TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technology
Department of Computer Control

Tobechukwu Onyedi 172954IASM

FOMCONPY: A PYTHON LIBRARY FOR FRACTIONAL-ORDER MODELLING AND CONTROL WITH APPLICATIONS TO DISTRIBUTED CONTROL SYSTEMS

Master Thesis

Supervisor
Aleksei Tepljakov
PhD

Tallinn 2020

Declaration/ Deklaratsioon

I hereby certify that I am the sole author of this master's thesis. Investigations and achievements submitted for the master's degree at Tallinn University of Technology has not been submitted anywhere else for a degree. All materials and references used have been referred.

Käesolevaga kinnitan, et esitatud töö *FOMCONpy: Pythoni teek murruliste modelleerimise ja juhtimise jaoks koos hajutatud juhtimissüsteemide rakendusega* on minu isikliku töö tulemus. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud. Käesolevat tööd ei ole varem esitatud kusagil mujal.

Author/Autor: Tobechukwu Onyedi .....

(Signature/Allkiri)

Date/Kuupäev: May 10, 2020

Abstract

FOMCONpy: A Python library for Fractional-order Modelling and Control with Applications to Distributed Control Systems

This thesis is devoted to the application of fractional-order calculus to modelling, identification and control of dynamic systems in the domain of Internet of Things (IoT) specifically targeting distributed control systems (DCS). The first part of the thesis introduces the reader to the concepts of fractional-order modelling and control. In the second part, the FOMCONpy ("Fractional-order Modelling and Control for Python") developed by the author is presented in the form of an expository manual with demo examples showcasing the functionality of the toolbox.

All mathematical calculations and simulations in this thesis are done using JetBrains' PyCharm IDE and Anaconda's IDE Python 3.7.4 interpreter on Windows 10 operating system. To facilitate real-world use of the introduced methods and algorithms, the functionality of the toolbox is verified on embedded platforms – represented by the popular Raspberry PI 3B and 4 devices – using Thonny Python IDE to meet the needs of IoT domain.

As a result of this work, a Python library for fractional-order modelling and control is introduced which is also capable of operating on low-cost IoT hardware. The biggest advantage of this implementation of FOMCONpy library is its inclusion into the ever growing Python software ecosystem. It is envisioned that this will allow a more rapid dissemination of relevant results across related industrial domains stemming from reported successful applications of fractional calculus to real-life problems.

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

Acknowledgements

This work is dedicated to my dad, late Mr. Ephraim Iheanacho Onyedi who motivated me to begin the journey towards earning a Master's degree. I wish you were here to see your son's achievement, but I know you're celebrating my successes, where you are.

I will like to appreciate my supervisor, Dr. Aleksei Tepljakov, for his supervision, advice, support and technical explanations throughout the course of this work. It was an absolute pleasure working with you.

A very big thank you to my mum, Mrs. Juliana Uzoyibo Onyedi for her love, care and continual words of blessing. To my siblings for their continual words of encouragement. Ifeoma Onyedi, Dr. Chukwuemeka Emmanuel Onyedi, Ifeyinwa Janet Onyedi, Chidimma Emuh, Rotanna Onyedi and Chukwuma Innocent Onyedi.

Sincere appreciation to Peter Osuere for his words, guidance and advice about the field of Robotics, Control and Process automation.

This work is partially based upon works from COST Action CA15225, a network supported by COST (European Cooperation in Science and Technology) and the Estonian Research Council grant PRG658. The research was also partly supported by a teaching grant awarded to Dr. Aleksei Tepljakov in 2019 by the School of Information Technologies, Tallinn University of Technology.

List of abbreviations and terms

OS	Operating System
GUI	Graphical User Interface
FOMCON	Fractional Order Modelling and Control
FOMCONpy	Fractional Order Modelling and Control for Python
IoT	Internet of Things
DCS	Distributed Control System
PID	Proportional Integral Differential
FO-FOPDT	Fractional-order First Order Plus Dead Time
FOPID	Fractional-order Proportional Integral and Differential
LM	Levenberg Marquardt
TRR	Trust Region Reflective
RFID	Radio Frequency Identification
FRFT	Fractional Fourier Transform
VST	Variable Speed Turbine
LTI	Linear Time Invariant
BESS	Battery Energy Storage System
RBF	Radial Basis Function
AI	Artificial Intelligence
NN	Neural Networks
APF	Active Power Filter
CAS	Computer Algebra Systems
CT	Computational Tools
IDE	Integrated Development Environment
IIR	Infinite Impulse Response
FIR	Finite Impulse Response
MIMO	Multiple Input, Multiple Output
SISO	Single Input, Single Output
FOSAM	Fractional-order System Analysis Module
FOSIM	Fractional-order System Identification Module
FOSCOM	Fractional-order System Control Module
WLAN	Wireless Local Area Network

Table of Contents

List of Figures

List of Tables

Listings

1. Introduction

In this chapter, the reader is introduced briefly to fractional calculus, its applications and the potential opportunity in the IoT domain which is the main motivation and contribution of the writer.

1.1 Literature Survey

The fractional-order calculus concept has been known since the development of classical (integer-order) calculus with reference being associated with L'Hospital and Leibniz in 1695 where half-order derivative was mentioned [1]. Since then, fractional-order calculus has been a subject of subtly active discussion [2] though the concept still tingle in the ears of many scholars perhaps because it could be mathematically more complex compared to its integer-order counterpart.

In the last three centuries, the mathematical theory has been established [3]–[9]. Three hundred years will be quite a lot of time for a concept to be based on theory alone as fractional-order calculus has been used in complex mathematical and physical challenges [10]. It has been used in signal processing and semiconductor manufacturing [11]. RFID tag anti-collision algorithm based on Fractional Fourier Transform (FRFT) to support high-efficiency reading of multiple chipless tags [12]. Meghni et al proposed a robust adaptive fractional-order controller could serve as a generic supervisory control scheme for a variable speed turbine (VST) and battery energy storage system (BESS) to increase efficiency and solve instability and intermittent problems of wind power turbines. In other-words, fractional-order controller was used as a controller for other controllers [13]. In all the above references, fractional-order systems have been found to be more efficient and robust than classical-order systems [14], however it is seldomly used in the industries

because of higher computational complexity and financial cost compared to integer-order [15]. it is the authors goal to proffer a solution to minimizing the possible computational and financial cost in this thesis work.

These days, researchers already combined fractional-calculus with other control algorithm to proffer alternative solutions to challenges [16]. in artificial intelligence (AI) domain, fractional-order calculus is combined with dual radial basis function (RBF) neural networks (NNs) to enhance the performance of a three-phase shunt active power filter (APF) themed *Adaptive Fractional Sliding Mode Control* [17]. Genetic algorithm combined with fractional-order calculus was used to tune a Fractional-order PID controller in [18].

It is almost very obvious, that the interest of researchers in fractional-calculus was greatly promoted by the availability of more efficient computer algebra systems (CAS) and computational tools (CT) like *MATLAB, LabView*, and in recent time, *Python*. Several toolboxes for fractional-order systems exist in *MATLAB*. For example, the *FOTF* toolbox [19], the *NINTEGER* toolbox [20], The *CRONE* toolbox and the *FOMCON* toolbox [14], [21].

1.2 Problem Statement

Developing with or using MATLAB toolboxes like those mentioned in Sec. 1.1 require multiple licenses. The user must have at least a Control Toolbox license and Optimization Toolbox licence [14] excluding other licenses. These licenses cost a premium and without a campus-wide subscription, it is costly to use these facilities.

Internet of Things (IoT) is one of the next big thing in technology with various types of commercial devices: trackers, wearables, controllers, meters and sensors in different industries, including utilities, automotive, transport, logistics, agriculture, manufacturing, healthcare, warehousing and mining all connected to the cloud [22]. Blue chip companies like Microsoft and Ericsson invested and forecast respectively, that IoT will be worth

trillions of dollars by 2025 [23], [24]. These facts and forecasts prompted the development of this library and to the best of the author's knowledge, there is no toolbox that provides fractional calculus modelling and control tools for the *Python* ecosystem and IoT community that targets novel implementations of Distributed Control Systems (DCS).

With each new releases of MATLAB versions, there is the tendency that function in previous version becomes obsolete in newer version [14], thus the problem of continuous re-scrutiny and refactoring of previously written MATLAB functions and codes to be compatible with each new release of MATLAB software that happens twice a year. In contrast, the *Python* ecosystem has been proven to be more robust and stable.

1.3 Purpose

As mentioned in Section 1.1, *Python* programming language has a contributed control systems package available. It is easier to learn compared to other programming languages. Its more generic than *MATLAB* which from the onset was a matrix manipulation package that was modified later by adding programming language to it [25]. Unlike *MATLAB*, one amazing features of *Python*, is the object oriented programming availability and it is open-source thus its greatly increase in popularity.

Based on these considerations, the author is strongly convinced that having a fractional-order system identification, modelling and control tool in *Python* will facilitate the usage of fractional-order systems in the industries. Thus this thesis is devoted to the study of possibilities provided by fractional-order calculus in system theory, modelling, identification and control. It discusses a new library *FOMCONpy* that analyzes, identifies and controls fractional-order dynamic systems. *FOMCONpy* is written in *Python* programming language (which to the best of the authors knowledge is the first of its kind with support for IoT devices). The library contains various utilities and a graphical user interface (GUI) for quick adaptation to both new users and those already using its *MATLAB* counterpart *FOMCON*.

1.4 Outline of the Thesis

It is suggested to the reader to go through this thesis the way it has been written without skipping a chapter as the theory of fractional-order calculus can be somewhat complex especially if this is the first time to encounter the subject. The thesis has the following structure:

In Chapter 2, the reader is presented with an overview of the theoretical concepts of fractional-order calculus, its approximations to classical-order and application to system theory.

In Chapter 3, methods and algorithms used in *FOMCONpy* for time domain identification of fractional-order model are discussed.

Chapter 4 introduces the reader to fractional-order control.

In Chapter 5, the reader is presented a setup and manual guide for the *FOMCONpy* library, its relation to other *Python* libraries and recommended integrated development environment (IDE) for both Windows and Linux (IoT) users. Major functions of the library and multiple demos are given as examples with command line and GUI scenarios.

Lastly, in Chapter 6, conclusions are drawn and prospects for further development of the library are stated.

2. Introduction to Fractional calculus

In this chapter we shall discuss the basics of fractional-order calculus, its applications in system theory and approximation to classical-order calculus. The chapter outline is 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 a method for approximating fractional-order model to integer-order model which was proposed by Oustaloop [26]. Section 2.6 presents some aspects of discrete fractional system. Finally, in Section 2.7 we discuss some optimization algorithms used in this work for the reader to understand the theory in the identification process.

2.1 Basis of Fractional-Order Calculus

The non-integer derivative and integral function can be represented with the symbol ${}_a\mathscr{D}_t^\alpha$, is often used as a mathematical operator for fractional order calculus, where a and t represents the start and stop limits of the operation respectively. if $\alpha > 0$, $\mathscr{D}$ is a derivative operator. if $\alpha < 0$, $\mathscr{D}$ is an integral operator.

$$ {}_a\mathscr{D}_t^\alpha = \begin{cases} \frac{\mathrm{d}^\alpha}{\mathrm{d}t^\alpha} & \Re(\alpha) > 0 \\ 1 & \Re(\alpha) = 0 \\ \int_a^t (\mathrm{d}t)^{-\alpha} & \Re(\alpha) < 0 \end{cases} \tag{2.1} $$

α is assumed to be a real number, but it can also be a complex number [1] however, in this thesis, only $\alpha \in \mathbb{R}$ is considered.

2.1.1 Fractional Calculus Definitions

There are 3 definitions of fractional-order derivative discussed in this work. They are defined by Riemann-Liouville, Caputo and Grinwald-Lenikov [9]. They are summarized below:

Definition 2.1.1.1 *Riemann-Liouville's Definition*

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

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

Definition 2.1.1.2 *Grünwald-Letnikov's Derivative Definition*

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

where

$$ \binom{\alpha}{k} = \frac{n(n-1)(n-2)\ldots(n-k+1)}{k!} \tag{2.4} $$

and $\left|\frac{t-a}{h}\right|$ *is the integer part.*

Grundwald-Letnikov definition is suggested to be very useful in fractional-order system application [9].

Definition 2.1.1.3 *Caputo's Definition*

$$ {}_a\mathscr{D}_t^{-\alpha} f(t) = \frac{1}{\Gamma(m-\alpha)} \int_a^t \frac{f(\tau)}{(t-\tau)^{\alpha-m+1}} \mathrm{d}\tau, \tag{2.5} $$

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

Caputo's definition is considered to be useful if assigning initial conditions [27].

2.1.2 Properties of Fractional-Order operator

Properties of fractional-order operator ${}_a\mathscr{D}_t^{\alpha} f(x)$ can be found in [1], [9], [28] and are as follows for $a=0$:

- If $f(x)$ is an analytic function of variable x, then ${}_a\mathscr{D}_t^{\alpha} f(x)$ is also an analytical function of x and α.
- The fractional operator ${}_a\mathscr{D}_t^{\alpha}$ and the classical derivative operator ${}_a\frac{d^n}{dx^n}$, $n\in\mathbb{Z}^+$ are equivalent if $\alpha=n$.
- The fractional operator ${}_a\mathscr{D}_t^{\alpha}$ and the classical integral of order $n\in\mathbb{Z}^-$ are equivalent if $\alpha=-n$.
- For the fractional-order operator ${}_a\mathscr{D}_t^{\alpha} f(x)=f(x)$, when $\alpha=0$. This is called the identity operator.
- ${}_a\mathscr{D}_t^{\alpha} f(x)$ and its first $(n-1)$th-order derivatives must vanish to zero at $t=x$.
- The fractional-order operator is linear:

$$U\cdot{}_a\mathscr{D}_t^{\alpha} f(x)+V\cdot{}_a\mathscr{D}_t^{\alpha} g(x)={}_a\mathscr{D}_t^{\alpha}\Big[U\cdot f(x)+V\cdot g(x)\Big]. \tag{2.6}$$

- Fractional-order operators with $\mathbb{R}(\alpha)>0$, $\mathbb{R}(\beta)>0$, holds the additive law of exponents with the following condition:

$${}_a\mathscr{D}_t^{\alpha}\,{}_a\mathscr{D}_t^{\beta} f(x)={}_a\mathscr{D}_t^{(\alpha+\beta)} f(x). \tag{2.7}$$

2.2 Laplace Transform of Fractional-Order Calculus

A fundamental mathematical entity in system and control engineering is the Laplace integral transform. A function $f(t)$ can be transformed to the function $F(s)$ of a complex variable s. It is defined as:

$$F(s) = \mathscr{L}[f(t)] = \int_0^{\infty} e^{-st} f(t) \mathrm{d}t. \tag{2.8}$$

The Laplace transform $F(s)$ can be reversed to its initial function $f(t)$ by applying the reverse Laplace transform defined as:

$$f(t) = \mathscr{L}^{-1}[F(s)] = \frac{1}{j2\pi} \int_{c-j\infty}^{c+j\infty} F(s) \mathrm{d}s. \tag{2.9}$$

Let us define the Laplace transform for fractional calculus definitions in Section 2.1.1

Definition 2.2.1 *Laplace transform of Riemann-Liouville's fractional operator*

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

Definition 2.2.2 *Laplace transform of Grundwald-Letnikov's fractional operator*

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

Definition 2.2.3 *Laplace transform of Caputo's fractional operator*

$$\mathscr{L}[\mathscr{D}^{\alpha} f(t)] = s^{\alpha} F(s) - \sum_{k=0}^{m=1} s^{\alpha-k-1} f^{k}(0), \text{ where } (m-1 \leqslant \alpha < m). \tag{2.12}$$

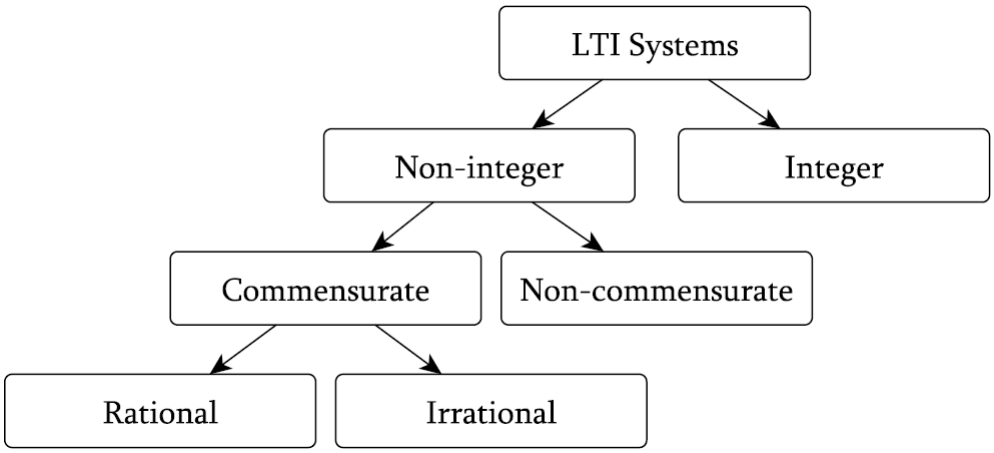


Figure 1. Classification of LTI Systems [21]

The Laplace transform is popularly used in system modelling and control and same was used in this thesis.

