Contents

Pre	eface			v	
1.	Intro	duction	to Simulation and Computer-aided Design of Control Systems	1	
	1.1 A Brief Historic Review of the Development of Computer-aided				
		Design	of Control Systems	1	
	1.2	Introdu	uction to the CACSD Languages and Environments	2	
	1.3	Develo	pment of Simulation Software	5	
	1.4	MATL	AB/Simulink and Their CACSD Toolboxes	6	
	1.5	Overvi	ew of CACSD Approaches	8	
	1.6	Fundai	mental Structures and Contents of This Book	10	
	1.7	Proble	ms	12	
	Biblio	ography	and References	14	
2.	Fund	amental	s of MATLAB Programming	17	
	2.1	Basics	in MATLAB Programming	18	
		2.1.1	Variables and Constants in MATLAB	18	
		2.1.2	Data Structures	19	
		2.1.3	Basic Statement Structures of MATLAB	20	
		2.1.4	Colon Expressions	21	
		2.1.5	Sub-matrix Extraction	22	
	2.2	Basic I	Mathematical Operations	22	
		2.2.1	Algebraic Calculations of Matrices	22	
		2.2.2	Logic Operations of Matrices	24	
		2.2.3	Relationship Operations of Matrices	24	
		2.2.4	Simplifications and Presentations of Analytical Results	24	
		2.2.5	Basic Number Theory Computations	26	
	2.3	Flow C	Control Structures in MATLAB Programming	27	
		2.3.1	Loop Control Structures	27	
		2.3.2	Conditional Structure	28	
		2.3.3	Switch Structure	29	

-	-
- 3	с.
-	r

		2.3.4	Trial Structure	29
	2.4	Functi	on Writing and Debugging	30
		2.4.1	Basic Structure of MATLAB Functions	30
		2.4.2	Functions with Variable Numbers of Inputs and Outputs .	33
		2.4.3	Anonymous and Inline Functions	33
		2.4.4	Pseudo Codes	34
	2.5	Two-d	limensional Graphics	34
		2.5.1	Basic Statements of Two-dimensional Plotting	34
		2.5.2	Other Graphics Functions with Applications	37
		2.5.3	Implicit Function Visualizations	39
		2.5.4	Graph Editing and Decorations	39
	2.6	Three-	-dimensional Visualization	41
		2.6.1	Three-dimensional Curves	41
		2.6.2	Three-dimensional Surfaces	41
		2.6.3	Viewpoint Setting in 3D Plots	44
	2.7	Graph	ical User Interface Design in MATLAB	45
		2.7.1	Graphical User Interface Tool – Guide	46
		2.7.2	Handle Graphics and Properties of Objects	46
		2.7.3	Menu System Design	52
		2.7.4	An Illustrative Example in GUI Design	52
		2.7.5	Toolbar Design	54
		2.7.6	Embedding ActiveX Components in GUIs	56
	2.8	Proble	ems	57
	Bibli	ography	and References	60
3.	MAT	LAB So	olutions to Scientific Computation Problems	61
	3.1	MATI	AB Solutions to Linear Algebra Problems	62
		3.1.1	Fundamental Analysis of Matrices	62
		3.1.2	Matrix Decomposition	64
		3.1.3	Matrix Exponential $e^{\mathbf{A}}$ and Exponential Function $e^{\mathbf{A}t}$	66
	3.2	Solutio	ons of Algebraic Equations	66
		3.2.1	Solutions of Linear Algebraic Equations	66
		3.2.2	Solutions of Nonlinear Equations	69
		3.2.3	Solutions of Nonlinear Matrix Equations	72
	3.3	Solutio	ons of Ordinary Differential Equations	74
		3.3.1	Numerical Solutions to First-order Explicit ODEs	74
		3.3.2	Conversions of ODEs	77
		3.3.3	Validations of Numerical Solutions	78
		3.3.4	Analytical Solutions to Linear ODEs	80
	3.4	MATI	AB Solutions to Optimization Problems	81
		3.4.1	Unconstrained Optimization Problems	81
		3.4.2	Constrained Optimization Problems	82

4.

xi

Contents

	3.4.3	Least Squares Curve Fitting	84
3.5	Laplac	e and z Transforms and MATLAB Solutions	86
	3.5.1	Laplace Transform	86
	3.5.2	z Transform	87
3.6	Proble	ms	88
Biblio	ography	and References	94
Math	ematica	l Models of Linear Control Systems	95
4.1	Linear	System Models of Linear Continuous Systems	96
	4.1.1	Transfer Function Models	96
	4.1.2	State Space Models	99
	4.1.3	State Space Models with Internal Delays	100
	4.1.4	Zero–pole–gain Models $\ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots$	100
	4.1.5	Transfer Function Matrices of Multivariable Systems $\ . \ .$.	102
4.2	Mathe	matical Models of Linear Discrete-time Systems	103
	4.2.1	Discrete-time Transfer Function Models	103
	4.2.2	Discrete-time State Space Models	104
4.3	Equiva	lent Conversions of System Models	105
	4.3.1	Conversion Between Continuous and Discrete-time Models	105
	4.3.2	Converting to Transfer Function Models	107
	4.3.3	State Space Realization of Control Systems	108
	4.3.4	Balanced Realizations	108
	4.3.5	Minimum Realization of State Space Models	109
	4.3.6	Conversion between Transfer Functions and Symbolic	
		Expressions	110
4.4	Block	Diagram Description and Simplification	110
	4.4.1	Typical Connections of Control Systems	110
	4.4.2	Delay Loop Processing with State Space Models	114
	4.4.3	Equivalent Transforms When the Nodes Are Moved	116
	4.4.4	Simplification of Complicated Block Diagrams	117
	4.4.5	Model Simplification Using An Algebraic Approach	119
4.5	Model	Reduction of Linear Systems	121
	4.5.1	Padé Approximations and Routh Approximations	122
	4.5.2	Padé Approximations to Models with Time Delays	125
	4.5.3	Sub-optimal Model Reduction to Models with Time Delays	127
	4.5.4	Reduction Approaches for State Space Models	131
4.6	Identif	ication of Linear Systems	134
	4.6.1	Identification of Discrete-time Models	134
	4.6.2	Order Selection in Identification	138
	4.6.3	Generation of Signals for Identification	140
	4.6.4	Identification of Continuous Systems	141
	4.6.5	Identification of Multivariable Systems	142

xii

 ${\it Modeling, Analysis \ and \ Design \ of \ Control \ Systems \ in \ MATLAB \ and \ Simulink}$

		4.6.6	Least Squares Recursive Identification
	4.7	Proble	ms
	Biblie	ography	and References
5.	Com	puter-Ai	ded Analysis of Linear Control Systems 151
	5.1	Proper	ties of Linear Control Systems
		5.1.1	Stability of Linear Systems
		5.1.2	Internal Stability of Feedback Control Systems 155
		5.1.3	Similarity Transformation of Linear Control Systems 156
		5.1.4	Controllability of Linear Systems
		5.1.5	Observability of Linear Systems
		5.1.6	Canonical Kalman Decompositions
		5.1.7	MATLAB Solutions to Canonical State Space Models 161
		5.1.8	Norms of Linear Systems
	5.2	Analyt	cical Time Domain Responses of Linear Systems 166
		5.2.1	Analytical Solutions with Direct Integration Method 166
		5.2.2	Analytical Solutions with State Augmentation Method $~$. $~$ 167
		5.2.3	Analytical Solutions with Laplace and z Transforms 169
		5.2.4	Time Responses of Systems with Nonzero Initial
			Conditions
		5.2.5	Time Response Specifications of Second-order Systems 173
	5.3	Numer	rical Solutions of Time Domain Responses
		5.3.1	Step Responses and Impulse Responses
		5.3.2	Time Domain Responses for Arbitrary Inputs 180
		5.3.3	Responses for Systems with Nonzero Initial Conditions 181
	5.4	Root I	Locus Analysis
	5.5	Freque	ency Domain Analysis
		5.5.1	Frequency Domain Analysis of Single Variable Systems 189
		5.5.2	Stability Assessment of Feedback Systems 193
		5.5.3	Gain Margins and Phase Margins
	5.6	Freque	ency Domain Analysis of Multivariable Systems 196
		5.6.1	Frequency Domain Analysis of Multivariable Systems 196
		5.6.2	Diagonal Dominance Analysis
		5.6.3	Singular Value Plots for Multivariable Systems 202
	5.7	Proble	ms $\ldots \ldots 203$
	Biblie	ography	and References
6.	Simu	link and	Simulation of Nonlinear Systems 209
	61	Funda	mentals of Simulink Modeling 210
	0.1	611	Introduction to Simulink 910
		619	Commonly Used Blocks in Simulink 211
		613	Other Commonly Used Blocksets 216
		0.1.0	Other Commonly Used Diocksets

Contents

	•		
x	1	1	1
\mathbf{r}	-	-	1

	6.2	Simuli	nk Modeling and Simulation	217
		6.2.1	Introduction to Simulink Modeling Methodology	217
		6.2.2	Simulation Algorithms and Simulation Parameter	
			Selections	221
		6.2.3	An Illustrative Example of Simulink Modeling	223
	6.3	Simuli	nk Modeling of Various Control Systems	226
	6.4	Analys	sis and Simulation of Nonlinear Systems	237
		6.4.1	Modeling of Piecewise Nonlinearities	237
		6.4.2	Linearization of Nonlinear Systems	239
	6.5	Subsys	stem and Model Masking Methods	243
		6.5.1	Subsystem Creation	243
		6.5.2	Subsystem Masking	244
		6.5.3	Constructing Users' Own Block Library	249
	6.6	M-func	ction, S-function and Their Applications $\ldots \ldots \ldots \ldots$	250
		6.6.1	Basic Structure of M-function Blocks	250
		6.6.2	Basic Structures of S-functions	251
		6.6.3	Examples of MATLAB S-function Programming	252
		6.6.4	Mask an S-Function Block	257
	6.7	Proble	ms	258
	Biblie	ography	and References \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots	261
7	Class	ical Des	ign Approaches of Control Systems	263
1.	7 1			200
	(.1	Design	I of Phase Lead–lag Compensators	264
		(.1.1	Lead-leg Compensators	264
	7.0	(.1.2 Ctata (A Design Algorithm for Lead-lag Compensator	200
	1.2	State 2	Space-based Controller Design Strategies	270
		7.2.1		270
		(.2.2	Linear Quadratic Optimal Regulators	271
		7.2.3	Pole Placement Controller Design	273
	7.9	(.2.4 Outing	Observer Design and Observer-based Regulators	270
	1.5	Optim 7.2.1	al Controller Design	281
		7.2.0	Introduction to Optimal Control	281
		7.3.2	An Optimal Controller Design Interface	284
	77 4	(.3.3	Uther Applications of OCD Interface	289
	1.4	Contro	Letre desting to MATLAR Control System Toolbox	290
		7.4.1	Introduction to MAILAB Controller Design Interface	290
		(.4.2	An Example of Parameter Automatic Tuning for Single	202
	7 5	D	variable Systems	293
	6.)	rreque	Diagonal Deminant and David diagonal distriction	290
		7.5.1	Diagonal Dominant and Escudo-diagonalization	291
		(.5.2	Parameter Optimization Design for Multivariable Systems	302
		1.5.3	Optimal Controller Design with OCD Interface	308

xiv		Modeling, A	Analysis and Design of Control Systems in MATLAB and Simulink	
	7.6	Decoupl 7.6.1 7.6.2	ing Control of Multivariable Systems	310 310 311
	7.7 Bibli	Problem ography a	nd References	$314 \\ 317$
8.	Para	meter Tur	ing of PID Controllers	319
	8.1	Introduce 8.1.1 8.1.2 8.1.3	ction to PID Controller Design	320 320 322 323
	8.2	First-ord 8.2.1 8.2.2 8.2.3 8.2.4	der Delay Model Approximation to Plant Models FOPDT Model by Step Responses	324 324 326 327 327
	8.3	Paramet 8.3.1 8.3.2 8.3.3 8.3.4 8.3.5	Zer Tuning of PID Controllers for FOPDT Plants Ziegler–Nichols Empirical Formula Improved Ziegler–Nichols Algorithm Improved PID Control Structure and Algorithms Chien–Hrones–Reswick Parameter Tuning Algorithm Optimal PID Controller Tuning Rule	328 328 330 332 335 336
	8.4 8.5 8.6 8.7 Bibli	PID Tur PID Par 8.5.1 8.5.2 8.5.3 8.5.4 8.5.5 8.5.6 OptimP Problem iography a	her — A PID Controller Design Interface	338 341 342 343 344 347 350 351 356 358
9.	Robi 9.1	Linear G 9.1.1 9.1.2	I and Robust Controller Design Quadratic Gaussian Control LQG Problems Solving LQG Problems with MATLAB LOC Control with Loss Transfer Deserver	359 360 360 360
	9.2	General 9.2.1	Descriptions to Robust Control Problems	369 369

Contents

v	τ.	
A	v	

		9.2.2	Structures of Robust Controllers	369
		9.2.3	Description of Loop Shaping Techniques	372
		9.2.4	MATLAB Description of Robust Control Systems	373
	9.3	Norm-	based Robust Controller Design	377
		9.3.1	Design of \mathcal{H}_{∞} and \mathcal{H}_2 Robust Controllers	377
		9.3.2	Other Robust Controller Design Functions	382
		9.3.3	Youla Parameterization	386
	9.4	Linear	Matrix Inequality Theory and Solutions	387
		9.4.1	General Descriptions of Linear Matrix Inequalities	387
		9.4.2	MATLAB Solutions to Linear Matrix Inequality Problems	390
		9.4.3	Optimization Problem Solutions with YALMIP Toolbox .	393
		9.4.4	Simultaneous Stabilization Multiple Linear Models	394
		9.4.5	Robust Optimal Controller Design with LMI Solvers	395
	9.5	Quanti	tative Feedback Theory and Design Methods	397
		9.5.1	Introduction to Quantitative Feedback Theory	397
		9.5.2	QFT Design Method for Single Variable Systems	398
	9.6	Proble	ms	404
	Biblic	ography	and References $\hdots \hdots \hdots$	405
10	A 1			105
10.	Ada	ptive an	d Intelligent Control Systems Design	407
	10.1	Design	of Adaptive Control Systems	408
		10.1.1	Design and Simulation of Model Reference Adaptive	
			Control Systems	409
		10.1.2	Solutions of Polynomial Diophantine Equations	411
		10.1.3	d-step Ahead Forecast	413
		10.1.4	Design of Minimum Variance Controllers	414
		10.1.5	Generalized Minimum Variance Control	416
	10.2	Simula	tion and Design of Model Predictive Control Systems	418
		10.2.1	Dynamic Matrix Control	419
		10.2.2	Design of Model Predictive Controllers with MATLAB	420
		10.2.3	Design and Simulation of Model Predictive Control for	
			Complicated Plants	426
		10.2.4	Generalized Predictive Control Systems and Simulations .	429
	10.3	Fuzzy	Control and Fuzzy Logic Controller Design	432
		10.3.1	Fuzzy Logic and Fuzzy Inference	432
		10.3.2	Design of Fuzzy PD Controller	434
		10.3.3	Design of Fuzzy PID Controllers	439
	10.4	Neural	Networks and Neural Network Controller Design $\ . \ . \ .$.	442
		10.4.1	Introduction to Neural Networks	443
		10.4.2	Design of PID Controller with Single Neurons	444
		10.4.3	PID Controller with Back-propagation Neural Networks	447

xvi

$Modeling, \ Analysis \ and \ Design \ of \ Control \ Systems \ in \ MATLAB \ and \ Simulink$

		10.4.4	PID Controller with Radial Basis Function-based Neural	
			Network	449
		10.4.5	Design and Simulation of Neural Network Controllers	451
	10.5	Simular	tion Analysis of Iterative Learning Control	455
		10.5.1	Principles of Iterative Learning Control	456
		10.5.2	Iterative Learning Control Algorithms	457
	10.6	Design	of Global Optimal Controllers	462
		10.6.1	Introduction to Genetic Algorithm	462
		10.6.2	Solving Global Optimization Problems with Genetic	
			Algorithms	463
		10.6.3	Particle Swarm Optimization Algorithms and Applications	466
		10.6.4	Optimal Controller Design with Global Optimization	
			Algorithms	467
	10.7	Proble	ns	470
	Biblic	graphy	and References	473
11.	Ana	lysis and	l Design of Fractional-order Systems	477
	11.1	Definiti	ions and Numerical Computations in Fractional-order	
		Calculu	18	478
		11.1.1	Definitions of Fractional-order Calculus	478
		11.1.2	The Relationship of Different Definitions	479
		11.1.3	Properties of Fractional-order Calculus	480
	11.2	Numer	ical Computations in Fractional-order Calculus	480
		11.2.1	Numerical Solutions with Grünwald–Letnikov Definition $\ .$	481
		11.2.2	Numerical Solutions with Caputo Definition	482
		11.2.3	Mittag–Leffler Functions and Their Computations	483
	11.3	Solutio	ns of Linear Fractional-order Systems	485
		11.3.1	Numerical Solutions of Linear Fractional-order Differential	
			Equations	485
		11.3.2	Numerical Solutions of Caputo Differential Equations	487
		11.3.3	Some Important Laplace Transforms	489
		11.3.4	Analytical Solutions of Commensurate-order Linear	
			Differential Equations	489
		11.3.5	Analytical Solutions of Linear Fractional-order Differential	
			Equations	491
	11.4	Modeli	ng and Analysis of Fractional-order Transfer Functions	491
		11.4.1	FOTF — Creation of a MATLAB Object	492
		11.4.2	Interconnections of FOTF Blocks	493
		11.4.3	Analysis of FOTF Objects	496
		11.4.4	Frequency Domain Analysis of FOTF Objects	499
		11.4.5	Time Domain Analysis of FOTF Objects	499
		11.4.6	Root Locus for Commensurate-order Systems	500

Contents

xv	1	1	
	-	•	

		11.4.7 State Space Medels of Commonsurate order Systems	501
	11 5	11.4.7 State Space Models of Commensurate-order Systems	501
	11.5	Approximation and Reduction of Fractional-order Systems	502
		11.5.1 Oustaloup Filter for Fractional-order Differentiators	502
		11.5.2 Approximations of Fractional-order Controllers	505
		11.5.3 Optimal Reduction Algorithm for Fractional-order Models	506
	11.6	Simulation Methods for Complicated Fractional-order Systems	507
		11.6.1 Simulation with Numerical Laplace Transform	507
		11.6.2 Block Diagram Modeling and Simulation of Linear	
		Fractional-order Systems	509
		11.6.3 Block Diagram Modeling and Simulation of Nonlinear	
		Fractional-order Systems	512
	11.7	Design of Optimal Fractional-order PID Controllers	512
		11.7.1 Optimal Design of $PI^{\lambda}D^{\mu}$ Controllers	512
		11.7.2 OptimFOPID — An Optimal Fractional-order PID	
		Controller Design Interface	517
	11.8	Problems	519
	Biblic	ography and References	520
12.	Hare	dware-in-the-loop Simulation and Real-time Control	523
	12.1	Introduction to dSPACE and Commonly Used Blocks	523
	12.2	Introduction to Quanser System and Its Blocks	525
		12.2.1 Introduction to Commonly Used Blocks in Quanser	525
		12.2.2 Brief Introduction to Plants in Quanser Rotary Series	526
	12.3	An Example of Hardware-in-the-loop Simulation and Real-time	0 - 0
		Control	526
		12.3.1 Mathematical Description of the Plant Model	526
		12.3.2 Real-time Experiments with Quanser	529
		12.3.3 Real-time Experiments with dSPACE	531
	124	Low-cost Realizations of Hardware-in-the-loop Simulation	533
	12.1	12.4.1 Arduino Interface Installation and Settings	533
		12.4.2 Applications of Arduino Control	534
		12.4.2 The MESA Box	536
	19.5	Problems	537
	Biblic	araphy and References	537
	DIDII		001
Ap	pendix	A Some Practical Plant Models	539
-	A 1	Woll known Bonchmark Problems	530
	11.1	A 1.1 Control of the F-14 Aircraft Model	520
		A 1.9 ACC Banchmark Problem	540
	٨٥	Other Engineering Models	540
	A.Z	A 2.1 Control System Model	541
		A.2.1 Servo Control System Model	541
		A.Z.Z Mathematical Model of Inverted Pendullum	-042

A.2.3 AIRC Model	543
A.3 Problems	543
Bibliography and References	544
Index of Functions	545
Index	551

Chapter 11

Analysis and Design of Fractional-order Systems

Fractional-order system theory is an active field of research over the last two decades. In recent years, many progresses have been made in both fractional-order calculus and fractional-order control. The research on fractional-order control has both theoretical value and practical perspectives.

The so-called "fractional-order systems" really mean that the order of the systems are no longer integers, and this is different from all discussed in the book so far. We all know that $d^n y/dt^n$ represents the *n*th order derivative of *y* with respect to *t*. What happens if n = 1/2? This was the question a famous French mathematician Guillaume François Antoine L'Hôpital asked one of the inventors of calculus, Gottfried Wilhelm Leibniz^[1-3], 300 years ago. From that time on, researchers began to study fractional-order calculus problems. Thus, fractional-order calculus is a 300 year-old topic, however, earlier work focused on pure mathematics. In the 19th century, various definitions on fractional-order calculus appeared, and it was not until 1960, the first publication on non-integer order integrator appeared in the field of control^[4], however, few attention to the topics were received. In late 1990's, fractional-order PID controller appeared^[5-7].

