TALLINN UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

Department of Computer Control

Aleksei Tepljakov

Fractional-order Calculus based Identification and Control of Linear Dynamic Systems

Master thesis

Supervisors: Eduard Petlenkov,

Juri Belikov,

Department of Computer Control,

Tallinn University of Technology

TALLINN 2011

Declaration: I hereby declare that this master's thesis, my original investigation and achievement, submitted for the master's degree at Tallinn University of Technology, has not been submitted for any degree or examination.

Deklareerin, et käesolev magistritöö, mis on minu iseseisva töö tulemus, on esitatud Tallinna Tehnikaülikooli magistrikraadi taotlemiseks ja selle alusel ei ole varem taotletud akadeemilist kraadi.

Aleksei Tepljakov

Contents

Abstract **6**

Kokkuvõte **7**

Acknowledgements **8**

1 Introduction **9**

1.1 Fractional-order Calculus 9

1.2 Outline of the Thesis 10

2 Fractional Calculus in System Theory **12**

2.1 Mathematical Background 12

2.1.1 Definitions 13

2.1.2 Properties 14

2.1.3 Examples 14

2.2 Laplace Transform 16

2.3 Fractional-order Models 17

2.4 Fractional System Analysis 19

2.4.1 Stability 19

2.4.2 Time Domain Analysis 20

2.4.3 Frequency Domain Analysis 22

2.5 Approximation of Fractional Operators 22

2.6 Discretization 23

3 Identification by Fractional Model **24**

3.1 Identification Basics 24

3.2 Time-domain Identification 25

3.3 Frequency-domain Identification 28

4 Fractional-order Control **32**

4.1 Fractional PID Controller 32

4.1.1 Introduction 32

4.1.2 Effects of Fractional Control Actions 33

4.1.3 Tuning and Optimization 35

4.2 Fractional Lead-Lag Compensator 37

4.3 TID Controller 38

5 FOMCON Toolbox for MATLAB **40**

5.1 Introduction 40

5.2 Overview 42

5.3 Used Notations 43

6 FOMCON User Manual **45**

6.1 System Analysis Module 45

6.1.1 The FOTF Object 45

6.1.1.1 Function `fotf()` 46

6.1.1.2 Function `newfotf()` 47

6.1.1.3 Block Interconnection Functions 48

6.1.1.4 Trimming Functions 50

6.1.2 Stability Analysis 51

6.1.2.1 Function `isstable()` 51

6.1.3 Time Domain Analysis 52

6.1.3.1 Function `lsim()` 52

6.1.3.2 Function `step()` 53
6.1.3.3 Function `impulse()` 54
6.1.4 Frequency Domain Analysis 55
6.1.4.1 Function `freqresp()` 55
6.1.4.2 Function `bode()` 55
6.1.4.3 Function `nyquist()` 56
6.1.4.4 Function `nichols()` 56
6.1.4.5 Function `margin()` 57
6.1.5 Fractional-order System Approximation 58
6.1.5.1 Function `oustapp()` 58
6.1.5.2 Object `fsparam()` 59
6.1.6 Object Conversion Functions 60
6.1.7 Graphical User Interface 61
6.1.7.1 Function `fotf_gui()` 61
6.1.8 System Analysis Examples 63
6.2 System Identification Module 67
6.2.1 Time Domain Identification 67
6.2.1.1 Object `fidata()` 67
6.2.1.2 Function `fid()` 69
6.2.2 Frequency Domain Identification 71
6.2.2.1 Object `ffidata()` 71
6.2.2.2 Function `ffid()` 72
6.2.2.3 Function `ffid_bf()` 73
6.2.3 Graphical User Interfaces 74
6.2.3.1 Function `fotfid()` 74
6.2.3.2 Function `fotfrid()` 76
6.2.4 Identification Examples 78

6.3 System Control Module 85
6.3.1 Fractional-Order Controller Design 85
6.3.1.1 Function `fracpid()` 85
6.3.1.2 Function `tid()` 86
6.3.1.3 Function `frlc()` 86
6.3.2 Integer-Order Controller Tuning By Process Model Approximation 87
6.3.2.1 Function `fotf2io()` 87
6.3.3 PI$^{\lambda}$D$^{\mu}$ Controller Optimization 88
6.3.3.1 Object `fpopt()` 88
6.3.3.2 Function `fpid_optimize()` 90
6.3.4 Graphical User Interfaces 91
6.3.4.1 Function `fpid()` 91
6.3.4.2 Function `fpid_optim()` 93
6.3.4.3 Function `iopid_tune()` 94
6.3.5 Controller Design and Optimization Examples 96
6.4 Simulink Blockset 102
6.4.1 Library Overview 102
6.4.2 Block Description 103
6.4.2.1 Fractional operator 103
6.4.2.2 Fractional derivative 104
6.4.2.3 Fractional integrator 104
6.4.2.4 Fractional Transfer Fcn 105
6.4.2.5 Fractional PID controller 106
6.4.2.6 TID controller 106
6.4.2.7 Discrete fractional Transfer Fcn 107
6.4.2.8 Discrete fractional PID controller 108
6.4.3 Examples 108

Discussion **112**

Conclusions **115**

Bibliography **122**

List of Publications **125**

Abstract

Fractional-order Calculus based Identification and Control of Linear Dynamic Systems

Present thesis is devoted to the research of fractional-order calculus in the context of modeling, identification and control of dynamic systems. The first part of the thesis introduces the reader to the theoretical concepts behind fractional-order calculus and its applications in system theory, system identification and control. In the second part the FOMCON ("Fractional-order Modeling and Control") toolbox for MATLAB, developed by the author, is presented and a detailed user manual for it is provided with illustrative examples.

All major calculations and simulations performed in this thesis are done in the MATLAB/Simulink environment. Additionally, some examples are performed using Maxima and Scilab computer algebra systems.

It is assumed that the reader has previous knowledge in the field of system theory.

Kokkuvõte

Murrulistel tuletistel põhinev lineaarsete dünaamiliste süsteemide identifitseerimine ja juhtimine

Antud väitekiri on pühendatud murruliste tuletiste rakenduste uurimisele dünaamiliste süsteemide modelleerimise, identifitseerimise ja juhtimise kontekstis. Väitekiri esimese osa sisuks on murruliste tuletistega seotud teooria. Samuti on antud selle rakendus süsteemiteoorias, mudeli identifitseerimisel ja süsteemide juhtimisel. Teine osa keskendub uuele, väitekirja autori poolt loodud MATLAB arvutuspaketile, mille nimeks on FOMCON ("Fractional-order Modeling and Control" ehk "Murdarvuline Modelleerimine ja Juhtimine"). Põhjalik arvutuspaketi kasutusjuhend on toodud selgitustega ning näidetega.

Kõik töös esitatud arvutused ja simulatsioonid on tehtud MATLAB/ Simulink keskkonnas. Mõned näited on tehtud ka Maxima ja Scilab arvutusplatformide abil.

Eeldatakse, et lugejal on olemas eelnevad teadmised süsteemiteooria valdkonnast.

Acknowledgments

This work was supported the Estonian Science Foundation Grant no. 8738.

Chapter 1

Introduction

1.1 Fractional-order Calculus

The concept of the differentiation operator $\mathscr{D} = \mathrm{d}/\,\mathrm{d}\,x$ is a well-known fundamental tool of modern calculus. For a suitable function f the n-th derivative is well defined as $\mathscr{D}^n f(x) = \mathrm{d}^{\,n} f(x)/\,\mathrm{d}x^n$, where n is a positive integer. However, what happens if we extend this concept to a situation, when the order of differentiation is arbitrary, for example, fractional? That was the very same question L'Hôpital addressed to Leibniz in a letter in 1695. Since then the concept of fractional calculus has drawn the attention of many famous mathematicians, including Euler, Laplace, Fourier, Liouville, Riemann, Abel and Laurent. But it was not until 1884 that the theory of generalized operators reached a satisfactory level of development for the point of departure for the modern mathematician [1].

However, fractional-order calculus was not particularly popular until recent years when benefits stemming from using its concepts became evident in various scientific fields, including system modeling and automatic control. It is also apparent that this rise of interest is related to accessibility of more efficient and powerful computational tools provided by the evolution of technology and introduction of computer algebra systems (CAS), such as *MATLAB* and *Mathematica*.

Recent findings support the notion that fractional-order calculus should be employed where more accurate modeling and robust control are concerned. Specifically, fractional-order calculus found its way into complex mathematical and physical problems [6], [10]. In general, fractional-order calculus may be useful when modeling any system

which has memory and/or hereditary properties [4].

In the field of automatic control fractional calculus is used to obtain more accurate models, develop new control strategies and enhance the characteristics of control systems. Several toolboxes have been developed for this particular set of tasks. Among them are MATLAB toolboxes *CRONE* [7], developed by the CRONE team, and *NINTEGER* , developed by Duarte Valério [8].

This thesis is devoted to the research of possibilities provided by fractional calculus in system theory, identification and control and describes a new MATLAB toolbox *FOMCON*, developed by the author of the thesis, in which the corresponding features are implemented. The toolbox is built upon an existing toolset [2] and provides new functionality, convenience functions, various utilities and a set of graphical user interfaces (GUIs) to facilitate the workflow.

1.2 Outline of the Thesis

The thesis is organized as follows. In Chapter 2 the reader is presented with an overview of the mathematical concepts behind fractional-order calculus and its application in system theory. This chapter also introduces the reader to the notion of fractional-order models and basics of fractional-order system analysis. Finally, fractional system approximation by integer-order filters and discretization are also discussed.

In Chapter 3 methods for identifying a system by fractional-order model are presented. Both time-domain and frequency-domain identification algorithms are discussed.

Chapter 4 introduces the reader to the concept of fractional-order controllers. These include the fractional PID controller, generalized lead-lag compensator and the Tilt-Integral-Derivative controller. Methods on tuning and optimizing the fractional-order PID controller are also presented.

In Chapter 5 the reader is made familiar with the FOMCON toolbox for MATLAB, motivation behind its development and its relation to other existing toolboxes. The FOMCON user manual follows in Chapter 6, in which all the major functions and objects of the toolbox are presented and described. Illustrative examples are also given.

In the final chapters, the author's reflections on the topic of fractional calculus in the

context of the realized toolbox are given. In the last chapter conclusions are drawn and further toolbox development perspectives (such as porting it to a different computing platform) are addressed.

Chapter 2

Fractional Calculus in System Theory

In this chapter we shall discuss the mathematical basis of fractional calculus and its applications in system theory. The chapter is organized as follows. In Section 2.1 definitions of fractional-order operators are given along with their properties and some examples. In Section 2.2 the Laplace transform for fractional-order operators is defined. In Section 2.3 fractional system model representations are given. In Section 2.4 basics of time-domain and frequency-domain analysis of fractional systems are presented. In Section 2.5 the reader is introduced to methods of approximating a fractional-order system by an integer-order model. Finally, in Section 2.6 some aspects of fractional system discretization are given.

2.1 Mathematical Background

Fractional calculus is a generalization of integration and differentiation to non-integer order operator ${}_a\mathscr{D}_t^\alpha$, where a and t denote the limits of the operation and α denotes the fractional order such that

$${}_a\mathscr{D}_t^\alpha = \begin{cases} \frac{\mathrm{d}^\alpha}{\mathrm{d}\,t^\alpha} & \Re(\alpha) > 0, \\ 1 & \Re(\alpha) = 0, \\ \int_a^t (\mathrm{d}\,t)^{-\alpha} & \Re(\alpha) < 0, \end{cases}$$

where generally it is assumed that $\alpha \in \mathbb{R}$, but it may also be a complex number [3].

One of the reasons why fractional calculus is not yet found in elementary texts is a certain degree of controversy found in the theory [1]. This is why there is not a

single definition for a fractional-order differintegral operator. Rather there are multiple definitions which may be useful in a specific situation. Further several commonly used definitions of fractional-order operators are presented.

2.1.1 Definitions

We first define the fractional differintegral operator according to Riemann-Liouville, which is the most widely used definition in fractional calculus [5].

Definition 2.1.1 *(Riemann-Liouville)*

$$ {}_a\mathscr{D}_t^\alpha f(t) = \frac{1}{\Gamma(m-\alpha)} \left(\frac{\mathrm{d}}{\mathrm{d}\,t}\right)^m \int_a^t \frac{f(\tau)}{(t-\tau)^{\alpha-m+1}}\,\mathrm{d}\tau\,, \tag{2.1} $$

where $m-1<\alpha<m$, $m\in\mathbb{N}$,$\alpha\in\mathbb{R}^+$ *and* $\Gamma(\cdot)$ *is Euler's gamma function.*

Next, consider an alternative definition according to Caputo.

Definition 2.1.2 *(Caputo)*

$$ {}_0\mathscr{D}_t^\alpha f(t) = \frac{1}{\Gamma(m-\alpha)} \int_0^t \frac{f^{(m)}(\tau)}{(t-\tau)^{\alpha-m+1}}\,\mathrm{d}\,\tau, \tag{2.2} $$

where $m-1<\alpha<m$, $m\in\mathbb{N}$.

Another definition is the Grünwald-Letnikov one. This definition can be especially useful due to its importance in applications.

Definition 2.1.3 *(Grünwald-Letnikov)*

$$ {}_a\mathscr{D}_t^\alpha\, f\,(t) = \lim_{h\to 0} \frac{1}{h^\alpha} \sum_{j=0}^{\left[\frac{t-a}{h}\right]} (-1)^j \binom{\alpha}{j} f\,(t-jh)\,, \tag{2.3} $$

where $[\cdot]$ *means the integer part.*

2.1.2 Properties

Fractional-order differentiation has the following properties [2], [5], [4]:

1. If $f(t)$ is an analytic function, then the fractional-order differentiation ${}_0\mathscr{D}_t^{\alpha} f(t)$ is also analytic with respect to t.

2. If $\alpha = n$ and $n \in \mathbb{Z}^+$, then the operator ${}_0\mathscr{D}_t^{\alpha}$ can be understood as the usual operator $\mathrm{d}^{\,n} / \mathrm{d}\,t^n$.

3. Operator of order $\alpha = 0$ is the identity operator: ${}_0\mathscr{D}_t^{0} f(t) = f(t)$.

4. Fractional-order differentiation is linear; if $a,\ b$ are constants, then

$$ {}_0\mathscr{D}_t^{\alpha} \left[a f(t) + b g(t) \right] = a\, {}_0\mathscr{D}_t^{\alpha} f(t) + b\, {}_0\mathscr{D}_t^{\alpha} g(t). \tag{2.4} $$

5. For the fractional-order operators with $\Re(\alpha) > 0,\ \Re(\beta) > 0$, and under reasonable constraints on the function $f(t)$ it holds the additive law of exponents:

$$ {}_0\mathscr{D}_t^{\alpha} \left[{}_0\mathscr{D}_t^{\beta} f(t) \right] = {}_0\mathscr{D}_t^{\beta} \left[{}_0\mathscr{D}_t^{\alpha} f(t) \right] = {}_0\mathscr{D}_t^{\alpha+\beta} f(t) \tag{2.5} $$

6. The fractional-order derivative commutes with integer-order derivative

$$ \frac{\mathrm{d}^{\,n}}{\mathrm{d}\,t^n} \left({}_a\mathscr{D}_t^{\alpha} f(t) \right) = {}_a\mathscr{D}_t^{\alpha} \left(\frac{\mathrm{d}^{\,n} f(t)}{\mathrm{d}\,t^n} \right) = {}_a\mathscr{D}_t^{\alpha+n} f(t), \tag{2.6} $$

   under the condition $t = a$ we have $f^{(k)}(a) = 0,\ (k = 0,\ 1,\ 2,\ ...,\ n-1)$.

2.1.3 Examples

Further, several fractional derivative computation examples are provided.

Example 2.1.1 Let us compute the Riemann-Liouville fractional derivative of order $\alpha = \frac{1}{2}$ for an elementary function $f(t) = t^2$ taking $a = 0$:

$$ {}_0\mathscr{D}_t^{\frac{1}{2}} \left(t^2 \right) \overset{RL}{=} \frac{1}{\Gamma(1-\frac{1}{2})} \frac{\mathrm{d}}{\mathrm{d}\,t} \left(\int_0^t \frac{\tau^2}{(t-\tau)^{\frac{1}{2}-1+1}} \,\mathrm{d}\,\tau \right) = \frac{1}{\sqrt{\pi}} \frac{\mathrm{d}}{\mathrm{d}\,t} \left(\frac{16 \cdot t^{\frac{5}{2}}}{15} \right) = \frac{8\, t^{\frac{3}{2}}}{3\sqrt{\pi}}. $$

It can be shown, that in this case Caputo's definition yields the same result:

$$ {}_0\mathscr{D}_t^{\frac{1}{2}}\left(t^2\right) \overset{C}{=} \frac{1}{\Gamma(1-\frac{1}{2})}\int_0^t \frac{\left(\tau^2\right)^{(1)}}{(t-\tau)^{\frac{1}{2}-1+1}}\,\mathrm{d}\,\tau = \frac{1}{\sqrt{\pi}}\int_0^t \frac{2\tau}{(t-\tau)^{\frac{1}{2}}}\,\mathrm{d}\,\tau = \frac{8\,t^{\frac{3}{2}}}{3\,\sqrt{\pi}}. $$

A CAS may be used to carry out these symbolic computations. For example, an open-source application *Maxima*[1] can be used to define the functions *RLDif(*$f(x)$, x, α, a*)* and *CapDif(*$f(x)$, x, α*)* respectively:

```
RLDif(fn,x,alp,a):=
        block ([m:[], fn:fn, it:[], v:[], x:x, _tau],
        assume(x>0, _tau>0),
        m: ceiling(alp), v: m-alp,
        fnt(_tau):=ev(subst(_tau, x, fn)),
        it: integrate(fnt(_tau)/((x-_tau)^(alp-m+1)),_tau,a,x),
        1/gamma(v)*diff(it,x,m))


CapDif(fn,x,alp):=
        block ([m:[], fn:fn, df:[], dff:[], v:[]],
        assume(x>0),
        fn(x):=ev(fn),
        m: ceiling(alp),
        v: m-alp,
        df: diff(fn(x), x, m),
        dff(_tau):=ev(subst(_tau, x, df)),
        1/gamma(v)*
        integrate(dff(_tau)/((x-_tau)^(alp-m+1)), _tau, 0, x))
```

The above examples can then be computed using newly defined functions by typing the following in *Maxima*:

```
RLDif(t^2, t, 1/2, 0)
CapDif(t^2, t, 1/2)
```

And in both cases *Maxima* returns the answer:

[1] *Maxima* can be freely downloaded from the Internet. Binary packages for every major platform are available here: `http://sourceforge.net/projects/maxima/files/`

$$\frac{8\,t^{\frac{3}{2}}}{3\,\sqrt{\pi}}.$$

Example 2.1.2 Compute the fractional derivative of order $\alpha = \frac{1}{3}$ for function $f_1(t) = \mathrm{e}^{5t}$ and the fractional derivative of order $\alpha = \frac{1}{2}$ for function $f_2(t) = \sin(3t)$.

It can be deduced, that with $a = 0$ the result of the operation ${}_0\mathscr{D}_t^{\frac{1}{3}}\left(e^{5t}\right)$ will not be an elementary function. However, taking $a = -\infty$ we will obtain a solution using the following *Maxima* command:

```
RLDif(%e^(5*t), t, 1/3, -inf)
```

The returned value is $5^{\frac{1}{3}}\,\mathrm{e}^{5\,t}$ and thus

$$_{-\infty}\mathscr{D}_t^{\frac{1}{3}}\left(e^{5t}\right) = 5^{\frac{1}{3}}\mathrm{e}^{5t}.$$

With $a = -\infty$ the definition (2.1) can be viewed as a Weyl differintegral. This modification of the definition is also useful for periodic functions, so to compute $_{-\infty}\mathscr{D}_t^{\frac{1}{2}}\left(\sin(3t)\right)$ we will use the following:

```
RLDif(sin(3*t), t, 1/2, -inf)
```

The output, after simplification, is $\frac{\sqrt{6}}{2}\left(\sin(3t) + \cos(3t)\right)$. After applying trigonometric transformations, the final result is obtained as

$$_{-\infty}\mathscr{D}_t^{\frac{1}{2}}\left(\sin(3t)\right) = \sqrt{3}\sin\left(3t + \frac{\pi}{4}\right).$$

2.2 Laplace Transform

The Laplace integral transform in an essential tool in dynamic system and control engineering. A function $F(s)$ of the complex variable s is called the Laplace transform of the original function $f(t)$ and defined as

$$F(s) = \mathscr{L}\left[f(t)\right] = \int_0^\infty \mathrm{e}^{-st} f(t)\,\mathrm{d}\,t \tag{2.7}$$

The original function $f(t)$ can be recovered from the Laplace transform $F(s)$ by applying the reverse Laplace transform defined as

$$f(t) = \mathscr{L}^{-1}\left[F(s)\right] = \frac{1}{j2\pi}\int_{c-j\infty}^{c+j\infty} \mathrm{e}^{st}F(s)\,\mathrm{d}s, \tag{2.8}$$

where c is greater than the real part of all the poles of function $F(s)$ [2].

We shall now define the Laplace transforms for the fractional-order operators defined in Section 2.1.

Definition 2.2.1 *(Laplace transform of the Riemann-Liouville fractional operator)*

$$\mathscr{L}\ \left[\mathscr{D}^{\alpha}f(t)\right] = s^{\alpha}F(s) - \sum_{k=0}^{m-1} s^{k}\left[\mathscr{D}^{\alpha-k-1}f(t)\right]_{t=0}, \tag{2.9}$$

where $(m-1 \leq \alpha < m)$.

Definition 2.2.2 *(Laplace transform of the Caputo fractional operator)*

$$\mathscr{L}\ \left[\mathscr{D}^{\alpha}f(t)\right] = s^{\alpha}F(s) - \sum_{k=0}^{m-1} s^{\alpha-k-1}f^{(k)}(0)\,, \tag{2.10}$$

where $(m-1 \leq \alpha < m)$.

Definition 2.2.3 *(Laplace transform of the Grünwald-Letnikov fractional operator)*

$$\mathscr{L}\ \left[\mathscr{D}^{\alpha}f(t)\right] = s^{\alpha}F(s). \tag{2.11}$$

2.3 Fractional-order Models

A fractional-order continuous-time dynamic system can be expressed by a fractional differential equation of the following form [2, 3]:

$$\begin{aligned} H(\mathscr{D}^{\alpha_0\alpha_1\cdots\alpha_n})\,y(t) &= G(\mathscr{D}^{\beta_0\beta_1\cdots\beta_n})\,u(t), \\ H(\mathscr{D}^{\alpha_0\alpha_1\ldots\alpha_n}) &= \sum_{k=0}^{n} a_k\mathscr{D}^{\alpha_k}, \\ G(\mathscr{D}^{\beta_0\beta_1\ldots\beta_n}) &= \sum_{k=0}^{m} b_k\mathscr{D}^{\beta_k}, \end{aligned} \tag{2.12}$$

where $a_k,\ b_k \in \mathbb{R}$.

In explicit form:

$$\begin{aligned} a_n \mathscr{D}^{\alpha_n} y(t) + a_{n-1} \mathscr{D}^{\alpha_{n-1}} y(t) + \cdots + a_0 \mathscr{D}^{\alpha_0} y(t) \quad &= \\ b_m \mathscr{D}^{\beta_m} u(t) + b_{m-1} \mathscr{D}^{\beta_{m-1}} u(t) + \cdots + b_0 \mathscr{D}^{\beta_0} u(t) & \end{aligned} \tag{2.13}$$

The system is said to be of *commensurate-order* if in (2.13) all the orders of derivation are integer multiples of a base order γ such that $\alpha_k,\ \beta_k = k\gamma,\ \gamma \in \mathbb{R}^+$. The system can then be expressed as

$$\sum_{k=0}^{n} a_k \mathscr{D}^{k\gamma} y(t) = \sum_{k=0}^{m} b_k \mathscr{D}^{k\gamma} u(t). \tag{2.14}$$

If in (2.14) the order is $\gamma = 1/q,\ q \in \mathbb{Z}^+$, the system will be of rational order. The diagram with linear time-invariant (LTI) system classification is given in Figure (2.1).

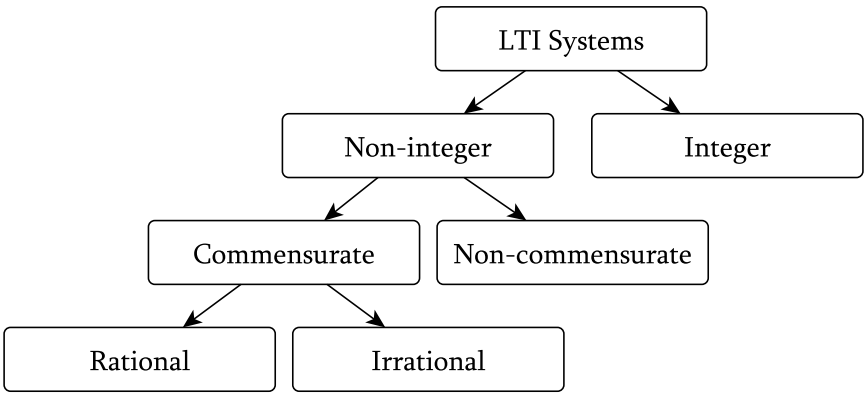


Figure 2.1: Classification of LTI systems

Applying the Laplace transform to (2.13) with zero initial conditions the input-output representation of the fractional-order system can be obtained in the form of a transfer function:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_m s^{\beta_m} + b_{m-1} s^{\beta_{m-1}} + \cdots + b_0 s^{\beta_0}}{a_n s^{\alpha_n} + a_{n-1} s^{\alpha_{n-1}} + \cdots + a_0 s^{\alpha_0}}. \tag{2.15}$$

In the case of a system with commensurate order γ, the continuous-time transfer function is given by

$$G(s) = \frac{\sum_{k=0}^{m} b_k \left(s^\gamma\right)^k}{\sum_{k=0}^{n} a_k \left(s^\gamma\right)^k}. \tag{2.16}$$

Taking $\lambda = s^\gamma$ the function (2.14) can be viewed as a pseudo-rational function $H(\lambda)$:

$$H(\lambda) = \frac{\sum_{k=0}^{m} b_k \lambda^k}{\sum_{k=0}^{n} a_k \lambda^k}. \tag{2.17}$$

Based on the concept of the pseudo-rational function, a state-space representation can be established in the form:

$$\begin{aligned} \mathscr{D}^\gamma x(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{2.18}$$

The state-space model allows representation of multiple input, multiple output (MIMO) fractional-order systems. The following equation may be used to convert the state-space representation to a transfer function:

$$G(s) = C(s^\gamma I - A)^{-1}B + D, \tag{2.19}$$

where I is the identity matrix.

2.4 Fractional System Analysis

2.4.1 Stability

In order to determine stability of a fractional system given by (2.14) consider the following theorem [11, 3].

