \quad (8.5.16)$$

Similarly, it can be shown that the hidden layer updates yield passive NN. The proof can be obtained from chapter 7.  $\square$

The next result shows that the modified tuning algorithms in Table 8.4.2 yield a stronger passivity property for the NN. The proof is an extension of the previous one.

**Theorem 8.5.5 (Multilayer NN Passivity of Algorithms without PE) :**

The modified weight tuning algorithms (8.4.55) and (8.4.57) make the map from,  $(k_v r(k) + \bar{e}_g(k) + g(x) u_d(k) + \epsilon(k) + d(k))$  for the case of (8.4.55) and  $(k_v r(k) + \bar{e}_f(k) + g(x) u_d(k) + \epsilon(k) + d(k))$  for the case of (8.4.57), to  $-\tilde{W}_{3f}^T(k) \hat{\varphi}_{3f}(k)$  and  $-\tilde{W}_{3g}^T(k) \hat{\varphi}_{3g}(k)$  state strict passive maps.

The weight tuning algorithms for the hidden layers (8.4.54) and (8.4.56) make the map from  $y_{if}(k) + B_{if} k_v r(k)$  for the case of (8.4.54) and  $y_{ig}(k) + B_{ig} k_v r(k)$  for the case of (8.4.56), to  $\tilde{W}_{if}^T(k) \varphi_{if}(x(k))$  and  $\tilde{W}_{ig}^T(k) \varphi_{ig}(x(k))$  passive maps.  $\square$

*It has been shown that the filtered tracking error system (8.2.27) in Fig. 8.5.1 and (8.2.28) is state strict passive, while the NN weight error block is passive using the tuning rules in Table 8.4.1. Thus, it can be concluded that the closed-loop system is passive. Therefore, according to the Passivity Theorem one can conclude that the inputs/output signals of each block are bounded as long as the disturbances are bounded. Though passive, however, the closed-loop system is not state strict passive so this does not yield boundedness of the internal states of the lower blocks (e.g.  $\tilde{W}_f(k)$  and  $\tilde{W}_g(k)$ ) unless PE holds.*

*On the other hand, the enhanced tuning rules of Table 8.4.2 yield a SSP weight tuning block in the figure, so that the closed-loop systems is SSP. Thus, the internal states of the lower blocks (e.g.  $\tilde{W}_f(k)$  and  $\tilde{W}_g(k)$ ) are bounded even if PE does not hold. Thus, the modified tuning algorithms guarantee SSP of the weight tuning blocks, so that the closed-loop system is SSP. Therefore, internal stability can be guaranteed even in the absence of PE. Similar analysis can be extended to multilayer case as well.*

## 8.6 CONCLUSIONS

*In Chapter 7 we showed how to design NN controllers that use discrete-time updates for Brunovsky form systems having known control influence coefficient. If one samples a continuous-time system, however, the discrete-time system is generally of the form  $x(k+1) = f(x(k)) + g(x(k))u(k)$ , with both  $f(x(k))$  and  $g(x(k))$  unknown. In this chapter we showed how to use two neural networks to estimate both  $f(\cdot)$  and  $g(\cdot)$ . This causes great problems, for to keep the control signals bounded, one must guarantee that the NN estimate for  $g(x(k))$  never goes to zero. This was accomplished by using a switching sort of tuning law for the NN that estimates  $g(\cdot)$ . Two families of controllers were derived—one using linear-in-the-parameter NN and another using multilayer NN.*

## 8.7 REFERENCES

- Aström, K. J., and B. Wittenmark, *Adaptive Control*, Addison-Wesley Company, Reading, Massachusetts, 1989.
- Chen, F.-C., and H.K. Khalil, “Adaptive control of nonlinear discrete-time systems using neural networks,” *IEEE Trans. on Automatic Control*, vol. 40, no. 5, pp. 791-801, May 1995.
- Cybenko, G., “Approximations by superpositions of sigmoidal activation function”, *Math, Contr., Signals, Syst*, vol. 2, no. 4, pp. 303-314, February 1989.
- Goodwin, G.C., and K.S. Sin, *Adaptive Filtering, Prediction, and Control*, Prentice-Hall, Englewood Cliffs, NJ, 1984.
- Jagannathan, S., and F.L. Lewis, “Multilayer discrete-time neural net controller with guaranteed performance”, *IEEE Trans. on Neural Networks*, vol. 7, no. 1, pp. 107-130, January 1996a.

- Jagannathan, S., and F.L. Lewis, "Discrete-time neural net controller for a class of nonlinear dynamical systems," IEEE Trans. Automat. Control, vol. 41, no. 11, pp. 1693-1699, Nov. 1996b.*
- Jagannathan, S., and F.L. Lewis, "Robust implicit self-tuning regulator: convergence and stability," Automatica, vol. 32, no. 12, pp. 1629-1644, 1996c.*
- Jagannathan, S., "Discrete-time adaptive control of feedback linearizable nonlinear systems", Proc. IEEE Conf. Decision and Control, pp. 4747-4751, Kobe, December 1996a.*
- Jagannathan, S., "Adaptive control of unknown feedback linearizable systems in discrete-time using neural networks", Proc. of the IEEE Conf. on Robotics and Automation, vol. 1, pp.258-263, April 96a.*
- Kanellakopoulos, I., P.V. Kokotovic, and A.S. Morse, "Systematic design of adaptive controllers for feedback linearizable systems", IEEE Trans. on Automatic Control, vol. 36, pp. 1241-1253, 1991.*
- Liu, C.C., and F. Chen, "Adaptive control of nonlinear continuous systems Using neural networks-general relative degree and MIMO cases", Int. Jounal of Control, vol.58, pp.317-335, 1991.*
- Narendra, K.S., and A.M. Annaswamy, "A new adaptive law for robust adaptation without persistent excitation," IEEE Trans. on Automatic Control, vol.AC-32, no.2, pp. 134-145, February 1987.*
- Narendra, K.S., and K.S. Parthasarathy, "Identification and control of dynamical systems using neural networks," IEEE Trans. on Neural Networks, vol. 1, no.1, pp. 4-27, March 1990.*
- Mpitsos, G.J., and R.M. Burton, Jr, "Convergence and divergence in neural networks: Processing of chaos and biological analogy", Neural Networks, vol. 5, pp. 605-625, 1992.*
- Polycarpou, M.M., and P.A. Ioannou, "Identification and control using neural network models: design and stability analysis", Dept. of Elec. Engg, Tech Report.91-09-01, September 1991.*
- Rovithakis, G.A., and M.C. Christodoulou, "Adaptive control of unknown plants using dynamical neural networks", IEEE Trans. on Systems, Man, and Cybernetics, vol.24, no. 3, pp.400-411, March 1994.*
- Sadegh, N., "A perceptron network for functional identification and control of non-linear systems", IEEE Trans. on Neural Networks, vol. 4, no. 6, pp. 982-988, November 1993.*
- Sanner, R.M., and J.-J. Slotine, "Gaussian networks for direct adaptive control", IEEE Trans. on Neural Networks, vol. 3, no. 6, pp. 837-863, November 1992.*
- Slotine, J.J., and W. Li, Applied Nonlinear Control, Prentice Hall Inc, Englewood Cliffs, NJ, 1991.*

Sontag, E., "Feedback stabilization using two-hidden-layer nets", *IEEE Trans. on Neural Networks*, vol. 3, no. 6, pp. 981-990, November 1992.

Sussmann, H.J., "Uniqueness of the weights for minimal feedforward nets with given input-output map", *Neural Networks*, vol. 5, pp. 589-593, 1992.

Yeşildirek, A., and F.L. Lewis, "Feedback linearization using neural networks", *Automatica*, vol. 31, no. 11, pp. 1659-1664, November 1995.

## 8.8 PROBLEMS

### Section 8.2

**Problem 8.2-1 : One-Layer Neural Network.** Consider the system described by

$$x(k+1) = f(x(k), x(k-1)) + 2u(k), \quad (8.8.1)$$

where  $f(x(k), x(k-1)) = \frac{x(k)x(k-1)[x(k)+1.0]}{1+x^2(k)+x(k-1)}$ . Design a one-layer neural network controller with and/or without learning phase and by using the developed delta rule-based weight tuning algorithm and appropriately choosing the adaptation gains. Repeat the problem by using the modified update weight tuning method.

**Problem 8.2-2 : One-layer Neural Network.** For the system described by

$$x(k+1) = f(x(k), x(k-1)) + 2u(k), \quad (8.8.2)$$

where  $f(x(k), x(k-1)) = \frac{x(k)}{1+x(k)} + u^3(k)$ . Design a one-layer neural network controller with and/or without learning phase and by using the developed delta rule-based weight tuning algorithm and appropriately choosing the adaptation gains. Repeat the problem by using the modified update weight tuning method.

### Section 8.4

**Problem 8.4-1 : Stability and Convergence for an  $n$ -layer NN.** Assume the hypotheses presented for three-layer NN and use the weight updates presented in (8.4.1)-(8.4.4) and extend the stability and boundedness of tracking error and error in weight updates for  $n$ -layer NN.

**Problem 8.4-2 : Stability and Convergence for an  $n$ -layer NN.** Assume the hypotheses presented for three-layer NN and use the weight updates presented in (8.4.54)-(8.4.57) with projection algorithm and show the stability and boundedness of tracking error and error in weight updates for  $n$ -layer NN.

**Problem 8.4-3 : The  $n$ -Layer NN for Control.** Perform a MATLAB simulation using a  $n$ -layer NN for the Example 8.4.1. Show the advantage of adding more layers by picking less number of hidden-layer neurons and layers more than three. Use both delta-rule and projection algorithm.