The operator \mathscr{D}^{α} is used in this book to describe fractional-order differentiation and integration. When $\alpha > 0$, α th order derivative is used, while $\alpha < 0$ means $-\alpha$ th order integration, and $\alpha = 0$ means the original function. This unified notation will be used throughout the book.

Strictly speaking, "fractional-order" is a misused term. The precise one should be "non-integer-order", since the order can either be fractional, or an irrational, or even complex numbers. For instance, $d^{\sqrt{2}}y/dt^{\sqrt{2}}$ means the $\sqrt{2}$ th order derivative. However, the word "fractional-order" was used for a very time among the researchers. In this book, "fractional-order" is used, however, it actually means "non-integer-order".

In Section 11.1, various definitions on fractional-order calculus and their MATLAB implementations are presented. Properties of fractional-order calculus are also presented. Mittag-Leffler functions and commonly used Laplace transforms in fractional-order functions are also presented. In Section 11.2, numerical computations of fractional-order derivatives are presented. Also Mittag-Leffler functions and commonly used Laplace transforms in fractional-order functions are also presented. Section 11.3 presents the numerical and analytical solutions of linear fractional-order differential equations. The fractional-order transfer function is used as an example to demonstrate object-oriented programming in Section 11.4, where the methods of class creation, overload function design are all illustrated. With the programming technique, linear fractional-order systems analysis — stability, time and frequency domain analysis are presented. In Section 11.5,

integer-order approximation to fractional-order operators and fractional-order systems are presented, and sub-optimal reduction technique are proposed. In Section 11.6, block diagram-based nonlinear fractional-order system simulation techniques are illustrated. In Section 11.7, optimal design methods and interface of fractional-order PID controllers for fractional-order plants are presented.

11.1 Definitions and Numerical Computations in Fractional-order Calculus

In the development of fractional-order calculus, various definitions were proposed and applied. The most widely used Cauchy integral formula, Grünwald–Letnikov definition, Riemann–Liouville definition and Caputo definition are all extended directly from integralorder calculus. In this section, the definitions and their relationships are given, then the numerical computation and properties are presented.

11.1.1 Definitions of Fractional-order Calculus

1. Fractional Cauchy integral formula

The formula is extended directly from its integer-order counterpart

$$\mathscr{D}_t^{\gamma} f(t) = \frac{\Gamma(\gamma+1)}{2\pi \mathrm{j}} \int_{\mathrm{C}} \frac{f(\tau)}{(\tau-t)^{\gamma+1}} \mathrm{d}\tau, \qquad (11.1)$$

where C is the closed-path that encircles all the poles of the function f(t), and \mathcal{D}_t is the fractional-order differentiation operator.

2. Grünwald–Letnikov definition

The definition of fractional-order differentiation is

$${}_{t_0}\mathscr{D}_t^{\alpha}f(t) = \lim_{h \to 0} \frac{1}{h^{\alpha}} \sum_{j=0}^{[(t-t_0)/h]} (-1)^j \binom{\alpha}{j} f(t-jh),$$
(11.2)

where $w_j^{(\alpha)} = (-1)^j \begin{pmatrix} \alpha \\ j \end{pmatrix}$ is the binomial coefficients of $(1-z)^{\alpha}$, and it can be obtained iteratively from

$$w_0^{(\alpha)} = 1, \ w_j^{(\alpha)} = \left(1 - \frac{\alpha + 1}{j}\right) w_{j-1}^{(\alpha)}, \ j = 1, 2, \cdots$$
 (11.3)

The following formula can be used in calculating fractional-order derivative

$${}_{t_0}\mathscr{D}_t^{\alpha}f(t) \approx \frac{1}{h^{\alpha}} \sum_{j=0}^{[(t-t_0)/h]} w_j^{(\alpha)} f(t-jh).$$
(11.4)

If step size h is small enough, Eqn. (11.4) can be used to directly evaluate the approximate values of fractional-order differentiation. It can be shown that^[2] the accuracy of the definition is o(h).

3. Riemann–Liouville definition

Fractional-order integral is defined as

$${}_{t_0}\mathscr{D}_t^{-\alpha}f(t) = \frac{1}{\Gamma(\alpha)} \int_{t_0}^t \frac{f(\tau)}{(t-\tau)^{1-\alpha}} \mathrm{d}\tau, \qquad (11.5)$$

where $0 < \alpha < 1$, and t_0 is the initial instance. Normally $t_0 = 0$, and the notation is simplified to $\mathscr{D}_t^{-\alpha} f(t)$. Riemann–Liouville definition is the most widely used definition in fractional-order calculus. Especially the subscripts on both sides of \mathscr{D} are the lower and upper bounds in the integral^[8].

Fractional-order differentiation can also be defined. Assume that the order α of differentiation satisfies $n - 1 < \beta \leq n$, the differentiation is defined as

$${}_{t_0}\mathscr{D}_t^{\beta}f(t) = \frac{\mathrm{d}^n}{\mathrm{d}t^n} \left[{}_{t_0}\mathscr{D}_t^{-(n-\beta)}f(t) \right] = \frac{1}{\Gamma(n-\beta)} \frac{\mathrm{d}^n}{\mathrm{d}t^n} \left[\int_{t_0}^t \frac{f(\tau)}{(t-\tau)^{\beta-n+1}} \mathrm{d}\tau \right].$$
(11.6)

4. Caputo definition

The definition of Caputo fractional-order differentiation is

$${}_{t_0}\mathscr{D}_t^{\alpha}f(t) = \frac{1}{\Gamma(1-\gamma)} \int_{t_0}^t \frac{f^{(m+1)}(\tau)}{(t-\tau)^{\gamma}} \mathrm{d}\tau,$$
(11.7)

where $\alpha = m + \gamma$, or $m = \lfloor \alpha \rfloor$ is an integer such that $0 < \gamma < 1$. Similarly, the Caputo fractional-order integral is defined as

$${}_{t_0}\mathcal{D}_t^{-\alpha}f(t) = \frac{1}{\Gamma(\alpha)} \int_{t_0}^t \frac{f(\tau)}{(t-\tau)^{1-\alpha}} \mathrm{d}\tau, \quad \alpha > 0,$$
(11.8)

and it is exactly the same as Riemann–Liouville definition in Eqn. (11.5).

11.1.2 The Relationship of Different Definitions

It can be shown that for a wide class of practical functions, the Grünwald–Letnikov and Riemann–Liouville definitions are equivalent^[2]. In this book, we shall not distinguish them.

The major differences of Caputo definition and Riemann–Liouville definition are that the former considers the nonzero initial condition problems. Thus, Caputo definition is more suitable to deal with fractional-order differential equations with nonzero initial conditions.

If the initial value of function f(t) is nonzero, and $\alpha \in (0, 1)$, compared with the definitions of Caputo's and Riemann–Liouville's, it can be seen that

$${}_{t_0}^{\mathrm{C}}\mathscr{D}_t^{\alpha}f(t) = {}_{t_0}^{\mathrm{RL}}\mathscr{D}_t^{\alpha}(f(t) - f(t_0)), \qquad (11.9)$$

where the fractional-order derivative of constant $f(t_0)$ is

$${}^{\mathrm{RL}}_{t_0}\mathscr{D}^{\alpha}_t f(t_0) = \frac{f(t_0)(t-t_0)^{-\alpha}}{\Gamma(1-\alpha)}.$$
(11.10)

The relationship of these two definitions are

$${}_{t_0}^{\rm C} \mathscr{D}_t^{\alpha} f(t) = {}_{t_0}^{\rm RL} \mathscr{D}_t^{\alpha} f(t) - \frac{f(t_0)(t-t_0)^{-\alpha}}{\Gamma(1-\alpha)}.$$
(11.11)

More generally, if $\alpha > 1$, denote $m = \lceil \alpha \rceil$, then

$${}_{t_0}^{\rm C}\mathscr{D}_t^{\alpha}f(t) = {}_{t_0}^{\rm RL}\mathscr{D}_t^{\alpha}f(t) - \sum_{k=0}^{m-1} \frac{f^{(k)}(t_0)}{\Gamma(k-\alpha+1)} (t-t_0)^{k-\alpha},$$
(11.12)

and the relationship shown for $0 < \alpha < 1$ is just a special case of the formula.

11.1.3 Properties of Fractional-order Calculus

Some of the important properties of fractional-order calculus are summarized below without $\mathrm{proofs}^{[9]}$

(1) The fractional-order differentiation ${}_{t_0}\mathscr{D}_t^{\alpha}f(t)$ of an analytic function f(t) with respect to t is also analytic.

(2) If $\alpha = n$, the fractional-order derivative is identical to integer-order derivative, and also $t_0 \mathscr{D}_t^0 f(t) = f(t)$.

(3) The fractional-order differentiation is linear, i.e., for any constants c, d

$${}_{t_0}\mathscr{D}^{\alpha}_t \Big[cf(t) + dg(t) \Big] = c {}_{t_0}\mathscr{D}^{\alpha}_t f(t) + d {}_{t_0}\mathscr{D}^{\alpha}_t g(t).$$

$$(11.13)$$

(4) Fractional-order differentiation operators satisfy commutative-law, i.e.,

$${}_{t_0}\mathscr{D}_t^{\alpha}\left[{}_{t_0}\mathscr{D}_t^{\beta}f(t)\right] = {}_{t_0}\mathscr{D}_t^{\beta}\left[{}_{t_0}\mathscr{D}_t^{\alpha}f(t)\right] = {}_{t_0}\mathscr{D}_t^{\alpha+\beta}f(t).$$
(11.14)

(5) The Laplace transform of the fractional-order derivative is

$$\mathscr{L}\Big[{}_{t_0}\mathscr{D}^{\alpha}_t f(t)\Big] = s^{\alpha} \mathscr{L}\Big[f(t)\Big] - \sum_{k=1}^{n-1} s^k \Big[{}_{t_0}\mathscr{D}^{\alpha-k-1}_t f(t)\Big]_{t=t_0}.$$
(11.15)

Especially, if the initial values of the function f(t) and its derivatives at $t = t_0$ are all zero, $\mathscr{L}\left[t_0 \mathscr{D}_t^{\alpha} f(t)\right] = s^{\alpha} \mathscr{L}\left[f(t)\right]$. It is the same as the one in integer-order calculus, and it is the basis of fractional-order transfer functions, for the analysis and design of fractional-order systems.

11.2 Numerical Computations in Fractional-order Calculus

In this section, numerical computation of Grünwald–Letnikov and Caputo derivatives of given function f(t) are presented. Also, the computations of the important functions in fractional-order calculus, the Mittag–Leffler functions are presented.

11.2.1 Numerical Solutions with Grünwald–Letnikov Definition

With Grünwald–Letnikov definition, the following MATLAB function can be immediately written to evaluate the fractional-order differentiations^[10], which implements directly Eqn. (11.4).

```
function dy=glfdiff(y,t,gam)
h=t(2)-t(1); w=1; y=y(:); t=t(:); a0=y(1); dy(1)=0;
if a0~=0 & gam>0, dy(1)=sign(a0)*inf; end
for j=2:length(t), w(j)=w(j-1)*(1-(gam+1)/(j-1)); end
for i=2:length(t), dy(i)=w(1:i)*[y(i:-1:1)]/h^gam; end
```

The syntax of the function is $y_1 = \texttt{glfdiff}(y, t, \gamma)$, where y and t are the vectors composed of sample values of the given function and time, respectively. The vector t should be evenly spaced. The argument γ is the order. The γ th order derivative of y is returned in y_1 vector. If $y(t_0) \neq 0$, the first value in the derivative is set to infinity.

Example 11.1. In integer-order calculus, it is known that the derivative function of a step function is an impulse function, and first-order integral is straight line. The following statements can be used to calculate the 0.5th order derivative and integral, and the results are shown in Fig. 11-1.

```
>> t=0:0.01:5; u=ones(size(t));
y1=glfdiff(u,t,0.5); y2=glfdiff(u,t,-0.5);
plot(t,y1,'-',t,y2,'--')
```



Fig. 11-1 Fractional derivative and integral of a step function.

It can be seen that the fractional integrals are no longer straight lines, and the derivatives are no longer impulse functions. Gradual changes are observed in fractional-order integrals and derivatives. The fractional-order calculus is regarded as having memories.

Example 11.2. Consider the sinusoidal function $f(t) = \sin(3t + 1)$. Its 0.3th order derivative is shown in Fig. 11-2(a), and the surface plots for the derivatives of different orders are shown in Fig. 11-2(b).

```
>> t=0:0.01:5; u=sin(3*t+1); ww=0:0.1:1; Y=[];
y1=glfdiff(u,t,0.3); y2=3^0.3*sin(3*t+1+0.3*pi/2);
```



The α th order derivative under Cauchy integral formula is $3^{\alpha} \sin(3t + 1 + \alpha \pi/2)$. This derivative curve is also given in Fig. 11-2(a). It can be seen that the major difference between the two definitions is, in Grünwald–Letnikov definition, the initial values of f(t) is assumed to be zero for $t \leq 0$. Thus, the function jumped from 0 to $\sin 1$ at time $t = 0_+$. While in the Cauchy formula, the function at $t \leq 0$ time is still assumed to be $\sin(3t+1)$, thus there is no jump at $t = 0_+$.

It can be seen from the fractional-order derivatives of sinusoidal wave in Fig. 11-2(b) that gradual changes between sinusoidal and cosine waves are obtained, while in integer-order calculus, only sinusoidal and cosine waves are obtained. Thus, fractional-order calculus are more informative than the traditional integer-order calculus.

11.2.2 Numerical Solutions with Caputo Definition

From the relationships of the definitions, the following function can be used to calculate Caputo derivatives, through Eqn. (11.12).

```
function dy=caputo(y,t,gam,L,vec)
t0=t(1); dy=glfdiff(y,t,gam); if nargin<=3, L=10; end
if gam>0, m=ceil(gam); if gam<=1, vec=y(1); end
for k=0:m-1, dy=dy-vec(k+1)*(t-t0).^(k-gam)./gamma(k+1-gam); end
yy1=interp1(t(L+1:end),dy(L+1:end),t(2:L),'spline'); dy(2:L)=yy1;
end</pre>
```

The syntax of the function is $y_1 = \text{caputo}(y, t, \alpha, y_0, L)$, where if $\alpha \leq 0$, the results of Grünwald–Letnikov are returned directly; If $\alpha < 1$, the initial value of $y(t_0)$ is directly extracted from vector y. If $\alpha > 1$, $y_0 = [y(t_0), y'(t_0), \cdots, y^{(m-1)}(t_0)]$ should be provided, where $m = \lceil \alpha \rceil$. In numerical implementation, it is always found that the first few terms may have large errors, thus it is better to reconstruct the first L terms with interpolation approaches. The default value of L is 10, however, if the order of differentiation is high, L should be increased.

Example 11.3. Consider the function $f(t) = \sin(3t+1)$ studied in Example 11.2.

It can be seen that at t = 0, the initial value of the function is $f(0) = \sin 1$. The difference between the two definitions is $d(t) = t^{-0.3} \sin 1/\Gamma(0.7)$. The following statements can be used to find the 0.3th order derivatives using Grünwald–Letnikov definition and Caputo definition, as shown in Fig. 11-3(a). The difference is also shown.

>> t=0:0.01:pi; y=sin(3*t+1); d=t.^(-0.3)*sin(1)/gamma(0.7); y1=glfdiff(y,t,0.3); y2=caputo(y,t,0.3); plot(t,y1,t,y2,'--',t,d,':')



Since ${}_{0}^{C}\mathscr{D}_{t}^{2.3}y(t)$ is to be calculated, the initial values y'(0), y''(0) are needed. These values can be obtained with symbolic computation and convert the result back to double precision variables. The 1.3th and 2.3th order derivatives can also be obtained as shown in Fig. 11-3(b).

```
>> syms t; y=sin(3*t+1); y00=sin(1); y10=double(subs(diff(y,t),t,0));
y20=double(subs(diff(y,t,2),t,0)); t=0:0.01:pi; y=sin(3*t+1);
y1=caputo(y,t,1.3,[y00 y10],15); y2=caputo(y,t,2.3,[y00,y10,y20],40);
plotyy(t,y1,t,y2)
```

11.2.3 Mittag-Leffler Functions and Their Computations

We all know that the exponential function is very important in the solutions of integerorder systems. Like exponential function in integer-order systems, the so-called Mittag– Leffler can be regarded as an extension of exponential functions to fractional-order calculus.

1. Mittag–Leffler function with one parameter

The definition is

$$\mathscr{E}_{\alpha}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k+1)},\tag{11.16}$$

where $\alpha \in \mathbb{C}$. The convergent condition for the infinite series is $\mathscr{R}(\alpha) > 0$.

It is obvious that exponential function \mathbf{e}^z is a particular case of Mittag–Leffler function, with $\alpha=1.$

$$\mathscr{E}_1(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(k+1)} = \sum_{k=0}^{\infty} \frac{z^k}{k!} = e^z.$$
 (11.17)

Other particular cases of Mittag-Leffler functions can be derived

$$\mathscr{E}_2(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(2k+1)} = \sum_{k=0}^{\infty} \frac{(\sqrt{z})^{2k}}{(2k)!} = \cosh\sqrt{z},$$
(11.18)

$$\mathscr{E}_{1/2}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(k/2+1)} = e^{z^2} (1 + \operatorname{erf}(z)) = e^{z^2} \operatorname{erfc}(-z).$$
(11.19)

2. Mittag-Leffler function with two parameters

Two-parameter Mittag–Leffler function can be defined when 1 in the Γ function of the one-parameter Mittag–Leffler function is substituted by a free variable β

$$\mathscr{E}_{\alpha,\beta}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k + \beta)},\tag{11.20}$$

where $\alpha, \beta \in \mathbb{C}$, and the convergent conditions for $z \in \mathbb{C}$ are $\mathscr{R}(\alpha) > 0$ and $\mathscr{R}(\beta) > 0$. If $\beta = 1$, the two-parameter Mittag–Leffler function is changed to one-parameter Mittag–Leffler function, i.e.,

$$\mathscr{E}_{\alpha,1}(z) = \mathscr{E}_{\alpha}(z). \tag{11.21}$$

Besides, three- and four-parameter generalized Mittag-Leffler functions can also be defined^[11]. A MATLAB function ml_func() for the computation of generalized Mittag-Leffler functions and their integer-order derivatives^[5].

```
function f=ml_func(aa,z,n,eps0)
aa=[aa,1,1,1]; a=aa(1); b=aa(2); c=aa(3); q=aa(4);
f=0; k=0; fa=1; if nargin<4, eps0=eps; end</pre>
if nargin<3, n=0; end
if n==0
   while norm(fa,1)>=eps0
      fa=gamma(k*q+c)/gamma(c)/gamma(k+1)/gamma(a*k+b)*z.^k;
      f=f+fa; k=k+1;
   end
   if ~isfinite(f(1))
      if c==1 & q==1
         f=mlf(a,b,z,round(-log10(eps0))); f=reshape(f,size(z));
      else, error('Error: truncation method failed'); end, end
else
   aa(2)=aa(2)+n*aa(1); aa(3)=aa(3)+aa(4)*n;
   f=gamma(q*n+c)/gamma(c)*ml_func(aa,z,0,eps0);
end
```

The syntax of the function is $y=ml_func(v,z,n,\epsilon)$, where z is a vector, and the vector v can be set to $v = \alpha$ or $v = [\alpha,\beta]$, to indicate one- or two-parameter Mittag–Leffler functions. Vector v can also be assigned to a three or four element vector, $v = [\alpha, \beta, \gamma]$, or $v = [\alpha, \beta, \gamma, c]$, to indicate three- or four-parameter Mittag–Leffler functions. The argument n is the order of derivative for the Mittag–Leffler function. The argument ϵ is the error tolerance. Since truncation algorithm is used in the function, the speed of the function is relatively fast, however, the values may sometimes become infinite. In this

case, the mlf() function^[12] is embedded to tackle the problem. However, the speed may become very slow. The output vector \boldsymbol{y} is the Mittag-Leffler function.

Example 11.4. The following statements can be used to draw the Mittag–Leffler function $\mathscr{E}_1(-t)$, $\mathscr{E}_{3/2,3/2}(-t)$, and $\mathscr{E}_{1,2}(-t)$, as shown in Fig. 11-4. The exponential function is also drawn. It can be seen that $\mathscr{E}_1(-t)$ is the same as e^{-t} , and the decay rates of the other two curves are slower than exponential function.

```
>> t=0:0.1:5; y1=ml_func(1,-t); y2=ml_func([1,2],-t);
y3=ml_func([3/2,3/2],-t); plot(t,y1,t,y2,t,y3)
```



Fig. 11-4 Mittag–Leffler function curves.

11.3 Solutions of Linear Fractional-order Systems

Fractional-order systems are the direct extension of integer-order systems. In practical applications, there are some systems which can only be expressed accurately with fractional-order differential equations. In this section, analytical solutions for some linear fractional-order differential equations are first presented, and Mittag-Leffler functions are normally used to express the analytical solutions. For more linear fractional-order differential equation, closed-form numerical solution algorithms are presented.

11.3.1 Numerical Solutions of Linear Fractional-order Differential Equations

The typical form of linear fractional-order differential equation is given by

$$a_{1 t_0} \mathscr{D}^{\eta_1} y(t) + a_{2 t_0} \mathscr{D}^{\eta_2} y(t) + \dots + a_{n-1 t_0} \mathscr{D}^{\eta_{n-1}} y(t) + a_{n t_0} \mathscr{D}^{\eta_n} y(t) = b_{1 t_0} \mathscr{D}^{\gamma_1} u(t) + b_{2 t_0} \mathscr{D}^{\gamma_2} u(t) + \dots + b_{m t_0} \mathscr{D}^{\gamma_m} u(t),$$
(11.22)

where b_i and a_i are real coefficients, and γ_i and η_i are orders.