Theorem 2.4.1 *(Matignon's stability theorem) The fractional transfer function $G(s) = Z(s)/P(s)$ is stable if and only if the following condition is satisfied in σ-plane:*

$$\left|\arg(\sigma)\right| > q\frac{\pi}{2},\ \forall\sigma \in C,\ P(\sigma) = 0, \tag{2.20}$$

where $\sigma := s^q$. When $\sigma = 0$ is a single root of $P(s)$, the system cannot be stable. For $q = 1$, this is the classical theorem of pole location in the complex plane: no pole is in the closed right plane of the first Riemann sheet.

In general, for a commensurate-order fractional-order system in the form

$$\mathscr{D}^q w = f(w),$$

where $0 < q < 1$ and $w \in R^n$ the equilibrium points are calculated by solving

$$f(w) = 0.$$

The equilibrium points are asymptotically stable if all the eigenvalues λ_k of the Jacobian matrix $J = \frac{\partial f}{\partial w}$, evaluated at the equilibrium, satisfy the condition

$$\left|\arg(\mathrm{eig}(J))\right| = \left|\arg(\lambda_k)\right| > q\frac{\pi}{2},\ k = 1,\ 2,\ ...,\ n. \tag{2.21}$$

Alternatively, the stability condition can also be evaluated from the state-space representation of the system (2.18):

$$\left|\arg(\mathrm{eig}(A))\right| > q\frac{\pi}{2}, \tag{2.22}$$

where $0 < q < 1$ and $\mathrm{eig}(A)$ represents the eigenvalues of the state-space matrix A.

Stability regions of a fractional-order system are shown in Figure 2.2.

Remark 2.4.1 *Unlike integer-order systems, a stable continuous-time fractional-order system can have roots in the right half of the complex plane.*

2.4.2 Time Domain Analysis

In the time domain it is desired to obtain a transient response of a fractional-order dynamic system. One solution would be to use the inverse Laplace transform and the

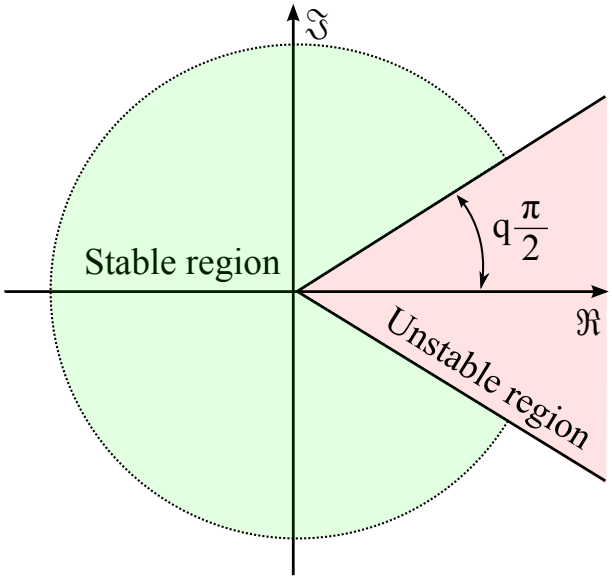


Figure 2.2: LTI fractional-order system stability region for $0 < q < 1$

Mittag-Leffler function proposed by Podlubny in [4]. However, this solution method may be time consuming and tedious.

Another solution involves numerical computation of fractional-order derivatives which is carried out by means of a revised Grünwald-Letnikov definition (2.3) rewritten as

$$ {}_a\mathscr{D}_t^{\alpha} f(t) = \lim_{h \to 0} \frac{1}{h^{\alpha}} \sum_{j=0}^{\left[\frac{t-a}{h}\right]} w_j^{(\alpha)} f(t - jh), \tag{2.23} $$

where h is the computation step-size and $w_j^{(\alpha)} = (-1)^j \binom{\alpha}{j}$ can be evaluated recursively from

$$ w_0^{(\alpha)} = 1,\ w_j^{(\alpha)} = \left(1 - \frac{\alpha + 1}{j}\right) w_{j-1}^{(\alpha)},\ j = 1,\ 2,\ \cdots . \tag{2.24} $$

To obtain a numerical solution for the equation in (2.13) the signal $\hat{u}(t)$ should be obtained first, using the algorithm in (2.23), where

$$ \hat{u}(t) = b_m \mathscr{D}^{\beta_m} u(t) + b_{m-1} \mathscr{D}^{\beta_m - 1} u(t) + \cdots + b_0 \mathscr{D}^{\beta_0} u(t). $$

The time response of the system can then be obtained using the following equation:

$$ y(t) = \frac{1}{\sum_{i=0}^{n} \frac{a_i}{h^{\alpha_i}}} \left[u(t) - \sum_{i=0}^{n} \frac{a_i}{h^{\alpha_i}} \sum_{j=1}^{\left[\frac{t-a}{h}\right]} w_j^{(\alpha)} y(t - jh) \right]. \tag{2.25} $$

2.4.3 Frequency Domain Analysis

Frequency-domain response may be obtained by substituting $s = j\omega$ in (2.15). The complex response for a frequency $\omega \in (0;\, \infty)$ can then be computed as follows:

$$R(\omega) = \frac{P(j\omega)}{Q(j\omega)} = \frac{b_m(j\omega)^{\beta_m} + b_{m-1}(j\omega)^{\beta_{m-1}} + \cdots + b_0(j\omega)^{\beta_0}}{a_n(j\omega)^{\alpha_n} + a_{n-1}(j\omega)^{\alpha_{n-1}} + \cdots + a_0(j\omega)^{\alpha_0}}, \tag{2.26}$$

where j is the imaginary unit.

2.5 Approximation of Fractional Operators

Due to availability of well-established tools for integer-order LTI systems analysis, the possibility of approximating the fractional model by an integer-order one is highly desirable. Several methods are summarized in [12].

The Oustaloup recursive filter, proposed in [13], gives a very good approximation of fractional operators in a specified frequency range and is widely used in fractional calculus.

For a frequency range $(\omega_b,\, \omega_h)$ and of order N the filter for an operator s^γ, $0 < \gamma < 1$, is given by

$$G_f(s) = K \prod_{k=-N}^{N} \frac{s + \omega'_k}{s + \omega_k}, \tag{2.27}$$

where

$$\omega'_k = \omega_b \left(\frac{\omega_h}{\omega_b}\right)^{\frac{k+N+\frac{1}{2}(1-\gamma)}{2N+1}},\ \omega_k = \omega_b \left(\frac{\omega_h}{\omega_b}\right)^{\frac{k+N+\frac{1}{2}(1+\gamma)}{2N+1}},\ K = \omega_h^\gamma. \tag{2.28}$$

A refined Oustaloup filter was proposed in [2], [5]. It is given by

$$s^\alpha \approx \left(\frac{d\omega_h}{b}\right)^\alpha \left(\frac{ds^2 + b\omega_h s}{d\,(1-\alpha)\, s^2 + b\omega_h s + d\alpha}\right) \prod_{k=-N}^{N} \frac{s + \omega'_k}{s + \omega_k}, \tag{2.29}$$

where

$$\omega_k = \left(\frac{b\omega_h}{d}\right)^{\frac{\alpha+2k}{2N+1}},\ \omega'_k = \left(\frac{d\omega_b}{b}\right)^{\frac{\alpha-2k}{2N+1}}. \tag{2.30}$$

It is expected, that a good approximation using (2.29) is obtained with $b = 10, d = 9$, which has been confirmed by theoretical analysis and experimentation.

Given these methods of approximating the fractional operator, a general method of approximation a fractional-order model by an integer-order one is now possible. Recall the property in (2.6). Thus, for fractional orders $\alpha \geq 1$ it holds

$$s^{\alpha} = s^{n} s^{\gamma}, \tag{2.31}$$

where $n = \alpha - \gamma$ denotes the integer part of α and s^{γ} is obtained by the Oustaloup approximation by using either (2.27) or (2.29), with the latter being preferred in most cases.

2.6 Discretization

Discretization is essential in implementation of controllers. Several discretization methods have been developed for fractional-order models. These include FIR (finite response filter) and IIR (infinite response filter) realizations, the latter being preferred to the former due to a lower order of this type of filter [2].

Taking into account continuous-time rational-order approximations discussed in Section 2.5, the following method for obtaining a discrete-time model can be proposed.

1. Approximate the continuous-time fractional model by a rational-order transfer function $G_c(s)$ using an Oustaloup filter.

2. Use a discrete transformation with a sample period T and obtain a discrete-order approximation $G_d(z)$ of the fractional model.

Tustin method (or bilinear transformation method) can be used. It relates the s and z domains with the following substitution formula:

$$s = \frac{2z - 1}{Tz + 1}, \tag{2.32}$$

where T is the sample period. Prewarping of the critical frequencies of $G_c(s)$ may be required so that frequency responses of $G_c(j\omega)$ and $G_d(j\omega)$ are equal after the discretization.

Chapter 3

Identification by Fractional Model

This chapter is devoted to research of methods of identification by a fractional-order model. It is organized as follows. In Section 3.1 general information on system identifications is provided. In Section 3.2 methods of model identification from experimental time-domain data are given. And in Section 3.3 we take a look at identification methods using frequency-domain data.

3.1 Identification Basics

The goal of identification is to infer a dynamic system model based upon data, measured during an experiment. In general, it is necessary to obtain a relationship between system inputs and outputs under external stimuli (input signals, disturbances) in order to determine and predict the system behavior. A general form of a single input-single output system with disturbances is given in Figure 3.1.

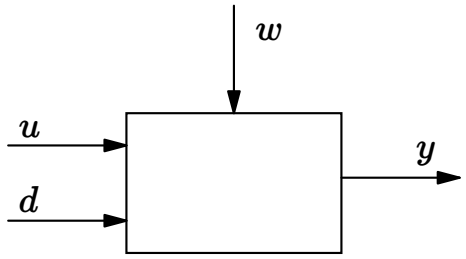


Figure 3.1: A system with input u, output y, measured disturbance d and unmeasured disturbance w

The general procedure of system identification is summarized in Figure 3.2 and consists of the following stages [14].

1. Design the experiment. For dynamic systems it is usual to collect transient response data in the time-domain by applying a set of predetermined input signals, or frequency response (magnitude and phase) in the frequency domain (e.g. by doing a frequency sweep).

2. Record the dataset based on an experiment. The collected data must be as informative as possible subject to constraints at hand.

3. Choose a set of models and/or the model structure and the criterion to fit.

4. Calculate the model using a suitable algorithm, e.g. least squares method.

5. Validate the obtained model. It is desirable to use two different datasets for identification and validation.

6. If the model is satisfactory, use it for whatever purpose desired. Otherwise, revise modeling/identification strategy and repeat the above steps.

A crucial step in the whole identification process is determining the amount of contribution of noise and disturbances to the collected data, which may need filtering and processing before actually being used in the identification algorithm.

Further we investigate ways to identify a system by a fractional-order model based on time-domain and frequency-domain experiments.

3.2 Time-domain Identification

The objective of time-domain identification is to obtain a fractional model in the form

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_m s^{\beta_m} + b_{m-1} s^{\beta_{m-1}} + \cdots + b_0 s^{\beta_0}}{a_n s^{\alpha_n} + a_{n-1} s^{\alpha_{n-1}} + \cdots + a_0 s^{\alpha_0}} \tag{3.1}$$

from registered system input and output. Several methods of identification are available, including equation error and output error approaches, and are described in [15]. Based on these ideas, a more general approach can be used.

According to [11, 15] a commensurate-order transfer function $G(s)$ is BIBO (bounded input-bounded output) stable, iff

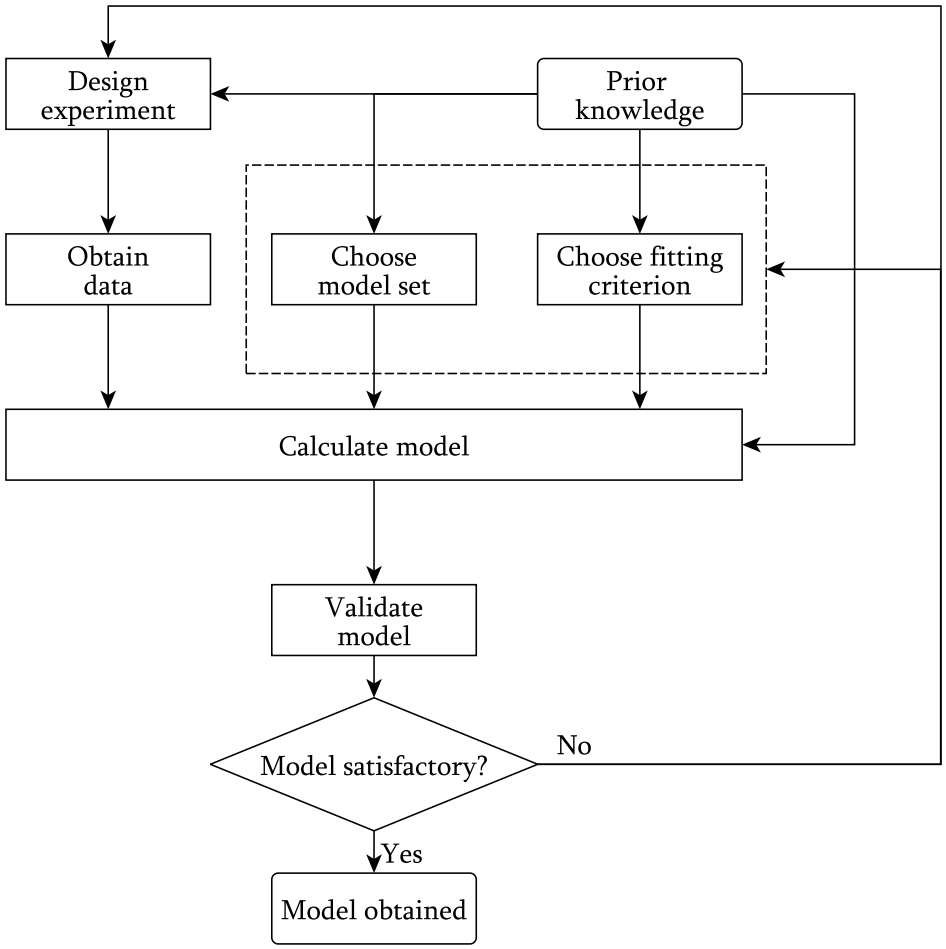


Figure 3.2: The system identification procedure

$$0 < \gamma < 2, \tag{3.2}$$

where γ is the commensurate order, and for every s^γ-pole, $s_k \in \mathbb{C}$ of $G(s)$ it holds the stability criterion in (2.21):

$$\left|\arg(s_k)\right| > \gamma\frac{\pi}{2}.$$

Thus, the initial guess model may be chosen in accordance with (3.2) by selecting a commensurate order γ. Given a fractional-order polynomial with order n the highest differential order will then be $n \cdot \gamma$.

Once experimental data is collected and an initial guess model is obtained (either randomly or from prior knowledge) the identification problem could be restated as a problem of optimizing a set of parameters θ of model (3.1) given by

$$
\begin{aligned}
a_p &= [\ a_n \quad a_{n-1} \quad \cdots \quad a_0\ ], \quad \alpha_p = [\ \alpha_n \quad \alpha_{n-1} \quad \cdots \quad \alpha_0\ ], \\
b_z &= [\ b_m \quad b_{m-1} \quad \cdots \quad b_0\ ], \quad \beta_z = [\ \beta_n \quad \beta_{n-1} \quad \cdots \quad \beta_0\ ],
\end{aligned} \tag{3.3}
$$

where a_p, b_z denote pole and zero polynomial differential operator coefficients and α_p, β_z denote the corresponding exponents (orders of differentiation).

If the number of optimized parameters is too high, the problem may be ill-conditioned. One identification strategy is to choose a commensurate-order for the model and fix the resulting fractional orders. Another involves fixing the coefficients and optimizing the exponents. Thus, given (3.3), one can see that there is a total of nine possible optimization parameter sets and they are as follows:

- full optimization, $\theta = [\ a_p \quad \alpha_p \quad b_z \quad \beta_z\ ]$,
- fix orders, optimize coefficients, $\theta = [\ a_p \quad b_z\ ]$,
- fix coefficients, optimize orders, $\theta = [\ \alpha_p \quad \beta_z\ ]$.

With pole polynomial fixed:

- optimize zero polynomial coefficients and orders, $\theta = [\ b_p \quad \beta_z\ ]$,
- fix orders, optimize zero polynomial coefficients, $\theta = b_z$,
- fix coefficients, optimize zero polynomial orders, $\theta = \beta_z$.

With zero polynomial fixed:

- optimize pole polynomial coefficients and orders, $\theta = [\ a_p \quad \alpha_z\ ]$,
- fix orders, optimize pole polynomial coefficients, $\theta = a_z$,
- fix coefficients, optimize pole polynomial orders, $\theta = \alpha_z$.

These options may be useful for large-scale problems (high-order models) to ensure a well-conditioned optimization problem.

With previous considerations, an output error method can be used with a least-square approach. The optimization criterion will be the output error norm $\left\|e\left(t\right)\right\|_2^2$ given by

$$e\left(t\right) = y\left(t\right) - \hat{y}\left(t\right), \tag{3.4}$$

where $y(t)$ is the experimental output signal and $\hat{y}(t)$ is the one obtained by simulation of the identified model under the experimental input signal $u(t)$.

3.3 Frequency-domain Identification

Several methods of identifying a fractional-order model from frequency response data (frequency, magnitude and phase) are described in [16], [17] and are summarized below.

The method proposed by Hartley and Lorenzo allows to obtain a fractional-order model in the form

$$G(s) = c_n s^{n\gamma} + c_{n-1} s^{(n-1)\gamma} + \cdots + c_1 s^{\gamma} + c_0 \tag{3.5}$$

or

$$G(s) = \frac{1}{c_n s^{n\gamma} + c_{n-1} s^{(n-1)\gamma} + \cdots + c_1 s^{\gamma} + c_0}, \tag{3.6}$$

where n is the order of the polynomial and γ is the commensurate order, both of which need to be chosen by the user. The coefficients for (3.5) c_0, c_1, $\ldots$, c_n are then found by solving the following equation:

$$\begin{bmatrix} G(j\omega_1) \\ G(j\omega_2) \\ \vdots \\ G(j\omega_m) \end{bmatrix} = \begin{bmatrix} 1 & (j\omega_1)^{\gamma} & (j\omega_1)^{2\gamma} & \cdots & (j\omega_1)^{n\gamma} \\ 1 & (j\omega_2)^{\gamma} & (j\omega_2)^{2\gamma} & \cdots & (j\omega_2)^{n\gamma} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (j\omega_m)^{\gamma} & (j\omega_m)^{\gamma} & \cdots & (j\omega_m)^{\gamma} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}, \tag{3.7}$$

where ω_1, ω_2, $\ldots$, ω_m are the sampling frequencies. For model in (3.6) the left-hand side of (3.7) will have the inverse of the frequency behavior. Clearly, the second case is more useful in practical situations.

Another frequency-domain identification approach is proposed by Duarte Valério and is based on an algorithm developed by Levy. We investigate two variants of this algorithm. Both variants allow to find a model in form

$$G(s) = \frac{b_m s^{m\gamma} + b_{m-1} s^{(m-1)\gamma} + \cdots + b_1 s^{\gamma} + b_0}{a_n s^{n\gamma} + a_{n-1} s^{(n-1)\gamma} + \cdots + a_1 s^{\gamma} + 1} \tag{3.8}$$

by minimizing the square norm given by

$$\begin{aligned} \varepsilon &= G(j\omega)\left[a_n (j\omega)^{n\gamma} + \cdots + a_1 (j\omega)^{\gamma} + 1\right] \\ &- \left[b_m (j\omega)^{m\gamma} + \cdots + b_1 (j\omega)^{\gamma} + b_0\right] \end{aligned} \tag{3.9}$$

at all frequencies. The commensurate order γ as well as highest orders n, m of the polynomials are supplied by the user.

The algorithm (as implemented in [8]) finds parameters for an experimental frequency response given by $G(j\omega) = R(\omega) + jI(\omega)$ by solving the equation

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} b_0 \\ \vdots \\ b_m \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} e \\ g \end{bmatrix}, \tag{3.10}$$

where

$$\begin{aligned} A_{k,l} &= \sum_{p=1}^{f} \left(-\Re\left[(j\omega_p)^{k\gamma}\right] \Re\left[(j\omega_p)^{l\gamma}\right] - \Im\left[(j\omega_p)^{k\gamma}\right] \Im\left[(j\omega_p)^{l\gamma}\right] \right), \\ k &= 0 \ldots m,\ l = 1 \ldots m, \end{aligned} \tag{3.11}$$

$$
\begin{aligned}
B_{k,l} &= \sum_{p=1}^{f}\Bigg(\Re\left[(j\omega_p)^{k\gamma}\right]\Re\left[(j\omega_p)^{l\gamma}\right]R_p \\
&+ \Im\left[(j\omega_p)^{k\gamma}\right]\Re\left[(j\omega_p)^{l\gamma}\right]I_p \\
&- \Re\left[(j\omega_p)^{k\gamma}\right]\Im\left[(j\omega_p)^{l\gamma}\right]I_p \\
&+ \Im\left[(j\omega_p)^{k\gamma}\right]\Im\left[(j\omega_p)^{l\gamma}\right]R_p\Bigg), \\
k &= 0\ldots m,\, l = 1\ldots n,
\end{aligned} \tag{3.12}
$$

$$
\begin{aligned}
C_{k,l} &= \sum_{p=1}^{f}\Bigg(\left\{\Im\left[(j\omega_p)^{k\gamma}\right]I_p - \Re\left[(j\omega_p)^{k\gamma}\right]R_p\right\}\Re\left[(j\omega_p)^{l\gamma}\right] \\
&+ \left\{-\Re\left[(j\omega_p)^{k\gamma}\right]I_p - \Im\left[(j\omega_p)^{k\gamma}\right]R_p\right\}\Im\left[(j\omega_p)^{l\gamma}\right]\Bigg), \\
k &= 1\ldots n,\, l = 0\ldots m,
\end{aligned} \tag{3.13}
$$

$$
\begin{aligned}
D_{k,l} &= \sum_{p=1}^{f}\Bigg[\left(R_p^2 + I_p^2\right)\Big\{\Re\left[(j\omega_p)^{k\gamma}\right]\Re\left[(j\omega_p)^{l\gamma}\right] \\
&+ \Im\left[(j\omega_p)^{k\gamma}\right]\Im\left[(j\omega_p)^{l\gamma}\right]\Big\}\Bigg], \\
k &= 1\ldots n,\, l = 1\ldots n,
\end{aligned} \tag{3.14}
$$

$$
\begin{aligned}
e_{k,1} &= \sum_{p=1}^{f}\left\{-\Re\left[(j\omega_p)^{k\gamma}\right]R_p - \Im\left[(j\omega_p)^{k\gamma}\right]I_p\right\}, \\
k &= 0\ldots m,
\end{aligned} \tag{3.15}
$$

$$
\begin{aligned}
g_{k,1} &= \sum_{p=1}^{f}\left\{-\Re\left[(j\omega_p)^{k\gamma}\right]\left(R_p^2 + I_p^2\right)\right\}, \\
k &= 1\ldots n.
\end{aligned} \tag{3.16}
$$

The second variant of the algorithm introduces weights to the square norm such that

$$\begin{aligned} \varepsilon' &= w \cdot G(j\omega)\left[a_n(j\omega)^{n\gamma} + \cdots + a_1(j\omega)^{\gamma} + 1\right] \\ &- \left[b_m(j\omega)^{m\gamma} + \cdots + b_1(j\omega)^{\gamma} + b_0\right], \end{aligned} \tag{3.17}$$

where weights w are frequency dependent and for frequencies ω_i, $i = 1 \ldots f$

$$w = \begin{cases} \frac{\omega_2 - \omega_1}{2\omega_1^2} & , i = 1 \\ \frac{\omega_{i+1} - \omega_{i-1}}{2\omega_i^2} & , 1 < i < f \\ \frac{\omega_f - \omega_{f-1}}{2\omega_f^2} & , i = f \end{cases} \tag{3.18}$$

Weights are intended to improve the quality of the approximation at low frequencies.

Additionally, an optimization problem may be stated for a set of parameters

$$\theta = \begin{bmatrix} \gamma & n & m \end{bmatrix}. \tag{3.19}$$

An objective function to minimize is given by a performance index in the following form

$$J = \frac{1}{n_\omega} \sum_{i=1}^{n_\omega} \left| G(j\omega) - \hat{G}(j\omega) \right|^2, \tag{3.20}$$

where n_ω is the number of frequencies in ω, G is the original plant, from which the response was obtained, and $\hat{G}$ is the identified model.

Chapter 4

Fractional-order Control

In this chapter we focus on three different controller types, all of which were are obtained by extending integer-order controllers by concepts of fractional calculus.

The chapter is organized as follows. In Section 4.1 a generalized version of a conventional PID controller is introduced, namely the $PI^{\lambda}D^{\mu}$ controller. A summary of the fractional integrator and differentiator control actions is presented. Tuning and optimization methods are also discussed. In Section 4.2 we take a look at the generalization of a lead-lag compensator type controller to the fractional case. Finally, in Section 4.3 a modified PID controller is introduced, called the TID (Tilt-Integral-Derivative) controller.

4.1 Fractional PID Controller

4.1.1 Introduction

The notion of a fractional PID controller was introduced by Podlubny in [18]. This generalized controller is called the $PI^{\lambda}D^{\mu}$ controller[1], and has an integrator with an order λ and a differentiator of order μ. In the same paper Podlubny demonstrated, that the fractional-order controller had a better response than an integer-order one when used in a control loop with a fractional-order plant. In more recent researches [19, 20] it has been confirmed, that the fractional controller outperforms the integer-order PID controller.

[1] Notation $PI^{\lambda}D^{\delta}$ is also used

The control action of the PI$^{\lambda}$D$^{\mu}$ controller can be expressed as follows:

$$u(t) = K_p e(t) + K_i \mathscr{D}^{-\lambda} e(t) + K_d \mathscr{D}^{\mu} e(t), \tag{4.1}$$

where $e(t)$ is the error signal. After applying the Laplace transform to (4.1) assuming zero initial conditions, the following equation is obtained:

$$G_c(s) = K_p + \frac{K_i}{s^{\lambda}} + K_d s^{\mu} \tag{4.2}$$

Obviously, when taking $\lambda = \mu = 1$ the result is the classical integer-order PID controller. With more freedom in tuning the controller, the four-point PID diagram can now be seen as a PID controller plane, which is conveyed in Figure 4.1.