### Section 8.5

**Problem 8.5-1 : Passivity Properties for an  $n$ -layer NN.** *Show the passivity properties of the input and hidden layers for a  $n$ -layer neural network using delta rule-based weight tuning.*

**Problem 8.5-2 : Passivity Properties for an  $n$ -layer NN Using Modified Weight Tuning.** *Show the passivity properties of the input and hidden layers for a  $n$ -layer neural network using improved weight tuning.*

## Chapter 9

# State Estimation Using Discrete-Time Neural Networks

*System identification is the process of determining a dynamical model for an unknown plant that can be used for feedback control purposes. The state estimation problem involves determining the unknown internal state of a dynamical plant. System identification provides one technique for estimating the states. The area of system identification has received much attention over the past two decades. It is now a mature field, and many powerful methods are at the disposal of control engineers. On-line system identification methods used to date are mostly based on recursive methods such as least squares (Ren and Kumar 1994). However, most of these techniques are for models that are linear in the parameters. In order to relax the linearity in the parameters assumption, NN are being widely employed for system identification since these networks learn complex mappings from a set of examples. Due to their approximation properties as well as the inherent adaptation features of these networks, NN present a potentially appealing alternative to modeling of nonlinear systems. Furthermore, from a practical perspective, the massive parallelism and fast adaptability of neural network implementations provide additional incentives for further investigation.*

*Several approaches have been presented in system identification without using NN (Ljung and Söderström 1983, Goodwin and Sin 1984, Landau 1979, Narendra and Annaswamy 1989) and using NN (Narendra and Parthasarathy 1990, Polycarpou and Ioannou 1991, Sadegh 1993, Sira-Ramirez and Zak 1991, Jagannathan and Lewis 1996). However, most of the schemes for system identification using multilayer NN have been demonstrated through empirical studies, or convergence of the output error is shown in ideal conditions (Narendra and Parthasarathy 1990). Others (Polycarpou and Ioannou 1991, Sadegh 1993) have shown the stability of the overall system or convergence of the output error using a linearity in the parameters assumption.*

*Important structures in neural network control are the recurrent or dynamic*

*neural networks (Rovithakis and Christodoulou 1994), in which the NN has its own internal dynamics (Narendra and Partha Sarathy 1990). Most identification schemes using both multilayer feedforward and recurrent NN include identifier structures which do not guarantee the boundedness of the identification error of the system in nonideal conditions even in an open-loop configuration. In addition, convergence proofs are only given under some stringent conditions such as initialization of the NN with stabilizing weights in the neighborhood of a global minimum, which is a very unrealistic assumption since stabilizing weights are difficult to find. With improper initialization, many papers report undesirable behavior. Furthermore, the backpropagation tuning algorithm, often used for system identification, requires the evaluation of sensitivity derivatives along the network signal paths which is usually impossible in closed-loop uncertain systems since the required Jacobians are unknown.*

*The main objective of this chapter is to provide techniques for estimating the internal states of unknown nonlinear systems using dynamical NN (Jagannathan and Lewis 1996). This will be achieved by identifying the unknown nonlinear dynamics of the plant. It is noted that solving the state estimation problem involves only a small subset of the topic of system identification. In order to relax the linearity in the unknown parameters assumption and show the boundedness of the state estimation errors and identification errors using multilayer NN, novel learning schemes are investigated to identify four classes of discrete-time systems that are commonly used in the literature. Here, weights of the multilayer NN are tuned on-line with no preliminary off-line learning phase needed. The weight tuning mechanisms guarantee convergence of the NN weights when initialized at zero, even though there do not exist ‘ideal weights’ such that the NN perfectly reconstructs a certain required function that approximates the desired nonlinear system. The identifier structure ensures good performance (bounded identification error and weight estimates) as shown through a Lyapunov’s approach, so that convergence to a stable solution is guaranteed with mild assumptions.*

*The identifier is composed of a neural network incorporated into a dynamical system, where the structure comes from error notions standard in the system identification and control literature. It is shown that the weight tuning algorithm using the delta rule in each layer yields a passive neural network; this guarantees the boundedness of all the signals in the system. It is found that the maximum permissible rate for the developed tuning algorithm decreases as the NN size increases; this is a major drawback. A projection algorithm, as used in adaptive control (Jagannathan 1994), is shown to easily correct the problem. The convergence analysis presented using a three-layer NN is extended to a general  $n$ -layer case. For more details see (Jagannathan 1994).*

*Once a NN has been tuned to identify a dynamical system, it is of great interest to determine the structural information contained in the learned NN weights. Structural information can be very useful in controller design. This can be accomplished in many ways, including the Volterra series approach in the work of Billings and coworkers (Billings et al. 1992, Fung et al. 1997) which determines a generalized frequency response function (GFRF) of a given NN.*

## 9.1 IDENTIFICATION OF NONLINEAR DYNAMICAL SYSTEMS

The ability of neural networks to approximate large classes of nonlinear functions makes them prime candidates for the identification of nonlinear plants. Four models representing multi-input/multi-output (MIMO) nonlinear systems are in common use (Landau 1979, 1993; Narendra and Parthasarathy 1990). These four models are in nonlinear autoregressive-moving average (ARMA) form, and cover a very large range of plants. They are therefore considered here. These four nonlinear canonical forms are:

$$\begin{aligned} x(k+1) = & \sum_{i=0}^{n-1} \alpha_i x(k-i) + g(u(k), u(k-1), \dots, u(k-m+1)) \\ & + d(k), \end{aligned} \quad (9.1.1)$$

$$\begin{aligned} x(k+1) = & f(x(k), x(k-1), \dots, x(k-n+1)) + \sum_{i=0}^{m-1} \beta_i u(k-i) \\ & + d(k), \end{aligned} \quad (9.1.2)$$

$$\begin{aligned} x(k+1) = & f(x(k), x(k-1), \dots, x(k-n+1)) \\ & + g(u(k), u(k-1), \dots, u(k-m+1)) + d(k), \end{aligned} \quad (9.1.3)$$

$$\begin{aligned} x(k+1) = & f(x(k), x(k-1), \dots, x(k-n+1); u(k), u(k-1), \dots, u(k-m+1)) \\ & + d(k), \end{aligned} \quad (9.1.4)$$

with unknown nonlinear functions  $f(\cdot) \in \mathbb{R}^n$ ,  $g(\cdot) \in \mathbb{R}^n$ , state  $x(k) \in \mathbb{R}^n$ , coefficients  $\alpha_i \in \mathbb{R}^{nxn}$ ,  $\beta_i \in \mathbb{R}^{nxn}$ , control  $u(k) \in \mathbb{R}^n$ , and  $d(k) \in \mathbb{R}^n$  an unknown disturbance vector acting on the system at the instant  $k$  with  $\|d(k)\| \leq d_M$  a known constant. If Model I is selected then  $\alpha_i \in \mathbb{R}^{nxn}$  are chosen such that the roots of the polynomial  $z^n - \alpha_0 z^{n-1} - \dots - \alpha_{n-1} = 0$  lie in the interior of the unit circle. The four models are shown in Fig. 9.1.1.

## 9.2 IDENTIFIER DYNAMICS FOR MIMO SYSTEMS

Consider MIMO discrete-time nonlinear systems given in multivariable form as one of the models (9.1.1) through (9.1.4). The problem of identification consists of setting up a suitably parametrized identification model and adjusting the parameters of the model so that when subjected to the same input  $u(k)$  as the plant, it produces an output  $\hat{x}(k)$  that is close to the actual  $x(k)$ . Taking the structure of the identifier the same as that of the plant, the plants given in (9.1.1) through (9.1.4) are identified respectively by the following estimators:

$$\begin{aligned} x(k+1) = & \sum_{i=0}^{n-1} \alpha_i x(k-i) + \hat{g}(u(k), u(k-1), \dots, u(k-m+1)) \\ & + d(k), \end{aligned} \quad (9.2.1)$$

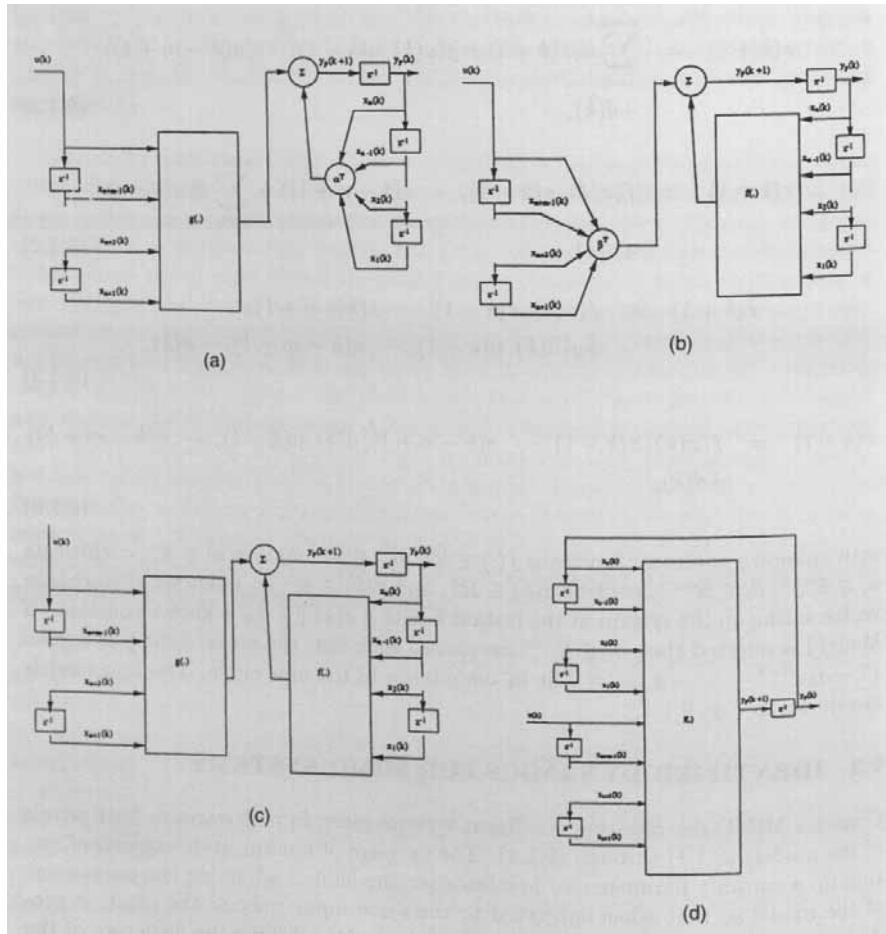


Figure 9.1.1: Multilayer neural network identifier models.