Let us consider a simpler differential equation

$$a_{1 t_0} \mathscr{D}_t^{\eta_1} y(t) + a_{2 t_0} \mathscr{D}_t^{\eta_2} y(t) + \dots + a_{n-1 t_0} \mathscr{D}_t^{\eta_{n-1}} y(t) + a_n \mathscr{D}_t^{\eta_n} y(t) = u(t), \qquad (11.23)$$

where u(t) is a given function. The Grünwald–Letnikov definition given in Eqn. (11.4) can be used directly, and the closed-form solution to the original differential equation is written as

$${}_{t_0}\mathcal{D}_t^{\eta_i}y(t) \approx \frac{1}{h^{\eta_i}} \sum_{j=0}^{[(t-t_0)/h]} w_j^{(\eta_i)}y_{t-jh} = \frac{1}{h^{\eta_i}} \left[y_t + \sum_{j=1}^{[(t-t_0)/h]} w_j^{(\eta_i)}y_{t-jh} \right], \quad (11.24)$$

where $w_0^{(\beta_i)}$ can be evaluated recursively from Eqn. (11.3). Substitute it back to the original equation, the closed-form solution of the differential equation can be written as^[10]

$$y_t = \frac{1}{\sum_{i=1}^n \frac{a_i}{h^{\eta_i}}} \left[u_t - \sum_{i=1}^n \frac{a_i}{h^{\eta_i}} \sum_{j=1}^{[(t-t_0)/h]} w_j^{(\eta_i)} y_{t-jh} \right].$$
 (11.25)

For the differential equation in Eqn. (11.22), the equivalent input u(t) can be calculated first, then Eqn. (11.25) can be used to solve the numerical solution of the original equation. The following MATLAB function can be written

```
function y=fode_sol(a,na,b,nb,u,t)
h=t(2)-t(1); D=sum(a./[h.^na]); nT=length(t);
vec=[na nb]; W=[]; D1=b(:)./h.^nb(:); nA=length(a);
y1=zeros(nT,1); W=ones(nT,length(vec));
for j=2:nT, W(j,:)=W(j-1,:).*(1-(vec+1)/(j-1)); end
for i=2:nT,
    A=[y1(i-1:-1:1)]'*W(2:i,1:nA);
    y1(i)=(u(i)-sum(A.*a./[h.^na]))/D;
end
for i=2:nT, y(i)=(W(1:i,nA+1:end)*D1)'*[y1(i:-1:1)]; end
```

The syntax of the function is $y = \text{fode_sol}(a, \eta, b, \gamma, u, t)$, where time vector t and input vector u should be given, and y returns the numerical solution, vectors a and η are the coefficients and orders of output y(t) in the equation, while b and γ are the coefficients and orders of input signal u(t).

Example 11.5. Consider the linear fractional-order differential equation

 $\mathscr{D}^{1.6}y(t) + 10\mathscr{D}^{1.2}y(t) + 35\mathscr{D}^{0.8}y(t) + 50\mathscr{D}^{0.4}y(t) + 24y(t) = \mathscr{D}^{1.2}u(t) + 3\mathscr{D}^{0.4}u(t) + 5u(t),$

with zero initial conditions. Assume the input u(t) is unit step signal. The following statements can be used to solve the differential equation, and the solution is shown in Fig. 11-5.

>> a=[1,10,35,50,24]; na=[1.6 1.2 0.8 0.4 0]; b=[1 3 5]; nb=[1.2 0.4 0]; t=0:0.01:10; u=ones(size(t)); y=fode_sol(a,na,b,nb,u,t); plot(t,y)

Analysis and Design of Fractional-order Systems



Fig. 11-5 Numerical solution of the different equation under unit step input.

11.3.2 Numerical Solutions of Caputo Differential Equations

If the initial values of the input and output signals are not zero, Caputo definition should be used, and we refer the fractional-order differential equations as Caputo differential equations.

Consider the Caputo linear differential equation

$$a_{n} {}_{t_{0}}^{\mathrm{C}} \mathscr{D}_{t}^{\beta_{n}} y(t) + a_{n-1} {}_{t_{0}}^{\mathrm{C}} \mathscr{D}_{t}^{\beta_{n-1}} y(t) + \dots + a_{1} {}_{t_{0}}^{\mathrm{C}} \mathscr{D}_{t}^{\beta_{1}} y(t) + a_{0} {}_{t_{0}}^{\mathrm{C}} \mathscr{D}_{t}^{\beta_{0}} y(t) = \hat{u}(t).$$
(11.26)

For convenience, assume that $\beta_n > \beta_{n-1} > \cdots > \beta_1 > \beta_0 \ge 0$. The right-handside contains only $\hat{u}(t)$ function. For equations with linear combinations of u(t) and its derivatives, the right-hand-side $\hat{u}(t)$ should be calculated first.

If $m = \lceil \beta_n \rceil$, *m* initial values, $y(t_0), y'(t_0), \dots, y^{(m-1)}(t_0)$, are expected to uniquely solve the Caputo differential equations. Thus, the auxiliary variable z(t) can be introduced

$$z(t) = y(t) - y(t_0) - y'(t_0) t - \dots - y^{(m-1)}(t_0) t^{m-1}.$$
 (11.27)

The initial values of the first (m - 1)th order derivatives of z(t) are zero. Slightly change the form of the expression yield

$$y(t) = z(t) + y(t_0) + y'(t_0)t + \dots + y^{(m-1)}(t_0)t^{m-1}.$$
(11.28)

Since the initial values of z(t) are zeros, ${}_{t_0}^{C} \mathscr{D}_t^{\beta_i} z(t) = {}_{t_0}^{RL} \mathscr{D}_t^{\beta_i} z(t)$. The β_i th order Caputo derivative of the polynomial $y(t_0) + y'(t_0) t + \cdots + y^{(m-1)}(t_0) t^{m-1}$ can be obtained with the following function

function s=poly2caputo(a,r), syms u tau; s=int(diff(poly2sym(a,'tau'),ceil(r))/((u-tau)^(r-ceil(r)+1))... /gamma(ceil(r)-r),tau,0,u);

The syntax of the function is $s = poly2caputo(a, \beta)$, where a is the initial condition vector $a = [y^{(m-1)}(t_0), \dots, y'(t_0), y(t_0)]$, β is the order of differentiation. The returned variable s is symbolic expression of the polynomial, with independent variable u.

With the compensation function described earlier, the original differential equation

can be converted to

$$a_{n} {}_{t_{0}}^{\mathrm{RL}} \mathscr{D}_{t}^{\beta_{n}} y(t) + a_{n-1} {}_{t_{0}}^{\mathrm{RL}} \mathscr{D}_{t}^{\beta_{n-1}} y(t) + \dots + a_{1} {}_{t_{0}}^{\mathrm{RL}} \mathscr{D}_{t}^{\beta_{1}} y(t) + a_{0} {}_{t_{0}}^{\mathrm{RL}} \mathscr{D}_{t}^{\beta_{0}} y(t)$$

$$= \hat{u}(t) - \sum_{i=0}^{n} a_{i} {}_{t_{0}}^{C} \mathscr{D}^{\beta_{i}} \Big[y(t_{0}) + y'(t_{0})t + \dots + y^{(m-1)}(t_{0})t^{m-1} \Big].$$
(11.29)

Similar to fode_sol() function, the following MATLAB function can be written to solve Caputo differential equation. Currently, this function can only be used in solving differential equations with right-hand-side of the equation contains only u(t). The syntax of the function is $y = \text{fode_caputo}(a, n_a, y_0, u, t)$, where u is the sample of the input signal, and t is the evenly spaced time vector.

```
function [y,z]=fode_caputo(a,na,y0,u,t)
h=t(2)-t(1); D=sum(a./[h.^na]); nT=length(t); nb=0; b=1;
vec=[na nb]; W=[]; D1=b(:)./h.^nb(:); nA=length(a);
y1=zeros(nT,1); W=ones(nT,length(vec));
for i=1:length(a), u=u-a(i)*subs(poly2caputo(y0,na(i)),'u',t); end
for j=2:nT, W(j,:)=W(j-1,:).*(1-(vec+1)/(j-1)); end
for i=2:nT,
    A=[y1(i-1:-1:1)]'*W(2:i,1:nA); y1(i)=(u(i)-sum(A.*a./[h.^na]))/D;
end
z=y1'; y=z+polyval(y0,t);
```

Example 11.6. Consider the Caputo linear differential equation

$$\mathscr{D}_{t}^{3.5}y(t) + 8\mathscr{D}_{t}^{3.1}y(t) + 26\mathscr{D}_{t}^{2.3}y(t) + 73\mathscr{D}_{t}^{1.2}y(t) + 90\mathscr{D}_{t}^{0.5}y(t) = 90\sin t^{2},$$

with initial conditions, y(0) = 1, y'(0) = -1, y''(0) = 2, and y'''(0) = 3.

The initial condition vector can be entered, and with the following statements, the Caputo differential equation can be solved, as shown in Fig. 11-6.

```
>> a=[1,8,26,73,90]; n=[3.5,3.1,2.3,1.2,0.5]; t=0:0.001:10;
u=90*sin(t.^2); y0=[3 2 -1 1]; y=fode_caputo(a,n,y0,u,t); plot(t,y)
```



Fig. 11-6 Numerical solutions of Caputo differential equations.

11.3.3 Some Important Laplace Transforms

In the analytical solution approach presented later, some of the important Laplace transforms are needed. Here, we list some of the useful formula. All the Laplace transforms presented later are the variations of the following formula^[11, 13]

$$\mathscr{L}^{-1}\left[\frac{s^{\alpha\gamma-\beta}}{(s^{\alpha}+a)^{\gamma}}\right] = t^{\beta-1}\mathscr{E}^{\gamma}_{\alpha,\beta}\left(-at^{\alpha}\right).$$
(11.30)

For different values of the parameters, the following formula can be derived

(1) If $\gamma = 1$, and $\alpha \gamma = \beta$, or $\beta = \alpha$, the above formula can be written as

$$\mathscr{L}^{-1}\left[\frac{1}{s^{\alpha}+a}\right] = t^{\alpha-1}\mathscr{E}_{\alpha,\alpha}\left(-at^{\alpha}\right).$$
(11.31)

This formula can be regarded as the analytical solution of the impulse response of the fractional-order transfer function $1/(s^{\alpha}+a)$. It can be seen that the essential representation in fractional-order system is Mittag–Leffler function, just as the exponential function in integer-order systems.

(2) If $\gamma = 1$, and $\alpha \gamma - \beta = -1$, or $\beta = \alpha + 1$, Eqn. (11.30) can be written as

$$\mathscr{L}^{-1}\left[\frac{1}{s(s^{\alpha}+a)}\right] = t^{\alpha}\mathscr{E}_{\alpha,\alpha+1}\left(-at^{\alpha}\right).$$
(11.32)

The formula can be regarded as the analytical solution of the step response of the transfer function $1/(s^{\alpha} + a)$. The above equation can also be written as

$$\mathscr{L}^{-1}\left[\frac{1}{s(s^{\alpha}+a)}\right] = \frac{1}{a}\left[1 - \mathscr{E}_{\alpha}\left(-at^{\alpha}\right)\right].$$
(11.33)

(3) If $\gamma = k$ is an integer, and $\alpha \gamma = \beta$, i.e., $\beta = \alpha k$, Eqn. (11.30) is written as

$$\mathscr{L}^{-1}\left[\frac{1}{(s^{\alpha}+a)^{k}}\right] = t^{\alpha k-1} \mathscr{E}^{k}_{\alpha,\alpha k}\left(-at^{\alpha}\right).$$
(11.34)

and the formula can be regarded as the impulse response of $1/(s^{\alpha} + a)^k$.

(4) If $\gamma = k$ is an integer, and $\alpha \gamma - \beta = -1$, i.e., $\beta = \alpha k + 1$, Eqn. (11.30) can be written as

$$\mathscr{L}^{-1}\left[\frac{1}{s(s^{\alpha}+a)^{k}}\right] = t^{\alpha k} \mathscr{E}^{k}_{\alpha,\alpha k+1}\left(-at^{\alpha}\right).$$
(11.35)

which can be regarded as the unit step response of $1/(s^{\alpha} + a)^k$.

11.3.4 Analytical Solutions of Commensurate-order Linear Differential Equations

Consider the orders in Eqn. (11.22). If a greatest common divisor α can be found among the orders, such that the original equation can be written as

$$a_{1}\mathscr{D}_{t}^{n\alpha}y(t) + a_{2}\mathscr{D}_{t}^{(n-1)\alpha}y(t) + \dots + a_{n}\mathscr{D}_{t}^{\alpha}y(t) + a_{n+1}y(t) = b_{1}\mathscr{D}_{t}^{m\alpha}v(t) + b_{2}\mathscr{D}_{t}^{(m-1)\alpha}v(t) + \dots + b_{m}\mathscr{D}_{t}^{\alpha}v(t) + b_{m+1}v(t),$$
(11.36)

the original differential equation is referred to as commensurate-order differential equations of the base order α . Denote $\lambda = s^{\alpha}$, the original differential equation can be expressed by the integer-order transfer function of λ . If there is no repeated poles in the system, the original transfer function can be written as the following form with the partial fraction expansion technique

$$G(\lambda) = \sum_{i=1}^{n} \frac{r_i}{\lambda + p_i} = \sum_{i=1}^{n} \frac{r_i}{s^{\alpha} + p_i}.$$
 (11.37)

From the formula of Laplace transform given in Eqns. (11.31) and (11.32), the analytical solution of the impulse and step responses of the system can be obtained as

$$\mathscr{L}^{-1}\left[\sum_{i=1}^{n} \frac{r_i}{s^{\alpha} + p_i}\right] = \sum_{i=1}^{n} r_i t^{\alpha - 1} \mathscr{E}_{\alpha, \alpha}\left(-p_i t^{\alpha}\right),\tag{11.38}$$

$$\mathscr{L}^{-1}\left[\sum_{i=1}^{n} \frac{r_i}{s(s^{\alpha} + p_i)}\right] = \sum_{i=1}^{n} r_i t^{\alpha} \mathscr{E}_{\alpha,\alpha+1}\left(-p_i t^{\alpha}\right).$$
(11.39)

The latter can alternatively written as

$$\mathscr{L}^{-1}\left[\sum_{i=1}^{n} \frac{r_i}{s(s^{\alpha} + p_i)}\right] = \sum_{i=1}^{n} \frac{r_i}{p_i} \left[1 - \mathscr{E}_{\alpha}\left(-p_i t^{\alpha}\right)\right].$$
(11.40)

If the system has repeated poles, Eqns. (11.34) and (11.35) should be used to write the analytical solutions to impulse and step responses.

Example 11.7. Consider the fractional-order differential equation

$$\mathscr{D}^{1.2}y(t) + 5\mathscr{D}^{0.9}y(t) + 9\mathscr{D}^{0.6}y(t) + 7\mathscr{D}^{0.3}y(t) + 2y(t) = u(t),$$

with zero initial conditions, where u(t) is the unit step signal. If base order is selected as $\lambda = s^{0.3}$, the transfer function can be written as

$$G(\lambda) = \frac{1}{\lambda^4 + 5\lambda^3 + 9\lambda^2 + 7\lambda + 2}$$

The following MATLAB statements can be used to find its partial fraction expansion regarding to λ ,

>> num=1; den=[1 5 9 7 2]; [r,p]=residue(num,den) and the results can be obtained as

$$G(\lambda) = -\frac{1}{\lambda+2} + \frac{1}{\lambda+1} - \frac{1}{(\lambda+1)^2} + \frac{1}{(\lambda+1)^3}$$

and the Laplace transform of the output signal can be written as

$$Y(s) = \frac{1}{s}G(\lambda) = -\frac{1}{s(s^{0.3}+2)} + \frac{1}{s(s^{0.3}+1)} - \frac{1}{s(s^{0.3}+1)^2} + \frac{1}{s(s^{0.3}+1)^3}.$$

Thus, the analytical solution to the step input can be obtained as

$$y(t) = -t^{0.3} \mathscr{E}_{0.3,1.3} \left(-2t^{0.3} \right) + t^{0.3} \mathscr{E}_{0.3,1.3} \left(-t^{0.3} \right) - t^{0.6} \mathscr{E}_{0.3,1.6}^2 \left(-t^{0.3} \right) + t^{0.9} \mathscr{E}_{0.3,1.9}^3 \left(-t^{0.3} \right) \\ = \frac{1}{2} + \frac{1}{2} \mathscr{E}_{0.3} \left(-2t^{0.3} \right) + \mathscr{E}_{0.3} \left(-t^{0.3} \right) - \left[1 - \mathscr{E}_{0.3} \left(-t^{0.3} \right) \right]^2 + \left[1 - \mathscr{E}_{0.3} \left(-t^{0.3} \right) \right]^3.$$

11.3.5 Analytical Solutions of Linear Fractional-order Differential Equations

Consider the following (n + 1)-term fractional-order differential equation

$$a_n \mathscr{D}_t^{\beta_n} y(t) + a_{n-1} \mathscr{D}_t^{\beta_{n-1}} y(t) + \dots + a_0 \mathscr{D}_t^{\beta_0} y(t) = u(t), \qquad (11.41)$$

with step input. The analytical solution can be written as

$$y(t) = \frac{1}{a_n} \sum_{m=0}^{\infty} \frac{(-1)^m}{m!} \sum_{\substack{k_0 + k_1 + \dots + k_{n-2} = m \\ k_0 \ge 0, \ \dots, \ k_{n-2} \ge 0}} (m; k_0, k_1, \dots, k_{n-2})$$

$$\prod_{i=0}^{n-2} \left(\frac{a_i}{a_n}\right)^{k_i} t^{(\beta_n - \beta_{n-1})m + \beta_n + \sum_{j=0}^{n-2} (\beta_{n-1} - \beta_j)k_j - 1} (11.42)$$

$$\mathscr{E}^{(m)}_{\beta_n - \beta_{n-1}, \ \beta_n + \sum_{j=0}^{n-2} (\beta_{n-1} - \beta_j)k_j} \left(-\frac{a_{n-1}}{a_n} t^{\beta_n - \beta_{n-1}}\right),$$

where $(m; k_0, k_1, \cdots, k_{n-2})$ is defined as

$$(m; k_0, k_1, \cdots, k_{n-2}) = \frac{m!}{k_0! k_1! \cdots k_{n-2}!}.$$
(11.43)

It is difficult to write out analytical solutions to a certain system, since the solution form is too complicated. This method is quite useful in practical problems.

11.4 Modeling and Analysis of Fractional-order Transfer Functions

Consider the linear fractional-order system shown in Eqn. (11.22). If the initial values of the input signal u(t) and output y(t) and their derivatives are all zero, with Laplace transform, the fractional-order transfer function when appended a time delay T can be written as

$$G(s) = \frac{b_1 s^{\gamma_1} + b_2 s^{\gamma_2} + \dots + b_m s^{\gamma_m}}{a_1 s^{\eta_1} + a_2 s^{\eta_2} + \dots + a_{n-1} s^{\eta_{n-1}} + a_n s^{\eta_n}} e^{-Ts}.$$
 (11.44)

Compared with the integer-order transfer functions, apart from the numerator and denominator coefficients, the orders can also be declared. Thus, normally four vectors and a delay constant can be used to describe uniquely the fractional-order transfer function model in Eqn. (11.44).

Since this model is useful in the analysis and design of linear fractional-order systems, we can construct a MATLAB class FOTF to describe the model as it is done in TF class, in Control System Toolbox. When the class is created, overload functions can be written to implement the modeling, analysis and design tasks, in a simple and straightforward way.

In this section, creation of class, and object-oriented programming technique are demonstrated first, then based on the class, the modeling and analysis of fractional-order transfer functions are presented.

11.4.1 FOTF — Creation of a MATLAB Object

If one wants to create a MATLAB class, a name of the class should be selected. For instance, for fractional-order transfer function, we selected FOTF as its name. A folder **@fotf** should be created for it, and all the files related io the class should be placed in the folder. Normally for a new class, at least two functions should be written, **fotf.m** is used to define the class, and **display.m** is used to display the class. The programming of the two functions and other supporting functions are illustrated below:

(1) **Defining FOTF class**. A function fotf.m should be written and placed in the **@fotf** folder. The listing of the function is

```
function G=fotf(a,na,b,nb,T)
if nargin==0,
    G.a=[]; G.na=[]; G.b=[]; G.nb=[]; G.ioDelay=0; G=class(G,'fotf');
elseif isa(a,'fotf'), G=a;
elseif nargin==1 & isa(a,'double'), G=fotf(1,0,a,0,0);
elseif nargin==1 & a=='s', G=fotf(1,0,1,1,0);
else, ii=find(abs(a)<eps); a(ii)=[]; na(ii)=[];
    ii=find(abs(b)<eps); b(ii)=[]; nb(ii)=[];
    if nargin==5, G.ioDelay=T; else, G.ioDelay=0; end
    G.a=a; G.na=na; G.b=b; G.nb=nb; G=class(G,'fotf');
end</pre>
```

The command $G=\texttt{fotf}(a, n_a, b, n_b, T)$ can be used to enter a FOTF object, where $a = [a_1, a_2, \dots, a_n], b = [b_1, b_2, \dots, b_m], n_a = [\eta_1, \eta_2, \dots, \eta_n]$ and $n_b = [\gamma_1, \gamma_2, \dots, \gamma_m]$ can be used to represent the coefficients and orders of the numerator and denominator of the system, and T is the delay constant. If there is no delay in the model, the variable can be omitted.