2.3 Fractional Order System Model

A dynamic continuous-time system can be explicitly represented using a fractional differential equation as below [1], [9].

$$\begin{aligned} &a_n \mathscr{D}^{\alpha_n} y(t) + a_{n-1} \mathscr{D}^{\alpha_{n-1}} y(t) + \cdots + a_1 \mathscr{D}^{\alpha_1} y(t) + a_0 \mathscr{D}^{\alpha_0} y(t) = \\ &b_m \mathscr{D}^{\beta_m} u(t) + b_{m-1} \mathscr{D}^{\beta_{m-1}} u(t) + \cdots + b_1 \mathscr{D}^{\beta_1} u(t) + b_0 \mathscr{D}^{\beta_0} u(t) \end{aligned} \tag{2.13}$$

A scenario where an integer base order $\gamma \in \mathbb{R}^+$ exist, such that $\alpha_k, \quad \beta_k = k \cdot \gamma$. The system is said to be of commensurate order. An expression of a commensurate order system is equation (2.14)

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

The system (2.14) will be of rational order if $\gamma = \frac{1}{q}$, $q \in \mathbb{Z}^+$. See Figure 1 for hierarchical classification of rational order in Linear Time Invariant (LTI) systems. Applying the Grundwald-letnikov's Laplace transform formula stated in equation (2.11) to (2.13) we

obtain the following,

$$\begin{aligned} a_n s^{\alpha_n} Y(s) + a_{n-1} s^{\alpha_{n-1}} Y(s) + \cdots + a_1 s^{\alpha_1} Y(s) + a_0 s^{\alpha_0} Y(s) = \\ b_m s^{\beta_m} U(s) + b_{n-1} s^{\beta_{n-1}} U(s) + \cdots + b_1 s^{\beta} U(s) + b_0 s^{\beta_0} U(s) \end{aligned} \tag{2.15}$$

applying the commutative property of the fractional derivative operator discussed in Section 2.1.2,

$$\begin{aligned} Y(s)\left[a_n s^{\alpha_n} + a_{n-1} s^{\alpha_{n-1}} + \cdots + a_1 s^{\alpha_1} + a_0 s^{\alpha_0}\right] = \\ U(s)\left[b_m s^{\beta_m} + b_{n-1} s^{\beta_{n-1}} + \cdots + b_1 s^{\beta} + b_0 s^{\beta_0}\right] \end{aligned} \tag{2.16}$$

from equation (2.16), a transfer function for fractional-order system can be generated.

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_m s^{\beta_m} + b_{m-1} s^{\beta_{m-1}} + \cdots + b_1 s^{\beta} + b_0 s^{\beta_0}}{a_n s^{\alpha_n} + a_{n-1} s^{\alpha_{n-1}} + \cdots + a_1 s^{\alpha_1} + a_0 s^{\alpha_0}} \tag{2.17}$$

Recall the commensurate order γ fractional-order system, discussed in equation (2.14), it can also be expressed as:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{\sum_{k=0}^{m} b_k (s^{\gamma})^k}{\sum_{k=0}^{n} a_k (s^{\gamma})^k} \tag{2.18}$$

Taking $\Lambda = s^{\gamma}$ the function (2.18) can be viewed as a *pseudo-rational* transfer function $H(\Lambda)$:

$$H(\Lambda) = \frac{\sum_{k=0}^{m} b_k \Lambda^k}{\sum_{k=0}^{n} a_k \Lambda^k} \tag{2.19}$$

Based on this pseudo-rational function concept, a state-space representation can be formed which allows the representation of multiple input, multiple output (MIMO) fractional-order systems [21] but the state-space model is beyond the scope of this work. In fact, with the transfer function representation of system dynamics we can still achieve meaningful results even when modelling multivariate systems since we can also use arrays of transfer functions in that case.

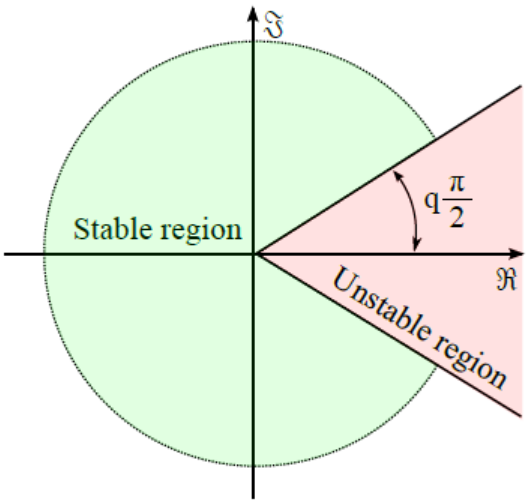


Figure 2. Stability region of LTI fractional-order system for range: $0 < q < 1$ [21]

2.4 Fractional System Analysis

In this section we shall discuss some key concepts used in the analysis of a fractional system. These are in no way different from convectional analysis of classical systems, however the approach is more complex. We shall discuss concepts like Stability, time response and frequency response.

2.4.1 Stability

Stability has always been a key-point in the analysis of classical-order systems. So for a fractional-order system given by the fractional-order transfer function (2.14), we also do same stability analysis. Let us review the following theorem found in [1], [9].

Theorem 2.4.1 *Matignon's Stability Theorem According to Matignon, a fractional-order transfer function* $G(s) = Z(s)/P(s)$ *is stable only if:*

$$|arg(\sigma)| > q\frac{\pi}{2}, \forall\ \sigma \in C,\ P(\sigma) = 0,\ \textit{where}\ \sigma := s^q \tag{2.20}$$

When $\sigma = 0$, it indicates a single root of $P(s)$, thus the system cannot be stable. When $q = 1$, this is same as the classical case (that says pole roots should not be at the right-half plane of the Riemann sheet). The algorithm for checking the stability of the system in (2.17) can be summarized in the following steps:

1. Find the commensurate order q of $P(s)$, i.e find coefficients $a_1,\ a_2,\ a_3, \ldots a_n$ in (2.19).
2. Solve for the roots: $\mathscr{D}^q w = f(w)$, where $0 < q < 1$ and $w \in \mathbb{R}^n$, the roots are calculated by solving $f(w) = 0$
3. The roots are asymptotically stable if all the eigenvalues λ_k of the Jacobian matrix $J = \frac{\partial f}{\partial \omega}$, evaluated at the roots satisfy the condition given in equation (2.21).

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

Stability condition can also be evaluated from the state-space representation of the system but will not be considered in this work. Figure 2, shows the stability region of a fractional-order system.

2.4.2 Time Domain Analysis

Be it a classical-order system or a fractional-order system, the objective of analysis in time domain is to obtain the transient response of the system. For the fractional-order system, the Mittag-Leffler function could be used as proposed by Podlubny [6], but this could be over-lengthy and difficult to implement. Another option is the numerical computation of fractional derivative using a revised Grundwald-Lenitkov definition in 2.1.1.2.

It is rewritten as follows:

$${}_a\mathscr{D}_t^{\alpha} f(t) = \lim_{h \to 0} \frac{1}{h^{\alpha}} \sum_{j=0}^{[\frac{t-a}{h}]} (\omega_j)^{\alpha} f(t - jh), \tag{2.22}$$

where h is the step-size of the computation and $\omega_j^\alpha = (-1)^j . \begin{pmatrix} \alpha \\ j \end{pmatrix}$ is calculated recursively from

$$\omega_0^\alpha = 1,\ \omega_j^\alpha = \left(1 - \frac{\alpha+1}{j}\right)\omega_{j-1}^\alpha,\ j = 1,\ 2,\ 3,\ \ldots\ . \tag{2.23}$$

For the fractional-order system expressed in equation (2.13), one can obtain its time-domain analysis (step response) by first getting the numeric computation of signal $\hat{u}(t)$ given as

$$\hat{u}(t) = b_m \mathscr{D}^{\beta_m} u(t) + b_{m-1}\mathscr{D}^{\beta_{m-1}} u(t) + \cdots + b_1 \mathscr{D}^{\beta_1} u(t) + b0\mathscr{D}^{\beta_0} u(t). \tag{2.24}$$

then using the following equation:

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

2.4.3 Frequency Domain Analysis

Frequency domain analysis is simply getting the frequency response of a system. In other words, getting how the properties of the input signal change with frequency as it is transferred to the output. The frequency response may be computed by substituting $j\omega$ for s in equation (2.17). The complex response for a frequency $\omega \in (0,\ \ldots,\ \infty)$ can be computed using below:

$$B(j\omega) = \frac{P(j\omega)}{Z(j\omega)} = \frac{b_m(j\omega)^{\beta_m} + b_{n-1}(j\omega)^{\beta_{n-1}} + \cdots + b_1(j\omega)^{\beta} + b_0(j\omega)^{\beta_0}}{a_n(j\omega)^{\alpha_n} + a_{n-1}(j\omega)^{\alpha_{n-1}} + \cdots + a_1(j\omega)^{\alpha_1} + a_0(j\omega)^{\alpha_0}} \tag{2.26}$$

where j is the imaginary unit. There is also a useful relation for the noniteger power $\alpha \in \mathbb{R}$ of the imaginary unit. Consider the following

$$j^\alpha = \cos\left(\frac{\alpha\pi}{2}\right) + j\sin\left(\frac{\alpha\pi}{2}\right). \tag{2.27}$$

2.5 Fractional-Order Approximation

Integer-order systems are often used in industries perhaps because there are variety of tools for its analysis, thus the approximations of fractional-order models by an integer-order one will be valuable to compare or verify the accuracy of fractional-order with integer-order systems that is already well-established. In [29], many approximation methods are discussed but in work, the Oustaloup recursive filter proposed in [30], was used because it has a very high approximation fitness for a specified frequency range [31].

Assume that the expected fitting range is (ω_b, ω_h) and of order N, the filter for an operator s^γ, $0 < \gamma < 1$, is given as

$$G_f(s) = K \prod_{k=-N}^{N} \frac{s+\omega'_k}{s+\omega_k} \tag{2.28}$$

where the poles, zeros and gain of the filter can be computed from (2.31).

$$\begin{aligned} \omega'_k &= \omega_b \cdot \omega_u^{(2k-1-\alpha)/N} \\ \omega_k &= \omega_b \cdot \omega_u^{(2k-1+\alpha)/N} \\ K &= \omega_h^\alpha, \quad \omega_u = \sqrt{\omega_h/\omega_b} \end{aligned} \tag{2.29}$$

In [9], [31] a refined Oustaloup filter was proposed.it is given as

$$s^\alpha \approx \left(\frac{d\omega_h}{b}\right)_\alpha \left(\frac{ds^2 + b\omega_h s}{d(1-\alpha)s^2 + b\omega_h s + \alpha}\right) \prod_{k=-N}^{N} \frac{s+\omega'_k}{s+\omega_k} \tag{2.30}$$

where the zeros and poles of rank k is written as

$$\omega'_k = \left(\frac{b\omega_h}{d}\right)^{\frac{\alpha-2k}{2N+1}}, \ \omega_k = \left(\frac{b\omega_h}{d}\right)^{\frac{\alpha+2k}{2N+1}}, \ K = \omega_h^\gamma. \tag{2.31}$$

It is expected that optimal approximation of (2.30) is obtained when $b = 10$, and $d = 9$, confirmed from experiments and theoretical analysis.

Note 2.5.1 *For a fractional-order* $\alpha \geq 1$*, it holds that* $s^{\alpha} = s^{n} \cdot s^{\gamma}$

2.6 Fractional System Discretization

In the implementation of controllers, discretization is essential. Several discretization methods have been developed for fractional-order models[28]. These include infinite response filter (IIR) and finite response filter (FIR) realizations, the former being preferred to the latter due to the lower order of this type of filter[9]. 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.

Continuous-time controllers are usually specified as a transfer function $C(s)$, with the world going digital, it is natural to look for methods that will transform the continuous transfer function $C(s)$ to a pulse transfer function $C_d(z)$ so that the corresponding behaviours of the two system are approximately same. When it comes to discrete-time fractional differ-integrator implementation, it is not a simple task [28]. Some methods used to relate the s and z domain mathematically as follows:

2.6.1 Forward difference or Euler's Method

The forward difference method is defined by the following equation:

$$s \approx \frac{z-1}{T} \tag{2.32}$$

One challenge with this approximation is that it is possible that a stable continuous system is mapped to an unstable discrete-time system.

2.6.2 Backward Difference

The backward difference method is defined by the following equation:

$$s \approx \frac{z-1}{zT} \tag{2.33}$$

A stable continuous-time system using (2.33) will always give a stable discrete-time system.

2.6.3 Tustin or Trapezoidal or bilinear method

It relates the s and z domains with the following substitution formula:

$$s \approx \frac{2z-1}{Tz+1} \tag{2.34}$$

This method has the most advantage because the left-half s-plane is transformed into the unit disc in the z-plane (i.e. it maps points $s = 0$ and $s = \infty$ to the points $z = 1$ and $z = -1$, respectively. It is easy to implement, well known in finite dimensions, preserves conservativity (i.e, frequency response are equal after discretization) and theoretically simple.

2.7 Optimization Methods

Application of numerical optimization methods form an important part of the work especially in the identification of fractional-order system. Thus the need to provide methods applied in this section. The employed algorithm methods are based on the available methods in *Python's* `scipy.optimize.leastsquares` library [32].

2.7.1 Nonlinear Least-Squares Method

The basic concept in this method is to obtain a model of an initially guessed system by means of minimization of the sum of squares of the error.

$$F = \sum_{i=1}^{n} \varepsilon_i^2 = \|\varepsilon\|^2, \tag{2.35}$$

where $\varepsilon_i = y_i - \hat{y}_i$ is the simulation error (residual), y_i is the true system output and $\hat{y}_i$ is the predicted output for collected samples $i = 1, 2, \ldots, N$.

Two minimization algorithm can be applied in conjunction with 2.7.1 to always minimize the error with every iteration. Consider the following:

2.7.2 Trust Region Reflective

This is a minimization method for handling large-scale bounded problems [33], [34]. Given a trust region Δ_k at every kth iteration the following steps are carried out [35]:

1. Compute F_k , g_k (gradient of F_k), D_k (positive diagonal matrix), H_k and C_k (scaling matrices), define the quadratic model

$$\psi_k(s) = g_k^T s + \frac{1}{2} s^T (H_k + C_k) s. \tag{2.36}$$

2. Compute a step s_k, with $x_k + s_k \in \text{int}(\mathscr{F})$, where $\mathscr{F}$ is the feasible region for searched variable values, by solving the sub-problem

$$\min_s \{\psi_k(s) : \|D_k s\| < \Delta_k, s \in S_k\} \tag{2.37}$$

   where S_k is a small-dimensional subspace in $\mathbb{R}^n$.

3. If $F(x_k + s_k) < F(x_k)$, then $x_{k+1} = x_k + s_k$, otherwise x_k remains unchanged for the

next iteration.

4. Adjusting the trust region Δk. In case of the least-squares problem the subspace S_k may be determined by taking into account.

$$\min_s \left\{ J_s + F \left\| \right\|_2^2 \right\} \tag{2.38}$$

where J is the Jacobian of F.

2.7.3 Levenberg-Marquardt

The search direction p_k of this method is defined by the solution of equations at iteration step k [36], [37]

$$(J_k^T \cdot J_k + \lambda_k \cdot I) p_k = J_k^T \cdot F_k, \tag{2.39}$$

where J_k is the Jacobian matrix, λ_k is a non-negative scalar, and I is the identify matrix. It is important to note that this search algorithm is **boundless**.

3. Fractional-order Identification Methods and Algorithms

This chapter is devoted to research of methods of identification using fractional-order model. It is organized as follows. In Section 3.1, a general view on system identification is provided. Section 3.2 discusses, methods of model identification from experimental time-domain data using methods and algorithm discussed in Chapter 3. Lastly in Section 3.3, we discuss the analysis of the identified model.

3.1 Basics of Model Identification

In this work, we mostly considered the black modelling approach [27] as no assumptions were made on the internal features of the studied system. Our target is to infer a dynamic system model based on data set measured during an experiment. In the identification process, its mandatory to obtain a relationship between set of system inputs and system output under external stimuli (e.g. input signals, noise, vibrations etc). This relationship helps predict and determine the system behaviour very accurately as its based on data and optimization algorithms.