$$\begin{aligned} x(k+1) &= \hat{f}(x(k), x(k-1), \dots, x(k-n+1)) \\ &\quad + \sum_{i=0}^{m-1} \beta_i u(k-i) + d(k), \end{aligned} \quad (9.2.2)$$

$$\begin{aligned} x(k+1) &= \hat{f}(x(k), x(k-1), \dots, x(k-n+1)) \\ &\quad + \hat{g}(u(k), u(k-1), \dots, u(k-m+1)) + d(k), \end{aligned} \quad (9.2.3)$$

$$\begin{aligned} x(k+1) &= \hat{f}(x(k), x(k-1), \dots, x(k-n+1); u(k), u(k-1), \dots, u(k-m+1)) \\ &\quad + d(k), \end{aligned} \quad (9.2.4)$$

with  $\hat{f}(\cdot)$  an estimate of  $f(\cdot)$  and  $\hat{g}(\cdot)$  an estimate of  $g(\cdot)$ .

In this work, NN will be used to provide the functional estimates  $\hat{f}(\cdot)$  and  $\hat{g}(\cdot)$ . Due to the universal approximation properties of NN there exist static NN that approximate  $f(\cdot)$  and  $g(\cdot)$ . When embedded in the dynamics (9.2.1)- (9.2.4), the result is a dynamic or recurrent NN estimator that, for the same initial conditions, produces the same output as the plant for any specified input. The identification procedure consists of adjusting the weights of the neural networks in the model using the weight updates presented in the next section to guarantee internal stability and closeness of  $\hat{x}(k)$  and  $x(k)$ .

Define the identification error as

$$e(k) = x(k) - \hat{x}(k). \quad (9.2.5)$$

Then the error dynamics of (9.1.1) through (9.1.4) and (9.2.5) can be expressed respectively as

$$e(k+1) = \tilde{g}(\cdot) + d(k), \quad (9.2.6)$$

$$e(k+1) = \tilde{f}(\cdot) + d(k), \quad (9.2.7)$$

$$e(k+1) = \tilde{f}(\cdot) + \tilde{g}(\cdot) + d(k), \quad (9.2.8)$$

$$e(k+1) = \tilde{f}(\cdot) + d(k), \quad (9.2.9)$$

where the functional estimation errors are given by

$$\tilde{f}(\cdot) = f(\cdot) - \hat{f}(\cdot), \quad (9.2.10)$$

and

$$\tilde{g}(\cdot) = g(\cdot) - \hat{g}(\cdot). \quad (9.2.11)$$

These are error systems wherein the identification error is driven by the functional estimation error.

In the remainder of this chapter, Equations (9.2.6) through (9.2.9) are used to focus on selecting NN tuning algorithms that guarantee the stability of the identification error  $e(k)$ . Note that (9.2.7) and (9.2.9) are similar except the nonlinear function in (9.2.9) is a more general function of the state vector, the input vector, and their delayed values. Denote by  $X(k)$  the appropriate argument of  $\tilde{f}(\cdot)$ , which

consists of  $x(k)$  and previous values in (9.2.7), and also includes  $u(k)$  and previous values in (9.2.9). Then both equations can be represented as

$$e(k+1) = \tilde{f}(X(k)) + d(k). \quad (9.2.12)$$

This is the error system resulting from either identifier (9.2.2) or (9.2.4).

Equations (9.2.6) and (9.2.8) are also similar except that  $\tilde{f}$  is missing in the former. For analysis purposes they are both taken as the more general system

$$e(k+1) = \tilde{f}(X(k)) + \tilde{g}(U(k)) + d(k), \quad (9.2.13)$$

where  $U(k)$  denotes  $u(k)$  and its previous values. This is the error system resulting from either identifier (9.2.1) or (9.2.3). Subsequent analysis considers these two forms of error system.

### 9.3 MULTILAYER NEURAL NETWORK IDENTIFIER DESIGN

In this section, multilayer NN are used to provide the approximations  $\hat{f}(\cdot), \hat{g}(\cdot)$  in the identifier systems. Stability analysis is performed by Lyapunov's direct method for multilayer NN weight tuning algorithms consisting of a delta rule in each layer. Note that one NN is needed to approximate  $\tilde{f}(\cdot)$  in the error system (9.2.12) whereas two NN are required, one for  $f(\cdot)$  and one for  $g(\cdot)$ , for (9.2.13).

Assume that there exist some constant ideal weights  $W_{1f}, W_{2f}, W_{1g}, W_{2g}$  and  $W_{3f}, W_{3g}$  for three-layer NN so that the nonlinear functions  $f(\cdot)$  in (9.2.12) and (9.2.13) and  $g(\cdot)$  in (9.2.13) can be written on a compact set  $S$  as

$$f(X(k)) = W_{3f}^T \varphi_{3f}[W_{2f}^T \varphi_{2f}[W_{1f}^T \varphi_{1f}(X(k))]] + \epsilon_f(k), \quad (9.3.1)$$

$$g(U(k)) = W_{3g}^T \varphi_{3g}[W_{2g}^T \varphi_{2g}[W_{1g}^T \varphi_{1g}(U(k))]] + \epsilon_g(k), \quad (9.3.2)$$

where the functional estimation errors satisfy  $\|\epsilon_f(k)\| < \epsilon_{Nf}$  and  $\|\epsilon_g(k)\| < \epsilon_{Ng}$ , with the bounding constants  $\epsilon_{Nf}$  and  $\epsilon_{Ng}$  known. Unless the net is 'minimal', the 'ideal' weights may not be unique (Sontag 1992) and (Sussmann 1992). The best weights may then be defined as those which minimize the supremum norm over  $S$  of  $\epsilon(k)$ . This issue is not a major concern here, as it is needed to know only the existence of such ideal weights; their actual values or uniqueness are of no concern. This assumption is similar to Erzberger's assumptions in the linear-in-the-parameters adaptive control. The major difference is that, while Erzberger's assumptions often do not hold, the approximation properties of NN guarantee that the ideal weights do always exist if  $f(x), g(x)$  are continuous over a compact set (Cybenko 1989, Park and Sandberg 1991).

**Assumption 9.3.1 (Bounded NN Weights)** The ideal weights are bounded by known positive values so that  $\|W_{1f}\| \leq W_{1fmax}$ ,  $\|W_{2f}\| \leq W_{2fmax}$ , and  $\|W_{3f}\| \leq W_{3fmax}$ . Similarly,  $\|W_{1g}\| \leq W_{1gmax}$ ,  $\|W_{2g}\| \leq W_{2gmax}$ , and  $\|W_{3g}\| \leq W_{3gmax}$ .

#### 9.3.1 Structure of the NN Controller and Error System Dynamics

Define the NN functional estimates for  $f(\cdot)$  and  $g(\cdot)$  by

$$\hat{f}(X(k)) = \hat{W}_{3f}^T(k) \varphi_{3f}(\hat{W}_{2f}^T(k) \varphi_{2f}(\hat{W}_{1f}^T(k) \varphi_{1f}(X(k)))), \quad (9.3.3)$$

$$\hat{g}(U(k)) = \hat{W}_{3g}^T(k) \varphi_{3g}(\hat{W}_{2g}^T(k) \varphi_{2g}(\hat{W}_{1g}^T(k) \varphi_{1g}(U(k)))), \quad (9.3.4)$$

with  $\hat{W}_{3f}(k), \hat{W}_{2f}(k), \hat{W}_{1f}(k), \hat{W}_{3g}(k), \hat{W}_{2g}(k), \hat{W}_{1g}(k)$  the current values of the weights as given by the tuning algorithms to be derived. The estimates of the input layer activation function outputs are denoted  $\hat{\varphi}_{1f}(k) = \varphi_{1f}(X(k))$  and  $\hat{\varphi}_{1g}(k) = \varphi_{1g}(U(k))$ . Then the estimates of the activation function outputs of the hidden and output layers are denoted by

$$\hat{\varphi}_{(i+1)f}(k) = \varphi(\hat{W}_{if}^T \hat{\varphi}_{if}(k)); \quad i = 1, \dots, n, \quad (9.3.5)$$

$$\hat{\varphi}_{(i+1)g}(k) = \varphi(\hat{W}_{ig}^T \hat{\varphi}_{ig}(k)); \quad i = 1, \dots, n, \quad (9.3.6)$$

where  $n = 3$ .