Similar to tf() function, function s = fotf('s') command can be used to declare an *s* operator for the fractional-order model. The command G = fotf(k) can be used to convert a constant to FOTF object. If *G* is an LTI object in Control System Toolbox, the command G = fotf(G) can be used to convert the TF object into an FOTF object.

(2) Writing display function. Another function, display.m, should be written in that folder, which is used to display the FOTF object, once it is created. The listing of the function is

```
function display(G)
strN=fpoly2str(G.b,G.nb); strD=fpoly2str(G.a,G.na);
nn=length(strN); nd=length(strD); nm=max([nn,nd]);
disp([char(' '*ones(1,floor((nm-nn)/2))) strN]), ss=[];
T=G.ioDelay; if T>0, ss=[' exp(-' num2str(T) 's)']; end
disp([char(' -'*ones(1,nm)), ss]);
disp([char(' '*ones(1,floor((nm-nd)/2))) strD])
function strP=fpoly2str(p,np)
if isempty(np), p=0; np=0; end
P=''; [np,ii]=sort(np,'descend'); p=p(ii);
for i=1:length(p), P=[P,num2str(p(i)),'s^',num2str(np(i)),'+']; end
P=strrep(strrep(strrep(P,'s^0+','+'),'s^1+','s+'),'s^1-','s-');
P=strrep(strrep(strrep(P,'+-','-'),'+1s','+s'),'-1s','-s');
```

```
strP=P(1:end-1); nP=length(strP);
if nP>=2 & strP(1:2)=='1s', strP=strP(2:end); end
```

Example 11.8. For the fractional-order transfer function model

$$G(s) = \frac{0.8s^{1.2} + 2}{1.1s^{1.8} + 1.9s^{0.5} + 0.4} e^{-0.5s},$$

the following MATLAB commands can be used to directly enter the model. The display result is displayed as

>> G=fotf([1.1,1.9,0.4],[1.8,0.5,0],[0.8,2],[1.2,0],0.5)

It should be noted that these files must be placed in **@fotf** folder, and should not be placed elsewhere. Otherwise, the files cannot be called, and they may affect the existing MATLAB files with the same names.

(3) Other facilities. Further, apart from the two essential functions fotf.m and display.m, if we want to access the members in the FOTF object directly with MATLAB, the following two files are written. With these functions, commands like G.nb and G.na = [0.1, 0.2] are supported

```
function A=subsasgn(G,index,InputVal)
switch index.subs
    case {'a','na','b','nb','ioDelay'},
        eval(['G.' index.subs,'=InputVal;']);
        if length(G.a)~=length(G.na) | length(G.b)~=length(G.nb)
            error('Error: field pairs (na,a) or (nb,b) mismatched.')
        else, A=fotf(G.a,G.na,G.b,G.nb,G.ioDelay); end
        otherwise, error('Error: Available fields are a, na, b, na, ioDelay.');
end
function A=subsref(G,index)
switch index.subs
    case {'a','na','b','nb','ioDelay'}, A=eval(['G.' index.subs]);
    otherwise,
        error('Error: Available fields are a, na, b, na, ioDelay.');
end
```

(4) Conversion function from TF to FOTF object. The following function should be placed in the Qtf folder.

```
function G1=fotf(G)
[n,d]=tfdata(tf(G),'v'); nn=length(n)-1:-1:0;
nd=length(d)-1:-1:0; G1=fotf(d,nd,n,nn,G.ioDelay);
```

11.4.2 Interconnections of FOTF Blocks

It has been shown in Chapter 4 that integer-order models can be calculated with +, * and feedback() functions to process the parallel, series and feedback connection. Similar to the idea, the following overload functions can be written. These files should be placed in the @fotf folder. Most of the functions are from the book [5], however, some of them are modified and extended.

(1) Multiplications of FOTF blocks. To define the expression $G = G_1 * G_2$, the overload function mtimes() should be written. This function is used to evaluate the series connection of two FOTF blocks, $G_1(s)$ and $G_2(s)$, the algorithm is

$$G(s) = G_1(s)G_2(s) = \frac{N_1(s)N_2(s)}{D_1(s)D_2(s)}.$$
(11.45)

The overload function can be written as

```
function G=mtimes(G1,G2)
G1=fotf(G1); G2=fotf(G2); na=[]; nb=[];
a=kron(G1.a,G2.a); b=kron(G1.b,G2.b);
for i=1:length(G1.na), na=[na,G1.na(i)+G2.na]; end
for i=1:length(G1.nb), nb=[nb,G1.nb(i)+G2.nb]; end
G=simple(fotf(a,na,b,nb,G1.ioDelay+G2.ioDelay));
```

(2) Adding FOTF blocks. The expression $G = G_1 + G_2$ should be described by the overload function plus() to evaluate the parallel connection of two FOTF blocks, with the algorithm

$$G(s) = G_1(s) + G_2(s) = \frac{N_1(s)D_2(s) + N_2(s)D_1(s)}{D_1(s)D_2(s)}.$$
(11.46)

The following overload function is implemented

```
function G=plus(G1,G2)
G1=fotf(G1); G2=fotf(G2); na=[]; nb=[];
if G1.ioDelay==G2.ioDelay
    a=kron(G1.a,G2.a); b=[kron(G1.a,G2.b),kron(G1.b,G2.a)];
    for i=1:length(G1.a),
        na=[na G1.na(i)+G2.na]; nb=[nb, G1.na(i)+G2.nb];
    end
    for i=1:length(G1.b), nb=[nb G1.nb(i)+G2.na]; end
    G=simple(fotf(a,na,b,nb,G1.ioDelay));
else, error('cannot handle different delays'); end
```

(3) Feedback function. The function $G = \text{feedback}(G_1, G_2)$ evaluates the negative connection of the two FOTF blocks. If the positive structure is used, the forward path can be converted to $-G_2$, such that negative feedback structure can still be used.

$$G(s) = \frac{G_1(s)}{1 + G_1(s)G_2(s)} = \frac{N_1(s)D_2(s)}{D_1(s)D_2(s) + N_1(s)N_2(s)}.$$
(11.47)

The listing of the overload function is

```
function G=feedback(F,H)
F=fotf(F); H=fotf(H); na=[]; nb=[];
if F.ioDelay==H.ioDelay
    b=kron(F.b,H.a); a=[kron(F.b,H.b), kron(F.a,H.a)];
    for i=1:length(F.b),
        nb=[nb F.nb(i)+H.nb]; na=[na,F.nb(i)+H.nb];
    end
    for i=1:length(F.a), na=[na F.na(i)+H.na]; end
```

G=simple(fotf(a,na,b,nb,F.ioDelay)); else, error('cannot handle blocks with different delays'); end

(4) Simple supporting functions. Function uminus() is used to evaluate $G_1(s) = -G(s)$, with the syntax $G_1 = -G$; Function $G = inv(G_1)$ is used to evaluate $G(s) = 1/G_1(s)$; Function minus() is used to evaluate $G(s) = G_1(s) - G_2(s)$, with $G = G_1 - G_2$. Function eq() judges whether two FOTF blocks G_1 and G_2 equal or not, with $key = G_1 = = G_2$, if equal, the returned key is 1.

```
function G=uminus(G1), G=G1; G.b=-G.b;
function G=inv(G1), G=fotf(G1.b,G1.nb,G1.a,G1.na,-G1.ioDelay);
function G=minus(G1,G2), G=G1+(-G2);
function key=eq(G1,G2), key=0; G=G1-G2;
if length(G.nb)==0 | norm(G.b)<1e-10, key=1; end
(5) Right division. With G=G_1/G_2 to evaluates G(s) = G_1(s)/G_2(s)
function G=mrdivide(G1,G2)
G1=fotf(G1); G2=fotf(G2); G=G1*inv(G2);
G.ioDelay=G1.ioDelay-G2.ioDelay;
if G.ioDelay<0, warning('block with positive delay'); end
```

(6) **Power function**. With $G = G_1 n$, the power of G_1 can be evaluated, if n is integer. Otherwise, only the case G_1 is a Laplace operator can be handled.

```
function G1=mpower(G,n)
if n==fix(n),
    if n>=0, G1=1; for i=1:n, G1=G1*G; end
    else, G1=inv(G^(-n)); end, G1.ioDelay=n*G.ioDelay;
elseif G==fotf(1,0,1,1), G1=fotf(1,0,1,n);
else, error('mpower: power must be an integer.'); end
```

(7) Simplification function. With G = simple(G), the coefficients of numerator and denominator are collected to simplify the description. Sub-function polyuniq() can be used to collect coefficients of polynomials. The sub function cannot be called directly.

Example 11.9. The following commands can be used to express the fractional-order PID controller $G_c(s) = 5 + 2s^{-0.2} + 3s^{0.6}$ into MATLAB workspace

```
>> s=fotf('s'); Gc=5+2*s^(-0.2)+3*s^0.6
```

Example 11.10. The fractional-order transfer function model

$$G(s) = \frac{(s^{0.3} + 3)^2}{(s^{0.2} + 2)(s^{0.4} + 4)(s^{0.4} + 3)},$$

can be entered into MATLAB workspace, with the following MATLAB statements, the expanded model can be obtained

$$G(s) = \frac{s^{0.6} + 6s^{0.3} + 9}{s + 2s^{0.8} + 7s^{0.6} + 14s^{0.4} + 12s^{0.2} + 24}$$

>> s=fotf('s'); G=(s^0.3+3)^2/(s^0.2+2)/(s^0.4+4)/(s^0.4+3)

Example 11.11. Assume that typical unity negative feedback control system is

$$G(s) = \frac{0.8s^{1.2} + 2}{1.1s^{1.8} + 0.8s^{1.3} + 1.9s^{0.5} + 0.4}, \ G_{\rm c}(s) = \frac{1.2s^{0.72} + 1.5s^{0.33}}{3s^{0.8}},$$

the following statements can be used to enter the model into MATLAB workspace

>> G=fotf([1.1,0.8 1.9 0.4],[1.8 1.3 0.5 0],[0.8 2],[1.2 0]);

Gc=fotf([3],[0.8],[1.2 1.5],[0.72 0.33]); GG=feedback(G*Gc,1)

and the closed-loop model obtained is

$$G(s) = \frac{0.96s^{1.59} + 1.2s^{1.2} + 2.4s^{0.39} + 3}{3.3s^{2.27} + 2.4s^{1.77} + 0.96s^{1.59} + 1.2s^{1.2} + 5.7s^{0.97} + 1.2s^{0.47} + 2.4s^{0.39} + 3}.$$

11.4.3 Analysis of FOTF Objects

1. Stability analysis

The stability assessment of commensurate-order systems can be carried out directly. If the base order of the commensurate-order system is $\lambda = s^{\alpha}$, the stable regions for the commensurate-order system are shown in Fig. 11-7. If the poles of the system of λ are located in the stable regions, then the system is stable, otherwise the system is unstable^[14]. For the base order α , the boundaries of stable regions are the straight lines with slopes of $\pm \alpha \pi/2$. When the base order is $\alpha = 1$, the system is of integer-order, and the stable boundary is changed to the imaginary axis, which agrees well with the cases in integer-order systems.



Fig. 11-7 Stable regions for commensurate-order systems.

Based on the idea, the following MATLAB function can be written. The function can be used to convert FOTF object to commensurate-order system first, and then the poles

of the system can be evaluated. Although sometimes the order of the commensurateorder system is extremely high, it can also be processed by MATLAB easily. The syntax $[key, \alpha, \epsilon, a_1] = isstable(G, a_0)$ can be used to assess the stability of the FOTF object, where key is one for stable. The argument α returns the base order, ϵ is the error tolerance in root finding, a_1 is the slopes of $\pm \alpha \pi/2$, and a_0 is the user selected base order, with default of 0.01.

```
function [K,alpha,err,apol]=isstable(G,a0)
if nargin==1, a0=0.01; end
a=G.na; a1=fix(a/a0); n=gcd(a1(1),a1(2));
for i=3:length(a1), n=gcd(n,a1(i)); end
alpha=n*a0; a=fix(a1/n); b=G.a; c(a+1)=b; c=c(end:-1:1);
p=roots(c); p=p(abs(p)>eps); err=norm(polyval(c,p));
plot(real(p),imag(p),'x',0,0,'o')
apol=min(abs(angle(p))); K=apol>alpha*pi/2;
xm=xlim; xm(1)=0; line(xm,tan(alpha*pi/2)*xm)
title('Pole-Zero Map'), xlabel('Real Axis'), ylabel('Imaginary Axis')
```

Example 11.12. Consider the fractional-order transfer function

$$G(s) = \frac{-2s^{0.63} - 4}{2s^{3.501} + 3.8s^{2.42} + 2.6s^{1.798} + 2.5s^{1.31} + 1.5}.$$

The following statements can be used to enter the model into MATLAB workspace first, then assess the stability of the system.

>> b=[-2,-4]; nb=[0.63,0]; a=[2,3.8,2.6,2.5,1.5]; na=[3.501,2.42,1.798,1.31,0]; G=fotf(a,na,b,nb); [key,alpha,err,apol]=isstable(G,0.001)

It is obvious that the base order is $\alpha = 0.001$, the commensurate-order model of the system can be rewritten as an integer-order transfer function of λ

$$G(\lambda) = \frac{-2\lambda^{630} - 4}{2\lambda^{3501} + 3.8\lambda^{2420} + 2.6\lambda^{1798} + 2.5\lambda^{1310} + 1.5}.$$

The roots of the polynomials of λ can be obtained automatically in the function, as shown in Fig. 11-8(a). The zoomed plots around the *x*-axis can be obtained in Fig. 11-8(b). It can be seen that all the poles of the system are located in the stable regions. Thus, the system is stable. Since $\alpha = 0.001$, the polynomial is of 3501th order. It may take some time to find all the poles.

2. Norms of fractional-order systems

The norms of the systems are important quantities in robust control design. The evaluation algorithms of the norms of fractional-order systems are illustrated here. The \mathcal{H}_2 and \mathcal{H}_∞ norms of G(s) are defined respectively as

$$||G(s)||_{2} = \sqrt{\frac{1}{2\pi j} \int_{-j\infty}^{j\infty} G(s)G(-s)ds},$$
(11.48)

$$||G(s)||_{\infty} = \sup_{\omega} |G(j\omega)|.$$
(11.49)

Pole-Zero Man Pole-Zero Mar $\times 10^{-3}$ Axis Imaginary Axis stable boundary magin -1.5 -1.5 -0.5 0 Real Axis 0.2 0.6 0.8 Real Axis 1.4 1.6 (a) pole locations (b) zoomed plot Fig. 11-8 Pole positions and stability assessment.

It can be seen that the $||G(s)||_2$ norm can be evaluated through numerical integration methods, while $||G(s)||_{\infty}$ norm can be obtained with numerical optimization approaches. The overload function norm() can be written and placed in the @fotf folder, with the syntaxes norm(G) and norm(G,inf). In old versions of MATLAB, the integral() function can be replaced with quadgk().

where the low-level frequency response function freqresp() is given by

```
function H1=freqresp(w,G)
a=G.a; na=G.na; b=G.b; nb=G.nb; j=sqrt(-1); T=G.ioDelay;
for i=1:length(w)
    P=b*(w(i).^nb.'); Q=a*(w(i).^na.'); H1(i)=P*exp(-T*w(i))/Q;
end
```

Example 11.13. Consider the fractional-order model given in Example 11.12. The norms of the system can be evaluated, and the results are $n_1 = 2.7168$, and $n_2 = 8.6115$.

```
>> a=[2,3.8,2.6,2.5,1.5]; na=[3.501,2.42,1.798,1.31,0];
b=[-2,-4]; nb=[0.63,0]; G=fotf(a,na,b,nb);
n1=norm(G), n2=norm(G,inf)
```

498

Modeling, Analysis and Design of Control Systems in MATLAB and Simulink

11.4.4 Frequency Domain Analysis of FOTF Objects

Consider a fractional-order transfer function G(s). If $j\omega$ is used to substitute s, through simple complex number computation, the exact frequency response data can be obtained directly. The data can be written in the form of the **frd()** function in the Control System Toolbox, so that the frequency domain analysis functions such as **bode()** can be used to draw frequency domain plots. Overload functions for these can also be written, and placed in **@fotf** folder. The listing of the overload function **bode()** is as follows, with the supporting function **freqresp()** as its kernel.

```
function H=bode(G,w)
if nargin==1, w=logspace(-4,4); end
j=sqrt(-1); H1=freqresp(j*w,G); H1=frd(H1,w);
if nargout==0, bode(H1); else, H=H1; end
```

Similarly, overload functions for Nyquist plots and Nichols chart are

```
function nyquist(G,w)
if nargin==1, w=logspace(-4,4); end, H=bode(G,w); nyquist(H);
function nichols(G,w)
if nargin==1, w=logspace(-4,4); end, H=bode(G,w); nichols(H);
```

Example 11.14. Consider again the fractional-order model in Example 11.12. The following statements can be used to draw the Body diagram and Nyquist plot as shown in Figs. 11-9(a) and (b).

```
>> b=[-2,-4]; nb=[0.63,0]; w=logspace(-2,2);
a=[2,3.8,2.6,2.5,1.5]; na=[3.501,2.42,1.798,1.31,0];
G=fotf(a,na,b,nb); bode(G,w);
figure, w=logspace(-2,4,400); nyquist(G,w); grid
```



Fig. 11-9 Frequency domain plots of fractional-order transfer function.

11.4.5 Time Domain Analysis of FOTF Objects

Based on the closed-form solutions of linear fractional-order differential equations, and its MATLAB implementation in function fode_sol(), the overload step response function

step() and arbitrary input time response function lsim() of the fractional-order transfer functions can easily be written

```
function y=step(G,t)
y1=fode_sol(G.a,G.na,G.b,G.nb,ones(size(t)),t);
ii=find(t>G.ioDelay); lz=zeros(1,ii(1)-1);
y1=[lz, y1(1:end-length(lz))];
if nargout==0,
  plot(t,y1,t,c_term(G.b,G.nb)/c_term(G.a,G.na),'--'),
   title('Step Response'), xlabel('Time (Sec)'), ylabel('Magnitude')
else, y=y1; end
function c=c_term(a,na) % this function is to find constant term in polynomials
i=find(na==0); c=0; if length(i)>0, c=a(i(1)); end
function y=lsim(G,u,t)
y1=fode_sol(G.a,G.na,G.b,G.nb,u,t);
ii=find(t>G.ioDelay); lz=zeros(1,ii(1)-1);
y1=[lz, y1(1:end-length(lz))];
if nargout==0, plot(t,y1,t,u,'--'),
   title('Step Response'), xlabel('Time (Sec)'), ylabel('Magnitude')
else, y=y1; end
The syntaxes of the two functions are
```

 $y = \operatorname{step}(G, t)$, and $y = \operatorname{lsim}(G, u, t)$

where G is the FOTF model, t is an evenly spaced time vector, u is a vector of the input samples. We tried to make the syntaxes of these overload functions similar to those in the Control System Toolbox. It should be noted that vector t cannot be omitted here.

Example 11.15. Consider the following fractional-order differential equation

 $\mathscr{D}_{t}^{3.5}y(t) + 8\mathscr{D}_{t}^{3.1}y(t) + 26\mathscr{D}_{t}^{2.3}y(t) + 73\mathscr{D}_{t}^{1.2}y(t) + 90\mathscr{D}_{t}^{0.5}y(t) = 90\sin t^{2},$

and it can be seen that the fractional-order transfer function is

 $G(s) = \frac{90}{s^{3.5} + 8s^{3.1} + 26s^{2.3} + 73s^{1.2} + 90s^{0.5}},$

and the input is $u(t) = \sin t^2$. The following statements can be used to draw the time response of the output y(t) as shown in Fig. 11-10, where the solid curve is the output while the dash plot is the input.

>> a=[1,8,26,73,90]; n=[3.5,3.1,2.3,1.2,0.5]; G=fotf(a,n,90,0); t=0:0.002:10; u=sin(t.^2); lsim(G,u,t);

Similar to other computation problems in MATLAB, the results obtained should be validated. Smaller step sizes can be tried to seen whether the same results can be obtained. If the results with smaller step sizes are the same, the results can be accepted, otherwise the step size should be reduced again. For this example, the results are validated.

11.4.6 Root Locus for Commensurate-order Systems

For commensurate-order systems, if the base order is α , we can let $\lambda = s^{\alpha}$, and the original system can be written as the integer-order model of λ , denoted by $G_1(\lambda)$. Function



Fig. 11-10 Input and output signals of the system.

rlocus() in Control System Toolbox can be used to draw the root locus of integer-order model $G_1(\lambda)$, and superimpose the stability boundaries $\pm \alpha \pi/2$ on the root locus. Based on the idea, the overload function **rlocus()** can be designed. The syntax of the function is **rlocus(G)**, where G is a FOTF object.

```
function rlocus(G)
nx=unique(round(1000*[G.na,G.nb])); nx=nx(nx~=0); nd=max(nx);
for i=1:length(nx), nd=gcd(nd,nx(i)); end
alpha=nd*0.001; na=round(G.na/alpha); nb=round(G.nb/alpha);
b=G.a; den(a+1)=b; den=den(end:-1:1);
b=G.b; num(x+1)=b; num=num(end:-1:1); G1=tf(num,den);
rlocus(G1), xm=xlim; if xm(2)<=0, xm(2)=-xm(1); end
xm(1)=0; line(xm,tan(alpha*pi/2)*xm)</pre>
```

Interactive method in the original rlocus() function is inherited to get the critical gain of the system with mouse clicks.