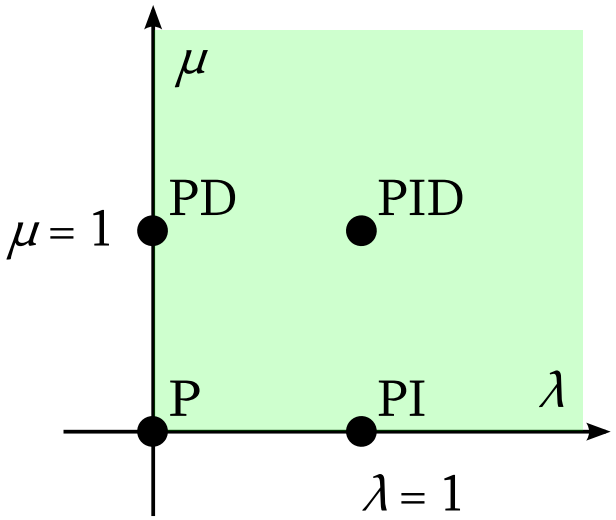


Figure 4.1: The PI$^{\lambda}$D$^{\mu}$ controller plane

4.1.2 Effects of Fractional Control Actions

Further we briefly summarize the effects of extending the integral and derivative control actions to the fractional case [2]. These control actions are of type

$$\begin{aligned} C(s) &= K\, s^{\gamma}, \\ \gamma &\in [-1;\, 1]\,. \end{aligned} \tag{4.3}$$

Let us first explore the fractional integrator with $\gamma \in [-1;\, 0]$. Under a square error

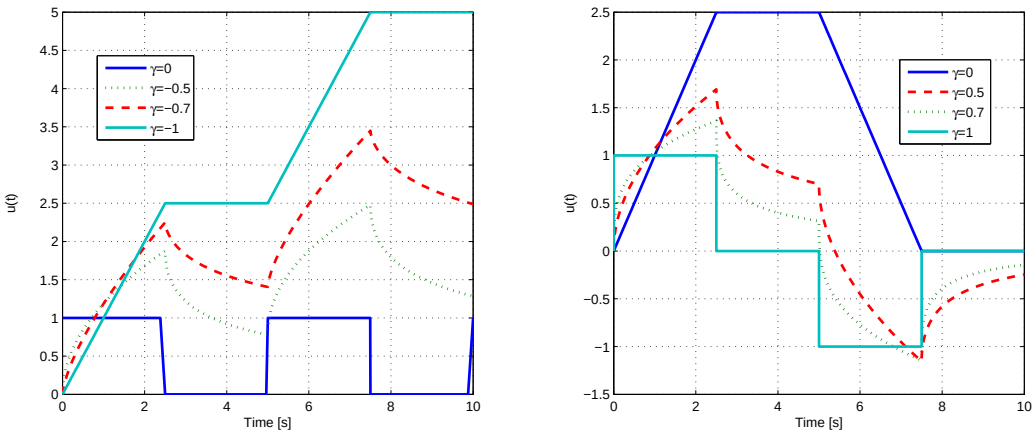
signal and with different orders γ and $K = 1$ the control action $u(t)$ will have the forms depicted in Figure 4.2a.

In the frequency domain by varying γ the following can be achieved:

- a constant increment in the slopes of the magnitude curve, that varies between $-20\,\mathrm{dB/dec}$ and $0\,\mathrm{dB/dec}$,
- a constant delay in the phase plot that varies between $-\pi/2\,\mathrm{rad}$ and $0\,\mathrm{rad}$.

Next, the fractional differentiator is investigated with $\gamma \in [0;\, 1]$. Its control action under a trapezoidal signal is shown in Figure 4.2b. And in the frequency domain by varying γ the following is possible:

- a constant increment in the slopes of the magnitude that varies between $0\,\mathrm{dB/dec}$ and $20\,\mathrm{dB/dec}$,
- a constant delay in the phase plot that varies between $0\,\mathrm{rad}$ and $\pi/2\,\mathrm{rad}$.



(a) Fractional integrator $s^{-\gamma}$ control actions

(b) Fractional differentiator s^{γ} control actions

Figure 4.2: Fractional integrator and derivative control actions in the time domain

Consider a comparison between a classical PID and a fractional one in the frequency domain given in Figure 4.3 by means of a Bode diagram.

As it can be seen, introducing fractional orders for the integrator and differentiator components of the PID controller has potential benefits due to providing more degrees of freedom. It is expected, that fractional PID controllers will replace their classical, integer-order variants in industrial control applications [3]. Tuning and auto-tuning techniques play a significant role in accelerating this process.

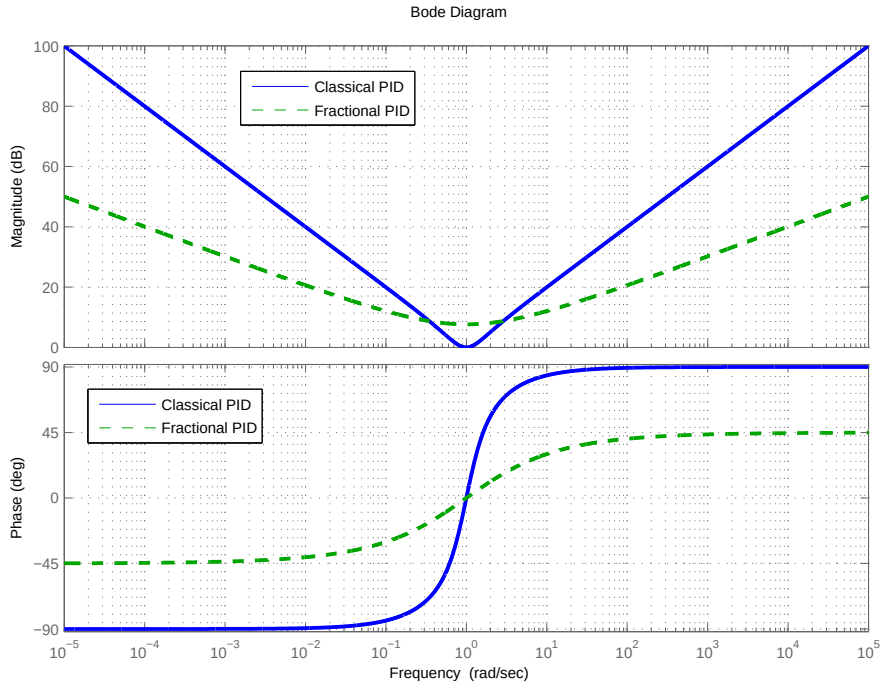


Figure 4.3: Bode diagram of a frequency response of a classical PID controller with $K_p = K_i = K_d = 1$ and a fractional PID controller with $K_p = K_i = K_d = 1$, $\lambda = \mu = 0.5$.

4.1.3 Tuning and Optimization

Several methods of tuning fractional PID controllers were proposed over the years [2, 21, 23, 24]. However, there is still no well-established, general tuning algorithm. Nevertheless, optimization techniques may be applied to the tuning problem instead.

There are several aspects to the problem of designing a fractional PID controller via optimization:

- the type of plant to be controlled (fractional or integer-order),
- fractional PID design specifications,
- optimization criterion,
- parameters to optimize,
- obtaining initial parameters to optimize.

A general approach is desired for optimizing a PID control loop regardless of the plant type. However, there is a vast array of well-established tuning techniques for common model types (e.g. FOPDT[2]). So if a fractional model can be approximated by a simple

[2]First order plus dead time

model to a certain degree of validity it may be used to obtain initial, integer-order PID parameters. These parameters can then be further optimized to achieve better performance.

Most frequently used design specifications can be derived from the frequency domain evaluation of the open loop $C(j\omega)G(j\omega)$, where $C(j\omega)$ is the controller and $G(j\omega)$ is the plant. These specifications include:

- gain margin G_m, obtained as a difference between unity gain and gain at the frequency where the phase angle is $-180°$,
- phase margin φ_m, obtained as a difference between $-180°$ and phase angle at the gain crossover frequency.

In terms of noise and disturbance rejection, the following measures can be used.

- Complementary sensitivity function $T(j\omega)$:

$$T(j\omega) = \frac{C(j\omega)G(j\omega)}{1 + C(j\omega)G(j\omega)}. \tag{4.4}$$

- Sensitivity function $S(j\omega)$:

$$S(j\omega) = \frac{1}{1 + C(j\omega)G(j\omega)}. \tag{4.5}$$

The parameter set to optimize consists of fractional PID parameters:

$$\theta = \begin{bmatrix} K_p & K_i & K_d & \lambda & \mu \end{bmatrix}. \tag{4.6}$$

However, it may be beneficial to optimize all parameters independently, similarly to a method discussed in Section 3.2 of Chapter 3.

Finally, the following performance criteria can be used as objective functions for optimization:

- integral square error $ISE = \int_0^t e^2(t)\ \mathrm{d}t$,
- integral absolute error $IAE = \int_0^t \left|e(t)\right|\ \mathrm{d}t$,
- integral time-square error $ITSE = \int_0^t te(t)^2\ \mathrm{d}t$,

- integral time-absolute error $ITAE = \int_0^t t \left|e(t)\right| \, \mathrm{d}t$,

where $e(t) = 1 - y(t)$, $y(t)$ is the tuned fractional control system closed-loop step response.

4.2 Fractional Lead-Lag Compensator

Lead-lag compensators are a well-known type of feedback controller widely used in practice. Extending it with ideas from fractional calculus can lead to a more robust controller.

A fractional-order lead-lag compensator has the following transfer function:

$$G_c(s) = K_c \left(\frac{s + \frac{1}{\lambda}}{s + \frac{1}{x\lambda}} \right)^{\alpha} = K_c x^{\alpha} \left(\frac{\lambda s + 1}{x\lambda s + 1} \right)^{\alpha}, \; 0 < x < 1, \tag{4.7}$$

where α is the fractional order of the controller, $\frac{1}{\lambda} = \omega_z$ is the zero frequency and $\frac{1}{x\lambda} = \omega_p$ is the pole frequency when $\alpha > 0$.

When $\alpha > 0$ the controller (4.7) corresponds to a fractional-order lead compensator and when $a < 0$ it corresponds to a fractional lag compensator.

Taking $k' = K_c x^{\alpha}$ another used form of the fractional-order lead-lag compensator is obtained:

$$G_c(s) = k' \left(\frac{\lambda s + 1}{x\lambda s + 1} \right)^{\alpha}. \tag{4.8}$$

The contribution of parameter α is such, that the lower its value, the longer the distance between the zero and pole and vice versa so that the contribution of phase at a certain frequency stands still. This makes the controller more flexible and allows a more robust approach to the design. Tuning and auto-tuning techniques are discussed in [2].

It can be seen, that the form (4.8) is an implicit form of a fractional-order transfer function. Thus it cannot be directly realized by approximation methods discussed in Chapter 2. However, there is a workaround. Since the frequency response can be directly obtained, a frequency identification method can be employed to implement the fractional-order lead-lag compensator such as described in Section 3.3 of Chapter 3.

4.3 TID Controller

The TID (Tilt-Integral-Derivative) controller was first proposed in [25] and summarized in [21]. It can be described by the following transfer function (the structure of the TID controller is also depicted in Figure 4.4).

$$G_c(s) = \frac{K_t}{s^{\frac{1}{n}}} + \frac{K_i}{s} + K_d s, \tag{4.9}$$

where $K_t/s^{\frac{1}{n}}$ is the *Tilt* type compensator and $n \in \mathbb{R}$, $n > 0$, preferably $n \in [2; 3]$. It can be seen, that the TID controller corresponds to a conventional PID controller with proportional gain replaced by the compensator component.

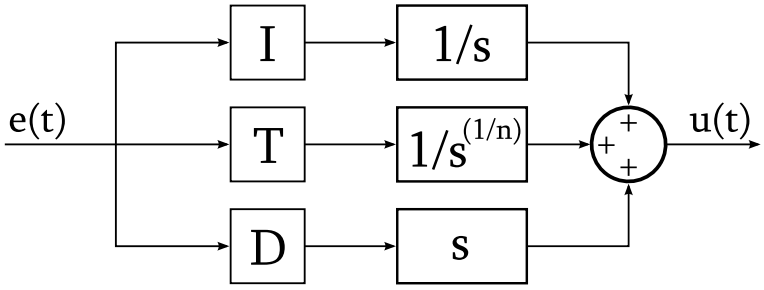


Figure 4.4: TID controller structure

The motivation for this type of controller is from the consideration of Bode's theoretically optimal loop response (see Figure 4.5). The goal of the feedback control system is to minimize the effect of disturbances at the output of the system and to minimize sensitivity of the closed-loop response to parameter variations of the controlled plant. To satisfy these requirements, the feedback of the system must be maximized while being properly weighted in frequency. This is partially achieved by introducing the compensator component.

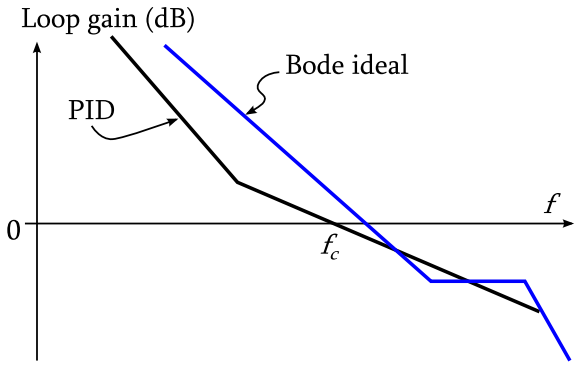


Figure 4.5: Bode plots for PID controlled plant and the ideal loop response

A comparison of a conventional PID controller frequency response and a TID controller frequency response is depicted in Figure 4.6.

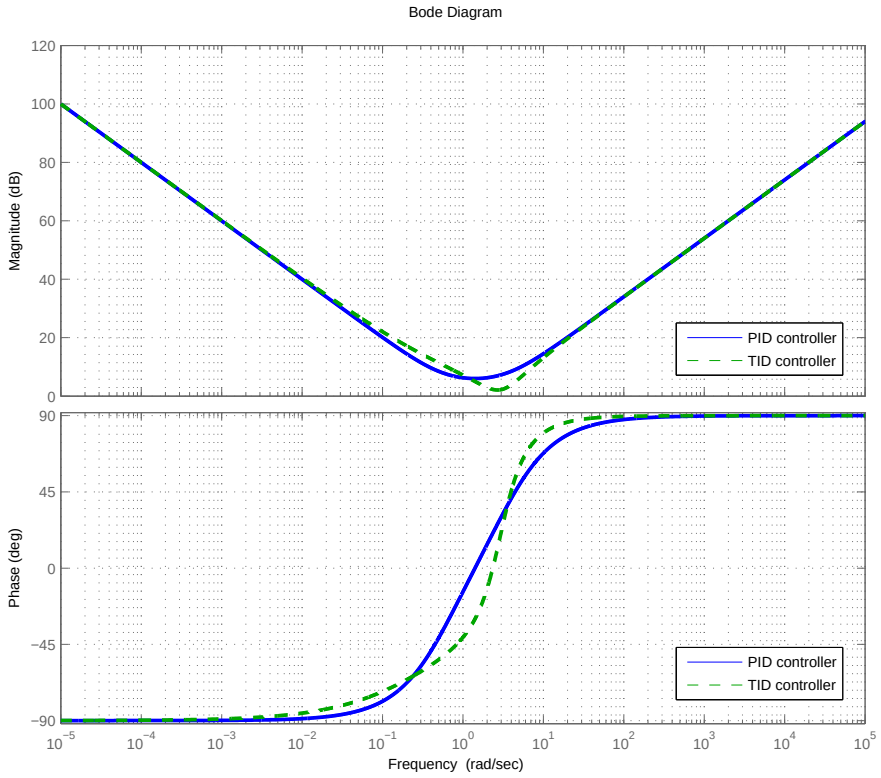


Figure 4.6: A comparison of the frequency response of a PID controller with $K_p = 2$, $K_i = 1$, $K_d = 0.5$ and a TID controller with $K_t = 2$, $n = 3$, $K_i = 1$, $K_d = 0.5$.

Chapter 5

FOMCON Toolbox for MATLAB

This chapter introduces the reader to the FOMCON toolbox for MATLAB in which all the major features discussed in previous chapters are implemented. The chapter is organized as follows. In Section 5.1 the reader is presented with an overview of the FOMCON toolbox. Its relationship with other existing toolboxes and motivation for its development are also given. In Section 5.2 an overview of the toolbox features is presented. Finally, in Section 5.3 notations used in the user manual in Chapter 6 are provided.

5.1 Introduction

FOMCON stands for "Fractional-Order Modeling and Control". It is a MATLAB toolbox and is built upon an existing mini toolbox called *FOTF* (Fractional-Order Transfer Functions), the source code for which is provided in literature [2, 3, 5].

The toolbox was developed and tested in MATLAB v. 7.7. However, most of the features are backwards-compatible and should work in earlier releases of MATLAB (v. 7.4-7.6).

The basic motivation for developing of FOMCON was the desire to obtain a set of useful and convenient tools to facilitate the research of fractional-order systems. This involved writing convenience functions (e.g. the polynomial string parser) and building graphical user interfaces (GUI's) to improve the general workflow. However, a full suite of tools was also desired due to certain limitations in existing toolboxes. Thus

the basic functionality of the toolbox was extended with advanced features (such as fractional-order system identification and $PI^{\lambda}D^{\mu}$ design).

As previously mentioned, there exist other MATLAB toolboxes dedicated to fractional-order control, such as CRONE and NINTEGER. These mostly focus on novel control strategies (such as the CRONE control), while FOMCON aims at extending classical control schemes with concepts of fractional calculus. FOMCON is linked to the other toolboxes and uses some of their functionality. A diagram showing toolbox relations to FOMCON is given in Figure 5.1.

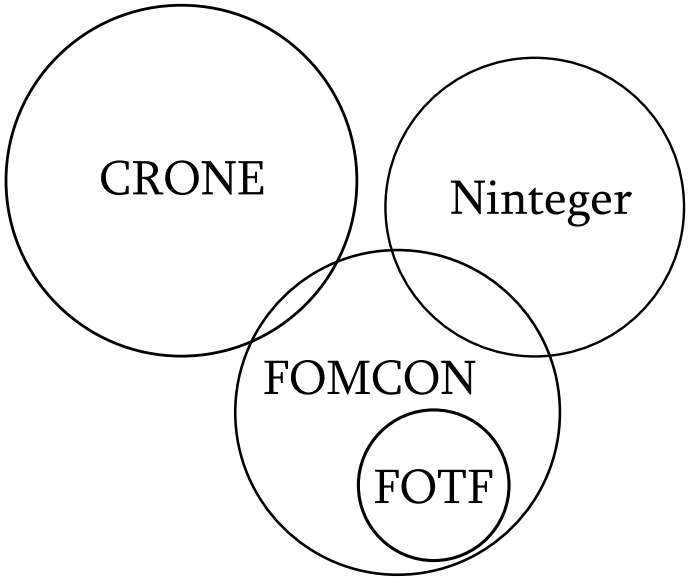


Figure 5.1: MATLAB fractional-order calculus oriented toolbox relations to FOMCON

With all previous considerations, the motivations for developing FOMCON can now be established.

- It is a product suitable for both beginners and more demanding users due to availability of graphical user interfaces and advanced functionality.
- It focuses on extending classical control schemes with concepts of fractional-order calculus.
- It can be viewed as a "missing link" between CRONE and NINTEGER.
- With the Simulink blockset the toolbox aims at a more sophisticated modeling approach.
- Due to availability of source code the toolbox can be ported to other platforms, such as *Scilab* or *Octave* (some limitations and/or restrictions may apply).

Further, an overview of the toolbox and its features is presented.

5.2 Overview

In FOMCON the main object of analysis is the fractional-order transfer function given by (2.15). The toolbox currently focuses on the SISO (single input-single output), LTI (linear time-invariant) systems.

The FOMCON toolbox consists of the following modules:

- Main module (fractional system analysis).
- Identification module (system identification in the time and frequency domains).
- Control module (fractional PID design, tuning and optimization tools as well as some additional features).

All the modules are interconnected and can be accessed from the graphical user interface as depicted in Figure 5.2.

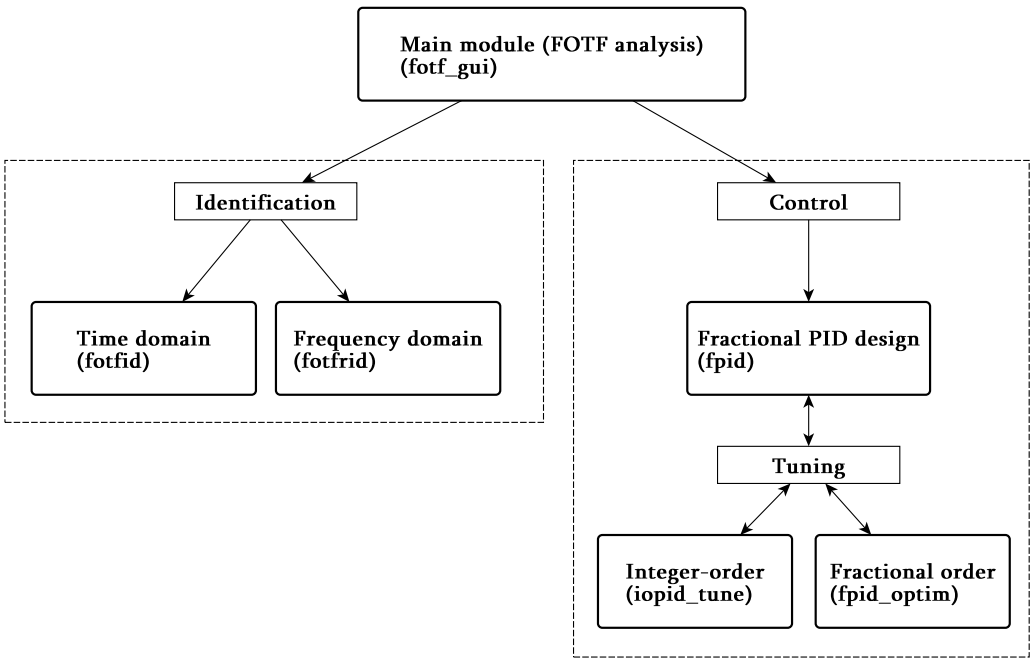


Figure 5.2: FOMCON module relations (name of corresponding function to open the GUI is given in brackets)

A Simulink blockset is also provided in the toolbox allowing more complex modeling tasks to be carried out. General (modular) approach to block construction was used where applicable. The following blocks are currently realized:

- General fractional operator.
- Fractional integrator and differentiator.
- Fractional transfer function (with discrete realization).
- PI$^{\lambda}$D$^{\mu}$ controller (with discrete realization).
- TID controller.

The FOMCON toolbox relies on the following MATLAB products:

- Control system toolbox, required for most features.
- Optimization toolbox, required for time domain identification and integer-order PID tuning for common process model approximation.

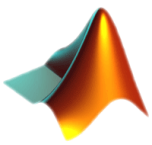
Several other tools are used directly (without any change). These tools are included in the toolbox as per the GPL license:

- `optimize()` function.
- Several NINTEGER toolbox frequency domain identification functions.

It is also possible to export fractional-order systems to the CRONE toolbox format (this feature requires the object-oriented CRONE toolbox to be installed).

5.3 Used Notations

Further follows a list of notations used throughout the user manual in Chapter 6.



The MATLAB logo designates a paragraph with additional information about the implementation of certain features in MATLAB and/or SIMULINK.

A paragraph decorated with an "*i*" sign contains additional information about features and their possible usages.

The warning sign indicates important information about certain features, such as significant tips, potential problems and software limitations.

Chapter 6

FOMCON User Manual

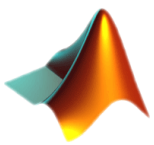
This chapter has the following structure. Sections 6.1 through 6.4 are organized as a user manual with illustrative examples in the end of every section.

6.1 System Analysis Module

The system analysis module is the main module of the FOMCON toolbox. It serves as a foundation for fractional-order control systems engineering and hence for all other modules.

6.1.1 The FOTF Object

The basic object used to analyze fractional-order systems is a fractional-order transfer function (FOTF) in the form (2.15). Analysis is carried out using overloaded Control System toolbox functions.



In its 7.x version tree MATLAB has enhanced support for object-oriented programming. It is thus beneficial to implement the fractional-order transfer function as a class (`fotf`) and write functions overloading those of Control System toolbox for this particular object type. These functions need to be placed in a folder beginning with the "@" symbol (`@fotf` in this case). This is the very idea behind the *FOTF* toolbox which served as a base for FOMCON.

Further a list of functions dealing directly with the `fotf` object is given.

6.1.1.1 Function `fotf()`

SYNTAX

```
G = fotf(a, na, b, nb, delta)
G = fotf(b)
G = fotf(G1)
G = fotf('s')
```

INPUT ARGUMENTS

- a, na, b, nb — fractional-order transfer function real coefficient and exponent vectors such that

$$G(s) = \frac{b_k s^{nb_k} + b_{k-1} s^{nb_{k-1}} + \cdots + b_0 s^{nb_0}}{a_l s^{na_l} + a_{l-1} s^{na_{l-1}} + \cdots + a_0 s^{na_0}},$$

- δ (optional) — input-output delay (positive real number) in seconds such that

$$G(s) = \frac{Z(s)}{P(s)} \cdot \mathrm{e}^{-\delta \cdot s},$$

- G_1 — an object of type `fotf` or `tf`,
- $'s'$ — a string containing the "s" character.

OUTPUT ARGUMENTS

- G — the resulting `fotf` object, a LTI system given by the fractional-order transfer function.

DESCRIPTION

This function is the initializing method for the `fotf` object. The object parameters can later can be set or retrieved via dot notation. An overload function `display()` is used to display the object in MATLAB.

If the "s" character is supplied as a single argument, a `fotf(1,0,1,1)` object is returned. This could be used as a basic building block for more complex systems.

REMARKS

It could be more convenient to input models into MATLAB using either the built-in parser with overloaded operation methods, the slightly more advanced string parser or by means of the graphical user interface.

EXAMPLES

In order to input a fractional-order transfer function $G(s) = \frac{s^{0.5}+1}{2s^{1.7}+s^{0.3}-5} \cdot \mathrm{e}^{-5s}$ into MATLAB the following statement can be used:

```
G = fotf([2 1 -5], [1.7 0.3 0], [1 1], [0.5 0], 5)
```

The object is then displayed in MATLAB as

```
Fractional-order transfer function:
    s^{0.5}+1
------------------ exp(-5s)
2s^{1.7}+s^{0.3}-5
```

To change the zero polynomial to $7.5s^{0.5} + 2$, the following command could be used:

```
G.b = [7.5 2]
```

and the object is displayed as

```
Fractional-order transfer function:
   7.5s^{0.5}+2
------------------ exp(-5s)
2s^{1.7}+s^{0.3}-5
```

6.1.1.2 Function `newfotf()`

SYNTAX

```
G = newfotf(zero_s, pole_s, delta)
G = newfotf(zero_s, [a na], delta)
G = newfotf([b nb], pole_s, delta)
G = newfotf([b nb], [a na], delta)
```

INPUT ARGUMENTS

- $zero_s$ — fractional zero polynomial string,
- $pole_s$ — fractional pole polynomial string,
- δ (optional) — input-output delay in seconds,
- $[\ a \quad na\ ]$ — pole polynomial coefficient and exponent vector,
- $[\ b \quad nb\ ]$ — zero polynomial coefficient and exponent vector.