Since the standard NN activation functions, including sigmoids, tanh, RBF etc., are bounded by known positive values for a given trajectory, one has  $\|\hat{\varphi}_{1f}(k)\| \leq \varphi_{1fmax}$ ,  $\|\hat{\varphi}_{2f}(k)\| \leq \varphi_{2fmax}$ , and  $\|\hat{\varphi}_{3f}(k)\| \leq \varphi_{3fmax}$ ,  $\|\hat{\varphi}_{1g}(k)\| \leq \varphi_{1gmax}$ ,  $\|\hat{\varphi}_{2g}(k)\| \leq \varphi_{2gmax}$ , and  $\|\hat{\varphi}_{3g}(k)\| \leq \varphi_{3gmax}$ .

The NN weight estimation errors are given by

$$\tilde{W}_{3f}(k) = W_{3f} - \hat{W}_{3f}(k), \quad \tilde{W}_{2f}(k) = W_{2f} - \hat{W}_{2f}(k), \quad \tilde{W}_{1f}(k) = W_{1f} - \hat{W}_{1f}(k), \quad (9.3.7)$$

and

$$\tilde{W}_{3g}(k) = W_{3g} - \hat{W}_{3g}(k), \quad \tilde{W}_{2g}(k) = W_{2g} - \hat{W}_{2g}(k), \quad \tilde{W}_{1g}(k) = W_{1g} - \hat{W}_{1g}(k). \quad (9.3.8)$$

The net layer output errors are defined as

$$\tilde{\varphi}_{2f}(k) = \varphi_{2f} - \hat{\varphi}_{2f}(k), \quad \tilde{\varphi}_{3f}(k) = \varphi_{3f} - \hat{\varphi}_{3f}(k). \quad (9.3.9)$$

and

$$\tilde{\varphi}_{2g}(k) = \varphi_{2g} - \hat{\varphi}_{2g}(k), \quad \tilde{\varphi}_{3g}(k) = \varphi_{3g} - \hat{\varphi}_{3g}(k). \quad (9.3.10)$$

Using the functional estimate of  $f(\cdot)$  and  $g(\cdot)$  given in (9.3.3) and (9.3.4), the error Equations (9.2.12) and (9.2.13) can be expressed as

$$e(k+1) = e_f(k) + \delta(k), \quad (9.3.11)$$

and

$$e(k+1) = e_f(k) + e_g(k) + \delta(k), \quad (9.3.12)$$

where one defines

$$e_f(k) \equiv \tilde{W}_{3f}^T(k) \hat{\varphi}_{3f}(k), \quad (9.3.13)$$

$$e_g(k) \equiv \tilde{W}_{3g}^T(k) \hat{\varphi}_{3g}(k), \quad (9.3.14)$$

and

$$\delta(k) \equiv W_{3f}^T \tilde{\varphi}_{3f}(k) + \epsilon_f(k) + d(k), \quad (9.3.15)$$

$$\delta(k) \equiv W_{3f}^T \tilde{\varphi}_{3f}(k) + W_{3g}^T \tilde{\varphi}_{3g}(k) + \epsilon_f(k) + \epsilon_g(k) + d(k). \quad (9.3.16)$$

The proposed controller structure is shown in Fig. 9.3.1. The next step is to determine the weight updates so that the tracking performance of the identification error dynamics is guaranteed.

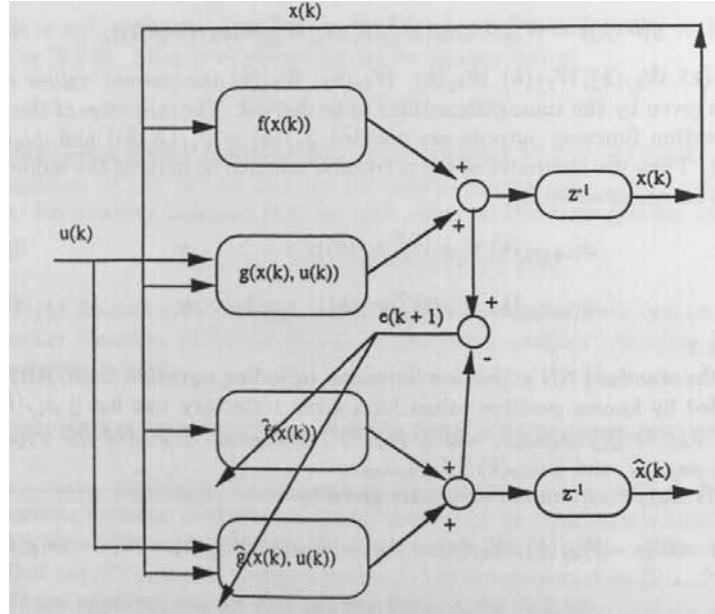


Figure 9.3.1: Multilayer neural network identifier structure.

### 9.3.2 Three-Layer Neural Network Weight Updates

A family of NN weight tuning paradigms that guarantee the stability of the error systems (9.3.11) and (9.3.12) are presented in this section. It is required to demonstrate that the identification error  $e(k)$  is suitably small and that the NN weight estimates in (9.3.3), (9.3.4) remain bounded, given a bounded input  $u(k)$ . The next result provides NN weight tuning algorithms that guarantee stable identification. Persistence of excitation (PE) for a multilayer discrete-time NN (Jagannathan and Lewis 1996) is defined during the proof.

#### Theorem 9.3.1 (Three-Layer NN Identifier) :

Given an unknown system in one of the four forms (9.1.1) through (9.1.4), select the estimator from the respective form (9.2.1) through (9.2.4) and let  $\hat{f}(\cdot)$ , and  $\hat{g}(\cdot)$  if required, be given by NN as in (9.3.3) and (9.3.4). Let the NN functional reconstruction error and the disturbance bounds,  $\epsilon_{Nf}, \epsilon_{Ng}, d_M$ , respectively, be known constants. Let NN weight tuning be provided for the input and hidden layers as

$$\hat{W}_{1f}(k+1) = \hat{W}_{1f}(k) - \alpha_{1f}\hat{\varphi}_{1f}(k)[\hat{y}_{1f}(k) + B_{1f}e(k)]^T, \quad (9.3.17)$$

$$\hat{W}_{2f}(k+1) = \hat{W}_{2f}(k) - \alpha_{2f}\hat{\varphi}_{2f}(k)[\hat{y}_{2f}(k) + B_{2f}e(k)]^T, \quad (9.3.18)$$

$$\hat{W}_{1g}(k+1) = \hat{W}_{1g}(k) - \alpha_{1g}\hat{\varphi}_{1g}(k)[\hat{y}_{1g}(k) + B_{1g}e(k)]^T, \quad (9.3.19)$$

$$\hat{W}_{2g}(k+1) = \hat{W}_{2g}(k) - \alpha_{2g}\hat{\varphi}_{2g}(k)[\hat{y}_{2g}(k) + B_{2g}e(k)]^T, \quad (9.3.20)$$

where  $\hat{y}_{if}(k) = \hat{W}_{if}^T(k)\hat{\varphi}_{if}(k); \hat{y}_{ig}(k) = \hat{W}_{ig}^T(k)\hat{\varphi}_{ig}(k); i = 1, 2$ , and

$$\|B_{if}\| \leq \kappa_{if}, i = 1, 2 \quad (9.3.21)$$

$$\|B_{ig}\| \leq \kappa_{ig}, i = 1, 2. \quad (9.3.22)$$

Let the weight tuning for the output layer be given by

$$\hat{W}_{3f}(k+1) = \hat{W}_{3f}(k) + \alpha_{3f} \hat{\varphi}_{3f}(k) e^T(k+1) \quad (9.3.23)$$

$$\hat{W}_{3g}(k+1) = \hat{W}_{3g}(k) + \alpha_{3g} \hat{\varphi}_{3g}(k) e^T(k+1) \quad (9.3.24)$$

with  $\alpha_{if} > 0, \alpha_{ig} > 0, i = 1, 2, 3$  denoting constant learning rate parameters or adaptation gains.

Let the output vectors of the input, hidden, and output layers,  $\hat{\varphi}_{1f}(k), \hat{\varphi}_{2f}(k)$ , and  $\hat{\varphi}_{3f}(k), \hat{\varphi}_{1g}(k), \hat{\varphi}_{2g}(k)$ , and  $\hat{\varphi}_{3g}(k)$  be persistently exciting. Then the identification error  $e(k)$  and the errors in the weight estimates,  $\tilde{W}_{1f}(k), \tilde{W}_{2f}(k), \tilde{W}_{3f}(k), \tilde{W}_{1g}(k), \tilde{W}_{2g}(k)$ , and  $\tilde{W}_{3g}(k)$ , are UUB, with the bounds on  $e(k)$  specifically given by (9.3.37), provided the following conditions hold:

Condition (1);

$$\alpha_{if} \|\hat{\varphi}_{if}(k)\|^2 < \begin{cases} 2 & i = 1, 2, \\ 1 & i = 3, \end{cases} \quad (9.3.25)$$

for the error system (9.3.11) and (9.3.25) plus

$$\begin{aligned} \alpha_{ig} \|\hat{\varphi}_{ig}(k)\|^2 &< 2, i = 1, 2, \\ (\alpha_{if} \|\hat{\varphi}_{if}(k)\|^2 + \alpha_{ig} \|\hat{\varphi}_{ig}(k)\|^2) &< 1, \end{aligned} \quad (9.3.26)$$

for the error system (9.3.12).