Example 11.16. Assume that the fractional-order transfer function is given by

$$G(s) = \frac{1}{s^{3.5} + 10s^{2.8} + 35s^{2.1} + 50s^{1.4} + 24s^{0.7}}.$$

It can be seen that the base order is $\alpha = 0.7$. Let $\lambda = s^{0.7}$, the integer-order transfer function can be written as

$$G(\lambda) = \frac{1}{\lambda^5 + 10\lambda^4 + 35\lambda^3 + 50\lambda^2 + 24\lambda}.$$

The root locus of the fractional-order system can be obtained as shown in Fig. 11-11(a). Zooming the root locus, the critical gain can be read K = 371, as shown in Fig. 11-11(b).

>> G=fotf([1 10 35 50 24 0],0.7*[5:-1:0],1,0); rlocus(G)

11.4.7 State Space Models of Commensurate-order Systems

If the fractional-order system can be expressed as the commensurate-order transfer function $G(\lambda)$ with base order α , the matrices $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ for integer-order model $G(\lambda)$ can be obtained, and the state space representation of the fractional-order system can be

Modeling, Analysis and Design of Control Systems in MATLAB and Simulink



Fig. 11-11 Root locus analysis of fractional-order system.

written as

$$\begin{cases} \mathscr{D}^{\alpha} \boldsymbol{x}(t) = \boldsymbol{A} \boldsymbol{x}(t) + \boldsymbol{B} \boldsymbol{u}(t) \\ \boldsymbol{y}(t) = \boldsymbol{C} \boldsymbol{x}(t) + \boldsymbol{D} \boldsymbol{u}(t). \end{cases}$$
(11.50)

The state space models of fractional-order system are not covered in the book. The interested readers are advised to create their own FOSS class and overload functions. These tasks are left as a problem.

11.5 Approximation and Reduction of Fractional-order Systems

11.5.1 Oustaloup Filter for Fractional-order Differentiators

The Grünwald–Letnikov definition presented earlier can be used to evaluate accurately the fractional-order derivatives for given functions. However, there are certain limitations in control systems. Control systems are always described by block diagrams, for instance, the input signal to the plant model is often generated by its previous block, i.e., the controller. Thus, the control signal is not known precisely before numerical derivatives can be calculated. Thus, block diagram approximations, usually with filters, to the fractionalorder derivative actions are expected.

Normally, filters are classified as continuous and discrete ones, here, continuous filters to fit the Laplace operator s^{γ} are mainly discussed. It can be seen that fractional-order derivatives can be approximated by filters.

1. Oustaloup filter approximation

Several filter approximation approaches are summarized in [9], including continued fraction approximation, Charef approximation^[15] and Oustaloup approximation^[16]. Here, only Oustaloup algorithm is presented.

Since pure fractional-order derivative can be represented by straight lines in Bode diagrams, it is not possible to fit them at the whole frequency range with integer-order filters. A specific interested frequency interval $(\omega_{\rm b}, \omega_{\rm h})$ should be assigned, with the

continuous transfer function

$$G_{\rm f}(s) = K \prod_{k=1}^{N} \frac{s + \omega'_k}{s + \omega_k},\tag{11.51}$$

where the poles, zeros and gain can be obtained from

$$\omega_{k}' = \omega_{b} \omega_{u}^{(2k-1-\gamma)/N}, \quad \omega_{k} = \omega_{b} \omega_{u}^{(2k-1+\gamma)/N}, \quad K = \omega_{h}^{\gamma}, \quad (11.52)$$

with $\omega_{\rm u} = \sqrt{\omega_{\rm h}/\omega_{\rm b}}$. Based on the algorithm, the following MATLAB function can be written to implement Oustaloup filter. If y(t) is the input signal to the filter, the output of the filter can be approximately regarded as $\mathscr{D}_t^{\gamma} y(t)$.

```
function G=ousta_fod(gam,N,wb,wh)
k=1:N; wu=sqrt(wh/wb); wkp=wb*wu.^((2*k-1-gam)/N);
wk=wb*wu.^((2*k-1+gam)/N); G=zpk(-wkp,-wk,wh^gam); G=tf(G);
```

with the syntax $G = \text{ousta_fod}(\gamma, N, \omega_b, \omega_h)$, where γ is the order of derivative, N is the order of the filter. The variables ω_b and ω_h are the lower- and upper-bounds of the frequency of users' choice. Normally within the frequency range, the Bode diagram fractional-order operator are satisfactory, while the fitting outside the range is not. The algorithm presented here avoided the restriction on $\omega_b \omega_h = 1$, the two frequencies can be selected independently.

In the function, γ can either be positive or negative, for differentiation and integrals. Also, the absolute values of γ can be larger than 1, for instance, $\gamma = 3.7$. However, in this case, it is suggested to keep $-1 < \gamma < 1$, and leave the remaining integers as TF object. For instance, better to use $s^{3.7} = s^3 s^{0.7}$, or $s^{3.7} = s^4 s^{-0.3}$.

2. Improved Oustaloup filter

In practical applications, the orders of the numerator and denominator are the same, and if at the boundaries of the frequency the fitting is not good, improved Oustaloup filters^[17] and optimal filter design method^[18] can be used. The latter can also be extended to fit complex orders, however, its implementation is rather complicated.

The following improved Oustaloup filter is presented^[17]. The limitations of the filter is that the order should be between 0 and 1. The improved filter is

$$s^{\gamma} \approx \left(\frac{d\omega_{\rm h}}{b}\right)^{\gamma} \left(\frac{ds^2 + b\omega_{\rm h}s}{d(1-\gamma)s^2 + b\omega_{\rm h}s + d\gamma}\right) \prod_{k=1}^{N} \frac{s + \omega'_k}{s + \omega_k},\tag{11.53}$$

and the definitions of ω_k , ω'_k are the same as in Oustaloup filter. Two adjustable parameters b, d are introduced, and normally they can be set to b = 10, d = 9. The MATLAB implementation is given by

function G=new_fod(r,N,wb,wh,b,d)
if nargin==4, b=10; d=9; end, k=1:N; wu=sqrt(wh/wb);
wkp=wb*wu.^((2*k-1-r)/N); wk=wb*wu.^((2*k-1+r)/N);
G=zpk(-wkp,-wk,(d*wh/b)^r)*tf([d,b*wh,0],[d*(1-r),b*wh,d*r]);

3. Filter approximation of high-order fractional-order systems

Based on Oustaloup filter and its improved form, high-order integer-order approximation to fractional-order transfer function, i.e., each fractional-order term can be

approximated with the filters. Based on the idea, the following MATLAB function is designed, and placed in the @fotf folder.

The syntax of the function is $G_1 = \text{high_order}(G, \text{filter}, \omega_b, \omega_h, N)$, where G is the FOTF object, filter can be selected as 'ousta_fod' or 'new_fod'. The arguments ω_b , ω_h and N are the parameters of the Oustaloup filters. The default values are $\omega_b = 10^{-3}$, $\omega_h = 10^3$, N = 5, and the default filter is Oustaloup filter.

It is interesting to note that if the FOTF object G is, in fact, an integer-order model, $G_1 = \text{high_order}(G)$ will convert it into a TF object G_1 .

Example 11.17. Consider high-order fractional-order transfer function

 $G(s) = \frac{-2s^{0.63} - 4}{2s^{3.501} + 3.8s^{2.42} + 2.6s^{1.798} + 2.5s^{1.31} + 1.5}.$

Selecting frequency interval (ω_1, ω_2) , and suitable order N, the high-order integer-order approximation can be obtained, with examples $s^{3.501} = s^3 s^{0.501}$. The low-level command is rather complicated and tedious

```
>> N=9; w1=1e-3; w2=1e3; g1=ousta_fod(0.501,N,w1,w2); s=tf('s');
g2=ousta_fod(0.42,N,w1,w2); g3=ousta_fod(0.798,N,w1,w2);
g4=ousta_fod(0.31,N,w1,w2); g5=ousta_fod(0.63,N,w1,w2);
G1=(-2*g5-4)/(2*s^3*g1+3.8*s^2*g2+2.6*s*g3+2.5*s*g4+1.5)
```

Alternatively, high_order() function can be used directly

```
>> b=[-2 -4]; nb=[0.63 0]; a=[2 3.8 2.6 2.5 1.5];
na=[3.501 2.42 1.798 1.31 0]; G=fotf(a,na,b,nb);
G2=high_order(G,'ousta_fod',w1,w2,N); order(G2)
bode(G1,G2); hold on; bode(G); t=0:0.004:30;
figure; y=step(G,t); step(G1,G2,30); line(t,y)
```

It can be seen that a 45th order integer-order model can be obtained. The exact Bode diagram and its integer-order approximation can be obtained as shown in Fig. 11-12(a). It can be seen that the magnitude curves are almost the same in the specified frequency interval, and the phase difference is 360° , thus they are effectively the same as well. For larger frequency intervals, the frequency response fitting are also satisfactory. Step responses of the fractional-order model and the integer-order approximation can also be obtained as shown in Fig. 11-12(b). It can be seen that the approximation of step response is also satisfactory.



11.5.2 Approximations of Fractional-order Controllers

In control system design, the controller obtained may be rather complicated, and may not be easy to implement. For instance, if $[(as + b)/(cs + d)]^{\alpha}$ term is contained in the controller, the following procedures can be used in controller approximation:

(1) Generate exact frequency response samples to the fractional-order controller;

- (2) Select suitable orders in numerator and denominator for the controller;
- (3) Use function invfreqs() to get the frequency response fitting model;

(4) Validate the fitting results. If it is not satisfactory, the orders should be increased in step (2), or go to step (1) to select again the interested frequency range until a satisfactory model can be obtained.

Example 11.18. Consider the following fractional-order QFT controller model^[5]

$$G_{\rm c}(s) = 1.8393 \left(\frac{s+0.011}{s}\right)^{0.96} \left(\frac{8.8 \times 10^{-5} s+1}{8.096 \times 10^{-5} s+1}\right)^{1.76} \frac{1}{(1+s/0.29)^2}.$$

The controller is rather complicated, so that approximate integer-order controllers are expected, with frequency response fitting approach. Since the frd() function in MATLAB can only be used to deal with integer-order systems, its member variable ResponseData can be used to complete the computation with non-integer powers. Thus, the frequency response data of G(s) can be obtained. Function invfreqs() can then be used to get the fitting model, within the frequency range of $\omega \in (10^{-4}, 10^0)$ rad/s, with the following statements

```
>> w=logspace(-4,0); G1=tf([1 0.011],[1 0]); F1=frd(G1,w);
G2=tf([8.8e-5 1],[8.096e-5 1]); F2=frd(G2,w);
s=tf('s'); G3=1/(1+s/0.29)^2; F3=frd(G3,w); F=F1;
h1=F1.ResponseData; h2=F2.ResponseData; h3=F3.ResponseData;
h=1.8393*h1.^0.96.*h2.^1.76.*h3; F.ResponseData=h;
[n,d]=invfreqs(h(:),w,4,4); G=tf(n,d)
```

The approximate controller obtained is

$$G(s) = \frac{2.213 \times 10^{-7} s^4 + 1.732 \times 10^{-6} s^3 + 0.1547 s^2 + 0.001903 s + 2.548 \times 10^{-6}}{s^4 + 0.5817 s^3 + 0.08511 s^2 + 0.000147 s + 1.075 \times 10^{-9}}$$

We can use a larger frequency interval $(10^{-6}, 10^2)$ rad/s to validate the fitting results.

It can be seen the Bode diagram of the two controllers is shown in Fig. 11-13. It can be seen that apart from the frequency range at very low frequency, the fitting is satisfactory in other frequencies. If we want to enhance the fitting results, the number of frequency samples in step (1) should be increased.

```
>> w=logspace(-6,2,200); F1=frd(G1,w); F2=frd(G2,w); F=F1;
F3=frd(G3,w); h1=F1.ResponseData; h2=F2.ResponseData;
h3=F3.ResponseData; h=1.8393*h1.^0.96.*h2.^1.76.*h3;
F.ResponseData=h; bode(F,'-',G,'--',w)
```



Fig. 11-13 Comparisons of fractional-order QFT controller and integer-order approximation.

11.5.3 Optimal Reduction Algorithm for Fractional-order Models

To get better reduction model, the error between the original model and reduced model to the same input signal should be defined, and the target is to minimize the error criterion. The optimal reduced model can be converted to the minimization of the following objective function

$$J = \min_{a} \| \widehat{G}(s) - G_{r/m,\tau}(s) \|_{2}, \qquad (11.54)$$

where $\boldsymbol{\theta}$ is the vector of undetermined model parameters, i.e.,

$$\boldsymbol{\theta} = \begin{bmatrix} \beta_1, \beta_2, \cdots, \beta_r, \alpha_1, \alpha_2, \cdots, \alpha_m, \tau \end{bmatrix}^{\mathrm{T}}.$$
(11.55)

Since there is delay terms in Eqn. (11.54), Padé approximation is used for them, and the objective function can be changed to the following norm evaluation form

$$J = \min_{\mathbf{a}} \| \widehat{G}(s) - \widehat{G}_{r/m}(s) \|_2.$$
(11.56)

There is no analytical solution to the problem, thus numerical optimization technique can be used. Based on the optimal model reduction algorithm in Chapter 4, we can solve directly the optimal model reduction problem.

Example 11.19. Consider again the fractional-order transfer function in Example 11.17, the following commands can be used to get 45th order approximate model

```
>> b=[-2 -4]; nb=[0.63 0]; a=[2 3.8 2.6 2.5 1.5];
na=[3.501 2.42 1.798 1.31 0]; G=fotf(a,na,b,nb);
G1=high_order(G,'ousta_fod',1e-3,1e3,9); order(G1)
```

Since the order of the original approximation is too high, model reduction techniques can be used with MATLAB function opt_app() presented in Chapter 4. The third-order reduced model is obtained as

 $G_{\rm r}(s) = \frac{0.6122s^2 + 0.6244s + 0.02588}{s^3 + 0.2014s^2 + 0.1972s + 0.01494}$

Comparisons on step responses and Bode diagrams are shown in Figs. 11-14(a) and (b). It can be seen that the approximate third-order model is very close to that of the original system. For high frequency magnitude fitting, it appears that the fitting is not good. However, since logarithmic scale is used, there should not be too much difference.

```
>> Gr=opt_app(G1,2,3,0); step(G1,Gr,'--',30);
hold on, step(G,0:0.01:30);
figure; bode(G1,Gr,'--'); hold on; bode(G);
```



11.6 Simulation Methods for Complicated Fractional-order Systems

11.6.1 Simulation with Numerical Laplace Transform

It has been shown that inverse Laplace transforms can be used to solve simulation problems. However, Laplace and inverse Laplace transforms are not always solvable for complicated systems. For instance, the system with the controller in Example 11.18 cannot be easily solved. Thus, numerical techniques have to be used instead. The INVLAP() function is a powerful tool for performing numerical inverse Laplace transforms^[19,20], with the syntax $[t, y] = INVLAP(fun, t_0, t_f, n)$.

The essential input arguments are: fun, which is a string describing the Laplace transform form F(s), and (t_0, t_f) , which is the time interval, with n points.

June 12, 2014 8:21

508 Modeling, Analysis and Design of Control Systems in MATLAB and Simulink

There was a bug in the original code when $t_0 = 0$, and this is fixed in the package of the book. The remaining input arguments are internal parameters and the use of default values is suggested. More input arguments can be used in the function call, details can be found in doc INVLAP.

Example 11.20. Considering a fractional-order transfer function

 $G(s) = \frac{(s^{0.4} + 0.4s^{0.2} + 0.5)}{\sqrt{s}(s^{0.2} + 0.02s^{0.1} + 0.6)^{0.4}(s^{0.3} + 0.5)^{0.6}},$

where the input is $u(t) = e^{-0.2t} \sin t$, the Laplace transform of it can be obtained and converted to a string. The output signal can be obtained as shown in Fig. 11-15. The execution speed of INVLAP() is extremely fast, the whole process needing only 0.3 s.

>> G='(s^0.4+ 0.4*s^0.2+0.5)/(s^0.2+0.02*s^0.1+0.6)^0.4/(s^0.3+0.5)^0.6';
syms t; u=exp(-0.2*t)*sin(t); Y=['(' G ')*' char(laplace(u))];

[t,y]=INVLAP(Y,0.01,25,1000); plot(t,y)



Fig. 11-15 Response to complicated fractional-order system.

To further illustrate the topic, suppose that the input signal is given by a set of sample points. Of course, using the symbolic Laplace transform is not possible for functions given by sample points. Interpolation, and then numerical integration, should be performed to find the numerical Laplace transform. Based on the source code of INVLAP(), numerical Laplace transform functions can be embedded to solve complicated problems. A new MATLAB function is written, with the syntax

[t, y] =num_laplace(fun, t_0 , t_f , n, x_0 , u_0)

where fun is a string describing the transfer function of the system. The extra arguments x_0 and u_0 , specify the sample points of the input signal. Note that the function is extremely slow. The listing of the M-function is

```
function [t,y]=num_laplace(G,t0,tf,nnt,x0,u0)
FF=strrep(strrep(strrep(G,'*','.*'),'/','./'),'^','.^');
a=6; ns=20; nd=19; t=linspace(t0,tf,nnt);
if t0==0, t=t(2:end); nnt=nnt-1; end % the original bug is fixed here
n=1:ns+1+nd; alfa=a+(n-1)*pi*1j; beta=-exp(a)*(-1).^n; n=1:nd;
bdif=fliplr(cumsum(gamma(nd+1)./gamma(nd+2-n)./gamma(n)))./2^nd;
beta(ns+2:ns+1+nd)=beta(ns+2:ns+1+nd).*bdif; beta(1)=beta(1)/2;
```

Example 11.21. If the analytical form of the input function is not known, but instead only a set of sample points is given, the following statements can be given to calculate the output signal. After around 30 s, the numerical solution can be found and it is exactly the same as the one shown in Fig. 11-15. It can be seen from the example that extremely complicated problems can be solved in this way.

>> x0=0:0.5:25; y0=exp(-0.2*x0).*sin(x0);
[t,y]=num_laplace(G,0,25,200,x0,y0); plot(t,y)

11.6.2 Block Diagram Modeling and Simulation of Linear Fractional-order Systems

It can be seen from the presentation earlier that the best way to evaluate the fractionalorder derivatives to signals inside the system is to use filters with Oustaloup algorithms and other similar algorithms. Besides, since the orders of numerator and denominator are the same in Oustaloup filter and may lead to algebraic loops in simulation, a low-pass filter can be appended with bandwidth $\omega_{\rm h}$. The Simulink model in Fig. 11-16(a) can be established to approximate fractional-order differentiation. We can mask the model into a reusable fractional-order differentiation block. With suitably selected frequency ranges and order, the block can be used in the block diagram-based modeling of complicated nonlinear fractional-order systems.

The masked block is given in Fig. 11-16(b). Double click the block, the dialog box is shown in Fig. 11-16(c). The necessary parameters can be assigned in the dialog box. In the initialization column in the masking process, the following statements can be specified, and the labels in the icon can correctly be displayed.

```
wb=ww(1); wh=ww(2);
if key==1, G=ousta_fod(gam,n,wb,wh); else, G=new_fod(gam,n,wb,wh); end
num=G.num{1}; den=G.den{1}; T=1/wh; str='Fractional\n';
if isnumeric(gam)
    if gam>0, str=[str, 'Der s^' num2str(gam) ];
    else, str=[str, 'Int s^{' num2str(gam) '}]; end
else, str=[str, 'Der s^gam']; end
```

In real simulation processes, the algorithms ode15s or ode23tb are recommended, since the model is likely to be stiff equations. The following examples are used to demonstrate of fractional-order differential equations.

Example 11.22. Consider the linear fractional-order differential equation studied in Example 11.15, expressed as

$$\mathscr{D}_{t}^{3.5}y(t) + 8\mathscr{D}_{t}^{3.1}y(t) + 26\mathscr{D}_{t}^{2.3}y(t) + 73\mathscr{D}_{t}^{1.2}y(t) + 90\mathscr{D}_{t}^{0.5}y(t) = 90\sin t^{2}$$

 $1 \longrightarrow 1 \longrightarrow 1 \longrightarrow 1$ $1 \longrightarrow 1$

	Function block Parameters; Fractional DDS
$\frac{n(s)}{n(s)} \rightarrow \frac{1}{T.s+1} \rightarrow 1$	Parameters Derivative order gamma
fer Fcn Transfer Fcn1	50.6
(a) filter	Frequency range [wb,wh]
(-)	ww
Fractional	Apprixmation order
Der s^0.9	4
	OK Cancel Help Apply

(b) masked block (fodblk.mdl)

(c) parameter dialog box

Fig. 11-16 Design of fractional-order differentiator block.

Let $z(t) = \mathscr{D}_t^{0.5} y(t)$, the original model can be rewritten as

$$z(t) = \sin t^2 - \frac{1}{90} \left[\mathscr{D}_t^3 z(t) + 8\mathscr{D}_t^{2.6} z(t) + 26\mathscr{D}_t^{1.8} z(t) + 73\mathscr{D}_t^{0.7} z(t) \right].$$

Simulink model for the differential equation can be established, as shown in Fig. 11-17. After simulation, the output signal can be obtained and it is exactly the same as the one obtained in Example 11.15.



Fig. 11-17 Simulink representation for the linear fractional-order model (c11fode1.mdl).

Two more masked blocks are established for the package of the book, one for fractionalorder transfer function, and the other for fractional-order PID controller. These blocks can be used directly in Simulink as shown in Fig. 11-18(a).



Fig. 11-18 Simulink library for fractional-order linear blocks.