OUTPUT ARGUMENTS

- G — the resulting `fotf` object.

DESCRIPTION

This function implements a simple string parser to obtain the desired fractional-order transfer function. It may be more convenient to use it instead of the `fotf()` method.

EXAMPLE

In order to supply the previously discussed transfer function $G(s) = \frac{s^{0.5}+1}{2s^{1.7}+s^{0.3}-5} \cdot \mathrm{e}^{-5s}$ into MATLAB the following command could be used

```
G=newfotf('s^0.5+1', '2s^1.7+s^0.3-5', 5);
```

and the same result as above is obtained.

6.1.1.3 Block Interconnection Functions

The functions that realize `fotf` block interconnections are grouped together in Table 6.1 with syntax and corresponding comments.

Functions `plus()` and `feedback()` do not support fractional-order transfer functions with different time delays.

Table 6.1: `fotf` block interconnection functions

FUNCTION	DESCRIPTION	SYNTAX
`feedback()`	System negative feedback	`feedback(G1, G2)`
`inv()`	System inverse $\frac{1}{G}$	`inv(G)`
`minus()`	System subtraction (parallel connection)	`G1 - G2`
`mpower()`	Power of given system ($n \in \mathbb{Z}$)	`G1^n`
`mrdivide()`	System division (series connection)	`G1 / G2`
`mtimes()`	System multiplication (series connection)	`G1 * G2`
`plus()`	System addition (parallel connection)	`G1 + G2`
`uminus()`	Unary minus	`-G`

EXAMPLE

Let us assume that a fractional system is given by a block diagram in Figure 6.1, where:

$$\begin{aligned} G_1(s) &= \frac{1}{s^{0.5}+1}, G_2(s) = \frac{s^{0.3}+1}{s^{2.5}+s+1}, \\ G_3(s) &= \frac{2}{s^{0.1}+1}, G_4(s) = \frac{1}{15s+1}. \end{aligned}$$

To obtain a full model $G(s)$ the following MATLAB commands need to be used:

```
G1 = fotf([1 1], [0.5 0], 1, 0);
G2 = fotf([1 1 1], [2.5 1 0], [1 1], [0.3 0]);
G3 = fotf([1 1], [0.1 0], 2, 0);
G4 = fotf([15 1], [1 0], 1, 0);
G = feedback(G1*(G2-G3), G4);
```

The system is then obtained as

$$G(s) = \frac{-30s^{3.5} - 2s^{2.5} - 30s^2 + 15s^{1.4} + 15s^{1.3} + 15s^{1.1} - 17s + s^{0.4} + s^{0.3} + s^{0.1} - 1}{\begin{array}{c} 15s^{4.1} + 15s^4 + 15s^{3.6} + 15s^{3.5} + s^{3.1} + s^3 + 16s^{2.6} + 14s^{2.5} + 15s^{2.1} \\ +15s^2 + 16s^{1.6} + 16s^{1.5} + 16s^{1.1} + 14s + s^{0.6} + s^{0.5} + s^{0.4} + s^{0.3} + 2s^{0.1} \end{array}}$$

It can be seen from this example, that from relatively simple initial systems a fairly complicated fractional-order transfer function was obtained.

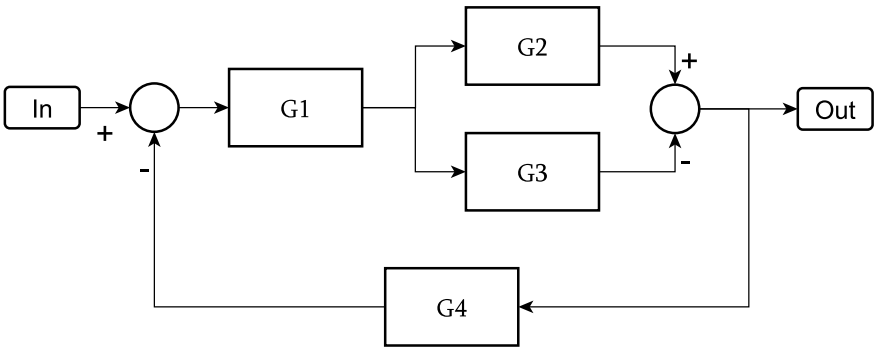


Figure 6.1: Example system interconnection

6.1.1.4 Trimming Functions

SYNTAX

```
G1 = trunc(G, p_ex, p_cf)
G1 = round(G, p_ex, p_cf)
```

INPUT ARGUMENTS

- G — a `fotf` object,
- p_{ex} — exponent trimming precision,
- p_{cf} (optional) — coefficient trimming precision.

OUTPUT ARGUMENTS

- G_1 — trimmed `fotf` object.

DESCRIPTION

The trimming functions are used to either truncate or round the transfer function exponents and coefficients to the given precision. This may be desired when analyzing commensurate-order systems where the commensurate-order is too low and as a consequence the rational-order pseudo transfer function order is too high.

EXAMPLES

Consider a system given by $G(s) = \frac{1}{15s^{1.3442}+5s^{0.44987}-15}$. Suppose the desired precision for the differentiation orders is $p_{ex} = 0.01$. Then truncating this system via

```
trunc(G, 1e-2)
```

will result in

```
Fractional-order transfer function:
           1
-----------------------
15s^{1.34}+5s^{0.44}-15
```

while rounding using

```
round(G, 1e-2)
```

will return

```
Fractional-order transfer function:
           1
-----------------------
15s^{1.34}+5s^{0.45}-15
```

6.1.2 Stability Analysis

6.1.2.1 Function `isstable()`

SYNTAX

```
[K, q, err, apol] = isstable(G, doPlot)
```

INPUT ARGUMENTS

- G — the `fotf` system for which the stability condition is evaluated,
- $doPlot$ (optional) — boolean value, if set to *true* — the stability diagram is plotted.

OUTPUT ARGUMENTS

- K — boolean value, if *true* then the system given by G is stable,
- q — transfer function pole polynomial commensurate order,
- err — error norm,
- $apol$ — minimum stability condition criterion.

DESCRIPTION

This function realizes the algorithm described in Subsection 2.4.1. The system is deemed stable if

$$apol > q\frac{\pi}{2},$$

where $apol = \min(\left|\arg(p)\right|)$ and p is a vector containing the roots of the pseudo-rational polynomial. To keep the order of the polynomial from becoming too high, the minimum allowed commensurate-order is $q = 0.01$. If the system commensurate-order is smaller, the orders are automatically truncated.

If $doPlot$ is set to *true*, a diagram is plotted and is populated with poles of the fractional-order system G. The shaded area represents the unstable region. If any poles are inside it, the system is not stable.

6.1.3 Time Domain Analysis

6.1.3.1 Function `lsim()`

SYNTAX

```
y = lsim(G, u, t)
```

INPUT ARGUMENTS

- G — `fotf` system to be simulated,
- u — input signal $u(t)$ vector,
- t — time vector, consisting of regularly spaced time samples such that $t = [0; t_{final}]$ with constant step $\mathrm{d}\,t$.

OUTPUT ARGUMENTS

- y — row vector containing system response $y(t)$ under input signal $u(t)$ at times given in t.

DESCRIPTION

This function uses the Grünwald-Letnikov definition based computing algorithm described in Subsection 2.4.2.

Since this function uses recursive methods to compute the time response, it may take some time for simulating a large number of points. For such a case it may be desired to show a progress bar. In order to show it a command `show_pbar on` can be typed in MATLAB command window.

Since fixed-step computation is used, it may be required to increase the number of simulation points, i.e. take a smaller time step to achieve the desired level of accuracy.

6.1.3.2 Function `step()`

SYNTAX

```
y = step(G, t)
```

INPUT ARGUMENTS

- G — `fotf` system to be simulated,
- t (optional) — vector with time samples.

OUTPUT ARGUMENTS

- y — row vector containing system response $y(t)$ under input signal $u(t) = 1$ at times given in t.

DESCRIPTION

Computes the step response of a system given by G using `lsim()`. If input argument t is omitted, a simple auto-ranging procedure is used.

6.1.3.3 Function `impulse()`

SYNTAX

```
y = impulse(G, t)
```

INPUT ARGUMENTS

- G — `fotf` system to be simulated,
- t (optional) — time vector.

OUTPUT ARGUMENTS

- y — row vector containing system response $y(t)$ under an impulse signal at times given in t.

DESCRIPTION

Computes the impulse response of a system given by G using `lsim()`. If input argument t is omitted, a default range of $t = [0;\ 100]$ is used with step of 0.1.

6.1.4 Frequency Domain Analysis

6.1.4.1 Function `freqresp()`

SYNTAX

```
r = freqresp(G, w)
```

INPUT ARGUMENTS

- G — `fotf` system for which to obtain the complex frequency response,
- ω — frequency vector in rad/s at which to evaluate $G(j\omega)$.

OUTPUT ARGUMENTS

- r — column vector containing complex frequency response of system $G(j\omega)$ at frequencies given in ω.

DESCRIPTION

Computes the complex frequency response of system G as described in Subsection 2.4.3. Frequency vector ω can be obtained by using the MATLAB `logspace()` function.

6.1.4.2 Function `bode()`

SYNTAX

```
[mag, ph] = bode(G, w)
[mag, ph, w] = bode(G)
H = bode(G)
H = bode(G, w)
bode(G)
bode(G, w)
```

INPUT ARGUMENTS

- G — `fotf` system for which to obtain the Bode plot parameters,

- ω — frequency vector in rad/s at which to evaluate $G(j\omega)$.

OUTPUT ARGUMENTS

- mag — response magnitude at ω,
- ph — response phase in degrees at ω,
- H — MATLAB `frd` object.

DESCRIPTION

Computes the complex response of system G first using `freqresp()` and internally converts the resulting response to a `frd` object, which is then used to obtain the required parameters by passing it to the `bode()` function of System Control toolbox.

If no output argument is specified, plots the Bode diagram.

6.1.4.3 Function `nyquist()`

SYNTAX

```
nyquist(G, w)
```

INPUT ARGUMENTS

- G — `fotf` system for which to draw the Nyquist diagram,
- ω — frequency vector in rad/s at which to evaluate $G(j\omega)$.

DESCRIPTION

Plots the Nyquist diagram for system G. Uses `bode()` to obtain the `frd` object and then passes it to the Control System toolbox function `nyquist()` to draw the plot.

6.1.4.4 Function `nichols()`

SYNTAX

```
nichols(G, w)
```

INPUT ARGUMENTS

- G — `fotf` system for which to draw the Nichols diagram,
- ω — frequency vector in rad/s at which to evaluate $G(j\omega)$.

DESCRIPTION

Plots the Nichols diagram for system G. Similarly to `nyquist()`, uses `bode()` to obtain the `frd` object and then passes it to the Control System toolbox function `nichols()` to draw the plot.

6.1.4.5 Function `margin()`

SYNTAX

```
[Gm, Pm, w_cg, w_cp] = margin(G)
[Gm, Pm, w_cg, w_cp] = margin(mag, phase, w)
```

INPUT ARGUMENTS

- G — `fotf` system for which to obtain the stability margins,
- mag — frequency response magnitude,
- ph — frequency response phase (in degrees),
- ω — frequency vector in rad/s.

OUTPUT ARGUMENTS

- Gm — gain margin,
- Pm — phase margin,
- ω_{cg} — gain crossover frequency in rad/s,
- ω_{cp} — phase crossover frequency in rad/s.

DESCRIPTION

This function computes the open-loop system G stability margins and associated frequencies. It uses Control System toolbox `margin()` function with the obtained (or given) frequency response.

6.1.5 Fractional-order System Approximation

6.1.5.1 Function `oustapp()`

SYNTAX

```
Z = oustapp(G, w_b, w_h, N, method)
```

INPUT ARGUMENTS

- G — fractional-order transfer function (`fotf` object) to approximate,
- ω_b (optional) — lower frequency bound in rad/s, default is $\omega_b = 0.001$,
- ω_h (optional) — higher frequency bound in rad/s, default is $\omega_h = 1000$,
- N (optional) — order of approximation, default is $N = 5$,
- $method$ (optional) — method of approximation, can be $'oust'$ for the Oustaloup filter or $'ref'$ for the refined Oustaloup filter, default is $'oust'$.

OUTPUT ARGUMENTS

- Z — MATLAB `zpk` object containing the approximated continuous-time integer-order filter.

DESCRIPTION

Computes an integer-order continuous Oustaloup filter approximation of the fractional-order transfer function given by G as per the algorithms in Section 2.5. The approximation is only valid in the specified frequency range $\omega = [\omega_b; \omega_h]$.

EXAMPLE

Consider a fractional-order operator $s^{0.5}$. To approximate it by a refined Oustaloup filter in a frequency range $\omega = [0.01,\ 100]$ with order $N = 2$ the following command can be entered:

```
z = oustapp(fotf('s')^0.5, 1e-2, 1e2, 2, 'ref')
```

The following filter is obtained:

$$G(s) = \begin{array}{c} 18.9737\, s\,(s+111.1)(s+25.12) \\ \times(s+3.981)(s+0.631)(s+0.1)(s+0.01585) \\ \hline (s+222.2)(s+63.1)(s+10)(s+1.585) \\ \times(s+0.2512)(s+0.03981)(s+0.0045) \end{array} .$$

6.1.5.2 Object `fsparam()`

SYNTAX

```
p = fsparam(G, method, w, N)
```

INPUT ARGUMENTS

- G — the fotf plant to be included in the simulation structure,
- $method$ (optional) — approximation method, can be either $'oust'$ (default) or $'ref'$,
- ω (optional) — approximation frequency range in rad/s, default is $\omega = [0.001;\ 1000]$,
- N (optional) — approximation order, default is $N = 5$.

OUTPUT ARGUMENTS

- p — fractional-order simulation parameter structure.

DESCRIPTION

This is a utility object used to set fractional-order transfer function Oustaloup filter approximation parameters for simulation and is used by other toolbox functions.

METHODS

Table 6.2: `fsparam` object methods

METHOD	DESCRIPTION	SYNTAX
`oustapp()`	Get Oustaloup filter approximation of fractional-order plant as per `fsparam` object properties	`Z = oustapp(p)`

6.1.6 Object Conversion Functions

Currently there is two kinds of object conversion functions present in FOMCON toolbox :

- Conversion of `fotf` to FOMCON fractional-order state space system object,
- Conversion of `fotf` to the object-oriented CRONE toolbox objects.

These functions are summarized in Table 6.3.

Table 6.3: `fotf` conversion functions

FUNCTION	DESCRIPTION	SYNTAX
`tf2ss()`	Convert to fractional-order state-space object `foss`	`sys1 = tf2ss(G)`
`tf2ss_c()`	Convert to fractional-order state-space object `frac_ss` (CRONE)	`sys1 = tf2ss_c(G)`
`tf2tf_c()`	Convert to fractional-order transfer function `frac_tf` (CRONE)	`G1 = tf2tf_c(G)`

REMARKS

The functions `tf2ss_c()` and `tf2tf_c()` require the object-oriented CRONE toolbox to be installed.

EXAMPLES

Consider a system given by $G(s) = \frac{s^{0.5}+1}{2s^{2.5}+5s^{0.5}+3.5}$. To convert it to a fractional-order state-space object the following commands can be used:

```
G  = newfotf('s^{0.5}+1','2s^{2.5}+5s^{0.5}+3.5');
s1 = tf2ss(G);
```

A `foss` object is then returned with the following properties:

$$
\begin{aligned}
A &= \begin{bmatrix} 0 & 0 & 0 & -2.5 & -1.75 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, B = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \\
C &= \begin{bmatrix} 0 & 0 & 0 & 0.5 & 0.5 \end{bmatrix}, D = 0, q = 0.5,
\end{aligned}
$$

Where $A,\ B,\ C,\ D$ are fractional-order state-space matrices and $q = \gamma$ is the commensurate-order in (2.18).

To convert this system into a CRONE fractional-order transfer function the following can be entered in MATLAB command-line:

```
G1 = tf2tf_c(G)
```

MATLAB then returns the `frac_tf` object:

```
 Frac_tf transfer function :
        ( s^0.5 + 1 )
-----------------------
( 2 s^2.5 + 5 s^0.5 + 3.5 )
```

6.1.7 Graphical User Interface

6.1.7.1 Function `fotf_gui()`

SYNTAX

```
fotf_gui
```

DESCRIPTION

The graphical user interface for the main module is called *FOTF Viewer*. It is shown in Figure 6.2. The user interface is divided into two panels.

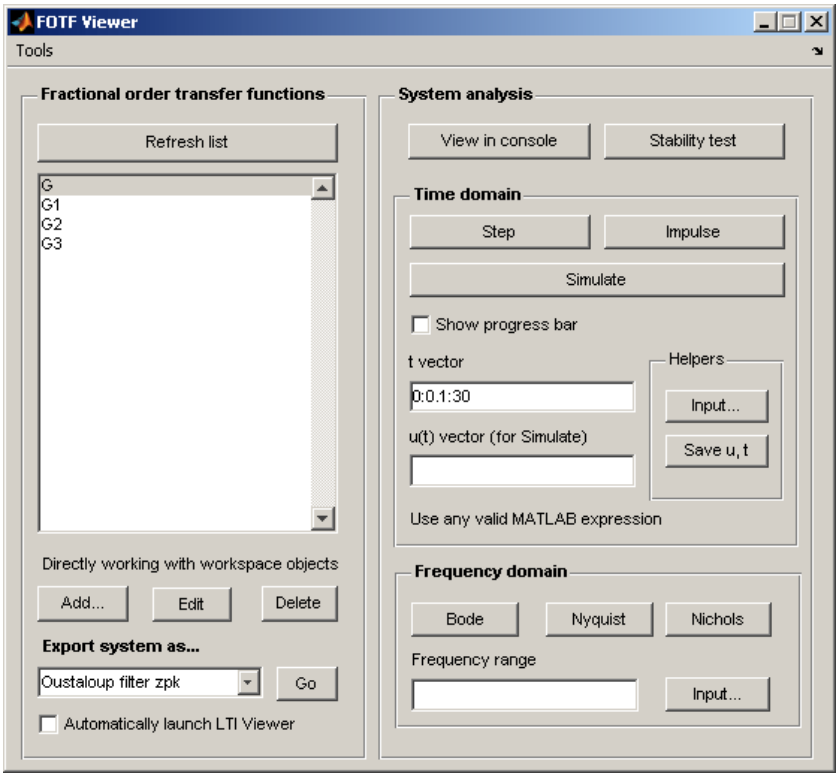


Figure 6.2: *FOTF Viewer* graphical user interface

The left panel, entitled *Fractional-order transfer functions*, contains a list of `fotf` objects that currently exist in the base workspace (the list can be refreshed by clicking the *Refresh* button). It also has means for adding new objects, editing the existing ones and deleting unneeded objects. The export feature has five options for exporting `fotf` objects:

- an Oustaloup filter,
- a refined Oustaloup filter,
- a fractional state-space (`foss` object),
- a fractional transfer function (`frac_tf` CRONE object),
- a fractional state-space (`frac_ss` CRONE object).

For the first two options it is possible to automatically run the Control System toolbox tool *LTI Viewer* right after the export is complete. In order to do this, one should tick the *Automatically launch LTI Viewer* checkbox.

The *System analysis* panel on the right comprises the following features: it allows checking fractional-order system stability, simulating it in the time domain and analyzing the frequency response in the frequency domain. Some helpful dialogs are also included for convenience.

The *Tools* menu is used to access other graphical user interfaces:

- time-domain identification tool (`fotfid`),
- frequency-domain identification tool (`fotfrid`),
- fractional PID design tool (`fpid`).

The currently selected system in the `fotf` list will be used for the PI$^{\lambda}$D$^{\mu}$ controller design.

6.1.8 System Analysis Examples

Example 6.1.1 Consider a dynamic system, defined by the following fractional-order differential equation:

$$2\mathscr{D}^{3.501}y(t) + 3.8\mathscr{D}^{2.42}y(t) + 2.6\mathscr{D}^{1.798}y(t) + 2.5\mathscr{D}^{1.31}y(t) + 1.5y(t) = -2\mathscr{D}^{0.63}u(t) + 4u(t).$$

By applying the Laplace transform with zero initial conditions, the corresponding fractional-order transfer function is obtained as

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

In order to analyze this system, the graphical user interface described in Subsection 6.1.7 could be used. After launching it by typing

```
fotf_gui
```

it is possible to add the fractional-order transfer function by pressing the *Add...* button. A dialog will pop up. The system workspace name, transfer function polynomials and

input-output delay can now be entered as shown in Figure 6.3. After pressing the *OK* button the system will be saved to workspace under a variable name "G" and will appear in the GUI system list. To analyze it one first needs to select this system in the list.

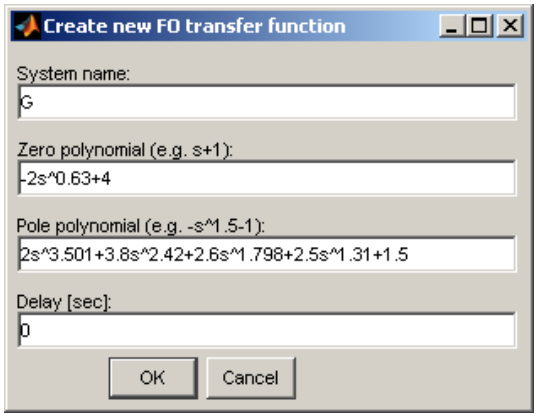
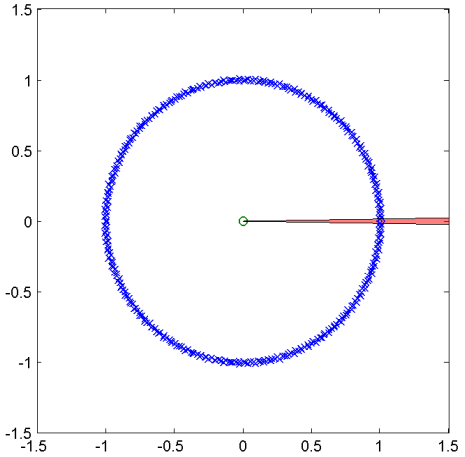
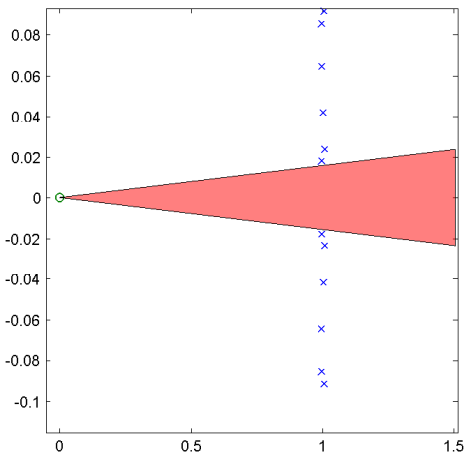


Figure 6.3: *Create new FO transfer function* dialog example

To test this system for stability one would press the *Stability test* button located in the right panel. The stability condition analysis will then be carried out. A diagram with system poles will be drawn (see 6.4a Figures and 6.4b) and an information dialog will be shown with the result (see Figure 6.4c). Additional information will also be written into the MATLAB command window.



(a) System poles



(b) Zoomed plot with unstable area

(c) Stability analysis result

Figure 6.4: System G stability analysis example

The minimum commensurate order allowed for stability analysis is $q = 0.01$. Thus, in order to determine system stability the orders are truncated and for this particular task the system becomes

$$G'(s) = \frac{-2s^{0.63} + 4}{2s^{3.5} + 3.8s^{2.42} + 2.6s^{1.79} + 2.5s^{1.31} + 1.5}.$$

It can be seen, that in this case the system is deemed stable.

The step response in range of $t = [0; 70]$ with time step $\mathrm{d}\,t = 0.01$ is given in Figure 6.5. This result is obtained by typing `0:0.01:70` in the *t vector* textbox in the *Time domain* subpanel and pressing the *Step* button. It can be seen, that the system shows damped oscillation under the step signal.

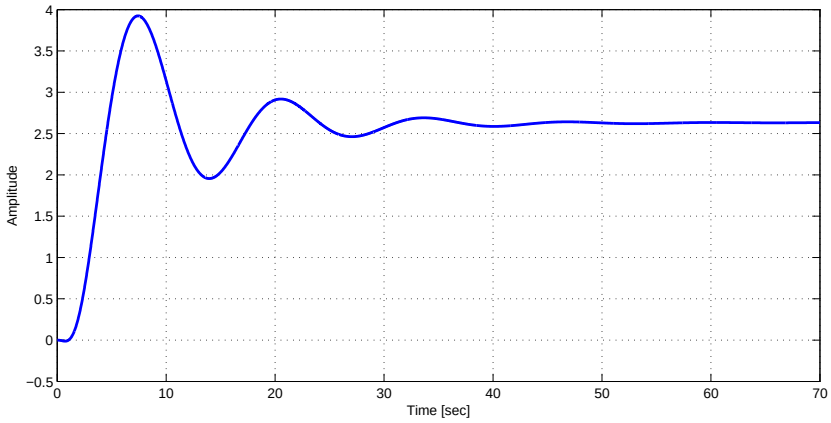


Figure 6.5: Step response of system `G`

Consider a fractional controller for this system in the form

$$G_c = 0.18 + \frac{0.29}{s^{0.93}} + 1.14s^{1.06}.$$

In order to input it into MATLAB the following can be typed in the command line:

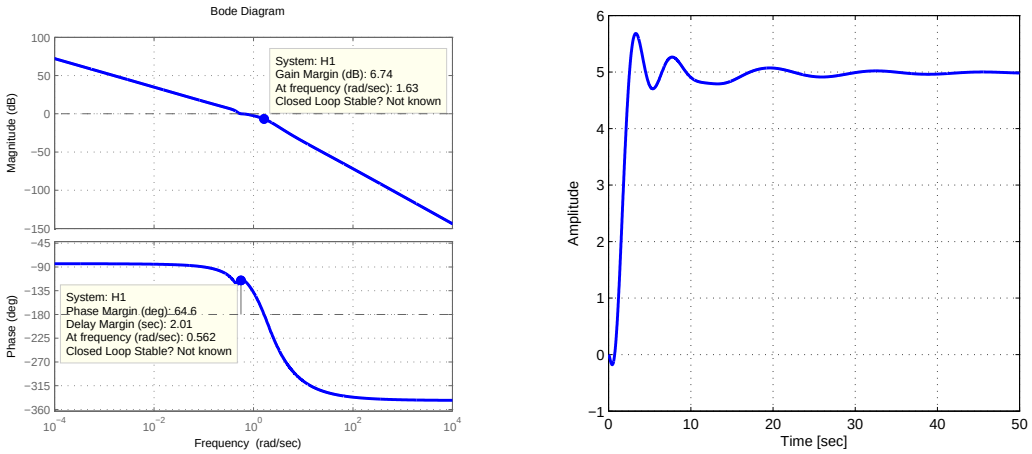
```
s = fotf('s'); Gc = 0.18+0.29/s^0.93+1.14*s^1.06;
```

Now, consider two system compositions, where `GCol` is the open-loop system and `GCcl` is the closed-loop control system:

```
GCol = G * Gc;
GCcl = feedback(G*Gc, 1);
```

Clicking *Refresh* in the left panel of the *FOTF Viewer* will update the list in which all three of the newly defined systems should be present. Let us plot a Bode diagram

for the open-loop system `GCol`. This can be done by clicking the ***Bode*** button in the ***Frequency domain*** subpanel. The obtained frequency response (with associated stability margins) is depicted in Figure 6.6a. Also consider a simulation of the control system with a set value of $SV = 5$ and in the range $t = [0;\ 50]$ with $\mathrm{d}\,t = 0.01$. The result is shown in Figure 6.6b.