Condition (2);

$$c_0 < 1, \quad (9.3.27)$$

where  $c_0$  is given for the error system (9.3.11) as

$$c_0 = \sum_{i=1}^2 \frac{\kappa_{if}^2}{(2 - \alpha_{if} \|\hat{\varphi}_{if}(k)\|^2)}, \quad (9.3.28)$$

and for the error system (9.3.12) as

$$c_0 = \sum_{i=1}^2 \left[ \frac{\kappa_{if}^2}{(2 - \alpha_{if} \|\hat{\varphi}_{if}(k)\|^2)} + \frac{\kappa_{ig}^2}{(2 - \alpha_{ig} \|\hat{\varphi}_{ig}(k)\|^2)} \right]. \quad (9.3.29)$$

Note: The parameters  $\alpha_{if}, \alpha_{ig}; \forall i = 1, \dots, 3$  and  $c_0$  depend upon the trajectory. Given a trajectory, the constants  $\alpha_{if}, \alpha_{ig}; \forall i = 1, \dots, 3$  with  $c_0, c_1$  and  $c_2$  can be determined.

Proof:

Let  $\Omega$  and  $\Omega_U$  be subsets of  $\Re^n$  and  $\Re^m$  respectively such that  $e(0) \in \Omega$  and  $\tilde{W}_i(0) \in \Gamma_U, \forall i = 1, \dots, 3$  and the NN approximation holds. Using the Lyapunov function candidate

$$J = e^T(k) e(k) + \sum_{i=1}^3 \left[ \frac{1}{\alpha_{if}} \text{tr}(\tilde{W}_{if}(k) \tilde{W}_{if}(k)) \right], \quad (9.3.30)$$

define

$$l^2 = \sup_{(e, \tilde{W}_i) \in \Omega \times \Omega_U} [e^T(k) e(k) + \sum_{i=1}^3 \left[ \frac{1}{\alpha_{if}} \text{tr}(\tilde{W}_{if}^T(k) \tilde{W}_{if}(k)) \right]]. \quad (9.3.31)$$

Consider  $\Delta J$  on the set  $\chi = [(e, \tilde{W}_i) : J(e, \tilde{W}_i) \leq l^2]$ .

Error System (9.3.11): Define the Lyapunov function candidate as in (9.3.30) and whose first difference,  $\Delta J \forall (e, \tilde{W}_i) \in \chi$ , is given by

$$\Delta J = e^T(k+1)e(k+1) - e^T(k)e(k) + \sum_{i=1}^3 \left[ \frac{1}{\alpha_{if}} \text{tr}[\tilde{W}_{if}(k+1)\tilde{W}_{if}(k+1) - \tilde{W}_{if}^T(k)\tilde{W}_{if}(k)] \right]. \quad (9.3.32)$$

Substituting the Equations (9.3.17), (9.3.18), and (9.3.23) yields,

$$\begin{aligned} \Delta J &\leq -(1-c_0)[\|e(k)\|^2 - 2\frac{c_1}{(1-c_0)}\|e(k)\| - \frac{c_2}{(1-c_0)} \\ &\quad - [1 - \alpha_{3f}\hat{\varphi}_{3f}^T(k)\hat{\varphi}_{3f}(k)]\|e_f(k) - \frac{\alpha_{3f}\hat{\varphi}_{3f}(k)\hat{\varphi}_{3f}(k)\delta(k)}{[1 - \alpha_{3f}\hat{\varphi}_{3f}^T(k)\hat{\varphi}_{3f}(k)]}\|^2 \\ &\quad - \sum_{i=1}^2 (2 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))\|\tilde{W}_{if}(k)\hat{\varphi}_{if}(k) - \frac{(1 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))}{(2 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))} \\ &\quad (W_{if}^T\hat{\varphi}_{if}(k) + B_{if}e(k))\|^2 \end{aligned} \quad (9.3.33)$$

where

$$c_1 = \sum_{i=1}^2 \frac{\kappa_{if} \|\hat{\varphi}_{if}(k)\| W_{ifmax}}{(2 - \alpha_{if} \|\hat{\varphi}_{if}(k)\|^2)} \quad (9.3.34)$$

and

$$c_2 = [\frac{\delta_{max}^2}{[1 - (\alpha_{3f} \|\hat{\varphi}_{3f}(k)\|^2 + \alpha_{3g} \|\hat{\varphi}_{3g}(k)\|^2)]} + \sum_{i=1}^2 \frac{\|\hat{\varphi}_{if}(k)\|^2 W_{ifmax}^2}{(2 - \alpha_{if} \|\hat{\varphi}_{if}(k)\|^2)}] \quad (9.3.35)$$

with

$$\delta_{max} = W_{3fmax}\hat{\varphi}_{3fmax} + \epsilon_{Nfmax} + d_M. \quad (9.3.36)$$

Since  $c_0, c_1, c_2$  are positive constants,  $\Delta J \leq 0$  as long as

$$\|e(k)\| > \frac{1}{(1-c_0)}[c_1 + \sqrt{c_1^2 + c_2(1-c_0)}]. \quad (9.3.37)$$

$|\sum_{k=k_0}^{\infty}| = |J(\infty) - J(0)| < \infty$  since  $\Delta J \leq 0$  as long as (9.3.25), (9.3.27) and (9.3.37) hold. The definition of  $J$  and inequality (9.3.37) imply that every initial condition in the set  $\chi$  will evolve entirely in  $\chi$ . That is, whenever the identification error  $e(k)$  is outside the region defined (9.3.37),  $J(e, \tilde{W}_i)$  will decrease. This further implies that  $\|e(k)\|$  will not increase and will remain in  $\chi$ . This demonstrates that the identification error  $e(k)$  is bounded for all  $k \geq 0$  and it remains to show that the weight estimates  $\hat{W}_{1f}(k), \hat{W}_{2f}(k), \hat{W}_{3f}(k), \hat{W}_{1g}(k), \hat{W}_{2g}(k)$ , and  $\hat{W}_{3g}(k)$  or equivalently  $\tilde{W}_{1f}(k), \tilde{W}_{2f}(k), \tilde{W}_{3f}(k), \tilde{W}_{1g}(k), \tilde{W}_{2g}(k)$ , and  $\tilde{W}_{3g}(k)$ .

Error System (9.3.12): Let  $\Omega$  and  $\Omega_U$  be subsets of  $\Re^n$  and  $\Re^m$  respectively such that  $e(0) \in \Omega$  and  $\tilde{W}_i(0) \in \Omega_U, \forall i = 1, \dots, 3$  for both  $f$  and  $g$ , (here  $m = 6$ ) and the NN approximation holds. For the error system (9.3.12), using the Lyapunov function candidate

$$J = e^T(k)e(k) + \sum_{i=1}^3 \left[ \frac{1}{\alpha_{if}} \text{tr}(\tilde{W}_{if}^T(k)\tilde{W}_{if}(k)) + \frac{1}{\alpha_{ig}} \text{tr}(\tilde{W}_{ig}^T(k)\tilde{W}_{ig}(k)) \right], \quad (9.3.38)$$

define

$$l^2 = \sup_{(e, \tilde{W}_i) \in \Omega \times \Omega_U} [e^T(k)e(k) + \sum_{i=1}^3 \left[ \frac{1}{\alpha_{if}} \text{tr}(\tilde{W}_{if}^T(k)\tilde{W}_{if}(k)) + \frac{1}{\alpha_{ig}} \text{tr}(\tilde{W}_{ig}^T(k)\tilde{W}_{ig}(k)) \right]]. \quad (9.3.39)$$

Consider  $\Delta J$  on the set  $\chi = ((e, \tilde{W}_i); J(e, \tilde{W}) \leq l^2)$ . The first difference,  $\Delta J \forall (e, \tilde{W}) \in \chi$ , is

$$\begin{aligned} \Delta J &\leq e^T(k+1)e(k+1) - e^T(k)e(k) \\ &+ \sum_{i=1}^3 \left[ \frac{1}{\alpha_{if}} \text{tr}[\tilde{W}_{if}^T(k+1)\tilde{W}_{if}(k+1)\tilde{W}_{if}^T(k)\tilde{W}_{if}(k)] \right] \\ &+ \sum_{i=1}^3 \left[ \frac{1}{\alpha_{ig}} \text{tr}[\tilde{W}_{ig}^T(k+1)\tilde{W}_{ig}(k+1) - \tilde{W}_{ig}^T(k)\tilde{W}_{ig}(k)] \right] \end{aligned} \quad (9.3.40)$$