510

Trans

Analysis and Design of Fractional-order Systems

The code for the Initialization pane of the $\mathsf{Approximate}\ \mathsf{FOTF}\ \mathsf{model}\ \mathrm{block}\ \mathrm{can}\ \mathrm{be}\ \mathrm{expressed}\ \mathrm{as}$

```
if strcmp(class(na),'fotf'), G=na;
else
    if length(na)~=length(a),
        errordlg('Error','Mismatch on the denominator')
    end
    if length(b)~=length(nb),
        errordlg('Error','Mismatch on the denominator')
    end
    G=fotf(a,na,b,nb);
end
if kFilter==1, str='ousta_fod'; else, str='new_fod'; end
wb=ww(1); wh=ww(2);
G0=high_order(G,str,wb,wh,N); [numG,denG]=tfdata(G0,'v');
```

Example 11.23. With the Approximate fractional-order transfer function block, the Simulink model for the differential equation in Example 11.15 can be established as shown in Fig. 11-18(b). Double click the Approximate fractional-order transfer function block, the parameters of the system can be entered as shown in Fig. 11-19. Alternatively, if the FOTF object G is established in MATLAB workspace, we can simply enter G in the Orders of denominator edit box. It can be seen that the model is much simpler than the one in the previous example. Simulation result with the new model is exactly the same as the one in the previous example.

Subsystem (mask)	
Parameters	
Orders of the denominator	
[2.6 1.5 1.3 0.9 0]	
Coefficients of the denominator	
[1 3.3 2.9 3.32 1]	
Orders of the numerator	
0	
Coefficients of the numerator	
1	
Filter order	
7	
Interested frequency range [wb wh]	
[1e-3 1e3]	
Types of the filter Oustaloup filter	Ť
OK Cancel Help	Apply

Fig. 11-19 Parameter dialog box for the FOTF block.

Block Diagram Modeling and Simulation of Nonlinear 11.6.3Fractional-order Systems

For complicated nonlinear fractional-order systems, the overload functions step(). lsim() and the numerical inverse Laplace transforms cannot be used. Block diagrambased modeling and simulation strategies are more important. Here, the modeling and simulation methods are illustrated through an example.

Example 11.24. Consider the following nonlinear fractional-order differential equation

$$\frac{3\mathscr{D}^{0.9}y(t)}{3+0.2\mathscr{D}^{0.8}y(t)+0.9\mathscr{D}^{0.2}y(t)} + \left|2\mathscr{D}^{0.7}y(t)\right|^{1.5} + \frac{4}{3}y(t) = 5\sin 10t.$$

Based on the equation, the explicit form of the y(t) signal is written as

$$y(t) = \frac{3}{4} \left[5\sin 10t - \frac{3\mathscr{D}^{0.9}y(t)}{3 + 0.2\mathscr{D}^{0.8}y(t) + 0.9\mathscr{D}^{0.2}y(t)} - \left|2\mathscr{D}^{0.7}y(t)\right|^{1.5} \right].$$

A Simulink model can be constructed as shown in Fig. 11-20(a). It can be seen from the simulation model that the accuracy of the simulation results are to some extent, related to the specifications of the filters. Different frequency intervals and filter orders may affect the simulation results. Simulation results for different frequency intervals and filter orders are obtained in Fig. 11-20(b). It can be seen that all the curves agree well.



Fig. 11-20 Simulink model and simulation results.

11.7Design of Optimal Fractional-order PID Controllers

Optimal Design of $PI^{\lambda}D^{\mu}$ Controllers 11.7.1

The structures of the fractional-order PID controllers are different from the PID controllers studied in Chapter 8. The fractional $PI^{\lambda}D^{\mu}$ model can be expressed as

$$G_{\rm c}(s) = K_{\rm p} + \frac{K_{\rm i}}{s^{\lambda}} + K_{\rm d}s^{\mu}.$$
 (11.57)

In the illustration shown in Fig. 11-21, the orders of integral and differentiation are

used as the two axes. It can be seen that the conventional PID-type controllers are just a few specific points on the order planes. However, the orders of the controllers in fractional-order PID controllers can be relatively arbitrarily chosen. Normally with stability considerations, $0 < \lambda, \mu < 2$. Because there are two more parameters to tune than the conventional PID controller, the fractional-order PID controllers are usually more flexible, and may expect better performances^[21].



Fig. 11-21 Illustration of fractional-order PID controllers.

From loop shaping point of view, the slopes in Bode magnitude plots are multiples of $20 \, dB/dec$ in integer-order systems, while in fractional-order systems there is no such restrictions. Thus, the shape can be arbitrarily shaped. For instance, at the places around the crossover frequency, the slope of the magnitude plot can be assigned to very small values, such that the robustness of the closed-loop system can be increased.

Example 11.25. For the given fractional-order plant model

$$G(s) = \frac{1}{s^{2.6} + 2.2s^{1.5} + 2.9s^{1.3} + 3.32s^{0.9} + 1},$$

it might be difficult to design a PID controller. Thus, there are some attempts to approximate the model with FOPDT model $G_{\rm p}(s) = k {\rm e}^{-Ls}/(Ts+1)$, so that the methods in Chapter 8 can be used to design PID controllers. For instance, Wang–Juang–Chan algorithm^[22] can be used to design optimal ITAE criterion PID controller

$$K_{\rm p} = \frac{(0.7303 + 0.5307T/L)(T + 0.5L)}{K(T + L)}, \quad T_{\rm i} = T + 0.5L, \quad T_{\rm d} = \frac{0.5LT}{T + 0.5L}.$$
 (11.58)

It can be seen that the FOPDT model can be obtained with

>> N=5; w1=1e-3; w2=1e3; s=fotf('s'); G=1/(s^2.6+3.3*s^1.5+2.9*s^1.3+3.32*s^0.9+1); G0=high_order(G,'ousta_fod',w1,w2,N); Gr=opt_app(G0,0,1,1)

The reduced model can be obtained as $G_r(s) = 0.1836e^{-0.827s}/(s+0.1836)$. Then a PID controller can be designed with the following statements

>> L=Gr.ioDelay; [n,d]=tfdata(Gr,'v'); K=n(2)/d(2); T=d(1)/d(2); Ti=T+0.5*L; Kp=(0.7303+0.5307*T/L)*Ti/(K*(T+L)); Td=(0.5*L*T)/(T+0.5*L); s=tf('s'); Gc=Kp*(1+1/Ti/s+Td*s), w=logspace(-4,4,200); C=fotf(Gc);

```
H=bode(G*C,w); bode(G0*Gc,'-',H,'--'); figure;
t=0:0.01:20; step(feedback(G0*Gc,1),20),
y=step(feedback(G*C,1),t); hold on; plot(t,y,'--')
```

The PID controller can be designed as $G_{c}(s) = 3.9474 (1 + 1/(5.8232s) + 0.3843s).$ Under such a controller, the open-loop Bode diagram and closed-loop step response can be obtained as shown in Figs. 11-22(a) and (b). It can be seen that the two systems are quite close.



Fig. 11-22 Fractional-order PID control results.

Example 11.26. Consider the plant model in the previous example. Now the searching algorithm can be used to design optimal $PI^{\lambda}D^{\mu}$ controller. The following MATLAB function can be written to describe the objective function

```
function fy=fpidfun(x,G,t,key)
s=fotf('s'); C=x(1)+x(2)*s^(-x(4))+x(3)*s^(x(5));
dt=t(2)-t(1); y=step(feedback(G*C,1),t); e=1-y;
if key==1, fy=dt*sum(t.*abs(e)); else, fy=dt*sum(e.^2); end
disp([x(:); fy].')
```

where in the last statement, the intermediate results can be obtained. The function has three additional arguments, G is the FOTF plant model, t is the evenly spaced time vector, and key is the criterion, with 1 for ITAE criterion, otherwise for ISE criterion.

Assume the terminate time is 10 s, and assume the parameters of the $PI^{\lambda}D^{\mu}$ controller are all smaller than 10, and the orders are in the interval (0,2). The function fminsearchbnd() is recommended to find optimal $PI^{\lambda}D^{\mu}$ controller

```
>> xm=zeros(5,1); xM=[10; 10; 10; 2; 2];
  x0=[Kp,Kp/Ti,Kp*Td,1,1].'; t=0:0.01:20;
  x=fminsearchbnd(@fpidfun,x0,xm,xM,[],G,t,1)
   s=fotf('s'); Gc1=x(1)+x(2)*s^(-x(4))+x(3)*s^(x(5));
   step(feedback(G*Gc1,1),t);
   y=step(feedback(G*C,1),t); hold on; plot(t,y,'--')
```

The optimal controller is $G_{\rm c}(s) = 10 + 2.3088s^{-0.9877} + 8.9811s^{0.4286}$. The step responses of the systems under this controller and the one obtained in the previous example,

are obtained as shown in Fig. 11-23. It can be seen that the fractional-order PID controller is better than the integer-order controller.



Fig. 11-23 Comparisons of different PID controllers.

Based on the above idea, an optimal fractional-order PID controller design function can be designed for linear fractional-order plants

```
function [Gc,x,y]=fpidtune(G,type,t,key,x0,xm,xM,ff)
if nargin==7, ff=optimset; ff.MaxIter=50; end
x=fminsearchbnd(@fpidfuns,x0,xm,xM,ff,G,t,key,type);
[y,Gc]=fpidfuns(x,G,t,key,type);
```

The syntax of the function is

 $[G_{c}, x, y] =$ fpidtune $(G, type, t, key, x_{0}, x_{m}, x_{M}, ff)$

where G is the FOTF plant model, type is the expected controller type, with options 'fpid', 'fpi', 'fpi', 'fpidx', and 'pid', with 'fpidx' for PID^{μ} controller with integer integral. The argument t is the evenly spaced time vector, key is the type of criterion, with options 'itae', 'ise', 'iae' and 'itse', with 'itae' recommended. The definitions of variables x_0 , x_m , x_M are the same as defined earlier. Variable ff is the optimization control template, and it can be omitted.

For different types of fractional-order controllers and criteria, the supporting MATLAB function describing the objective function can be written as

```
function [fy,C]=fpidfuns(x,G,t,key,type), s=fotf('s');
switch type
    case 'fpid', C=x(1)+x(2)*s^(-x(4))+x(3)*s^(x(5));
    case 'fpi', C=x(1)+x(2)*s^(-x(3));
    case 'fpid', C=x(1)+x(2)*s^x(3);
    case 'fpidx', C=x(1)+x(2)/s+x(3)*s^x(4);
    case {'pid','PID'}, C=x(1)+x(2)/s+x(3)*s;
end
dt=t(2)-t(1); y=step(feedback(G*C,1),t); e=1-y;
switch key
    case {'itae','ITAE'}, fy=dt*sum(t.*abs(e));
    case {'itae','ISE'}, fy=dt*sum(e.^2);
    case {'itae','IAE'}, fy=dt*sum(t.*e.^2);
```

otherwise, error('Error: available criteria are itae, ise, iae, itse.') end

disp([x fy])

Since this is an open structure with switch commands, other controller structures and criteria selections can be added to the source code directly by the readers, if necessary.

Example 11.27. Consider again the problem in Example 11.26. The optimal fractionalorder PID controller can be obtained directly with the following MATLAB function, and the results are exactly the same as the ones in the previous example.

```
>> s=fotf('s'); G=1/(s^2.6+3.3*s^1.5+2.9*s^1.3+3.32*s^0.9+1);
xm=zeros(5,1); xM=[10; 10; 10; 2; 2]; x0=[1;1;1;1;1].';
t=0:0.01:20; [Gc,x]=fpidtune(G,'fpid',t,'itae',x0,xm,xM)
```

The following statements can also be used to design the optimal PID controller

```
>> xm=zeros(3,1); xM=[10; 10; 10]; x0=[1;1;1].';
t=0:0.01:20; [Gc1,x]=fpidtune(G,'pid',t,'itae',x0,xm,xM)
step(feedback(G*Gc,1),t); hold on; step(feedback(G*Gc1,1),t);
```

with an optimal conventional PID controller $G_{c1} = 9.9945 + 1.5107/s + 9.1101s$. The closed-loop step responses under the two controllers are shown in Fig. 11-24. It can be seen that the closed-loop response under $\mathrm{PI}^{\lambda}\mathrm{D}^{\mu}$ controller is much better than the conventional PID controller for the fractional-order plant model.



Fig. 11-24 Comparisons of different PID controllers.

If the plant model and controller model are both approximated with Oustaloup filters, the following statements should be specified, and integer-order closed-loop model, usually of extremely high order (for this example, a 43th order closed-loop model is obtained, with 7th order Oustaloup filter), can be obtained. The closed-loop step response of the high order integer-order system is almost the same as the one obtained in Fig. 11-24.

```
>> Gc0=high_order(Gc,'ousta_fod',1e-3,1e3,7);
G0=high_order(G,'ousta_fod',1e-3,1e3,7); G1=feedback(G0*Gc0,1);
order(G1), step(G1,t); hold on; step(feedback(G*Gc,1),t)
Gc10=high_order(Gc1); step(feedback(G0*Gc10,1),t)
```

Example 11.28. The fractional-order PID control system is modeled with Simulink as shown in Fig. 11-25. When the parameters in the plant and fractional-order PID controller

Analysis and Design of Fractional-order Systems

are specified, simulation results can be obtained. It can be seen that the control results are exactly the same as the one obtained in the previous example.



Fig. 11-25 Fractional-order PID controller system (file name: fPID_simu.mdl).

11.7.2 OptimFOPID — An Optimal Fractional-order PID Controller Design Interface

Based on the algorithms presented earlier, a graphical user interface, OptimFOPID, is designed. This function can be used to design optimal fractional-order PID controllers with user interface^[23].

The plant model G in FOTF format should be entered into MATLAB workspace first. Then type optimfopid command at MATLAB prompt. The user interface shown in Fig. 11-26 will be displayed. Click Plant model, the model G can be loaded into the interface. Then the controller type, object function type and terminate simulation time should be selected in the interface. Clicking Optimize button will invoke the optimal controller design process, and finally the optimal controller can be obtained in G_c , in MATLAB workspace. Clicking Closed-loop response button will show the step response of the closed-loop system.

Example 11.29. Consider the plant model

$$G(s) = \frac{1}{0.8s^{2\cdot 2} + 0.5s^{0.9} + 1}$$

The following procedures can be used to design optimal fractional-order PID controller.

(1) Type optimfopid command to invoke the interface.

(2) Enter the FOTF model G into MATLAB workspace, and click Plant model to load the model into the interface.

>> G=fotf([0.8 0.5 1],[2.2 0.9 0],1,0)

(3) Set the upper bounds of the controller parameters to 15, and terminate time at8. It should be noted that the upper bounds of controller parameters may affect the final search results.

(4) Click the **Optimize** button to initiate the optimization process, and the optimal fractional-order controller can be obtained, and for this example, the optimal vector is

$$\boldsymbol{x} = [6.5954 \ 15.7495 \ 11.4703 \ 0.9860 \ 1.1932].$$

The controller model can be written as

$$G_{\rm c}(s) = 6.5954 + \frac{15.7495}{s^{0.986}} + 11.4703s^{1.1932}.$$

(5) Click Closed-loop response to draw the closed-loop step response of the system, as shown in Fig. 11-27(a). Since the order of integrator is very close to 1, PID[^]mu item from

Optimize	er for Fract	ional-ord	er PID Cor	ntroller		
1						Plant model
0.8 -						Optimize
0.6						Closed-loop response
						Time vector 0:0.01:10
0.4						Controller type
0.2 -						Pi^mu D^\lambda A Pi^mu PD^\lambda PID^\lambda
						Optimization criterion
0				1		TTAE *
0	0.2	0.4	0.6	8.0	1	ISE +
						Optimization algorithm
Upper bound		20		() Hold	d	Search with boundaries
Lows	er bound	D				d Ill +
						-

Fig. 11-26 Optimal fractional-order PID controller design interface.

the controller type listbox can be selected, and optimal PID^{μ} controller can be designed. The result is very close to the one obtained by $PI^{\lambda}D^{\mu}$ controller.

```
>> t=0:0.01:8; y=step(feedback(G*Gc,1),t); plot(t,y)
```



If the PID item from the Controller Type listbox is selected, and then if Optimize button is clicked, the optimal integer-order PID controller can be designed, and the closed-loop step response can be obtained as shown in Fig. 11-27(b). It can be seen that for this example, the results of fractional-order PID controller is better than the integer-order one.

For linear fractional-order plant models, the OptimFOPID interface can be used to

Analysis and Design of Fractional-order Systems

directly design fractional-order PID controllers in a user friendly manner. There are also limitations in the interface. For instance, in the current version, the plant should not contain time delays. Also, controllers with actuator saturation cannot be processed.

11.8 Problems

(1) Assume that the fractional-order differential equation is $^{[2]}$

$$0.8\mathscr{D}_t^{2.2}y(t) + 0.5\mathscr{D}_t^{0.9}y(t) + y(t) = 1, \ y(0) = y'(0) = y''(0) = 0,$$

find the numerical solutions. If orders 2.2 and 0.9 are approximated by integers 2 and 1, the original differential equation can be approximated by integer-order differential equation. Please compare the approximation results.

(2) With the code for Mittag–Leffler functions, verify the following

(i)
$$\mathscr{E}_{\alpha,\beta}(x) + \mathscr{E}_{\alpha,\beta}(-x) = 2\mathscr{E}_{\alpha,\beta}(x^2)$$
, (ii) $\mathscr{E}_{\alpha,\beta}(x) - \mathscr{E}_{\alpha,\beta}(-x) = 2x\mathscr{E}_{\alpha,\alpha+\beta}(x^2)$,
(iii) $\mathscr{E}_{\alpha,\beta}(x) = \frac{1}{\Gamma(\beta)} + \mathscr{E}_{\alpha,\alpha+\beta}(x)$, (iv) $\mathscr{E}_{\alpha,\beta}(x) = \beta\mathscr{E}_{\alpha,\beta+1}(x) + \alpha x \frac{\mathrm{d}}{\mathrm{d}x} \mathscr{E}_{\alpha,\beta+1}(x)$.

(3) Two filter approximation approaches are proposed in the chapter on fractional-order derivatives. Please compare the two filters for the following fractional-order system, in frequency and step response fitting.

$$G(s) = \frac{s+1}{10s^{3.2} + 185s^{2.5} + 288s^{0.7} + 1}$$

(4) Analyze the stability of the closed-loop system, and draw Bode diagram and closed-loop step response for the following system.

$$G(s) = \frac{s^{1.2} + 4s^{0.8} + 7}{8s^{3.2} + 9s^{2.8} + 9s^2 + 6s^{1.6} + 5s^{0.4} + 9}, \quad G_c(s) = 10 + \frac{9}{c^{0.97}} + 10s^{0.98}.$$

(5) Draw root locus for the following fractional-order plant models and find the critical gains of them.

(i)
$$G_1(s) = \frac{s^{1.5} + 9s + 24s^{0.5} + 20}{3s^2 + 16s^{1.5} + 9s + 20s^{0.5}}$$
, (ii) $G_2(s) = \frac{s+1}{10s^{3.2} + 185s^{2.5} + 288s^{0.7} + 1}$

- (6) With reference to the FOTF class definition and overload function programming methods, please define a FOSS class for state space representation of commensurateorder systems, and write suitable overload functions. Write out FOTF and FOSS conversion functions. The transfer functions in Problem 5 can be used to validate the class.
- (7) Consider the complicated plant models

$$G(s) = \frac{(s^{0.2} + 3s^{0.1} + 3)^{0.6}}{s^{0.7}(s^{0.1} + 2)^{0.5}(s^{0.4} + 2)^{0.3}}$$

It is obvious that FOTF class cannot be used to handle the plant model. Thus, the overload **bode()** function cannot be used directly. Please draw the Bode diagram of the system through low-level commands.

(8) Please consider the following fractional-order control system with

$$G(s) = \frac{1}{s^{0.5}(s^{0.2}+2)^{0.7}} e^{-1.3s}, \text{ with } G_{\rm c}(s) = 0.8 + \frac{2}{s^{0.45}} + 0.6s^{0.3}.$$

Again the FOTF class cannot be used to handle the closed-loop system representation. Please try to draw the unit step response of the closed-loop system through numerical inverse Laplace transform approach.

(9) Solve the following nonlinear fractional-order differential equation with zero initial conditions, where $f(t) = 2t + 2t^{1.545}/\Gamma(2.545)$.

$$\mathscr{D}^{2}x(t) + \mathscr{D}^{1.455}x(t) + \left[\mathscr{D}^{0.555}x(t)\right]^{2} + x^{3}(t) = f(t).$$

(10) For the fractional-order model

$$G_1(s) = \frac{5}{s^{2.3} + 1.3s^{0.9} + 1.25}, \ G_2(s) = \frac{5s^{0.6} + 2}{s^{3.3} + 3.1s^{2.6} + 2.89s^{1.9} + 2.5s^{1.4} + 1.2},$$

please find integer-order approximations, and find out the suitable order of the filters. Find also a suitable reduced order model, and compare frequency domain and step response of the reduced order systems.

(11) Find suitable low-order approximations to the following fractional-order models, and compare frequency domain fitting results.

(i)
$$G(s) = \frac{25}{(s^2 + 8.5s + 25)^{0.2}}$$
, (ii) $G(s) = \frac{562920(s + 1.0118)^{0.6774}}{(s^2 + 54.7160s + 590570)^{0.8387}}$.