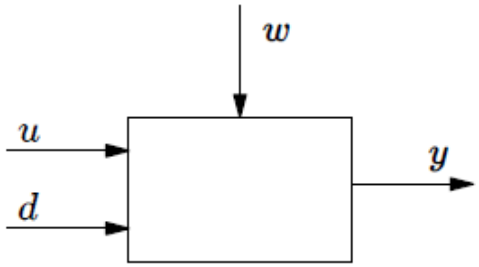


Figure 3. System with input u, output y, measured disturbance d and unknown noise w

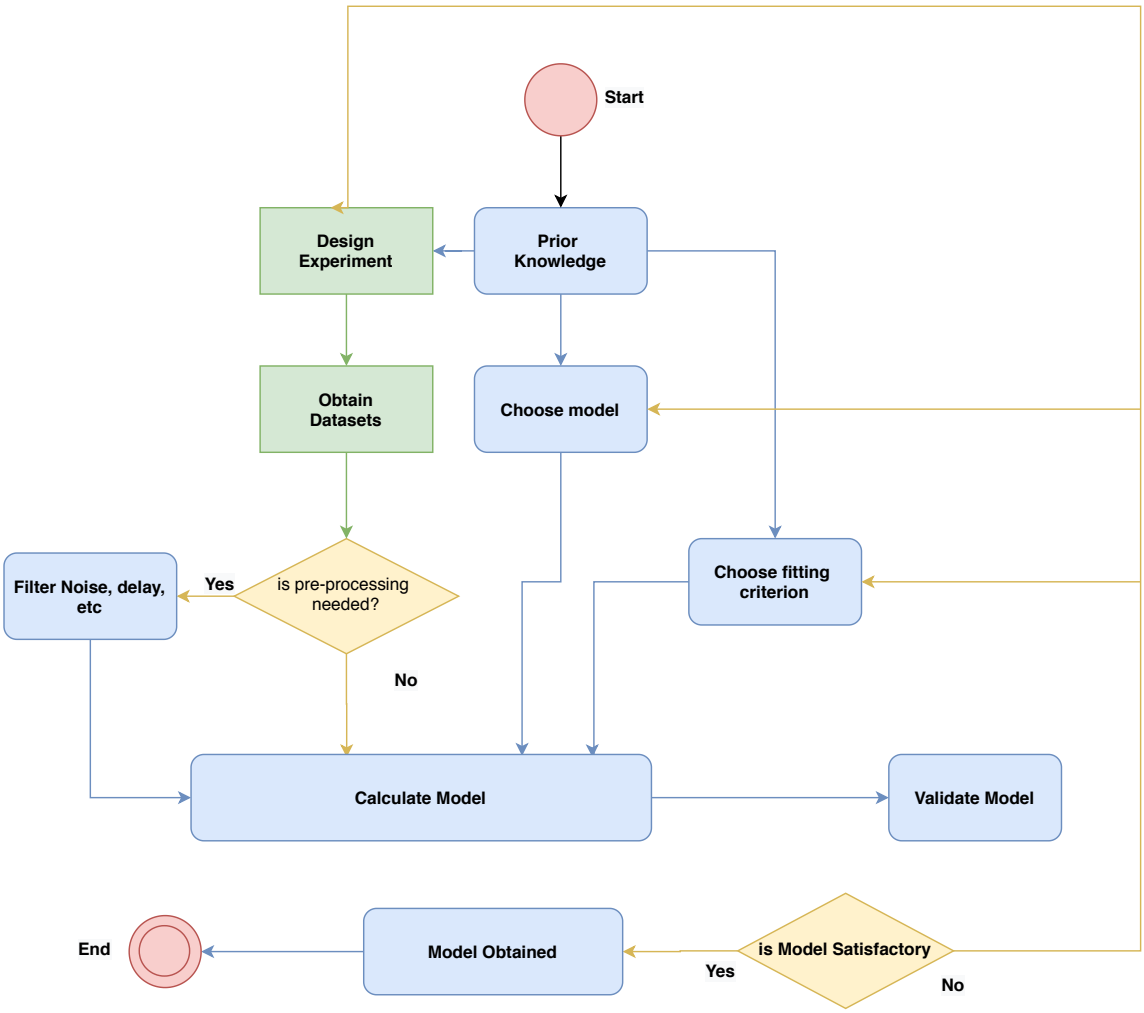


Figure 4. System Identification Procedure

Consider Figure 3, it shows a general form of a single input-single output system with disturbance.The stages of system identification is discussed in the Section 3.1.1. A flow diagram is provided in Figure 4 for the readers' perusal.

3.1.1 System Identification stages

1. Design the experiment. For dynamic systems, a suitable approach is to collect transient response data in the time-domain by applying a set of known inputs signals. In the frequency domain, frequency response (phase and magnitude) data can also be collected by doing a frequency sweep.
2. Collate the data-sets based on the designed experiment. It is paramount that data is as informative as possible while subject to whatever present constrains.

3. Select the model structure and the criterion to fit.
4. Calculate the model using a cost function and suitable algorithm. (e.g, nonlinear least squares).
5. Validate the obtained model. It is advisable to use different data-set for identification and validation.
6. Is model accuracy satisfactory? If it is, use your model as you desire. if not, revamp modelling and/or identification strategy then repeat steps 1 to 5.

Note 3.1.1 *In the identification process, there could be the need to pre-process data-set to filter-off none useful components (like contributing noise, delays) before actually using it in the identification algorithms.*

3.2 Time-domain Identification

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

The objective of time-domain identification is to use experimental data from a single input single output non linear system to obtain a obtain a mathematical fractional-order representation (model) of the system in the form (3.1).

3.2.1 Time Domain Data Generation

Suppose that experimental data is collected from a general single input, single output (SISO) nonlinear system $\psi : I \to O$, where $(I, O) \subset \mathbb{R}^2$ denote the measured input and output signals, respectively, such that

$$z(t) = \psi(v(t)) + \mathfrak{D} \tag{3.2}$$

where $z(t)$, $v(t)$ and $\mathfrak{N}$ denotes the system output, input and noise respectively. Thus (3.2) represents a data set holding multiple samples from the system output $y_k = z(kt_s) + \mathfrak{N}$ and

system input $u_k = v(kt_s)$, under a uniform sample rate $t_s = t_{k+1} - t_k$.

$$Z_n = \{u_0, y_0, u_1, y_1, \ldots, u_n, y_n, t_s\},\ k = 0, 1, \ldots, n.$$

If $z(0) = y_0 \neq 0$, then an offset y_0 can be removed from each of the collected output samples since zero initial condition was assumed.It is represented mathematically as

$$y_k = y_k - y_0, \quad k = 0, 1, \ldots, N$$

3.2.2 Initial Guess Model

Several methods of identification are described in [27] and more specifically for fractional-order system and this work in [37]–[39] where equation error, output error approaches and least-squares were discussed. The approach postulates that a commensurate-order γ transfer function $G(s)$ is BIBO (bounded input-bounded output) stable if

$$0 < \gamma < 2 \tag{3.3}$$

where it holds that for every s^γ pole, $s_k \in \mathbb{C}$ of $G(s)$ it holds the stability criterion discussed in 2.4.1:

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

Thus the initial guess model to be used in the identification process can be chosen by selecting a commensurate-order γ that is in accordance with (3.3). It is strongly advised to used a commensurate order initial guess because the success of the identification process could depend mainly on the initial conditions set, of which the initial guess is one of them [9]. For example a fractional-order polynomial with order n, will have the highest differential order as $n \times \gamma$. A fractional order polynomial (s) with order $n = 4$ and

commensurate-order $\gamma = 0.25$ is as below

$$G(s) = 15s^{1.0} + 10s^{0.75} + 5s^{0.5} + s^{0.25} + 20.$$

3.2.3 Parameter Optimization

One of the reasons we discussed optimization in Section (2.7) is because the system identification problem is synonymous to the problem of optimizing a set of parameters θ of the model (3.1) given by

$$\begin{aligned} a_p &= \begin{bmatrix} a_n & a_{n-1} & \cdots & a_0 \end{bmatrix}, \quad \alpha_p = \begin{bmatrix} \alpha_n & \alpha_{n-1} & \cdots & \alpha_0 \end{bmatrix} \\ b_z &= \begin{bmatrix} b_m & b_{m-1} & \cdots & b_0 \end{bmatrix}, \quad \beta_z = \begin{bmatrix} \beta_n & \beta_{n-1} & \cdots & \beta_0 \end{bmatrix} \end{aligned} \tag{3.4}$$

a_p and b_z denote pole and zero polynomial differential operator coefficients, α_p and β_z denote their corresponding exponents (orders of differentiation), respectively; if $\alpha_0 = \beta_0 = 0$, then the system static gain is identified as $K = b_0/a_0$.

There are 9 possible parameter optimization sets θ_m depending on the users choice:

1. Optimize zero and pole polynomial
   - Full parameter identification, $\theta_m = [a_p, \quad \alpha_p, \quad b_z, \quad \beta_z]$
   - Static exponents, optimize coefficients only, $\theta_m = [a_p, \quad b_z]$
   - Static coefficients, optimize exponents only, $\theta_m = [\alpha_p, \quad \beta_z]$
2. Zero polynomial static, optimize pole polynomial only:
   - Identify pole polynomial coefficient and exponents, $\theta_m = [a_p, \quad \alpha_p]$
   - Static exponents, optimize pole polynomial coefficients only, $\theta_m = a_p$
   - Static coefficients, optimize pole exponents only, $\theta_m = \alpha_p$
3. Pole polynomial static, optimize zero polynomial only:
   - Identify zero polynomial coefficient and exponents, $\theta_m = [b_z, \quad \beta_z]$
   - Static exponents, optimize zero polynomial coefficients only, $\theta_m = b_z$

- Static coefficients, optimize zero exponents only, $\theta_m = \beta_z$

It is important to note that the order of the generated pole polynomial m and zero polynomial n in the initial guess model discussed in Section 3.2.2 should satisfy the condition:

$$\begin{aligned} \alpha_p &= [\gamma \cdot n \quad \gamma \cdot (n-1) \quad \cdots \quad 0] \\ \beta_z &= [\gamma \cdot m \quad \gamma \cdot (m-1) \quad \cdots \quad 0] \\ \text{such that} \quad & \left\{(n,m) \in \mathbb{Z}_+^2 : n \geqslant m\right\} \end{aligned} \tag{3.5}$$

In simple words, the order of pole polynomial, should be greater than or equal to the order of zero polynomial. The identification process is iterative and could sometimes be slow. The optimization criterion is the output mean squared nominal error $\|\varepsilon(t)\|_2^2$ given by

$$\varepsilon(t) = y(t) - \hat{y}(t), \tag{3.6}$$

where $y(t)$ is the experimental output and $\hat{y}(t)$ is the output obtained by simulation of the identification model using an experimental input $u(t)$.

3.3 Validation of Identified Model

After the identification process, a system model is identified. It is important to assess how accurate the identified model is in comparison with same or another experimental plant output data to certify the model.

Let us denote the second experimental plant output y_v and the identified model output y_m for a single input, single output case, so both y_v and y_m should be vectors of size $[N \times 1]$. The residuals are given as a vector containing the model output error given as

$$\varepsilon = y_v - y_m \tag{3.7}$$

The percentage fitness of the model may be expressed as

$$Fit = \left(1 - \frac{\|\varepsilon\|}{\|y_v - \bar{y}_v\|}\right) \cdot 100\% \tag{3.8}$$

where $\|\cdot\|$ means the euclidean norm and $\bar{y}_v$ denotes the mean of y_v. In the *FOMCONpy* library that will be introduced in Chapter 5, the percentage fitness of the identified model, is always computed after the identification process in command-line mode or when the `Validate` button is pressed in GUI mode . The user is left to make the choice if satisfied with the identified model.

4. Fractional-order Control

In this chapter we focus on fractional-order system control. It applies the fractional calculus concept to extend the classical integer-order control. The chapter is organised as follows. Section 4.1 introduces the conventional Proportional, Integral and Differential (PID) controller. Section 4.2 presents a summary of the Fractional-order PID (FOPID) controller and a summary of the fractional integral and differential control action. Section 4.3 discusses the FOPID tuning method used in the fomconpy with reason for using such tuning method.

4.1 Basics of Proportional, Integral and Differential Control

To better understand fractional-order control which is an extension of integer order control, the author thought it beneficial to first provide the reader with a summary of the foundation of proportional, integral and differential control.

4.1.1 Proportional Action

let us consider a controller in Fig. 5. It is to control a system with present output y, set-point/set-value/desired output r, manipulated or control variable u and the error $e = r - y$.

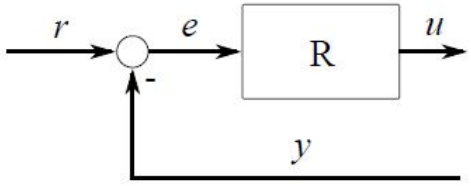


Figure 5. Block Diagram of a Controller

For a setup like in Fig. 5, the initial data for the controller are the y, r, and information about itself. The most common of industrial control action is the proportional (P) control action which simply compares the y to r to determine if an error exist. if an error exist, the controller will adjust its output u according to the parameter that have been set in itself. The control law u will be as in (4.1) for a proportional controller.

$$u = K_c \cdot e + u_0. \tag{4.1}$$

proportional control action responds to change in the magnitude of the error. However the proportional action alone will not ensure the system output y converges to the desired output r. In fact, there will always be an offset (also known as bias) so that $r \neq y$ thus the u_0 in (4.1).

4.1.2 Integral Action

Due to this offset u_0, there is the introduction of the Integral error to eliminate the offset, Unlike proportional action which simply moves an amount proportional to a change in y or r, integral action is continually applied until error has been eliminated. The integral action simply tells the output how fast to move. It is mathematically expressed as

$$u = K_i \int e \, \mathrm{d}t = \frac{1}{T_i} \int e \, \mathrm{d}t \tag{4.2}$$

where K_i is the integral gain (repeats per minutes), T_i is the integral time constant (minutes per repeat). Integration of error means the continual sum of controller error up to present time. Some effects of integral action are stated in the following bullets:

- Decreases the rise time (that is, faster rate to get to set-point).
- Increase settling time and overshoot.
- Causes the displacement of the root locus of the system towards the right-half plane (that is, fosters instability of the system).

However there occurs a scenario where the controller output is driven from a desired output level caused by large difference between the set-point (desired level) and the process variable (system output) this scenario is called "integral wind-up" (see [40] for more information).

4.1.3 Derivative Action

The derivative action causes the output signal to be offset by an amount proportional to the rate at which the input is changing. While proportional action tells how much changes based on error, derivative action tells how far to go when input ramps thus it has a futuristic behaviour.

$$u = K_c \cdot e + K_d \frac{\mathrm{d}e}{\mathrm{d}t} = K_c \left(e + T_d \frac{\mathrm{d}e}{\mathrm{d}t} \right) \tag{4.3}$$

T_d is the rate time and characterizes the derivative action. Adjusting T_d varies the amount of the derivative action, setting it to zero turns off the derivative action. Some effects on derivative action are stated in the following bullets:

- Causes the displacement of the root locus of the system towards the left half plane (that is fosters the stability of the system).
- Tends to emphasize the effects of noise at high frequencies.
- Decreases settling time and overshoot.

4.1.4 PID Control

A combination of the three control actions described above is referred to as PID control action u. This action is represented as

$$u = K_p \cdot e + K_i \int e \, \mathrm{d}t + K_d \frac{\mathrm{d}e}{\mathrm{d}t} \tag{4.4}$$

and the transfer function is represented as

$$C(s) = K_p + \frac{K_i}{s} + T_d \cdot s \tag{4.5}$$

or

$$C(s) = k\frac{(s/\omega_c)^2 + 2\delta_c s/\omega_c + 1}{s} \tag{4.6}$$

$$with \quad \omega_c = \sqrt{K_i/K_d}, \quad \delta = K/(2\sqrt{K_i \cdot K_p}, \quad k = K_i$$

For more details of integer order control can be found in [9], [40]

4.2 Fractional-order Proportional, Integral and Differential Controller

Podlubny [9], [41] introduced the generalised representation of the fractional-order PID (FOPID) controller called the $PI^{\lambda}D^{\mu}$ controller, where the integral has an order λ, the differential an order μ and unless $\lambda = \mu = 1$ cannot be same like the classical PID controller summarised in Section 4.1.4. In same [41], he demonstrated that FOPID controller has better response compared to PID which was also agreed to e.g, by Cech in [42]. Just as we did for PID controller, let us defined the control action for an FOPID controller in the time domain.

$$u(t) = K_p e(t) + K_i \mathscr{D}^{-\lambda} e(t) + K_d \mathscr{D}^{\mu} e(t), \tag{4.7}$$

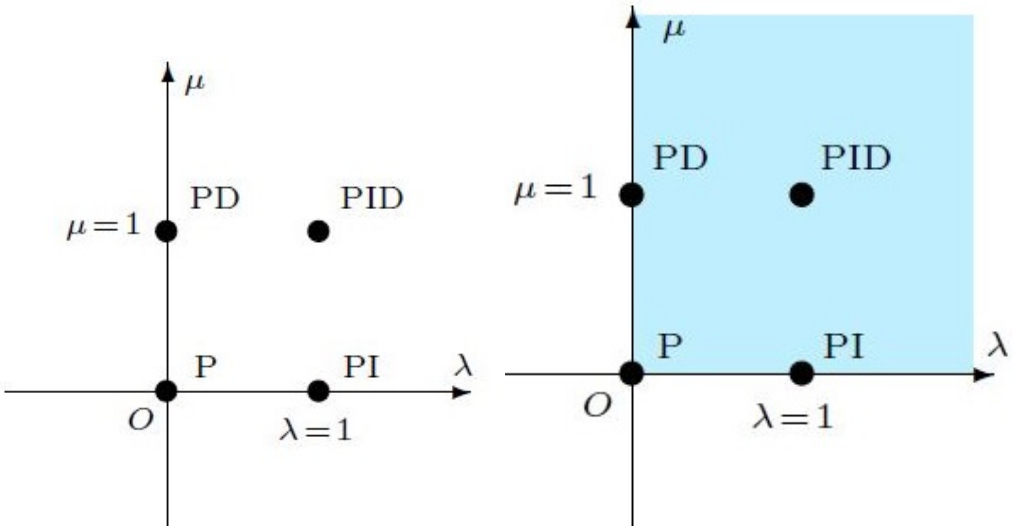
where $e(t)$ is the error signal.

In the frequency domain,

$$G_c(s) = K_p + \frac{K_i}{s^{\lambda}} + K_d s^{\mu}, \tag{4.8}$$

Fig. 6 shows the transformation effect from a PID of four-points only to a PID plane.

Further, let us summarize the effects of the extended integral and the extended differential



(a) Integer-order (4 points only) (b) Fractional-order (4 points + Plane)

Figure 6. classical PID vs Fractional-order PID [9]

control action. Consider a basic control action of type $K \cdot s^{\mu}$ for $\mu \in [-1,1]$. The basic control actions that will be considered are integral and differential:

Integral Action

For an integral action, $\mu \in [-1,0]$ with $K = 1$, varying orders of μ and a square error signal. In the time domain, the control action $u(t)$ will have the form depicted in Fig. 7, but in the frequency domain, the effect of varying μ between -1 and 0 is as follows:

- There is a constant increment in the slope of the magnitude curve varying between -20 dB/dec and 0 dB/dec.
- There is also be a constant delay in the phase plot, varying between -$\pi/2$ rad and 0 rad.