Substituting (9.3.17) through (9.3.24) in (9.3.40), one may obtain

$$\begin{aligned} \Delta J &\leq -(1 - c_0)[\|e(k)\|^2 - 2\frac{c_1}{(1 - c_0)}\|e(k)\| - \frac{c_2}{(1 - c_0)}] \\ &- [1 - (\alpha_{3f}\|\hat{\varphi}_{3f}(k)\|^2 + \alpha_{3g}\|\hat{\varphi}_{3g}(k)\|^2)] \\ &\| (e_f(k) + e_g(k)) - \frac{(\alpha_{3f}\hat{\varphi}_{3f}^T(k)\hat{\varphi}_{3f}(k) + \alpha_{3g}\hat{\varphi}_{3g}^T(k)\hat{\varphi}_{3g}(k)\delta(k))}{[1 - (\alpha_{3f}\hat{\varphi}_{3f}^T(k)\hat{\varphi}_{3f}(k) + \alpha_{3g}\hat{\varphi}_{3g}^T(k)\hat{\varphi}_{3g}(k))]} \|^2 \\ &- \sum_{i=1}^2 (2 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))\|\tilde{W}_{if}^T(k)\hat{\varphi}_{if}(k)\| \\ &- \frac{(1 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))}{(2 - \alpha_{if}\hat{\varphi}_{if}^T(k)\hat{\varphi}_{if}(k))} (W_{if}^T\hat{\varphi}_{if}(k) + B_{if}e(k))\|^2 \\ &- \sum_{i=1}^2 (2 - \alpha_{ig}\hat{\varphi}_{ig}^T(k)\hat{\varphi}_{ig}(k))\|\tilde{W}_{ig}^T(k)\hat{\varphi}_{ig}(k)\| \\ &- \frac{(1 - \alpha_{ig}\hat{\varphi}_{ig}^T(k)\hat{\varphi}_{ig}(k))}{(2 - \alpha_{ig}\hat{\varphi}_{ig}^T(k)\hat{\varphi}_{ig}(k))} (W_{ig}^T\hat{\varphi}_{ig}(k) + B_{ig}e(k))\|^2 \end{aligned} \quad (9.3.41)$$

where

$$\begin{aligned} c_1 &= [\frac{\delta_{max}}{[1 - (\alpha_{3f}\|\hat{\varphi}_{3f}(k)\|^2 + \alpha_{3g}\|\hat{\varphi}_{3g}(k)\|^2)]} + \sum_{i=1}^2 \frac{\kappa_{if}\|\hat{\varphi}_{if}(k)\|W_{ifmax}}{(2 - \alpha_{if}\|\hat{\varphi}_{if}(k)\|^2)} \\ &+ \sum_{i=1}^2 \frac{\kappa_{ig}\|\hat{\varphi}_{ig}(k)\|W_{igmax}}{(2 - \alpha_{ig}\|\hat{\varphi}_{ig}(k)\|^2)}], \end{aligned} \quad (9.3.42)$$

and

$$\begin{aligned} c_2 &= [\frac{\delta_{max}^2}{[1 - (\alpha_{3f}\|\hat{\varphi}_{3f}(k)\|^2 + \alpha_{3g}\|\hat{\varphi}_{3g}(k)\|^2)]} + \sum_{i=1}^2 \frac{\|\hat{\varphi}_{if}(k)\|^2 W_{ifmax}^2}{(2 - \alpha_{if}\|\hat{\varphi}_{if}(k)\|^2)} \\ &+ \sum_{i=1}^2 \frac{\|\hat{\varphi}_{ig}(k)\|^2 W_{igmax}^2}{(2 - \alpha_{ig}\|\hat{\varphi}_{ig}(k)\|^2)}], \end{aligned} \quad (9.3.43)$$

with

$$\delta_{max} = W_{3fmax}\tilde{\varphi}_{3fmax} + W_{3gmax}\tilde{\varphi}_{3gmax} + \epsilon_{Nf} + \epsilon_{Ng} + d_M. \quad (9.3.44)$$

Since  $c_0, c_1$  and  $c_2$  are positive constants,  $\Delta J \leq 0$  as long as (9.3.25), (9.3.26), (9.3.27), and (9.3.37) are satisfied. In addition  $|\sum_{k=k_0}^{\infty} \Delta J(k)| = |J(\infty) - J(0)| < \infty$  since  $\Delta J \leq 0$  as long as (9.3.25), (9.3.26), (9.3.27) and (9.3.37) hold. The definition of  $J$  and inequality (9.3.37) imply that every initial condition in the set  $\chi$  will evolve entirely within  $\chi$ . That is, whenever the identification error  $\|e(k)\|$  is outside the region defined by (9.3.37),  $J(e, \tilde{W}_i)$  will decrease. This further implies that  $\|e(k)\|$  will not increase and will remain in  $\chi$ . This demonstrates that the identification error  $e(k)$  is bounded for all  $k \geq 0$  and it remains to show that the weight estimates  $\hat{W}_{1f}(k), \hat{W}_{2f}, \hat{W}_{3f}(k), \hat{W}_{1g}(k), \hat{W}_{2g}(k)$  and  $\hat{W}_{3g}(k)$ , or equivalently  $\tilde{W}_{1f}(k), \tilde{W}_{2f}, \tilde{W}_{3f}(k), \tilde{W}_{1g}(k), \tilde{W}_{2g}(k)$  and  $\tilde{W}_{3g}(k)$ , are bounded.

The dynamics relative to error in weight estimates using (9.3.17) and (9.3.24) are given by

$$\tilde{W}_{1f}(k+1) = [I - \alpha_{1f}\hat{\varphi}_{1f}(k)\hat{\varphi}_{1f}^T(k)]\tilde{W}_{1f}(k) + \alpha_{1f}\hat{\varphi}_{1f}(k)[W_{1f}^T\hat{\varphi}_{1f}(k) + B_{1f}e(k)]^T, \quad (9.3.45)$$

$$\tilde{W}_{2f}(k+1) = [I - \alpha_{2f}\hat{\varphi}_{2f}(k)\hat{\varphi}_{2f}^T(k)]\tilde{W}_{2f}(k) + \alpha_{2f}\hat{\varphi}_{2f}(k)[W_{2f}^T\hat{\varphi}_{2f}(k) + B_{2f}e(k)]^T, \quad (9.3.46)$$

$$\tilde{W}_{1g}(k+1) = [I - \alpha_{1g}\hat{\varphi}_{1g}(k)\hat{\varphi}_{1g}^T(k)]\tilde{W}_{1g}(k) + \alpha_{1g}\hat{\varphi}_{1g}(k)[W_{1g}^T\hat{\varphi}_{1g}(k) + B_{1g}e(k)]^T, \quad (9.3.47)$$

$$\tilde{W}_{2g}(k+1) = [I - \alpha_{2g}\hat{\varphi}_{2g}(k)\hat{\varphi}_{2g}^T(k)]\tilde{W}_{2g}(k) + \alpha_{2g}\hat{\varphi}_{2g}(k)[W_{2g}^T\hat{\varphi}_{2g}(k) + B_{2g}r(k)]^T, \quad (9.3.48)$$

$$\tilde{W}_{3f}(k+1) = [I - \alpha_{3f}\hat{\varphi}_{3f}(k)\hat{\varphi}_{3f}^T(k)]\tilde{W}_{3f}(k) - \alpha_{3f}\hat{\varphi}_{3f}(k)[e_g(k) + \delta(k)]^T \quad (9.3.49)$$

$$\tilde{W}_{3g}(k+1) = [I - \alpha_{3g}\hat{\varphi}_{3g}(k)\hat{\varphi}_{3g}^T(k)]\tilde{W}_{3g}(k) - \alpha_{3g}\hat{\varphi}_{3g}(k)[e_f(k) + \delta(k)]^T \quad (9.3.50)$$

where the identification error is considered to be bounded. Applying the PE condition (7.2.8) (see Chapter 7), the identification error bound (9.3.37) and Lemma 7.2.1 (see Chapter 7) for the cases of  $\varphi(k) = \hat{\varphi}_i(k); i = 1, 2$ , the boundedness of  $\tilde{W}_{1f}(k), \tilde{W}_{2f}(k), \tilde{W}_{1g}(k)$ , and  $\tilde{W}_{2g}(k)$ , in (9.3.45) through (9.3.48), and hence of  $\hat{W}_{1f}(k), \hat{W}_{2f}(k), \hat{W}_{1g}(k)$  and  $\hat{W}_{2g}(k)$  are assured. For the case of the error system (9.3.11), the weight updates at the third layer of the NN are presented in (9.3.49) with  $e_g(k) = 0$ . Then applying the PE condition similar to the input and hidden layers, it is straightforward to guarantee the boundedness of  $\tilde{W}_{3f}(k)$  and hence of  $\hat{W}_{3f}(k)$ .

By contrast, for the case of error system (9.3.12) in order to show the boundedness of the error in weight estimates at the third layer for both NN, the passivity property of the weight updates are necessary in addition to the PE condition or otherwise, one has to assume that the initial parameter error estimates for both  $f(\cdot)$  and  $g(\cdot)$  are bounded. Assuming that the initial weight estimation errors are bounded for both NN  $f(\cdot)$  and  $g(\cdot)$  and applying the PE condition (7.2.8), the identification error bound (9.3.37) and using (9.3.49) and (9.3.50), one can conclude the boundedness of  $\tilde{W}_{3f}(k)$  and  $\tilde{W}_{3g}(k)$  or equivalently  $\hat{W}_{3f}(k)$  and  $\hat{W}_{3g}(k)$ .

The most elegant way of showing the boundedness of the identification error and weight estimates is to employ passivity theory. Assuming that the closed-loop system (9.3.11) and (9.3.12) with the weight updates (9.3.45) through (9.3.50) are passive, and employing passivity theorem (Slotine and Li 1991), one can readily conclude the boundedness of the identification error and the error in weight updates under the PE condition. However, in the next section, this assumption can be relaxed by showing that in fact the error in weight updates are passive.

Using the boundedness of both  $\|e(k)\|$  and the error in weight estimates, one can observe that  $(e, \tilde{W}_i)$  will not increase when both  $f(\cdot)$  and  $g(\cdot)$  NN are included for the error system (9.3.12) and only for  $f(\cdot)$  in the case of error system (9.3.11) but  $(e, \tilde{W}_i)$  will remain in  $\chi$ . Since  $\chi \supset \Omega x \Omega_U$ , this concludes the proof.  $\square$

Table 9.3.1: Multilayer Neural Net Identifier

*The weight tuning is given by:*

*Input and Hidden Layers:*

$$\hat{W}_{if}(k+1) = \hat{W}_{if}(k) - \alpha_{if}\varphi_{if}^*(k)[\hat{y}_{if}(k) + B_{if}e(k)]^T, \quad i = 1, \dots, n-1,$$

*and*

$$\hat{W}_{ig}(k+1) = \hat{W}_{ig}(k) - \alpha_{ig}\varphi_{ig}^*(k)[\hat{y}_{ig}(k) + B_{ig}e(k)]^T, \quad i = 1, \dots, n-1,$$

*where*  $\hat{y}_{if}(k) = \hat{W}_{if}(k)\varphi_{if}(k)$ ,  $\hat{y}_{ig}(k) = \hat{W}_{ig}(k)\varphi_{ig}(k)$ ,  $\|B_{if}\| \leq \kappa_{if}$ , *and*  $\|B_{ig}\| \leq \kappa_{ig}$ ,  $i = 1, \dots, n-1$ .