(12) Design optimal integer-order PID controller and $PI^{\lambda}D^{\mu}$ controller, and observe the control results.

$$G(s) = \frac{5s^{0.6} + 2}{s^{3.3} + 3.1s^{2.6} + 2.89s^{1.9} + 2.5s^{1.4} + 1.2}.$$

(13) Consider the fpidfuns() function. If the following controller and criterion are used, please extend the function

$$G_{\rm c}(s) = K_{\rm p} \left(1 + \frac{K_{\rm i}}{s} \right) \left(1 + \frac{K_{\rm d}s}{Ts+1} \right), \ I = \int_0^\infty t^2 e^2(t) {\rm d}t.$$

- (14) Extend the OptimFOPID interface, such that more optimization algorithms can be used, to design optimal $PI^{\lambda}D^{\mu}$ controllers with global optimization algorithms.
- (15) Consider the following uncertain fractional-order plant model $G = b/(as^{0.7} + 1)$, with nominal values a = b = 1, approximate the plant model with integer-order transfer function, and design robust optimal \mathcal{H}_{∞} controller, and observe control results with simulation methods for $a \in (0.2, 5)$, and $b \in (0.2, 1.5)$.

Bibliography and References

- Miller K S, Ross B. An introuction to fractinal calculus and fractional differential equations. New York: John Wiley & Sons, 1993
- [2] Podlubny I. Fractional differential equations. San Diago: Academic Press, 1999
- [3] Vinagre B M, Chen Y Q. Fractional calculus applications in automatic control and robotics. Las Vegas: 41st IEEE CDC, Tutorial workshop 2, 2002

References and Bibliography

- [4] Manabe S. The non-integer integral and its application to control systems. Japanese Institute of Electrical Engineers Journal, 1960, 80(860):589–597
- [5] Monje C A, Chen Y Q, Vinagre B M, Xue D Y, Feliu V. Fractional-order systems and controls — fundamentals and applications. London: Springer, 2010
- [6] Lakshmikantham V, Leela S. Theory of fractional dynamic systems. Cornwall, UK: Cambridge Scientific Publishers, 2010
- [7] Caponetto R, Dongola G, Fortuna L, Petráš I. Fractional order systems modeling and control applications. Singapore: World Scientific Publishing, 2009
- [8] Hilfer R. Applications of fractional calculus in physics. Singapore: World Scientific Publishing, 2000
- [9] Petráš I, Podlubny I, O'Leary P, Dorčák L, Vinagre B M. Analogue realization of fractional order controllers. Fakulta BERG, Technical University of Košice, 2002
- [10] Xue D Y, Chen Y Q. Solving applied mathematical problems with MATLAB. Boca Raton: CRC Press, 2008
- [11] Shukla A K, Prajapati J C. On a generalization of Mittag–Leffler function and its properties. Journal of Mathematical Analysis and Applications, 2007, 336(1):797–811
- [12] Podlubny I. Mittag-Leffler function, 2005. http://www.mathworks.cn/MATLAB central/fileexchange/8738-mittag-leffler-function
- [13] Kilbas A A, Saigob M, Saxena R K. Generalized Mittag–Leffler function and generalized fractional calculus operators. Integral Transforms and Special Functions, 2004, 15(1):31–49
- [14] Matignon D. Stability properties for generalized fractional differential systems. Matignon D, Montseny D, eds., Proceedings of the Colloquium Fractional Differential Systems: Models, Methods and Applications, 5. Paris, 1998 145–158
- [15] Charef A, Sun H H, Tsao Y Y, Onaral B. Fractal system as represented by singularity function. IEEE Transactions on Automatic Control, 1992, 37(9):1465–1470
- [16] Oustaloup A, Levron F, Mathieu B, Nanot F M. Frequency-band complex noninteger differentiator: characterization and synthesis. IEEE Transaction on Circuit and Systems-I: Fundamental Theory and Applications, 2000, TCS-47(1):25–39
- [17] Xue D Y, Zhao C N, Chen Y Q. A modified approximation method of fractional order system. Proceedings of IEEE Conference on Mechatronics and Automation. Luoyang, China, 2006: 1043–1048
- [18] Meng L, Xue D Y. An approximation algorithm of fractional order pole models based on an optimization process. Proceedings of Mechatronics and Embedded Systems and Applications. Qingdao, China, 2010: 486–491
- [19] Valsa J, Brančik L. Approximate formulae for numerical inversion of Laplace transforms. International Journal of Numerical Modelling: Electronic Networks, Devices and Fields, 1998, 11(3):153–166
- [20] Valsa J. Numerical inversion of Laplace transforms in MATLAB. MATLAB Central File ID: #32824, 2011
- [21] I. Podlubny. Fractional-order systems and $PI^{\lambda}D^{\mu}$ controllers. IEEE Transactions on Automatic Control, 1999, 44(1):208–214
- [22] Wang F S, Juang W S, Chan C T. Optimal tuning of PID controllers for single and cascade control loops. Chemical Engineering Communications, 1995, 132:15–34
- [23] Xue D Y, Chen Y Q. OptimFOPID: a MATLAB interface for optimum fractional-order PID controller design for linear fractional-order plants. Proceedings of Fractional Derivatives and Its Applications. Nanjing, China, 2012 #307

Index of Functions

A abs, 32, 36, 128, 129, 154, 156, 168 acker, 275, 280 adapt_sim, 415, 416 addmf, 433 addrule, 434 addvar, 433 aic, 139 **all**, 24 angle, 497 any, 24, 32, 156 apolloeq*, 79, 80 are, 69arx, 135, 136, 137, 139, 142, 143 assignin, 72, 281-283, 286, 468, 469 augtf, 373, 374, 375, 378-383, 386, 397 augw, 373 axes, 54, 55 axis, 37, 45, 194

Ð

C

c10bp_pid.mdl*, 447 c10funun*, 468 c10mfzpid.mdl*, 443 c10mga1*, 464 c10mhebb*, 445 c10mmras.mdl*, 410 c10mrbf.mdl*, 451 c10plant.mdl*, 451 c10shebb.mdl*, 445 c11fode1.mdl*, 510 c2d, 105, 106, 107, 115, 177, 192, 229 c2eggui1*, 49 c2eggui2*, 52 c2eggui4*, 55 $\texttt{c6mcompc.mdl*},\ 228$ c6mdde3.mdl*, 234 c6mloopa.mdl*, 239 c6mmulr.mdl*, 232 c6mnlrsys.mdl*, 236 c6mtimv.mdl*, 230 c6mtimva.mdl*, 233 c7mmopt.mdl*, 309 c7optm1*, 281, 282 c7optm2*, 282, 283 c7optm3*, 283 c8mantiw.mdl*, 324 canon, 162, 163 caputo*, 482, 483, 487 care, 272 ceil, 26, 482, 487 char, 172, 492chol, 65chrpid*, 335class, 492, 511 cmpc, 423, 424collect, 25, 118comet, 37

Modeling, Analysis and Design of Control Systems in MATLAB and Simulink

D

d2c, 106, 107, 115, 142 dcgain, 129, 174, 325, 326 dec2mat, 391-393, 395 decouple*, 310, 311 decouple_pp*, 313defuzz, 433 det, 13, 62, 63 dhinf, 382 diag, 273, 280, 299, 364 diff, 172, 483, 487, 495 dimpulse, 415diophantine_eq*, 412, 413, 417 disp, 88, 129, 245, 246, 250, 492, 514, 516 display*, 492 dlinmod, 241dlqr, 272dlyap, 68 doc, 30, 508double, **393**, 394, 395, 483 dsolve, 80, 81, 172

${\mathscr E}$

eye, 68, 113, 115, 120, 121, 156, 162, 168, 273, 301, 304, 305, 313, 379, 381, 392, 415 ezplot, **39**, 70, 74, 171, 182, 464

Ŧ

factor, 25, 26, 27, 63 factorial, 32 faddf, 201fdly, 201 feasp, 391, 392, 395 feather, 37fedmunds, 304, 306, 307 feedback, 153-155, 184, 194, 229, 268 feedbacksym*, 112, 118 fgersh, 199 fget, 305, 307 figure, 76, 78, 123, 125, 126, 129-133, 141, 191, 192, 235, 236, 268, 269, 307, 364, 365, 367, 376, 379, 383, 385, 397, 404, 424-426, 482, 514 fill, 37 fill3, 41 find, 24, 174, 199, 257, 267, 388, 467, 492, 500findop, 240findsum*, 30, 31 finv, 201 fix, 26, 495, 497 fliplr, 23, 156, 508 flipud, 23 floor, 26, 32, 492, 504 fmincon, 83, 84, 282, 283, 464 fminsearch, 82, 129, 282, 498 fminsearchbnd, 82, 514, 515 fminunc, 82 fmul, 201, 202 fmulf, 201fodblk.mdl*, 510 fode_caputo*, 488 fode_sol*, 486, 500 folipd*, 342, 343 fotf*, 492, 493-501, 504, 511, 514, 515 fourier, 86 fPID_simu.mdl*, 517 fpidfun*, 514 fpidfuns*, 515 fpidtune*, **515**, 516 frd, 142, 201, 499, 505, 506

Index of Functions

freqresp*, 498, 499
fsolve, 71, 72
funm, 66
fuz_pid*, 440
fuzzy, 435

G

ga, 463gamma, 482-484, 487, 508gaopt, 463, 465, 468 gaoptimset, 463, 464 gcbo, 51gcd, 26, 27, 497, 501 gcf, 49 get, 37, 49, 54 getDelayModel, 100, 116 getfopdt*, 325, 326, 327, 334, 335 getlmis, 391, 392, 393, 395 getopinfo, 240gevp, 391 glfdiff*, 481, 482, 483 gpc_1a*, 430 gram, 158, 160grid, 35, 54, 76, 184, 185, 190-192, 499 grpbnds, 400, 401 guide, 46, 56

 \mathscr{H}

h2syn, **378** han_td*, **253** hankel, 123, 275, 393 heaviside, 171 help, 12, 25, 31, 223, 382 high_order*, **504**, 507, 511, 513, 516 hilb, **26**, **62** hinflmi, **396**, 397 hinfsyn, **378**, 379-383, **385**, 386, 397 hist, **37**, 236 hold, 41, 70, 74, 137, 179, 182, 192, 228, 268, 269, 416, 507, 514, 516

Ĵ

iddata, 137, 139, 143
idinput, 141, 142, 143
ifourier, 86
ilaplace, 86, 87, 169-171, 173, 176
ilc_lsim*, 459, 460, 461

imag, 199, 200, 298, 497 image, 245, 246 impulse, 175, 180 imread, 56, 246 inagersh*, 199, 200-202, 299-301 initial, 181 inline, 33 int, 166, 167, 182, 487 int2str, 388 integral, 498, 509 interp1, 482, 509 intstable*, 155 INVLAP, 507, 508 inv, 12, 64, 66, 68, 120-122, 162, 164, 165, 199, 202, 275, 299-301, 305, 310, 313, 325, 395, 431, 495 invfreqs, 142, 505 **ipdctrl***, 341 isa, 492 isfinite, 325, 484, 498 isnumeric, 509 isprime, 26isstable, 153, 154, 497 iztrans, 88, 170-172

 \mathscr{K}

kalman, **361**, 365, 367 kron, 68, 494

Ľ

laplace, 86, 169, 170, 173, 508 lcm, 26, 27 leadlagc*, 267, 268-270 length, 33, 72, 124, 129, 162, 168, 201, 236, 240, 257, 267, 494, 495 line, 35, 410, 497, 501, 504 linearize, 241, 242 linmod, 241, 242 linmod2, 241 linprog, 84, 466 linspace, 236, 508 lmiterm, 391, 392, 393, 395 lmivar, 391, 392, 395 loglog, 37logspace, 142, 192, 200, 299-301, 304, 368, 403, 499, 505, 506, 513 lpshape, 400, 403lqg, 363, 364

Modeling, Analysis and Design of Control Systems in MATLAB and Simulink

lqr, 272, 273, 280, 365, 367, 368
lsim, 137, 142, 143, 180, 181, 277, 500
lsqcurvefit, 84, 85, 325
ltrsyn, 367, 368
ltru, 367
lu, 65
luenberger*, 164, 165
lyap, 68, 158
lyap2lmi*, 388, 389
lyapsym*, 68

М

margin, 195, 196, 269, 325, 330, 331, 334 mat2dec, 392, 393 max, 72, 269, 282, 283, 326, 431, 459, 492, 501mesh, 41, 42 meshgrid, 42, 44 mfedit, 433 mfrd*, 201, 202, 299-301, 304, 305 min, 199, 299, 495, 497 mincx, **391**, 393 minreal, 109, 143, 155, 280, 310, 313, 383, 504 minus*, 495 mksys, 373 ml_func*, 484, 485 mlf, 484, 485 modred, $\mathbf{132}$ more_sol*, 72, 73, 74 mpc, 425, 426 mpccon, 422, 423 mpcsim, 422, 423 mpctool, 426 mpower*, 495mrdivide*, 495 mtimes*, 494 multi_step*, 257 mv2fr, 199, 200-202 mvss2tf, 197 my_fact*, 32 my_fibo*, 32 myhilb*, 31, 32

Ň

nargin, 31, 50, 68, 72, 112, 128, 199, 267, 298, 484, 492, 499, 504

new_fod*, 503, 509 newfis, 433nichols, 190-192, 499 nnbp_pid*, 447 nnrbf_pid*, 449 nntool, 443nonlin, 286 norm, 63, 68-70, 72, 74, 85, 165, 166, 310, 313, 392, 484, 497, 498 normGeomSelect, 462 null, 67 num_laplace*, 508, 509 num2str, 253, 256, 492, 509 numden, 25, 110 nyquist, 54, 189, 190, 192, 194, 197, 365, 367, **499**

6

obsvsf*, 279, 280 ocd*, 284, 286 ode15s, 75, 78 ode23, 75 de45, 75, 76-80odeset, 75, 80 ohklmr, 133 ones, 143, 282, 283, 298, 306, 307, 415, 430, 450, 481, 486, 488 open_system, 211 operspec, $\mathbf{240}$ opt_app*, 128-131, 325, 343, 507, 513 opt_fun*, 129 optimfopid*, 517 optimpid*, 352 optimset, 72, 84, 85, 282, 515 order, 504, 507, 516orth, 65 ousta_fod*, 503, 504, 509, 513

P

pade, 126, 129, 186
pade_app*, 123, 126
pademod*, 123, 124
paderm*, 126, 179, 186
patternsearch, 462, 469
pcode, 34
pfshape, 400, 403
pid, 321, 323
pid_tuner*, 338

Index of Functions

pidstd, 321, 322, 323, 328, 335, 341-343 pidtool, 345 pidtune, 344, 354, 357 pinv, 64, 67 place, 275, 276, 278 plot, 34, 35, 36, 55, 74, 76, 79, 85, 137, 181, 192, 200, 230, 235 plot3, 41 plotbnds, 400, 401 plotnyq, 199 plotstep, 421, 422 plottmp, 398, 401 plotyy, 483 plus*, 494 polar, 37 pole, 153 poly, 63, 107, 275 poly2caputo*, 487, 488 poly2sym, 110, 176, 487 poly2tfd, 421, 422, 424 polyfit, 85 polyval, 64, 85, 488, 497 polyvalm, 64, 313 pretty, 80, 118pso_Trelea_vectorized, 469 psuediag*, 298, 300 pzmap, 152, 153, 154

\mathcal{Q}

quadgk, 498 quadprog, 84 quiver, **37**

\mathcal{R}

rand, 71, 72, 164, 446, 469
randn, 415
rank, 63, 67, 157, 158, 164
rat, 26
readfis, 434, 438
real, 156, 199, 200, 298, 497, 509
reg, 279
rem, 26
reshape, 68, 256, 448, 450, 484
residue, 490
return, 129
rlocus, 182, 183-188, 501
rls_ident*, 255
roots, 13, 497

rossler*, 76 rot90, **23**, 123 roulette, 462 round, **26**, 484, 501 routhmod*, 124, 125 rref, **67** rziegler*, 334

Š

satur_non*, 250 schmr, 132sdpvar, 393, 394, 395 sectbnds, 400, 401 semilogx, 37, 38, 305, 307 semilogy, 37 set, 37, 49, 50, 51, 54, 56, 98, 141, 174, 394, 395 setdiff, 156 setlmis, 391, 392, 395 shading, 44 sigma, 202sigmaplot, 202, 203 sign, 250, 253, 448, 481 sim, 223, 226, 228, 230, 233, 281-283, 286, 410, 425, 426, 468, 469 sim_observer*, 277, 278 simple, 25, 63, 80, 81, 87, 88, 112, 167, 495simset, 223, 230, 233 simsizes, 251, 253, 256, 257, 430 simulannealbnd, 462 sin, 23, 33, 267, 278, 325, 328, 459, 464, 481sisobnds, 399, 401 sisotool, 290, 291, 293, 348, 354 size, 68, 70, 72, 164, 199, 249, 298, 310, 381, 382, 388 smat2ss*, 375 solve, 13, 70 solvesdp, 393, 394, 395 sort, 492, 495 sprintf, 250 sqrt, 87, 129, 173, 199, 253, 267, 325, 415, 418, 498, 503 $\texttt{ss}, \ \textbf{99}, \ \textbf{100}, \ \textbf{104}, \ 106\text{--}109, \ 113\text{--}115, \ 132, \\$ 179ss_augment*, 168, 169 ss2ss, 156, 162, 165 sscanform*, 162, 163

ssdata, 381, 382 ureal, 375, 376, 384 st_contr*, 417, 418 usample, 375, 376 staircase_wave.mdl*, 257 stairs, 37, 38, 141, 228, 416, 422, 424 std_tf*, 312, 313 stem, 37, 38, 76, 421 stem3, 41 step, 129, 130, 132, 175, 176-180, 184, 194, 226, 242 strcmp, 511 strrep, 492, 508 subplot, 38, 45, 200, 226, 299, 300, 305, 307, 416, 424, 460 subs, 25, 70, 167, 171, 176, 182, 483, 488 subsasgn*, 493 subsref*, 493 sum, 27, 28, 164, 168, 446, 450, 459, 486, 488, 509, 514, 515 surf, 42, 44, 45, 174, 482 surfc, 42, 60 surfl, 60 svd, 65 sym, 19, 26, 62, 66-68, 112, 169, 170 sym2poly, 110 sym2tf*, 110 syms, 63, 66, 67, 70, 80, 81, 86, 87, 118, 166, 169, 170, 172, 181 sys2smat*, 374, 375, 386, 397

V varargin, 33, 49, 50 view, 44, 45 vpa, 19, 62

W

warning, 495 waterfall, 60 writefis, 434

X xlim, 44, 305, 307, 497 xor, 24

Y

ylim, 182, 190

Ľ

zero, 153zeros, 73, 74, 100, 106, 107, 109, 122, 124, 168, 256, 306, 307, 326, 388 ziegler*, 328, 329, 331, 332, 335 **zoom**, 56 zpk, 101, 104, 109, 124, 129, 132, 163, 229, 241, 268, 280, 364, 382, 403, 503 ztrans, 88, 170, 171

T

tf, 97, 98, 102, 103, 104, 106-108, 113,

 $\texttt{tfd2step}, \ \textbf{421}, \ 422, \ 424$ tfdata, 98, 313, 493, 511, 513 timmomt*, 122, 123, 126 totaldelay, 128 tournSelect, 462 trace, 63 trim, 240tzero, 204

tan, 35, 328, 497, 501

176tf2sym*, 110

U

ufopdt*, 343, 344 uigetfile, 56 uminus*, 495unpck, 396, 397

Modeling, Analysis and Design of Control Systems in MATLAB and Simulink

Index

A

A-MIGO algorithm, see approximate MI-GO algorithm, 349 A/D converter, 524, 525, 531 Abel theorem, 175 ACC benchmark problem, 540, 541, 543 Ackermann algorithm, 275 activation function, 443, 449 ActiveX, 56, 57 actuator saturation, 217, 230, 283, 320, 350, 353, 354, 445, 519additional parameter, 250, 254-257, 261 additive uncertainty, 376, 377 AIC, 139, 140 AIRC model, 543 Akaike information criterion, see AIC algebraic simplification, 119-121 analytical solution, 13, 19, 62, 67, 68, 74, 80, 81, 86, 88, 105, 152, 153, 166, 167, 277, 281, 485, 489-491, 506 anonymous function, 33, 71, 72, 75, 77, 82, 83, 85 anti-windup, 323, 324, 353 approximate differentiator, 324, 411 approximate MIGO algorithm, 348, 349 Arduino, 12, 523, 533-537 artificial intelligence, 407 ARX model, 135, 138 augmentation, 167 augmented system, 168, 169, 369, 371, 373-375, 379 auto-regressive exogenous model, see ARX model

automatic tuning, 10, 263, 293, 294, 319

\mathscr{B}

back-propagation algorithm, 444, 447 backward Euler algorithm, 322 balanced realization, 108, 131 base order, 490, 496, 497, 500, 501 basic set of solutions, 67, 89 Bass-Gura algorithm, 274 benchmark problem, 12, 91, 473, 539-543 bilinear transform, see also Tustin transform, 25, 105, 380, 381 binomial, 13, 478 bisection method, 58 block diagram, 6, 7, 11-13, 95, 116, 117, 119, 209, 210, 219, 225, 233, 243, 502, 509, 539 block library, 11, 210, 211, 249, 250, 524, 534Bode diagram, 52, 123, 130, 133, 188-196, 263, 264, 503, 504, 506, 514 BP, see back-propagation algorithm