Differential Action

The differential action is investigated with $\mu \in [0,1]$, varying orders of μ and and a trapezoidal error signal. In the time domain, the effects of the derivative action is as shown in Fig. 8 while in the frequency domain, the effect of varying μ between 0 and 1 is as follows:

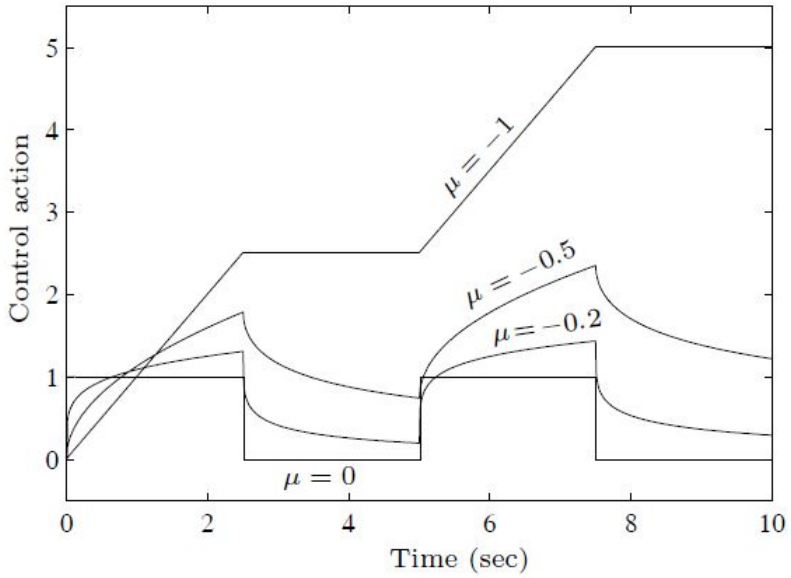


Figure 7. Integral control action for a square error signal and $\mu \in [-1, 0]$ [9]

- There is a constant increment in the slope of the magnitude curve varying between 0 dB/dec and 20 dB/dec.
- There is also be a constant delay in the phase plot, varying between 0 rad and $\pi/2$ rad.

It is quite obvious that extending the integrator and differentiator of PID to support fractional-order has potential benefits and hopefully, will replace the classical control in the industrial control application. The author anticipates this with the development of a tool like `fomconpy` which supports dynamic system modelling and distributed control.

4.3 Tuning of Fractional-order Proportional, Integral and Differential Controllers

In the industry, there are various processes and systems that need to be controlled. However, it is practically impossible and inefficient to make an out-of-the-box/plug-n-play controller for each and every process/system. Instead, a controller parameters (proportional, integral and differential) can be adjusted to suit the particular system or process that it will control. The process of adjusting the parameters of the controller to suit the particular case is

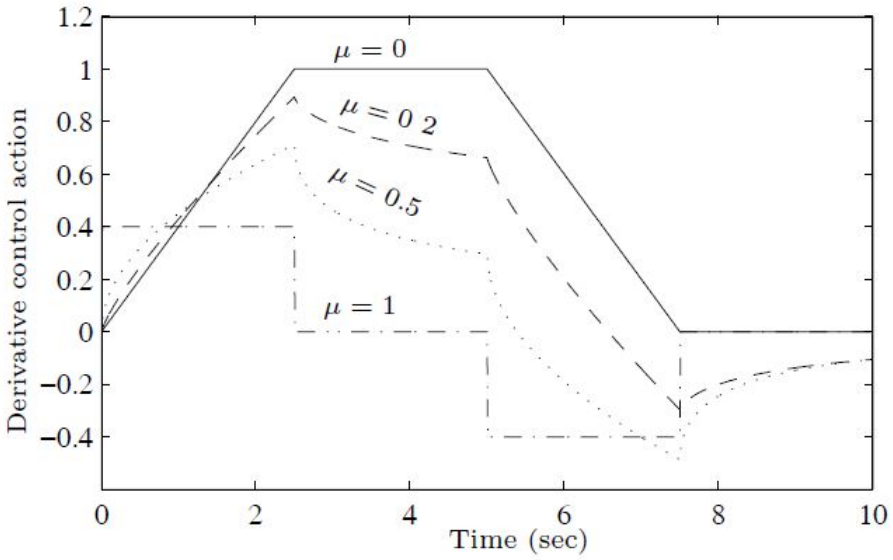


Figure 8. Differential control action for a trapezoidal error signal and $\mu \in [0,1]$ [9]

referred to as **tuning**.

Over the years, several tuning methods have been proposed by different researchers [9], [18], [43], [44] using different optimization techniques in the tuning problem depending on the design specification of the intended fractional-order PID controller. Most frequently used design specifications are derived from the frequency domain evaluation of the open loop $C(j\omega) \cdot G(j\omega)$, where $C(j\omega)$ is the controller and $G(j\omega)$ is the plant. In terms of noise and disturbance rejection, the following measures can be used:

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

$$S(j\omega) = \frac{1}{1 + C(j\omega)G(j\omega)} \tag{4.9}$$

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

$$T(j\omega) = \frac{C(j\omega)G(j\omega)}{1 + C(j\omega)G(j\omega)} \tag{4.10}$$

For this work, we focused on the tuning algorithm in [44], [45] which is a hybrid of the F-MIGO Tuning algorithm discussed in chapter 5 of [9]. Which says that for a fractional-order first order plus dead time (FO-FOPDT) model given as (4.14), the appropriate integral order λ of the controller can be obtained by considering basic system dynamics through the relative dead time parameter τ_c given as:

$$\tau_c = \frac{L_c}{L_c + T_c} \tag{4.11}$$

Then applying the approximate rule given in 4.12

$$\lambda = \begin{cases} 1.1, & \tau_c \geqslant 0.6 \\ 1.0, & 0.4 \leqslant \tau_c < 0.6 \\ 0.9, & 0.1 \leqslant \tau_c < 0.4 \\ 0.7, & \tau_c < 0.1 \end{cases} \tag{4.12}$$

For deciding the diffferentiial order μ, knowledge of the FO-FOPDT plant order α is necessary and they should have the relation in (4.13).

$$\mu \leqslant \alpha \tag{4.13}$$

The reasons for choosing this tuning method is because the tuning method is fast, takes less computation power (*FOMCONpy* library was developed with a focus to run smoothly on a miniature IOT devices) and it's suitable for a FO-FOPDT model given in (4.14).

$$G(s) = \frac{K}{T_c \cdot s^{\alpha} + 1} \mathrm{e}^{-L_c s} \tag{4.14}$$

we considered the following tuning specifications:

- Exact phase margin φ_m and corresponding crossover frequency ω_c,
- Gain variation robustness, that is there exist a requirement such that

$$\psi_g'(\omega_c) = 0 \tag{4.15}$$

where

$$\psi_g(\omega) = \arg(C(j\omega)) + \arg(G(j\omega)) + \pi + 2\pi n. \tag{4.16}$$

- Minimum gain margin G_m.

The following functions may be defined, based on the specifications above:

$$\kappa_1(K_p, K_i, K_d) = |C(j\omega)| \cdot |G(j\omega)| - 1, \tag{4.17}$$

$$\kappa_2(K_p, K_i, K_d) = \arg(C(j\omega)) + \arg(G(j\omega)) + \pi - \varphi_m - 2\pi n, \tag{4.18}$$

$$\kappa_3(K_p, K_i, K_d) = \psi'_{gm}(\omega), \tag{4.19}$$

where $\omega = \omega_c$.

Note 4.3.1 *Eqns. (4.17), (4.18) and (4.19) do not include the gain margin specification. However, the solution is only considered feasible, if the minimal gain margin is satisfied.*

To find the gains $g = [K_p, \quad K_i, \quad K_d]^T$ of the FOPID controller according to the specifications given above, we solve a system of nonlinear equations comprised of the design specification functions.

$$F_s = \begin{bmatrix} \kappa_1(\cdot) & \kappa_2(\cdot) & \kappa_3(\cdot) \end{bmatrix}^{\mathrm{T}} = 0 \tag{4.20}$$

Newton's method in several dimensions may be employed here. Starting from the initial estimate g_0, the iterative process begins until a particular stop condition is satisfied. On every step, a linear system $J\delta g = -F_s$ must be solved then the new controller gain vector $g^+ = g + \delta g$. Detailed algorithm and mathematical equation can be found in [44].

5. User Manual for FOMCONpy

In this chapter, we introduce the reader to the FOMCONpy library for python in which the author implemented major features already discussed in previous chapters. The chapter is organised as follows. In Section 5.1, the reader is presented an overview of the library and dependencies on other libraries. In Section 5.2 installation guide in presented for Windows and Linux user (catering for the IOT community). In Section 5.3, we present the fractional system analysis module with examples on command line and with GUI. Section 5.4, have illustrative examples about the fractional system Identification module in time domain using command line and using GUI. Section 5.5 shows the application of FOPID controller tuning using command line and finally, Section 5.6 presents the FOPID controller and tuner GUI with application to IOT and distributed control system.

5.1 Overview

FOMCONpy stand for "Fractional-Order Modelling and Control for Python". It is a *Python* library extended from it counterpart, *FOMCON* for MATLAB [46] but with features for IoT community and devices especially with the reality of 5G.

In recent years, there has been an enormous exchange of data as we try to “smarten” automated processes, this has prompted manufacturing industries to use wireless network communication for industrial automation [47] because cabling often hinders scaling and often suffers from wear and tear thereby increasing maintenance costs [48]. We anticipate that the development of a fractional calculus library in *Python* will facilitate and simplify the use of advanced fractional-order modelling, identification and control in the development of real industrial stand-alone applications in Industry 4.0 revolution.

The *FOMCONpy* library, provides graphical user interfaces (GUIs) and appropriate back-end functions that can be used to model, identify, design and optimize fractional-order systems, including fractional controllers. The goal is to develop a multi operating-system library for a wide range of users. Beginners enjoy the availability of complete GUIs plus added command-line features for more experienced users.

The library has 3 distinct modules namely:

1. Fractional-order System Analysis Module (FOSAM).
2. Fractional-order System Identification Module (FOSIM).
3. Fractional-order System Control Module (FOSCOM).

These modules shall be discussed in coming sections with command line and GUI examples. *FOMCONpy* was developed and tested using *Python* 3.7.4 interpreter which makes this version of *Python* the minimal supported version.

5.2 Setting-up FOMCONpy: A Quick Tutorial

FOMCONpy makes use of the following *Python* packages and their dependencies: numpy, scipy, control, maplotlib, pandas, xlrd, select, addict and PyQt5 (for GUI).

Setup will be discussed for Windows OS and Linux OS (similar for Raspberry PI IoT device).

5.2.1 Setup guide for Windows

1. Install Anaconda.
2. Install Git.
3. Navigate using the command prompt to any desired directory of choice.
4. Clone FOMCONpy from its repository:

   `git clone https://github.com/outstandn/fomcon.git`.

5. Change current working directory to *fomcon*: `cd fomcon`
6. Create *fomcon* environment: `conda env create -f environment.yml`
7. Activate *fomcon* environment: `conda activate fomcon`

5.2.2 Setup guide for Linux

Open a terminal and enter the following commands:

1. `sudo apt-get install git python3-pyqt5 python3-numpy python3-scipy python3-pandas python3-xlrd python3-matplotlib python3-pip python3-select`
2. `pip3 install control addict`
3. Continue with steps 3 to 5 in Section 5.2.1

5.3 Fractional-order System Analysis Module

The main module of the *FOMCONpy* library is the system analysis module. It is the foundation module for the fractional-order transfer function class called `FOTransFunc`. it can be found in the fotf module.

5.3.1 class `FOTransFunc()`

Before using the `FOTransFunc` class, you need to first import the `fotf` module

Syntax

- `G = FOTransFunc(num, nnum, den, nden, dt)`
- `G = FOTransFunc(Zero_s, Pole_s, dt)`
- `G = FOTransFunc([num, nnum], [den, nden] , dt)`
- `G = FOTransFunc("s")`
- `G = FOTransFunc(k)`

Arguments

- `num, nnum, den, nden[, dt]`

  `num` – vector coefficients of the zero polynomial.

  `nnum` – vector exponents of the zero polynomial.

  `den` – vector coefficient of the pole polynomial.

  `nden` – vector exponents of the pole polynomial.

  `dt` – delay (optional `float` or `int`).

- `[num, nnum], [den, nden].`

  `[num, nnum]` – matrix having 2 vectors, for coefficients and exponents of the zero polynomial.

  `[den, nden]` – matrix having 2 vectors, for coefficients and exponents of the pole polynomial.

- `Zero_s, Pole_s.`

  `Zero_s` – fractional zero polynomial string.

  `Pole_s` – fractional pole polynomial string.

- `"s"` – if 's' is the only argument, it returns `FOTransFunc(1,1,1,0).`
- `k` – a float or int only and returns `FOTransFunc(k,0,1,0).`

Returns

- `G` – the resulting `FOTransFunc` class object, represents the LTI system given by fractional-order transfer function.

Examples

To create a fractional-order transfer function $G(s) = \frac{-2s^{0.63}+4}{2.6s^{1.8}+2.5s^{1.31}+1.5}e^{-5s}$ in *Python*, the following command can be used:

```
1 import fotf
```

```
G=FOTransFunc([[-2,4],[0.63,0]],[[2.6,2.5,1.5],[1.8,1.31,0]],5)
#a second way to fill in arguments
G1=FOTransFunc([-2,4],[0.63,0],[2.5,1.5],[1.8,1.31,0],5)
#a third way to fill in arguments
G2=FOTransFunc("-2s^{0.63}+4","2.6s^{1.8}+2.5s^{1.31}+1.5",5)
#let us check if the three are equal
if G==G1==G2:
    print("G == G1 == G2")
    print(G)
```

Listing 5.1. Creating a `FOTransfunc` object

The output from the *Python* interpreter is as below:

```
G == G1 == G2

         -2s^{0.63} + 4
------------------------------exp(-5s)
2.6s^{1.8} + 2.5s^{1.31} + 1.5
```

Remarks

To support ease of use for FOMCON users, the function `fotf` and `newfotf` also exist in *FOMCONpy* to ease switching from MATLAB to *Python* and vice-versa. Both functions still return a `FOTransFunc` object. The function `fotf` can be used interchangeably with class `FOTransFunc` to create a new instances but `newfotf` can only be used to create new `FOTransFunc` object as specified in Section 5.3.2.

5.3.2 def `newfotf()`

This function is used to create `FOTransFunc` object but must always take 2 or 3 argument.

Syntax

- `G = newfotf([num nnum], [den nden], dt)`

- `G = newfotf(Zero_s, Pole_s, dt)`
- `G = newfotf([num nnum], Pole_s, dt)`
- `G = newfotf(Zero_s, [den, nden], dt)`

Arguments

- `Zero_s` – fractional zero polynomial string.
- `Pole_s` – fractional pole polynomial string.
- `dt` – optional delay (`int` or `float`).
- `[num nnum]` – zero polynomial co-efficient and exponent vector.
- `[den nden]` – pole polynomial co-efficient and exponent vector.

Returns

- `G` – the resulting `FOTransFunc` class object.

Example

Using same example as in 5.3.1, $G(s) = \frac{-2s^{0.63}+4}{2.6s^{1.8}+2.5s^{1.31}+1.5}e^{-5s}$, the following command can be used.

```
import fotf
G=newfotf([-2,4,0.63,0],[2.6,2.5,1.5,1.8,1.31,0],5)
#a second way to fill in arguments
G1=newfotf("-2s^0.63+4","2.6s^1.8+2.5s^1.31+1.5",5)
#a third way to fill in arguments
G2=newfotf([[-2,4],[0.63,0]],"2.6s^1.8+2.5s^1.31+1.5",5)
#a fourth way to fill in arguments
G3=newfotf("-2s^0.63+4",[[2.6,2.5,1.5],[1.8,1.31,0]],5)
#let us check if the four are equal
if G==G1==G2==G3:
    print("G == G1 == G2 == G3")
    print(G)
```

Listing 5.2. Creating a `FOTransfunc` object using the `newfotf` function

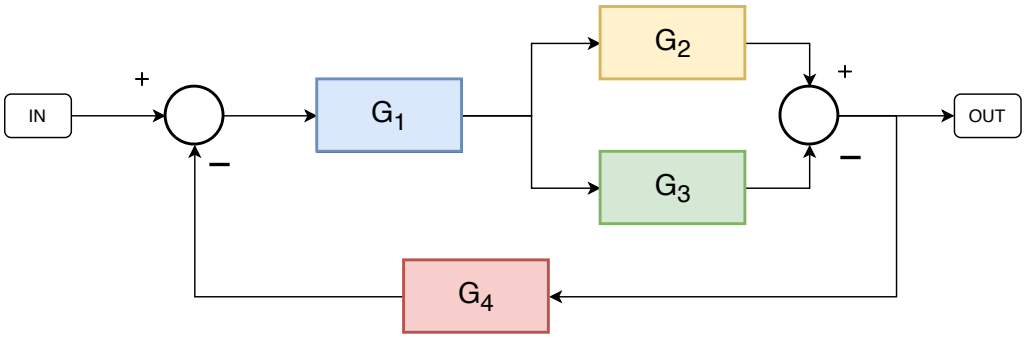


Figure 9. Example for system block interconnection

The output from the *Python* interpreter is as below which is same output as in Example of Section 5.3.1.

```
G == G1 == G2 == G3

         -2s^{0.63} + 4
------------------------------exp(-5s)
2.6s^{1.8} + 2.5s^{1.31} + 1.5
```

5.3.3 `FOTransFunc` Interconnection

These are operators and properties implemented in the `FOTransFunc` class that supports block interconnection. Table 1 has the details of these block interconnection operators.