*Output Layer: tune*

$$\hat{W}_{nf}(k+1) = \hat{W}_{nf}(k) + \alpha_{nf}\hat{\varphi}_{nf}(k)e^T(k+1),$$

$$\hat{W}_{ng}(k+1) = \hat{W}_{ng}(k) + \alpha_{ng}\hat{\varphi}_{ng}(k)e^T(k+1),$$

*with*  $\alpha_{if} > 0$ , *and*  $\alpha_{ig} > 0, \forall i = 1, \dots, n$  *denoting learning rate parameters or adaptation gains.*

### 9.3.2.1 Discussion

*Since*  $\|e(k)\|$  *cannot increase far beyond the right-hand side of (9.3.37), in applications this may be taken as a practical bound on the norm of the error e(k). Note from (9.3.37), that the identification error increases with the NN reconstruction error bounds and the disturbance bound d\_M, yet small identification errors, but not arbitrarily small, may be achieved by selecting c\_0.*

*As is typical of the algorithms given in this book, there is no preliminary off-line learning phase for the NN. Tuning is performed on-line in real-time. The required terms for tuning are easily evaluated given signals measured in the feedback loop.*

*The proof is easy to extend to case of general n-layer NN in the approximations (9.3.3) and (9.3.4) (Jagannathan 1994). The NN tuning scheme for NN identification of nonlinear systems is given in Table 9.3.1.*

## 9.4 PASSIVITY PROPERTIES OF THE NN

*In this section, an interesting property of the NN identifier is shown—the NN identifier with tuning algorithms given in Table 9.3.1 makes the closed-loop system passive. The practical importance of this is that additional unknown bounded disturbances do not destroy the stability and identification properties of the system. Passivity was discussed in Chapter 2. Note that the NN used in the identifiers in this chapter are feedforward NN with no dynamics. However, embedding them into the identifier dynamics turns them into dynamical or recurrent NN. Additional dy-*

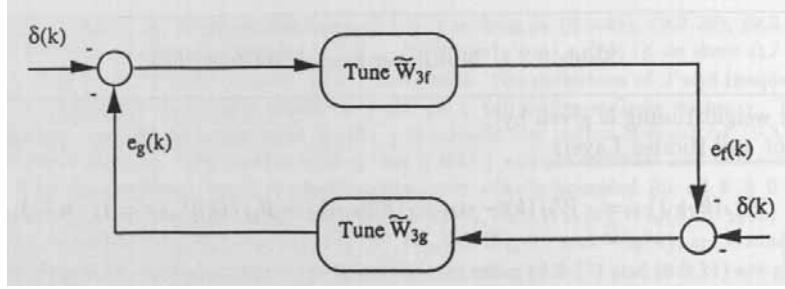


Figure 9.4.1: Neural network closed-loop identifier system.

*namics are introduced by tuning the NN on-line. Therefore, passivity properties can be defined.*

*The complete closed-loop structure using the NN identifier is given in Fig. 9.4.1. Note that all blocks appear in standard feedback configuration. Using the fact that the dynamical NN are passive and invoking the passivity theorem (Slotine and Li 1991) one can easily understand why the errors in the weight estimates of all the layers are bounded. The next result details the passivity properties engendered by the tuning rules in Table 9.3.1.*

**Theorem 9.4.1 (Three-layer NN Passivity Using Tuning Algorithms) :**

Given an unknown system in one of the four forms (9.1.1) through (9.1.4), select the estimator from the respective form (9.2.1) through (9.2.4) and let  $\hat{f}(\cdot)$ , and  $\hat{g}(\cdot)$  if required, be given by NN as in (9.3.3) and (9.3.4). Select the NN tuning algorithms specified in Theorem 9.3.1. Then:

(i) The weight tuning algorithms (9.3.17) through (9.3.20) make the maps from  $W_i^T \hat{\varphi}_i(k) + B_i e(k)$  to  $\tilde{W}_i^T(k) \hat{\varphi}_i(k); \forall i = 1, 2$  both passive maps for NN.

(ii) The weight tuning algorithms (9.3.23) and (9.3.24) make the map from,  $e_g(k) + \delta(k)$  for the case of (9.3.11), and  $e_f(k) + \delta(k)$  for the case of (9.3.12), to  $-\tilde{W}_{3f}^T(k) \hat{\varphi}_{3f}(k)$  and  $-\tilde{W}_{3g}^T(k) \hat{\varphi}_{3g}(k)$  a passive map.

Proof: (i) Define the Lyapunov function candidate

$$J = \frac{1}{\alpha_{1f}} \text{tr}[\tilde{W}_{1f}^T(k) \tilde{W}_{1f}(k)], \quad (9.4.1)$$

whose first difference is given by

$$J = \frac{1}{\alpha_{1f}} \text{tr}[\tilde{W}_{1f}^T(k+1) \tilde{W}_{1f}(k+1) - \tilde{W}_{1f}^T(k) \tilde{W}_{1f}(k)]. \quad (9.4.2)$$

Substituting the weight update law (9.3.17) in (9.4.2) yields

$$\begin{aligned} \Delta J &= -(2 - \alpha_{1f} \hat{\varphi}_{1f}^T(k) \hat{\varphi}_{1f}(k))(-\tilde{W}_{1f}^T(k) \hat{\varphi}_{1f}(k))^T (-\tilde{W}_{1f}^T(k) \hat{\varphi}_{1f}(k)) + \\ &\quad 2(1 - \alpha_{1f} \hat{\varphi}_{1f}^T(k) \hat{\varphi}_{1f}(k))(-\tilde{W}_{1f}^T(k) \hat{\varphi}_{1f}(k))^T (W_{1f}^T \hat{\varphi}_{1f}(k) + B_{1f} e(k)) + \\ &\quad \alpha_{1f} \hat{\varphi}_{1f}^T(k) \hat{\varphi}_{1f}(k)(W_{1f}^T \hat{\varphi}_{1f}(k) + B_{1f} e(k))^T (W_{1f}^T \hat{\varphi}_{1f}(k) + B_{1f} e(k)). \end{aligned} \quad (9.4.3)$$

Note (9.4.3) is in power form (see Chapter 2) as long as the condition (9.3.25) holds. This in turn guarantees the passivity of the weight tuning mechanism (9.3.17).

Similarly, it can be demonstrated that the error in weight updates using (9.3.18) through (9.3.20) are in fact passive.

(ii) Select the Lyapunov function candidate

$$J = \frac{1}{\alpha_{3f}} \text{tr}[\tilde{W}_{3f}^T(k)\tilde{W}_{3f}(k)], \quad (9.4.4)$$

whose first difference is given by

$$\Delta J = \frac{1}{\alpha_{3f}} \text{tr}[\tilde{W}_{3f}^T(k+1)\tilde{W}_{3f}(k+1) - \tilde{W}_{3f}^T(k)\tilde{W}_{3f}(k)]. \quad (9.4.5)$$

Use (9.3.23) in (9.4.5) to obtain

$$\begin{aligned} \Delta J = & -(2 - \alpha_{3f}\hat{\varphi}_{3f}^T(k)\hat{\varphi}_{3f}(k))(-\tilde{W}_{3f}^T(k)\hat{\varphi}_{3f}(k))^T(-\tilde{W}_{3f}^T(k)\hat{\varphi}_{3f}(k)) + \\ & 2(1 - \alpha_{3f}\hat{\varphi}_{3f}^T(k)\hat{\varphi}_{3f}(k))(-\tilde{W}_{3f}^T(k)\hat{\varphi}_{3f}(k))^T(e_g(k) + \delta(k)) + \\ & \alpha_{3f}\hat{\varphi}_{3f}^T(k)\hat{\varphi}_{3f}(k)(e_g(k) + \delta(k))^T(e_g(k) + \delta(k)) \end{aligned} \quad (9.4.6)$$

which is in power form (see Chapter 2) for discrete-time systems as long as the condition (9.3.25) holds.

Similarly, it can be demonstrated that the error in weight updates using (9.3.24) are in fact passive.  $\square$

## 9.5 SIMULATION RESULTS

### Example 9.5.1 (NN Identification of Discrete-Time Nonlinear System) :

Note that no preliminary off-line NN learning phase is needed in this example. In order to illustrate the performance of the NN identifier, a discrete-time nonlinear system is considered. Consider therefore the first-order multi-input/multi-output discrete-time nonlinear system described by

$$x(k+1) = f(x(k)) + u(k), \quad (9.5.1)$$

where  $x(k) = [x_1(k) \ x_2(k)]^T$ ,  $f(x(k)) = \begin{bmatrix} \frac{x_2(k)}{1+x_1^2(k)} \\ \frac{x_1(k)}{1+x_1^2(k)} \end{bmatrix}$ , and  $u(k) = [u_1(k) \ u_2(k)]^T$ . To achieve the objective of identifying this nonlinear system, select an estimator of the form given by (9.2.2), with  $\beta_i = 0, \forall i > 0$  and  $\beta_0 = I$ , the identity matrix. The input is a periodic step input of magnitude two units with a period of 30 s.