(a) Open-loop frequency response

(b) Closed-loop simulation

Figure 6.6: `G1` control system analysis

Example 6.1.2 For this example consider a dynamic model of a heating furnace discussed in [22, 23] given by a differential equation

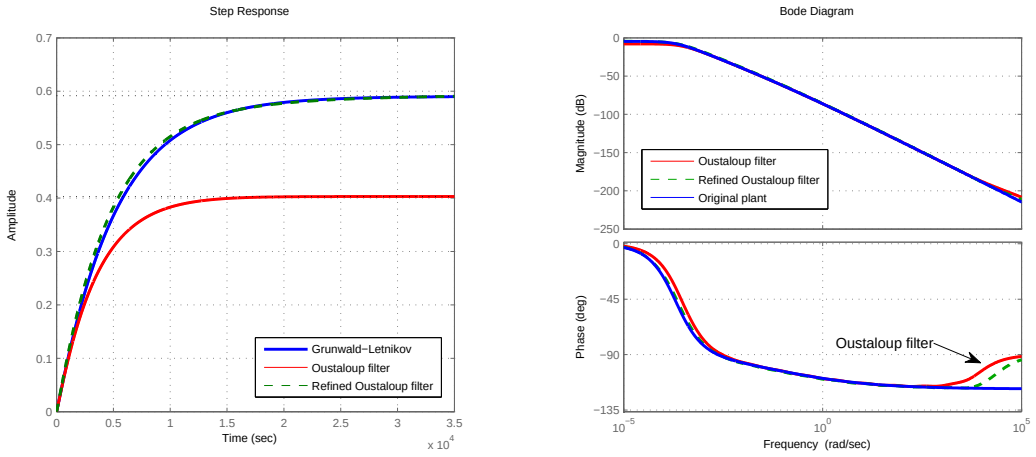
$$a_2 \mathscr{D}^{\alpha} y(t) + a_1 \mathscr{D}^{\beta} y(t) + a_0 y(t) = u(t),$$

with $\alpha = 1.31$, $\beta = 0.97$, $a_2 = 14994$, $a_1 = 6009.5$, $a_0 = 1.69$. In the Laplace domain, assuming zero initial conditions, the system is described by a fractional-order transfer function

$$G_1(s) = \frac{1}{14994 s^{1.31} + 6009.5 s^{0.97} + 1.69}.$$

We shall examine Oustaloup filter approximations of this fractional system. Converting the `fotf` system into a `zpk` object is possible from within the ***FOTF Viewer*** via the export utility. For example, let us create two filters, an Oustaloup filter `Z1` and a refined Oustaloup filter `Z2` with the default parameters ($\omega = [10^{-4};\ 10^{4}]$, $N = 5$) and compare the resulting system step response (at $t = [0;\ 35000]$ with $\mathrm{d}\,t = 0.5$) and frequency response characteristics (Figures 6.7a and 6.7b respectively).

From this example it can be clearly seen that only the refined Oustaloup filter proposed in [5] provides a valid approximation of the fractional-order system than the Oustaloup filter with the same approximation parameters. However, it is also possible to obtain a better approximation for this particular system with the Oustaloup filter by shifting the frequency range to $\omega = [10^{-6};\ 10^{2}]$.



(a) Step response comparison

(b) Frequency response comparison

Figure 6.7: Oustaloup and refined Oustaloup filter approximations of system `G1`

6.2 System Identification Module

The following functions provide means for time-domain and frequency-domain system identification by a fractional-order transfer function. Identification and validation data is given by specific data structures discussed in each subsection.

6.2.1 Time Domain Identification

6.2.1.1 Object `fidata()`

SYNTAX

```
id = fidata(y, u, t)
```

INPUT ARGUMENTS

- y — observed system output signal vector $y(t)$,

- u — observed system input signal vector $u(t)$,
- t — either a single value (sampling interval) or a sampling time vector, in seconds.

OUTPUT ARGUMENTS

- id — time-domain identification data structure containing the correctly formatted experimental data.

DESCRIPTION

This object is used to contain recorded identification information. It is used by the time-domain identification function `fid()`.

The supplied data is always checked first, ensuring equally sized column vectors. If the input vectors are not equally sized, then the following procedure will occur:

- Check length of y against length of u. If not equal, extend or trim u. Extending involves duplicating the last value of vector u until vector lengths match. Trimming cuts off the redundant part of u from the end of the vector.
- Check length of u against length of t. If not equal, extend or trim t. Extending populates the t vector by using the time-step obtained from previous values until vector lengths match. Otherwise the time vector is trimmed.

In case of unequal vector lengths a warning is always issued.

If t is given by a sampling interval, the time vector is built to match the length of vector u starting with zero time, i.e. $t = 0$.

METHODS

Table 6.4: `fidata` object methods

METHOD	DESCRIPTION	SYNTAX
`plot()`	Plots the identification/validation data	`plot(id)`
`validate()`	Validates model given by G by simulating it under validation data. A vector $err = y_G - y_{id}$ is returned. If the output argument is omitted, plots the validation diagram.	`err = validate(id, G)`

6.2.1.2 Function `fid()`

SYNTAX

```
[a,na,b,nb,Gid] = fid(fs, idd, fp, cl, el, type, op)
[a,na,b,nb,Gid] = fid(G, idd, fp, cl, el, type, op)
```

INPUT ARGUMENTS

- fs — fractional-order simulation parameter object `fsparam`, containing both initial guess model and Oustaloup filter approximation parameters,
- G — fractional-order transfer function initial guess model,
- idd — time-domain identification data object `fidata`,
- fp (optional) — option to fix either fractional-order polynomial during identification, it should be a vector containing two boolean values in form $[\ b_{fix} \quad a_{fix}\ ]$, where either can be set to *true* or *false*, if $b_{fix} = a_{fix} = 1$ no identification will be conducted and the initial guess will be immediately returned, defaults to $[\ 0 \quad 0\ ]$,
- cl (optional) — coefficient search limits in form of a vector $[\ c_{min} \quad c_{max}\ ]$, to suppress any limits, an empty matrix $[\,]$ should be supplied, which is also the default setting,

- el (optional) — exponent search limits in form of a vector $[\ e_{min}\quad e_{max}\ ]$, an empty matrix will suppress any limits, and it is the default setting,
- $type$ (optional) — identification type, which can be $'n'$ for free identification (all model parameters are optimized, default setting), $'c'$ is used to fix the polynomial coefficients and $'e'$ — to fix polynomial exponents,
- op (optional) — additional optimization options given by a MATLAB `optimset` object, defaults to setting the display of iterations during the optimization process.

OUTPUT ARGUMENTS

- a, na, b, nb — identified `fotf` object parameters,
- G_{id} — the corresponding `fotf` object.

DESCRIPTION

This function tries to identify a system by a fractional-order transfer function as described in Section 3.2. A non-linear least-squares method is employed for optimization by means of function `lsqnonlin()`, minimizing the output-error. The error vector is obtained by continuously simulating the identified system in the time domain.

The simulation type is determined by the first argument. If it is a `fotf` object, Grünwald-Letnikov simulation will be used, while if it is a `fsparam` structure, Oustaloup filter approximations are used with Control System toolbox function `lsim()` to obtain the response. The latter may improve identification speed when identifying complex models by a large experimental dataset.

The choice of initial model is very important because this determines the amount of parameters to optimize. Selecting different fix options and limiting the range of parameter search via limits may improve the result.

REMARKS

This function requires the MATLAB Optimization toolbox.

6.2.2 Frequency Domain Identification

6.2.2.1 Object `ffidata()`

SYNTAX

```
id = ffidata(mag, phase, w)
id = ffidata(r, w)
```

INPUT ARGUMENTS

- mag — frequency response magnitude vector in dB,
- $phase$ — frequency response phase angle vector in degrees,
- r — complex frequency response vector,
- ω — sampling frequencies vector in rad/s.

OUTPUT ARGUMENTS

- id — frequency-domain identification data structure containing the correctly formatted experimental data.

DESCRIPTION

The `ffidata` object contains the recorded frequency response data used for frequency-domain system identification by fractional model. Input arguments are always checked first, ensuring that these arguments are column vectors. Vectors with multiple dimensions are squeezed by using MATLAB's `squeeze()` function.

METHODS

Table 6.5: `ffidata` object methods

METHOD	DESCRIPTION	SYNTAX
`bode()`	Plots the identification/validation Bode diagram.	`bode(id)`
`validate()`	Validates model given by G by comparing the frequency response to the experimental one. Magnitude (err_m) and phase angle (err_p) error vectors are returned. If the output arguments are omitted, plots the validation diagram.	`[err_m, err_p] = validate(id, G)`

6.2.2.2 Function `ffid()`

SYNTAX

```
[a,na,b,nb,Gid,J] = ffid(idd, q, ord, method)
```

INPUT ARGUMENTS

- idd — frequency-domain identification data object `ffidata`,
- q — identified model (fractional-order transfer function) commensurate order such that $0 < q < 2$,
- ord — a vector containing desired polynomial orders in form $[n;\ m]$, where n is the pole polynomial order and m is the zero polynomial order, for method $'h'$ a single value n is required,
- $method$ (optional) — identification method string, can be $'h'$ for Hartley method, $'l'$ for Levy method or $'v'$ for Vinagre method (default).

OUTPUT ARGUMENTS

- $a,\ na,\ b,\ nb$ — identified `fotf` object parameters,
- G_{id} — the corresponding `fotf` object,
- J — identification error index.

DESCRIPTION

The `ffid()` function uses the algorithms described in Section 3.3. The NINTEGER toolbox implementation of the corresponding functions `hartley()`, `levy()` and `vinagre()` is used directly.

For the Hartley method the identified model has the form

$$G(s) = \frac{1}{c_n s^{nq} + c_{n-1} s^{(n-1)q} + \cdots + c_1 s^q + c_0}$$

while with Levy and Vinagre methods the model is identified in the form

$$G(s) = \frac{b_m s^{mq} + b_{m-1} s^{(m-1)q} + \cdots + b_1 s^q + b_0}{a_n s^{nq} + a_{n-1} s^{(n-1)q} + \cdots + a_1 s^q + 1},$$

for both cases $n,\ m$ are the pole and zero polynomial orders and q is the commensurate order of the system.

The identification error index has the following form

$$J = \frac{1}{n_\omega} \sum_{i=1}^{n_\omega} \left| G(j\omega) - \hat{G}(j\omega) \right|^2.$$

REMARKS

The required functions of NINTEGER toolbox are included in FOMCON without modification.

6.2.2.3 Function `ffid_bf()`

SYNTAX

```
[q,n,m,Gid,J] = ffid_bf(idd, method, init, nord, maxiter)
```

INPUT ARGUMENTS

- idd — frequency-domain identification data object `ffidata`,
- $method$ (optional) — identification method, can be either $'l'$ for Levy's method (default) or $'v'$ for Vinagre's method,

- $init$ (optional) — initial guess vector in the form $[q\, n\, m]$, where q is the commensurate order, n is the initial pole polynomial order, m is the initial zero polynomial order, default is $[1\ 1\ 1]$,
- $nord$ (optional) — maximum polynomial order, default is 10,
- $maxiter$ (optional) — maximum allowed iterations, default is set to unlimited.

OUTPUT ARGUMENTS

- q — found optimal system commensurate order,
- n, m — found optimal system fractional pole and zero polynomial orders,
- G_{id} — identified model given by `fotf` object,
- J — computed error index.

DESCRIPTION

This function searches for a best fit, that is for optimal parameters $[\ q\ \ n\ \ m\ ]$ for identifying a fractional-order model G by minimizing the error index J. Due to the currently used search algorithm, a local solution may be found, so it may be required to manually try different sets of initial parameters until a satisfying result is obtained.

REMARKS

The `optimize()` function is included with FOMCON toolbox.

6.2.3 Graphical User Interfaces

6.2.3.1 Function `fotfid()`

SYNTAX

```
fotfid
```

DESCRIPTION

This is the graphical user interface called *FOTF Time-domain Identification Tool*. It serves as the front-end for the `fid()` function. It is shown in Figure 6.8.

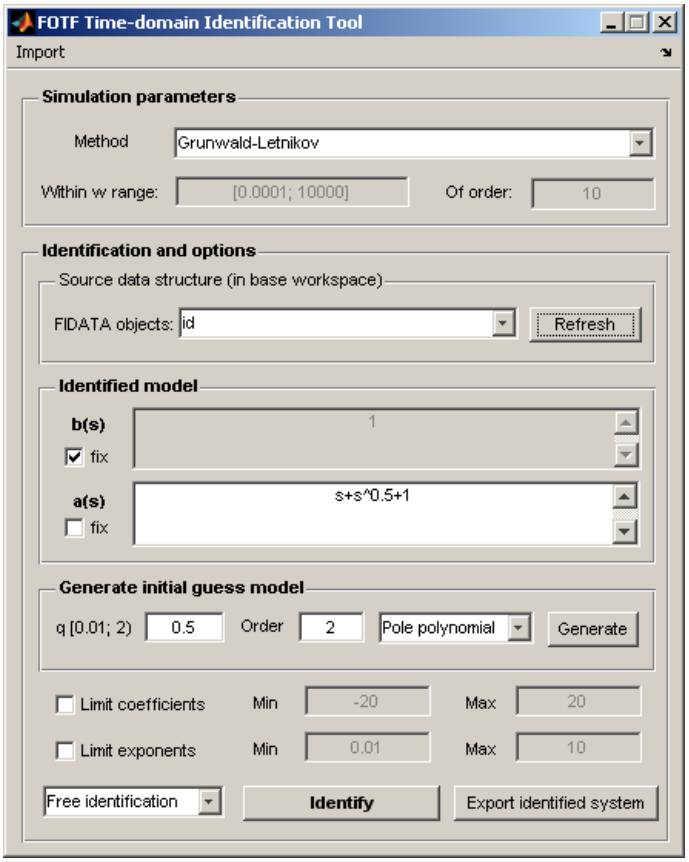


Figure 6.8: FOTF Time-domain Identification Tool user interface

The *Simulation parameters* panel allows selecting the preferred system simulation type. This includes the following:

- Grünwald-Letnikov evaluation of fractional derivatives.
- Oustaloup filter approximation.
- Refined Oustaloup filter approximation.

The last two options require setting the necessary approximation parameters.

The *Identification and options* panel has several control elements. First of all, a `fidata` structure needs to be selected from the list. The list is populated by base workspace objects of this type.

The *Identified model* panel contains textboxes with the fractional zero and pole polynomials in symbolic form. It also has checkboxes that allow to fix either polynomial during identification.

The *Generate initial guess model* panel allows to create an initial model. To do this, polynomials can be generated independently by specifying a commensurate-order q such that $0.01 \leq q < 2$, the order of the polynomial and hitting the *Generate* button.

Finally, the options panel provides means to limit the search range for coefficients and exponents. The identification type provides three options:

- free identification, i.e. all model parameters are optimized,
- fix exponents, good for obtaining fractional-order models by specifying a single commensurate order,
- fix coefficients, so only exponents are optimized.

Pressing the *Identify* button will invoke the `fid()` function and identification information will be shown in the MATLAB console. After the identification process completes, a plot with fitting results will be displayed. If the results are satisfactory, the system can be exported to workspace via the *Export to workspace* button.

Selecting $q = 1$ and fixing the exponents results in a classical, integer-order model identification problem. In such a case it is important to select either "Oustaloup filter" or "Refined Oustaloup filter" for simulation. Since fractional orders are not present in this case, approximation parameters are not important.

It is also possible to import an initial guess model via menu by selecting *Import→Initial guess model...* and typing the workspace name of the system.

6.2.3.2 Function `fotfrid()`

SYNTAX

```
fotfrid
```

DESCRIPTION

This is the graphical user interface called *FOTF Frequency-domain Identification Tool*. It serves as the front-end for the `ffid()` and `ffid_bf()` functions. The GUI is shown in Figure 6.9.

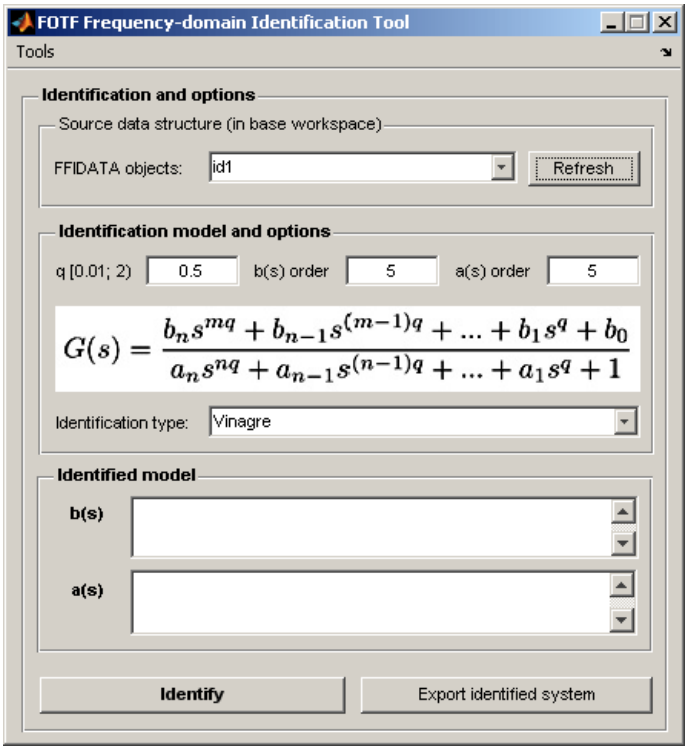


Figure 6.9: *FOTF Frequency-domain Identification Tool* user interface

The *Identification and options* panel contains the `ffidata` object selector and two sub-panels. The *Identification model and options* panel allows to choose the identified model parameters (commensurate-order $0.01 \leq q < 2$ and polynomial orders) and identification type has three choices:

- Hartley method,
- Levy method,
- Vinagre method.

An image is also displayed, showing the identified model structure for reference. The *Identified model* panel contains the recently identified model.

Pressing the *Identify* button will start the identification process. When it is finished, a plot will be drawn with identification results. The obtained model can then be saved to workspace clicking the *Export identified system* button.

Levy and Vinagre identification methods allow using best fit optimization that can be accessed from the menu by selecting *Tools→Best fit*. A dialog will pop up requesting additional information. It is required to set maximum polynomial orders and maximum number of iterations (0 for unlimited) in this dialog. The optimization procedure will then search for optimal model parameters (commensurate-order and fractional polynomial orders) and the best fit model.

6.2.4 Identification Examples

The following examples illustrate the proposed fractional-order model identification process. For the most part, known plants are used to generate identification data to verify the effectiveness of the identification algorithms. The graphical user interfaces are used for identification in all cases.

Example 6.2.1 Consider a fractional-order system given by

$$G_2(s) = \frac{1}{0.8s^{2.2} + 0.5s^{0.9} + 1}.$$

Let us generate an identification dataset based on the step response. A MATLAB function `prbs()` can be used for experimental input signal generation:

```
G2=newfotf('1','0.8s^2.2+0.5s^0.9+1');
t=(0:0.01:20)'; u=prbs(5,100,[0 1]);
u=u(450:length(t));
t=t(1:length(t)-449); y=lsim(G2,u,t)';
id1=fidata(y,u,t);
```

This creates an object `id1` of type `fidata` with 1552 points. Next, the graphical user interface is launched by typing

```
fotfid
```

Suppose that nothing is known about the initial system. Then, one can obtain results only by selecting various initial guess model commensurate orders and polynomial orders and by setting other options. For this example, one could fix the zero polynomial at "1" by ticking the checkbox next to it, select $q = 1.2$ and generate a fractional pole polynomial of order 2. Thus, the initial model is

$$G_{init}(s) = \frac{1}{s^{2.4} + s^{1.2} + 1}.$$

The other options are set as shown in Figure 6.10. Also, because of the relatively low number of identification points, the Grünwald-Letnikov simulation method can be used for this case.

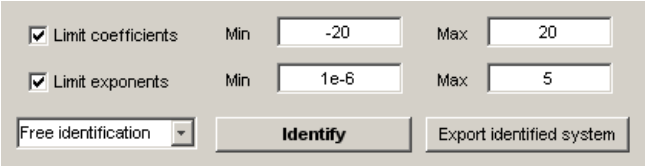


Figure 6.10: Identification options for `id1`

To begin the identification process one needs to press the *Identify* button. The result of the identification is illustrated in Figure 6.11. The obtained model has a fractional-order transfer function

$$\hat{G}(s) = \frac{1}{0.8s^{2.2} + 0.5s^{0.90002} + 1s^{2.9548e-006}}.$$

It can be seen, that the last term has an order $2.9548 \cdot 10^{-6} \to 0$. Thus the initial model was successfully recovered from the generated data. However, the initial guess was close to the final result. In a real situation, choosing the suitable initial model without any prior knowledge may take a lot of effort.

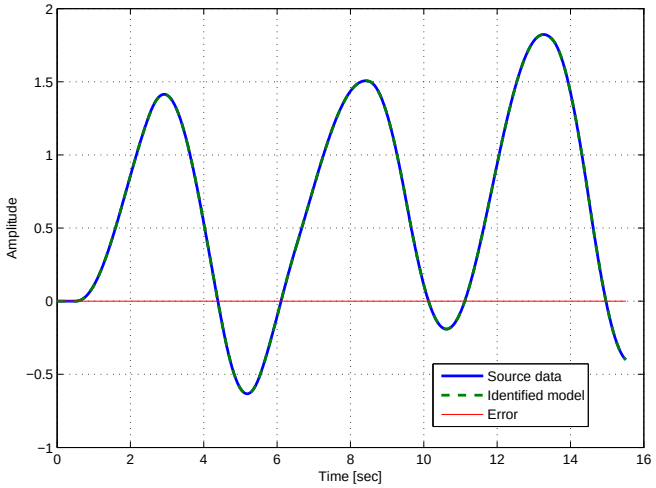


Figure 6.11: System G_2 identification result

Example 6.2.2 In the following example we will identify a real thermal system. It is given by a simple object consisting of a heating component and a cooling fan.

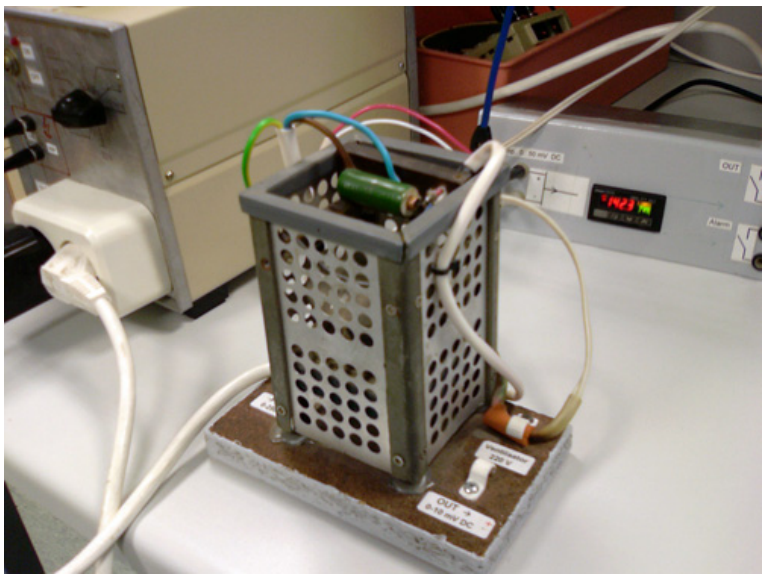


Figure 6.12: Thermal object used in experiment

The schematic diagram for this experiment is depicted in Figure 6.13. The voltage from the voltage source is directly applied to the heater. An approximate equation to determine the resulting temperature is given by

$$T = c \cdot U^2 + T_0 \ [^\circ\mathrm{C}] \,,$$

where c is a constant coefficient, U is the applied voltage in volts and T_0 is the initial temperature. The temperature is measured using a type *K* thermocouple with a DC output of 0...10 mV, amplified with a gain of 30 and fed into a *Velleman PCS100* oscilloscope, which is used to register both the temperature obtained from the amplified thermocouple signal and the voltage source signal.

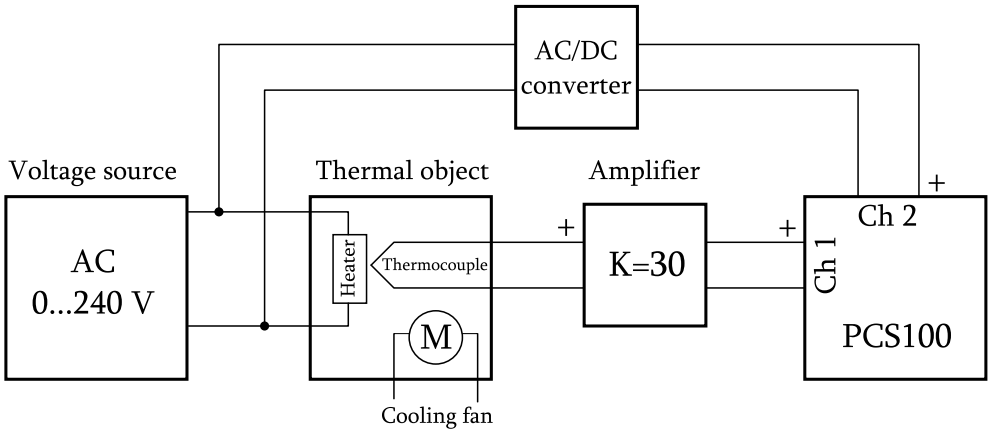


Figure 6.13: Experiment schematic diagram

Data was collected from three consecutive experiments. Different voltage set values were used. Due to some limitations of used software and hardware, a total of 1700

points were recorded with a sampling interval of $T_s = 2$ seconds. The obtained system output vector was then filtered ensuring zero phase distortion using a low-pass filter by means of MATLAB's `filtfilt()` function, and a transformation was applied so that the output signal vector would contain real temperature values in °C. To account for zero initial conditions requirement the temperature output signal was also shifted such that $t = 0 \rightarrow y(t) = 0$. The system input vector was reconstructed by using registered input signal transition times and also a transformation was applied such that $\hat{u}(t) = 0.01 \cdot u^2(t)$ — the obtained input signal is thus a rough approximation of the final temperature value. A plot of the identification dataset can be seen in Figure 6.14.