Example

Consider a fractional system given by the block diagram in Fig. 9, where:

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

The full model can be obtained using the following commands:

```
1 import fotf
2 G1 = newfotf([1, 0],[1, 1, 0.5, 0])      #Create system  G1
3 G2 = newfotf([1, 1, 0.3, 0], [1, 1, 1, 2.5, 1, 0])  #Create system  G2
```

```
G3 = newfotf([2, 0], [1, 1, 0.1, 0])     #Create system  G3
G4 = newfotf([1, 0], [15, 1, 1, 0])      #Create system  G4
G = (G1*(G2 - G3)).feedback(G4)          #Form the interconnections
print(G)
```

Listing 5.3. `FOTransFunc` block interconnection example

The output from the `Python` interpreter is as below:

```
                -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
-------------------------------------------------------------
 15s^{4.1}+15s^{4}+15s^{3.6}+15s^{3.5}+s^{3.1}+s^{3}+16s^{2}
 +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}
```

Table 1. `FOTransFunc` block interconnection operators

OPERATOR	DESCRIPTION	SYNTAX
feedback()	System negative feedback	G.feedback(G_2)
1/G	System inverse $\frac{1}{G}$	G^{-1}
–	System subtraction	$G - G_2$
**	Power of given system ($n \in \mathbb{Z}$)	G**n
/	System division (series connection)	G / G_2
*	System multiplication (series connection)	G * G_2
+	System addition (parallel connection)	G + G_2
–	Unary minus	$-G$

5.3.4 Stability Analysis: `FOTransFunc.isstable()`

With an instance of the `FOTransFunc` class, the user can perform stability analysis (discussed in Section 2.4.1) using the `.isstable()` method. Consider a `FOTransFunc` class instance `G` in Syntax below.

Syntax

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

or

```
[Stable,q,err,apol] = G.isstable()
```

Arguments

- `doPlot` – `True` or `False`. The user can specify if they wants a plotted Riemann sheet or not. Default value is `True`, if `doPlot` is not specified.

Return

- `Stable` – bool, it is set to `True` if system is stable, else `False`.
- `q` – Pole polynomial commensurate order, The minimum allowed commensurate-order is `q = 0.01`, if the commensurate order for the system is lower than `q = 0.01`, the stability analysis is done with this order.
- `err` – error norm.
- `apol` – minimum stability criterion.

5.3.5 Time Domain Analysis: `FOTransFunc.step()`

With an instance of the `FOTransFunc` class, the user can perform time domain analysis (discussed in Section 2.4.2) using the `.step()` method to calculate the step response of the instance. It uses the Grünwald-Letnikov definition computing algorithm described earlier in Definition 2.1.1.2. Consider an instance `G` of the `FOTransFunc` class in Syntax below.

Syntax

```
[t, y] = G.step(t=None, u=None, output=True, plot=True)
```

or

```
y = G.step(t, u=None, output=True, plot=True)
```

Arguments

- `t` – time vector `t` (optional) consisting of regularly spaced time samples such that `t`$= [0, t_{final}]$ with constant step δt. if not specified, method generate a time vector automatically.
- `u` – input signal vector `u(t)` (optional). if not specified, the method also generates an input vector of 1s' same length with that of the time vector.
- `output` – bool, set to `True` if you want a the computed output and `False` if computed outputs are not needed.
- `plot` – bool, set to `True` if you want a plot and `False` if plots are not needed.

Note 5.3.1 *Using this method means you either want a computed step* `output` *or a* `plot`*, Therefore, the user must set one of these to be* `True`*. The methods throws an error if these criteria is not met.*

Return

- `y` – vector of computed outputs. This is returned only if the arguments `output` is `True`.
- `t` – automatically computed time vector. This is returned only if the arguments `output` is `True` and input signal vector `u(t)` was automatically generated by the method.

5.3.6 Frequency Domain Analysis: `FOTransFunc.freqresp()`

With an instance of the `FOTransFunc` class, the user can perform Frequency domain analysis (discussed in 2.4.3) using the `.freqresp()` method to calculate the frequency response of the instance. It uses the Grünwald-Letnikov definition computing algorithm described earlier in Definition 2.1.1.2. Consider an instance `G` of the `FOTransFunc` class

in Syntax below.

Syntax

- `[magnitude, phase, w] = G.freqresp(minExp, maxExp, numPts)`
- `[magnitude, phase, w] = G.freqresp()`
- `G.freqresp(minExp, maxExp, numPts)`
- `G.freqresp()`

Arguments

- `minExp` – an `int` < 0. The minimum exponent of Frequency to compute. The actual computed frequency is 10^{minExp} rad/s.
- `maxExp` – an `int` > 0. The maximum exponent of Frequency to compute. The actual computed frequency is 10^{maxExp} rad/s.
- `numPts` – an `int` > 0. Number of points between the minimum frequency and maximum frequency.

Note 5.3.2 *The frequency vector ω used in the frequency response plot is obtained by using the `numpy.logspace()` function. If no arguments are supplied by the user, the method uses its defaults.*

Return

- `magnitude` – vector of computed magnitudes in decibel [dB].
- `phase` – vector of phase in degree [°].
- `w` – vector of the frequencies [rad/s].

5.3.7 Oustaloop Approximation: `FOTransFunc.oustaloop()`

With an instance of the `FOTransFunc` class, the user can compute the oustaloop filter approximation (discussed in 2.5) using the `.oustaloop()` method. The approximation is

only valid in the specified frequency range. Consider an instance `G` of the `FOTransFunc` class in Syntax below.

Syntax

- `TransferFunction = G.oustaloop(wb, wh, Order)`
- `G.oustaloop()`

Arguments

- `wb` – an `int` < 0. The minimum exponent of Frequency to compute. The actual computed frequency is 10^{wb} rad/s. Default value = -4.
- `wh` – an `int` > 0. The maximum exponent of Frequency to compute. The actual computed frequency is 10^{wh} rad/s. Default value = 4.
- `Order` – an `int` > 0. Approximation Order.

Return

- `TransferFunction` - An integer order transfer function of type `control.xferfcn.Transfe` based on the `Python control` library.

Example

Consider a fractional-order transfer function $G(s) = s^{0.5}$. Let's approximate it using oustaloop filter within the frequency range $\omega = [0.01, 100]$ with an order $N = 2$. The following command can be entered:

```
import fotf
g=newfotf("s^0.5","1")
print(g.oustaloop(-2,2,2))
```

Listing 5.4. `FOTransFunc` Oustaloop approximation example

The output on the *Python* interpreter is as follows:

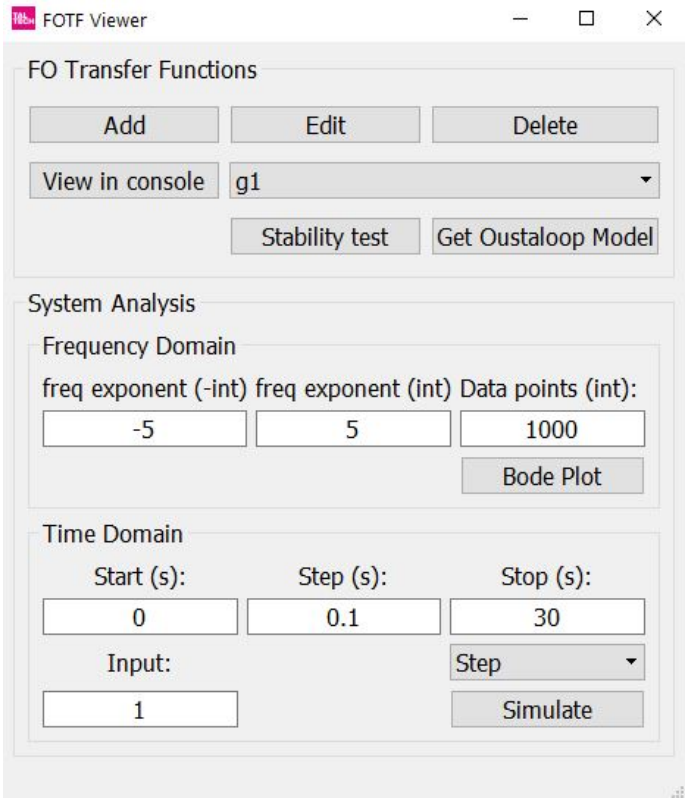


Figure 10. Fractional-order System Analysis Module GUI

```
10s^{5}+298.5s^{4}+1218s^{3}+768.5s^{2}+74.97s+1
-----------------------------------------------
s^{5}+74.97s^{4}+768.5s^{3}+1218s^{2}+298.5s+10
```

5.3.8 Fractional-order System Analysis Module GUI

The system analysis GUI is equipped with functions to analyse fractional-order system in one click. We can call it a user friendly "FOTransFunc" class. To get started in the same command prompt or terminal from Section 5.2, run the command:

```
python pyfomcon.py
```

The FOSAM GUI, as shown in Fig.10 has 2 group panels:

- **FO Transfer Function Panel**: This panel provides buttons that add/create, edit, delete, view the model in *Python* interpreter, check stability and convert the fractional-

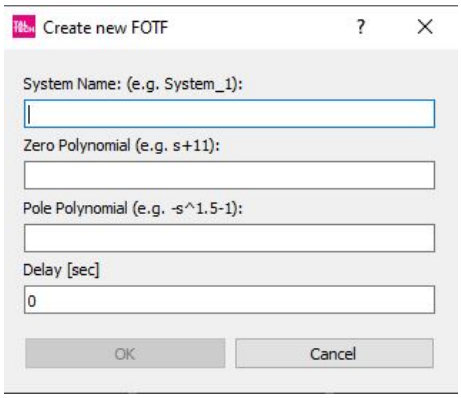
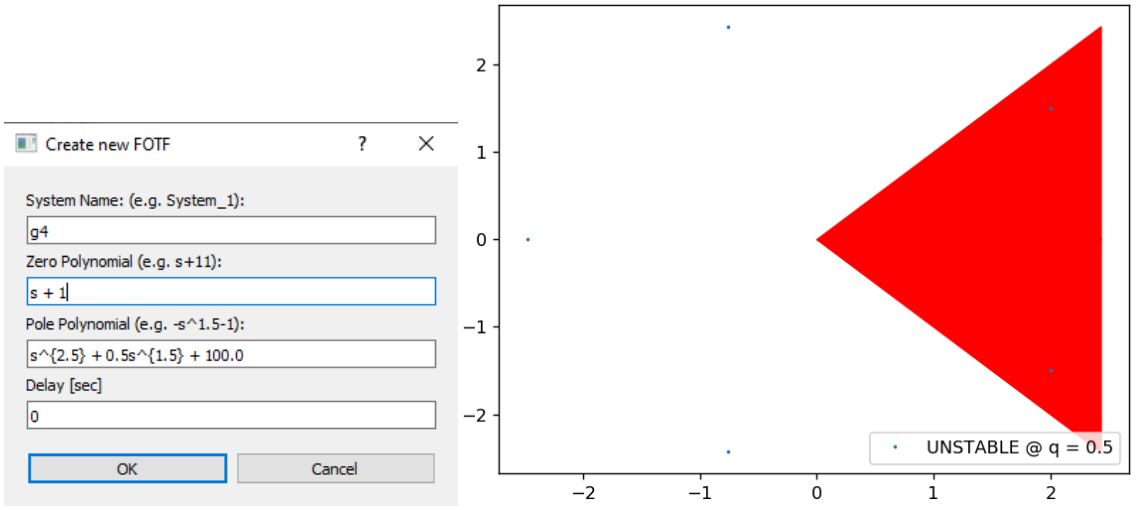


Figure 11. FOTransFunc Viewer Window

order system selected in the drop-down list. Fractional-order transfer functions may be added to the drop-down list by clicking the `Add` button. A new dialog, Fig. 11 allows the user to enter a unique name, zero polynomial, pole polynomial and delay for the intended fractional-order system. Multiple fractional-order systems can be added to the drop-down list. The selected system can be edited, deleted, oustaloop filtered, stability checked and further analysed using features available in the `System Analysis` panel.

- **System Analysis Panel**: This Section of the library offers function for both frequency and time domain analysis.
  - **Frequency Domain Analysis**: The `Bode Plot` button is used for this analysis. It is important to note that only integer `exponent` can be used to specify upper and lower frequency bounds. It calculates the frequency response of the system. The frequency response carries information about how the amplitude and phase of the signal passing through the system changes with frequency and can be computed by substituting $j\omega$ for s. This was already discussed in Section 2.4.3.
  - **Time Domain Analysis**: The `Simulate` button is used for this analysis. It is common sense that the start time t_{start} and the stop time t_{stop} must be positive integers and $t_{start} < t_{stop}$. The objective of analysis in time domain is to obtain the transient response of the system. For the fractional-order system, the numerical computation of fractional derivative using a revised



(a) Adding a FOTransFunc (b) System `g4` is unstable

Figure 12. Adding system `g4` and its Stability check plots

Grünwald-Letnikov definition was used. We already discussed this in Section 2.4.2.

Example

Consider the system given by:

$$G(s) = \frac{s+1}{s^{2.5} + 0.5s^{1.5} + 100} \tag{5.1}$$

To add this system as `g4`, one would fill in the systems details after clicking the `Add` button as shown in Fig. 12a. The user should note that the `OK` button will not be available until all text are filled. We can check the stability of the system as illustrated in Fig. 12b. It can be seen that two poles are inside the red shaded region, thus the system is unstable because the condition in equation (2.21) is not satisfied. In the plot we can see the legend that shows the commensurate order and says system is unstable.

The step response in time range $t = [0, \ldots, 30]$ with a step of $h = 0.1$ is obtained using the computation described in Section 2.4.2. The step response plot is depicted in Fig. 13. The Bode plot within frequency range $[0.00001, 100000]$ rad/s is shown in Fig. 14.

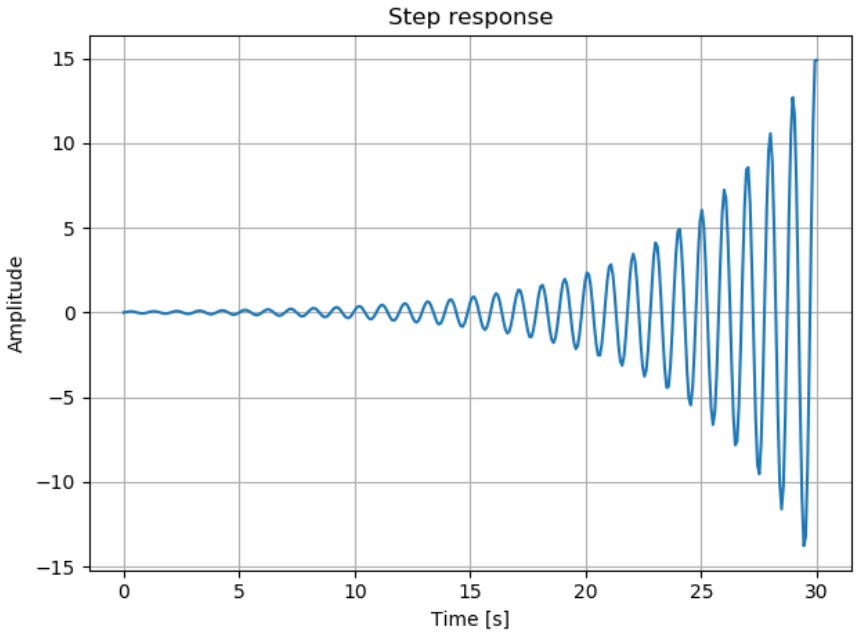


Figure 13. Step response of system `g4`: The system is unstable in accordance with the stability analysis result.

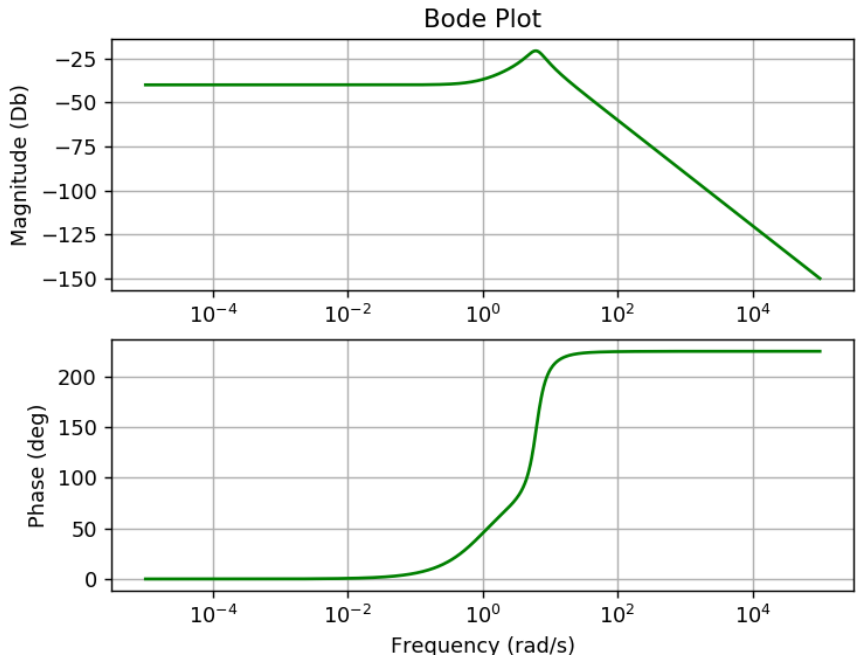


Figure 14. Frequency response of `g4`

	A	B	C	D
1	y	u	t	
2	0	0	0	
3	0.003631262	0.005	0.05	
4	0	0.01	0.1	
5	0	0.015	0.15	
6	0	0.02	0.2	
7	0.000293062	0.025	0.25	
8	0	0.03	0.3	
9	0	0.035	0.35	
10	0	0.04	0.4	
11	0.001470507	0.045	0.45	

Figure 15. Structure of identification/validation data with headings `y`, `u` and `t`

5.4 Fractional-order System Identification

The system identification module provides means for time-domain system identification. Using the identification module in the *Python* interpreter command line, user needs to import the `fomconoptimize` using the command:

- `from fomconoptimize import *`

5.4.1 Identification and Validation Data

Identification and validation data that can be used in the identification tool have specific data structure. To ease usage, Microsoft Excel file format (.xlsx and .xls) was selected. Data files are expected to have 3 columns with headings `u`, `y`, and `t`. Figure 15 shows an example data structure and as discussed in 3.2.1, Column `t` has a uniform sample rate of $0.05s$.