A sampling interval of 10 ms was considered. A three-layer NN was selected with two input, four hidden, and two output nodes. Sigmoidal activation functions were employed in all the nodes in the hidden layer. The initial conditions for the plant and the model were chosen to be  $[2, -2]^T$  and  $[0.1, 0.6]^T$ . The weights were initialized to zero with an initial threshold value of 3.0. No learning is performed initially to train the networks. The elements in the design matrix,  $B_i$ , are chosen to be 0.1. Consider the case where the constant learning rate parameter is replaced with the projection algorithm where the adaptation gains are selected to be  $\xi_1 = 1.0, \xi_2 = 1.0$ , and  $\xi_3 = 0.7$  with  $\zeta_1 = \zeta_2 = \zeta_3 = 0.001$ . Let us consider the case when a bounded disturbance given by

$$w(k) = \begin{cases} 0.0 & 0 \leq kT_m < 12 \\ 0.5 & kT_m \geq 12 \end{cases} \quad (9.5.2)$$

is acting on the plant at the time instant  $t$ . Fig. 9.5.1 presents the tracking response of NN controllers with the improved weight tuning and projection algorithm. The magnitude of

the disturbance can be increased, however, and the value should be bounded. The value shown in (9.5.2) is employed for simulation purposes only. From the figure it is clear that the response of the NN identifier is extremely impressive.  $\square$

## 9.6 CONCLUSIONS

*In this chapter a general NN identifier was derived that estimates the state for systems in any one of four standard nonlinear autoregressive-moving average forms. NN are used to estimate the nonlinear functions appearing in the dynamics so that the state estimate converges to the actual state in the unknown system. Nonlinear-in-the-parameters three-layer NN were used, so that the function approximation property of NN guarantees the existence of the identifier. Passivity properties of the NN identifier were discussed.*

## 9.7 REFERENCES

- Billings, S.A., H.B. Jamaluddin, and S. Chen, "Properties of neural networks with applications to modelling nonlinear dynamical systems," *Int. J. of Control*, vol. 55, pp. 193-224, 1992.
- Cybenko, G., "Approximations by superpositions of sigmoidal activation function", *Math. Contr., Signals, Syst.*, vol.2, no.4, pp.303-314, February 1989.
- Fung, C.F., S.A. Billings, and H. Zhang, "Generalised transfer functions of neural networks," *Mech. Systems and Signal Processing*, vol. 11, no. 6, pp. 843-868, 1997.
- Goodwin, G.C., and K.S. Sin, *Adaptive Filtering, Prediction, and Control*, Prentice-Hall, Englewood Cliffs, NJ, 1984.
- Jagannathan, S., "Intelligent control of nonlinear dynamical systems using multilayered neural networks," Ph.D. Thesis, Dept. of Electrical Engineering, The University of Texas at Arlington, Arlington, Texas 76019, August 1994.
- Jagannathan, S., and F.L. Lewis, "Identification of nonlinear dynamical systems using multilayered neural networks," *Automatica*, vol. 32, no. 12, pp. 1707-1712, 1996.
- Landau, I.D., *Adaptive Control: The Model Reference Approach*, Marcel Dekker, New York, 1979.
- Landau, I.D., "Evolution of adaptive control", *ASME J. Dynamic Syst. Measurements, Contr.*, vol. 115, pp. 381-391, June 1993.
- Ljung, L., and T. Söderström, *Theory and Practice of Recursive Identification*, MIT Press, Cambridge, MA, 1983.
- Narendra, K.S., and A.M. Annaswamy, *Stable Adaptive Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

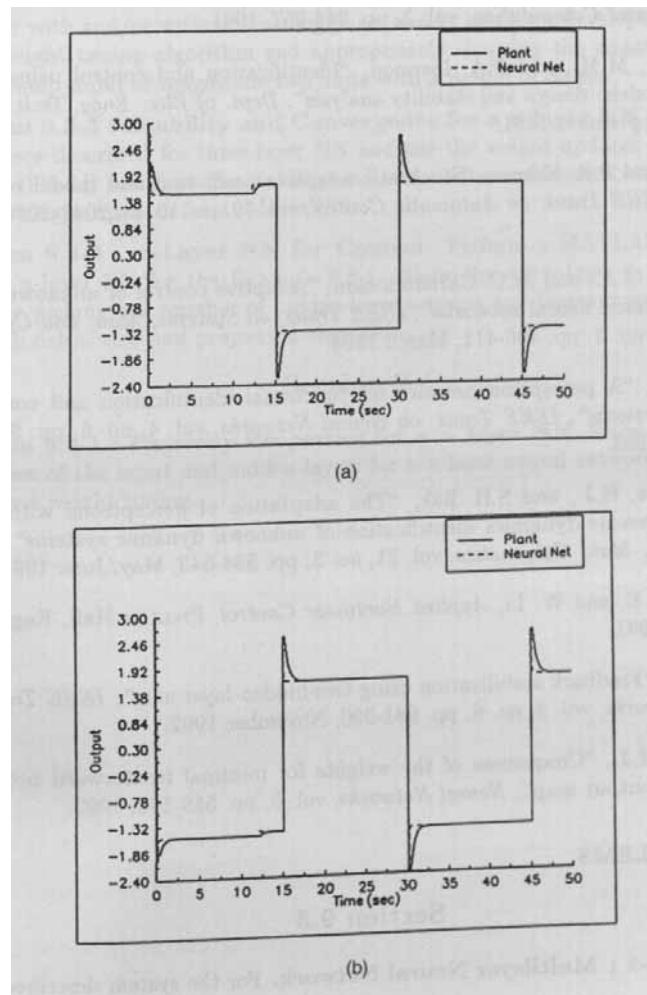


Figure 9.5.1: Response of neural network identifier with projection algorithm in the presence of bounded disturbances. (a) Desired and actual state 1. (b) Desired and actual state 2.

*Narendra, K.S., and K.S. Parthasarathy, "Identification and control of dynamical systems using neural networks", IEEE Trans. on neural networks, vol. 1, no.1, pp. 4-27, March 1990.*

*Park, J., and Sandberg, "Universal approximation using radial-basis-function networks", Neural Computation, vol. 3, pp. 246-257, 1991.*

*Polycarpou, M.M., and P.A. Ioannou, "Identification and control using neural network models: design and stability analysis", Dept. of Elec. Engg, Tech. Report 91-09-01, September 1991.*

*Ren, W., and P.R. Kumar, "Stochastic adaptive prediction and model reference control", IEEE Trans. on Automatic Control, vol. 39, no. 10, pp. 2047-2060, October 1994.*

*Rovithakis, G.A., and M.C. Christodoulou, "Adaptive control of unknown plants Using dynamical neural networks", IEEE Trans. on Systems, Man, and Cybernetics, vol. 24, no. 3, pp. 400-411, March 1994.*

*Sadegh, N., "A perceptron network for functional identification and control of nonlinear systems", IEEE Trans. on Neural Networks, vol. 4, no. 6, pp. 982-988, November 1993.*

*Sira-Ramirez, H.J., and S.H. Zak, "The adaptation of perceptrons with applications to inverse dynamics identification of unknown dynamic systems", IEEE Trans. Syst., Man, Cybernetics, vol. 21, no. 3, pp. 534-543, May/June 1991.*

*Slotine, J.-J.E, and W. Li, Applied Nonlinear Control, Prentice-Hall, Englewood Cliffs, NJ, 1991.*

*Sontag, E., "Feedback stabilization using two-hidden-layer nets", IEEE Trans. on Neural Networks, vol. 3, no. 6, pp. 981-990, November 1992.*

*Sussmann, H.J., "Uniqueness of the weights for minimal feedforward nets with given input-output map", Neural Networks, vol. 5, pp. 589-593, 1992.*

## 9.8 PROBLEMS

### Section 9.3

**Problem 9.3-1 : Multilayer Neural Network.** For the system described by

$$x(k+1) = f(x(k), x(k-1)) + u(k), \quad (9.8.1)$$

where  $f(x(k), x(k-1)) = \frac{x(k)x(k-1)[x(k)+1.0]}{1+x^2(k)+x^2(k-1)}$ .

Design a multilayer neural network identifier with and/or without learning phase and by using the developed delta rule-based weight tuning algorithm and appropriately choosing the adaptation gains for a sinusoidal input of a chosen magnitude and frequency.

**Problem 9.3-2 : Multilayer Neural Network.** For the system described by

$$x(k+1) = f(x(k), x(k-1)) + u(k), \quad (9.8.2)$$

where  $f(x(k), x(k-1)) = \frac{x(k)}{1+x(k)} + u^3(k)$ . Design a multilayer neural network identifier with and/or without learning phase and by using the developed delta rule-based weight tuning algorithm and appropriately choosing the adaptation gains. Select a step input of magnitude two units with a period of 30 sec.

**Problem 9.3-3 : Stability and Convergence for a  $n$ -layer NN.** Assume the hypotheses described for three-layer NN and use the weight updates presented in (9.3.17)-(9.3.24) and show the stability and boundedness of identification error and error in weight updates for a  $n$ -layer NN.

**Problem 9.3-4 :  $n$ -Layer NN for Control.** Perform a MATLAB simulation using a  $n$ -layer NN for the Example 9.5.1. Show the advantage of adding more layers by picking less number of hidden-layer neurons and layers more than three. Use both delta- rule and projection algorithm.

## Section 9.4

**Problem 9.4-1 : Passivity Properties for a  $n$ - layer NN.** Show the passivity properties of the input and hidden layers for a  $n$ -layer neural network using delta rule-based weight tuning.