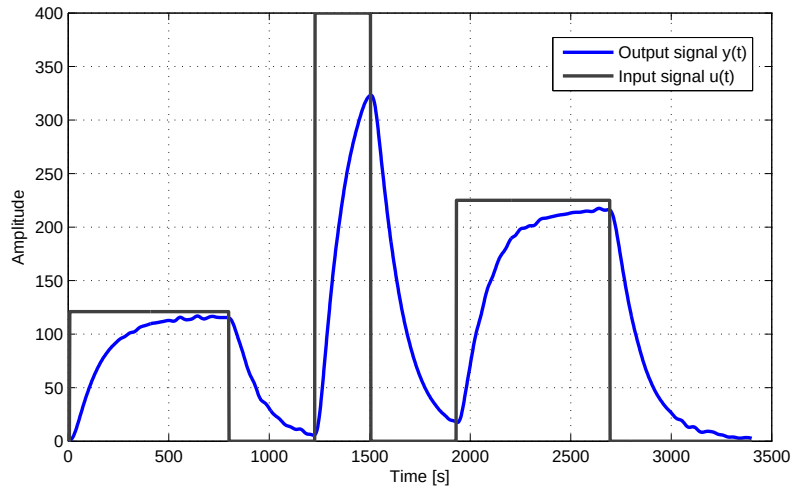


Figure 6.14: Plot of the identification dataset for the thermal object

The system can now be identified using the `fotfid` tool. From previous knowledge, we know that in case of an integer-order model, this system can be approximated by a second order model. Thus, one could get the initial guess model by generating a fractional pole polynomial with $q = 1$, $n = 2$ and fixing the zero polynomial at "1" so that a classical, integer order model is initially obtained in the form

$$G_{init}(s) = \frac{1}{s^2 + s + 1}.$$

The system is then identified using the *Free identification* method within coefficient limits $c_{lim} = [0;\ 3000]$ and exponent limits $e_{lim} = [10^{-9};\ 3]$. Identification yields the following model:

$$\hat{G}(s) = \frac{1}{2012.409s^{1.8063} + 107.2882s^{0.93529} + 1.0305}.$$

Validation is carried out with the same dataset as for identification. The corresponding plot is given in Figure 6.15.

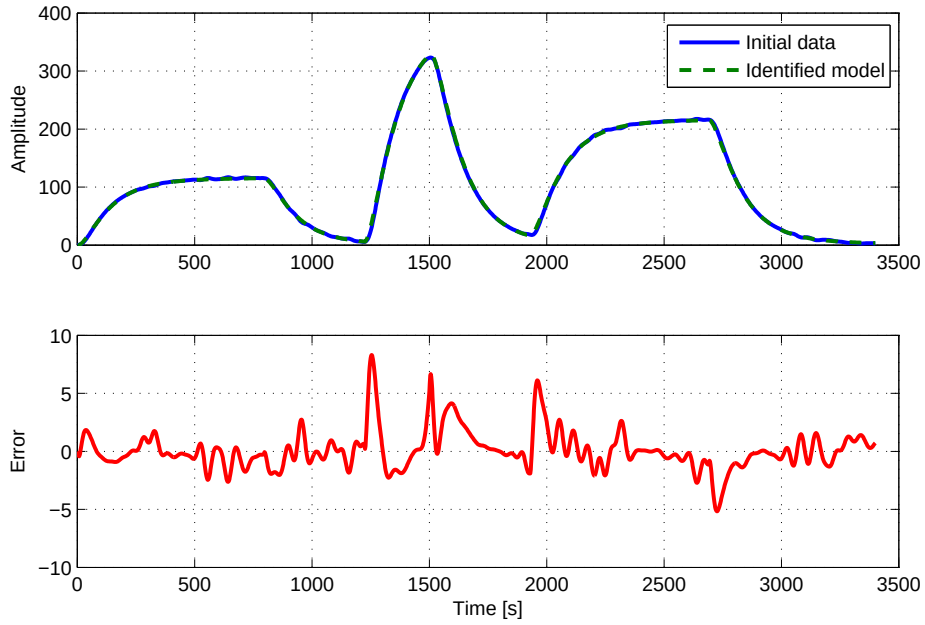


Figure 6.15: Thermal system fractional model validation

Two integer-order models were obtained from the same experimental dataset by using the MATLAB Identification toolbox for comparison. Results are provided in Table 6.6.

Table 6.6: Process model identification comparison

	IDENTIFIED MODEL	SQUARE ERROR NORM
$\hat{G}_{frac}(s) =$	$\frac{1}{2012.409s^{1.8063}+107.2882s^{0.93529}+1.0305}$	71.7702
$\hat{G}_{io1}(s) =$	$\frac{0.96039}{2655.4725s^2+151.626s+1}\mathrm{e}^{-1.1844s}$	72.3396
$\hat{G}_{io2}(s) =$	$\frac{0.96035}{2835.2438s^2+152.593s+1}$	81.8971

Taking the square error norm as a measure of model precision, one could say that the fractional-order model $\hat{G}_{frac}$ is more accurate than the integer-order models $\hat{G}_{io1}$ and $\hat{G}_{io2}$. This is to be expected due to the properties of fractional operators and the extra degrees of modeling freedom. However, in order to claim this explicitly one would need to use hardware and software methods with a more strict precision requirement.

A comparison of frequency responses of systems $\hat{G}_{frac}$ and $\hat{G}_{io2}$ is given in Figure 6.16. The system $\hat{G}_{io1}$ has a delay term which introduces rapid phase accumulation and is therefore not shown. Suffice it to say that its magnitude frequency behavior is

similar to that of system $\hat{G}_{io1}$.

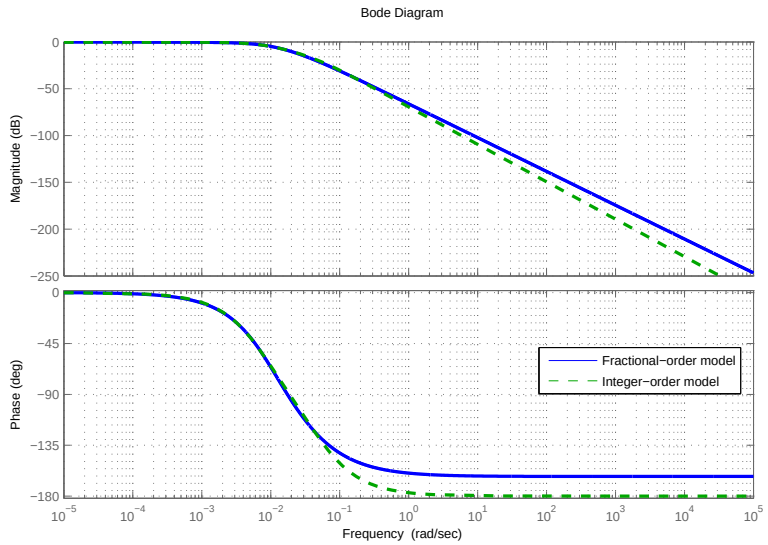


Figure 6.16: Frequency responses of identified fractional-order and integer-order systems

Example 6.2.3 In this example we will illustrate the use of the frequency-domain identification tool. Consider a plant given by

$$G(s) = \frac{s^{0.32} + 5}{100s^{1.92} + 20s^{0.96} - 5s^{0.64} + 1}.$$

For this example we will generate an identification dataset `fid1` with 50 logarithmically spaced frequency sample points in the range $\omega = [10^{-4}; 10^{4}]$. This can be done by writing the following into the MATLAB command line:

```
G    = newfotf('s^0.32+5', ...
                '100s^1.92+20s^0.96-5s^0.64+1');
w    = logspace(-4, 4, 50);
fid1 = ffidata(freqresp(G, w), w);
```

Now, it is time to launch the frequency domain identification tool by typing

```
fotfrid
```

Next, one should select the newly generated dataset, choose the identification parameters and algorithm. In order to save time, the *Best fit* tool can be used with *Levy* and *Vinagre* algorithms. If nothing about the model is known and it is impossible to predict the model orders, then the commensurate order and polynomial orders are to be chosen by trial and error.

For this example, if one selects $q = 0.2$ and uses the best fit optimization tool with a maximum order of $N = 6$ and unlimited number of iterations with the Vinagre method, then the following model is obtained:

$$G_{id}(s) = \frac{\begin{array}{c} 2.6322 \cdot 10^{-15} s^{1.92} - 1.4416 \cdot 10^{-13} s^{1.6} + 3.2699 \cdot 10^{-12} s^{1.28} \\ -3.7288 \cdot 10^{-11} s^{0.96} + 2.1969 \cdot 10^{-10} s^{0.64} + s^{0.32} + 5 \end{array}}{\begin{array}{c} 100 s^{1.92} + 1.6987 \cdot 10^{-8} s^{1.6} - 1.0219 \cdot 10^{-8} s^{1.28} \\ +20 s^{0.96} - 5 s^{0.64} - 1.5758 \cdot 10^{-10} s^{0.32} + 1 \end{array}}$$

Identification result is presented in Figure 6.17. Notice the residual terms. By truncating this system using

```
Gid = trunc(Gid, 1e-5, 1e-5)
```

the initial model is successfully recovered.

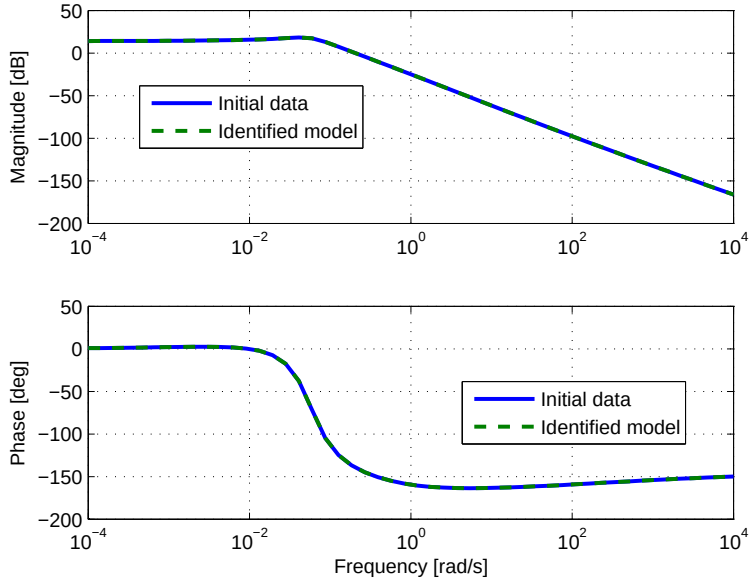


Figure 6.17: Frequency-domain identification example

Obviously, the system used in this example was relatively easy to identify. In practical cases, one should carefully consider the choice of the commensurate order. The identified system polynomials may be of a very high order, in which case the *Best fit* tool will be less useful due to large computational efforts involved.

For a practical example of frequency-domain identification see the fractional lead-lag compensator realization example in Subsection 6.3.5.

6.3 System Control Module

The functions in this module provide means to design fractional-order controllers. Tuning methods for the PI$^{\lambda}$D$^{\mu}$ controller are also provided.

6.3.1 Fractional-Order Controller Design

6.3.1.1 Function `fracpid()`

SYNTAX

```
[Gc, cttype] = fracpid(Kp, Ki, lambda, Kd, mu)
```

INPUT ARGUMENTS

- Kp — proportional gain,
- Ki — integrator gain,
- $lambda$ — integrator order (positive),
- Kd — differentiator gain,
- mu — differentiator order.

OUTPUT ARGUMENTS

- Gc — resulting controller given as `fotf` object,
- $cttype$ — string containing controller type (“P”, “PI”, “PD” or “PID”).

DESCRIPTION

This function returns a fractional-order transfer function that corresponds to a fractional-order PID controller given by

$$G_c = K_p + \frac{K_i}{s^{\lambda}} + K_d s^{\mu}.$$

6.3.1.2 Function `tid()`

SYNTAX

```
Gc = tid(Kt, n, Ki, Kd)
```

INPUT ARGUMENTS

- Kt — tilt gain,
- n — tilt integrator order denominator in $\frac{1}{n}$,
- Ki — integrator gain,
- Kd — differentiator gain.

OUTPUT ARGUMENTS

- Gc — resulting controller `fotf` object.

DESCRIPTION

This function returns a `fotf` object corresponding to a TID controller in form

$$G_c = \frac{K_t}{s^{1/n}} + \frac{K_i}{s} + K_d s.$$

6.3.1.3 Function `frlc()`

SYNTAX

```
[r, mag, ph] = frlc(K, x, lambda, alpha, w)
```

INPUT ARGUMENTS

- K — fractional-order lead-lag compensator parameter k',
- x — fractional-order lead-lag compensator parameter x,
- $lambda$ — fractional-order lead-lag compensator parameter λ,
- $alpha$ — fractional-order lead-lag compensator order α,

- ω — vector with frequency at which to calculate the compensator frequency response in rad/s.

OUTPUT ARGUMENTS

- r — complex frequency response vector,
- mag — frequency response magnitude vector,
- ph — frequency response phase angle vector in degrees.

DESCRIPTION

This function calculates the frequency response of a fractional-order lead-lag compensator given by

$$G_c(j\omega) = k' \left(\frac{\lambda j\omega + 1}{x\lambda j\omega + 1} \right)^{\alpha}$$

at frequencies given in ω. Since there is currently no direct way of realizing an implicit fractional-order system directly, this information can be used to identify a fractional or integer order model to implement the controller.

Magnitude and phase are only calculated when requested, i.e. if there is more than one output argument specified.

6.3.2 Integer-Order Controller Tuning By Process Model Approximation

6.3.2.1 Function `fotf2io()`

SYNTAX

```
[K, L, T] = fotf2io(sim, model, init, op)
```

INPUT ARGUMENTS

- sim — a `fsparam` data structure,

- $model$ — a string containing a process model identifier, can be $'fopdt'$, $'ipdt'$ or $'foipdt'$,
- $init$ (optional) — initial parameters in form $[\ K \quad L \quad T\ ]$, defaults to $[\ 50 \quad 50 \quad 50\ ]$,
- op (optional) — MATLAB `optimset` object with additional optimization parameters.

OUTPUT ARGUMENTS

- $K,\ L,\ T$ — process model parameters (see description).

DESCRIPTION

This function is used to obtain an process model approximation of a fractional-order system by optimizing a set of parameters by means of Optimization toolbox function `lsqnonlin()`. The error in step response is minimized.

There are three process model types:

- First-order plus dead time (FOPDT) given by $G(s) = \dfrac{K}{1+Ts}\,\mathrm{e}^{-Ls}$,
- Integrator plus dead time (IPDT) given by $G(s) = \dfrac{K}{s}\,\mathrm{e}^{-Ls}$,
- First-order integrator plus dead time (FOIPDT) given by $G(s) = \dfrac{K}{s(1+Ts)}\,\mathrm{e}^{-Ls}$.

These models are applicable only if the fractional-order step response is aperiodic. They can be used as means of integer-order controller tuning.

REMARKS

This function requires the MATLAB Optimization toolbox.

6.3.3 PI$^{\lambda}$D$^{\mu}$ Controller Optimization

6.3.3.1 Object `fpopt()`

SYNTAX

```
opt = fpopt(type,p_0,pmax,pmin,metric,gm,pm,strict,op)
```

INPUT ARGUMENTS

- $type$ — optimization type, can be set to fix $'n'$ (none; default), $'e'$ — exponents or $'c'$ — coefficients,
- p_0 — initial PID parameters in form $[\ K_p \quad K_i \quad K_d \quad \lambda \quad \mu\ ]$,
- $pmax$ — maximum values of parameters in p,
- $pmin$ — minimum values of parameters in p,
- $metric$ — a string with fractional PID performance metric, can be $'ise'$ (default), $'iae'$, $'itse'$ or $'itae'$,
- gm — gain margin specification (in dB), defaults to 10,
- pm — phase margin specification (in degrees), defaults to 60,
- $strict$ — strictness option (boolean value; default is *false*),
- op — additional optimization options `optimset` object (default setting limits the number of allowed optimization iterations to 50).

OUTPUT ARGUMENTS

- opt — data structure that contains fractional PID tuning options.

DESCRIPTION

This object is designed for convenient specification of fractional PID optimization options. The performance metric option refers to the following error indices:

- Integral square error $ISE = \int_0^t e^2(t)\ \mathrm{d}t$,
- Integral absolute error $IAE = \int_0^t \left|e(t)\right|\ \mathrm{d}t$,
- Integral time-square error $ITSE = \int_0^t te(t)^2\ \mathrm{d}t$,
- Integral time-absolute error $ITAE = \int_0^t t\left|e(t)\right|\ \mathrm{d}t$,

where e is the error vector.

The strictness option, if set to *true*, requires that the initial values of the fractional PID immediately satisfy the minimum design specifications given by gm (gain margin) and pm (phase margin), i.e. the initial solution in p_0 must be feasible. When set to $false$ will allow control system evaluation outside of these specifications.

All input arguments are optional.

6.3.3.2 Function `fpid_optimize()`

SYNTAX

```
[Kp,Ki,Kd,lam,mu] = fpid_optimize(fs, fopt, G)
```

INPUT ARGUMENTS

- fs — a `fsparam` simulation structure,
- $fopt$ — a `fpopt` optimization options structure,
- G (optional) — any valid MATLAB LTI system (`tf`, `zpk`, `ss`).

OUTPUT ARGUMENTS

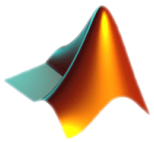
- K_p, K_i, K_d, lam, mu — obtained optimal fractional PID parameters.

DESCRIPTION

This function searches for optimal fractional PID parameters according to specifications in the `fpopt` object using principles found in Subsection 4.1.3 of Chapter 4. A control system is built on every optimization iteration and its step response is evaluated. The error vector is then obtained taking $e(t) = 1 - y(t)$, where $y(t)$ is the control system closed-loop step response. It is used to calculate the performance index, which is minimized by means of the `optimize()` function [26].

To fulfill the design specifications the open-loop frequency response is also evaluated at every iteration and gain and phase margins are obtained. These are compared to the specified ones so that conditions $Gm_{real} > Gm_{spec}$ and $Pm_{real} > Pm_{spec}$ hold true.

With the strict option set to *true*, the initial PID parameters are required to immediately fulfill the given design specifications.



`optimize()` uses a variant of the Nelder-Mead algorithm. In order to introduce bound constraints it uses coordinate transformation, while for other types of constraints (including non-linear constraints) it utilizes penalty functions. This function is especially useful when the objective function is either hard, or impossible to differentiate. It is also self-contained meaning that MATLAB's Optimization toolbox does not need to be installed for it to work.

After the optimization is complete, the achieved gain and phase margins are displayed. While there is currently no possibility to use the sensitivity functions as design specifications, the frequencies corresponding to desired signal damping in dB can be found by typing

```
[wt, ws] = csens(Gct, A, B, w);
```

where frequencies are such that $\forall\omega \geq \omega_t\,[\mathrm{rad/sec}] \rightarrow \left|T(j\omega_t)\right| = A\,[\mathrm{dB}]$ and $\forall\omega \leq \omega_s\,[rad/sec] \rightarrow \left|S(j\omega_s)\right| = B\,[\mathrm{dB}]$. These design specifications will be introduced in the next version of the FOMCON toolbox.

REMARKS

The `optimize()` function is included with FOMCON toolbox.

6.3.4 Graphical User Interfaces

6.3.4.1 Function `fpid()`

SYNTAX

```
fpid
```

DESCRIPTION

This is the Fractional PID Design Tool graphical user interface (see Figure 6.18). The interface contains an image of the unity feedback control system, which is used to

obtain the fractional control system. There are also two panels.

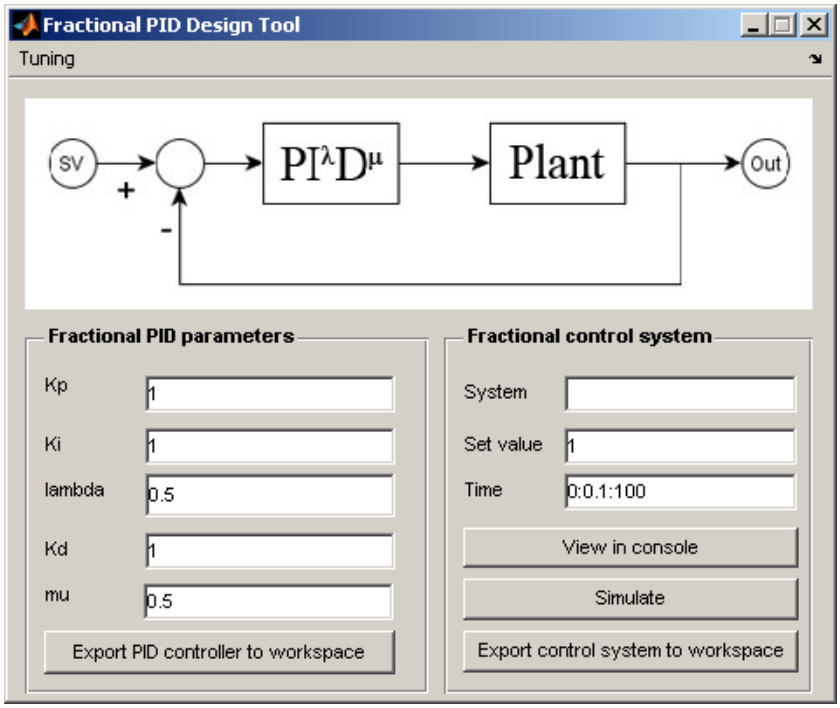


Figure 6.18: *Fractional PID Design Tool* graphical user interface

The *Fractional PID parameters* panel allows to enter the fractional PID parameters. Pressing the *Export PID controller to workspace* button will trigger a pop up asking for a workspace variable name to save the fractional controller.

The *Fractional control system* panel allows to view the controller, plant and full control system in the console, simulate the designed control system and export it to MATLAB workspace. Set the workspace variable name of the desired system in the *System* textbox.

The system set in the *System* textbox may be any valid LTI object (`tf`, `zpk`, `ss`). In such a case when simulating and exporting a control system an additional dialog is shown which asks for Oustaloup filter parameters to be used for the approximation of the control system.

The *Tuning* menu allows to access tuning tools:

- Integer-order PID tuning tool by process model approximation,
- Fractional PID optimization tool.

The initial values for fractional PID optimization will be current values, taken from the left panel.

6.3.4.2 Function `fpid_optim()`

SYNTAX

```
fpid_optim
```

DESCRIPTION

The graphical user interface of the *FPID Optimization Tool* is shown in Figure 6.19. This is the front-end for the `fpid_optimize()` function.

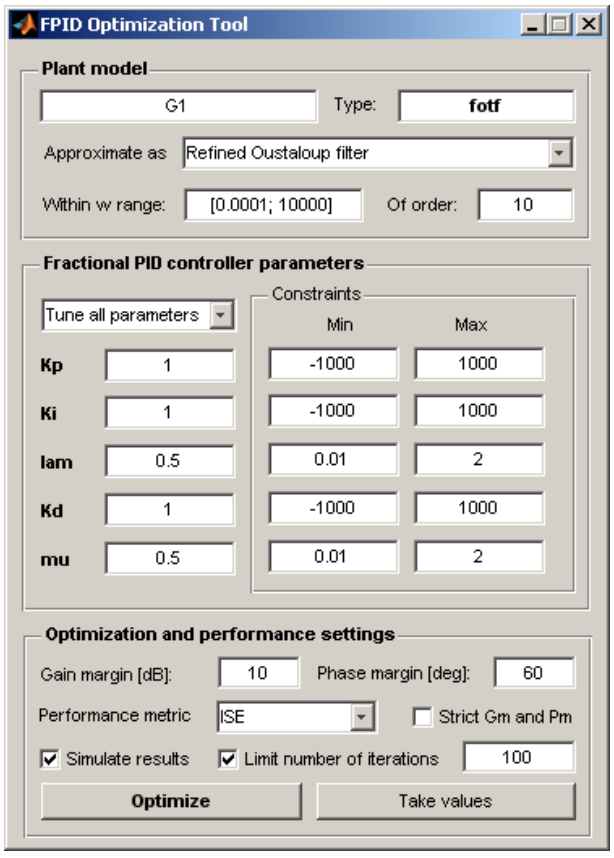


Figure 6.19: *FPID Optimization Tool* graphical user interface

The *Plant model* panel contains user controls to select the plant for which to obtain the fractional PID and simulation options to use for control system approximation. Please note, that even if the plant is not given by a `fotf` object, simulation parameters still need to be supplied for the controller approximation. Also note, that Grünwald-Letnikov simulation option is not provided. This is mainly due to simulation speed and

also because a sophisticated auto-ranging algorithm is used, courtesy of the Control System toolbox.

The *Fractional PID controller parameters* panel is used for entering all controller parameters, their minimum and maximum allowed values. Tuning method is also selected here, available options are:

- tune all parameters,
- fix exponents,
- fix gains.

When exponents are fixed at $\lambda = \mu = 1$, the tuning problem becomes that for the integer-order PID controller.

It is also possible to tune PI$^{\lambda}$ and PD$^{\mu}$ controllers with this tool. In order to do this, the minimum and maximum values, as well as values themselves of either K_d/μ or K_i/λ need to be set to zero. The corresponding parameters are then excluded from the optimization procedure.

Finally, the *Optimization and performance settings* panel is used to set the desired performance specifications, choose the performance metric (*ISE*, *IAE*, *ITSE* or *ITAE*), select the design specification strictness, simulate optimization results on completion and limit the number of optimization iterations. Pressing the *Optimize* button will commence the optimization process. After the optimal fractional PID settings are obtained, it is possible to launch the PID design tool (passing the values to it) by pressing the *Take values* button.

6.3.4.3 Function `iopid_tune()`

SYNTAX

```
iopid_tune
```

DESCRIPTION

The *Integer-order PID Tuning Tool* graphical user interface is shown in Figure 6.20. This tool is used to identify a fractional-order system by a process model and to tune an integer-order PID based on the obtained model.