Note 5.4.1 *Ensure that the length of* `y` *= length of* `u` *= length of* `t` *in Data*

5.4.2 def `fid()`

This is the main Identification function. It tries to identify a fractional-order transfer function as described in Section 3.1.1. Non-linear least-square method discussed in Section

2.7.1 is employed, minimizing the squared error. The error is obtained by simulating the identified system in the time domain. The simulation type is determined by the `opt` class instance discussed in Section 5.4.3.

Syntax

```
result = fid(idData_s, valData_s, optiset, limits, plot, plotid, cleanDelay)
```

Arguments

- `idData_s` – String path to identification data. Best to put file in fomcon main directory.
- `valData_s` – String path to validation data. Best to put file in fomcon main directory.
- `optiset` - An `opt` class object and will be discussed in subsection 5.4.3.
- `limits` – A cell array with two vectors containing polynomial coefficient and exponent limits in the form `[[CMIN, CMAX], [EMIN, EMAX]]`. Default: None (that is, no limits are imposed).
- `plot` – An array of type bool with length=2. Example `[True,True]` where position 0 is to plot `idData_s` and position 1 is to plot `ValData_s`. Default: `[False, False]`.
- `plotid` – An array of type bool with length=2. Example given, `[True,True]` where position 0 is to plot identified model's output against `idData_s` and position 1 is to plot identified model's output against `valData_s`. Default is `[True,True]`.
- `cleanDelay` - Used to clean identification data with delays before identification. 2.5 is delay in seconds. Default: `[True,2.5]`.

Returns

- `result` - `fidOutput()` class object with 7 properties

- `result.G` - Identified system (A `FOTransFunc` object)
- `result.y` - output of `result.G` using input data `result.u`
- `result.u` - *idData.xlsx*$_s$ input column `u`
- `result.t` - *idData.xlsx*$_s$ input column `t`
- `result.vy` - output of `result.G` using input data `result.vu`
- `result.vu` - *ValData.xlsx*$_s$ input column `u`
- `result.vt` - *ValData.xlsx*$_s$ input column `t`

5.4.3 def `opt()`

This is an option class. It contains the user specified identification option settings that will be used by the `fid()` function already discussed in Section 5.4.2.

Syntax

```
options = opt(initialGuess, simType, optiAlg, optiFix, polyFix,
optidelay = False)
```

Arguments

- `initialGuess` - initial guessed fractional-order system, that is, a `FOTransfunc` instance.
- `simType` - The method used in simulating the output of the identified system. User is to select 1 of 2 options namely:
  - simMethod.grunwaldLetnikov discussed in definition (2.1.1.2).
  - simMethod.oustaloop discussed in Section 2.5.
- `optiAlg` - The optimization algorithm to use. User is to select 1 of 4 options.
  - optAlgo.LevenbergMarquardt discussed in Section 2.7.3.
  - optAlgo.TrustRegionReflective discussed in Section 2.7.2.
  - optAlgo.Softl1 An extension of trust region reflective. See 'Notes' in [32].
  - optAlgo.RobustLoss An extension of trust region reflective. See 'Notes' in

[32].

- `optiFix` - User should select which parameter to optimize Coefficients or Exponents or both. See Section 3.2.3.
  - optFix.Free - Optimize both coefficients and exponents.
  - optFix.Exp- Static exponents, optimize coefficients.
  - optFix.Coeff - Static coefficients, optimize exponents.
- `polyFix` - a vector with two int values: [`FIXZERO`, `FIXPOLE`]. Position 0 is for Zero polynomial, postition `1` is for Pole polynomial. where `FIXZERO` or `FIXPOLE` can be 1, in which case the corresponding polynomial is static during identification, or 0 for unstatic. Note that in case `FIXZERO` = `FIXPOLE` = 1 the initial guess model will be immediately returned with no identification conducted. Default: [0; 0]
- `optidelay` - bool, Should the delay be optimized? `True` or `False`

5.4.4 Fractional-order System Identification Example Using Command Line

$$G(s) = \frac{-2s^{0.63} + 4}{2s^{3.501} + 3.8s^{2.42} + 2.6s^{1.798} + 2.5s^{1.31} + 1.5}. \tag{5.2}$$

Consider the system identification data `idenData.xlsx` and validation data `valiData.xlsx` in the *dataFiles* directory. To identify and validate a fractional model from this data in the *Python* interpreter, using (5.2) as initial guess, the command used is as follows:

```
from fomconoptimize import *
#create initial guess model
guess=newfotf("-2s^0.63+4","2s^3.501+3.8s^2.42+2.6s^1.798+2.5s^1.31+1.5
    ",0)
polyfixset = [0, 0]     #optimize both zero and pole
simType = simMethod.grunwaldLetnikov    #simulation method
algo = optAlgo.LevenbergMarquardt       #optimization Algorithm
free = optFix.Free      #optimize both coefficents and exponents
bounds = [[0, 20], [0, 10]]}
#create opt() class with optimization options and settings
```

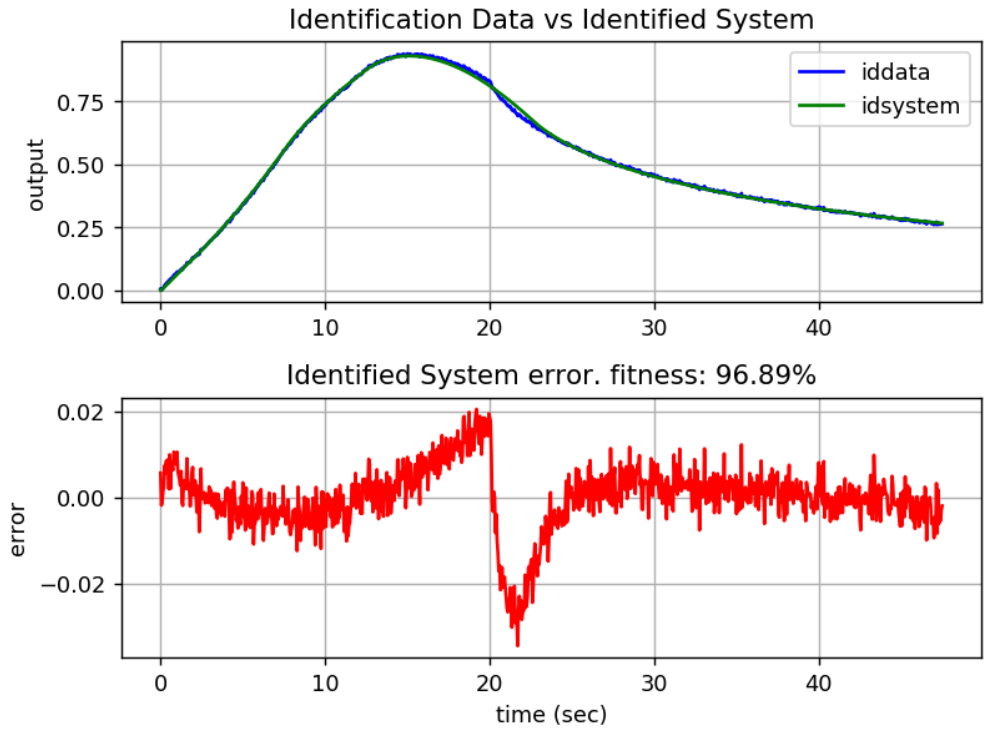


Figure 16. Identification Data vs Identified Model (5.3)

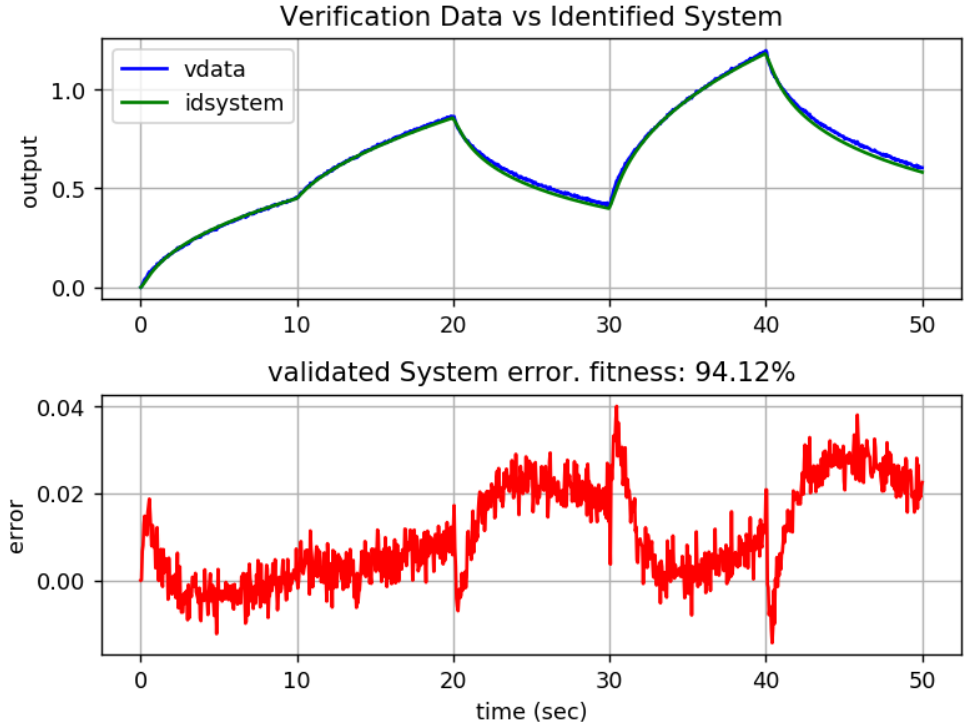


Figure 17. Validation Data vs Identified Model (5.3)

```
10 settings = opt(guess, simType, algo, free, polyfixset)
11 res=fid("dataFiles\idenData.xlsx", "dataFiles\ValiData.xlsx", settings,
       bounds, plot=[False,False], plotid=[True,True], cleanDelay=[True
      ,2.5])
12 print(res.G)
```

Listing 5.5. Fractional-order Identification example using Command prompt

The output on the python interpreter looks like this:

```
Please wait,  System Identification in progress...
Bounds will not be applied with Levenberg Marquardts
optimization algorithm
Time:  0:01:33.120218
```

$$\frac{-1.72s^{0.54}+3.92s^{0.67}}{0.84s^{2.29}+3.64s^{1.34}+2.03s^{2.22}+1.72s^{1.35}+4.01s^{1.33}} \tag{5.3}$$

Because the user specified to plot identified system against identification and validation data using option `plotid=[True, True]`, the plots are shown in Figure (16) and (17). Note that a preliminary analysis of the identification data was done, the option `cleanDelay = [True,2.5]` cleaned the first 2.5s of the identification data. Which is visible when Fig. 16 and Fig. 17 are compared. Then the question arises, what if we want to clean the last 2.5s of the data? This option is available in the fractional-order system identification GUI discussed in Section 5.4.7.

5.4.5 Identified System Comparison

Using same identification data from Section. 5.4.4 and initial guess (5.3) on FOMCONpy and FOMCON, system identification results were compared. with different identification parameter options (Exponents only, Coefficients only and Exponents + Coefficients) and two algorithms (LM and TRR). The performance criteria used was percentage fitness (3.8) discussed in Section 3.3. The validation plots and their identified systems can be found in

Table 2. Table comparing FOMCONpy vs FOMCON

	Exponents only	Coefficients only	Exponents + Coefficients
Levenberg Marquardt			
FOMCONpy	74.07	94.48%	91.87%
FOMCON	74.08%	93.97%	94.22%
Trust Region Reflective			
FOMCONpy	25.06%	94.08%	94.29%
FOMCON	25.00%	-74.44%	74.85%

Table 3. Computational Properties of Devices

Properties	Windows PC	Pi 4	Pi 3B
CPU (GHz)	2.7	1.5	1.2
CPU (No of Cores)	8	4	4
Memory (GB)	16	4	1
Memory (GHz)	2.4	2.4	0.9

appendix 1. From Table 2, we can see that *FOMCONpy* performs slightly better in more cases.

5.4.6 FOMCONpy on Windows PC vs Raspberry PI-4 vs Raspberry PI-3B

In Sec. 5.1, we indicated that FOMCONpy supports command-line for more advanced users. To demonstrate this, we compared the performance of FOMCONpy, running on Windows platform and on two IoT devices (Raspberry Pi 3B and Pi 4). The computational properties of the test devices are given in Tab. 3 for the readers perusal. Table 4 shows the time it takes to complete the identification process using initial guess (5.2) and `idenData.xlsx` stated in Sec. 5.4.4. Three parameter identification options were used in the comparison (Exponents only, coefficients only and both(Exponents + Coefficients)) along with two optimization algorithms: Leveberg Marquardt and Trust Region Reflective. In Table 4 considering the Exponents + Coefficients column, we can see that Windows PC is about 4 times faster than Pi 4 which is almost 3 times faster than Pi 3B.

Table 4. Identification time on Windows PC vs Raspberry PI-4vs Raspberry PI-3B

Levenberg Marquardt			
	Exponents only	Coefficients only	Exponents + Coefficients
Windows PC	22.671s	18.526s	25.813s
Raspberry Pi 4	42.628s	74.787s	115.669s
Raspberry Pi 3B	112.843s	196.648s	304.285s
Trust Region Reflective			
Windows PC	9.971s	8.526s	133.671s
Raspberry Pi 4	75.487s	29.124s	561.919s
Raspberry Pi 3B	200.454s	182.884s	1442.884s

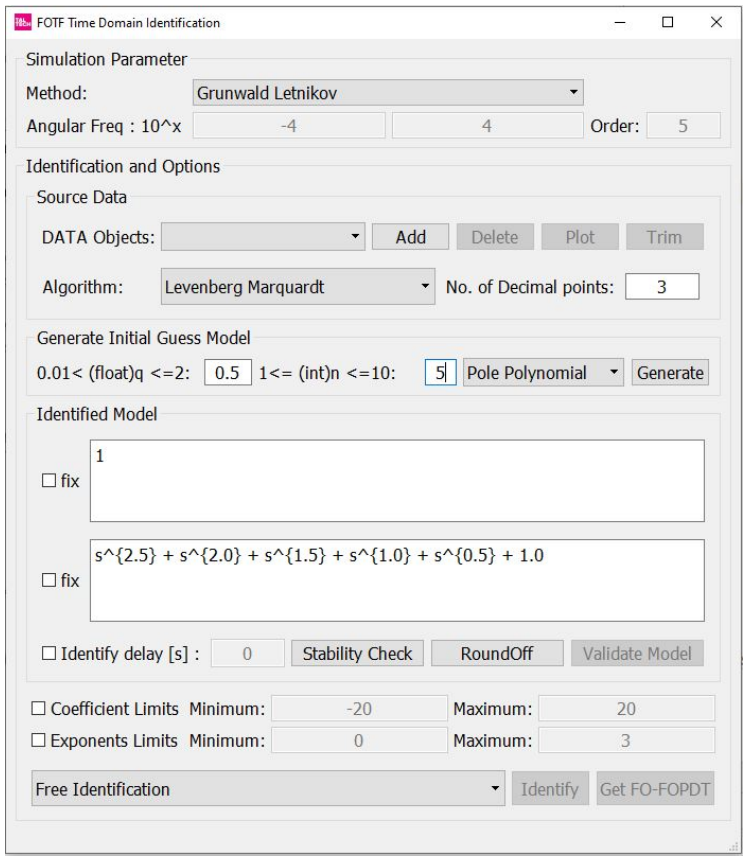


Figure 18. Fractional-order Time Identification GUI

5.4.7 Fractional-order System Identification GUI

The system identification GUI (see Fig. 18) is equipped with functions to obtain a fractional model in the form (2.17). It can be accessed by starting a command prompt, navigate to the directory of your downloaded FOMCONpy source code, then run the following commands:

```
python pyfotfid.py
```

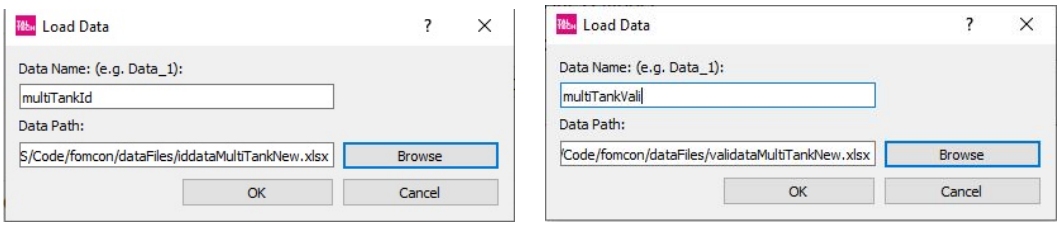
It is important to do some preliminary analysis of the data to exclude perhaps some delays. The `Source Data` group box, provide tools to `Add, Delete, Plot and Trim` data. Example is given in Section 5.4.8.

The user can generate an initial-guess model from a commensurate order q and the order `k` of the model of choice using the `Generate Initial Guess Model` group-box. Before starting the identification process, the user can specify what to optimize, only exponents, only coefficients or both (exponents + coefficients). When using LM algorithm, bounds are not used, but with Trust Region Reflective algorithm the user can specify bounds. Bound range for exponents must be positive while bounds for coefficients may be from negative to positive. *FOMCONpy* has checks to ensure these bounds are adhered to by the user. Best identification results are expected when the user already knows some properties of the system to be identified. If the properties of the system are unknown, the user could get a good result by trial and error when selecting the initial model guess. While the optimization process is running, so the user should wait for it to complete.