C

Caputo definition, 478–483, 487, 488 Cauchy integral formula, 478 characteristic equation, 182, 186, 187, 274 Chien–Hrones–Reswick algorithm, 319, 335, 336 Cholesky decomposition, 65 closed-loop model reduction, 380

commensurate-order, 490, 496, 497, 500-

autocorrelation function, 141

Modeling, Analysis and Design of Control Systems in MATLAB and Simulink

502, 519

- complementary sensitivity function, 372, 405
- complimentary sensitivity function, 372
- conditional structure, 18, 27, 28
- connection matrix, 119, 120
- constrained MPC design, 423, 424
- constrained optimization, 81–84, 357, 423, 425, 463, 544
- constraint, 82, 83, 271, 281–283, 387, 388, 390, 393, 399, 400, 422–425, 427–430, 465, 466
- continuous model identification, 141, 142 Control Desk, 523–525, 531, 532
- Control System Toolbox, 8, 68, 108, 110, 112, 114, 126, 137, 151–203, 209, 213, 216, 219, 226, 263, 272, 275, 290, 321, 322, 344, 345, 349, 352, 354, 361, 373, 390, 425, 492, 499–501
- controllability, 11, 61, 62, 108, 151, 152, 157–160, 276
- controllability Gramian, 108, 158
- controllable staircase form, 160, 162, 163
- controller reduction, 10
- controls, 47, 51, 75, 531, 532
- covariance matrix, 360, 361, 363
- critical gain, 151, 183, 185–187, 501, 502 crossover frequency, 263–266, 269, 270
- curve fitting, 61, 81, 84, 85, 325

D

- d-step ahead, 413, 414, 419 D/A converter, 525, 531 dead zone, 13, 218, 219 decoupling, 11, 178, 188, 202, 264, 297, 307, 308, 310–314 defuzzification, 433, 434 describing function, 210, 237 determinant, 62, 63, 88 diagonal dominant, 189, 200–202, 297– 301, 316 Diophantine equation, 411–413 discrete Lyapunov equation, 68 discrete Riccati equation, 69, 272 discrete-time PID controller, 213, 322 discrete-time state space, 95, 104, 105, 153
- discrete-time state space, 55, 104, 105, 155 discrete-time transfer function, 95, 103,
- 104, 106, 134, 142, 146, 170, 184, 192,

229

discretization, 105, 177 disturbance, 155, 194, 195, 235, 276, 308, 320, 359, 360, 363, 364, 377, 398, 409, 412, 416, 432, 534, 540 disturbance rejection, 339, 399 dominant mode method, 125 dSPACE, 12, 523–525, 531–533 dual, 160, 161 dynamic compensation matrix, 301 dynamic decoupling, 263, 313 dynamic matrix, 419, 420

Ë

eigenvalue, 4, 63, 64, 151–153, 161, 198, 199, 276, 298, 388 eigenvector, 4, 63, 161, 298 Euler algorithm, 75 explicit ODE, 75

Ŧ

F-14 aircraft model, 539, 540, 543 feasible solution, 83, 328, 388, 465 feedback connection, 110, 111, 113, 493 final value, 232 first-order plus dead-time, see FOPDT fixed-step algorithm, 221, 236, 530 FOLIPD plant, 342 FOPDT plant, 319, 324, 326-328, 336, 338, 341, 343, 344, 513 forgetting factor, 144, 416, 461 FOTF class, 491-501, 504, 511, 514, 515, 517fractional-order calculus, 477 Caputo definition, see \sim Cauchy integral formula, see \sim Grünwald–Letnikov definition, see \sim Riemann–Liouville definition, see \sim full-rank matrix, 63, 157, 160, 164, 275, 384 fully decoupled, 306, 311, 543 fuzzy inference, 407, 432–438, 441 Fuzzy Logic Toolbox, 217, 433, 434 fuzzy PID controller, 439-442 fuzzy set, 407, 432-435

Index

G

gain margin, 195, 196, 326 GAOT, 353, 462, 463 Gaussian disturbance, 361, 365 Gaussian white noise, 235 generalized eigenvalue problem, 388, 390 generalized inverse, 64 generalized Lyapunov equation, see also Sylvester equation, 68 generalized minimum variance, 417 generalized predictive control, 11, 429-432 Genetic Algorithm Optimization Toolbox, see GAOT genetic algorithm, 356, 407, 462-468, 470 Gershgorin circles, 198–200, 207, 299 Gershgorin theorem, 198, 207 Global Optimization Toolbox, 353, 462, 463, 466 GPC, see generalized predictive control $gradient, \, 463$ Gramian, 131, 132, 158 graphical user interface, 10, 17, 37, 45-57, 137, 421, 441, 517 Grünwald-Letnikov definition, 478-483,

486, 502 GUI, see graphical user interface, 535

\mathscr{H}

 \mathcal{H}_2 norm, 165, 498 Hankel matrix, 59, 274 Hankel norm, 133 hardware-in-the-loop, 523, 524, 529, 533 Arduino, see \sim dSPACE, see \sim Quanser, see \sim Hardy space, 9, 359 Heaviside function, 171 Hebb learning algorithm, 444 Hermitian matrix, 64, 387 Hermitian transpose, 22, 65 heuristic control, 418 HIL, see hardware-in-the-loop \mathcal{H}_{∞} norm, 9, 165, 390, 392, 394, 497, 498 Hurwitz criterion, 151, 152 hyper stability, 409

Ĭ

IAE, 336, 337, 339 identification, see system identification ILC, see iterative learning control implicit differential equation, 259 implicit function, 39, 70 impulse response, 179, 180, 418, 489 incremental PID controller, 323, 447 individual, 462 initial condition, 81, 90, 181, 487 input-output stability, 155 integral criterion, 261, 284 IAE criterion, see IAE ISE criterion, see ISE ITAE criterion, see ITAE integral separation, 12, 323, 477-483 intelligent control, 10, 11, 407 internal delay, 100, 104, 108, 114-116, 153, 179, 201, 213, 226, 453 internal stability, 155 internal state, 109, 157, 159 inverse Laplace transform, 86, 87, 169, 507 inverse matrix, 12, 64 inverse Nyquist array, 151, 196-198, 200-202, 263, 297-301 inverted pendulum, 316, 526, 542 IPD plant, 341 ISE, 127, 128, 288, 289, 336, 353, 354, 514 ITAE, 281, 285–289, 309, 336, 337, 353, 354, 469, 513

iterative learning control, 455-461

5

Jordanian canonical form, 161, 163 Jordanian matrix, 162 Jury table, 151, 153, 154

\mathscr{K}

Kalman decomposition, 160, 161 Kalman filter, 2, 5, 359–361, 363–366, 368 Kronecker product, 303

L

Laplace transform, 61, 86–88, 96, 97, 103, 107, 128, 169, 174, 180, 321, 477, 480,

Modeling, Analysis and Design of Control Systems in MATLAB and Simulink

489-491, 502 inverse Laplace transform, see \sim numerical inverse Laplace transform, see \sim numerical Laplace transform, see \sim lead-lag compensator, 11, 263-267, 269, 270, 281, 282, 293, 294, 314 least squares, 23, 84, 85, 92, 135, 144, 254, 255, 303, 304, 325 limit cycle, 237 linear quadratic, 11, 270-272, 281, 315, 359, 360, 390 linear quadratic criterion, 9, 272 linear time invariant, see LTI linearization, 210, 237, 239, 241, 242, 350, 543LMI Toolbox, 377 load disturbance, 320 loop shaping, 348, 372, 373, 377, 382, 384, 402, 513 loop structure, 27, 32, 84, 139, 267, 331, 409loop transfer recovery, see also LTR, 359, 360, 366 Lorenz equation, 90, 260 LQG, see linear quadratic Gaussian LTI, 96, 99, 105, 107, 108, 112, 114, 152, 165, 175, 179, 180, 201, 213, 226, 350, 352, 373 discrete-time state space, see \sim discrete-time transfer function, see \sim multivariable system, see \sim SS object, see \sim state space, see \sim TF object, see \sim transfer function, see \sim zero–pole–gain, see \sim ZPK object, see \sim LTR, see also loop transfer recovery, 366-368 LU decomposition, 13, 65 Luenberger canonical form, 161, 163–165 Luenberger observer, 2 Lyapunov equation, 66–68, 128, 158, 160 discrete Lyapunov equation, see \sim generalized Lyapunov equation, see Sylvester equation Lyapunov inequality, see \sim Lyapunov inequality, 388, 394 Lyapunov Theorem, 152, 388

М

M-circle, 189, 190, 193 M-function, 18, 30-34, 71, 72, 75, 76, 82, 83, 250, 251, 256, 261, 508 M-sequence, see PRBS, 134, 141 Maclaurin series, 122, 126 masked block, 11, 210, 243-249, 257, 441, 447, 449, 509, 510 MATLAB toolboxes Control System Toolbox, see \sim Fuzzy Logic Toolbox, see \sim Genetic Algorithm Optimization Toolbox, see \sim Global Optimization Toolbox, see \sim LMI Toolbox, see \sim Model Predictive Control Toolbox, see \sim Multivariable Frequency Design Toolbox. see \sim Neural Network Toolbox, see \sim Optimization Toolbox, see \sim Particle Swarm Optimization Toolbox, see \sim QFT Toolbox, see \sim Robust Control Toolbox, see \sim Signal Processing Toolbox, see \sim System Identification Toolbox, see \sim matrix equation, 61, 66, 72-74, 122 maximum principle, 281 measurement noise, 155, 320, 359, 360, 363 membership function, 433, 435-437 memory, 19, 457 MESA Box, 536, 537 MFD Toolbox, 197, 199-201, 304 MIGO algorithm, 348 minimum realization, 109, 121, 147, 161, 280minimum sensitivity problem, 9, 405 minimum variance, 411, 414, 415, 417, 471 Mittag-Leffler function, 477, 480, 483-485, 489 mixed sensitivity problem, 371, 379, 382 Model Predictive Control Toolbox, 420, 422, 424, 426, 541, 543 model predictive control, 11, 217, 407, 418, 422, 424-428, 430, 452, 471, 544 constrained MPC design, see \sim dynamic matrix, see \sim

Index

Model Predictive Control Toolbox, see \sim unconstrained MPC design, see \sim model reduction, 121-134, 289, 380, 506, 507model reference adaptive control, 407-410, 454.455 Moore–Penrose generalized inverse, 64 motor encoder, 525 MPC, see model predictive control MRAS, see model reference adaptive system, 408 multi-rate system, 226, 231, 232 multi-valued nonlinearity, 237-239 multiplicative uncertainty, 376, 377 Multivariable Frequency Design Toolbox, see MFD Toolbox, 8 multivariable model identification, 142, 143multivariable system, 102, 105, 113, 121, 177, 180, 196, 200, 202, 204, 226, 301, 309

N

- negative definite matrix, 387
- negative feedback, 112, 118, 199, 218, 494
- Neural Network Toolbox, 8, 443, 451, 454
- neural network, 407, 442–455
- Nichols chart, 151, 190–193, 195, 206, 207, 402, 499

nodal equation, 120

- nominal system, 383, 399, 400, 428
- nominal value, 375, 376, 471, 520
- non-integer order, 477
- non-minimal phase, 349, 381, 415
- nonnegative definite matrix, 69
- nonzero initial condition, 151, 181, 213, 479

null space, 67

- numerical inverse Laplace transform, 507, 512, 520
- numerical Laplace transform, 507-509
- $\begin{array}{l} Nyquist \ plot, \ 52, \ 151, \ 189, \ 190, \ 192, \ 193, \\ 207, \ 325, \ 326, \ 330, \ 331, \ 365, \ 366, \ 368, \\ 369, \ 499 \end{array}$

Ø

- object-oriented, 3, 8, 12, 46, 47, 424, 477, 491
- objective function, 82, 83, 127–129, 165, 281, 282, 284–286, 288, 289, 352, 353, 355
- observability, 11, 61, 62, 108, 151, 152, 157, 159, 160
- observability Gramian, 108, 160
- observable staircase form, 160
- observer, 9, 11, 157, 160, 263, 270, 276–280, 314, 359, 360, 377
- observer-based, 11, 263, 270, 279, 280, 362, 404
- *OCD*, 263, 284, 285, 287–289, 296, 297, 308–310, 351, 352, 467–470
- ODE, see also ordinary differential equation, 74, 96
- operating point, 239-241, 259, 420, 543
- optimal PID controller, 11, 286–288, 319, 336–338, 351–356, 468, 516, 544
- $OptimFOPID,\ 517-519$
- Optimization Toolbox, 8, 83, 84, 353, 463, 464
- optimization constrained optimization, see \sim linear programming, see \sim quadratic programming, see \sim
- unconstrained optimization, see \sim
- *OptimPID*, 11, 319, 351–356, 470, 544 *order selection*, 139
- ordinary differential equation, see also ODE, 61, 74–81, 86, 166, 175, 209 analytical solutions, see \sim Caputo equation, see \sim
 - Euler algorithm, see \sim
 - fractional-order differential equation, see \sim
 - implicit differential equation, see \sim
 - numerical algorithm, see \sim
 - Runge–Kutta algorithm, see \sim
- stiff equation, see \sim
- orthogonal, 65
- Oustaloup filter, 502–506, 509, 516
- overload, 20, 68, 179, 189, 424, 477, 491, 493, 494, 499, 501, 512, 519
- overshoot, 166, 175–177, 183, 264, 266, 268, 282, 283, 295, 296, 320, 323, 331, 333, 335, 336, 341, 345, 347

Modeling, Analysis and Design of Control Systems in MATLAB and Simulink

${\mathscr P}$

Padé approximation, 122-128, 179, 186, 187, 226, 241, 506 parallel connection, 110, 111, 201, 493, 494 parallel PID controller, 321, 322 parallel searching, 407, 463, 466 parameter optimization method, 9, 263, 296, 297, 302-308, 316 partial fraction expansion, 490 partial pole placement, 276 particle swarm algorithm, 11, 407, 408, 462, 467, 470 Particle Swarm Optimization Toolbox, 353, 356, 466 pattern search, 466, 469, 470 performance index, 271, 412, 420, 430 phase margin, 195, 206, 263-268, 347, 359, 364.404 PID controller, 11, 212, 244, 249, 286, 319-356, 407 anti-windup, see \sim approximate MIGO algorithm, see \sim automatic tuning, see \sim discrete-time Ziegler-Nichols tuning, see \sim fractional-order PID controller, see \sim fuzzy PID controller, see \sim incremental PID controller, see \sim integral separation, see \sim MIGO algorithm, see \sim optimal PID controller, see \sim OptimPID, see \sim parallel PID controller, see \sim PID-type ILC, see \sim PID_Tuner, see \sim refined Ziegler–Nichols tuning, see \sim standard PID controller, see \sim Ziegler–Nichols tuning, see \sim PID-type ILC, 457-461 piecewise nonlinearity, 237, 238, 249, 261 pole placement, 5, 11, 263, 270, 273-276, 312, 313, 315 Ackermann algorithm, see \sim Bass–Gura algorithm, see \sim partial pole placement, see \sim robust pole placement, see \sim pole-zero excess, 96 population, 462–464, 468 positive definite matrix, 65, 377, 388, 394

PRBS, 141–143 proper system, 96 pseudo code, 34 pseudo inverse, 64 pseudo random binary sequence, see PRB-S pseudo-diagonalization, 263, 297, 298, 300 pseudo-random binary sequence, 141

positive feedback, 111, 186, 187

- PSOt, see Particle Swarm Optimization Toolbox
- PWM signal generator, 524

2

- QFT, see also quantitative feedback theory, 397–404, 505, 506
- QFT Toolbox, 9, 397-399
- $quadratic \ equation, \ 69$
- quantitative feedback theory, see also QFT, 11, 360, 397

\mathcal{R}

- radial basis function, 449 ramp response, 180 rank, 61-63, 65, 67, 157, 276 RBF, see radial basis function real-time control, 143, 217, 283, 385, 523, 525, 529-531, 533 realizable system, 96 recursive algorithm, 32, 122, 128, 412, 456 recursive identification, 143, 144, 254, 255, 408, 415, 430 reduced row echelon form, 67 reference model, 341, 408-411, 422 refined PID controller, 332–334 refined Ziegler-Nichols tuning rule, 333, 334relative order, 96, 411, 459 repeated eigenvalue, 66, 162 repeated poles, 163, 170, 490 return difference, 198, 366, 404 Riccati differential equation, 271, 272 Riccati equation, 69, 73, 272, 273, 315, 361, 362Riemann-Liouville definition, 478, 479
- rise time, 175, 177, 295, 296
- Robust Control Toolbox, 8, 132, 363, 367,

Index

373, 375, 377, 378, 381, 382, 385, 390– 393, 396 robust pole placement, 275 robust stability, 369, 372, 399–401 robustness, 9, 359, 366, 398, 513 root locus, 2, 11, 52, 151, 182–188, 263, 291, 292, 501 Routh approximation, 124, 125 Routh criterion, 152 Runge–Kutta algorithm, 75

S

S-function, 144, 216, 250-257, 430, 440, 441, 445, 447, 449, 523 sampling interval, 98, 103-106, 227, 425, 427, 428, 440, 448 saturation, 35, 235, 250, 320 Schur complement, 389, 391 Schur decomposition, 132, 133, 272 self-tuning, 10, 11, 407-409, 411, 412, 414-417 semi-positive definite matrix, 360-362 sensitivity function, 366, 372, 399, 405 separation principle, 362, 363 series connection, 110, 118, 201, 259 series controller, 264, 320 servo system, 175, 264, 281, 284, 289, 339, 354, 447, 455, 524, 541-543 set-point, 118, 323, 333, 335, 337, 416, 425, 427, 428, 446, 534, 535 settling time, 166, 176, 177, 205, 282, 295, 321 Sigmoid function, 444 signal flow graph, 119, 120 signal generator, 232, 254, 256 Signal Processing Toolbox, 8 similarity transform, 64, 108, 156, 164 simulation parameter, 221-223, 428 Simulink, 376, 409, 411, 414, 421, 426 single-valued nonlinearity, 237-239 small gain theorem, 369 SS object, 100, 102, 114, 162, 373, 453 stability, 11, 61, 62, 124, 125, 151, 152, 405, 409, 477, 496, 497, 513 Hurwitz criterion, see \sim input–output stability, see \sim internal stability, see \sim Jury criterion, see \sim

Routh criterion, see \sim stability margin, see \sim stable boundary, see \sim stability margin, 364, 365 stable boundary, 400, 401, 496, 498 staircase waveform, 257, 354, 355, 453, 454, 460 standard PID controller, 321, 332 standard transfer function, 264, 312, 313, 405, 409 state feedback, 157, 263, 270, 272-276, 278-280 state space equation, 96, 156, 157, 250, 252, 255, 272, 275, 361, 369, 378, 392, 541.543 state transition matrix, 61, 89 static nonlinearity, 210, 238, 239, 250 steady-state error, 175, 323, 342, 349, 454, 455steady-state value, 174, 175, 281, 289, 296, 320, 326, 438 step response, 127, 171, 174-180, 324-326 stiff differential equation, 75, 77, 509 stochastic system, 210, 236 sub-optimal model reduction, 127-134 sub-optimal reduction, 128, 130, 131, 133, 134, 327, 339, 478 subspace, 159-161 superposition theorem, 181 switch structure, 29 switching system, 210, 226, 234, 235 Sylvester equation, 66, 68 Sylvester matrix, 412 symmetrical matrix, 68, 69, 158, 271, 387, 388, 391, 392, 395 System Identification Toolbox, 139, 217 system identification, 11, 134-144, 453 Akaike information criterion, see AIC ARX model, see \sim continuous model identification, see \sim least squares, see \sim multivariable model identification, see \sim order selection, see \sim PRBS signal, see \sim recursive algorithm, see \sim

system matrix, 301, 374, 375, 385, 396, 404

Modeling, Analysis and Design of Control Systems in MATLAB and Simulink

T

template, 398-401, 515

terminate time, 75, 175, 271, 272, 285, 286, 288, 289, 354, 355, 357, 427, 469, 514, 517 TF object, 98, 373, 492, 504 threshold, 234, 443 time domain response, 181, 207 time moment, 122, 123, 126 time varying system, 210, 226, 230-233, 350, 354, 397, 523 tracker-differentiator, 252, 254, 261 tracking error, 281, 372, 457, 459, 460 training, 443, 444, 452, 454, 455 transfer function, 86, 96-98, 105, 107-111, 114, 120 discrete-time transfer function, see \sim fractional-order transfer function, see standard transfer function, see \sim transfer function matrix, 102, 107, 108, 113, 115, 120, 121 trial structure, 18, 27, 29 triangular decomposition, see also triangular decomposition, 65 Tustin transform, see also bilinear transform, 105, 106, 381 two degree-of-freedom controller, 404 $two-port\ system,\ 371-375,\ 377-379,\ 405$

U

ultimate gain, 326, 329

- uncertain system, 360, 372, 373, 375, 376, 383, 384, 398–401, 404, 438
- $uncertainty, \ 360, \ 372, \ 377, \ 384, \ 397, \ 398, \\ 456$

additive uncertainty, see \sim

multiplicative uncertainty, see \sim

unconstrained MPC design, 420, 422, 423

- unconstrained optimization, 61, 81, 82, 420, 422, 423, 425, 462, 463, 467
- unit circle, 153, 154, 185, 190, 195
- unity negative feedback, 114, 115, 153, 154, 173, 182, 496

universe, 433–435, 441

unstable FOPDT model, 343

V

variable precision algorithm, 19 variable-step algorithm, 75, 221, 222

W

weighting matrix, 9, 144, 271–273, 280, 281, 360, 365, 366, 386
white noise, 211, 360, 414, 471, 539
WinCon, 523, 525, 530
working cycle, 455–461

Y

Youla parameterization, 9, 386

Ľ

- $z \ transform, \ 11, \ 86-88, \ 103, \ 170$
- zero initial condition, 103, 172, 486
- zero-pole-gain model, 11, 96, 100, 101, 104, 105, 107, 109, 152, 176
- Ziegler-Nichols method, 9, 319, 328–333, 335, 336

ZPK object, 101, 124