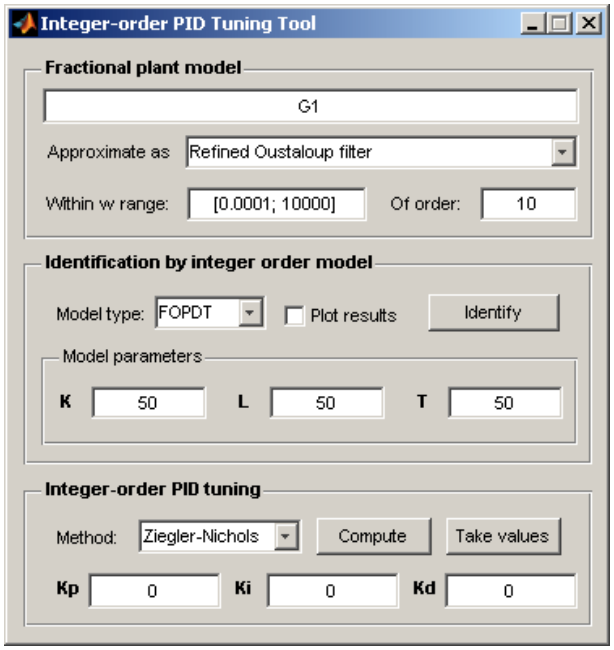


Figure 6.20: *Integer-order PID Tuning Tool* graphical user interface

The GUI is organized as follows. The Fractional plant model panel contains controls to choose the desired fractional-order model and approximation parameters for simulation, which is conducted during the identification process. The user can select between two types of approximation (either of the Oustaloup filter types).

The *Identification by integer order model*, i.e. process model panel is used to obtain the process model parameters by using the `fotf2io()` function. See its description for details. It is also possible to plot the identification results to assess obtained model validity.

Finally, the *Integer-order PID tuning* panel allows to find classical PID parameters by using some widely known methods summarized in [5], among which are the following:

- Ziegler-Nichols PID tuning formula,
- Åström-Hägglund (AMIGO) PID tuning formula,
- Chien-Hrones-Reswick (set point regulation with 20% overshoot) tuning formula,

- Chien-Hrones-Reswick (disturbance rejection with 20% overshoot) tuning formula,
- Cohen-Coon tuning formula.

Pressing the *Compute* button recalculates the PID gains with regard to process model parameters K, L, T, while pressing *Take values* brings up the *Fractional PID Design Tool*.

6.3.5 Controller Design and Optimization Examples

Example 6.3.1 Consider a system discussed previously, which is a model of a heating furnace:

$$G(s) = \frac{1}{14994s^{1.31} + 6009.5s^{0.97} + 1.69}.$$

In this example, we shall design a fractional PID controller for the given plant.

First of all, it can be seen from previous analysis (conducted in Subsection 6.1.8) that the step response of this fractional-order system is aperiodic. Therefore, one could try the FOMCON integer-order PID design tool first. To launch it, the following needs to be typed in MATLAB command window:

```
iopid_tune
```

and the user interface will be shown. One would then type the name of the MATLAB workspace variable into the *Fractional PID model* textbox. Considering that this plant is better approximated with the refined Oustaloup filter, this filter should be chosen in the simulation options. Next, one can select the model type "FOPDT" and set the initial guess parameters such that $K = L = T = 100$. With these parameters, the system is identified from the step response as the following FOPDT model (see Figure 6.21 for an illustration of the identification results):

$$G_{FOPDT}(s) = \frac{0.588586}{1 + 4801.86s}\mathrm{e}^{-11.2571}.$$

Next, the Ziegler-Nichols tuning formula is used to obtain integer-order PID parameters. These are calculated as $K_p = 802.915$, $K_i = 24.3806$, $K_d = 6.09515$.

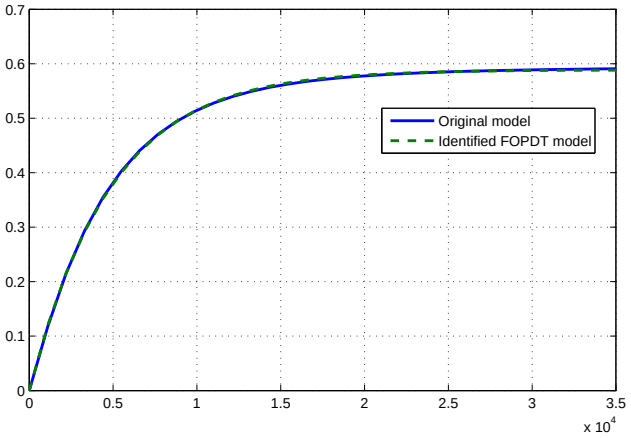


Figure 6.21: Fractional system identification by first-order plus dead time process model

We shall now research the potential improvement of these conventional PID settings by introducing the fractional orders to the integrator and differentiator. For this purpose we shall launch the fractional PID optimizing tool by typing

```
fpid_optim
```

and entering the gains obtained earlier. It is also possible to access this tool (copying the gain and exponent values automatically) by clicking *Take values* in the integer-order PID tuning tool and then choosing *Tuning→Optimize* from the menu of the fractional PID design tool.

In the optimization tool, the simulation method should be changed to "Refined Oustaloup filter", optimization method should be "Fix gains". Other options that are used in this case are shown in Figure 6.22.

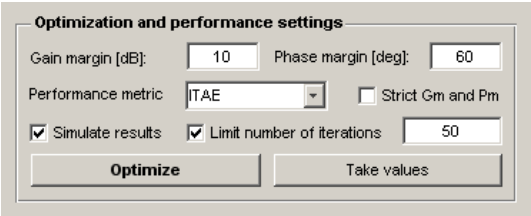
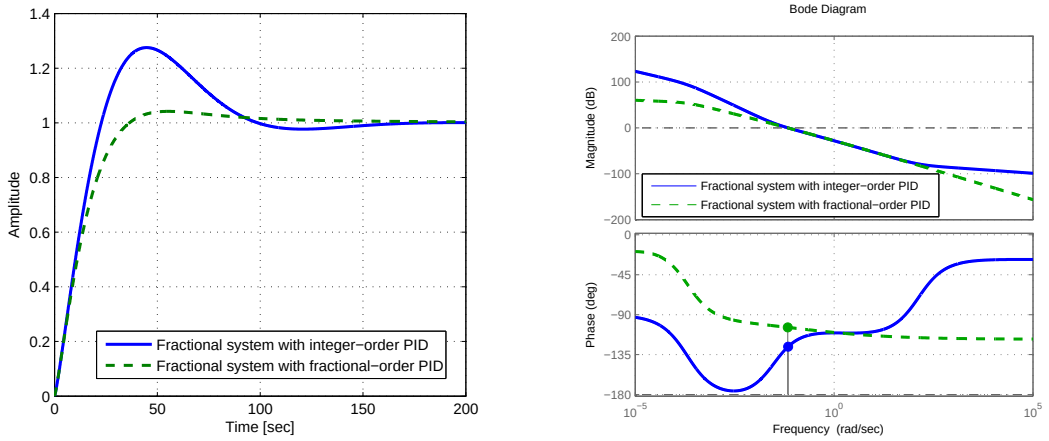


Figure 6.22: Optimization settings

Optimizing with initial orders set to $\lambda = \mu = 1$ does not bring a significant improvement possibly due to a local solution. However, changing $\lambda = 0.6$, $\mu = 0.5$ and limiting the search range (minimum and maximum values) to λ, $\mu = [0.1; 2]$ and running optimization again yields the following controller:

$$G_c(s) = 802.915 + \frac{24.3806}{s^{0.3207}} + 6.0952s^{0.1}.$$

The comparison of the step response of the control systems with different controllers is given in Figure 6.23a, while the comparison of the open-loop frequency response is shown in Figure 6.23b. It can be clearly seen, that by tuning only the orders of the fractional controller a significant improvement is achieved in both the transient response and frequency-domain characteristics.



(a) Closed-loop simulation

(b) Open-loop frequency response

Figure 6.23: System response comparison with integer-order and fractional-order PID controllers

Example 6.3.2 Consider again the plant given in the previous example, i.e.

$$G(s) = \frac{1}{14994s^{1.31} + 6009.5s^{0.97} + 1.69}.$$

We will now illustrate the design of a TID controller using a method proposed in [21].

First, assume the gain crossover frequency is $\omega_{cg} = 0.05\, rad/sec$ and the desired phase stability margin $\varphi_m = 45°$. According to the proposed tuning method (choosing the Tilt component parameter $n = 3$), the following steps are carried out:

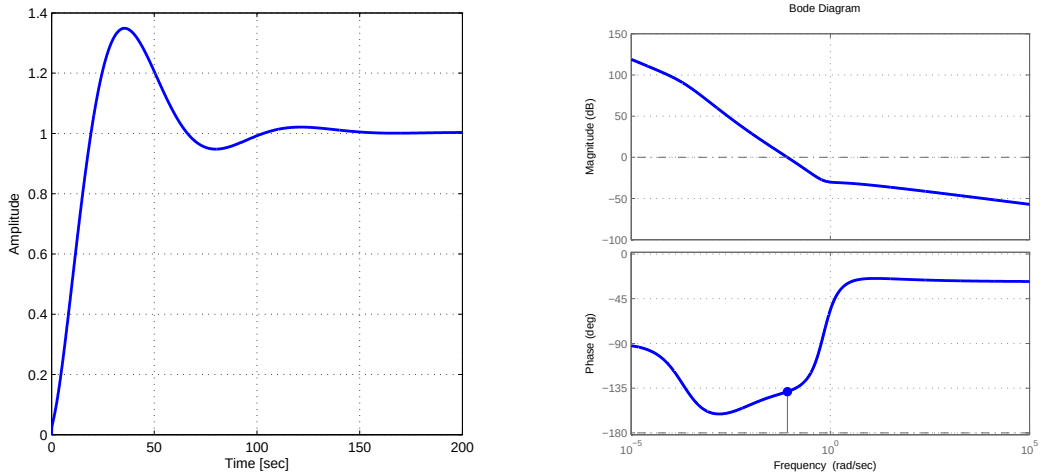
1. Setting the gains $K_i = K_d = 0$, the parameter Kt is sought, at which the loop gain at frequency ω_{cg} is 0 dB,
2. The gain K_d is sought such that the phase stability margin at ω_{cg} is about $5°$ larger than desired,
3. The gain K_i is calculated from $K_i = 0.25 K_t \omega_{cg}^{(1-1/n)}$.

Following this tuning procedure by analysing the open-loop frequency behavior the controller parameters are obtained as $K_t = 444$, $K_i = 15.065$, $K_d = 748$. The

controller and the full control system are then designed by typing the following in MATLAB:

```
G1  = newfotf('1','14994s^1.31+6009.5s^0.97+1.69');
Gc  = tid(444,3,15.065,748);
Gct = feedback(G1*Gc, 1);
```

The step response of the closed-loop control system is given in Figure 6.24a. Also consider the open-loop frequency response in Figure 6.24b. By introducing the K_i gain the phase stability margin was reduced, thus the need for a larger stability margin in tuning step 2 is made clear.



(a) Closed-loop simulation

(b) Open-loop frequency response

Figure 6.24: Control loop with the TID controller analysis

Example 6.3.3 Consider a model of a thermal object we obtained in Subsection 6.2.4:

$$G = \frac{1}{2012.4087s^{1.8063} + 107.2882s^{0.93529} + 1.0305}.$$

We will now design a PI$^{\lambda}$D$^{\mu}$ controller for this plant using the *FPID Optimization Tool*.

Initially the PID parameters are set to $K_p = K_i = K_d = 100$, $\lambda = \mu = 1$. The exponents are fixed so that an integer-order PID could be designed. Search limits are set to $K = [-500; 500]$ for gains and $\gamma = [0.01; 2]$. For simulation, the refined Oustaloup filter approximation is used with default parameters ($\omega = [0.0001; 10000]$, $N = 10$). Specifications are as follows. Gain margin is set to 10 dB, while phase margin to 60 degrees (non-strict). Performance metric is *ITAE*.

Optimization with these settings leads to the following integer-order PID controller parameter set: $K_p = 49.7747$, $K_i = 0.45718$, $K_d = 204.8202$. Obtained open-loop phase margin is $\varphi_m = 59.0896°$. Next the gains are fixed and integrator and differentiator orders are set to $\lambda = 0.9$ and $\mu = 0.8$ respectively. The strict option is enabled and phase margin specification is changed to $59°$ to make sure this minimum is maintained throughout the following optimization process. The optimization is then continued. As a result, the orders are found such that $\lambda = 1.0234$ and $\mu = 0.53387$.

A comparison of simulation of the designed control systems with a set value $SV = 150$ is shown in Figure 6.25. Again, by tuning only the orders of the controller a better result is achieved.

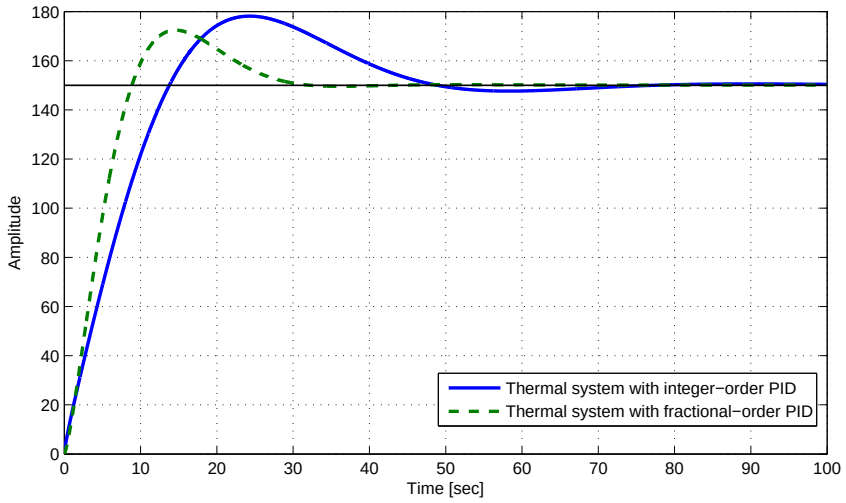


Figure 6.25: Control system for thermal object comparison

Example 6.3.4 Consider an integer-order plant given by a model

$$G(s) = \frac{2}{s(0.5s + 1)}.$$

In this example, we will realize a fractional lead-lag compensator for this plant discussed in [2]. The gain crossover frequency is chosen such that $\omega_{cg} = 10\, rad/sec$. At this frequency the plant has a magnitude of $-28.1291\,\mathrm{dB}$ and a phase of $-168.69°$. To achieve a magnitude of 0 dB at the gain crossover frequency and a phase margin $\varphi_m = 50°$ the fractional lead compensator is designed with parameters $k' = 10$, $x = 0.005$, $\lambda = 0.6404$, $\alpha = 0.5$ and thus has the following implicit fractional-order transfer function

$$G_c(s) = 10\left(\frac{0.6404s+1}{0.0032s+1}\right)^{0.5}.$$

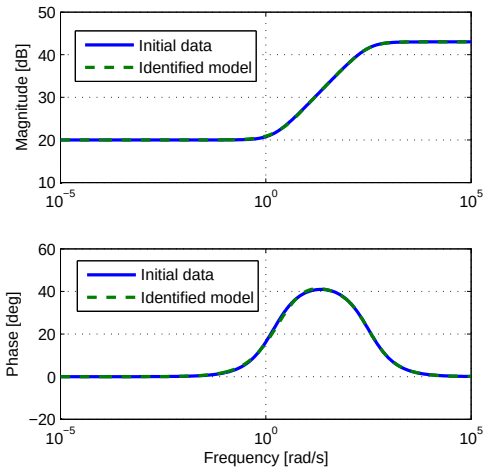
In order to implement this controller, consider the method described in Section 4.2. First, the frequency response data for this controller is obtained using the `frlc()` function in the range $\omega = [10^{-5};\, 10^{5}]$ and a frequency-domain identification dataset is created by typing the following in MATLAB:

```
w   = logspace(-5,5,1000);
r   = frlc(10,0.005,0.6404,0.5,w);
flc = ffidata(r,w);
```

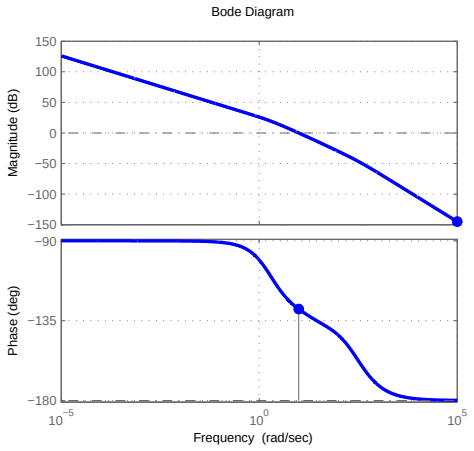
Next, the `fotfrid` tool is used to obtain a fractional-order approximation of the compensator. With $q = 0.492$, setting both polynomial orders to 4 and using the Vinagre method, the following fractional-order transfer function is obtained with an error $J = 0.014867$:

$$\hat{G}_c(s) = \frac{0.031325s^{1.968} + 0.30643s^{1.476} + 4.6284s^{0.984} + 4.0234s^{0.492} + 10.0005}{0.0002215s^{1.968} + 0.0021625s^{1.476} + 0.061928s^{0.984} + 0.41302s^{0.492} + 1}.$$

The frequency fitting result is also shown in Figure 6.26a. The open-loop control system frequency response is given in Figure 6.26b. It can be seen, that the desired crossover frequency $\omega_{cg} = 9.94$ and phase margin $\varphi_m = 51.6°$ are very close to specification.



(a) Controller identification result



(b) Control system open-loop frequency response

Figure 6.26: Fractional lead compensator realization

Finally, the step response of the designed control system is given in Figure 6.27.

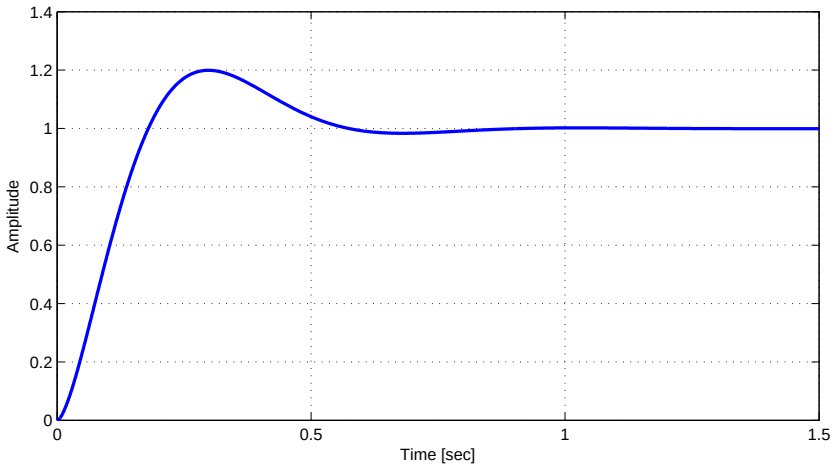


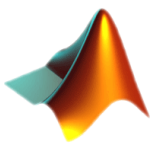
Figure 6.27: Control system with fractional lead compensator step response

6.4 Simulink Blockset

6.4.1 Library Overview

The FOMCON Simulink block library currently consists of 8 blocks and is shown in Figure 6.28.

The library is based on Oustaloup filter approximation as per the `oustapp()` function. The discrete blocks use the Control System toolbox function `c2d()` to obtain the discrete model from the Oustaloup filter LTI system. General block structure is used where applicable.



The blocks are built using Simulink's masking function, making it possible to define an arbitrary set of parameters and assign these to the subsystem blocks. Since custom block face drawing is also supported, it can be used to print useful information on the block, such as the integration/differentiation orders and fractional PID controller type.

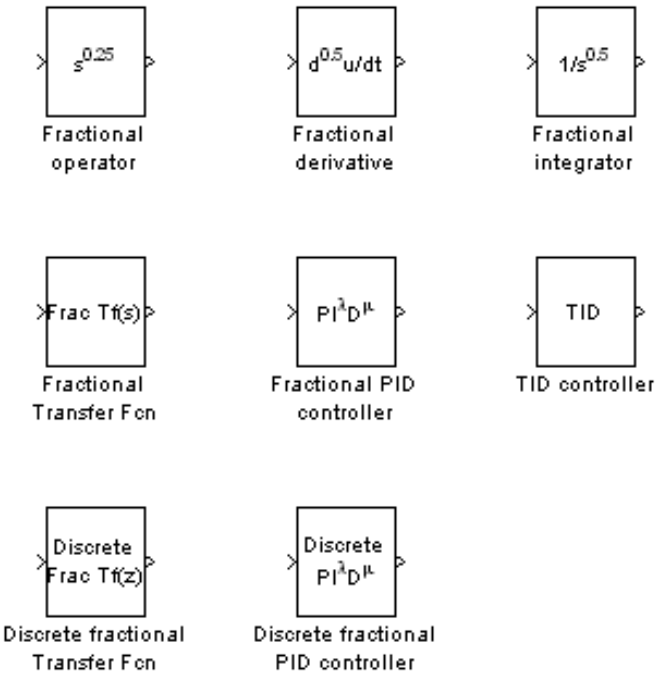


Figure 6.28: FOMCON Simulink library

In order to ensure efficient and accurate simulation, the model build with these blocks may be made up of stiff systems and an appropriate solver should be used in Simulink in such a case (`ode15s` or `ode23tb`).

Further, description for every block is provided.

6.4.2 Block Description

6.4.2.1 Fractional operator

BLOCK PARAMETERS

- *Gain* — block gain,
- *Order* — fractional operator order in range $(-1;\ 1)$,
- *Frequency range* — filter approximation frequency range in form $[\omega_b,\ \omega_h]$ in rad/sec,

- *Approximation order* — filter approximation order.

DESCRIPTION

This block represents a general fractional operator s^γ with $-1 < \gamma < 1$. Oustaloup refined filter is used to obtain the integer-order LTI system approximation which will be used for simulation. Additionally, a low-pass filter is applied internally with crossover frequency of $\frac{1}{\omega_h}$. This is used to prevent algebraic loops in Simulink.

6.4.2.2 Fractional derivative

BLOCK PARAMETERS

- *Derivative order* — fractional derivative order,
- *Frequency range* — filter approximation frequency range in form $[\omega_b, \omega_h]$ in rad.sec,
- *Approximation order* — filter approximation order,
- *Use Oustaloup refined filter* (checkbox) — option to use either the original Oustaloup filter, or the refined one.

DESCRIPTION

This block realizes a fractional derivative s^γ of order $\gamma > 0$, valid in the specified frequency range $[\omega_b, \omega_h]$. This means that for higher orders (such that $\gamma > 1$) an additional zero will be added to the LTI system for every integer order and it will inevitably lead to an improper system (in which there are more zeros than poles). To prevent this, for each added zero, a pole will also be added. This will potentially reduce simulation accuracy near frequency ω_h.

6.4.2.3 Fractional integrator

BLOCK PARAMETERS

- *Integration order* — fractional integrator order,

- *Frequency range* — filter approximation frequency range in form $[\omega_b,\, \omega_h]$ in rad/sec,
- *Approximation order* — filter approximation order,
- *Use refined Oustaloup filter* (checkbox) — option to use either the original Oustaloup filter, or the refined one.

DESCRIPTION

This block realizes a fractional integrator $s^{-\gamma}$ with fractional order $\gamma > 0$, valid in the specified frequency range $[\omega_b,\, \omega_h]$. A low-pass filter is used internally with crossover frequency of $\frac{1}{\omega_h}$ to prevent algebraic loops in Simulink.

6.4.2.4 Fractional Transfer Fcn

BLOCK PARAMETERS

- *Zero polynomial b(s)* — fractional zero polynomial string,
- *Pole polynomial a(s)* — fractional pole polynomial string,
- *Frequency range* — filter approximation frequency range in form $[\omega_b,\, \omega_h]$ in rad/sec,
- *Approximation order* — filter approximation order,
- *Use refined Oustaloup filter* (checkbox) — option to use either the original Oustaloup filter, or the refined one.

DESCRIPTION

This block realizes a fractional transfer function in form $G(s) = \frac{b(s)}{a(s)}$ in the frequency range $[\omega_b,\, \omega_h]$. A low-pass filter is also used in series with the LTI block with crossover frequency of $\frac{1}{\omega_h}$ to prevent algebraic loops in Simulink.

6.4.2.5 Fractional PID controller

BLOCK PARAMETERS

- *Kp*, *Ki*, *lambda*, *Kd*, *mu* — fractional PID parameters,
- *Frequency range* — filter approximation frequency range in form $[\omega_b, \omega_h]$ in rad/sec,
- *Approximation order* — filter approximation order.

DESCRIPTION

This block realizes the $PI^{\lambda}D^{\mu}$ controller in the frequency range $[\omega_b, \omega_h]$. Its structure is given in Figure 6.29. The refined Oustaloup filter is used for fractional operator approximation.

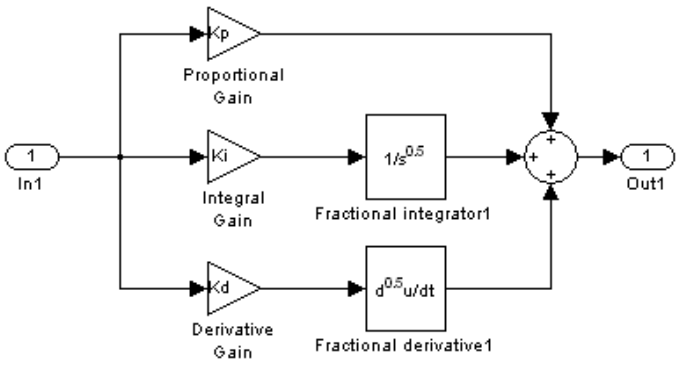


Figure 6.29: *Fractional PID controller* block structure

6.4.2.6 TID controller

BLOCK PARAMETERS

- *K*t, *n*, *K*i, *Kd* — Tilt-Integral-Derivative controller parameters,
- *Frequency range* — filter approximation frequency range in form $[\omega_b, \omega_h]$ in rad/sec,
- *Approximation order* — filter approximation order.

DESCRIPTION

This block realizes the TID (Tilt-Integral-Derivative) fractional controller in the frequency range $[\omega_b, \omega_h]$. Its structure is presented in Figure 6.30. The Tilt component is such that $K_t \cdot s^{-1/n}$, where K_t is the tilt gain and n is the fractional parameter. The refined Oustaloup filter is used for fractional operator approximation.