5.4.8 Fractional-order System Identification Example Using GUI

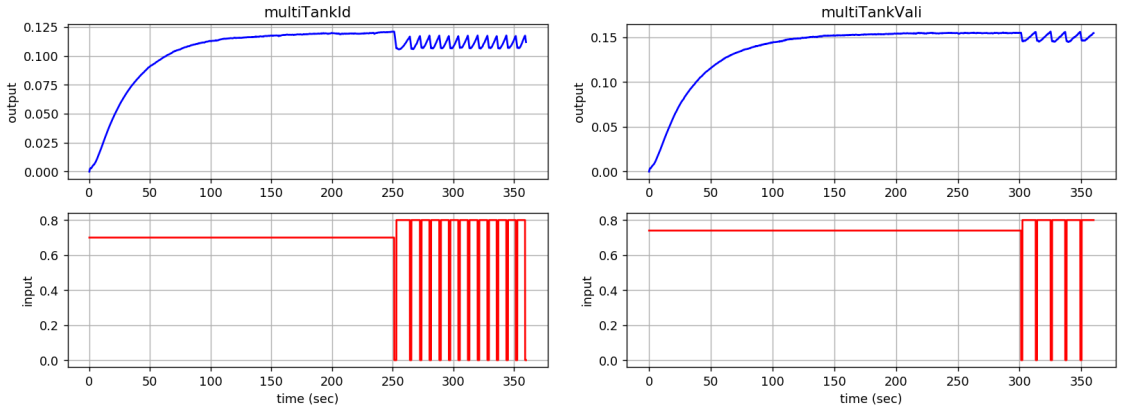
In `FOMCONpy`'s source code directory, there exist a `dataFiles` directory which contains the identification and validation data of the multi-tank system shown in Fig. 19 named: `iddataMultiTankNew.xlsx` and `validataMultiTankNew.xlsx` respectively. To identify and validate a fractional model from these data using the GUI we use the

Figure 19. Multi-tank System



(a) iddataMultiTankNew.xlsx (b) validataMultiTankNew.xlsx

Figure 20. Adding Identification and Validation Data using GUI

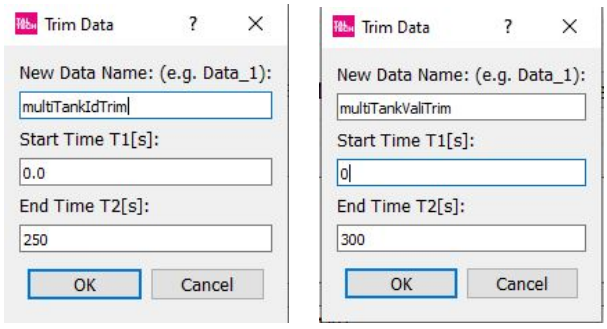


(a) multiTankId Plot (b) multiTankVali Plot

Figure 21. Plot Identification and Validation Data using GUI

following steps:

1. Click the `Add` button to add both data. A `Load Data` window pops out for the user to give a unique name and location of `.xlsx` file. In our case we use the names "multiTankId" and "multiTankVali". (See Fig. 20)
2. Click the `Plot` button to view the data maybe there is need to do some preliminary

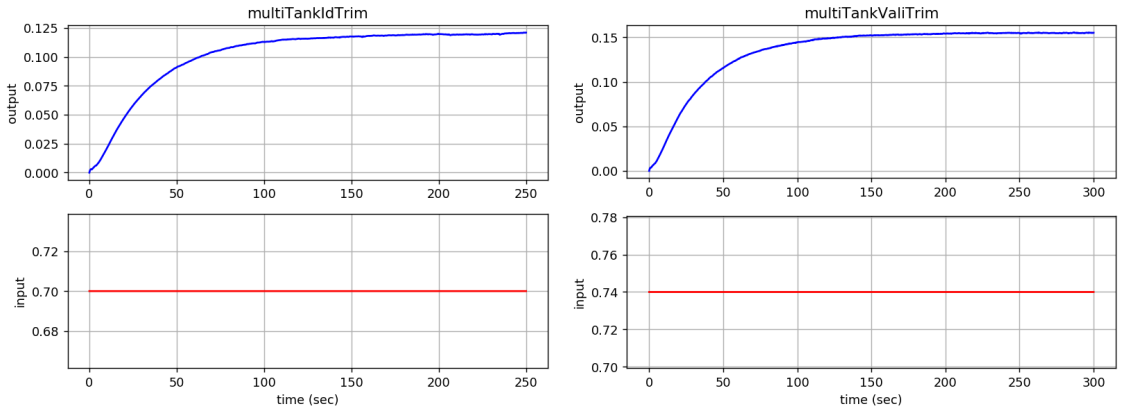


(a) Identification Trim Settings (b) Validation Trim Settings

Figure 22. Trimming Identification and Validation Data using GUI

analysis. From Fig. 21a we can see that after 250s, we have some irregularities for the Identification data. Likewise in Fig. 21b, we can see that from 300s, we have some irregularities. Thus there is need to trim the data.

3. To eliminate these irregularities from the data, we click the `Trim` button. A `Trim Data` dialog is loaded for the user to specify a new name, new start-time and new stop-time. In our case, "multiTankId" was trimmed to "multiTankIdTrim" with start and stop time set to 0s and 250s respectively (See Fig. 22a) while "multiTankVali" was trimmed to "multiTankValiTrim" with start and stop time set to 0s and 300s respectively (See Fig. 22b).
4. To View what the trimmed data looks like, select the appropriate data you want to view in the `DATA Objects` combo-box then click the `Plot` button. Fig. 23a shows what the data looks like after trimming.
5. We can use the `Delete` button to delete unused data from the `DATA Objects` combo-box.
6. Selete `multiTankIdTrim` as data to be used for identification from the `DATA Objects` combo-box.
7. Select the `Grunwald Letnikov` as the simulation method from the `Method` combo-box.
8. Select `Trust Region Reflection` as algorithm from the `Algorithm` combo-box.
9. We would like to use $\frac{1}{s^{0.5}+1.0}$ as an initial guess model. We can either use the `Generate Initial Guess Model` group box for this or we enter it manually in the Zero and Pole text-box.
10. Select `Free Identification` to optimise both coefficients and exponents
11. Click the `Identify` button to start the identification process. After identification, equation (5.4) was printed in the *Python* interpreter as the identified system. However in the system identification tool, the identified system is equation (5.5) because the



(a) multiTankIdTrim Plot (b) multiTankValiTrim Plot

Figure 23. Trimming Identification and Validation Data using GUI

.

user specified 3 in the `No. of decimal` text-box.

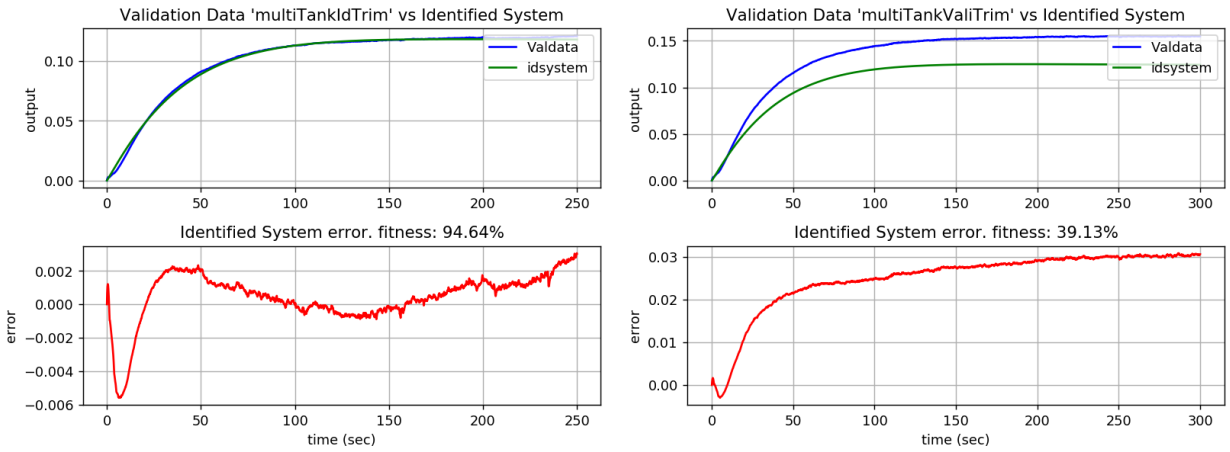
$$\frac{0.01310152}{3.495759s^{1.05779334}+0.07815751s^{1e-08}}. \tag{5.4}$$

$$\frac{0.013}{3.496s^{1.058}+0.078}. \tag{5.5}$$

12. If the identified model can be represented in a FO-FOPDT model (4.14) the `Get FO-FOPDT` button is automatically enabled. On clicked, the FO-FOPDT model is printed in the *Python* interpreter as shown in equation (5.6).

$$\frac{0.168}{44.727s^{1.058}+1.0}. \tag{5.6}$$

13. The identified system can be validate with the `validate` button. Validation plots are show in Fig. (24). It should be noted that the lower accuracy in the validation data is because identification data `multiTankIdTrim` has a set point of 0.125m while validation data multiTankValiTrim has a set point of 0.15m. Also the simulation is done with system inputs not error as we have in controllers.



(a) Eqn. (5.5) vs multiTankIdTrim

(b) Eqn. (5.5) vs multiTankValiTrim

Figure 24. Validating Identified Model using GUI

5.5 Fractional-order System Control Using Command Line

This module provides user with functions to design a fractional-order controller given a FO-FOPDT model. It contains method for tuning $\text{PI}^{\lambda}\text{D}^{\mu}$ controller with application to distributed system control.

5.5.1 def `mainFOFOPIDOPT()`

This function is used to tune the gain parameters of a given $\text{PI}^{\lambda}\text{D}^{\mu}$ controller using the method discussed in Section 4.3.

Syntax

```
tunedFOPID = mainFOFOPIDOPT(fofopdtModel, fopidGuess,
oustaloopOpt, designSpecs)
```

Argument

- `fofopdtModel` – The FO-FOPDT given by (4.14).
  - `fofopdtModel.K` – Gain.
  - `fofopdtModel.L` – Delay.

  - `fofopdtModel.T` – Time constant.
  - `fofopdtModel.alpha` – Order.
- `fopidGuess` - The FOPID initial guess parameters.
  - `fopidGuess.Kp` – Proportional Gain.
  - `fopidGuess.Ki` – Integral Gain.
  - `fopidGuess.Kd` – Differential Gain.
  - `fopidGuess.lam` – Order of the integral.
  - `fopidGuess.mu` – Order of the differential.
- `oustaloopOpt` – The oustaloop options
  - `oustaloopOpt.wb` – lower frequency bound (rad/s)
  - `oustaloopOpt.wh` – higher frequency bound (rad/s)
  - `oustaloopOpt.N` – Order (int)
  - `oustaloopOpt.Ts` – Sampling interval (s)
- `designSpecs` – The design specification
  - `designSpecs.wc` – critical frequency (rad/s)
  - `designSpecs.pm` – phase magine (rad/s)
  - `designSpecs.optnorm` – termination criteria (float)

Note 5.5.1 *The object type `Dict()` found in the Python library `addict` [49] is used to form unique dictionary object used in this function.*

Returns

- tunedFOPID - The tuned FOPID controller

Command Line Example

Consider the FO-FOPDT model $G(s)$ used in [44] given as:

$$G(s) = \frac{66.16\mathrm{e}^{-1.93s}}{12.72s^{0.5}+1} \tag{5.7}$$

To get a tuned FOPID controller based we this, we enter the following command

```
from fopidcontrol import fofopdtpidTuner as Tuner
#create oustaloop model
oustalModel=Dict(dict(wb=0.0001, wh=10000, N= 5, Ts= 0.01))
fopidGuessModel=Dict(dict(Kp=1, Ki=1, Kd=1, lam=0.9, mu=0.5))
#Design Specification
designSpec=Dict(dict(wc=0.1, pm=60, optnorm=0.001))
#FO-FOPDT Model
fofopdtModel=Dict(dict(K = 66.16, L = 1.93, T = 12.72, alpha = 0.5))
Tuner.ACTIVATETUNING=True #enable Tuning before starting the tuner
#Start the tuning process
tunedFOPID = Tuner.mainFOFOPIDOPT(fofopdtModel, fopidGuessModel,
    oustalModel, designSpec)
Tuner.ACTIVATETUNING = False    #disable Tuner after tuning is finished
```

Listing 5.6. FO-FOPDT tuning example using command prompt

The tuned $PI^{\lambda}D^{\mu}$ parameters obtained from *FOMCONpy* is the same as in [44]. The print out on the *Python* interpreter is as below:

```
Controller Tuning Started
The FOFOPDT Model:'K':66.16,'L':1.93,'T':12.72,'alpha':0.5
The Initial FOPID Guess:'Kp':1, 'Ki':1, 'Kd':1, 'lam':0.9,
'mu':0.5
The Tuned FOPID: 'Kp':-0.00293400435, 'Ki':0.01030466077,
'Kd':0.05335520401, 'lam':0.9, 'mu':0.5
Controller Tuninng Finished
```

5.6 Fractional-order Tuner and Controller GUI

This module provides user with functions to design a fractional-order controller given a FO-FOPDT model. It contains method for tuning $PI^{\lambda}D^{\mu}$ controller with application to distributed system control. To run the GUI client, start two command prompt in your downloaded *FOMCONpy* source code, then run the command 1 in the first command

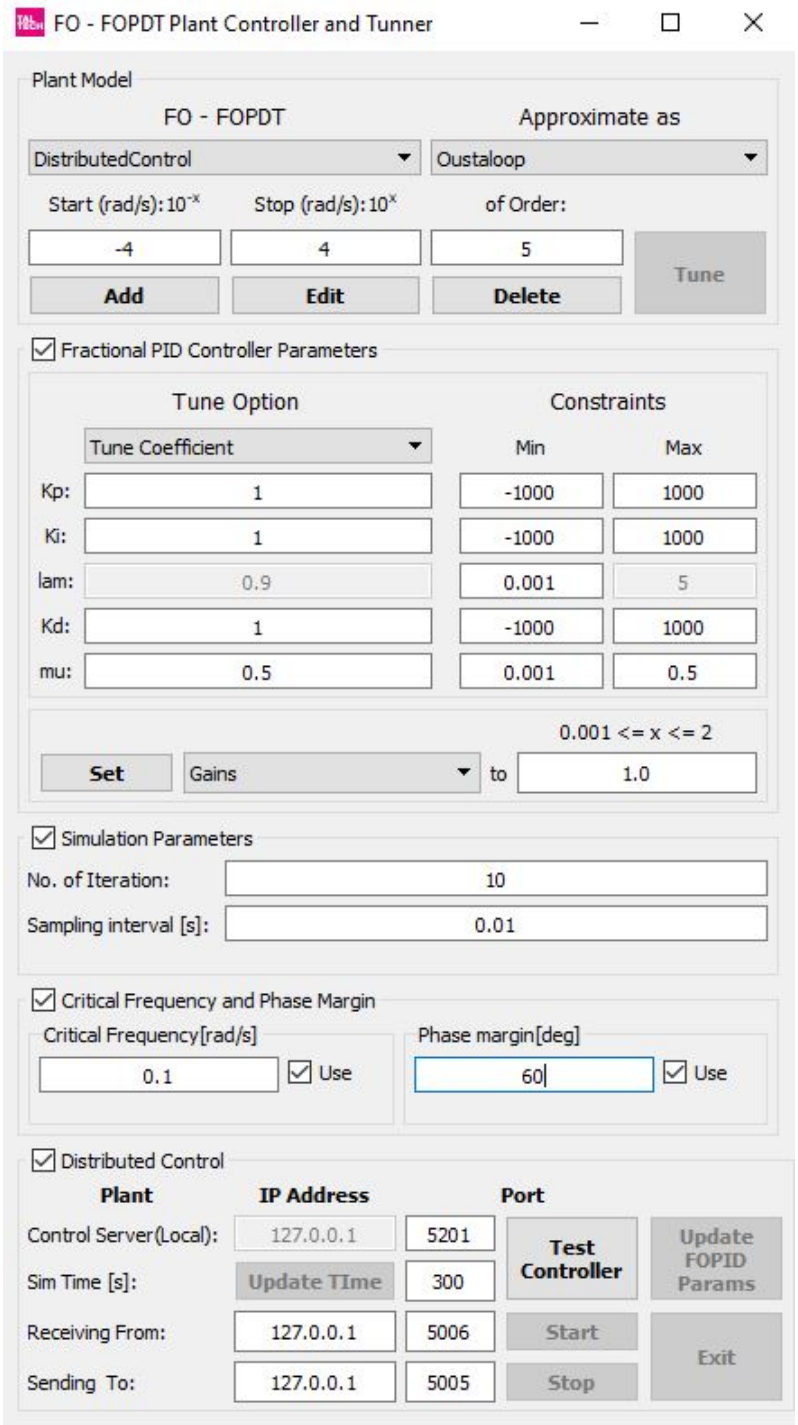


Figure 25. Fractional-order Controller and Tuner GUI

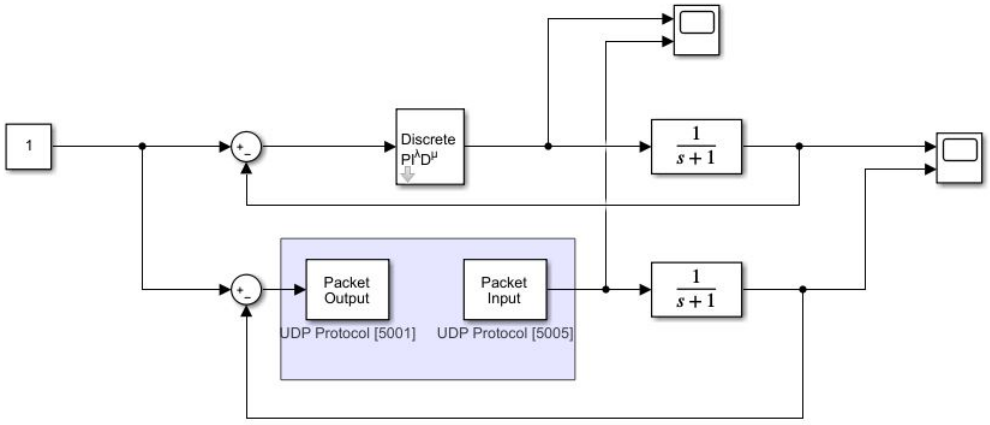


Figure 26. Simulink desktop real time model designed for testing purposes

prompt, then run command 2 in the second command prompt to start the server.