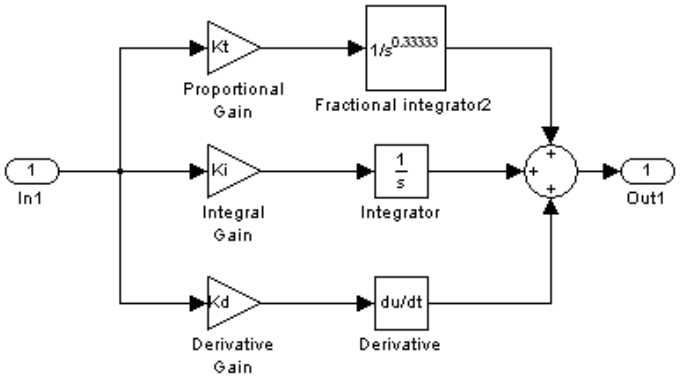


Figure 6.30: *TID controller* block structure

6.4.2.7 Discrete fractional Transfer Fcn

BLOCK PARAMETERS

- *Zero polynomial b(s)* — fractional zero polynomial string,
- *Pole polynomial a(s)* — fractional pole polynomial string,
- *Frequency range* — filter approximation frequency range in form $[\omega_b, \omega_h]$ in rad/sec,
- *Approximation order* — filter approximation order,
- *Use refined Oustaloup filter* (checkbox) — option to use either the original Oustaloup filter, or the refined one.
- *Discretization method* — function `c2d()` discretization method parameter,
- *Critical frequency* — used only with the *Prewarp* method, in rad/sec,
- *Sample time* — discrete sample time in seconds.

DESCRIPTION

This block realizes a discrete approximation of a continuous fractional transfer function given by $G(s) = \frac{b(s)}{a(s)}$ in the frequency range $[\omega_b, \omega_h]$. Discretization may be done by applying the method discussed in Section 2.6 of Chapter 2. This specific method requires to choose the *Prewarp* discretization method and set the *Critical frequency* (plant crossover frequency). Other methods are also applicable.

6.4.2.8 Discrete fractional PID controller

BLOCK PARAMETERS

- *Kp*, *Ki*, *lambda*, *Kd*, *mu* — fractional PID parameters,
- *Frequency range* — filter approximation frequency range in form $[\omega_b, \omega_h]$ in rad/sec,
- *Approximation order* — filter approximation order,
- *Use refined Oustaloup filter* (checkbox) — option to use either the original Oustaloup filter, or the refined one.
- *Discretization method* — function `c2d()` discretization method parameter,
- *Critical frequency* — used only with the *Prewarp* method, in rad/sec,
- *Sample time* — discrete sample time in seconds.

DESCRIPTION

This block realizes a discrete fractional PID controller in the frequency range $[\omega_b, \omega_h]$ using c2d(). The method described in Section 2.6 of Chapter 2 can be applied similarly to the discrete fractional-order transfer function block.

6.4.3 Examples

Example 6.4.1 Consider a non-linear fractional-order differential equation given in [2, 5]:

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

It is very hard to solve this equation analytically. To solve it numerically, a model can be built in Simulink, using the generalized *Fractional operator* block. The equation is first rewritten as

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

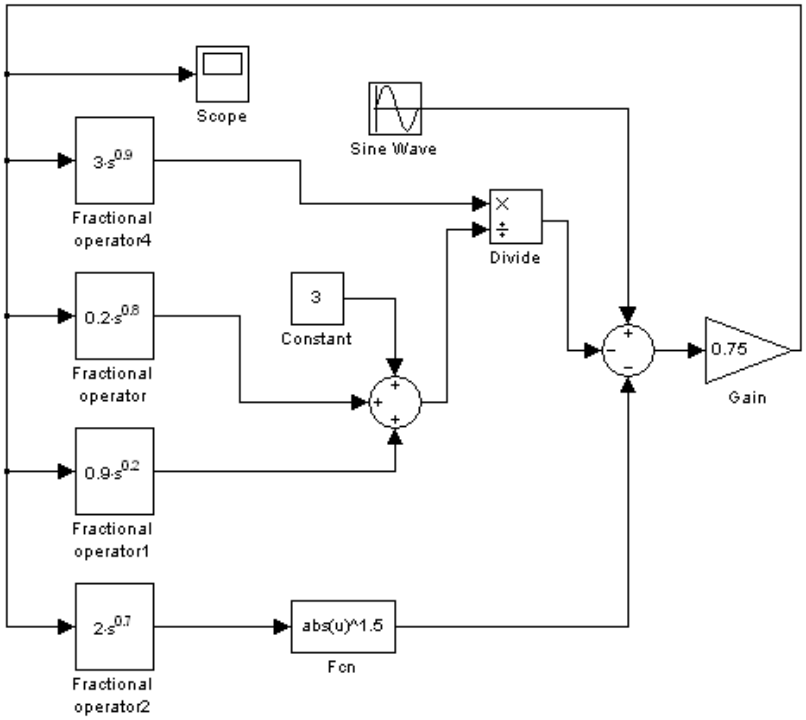
and the corresponding model is constructed in Simulink (Figure 6.31a). For every fractional block Oustaloup filter parameters are set such that $\omega=[10^{-4};\,10^{4}]$, $N=5$. The obtained simulation result, i.e. the differential equation solution is given in Figure 6.31b.

Example 6.4.2 Consider the thermal system, identified previously, and the corresponding fractional-order PID:

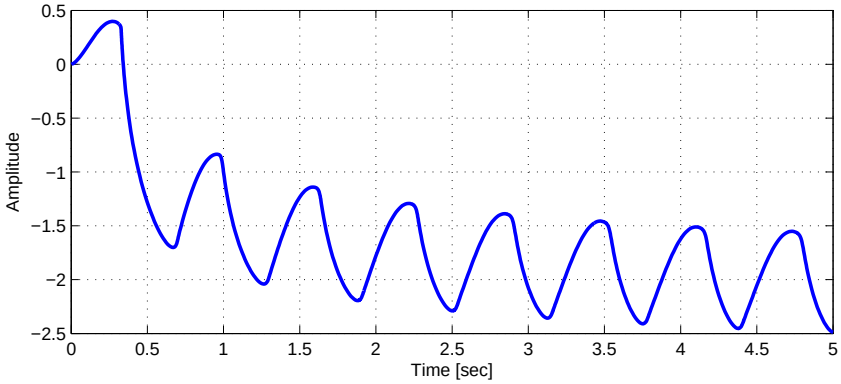
$$\begin{aligned} G(s) &= \frac{1}{2012.409s^{1.8063}+107.2882s^{0.93529}+1.0305}, \\ G_c(s) &= 49.7747+\frac{0.45718}{s^{1.0234}}+204.8202s^{0.53387}. \end{aligned}$$

In this example we will build the continuous-time model of this control system and also observe the behavior of the equivalent discrete-time model.

The obtained model is shown in Figure 6.32a. The set value was set to $SV=150$. The fractional-order transfer function and the PID controller were realized via corresponding blocks with approximation parameters $\omega=[10^{-5};\,10^{5}]$, $N=10$, Oustaloup's refined filter was used in all cases. The *Linear interpolation* method was used for discretization with sample time $T_s=1\,\text{sec}$. The results of the simulation are shown in Figure 6.32b.

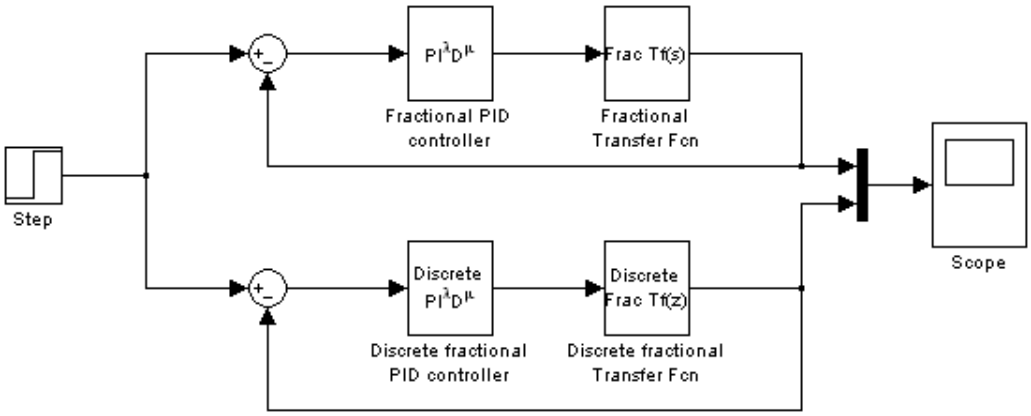


(a) Simulink model

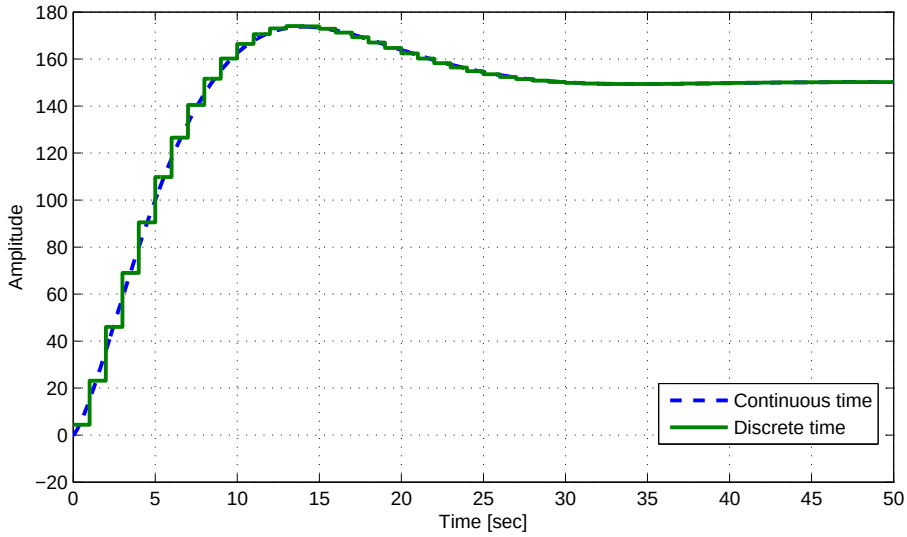


(b) Simulation result

Figure 6.31: Fractional-order differential equation numerical solution



(a) Simulink model



(b) Simulation result

Figure 6.32: Simulation of the thermal control system in continuous and discrete time

Discussion

In this chapter the author presents a brief analysis of the application of fractional calculus in the field of modeling, identification and control. Some advantages and drawbacks of the associated methods used in the FOMCON toolbox are also discussed.

The reason why fractional-order calculus is still absent in elementary mathematical texts is not only due to the controversy in definitions, but also because there is currently no clear interpretation of fractional integration and differentiation in neither geometric nor physical sense. While significant attempts to obtain such interpretations were made [27], they are still somewhat generic and certainly not very intuitive. However, the author strongly believes that with continued research a very powerful computational tool will be obtained. The explanation behind it may not necessarily be trivial.

One of the most apparent problems in fractional-order modeling is the problem of fractional-order system initial conditions which stems from the fact, that the current fractional derivative value depends on all of the past values. This problem cannot be easily solved.

The infinite memory problem is also the cause of modeling issues, in the context of the FOMCON toolbox this involves time-domain simulation of fractional-order systems by using the Grünwald-Letnikov definition. The issue here is that simulation time grows drastically with the number of simulation points. Thus, effective time-domain analysis of fractional-order systems using this method is limited. This also directly impacts system identification from time-domain data. A method should be sought to overcome this problem. An effective finite-memory approach may be required.

On the other hand, it is possible to approximate a fractional-order system with an integer-order one, using one of several available methods. In the context of this work, an effective frequency-domain based solution, called the recursive Oustaloup filter, was introduced. Since MATLAB has several differential equations solvers suitable for par-

ticular cases, the Oustaloup filter approximations can be used for time-domain simulation of fractional-order systems in place of the Grünwald-Letnikov method. However, the Oustaloup filter is frequency limited, which may lead to problems if the frequency range is not carefully chosen, as it has been previously shown. Also, for very complicated fractional-order systems the approximation may become inaccurate. Increasing the filter order may help but it is also limited by the available computing resources. Hence we face the infinite memory problem yet again. The same problem arises with discretization of fractional-order systems.

Some limitations can be found in FOMCON system identification module:

- Due to the aforementioned simulation problem and to some extent to the chosen strategy and identification means the identification process may be slow.
- Global optimal solution with the current best fit strategy may not be obtained using the frequency-domain identification tool.
- There is little control over the accuracy of the optimized fractional operator orders. Generally, identification results in non-commensurate systems. This holds true for both the time-domain and frequency-domain identification methods.

The latter problem cannot be solved by simply trimming or rounding orders with given precision. Model transformations are sometimes applicable, however a solid method of post-identification model fitting is still to be developed. Other limitations will be gradually eliminated.

In the FOMCON system control module there are several limitations:

- Limited design specifications in fractional-order PID tuning, limited control over the exact specification fullfillment requirement.
- Global optimal solution may not be found using the fractional-order PID optimization technique.
- No direct fractional lead-lag compensator design method.
- No automatic TID controller tuning/optimization method.

However, all these limitations are temporary and will be eventually overcome in the upcoming versions of the toolbox.

As for the Simulink toolbox, although it allows for a more sophisticated modeling approach, the block realization is currently limited to the frequency-bound Oustaloup filter approximation. While it has been shown, that this method is highly effective for modeling fractional-order systems, there are still issues, such as possible algebraic loops in Simulink and the proper LTI model simulation requirement, which need to be addressed. Also, more suitable discrete blocks need to be introduced.

Otherwise, it has been shown that fractional-order calculus offers several advantages in system control. Moreover, the author believes that the introduction of fractional-order calculus to automatic control systems in particular is a required step because it is evident, and conveyed throughout this paper, that the generalization of the derivative order to the fractional case will lead to more accurate models, advanced system identification techniques and development of efficient control strategies.

Conclusions

In this chapter the author presents an overview of the studied problems, achieved results and further topic development perspectives.

In this thesis the reader was introduced to methods of dynamic system modeling, identification and control in the context of fractional-order calculus. FOMCON, a new MATLAB fractional toolbox, in which these methods have been implemented, was presented and discussed. The examples considered in this thesis were limited to relatively simple models. However, there is no reason to doubt that proposed methods will be effective for much more complicated cases, albeit at an inevitable expense of processing speed. Further a list of advantages and drawbacks for each module is provided.

System analysis module

- Advantages
    - Fully-featured for fractional-order transfer function analysis.
    - Provides convenience methods, e.g. the string parser, and a graphical user interface for an effective workflow.
    - Has means of system export to other toolbox formats, e.g. to transfer the work to the CRONE toolbox.
    - Tightly integrated with the System Control toolbox which means highly effective algorithms are used for data processing.
- Drawbacks
    - Limited to the single input-single output linear time-invariant dynamic system case.

- Currently has no support for direct fractional-order state-space system analysis (although offers the ability to convert fractional-order transfer functions to their state-space represantation).
- There is no support for fractional-order system initial conditions.
- Time-domain simulation may require a lot of computational resources.
- No proper discretization method is implemented.
- Stability analysis has a fixed precision to solve the associated computational effort problem.

Identification module

- Advantages
  - Offers more accurate identification than the integer-order identification.
  - Encompasses both time-domain and frequency-domain identification methods.
  - Provides GUIs for all identification tools facilitating the initial guess model design and identification process.
  - Provides a vast array of options to achieve the best possible results with the proposed methods.
- Drawbacks
  - Time-domain identification may be slow and require a lot of computational resources.
  - No signal filtering feature is currently available, signal pre-filtering should be done by the user.
  - A globally optimal solution may not be found using the best fit tool of the frequency-domain identification.
  - Control over free identification in the time domain is limited.

Control module

- Advantages

- Offers methods for fracional-order PID controller design, tuning and optimization according to given specifications and with a vast array of options.
- A GUI is available for each feature.
- Controller optimization can handle fractional-order and integer-order systems.
- Controller optimization does not require the Optimization toolbox.
- Has methods of implementing of the TID controller and fractional lead-lag compensator.

- Drawbacks
  - Controller optimization has limited support of performance specifications.
  - No analytic tuning method for the fractional PID controller.
  - Limited support for integer-order PID tuning methods.
  - No TID compensator optimization.
  - No direct method for realization of the fractional lead-lag compensator.
  - No automatic tuning method for the fractional lead-lag compensator.

Simulink blockset

- Advantages
  - Provides sophisticated methods for fractional-order system modeling.
  - Offers a complete library featuring means of system analysis in discrete time by continuous-time model conversion.
  - General block structure is applied where possible.

- Drawbacks
  - Use of Oustaloup approximation of fractional operators with higher orders in transfer function and diffrentiator blocks may lead to improper systems, and the applied workaround may reduce accuracy near the high frequency bound.

- Simulation may be slow so that a stiff systems solver needs to be used at a potential expense of accuracy.
- The transfer function blocks only allow setting the fractional-order polynomials via text strings.

Research perspectives

With the above considerations, further research directions, involving the development of the FOMCON toolbox, can be outlined:

- Examine the possibility for fractional-order MIMO system modeling.
- Implement a tool for working with fractional-order state-space models.
- Search for a way to effectively implement the initial conditions problem.
- Implement more time-domain analysis methods, especially considering the need for an effective simulation method suitable for obtaining accurate responses on a larger time interval.
- Improve stability analysis precision for systems with a very low commensurate-order.
- Research methods for converting a high-order interger-order model to a fractional one possibly using frequency fitting techniques, e.g. converting an Oustaloup filter approximation back to the original model.
- Research and implement discrete fractional-order systems, expand the corresponding Simulink feature set.
- Research more effective time-domain identification methods, implement suitable signal filtering methods.
- Research the contribution of noise and disturbances to the identified model.
- Develop a method for obtaining a globally optimal solution to the frequency-domain identification problem.
- Develop a more flexible frequency-domain identification method, involving fitting arbitrary orders, i.e. not limited to a particular commensurate order of the polynomials.

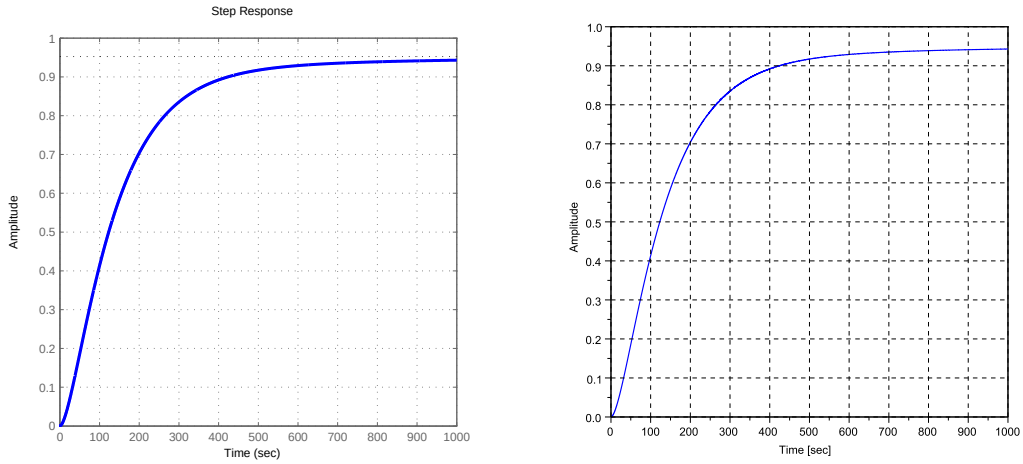
- Obtain a method for post-identification model fitting to decrease fractional operator orders precision requirement maintaining the accuracy of the identified system.
- Develop identification strategies per identified object class.
- Implement more performance specifications in fractional-order PID controller optimization.
- Research methods for analytic fractional PID tuning.
- Implement Tilt-Integral-Derivative controller tuning and optimization based on performance specifications.
- Implement fractional lead-lag compensator direct realization and tuning based on performance specifications.
- For all controller types, research auto-tuning methods and strategies.
- Make more general blocks for the Simulink library, implement other types of simulation, i.e. Grünwald-Letnikov; research possibilities for improving simulation efficiency with non-stiff system solvers.

Porting the FOMCON toolbox to another computing platform should also be considered for the sake of exposure and to expand the range of potential users. One of the obvious choices is *Scilab*, becuase it is open-source and also contains a Simulink-like environment. A research has been previously conducted by the author addressing possibilities for fractional-order modeling implementation in Scilab. The following problems were discovered:

- Scilab ODE solvers are somewhat limited where simulation of high-order systems is concerned.
- There is currently no direct support for object-oriented programming.
- There are no sophisticated tools for building graphical user interfaces for Scilab and general support for GUIs is limited.

The lack for support of object-oriented programming can be partially overcome by using a Scilab-specific data structure which also allows for method overloading. However, the need for an effective FODE computation algorithm and a suitable GUI design tool may lead to initiation of more Scilab related projects.

For now a small set of functions was realized for Scilab. Consider the comparison of the step response of the thermal system $G(s) = \frac{1}{2012.409s^{1.8063}+107.2882s^{0.93529}+1.0305}$ in MATLAB and Scilab given in Figure 6.33. It can be seen, that the same exact result was obtained. It should be noted, that the Oustaloup filter was used in both cases. In Scilab, the only way to simulate the recursive filter was to obtain its state-space representation. This issue should also be considered.



(a) Step response in MATLAB

(b) Step response in Scilab

Figure 6.33: Thermal system simulation in MATLAB and Scilab

The following topics are also considered to be important research directions.

- Fractional-order non-linear systems analysis.
- Fractional-order PID controller, TID controller and lead-lag compensator hardware implementation.
- Research of fractional-order calculus in artificial neural networks.
- Research of classical PID controller structures, i.e. the Smith Predictor, generalized to the fractional case.

Concluding comments

This thesis has illustrated the use of fractional-order calculus in dynamic system modeling, identification and control. The benefits stemming from applying fractional-order calculus to the problems of automatic control were made evident.

It can be concluded, that fractional-order calculus is a necessary generalization. Although our current mathematical tools in this field are somewhat limited, and even obtaining a numerical solution for the fractional-order derivatives may be tedious, it is expected that this topic will gain more attention in the coming years, efficient computational and analytical methods will be developed and the use of non-integer calculus will become standard practice.

Bibliography

[1] K.S. Miller, B. Ross, *An Introduction to the Fractional Calculus and Fractional Differential Equations*, Wiley, New York, 1993.

[2] C.A. Monje, Y.Q. Chen, B.M. Vinagre, D. Xue, V. Feliu, *Fractional-order Systems and Controls Fundamentals and Applications*, Springer-Verlag, London, 2010.

[3] Y.Q. Chen, I. Petráš, D. Xue, *Fractional Order Control - A Tutorial*, American Control Conference, pp. 1397-1411, 2009.

[4] I. Podlubny, *Fractional Differential Equations*, Academic Press, San Diego, CA, 1999.

[5] D. Xue, Y.Q. Chen, D.P. Atherton, *Linear Feedback Control Analysis and Design with MATLAB*, Advances in Design and Control, Siam, 2007.

[6] R. Hilfer, *Applications of Fractional Calculus in Physics*, World Scientific, Singapore, 2000.

[7] A. Oustaloup, P. Melchior, P. Lanusse, O. Cois, F. Dancla, *The CRONE toolbox for Matlab*, Computer-Aided Control System Design, pp. 190-195, 2000.

[8] D. Valério, *Toolbox ninteger for MatLab, v. 2.3*, [online] `http://web.ist.utl.pt/duarte.valerio/ninteger/ninteger.htm`, 2005.

[9] D. Das, *Functional Fractional Calculus for System Identification and Controls*, Springer, Berlin, 2008.

[10] A.A. Kilbas, H.M. Srivastava, J.J. Trujillo, *Theory and Applications of Fractional Differential Equations*, Elsevier, Amsterdam, 2006.

[11] D. Matignon, *Generalized Fractional Differential and Difference Equations: Stability Properties and Modeling Issues*, Proc. of Math. Theory of Networks and Systems Symposium, Padova, Italy, 1998.

[12] B.M. Vinagre, I. Podlubny, A. Hernández, V. Feliu, *Some Approximations of Fractional Order Operators Used in Control Theory and Applications*, [online] `http://mechatronics.ece.usu.edu/foc/cdc02tw/cdrom/Lectures/Lecture4/approx.pdf`.

[13] A. Oustaloup, F. Levron, B. Mathieu, F. Nanot, *Frequency-Band Complex Noninteger Differentiator: Characterization and Synthesis*. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, 2000, 47(1):25–40.

[14] L. Ljung, *System Identification Theory for the User (Second Edition)*, Prentice Hall, New Jersey, 1999.

[15] R. Malti, M. Aoun, J. Sabatier, A. Oustaloup, *Tutorial On System Identification Using Fractional Differentiation Models*, 14th IFAC Symposium on System Identification, Newcastle, Australia, pp. 606-611, 2006.

[16] T. Hartley, C. Lorenzo, *Fractional-order System Identification Based on Continuous-order Distributions*, Signal Processing, 83, pp. 2287-2300, 2003.

[17] D. Valério, J. Costa, *Levy's Identification Method Extended To Commensurate Fractional Order Transfer Functions*, EUROMECH Nonlinear Dynamics conference, Eindhoven, Netherlands, 2005.

[18] I. Podlubny, *Fractional-order systems and $PI^{\lambda}D^{\mu}$-controllers*, IEEE Trans. on Automatic Control, vol. 44, pp. 208-214, 1999.

[19] Y. Luo, Y.Q. Chen, *Fractional-order [proportional derivative] controller for robust motion control: Tuning procedure and validation*, American Control Conference, pp. 1412-1417, 2009.

[20] M. Čech, M. Schlegel, *The fractional-order PID controller outperforms the classical one*, Process control 2006, Pardubice Technical University, pp. 1-6, 2006.

[21] D. Xue, Y.Q. Chen, *A Comparative Introduction of Four Fractional Order Controllers*, Proceedings of the 4th World Congress on Intelligent Control and Automation, Shanghai, P.R. China, pp. 3228-3235, 2002.

[22] I. Podlubny, L. Dorcak, I. Kostial, *On Fractional Derivatives, Fractional-order Dynamic Systems and $PI^{\lambda}D^{\mu}$-controllers*, Proc. of the 36th IEEE CDC, San Diego, 1999.

[23] C.N. Zhao, D.Y. Xue, Y.Q. Chen, *A fractional order PID tuning algorithm for a class of fractional order plants*, Proceedings of the International Conference on Mechatronics and Automation. Niagara, Canada, pp. 216-221, 2005.

[24] C. Yeroglu, C. Onat, N. Tan, *A new tuning method for $PI^{\lambda}D^{\mu}$ controller*, Electrical and Electronics Engineering, ELECO 2009, pp. II-312-II-316, 2009.

[25] B.J. Lurie, *Three-parameter tunable tilt-integral-derivative (TID) controller*, US Patent US5371670, 1994.

[26] R. Oldenhuis, *Optimize*, MathWorks File Exchange, [online] `http://www.mathworks.com/matlabcentral/fileexchange/24298-optimize`, 2009.

[27] I. Podlubny, *Geometric and Physical Interpretation of Fractional Integration and Fractional Differentiation, Fractional Calculus and Applied Analysis*, vol. 5, no. 4, pp. 367-386, 2002.

List of Publications

1. A. Tepljakov, E. Petlenkov and J. Belikov, *FOMCON: Fractional-order Modeling and Control Toolbox for MATLAB*, MIXDES 2011, Gliwice, Poland, *Accepted for presentation*.