1. `python pyfopidopt.py`
2. `python fopidcontrol/controlServer.py`

The GUI is shown in Fig. 25. There are 5 group-boxes `Plant Model`, `Fractional PID Controller Parameter`, `Simulation Parameter`, `Critical Frequency and Phase Margin` and `Distributed Control`. The `Plant Model` group-box provides buttons for the user to add, edit, delete and tune FO-FOPDT plant models. it also provides text-boxes to fill in appropriate frequency ranges and order for oustaloop approximations.

The `Fractional PID Controller Parameter` provides text-boxes where the user can fill in initial $PI^{\lambda}D^{\mu}$ guess parameters before tuning. After tuning, these parameters are updated. There are other text-boxes to check if values obtained after tuning are within desired ranges. if there are not within ranges, the `Tune` button is automatically disabled. The initial guess of λ is automatically updated whenever the user selects or adds a new FO-FOPDT model.

The `Simulation Parameter` and `Critical Frequency and Phase Margin` group-boxes are used in the tuning process. the user can play play with their values especially the phase margin. The `Distributed Control` group-box is an important aspect of the GUI especially as control can only be done using the GUI (not

possible on command line). The user must click the `Test Controller` button to ensure communication between server and client is successful. Once its successful, the `Test Controller` button is disable and the `Update FOPID params` button is enabled. This button is used to send controller parameters to the control server.

5.6.1 Application to Distributed Control system

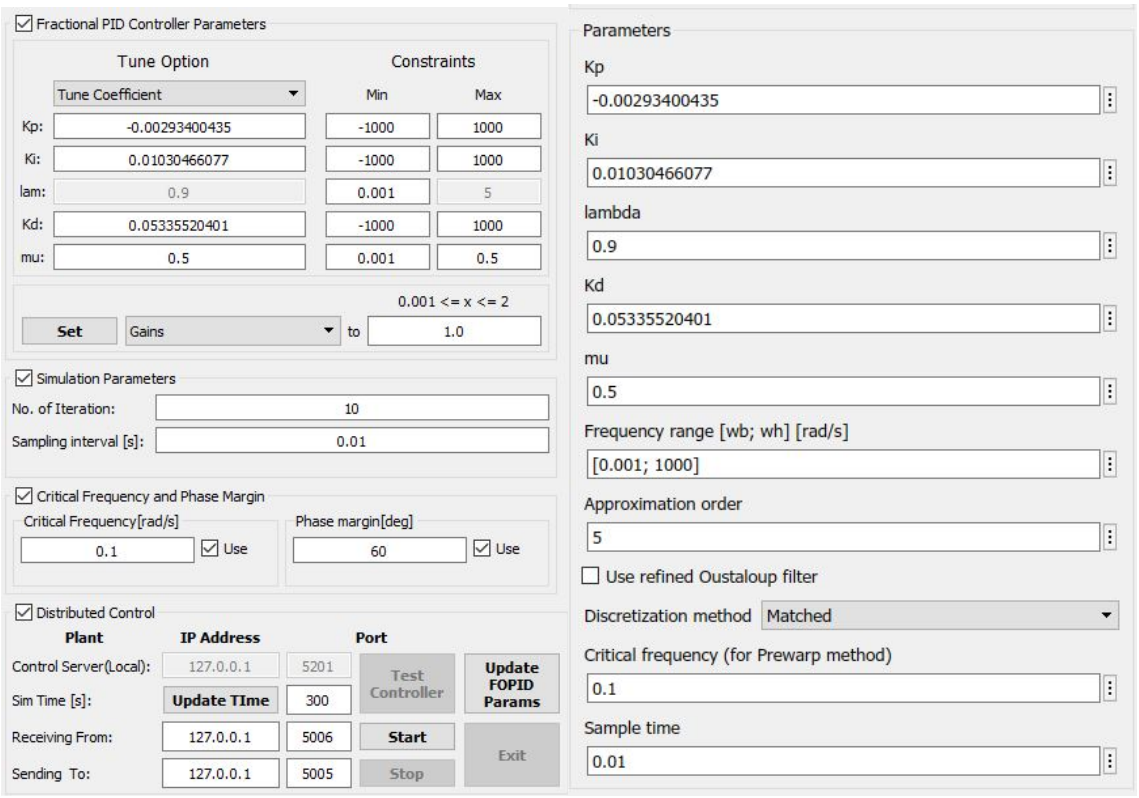
Consider a realtime simulink model in Fig. 26. We will attempt to control a system $\frac{1}{s+1}$ using MATLAB's PI$^{\lambda}$D$^{\mu}$ and compare with results using *FOMCONpy*'s PI$^{\lambda}$D$^{\mu}$. *FOMCONpy* receives UDP packets, computes the control law and sends back a UDP pocket.From Fig. 27 and 26 we can see that there is a similar UDP port 5005 which *Python* uses to communicate with the simulink model. The same tuned controller output from Section 5.5.1 example is setup in *FOMCONpy* and MATLAB PI$^{\lambda}$D$^{\mu}$ controller as shown in Fig. 27. The simulation was run for 300s and the output is shown in Fig. 28. We can see that the outputs are similar.

5.6.2 FOSCOM Application to IOT + Distributed Control

Recall the multi-tank system identification example in sec. 5.4.8. The identified FO-FOPDT system given by equation (5.6) was added and a tuned FOPID controller with the parameters below were loaded to the control server.

$$Kp : -3.38322,\ Ki : 5.36875,\ \lambda : 0.7,\ Kd : 21.73804,\ \mu : 0.5$$

The simulink model of the multi-tank system is show in Fig. 31. The *packet output* settings of the simulink model was as in Fig. 29a which is also highlighted yellow in Fig. 30. The *packet input* settings of the simulink model was as in Fig. 29b which is highlighted green in Fig. 30. The reference value was set to 0.13 and the process was run for 300s. From Fig. 32, it can be seen that raspberry Pi4 FOPID server controlled the system correctly with a settling time of 39.22s at 95% accuracy.



(a) `FOMCONpy` controller (b) MATLAB controller

Figure 27. Using same settings including the sampling time

.

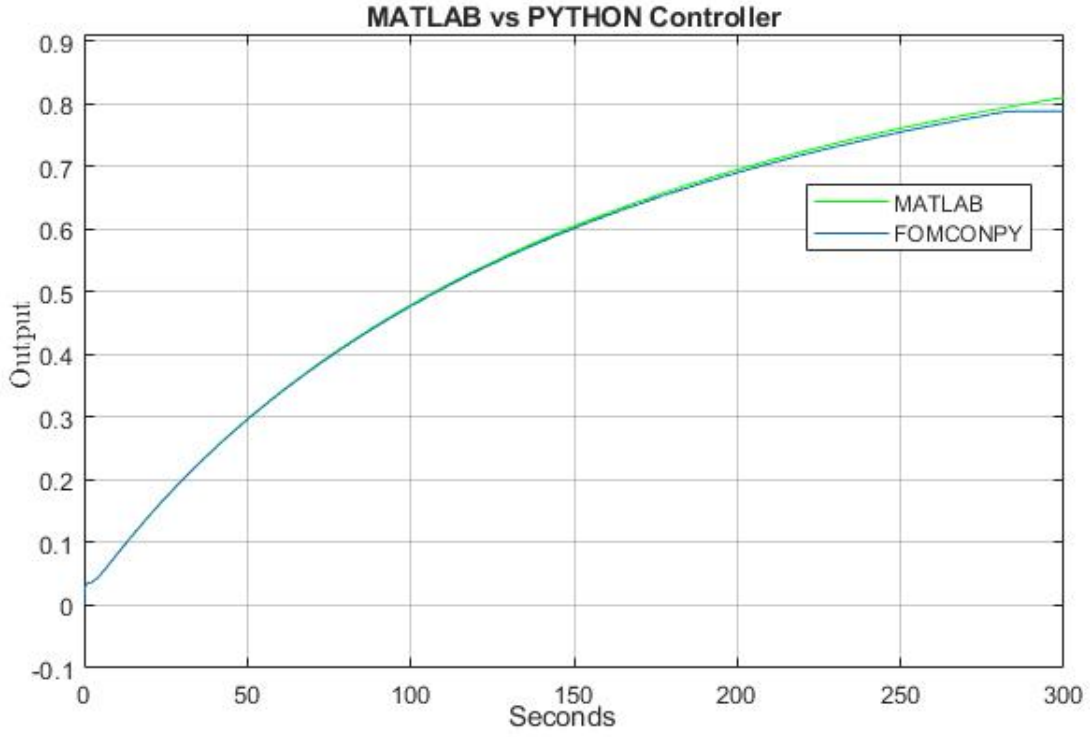
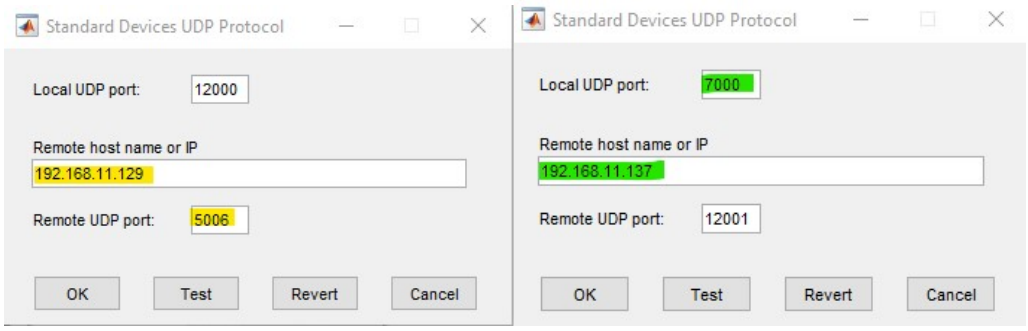


Figure 28. Real time control outputs MATLAB vs Python



(a) packet output setting (b) packet input setting

Figure 29. UDP IP and Port Settings

Figure 30. FOSCOM's DCS node with PI$^{\lambda}$D$^{\mu}$ parameters implemented on Raspberry Pi 4

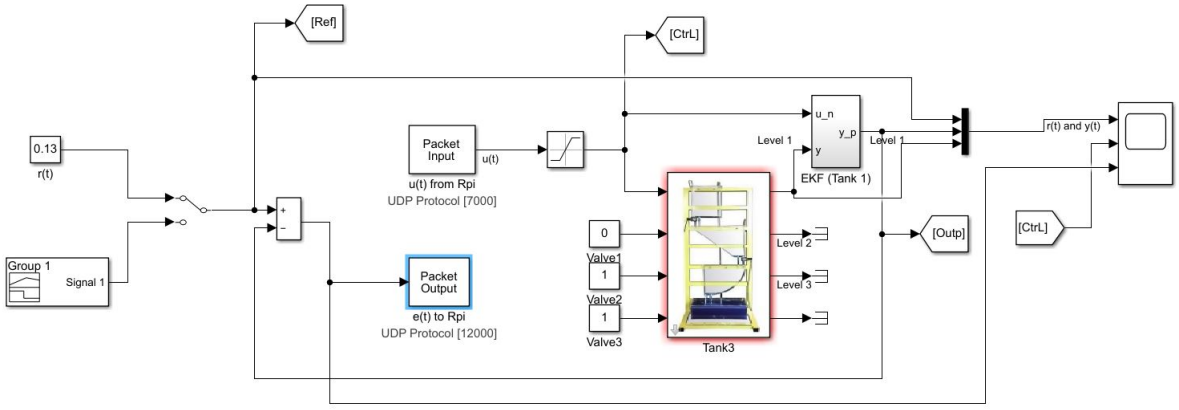


Figure 31. Simulink model connecting the DCS node with the FOPID controller implemented on Raspberry Pi with the laboratory model of the multi-tank system

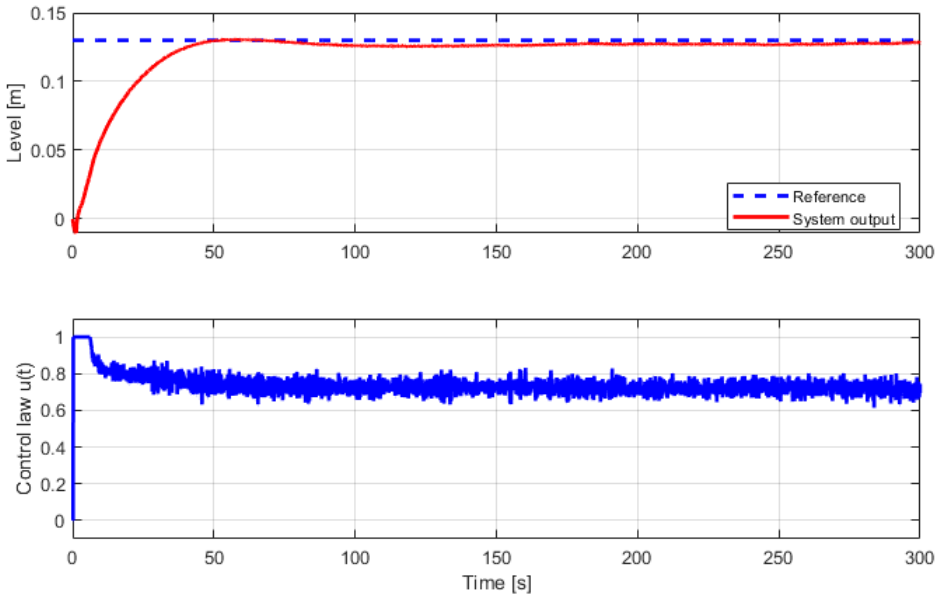


Figure 32. Multi-Tank output with single setpoint

5.6.3 FOSCOM Application to IOT

The author also conducted experiment using `FOMCONpy` running on a raspberry pi 4 device with disturbance using same multi-tank system in Fig. 19 over a wireless local area network (WLAN). Two experiments types were conducted:

1. multiple set-points
2. single set-point + on the spot controller parameter change:

It is observed that the system was able to settle within 100s after the induced disturbance for all scenarios.

Multiple set-points

Controller parameters were set as follows:

$$Kp : -3.38322,\ Ki : 5.36875,\ \lambda : 0.7,\ Kd : 21.73804,\ \mu : 0.5.$$

The signal builder in 31 was used to generate the input signal with 3 reference points (0.13m, 0.05m, 0.1m). The reference point is changed after 100s. From Fig. 33, we can

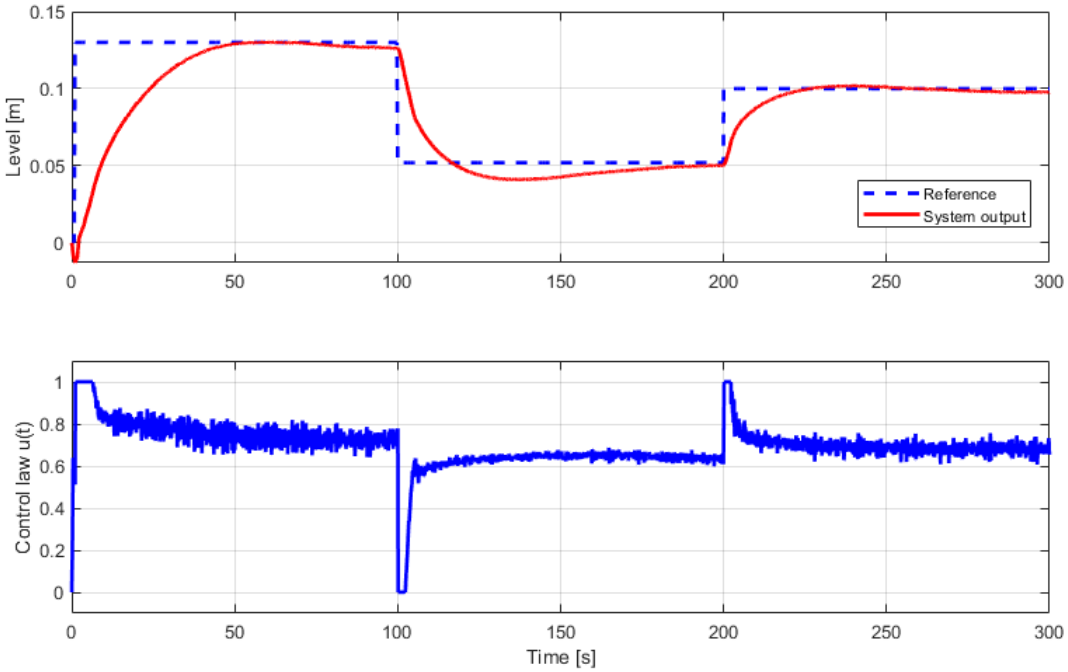


Figure 33. Multi-Tank output with multiple set-points

see that the settling time for the first reference 0.13m was 39.78s. Decreasing the reference to 0.05m had a 26.46% undershoot but settled after 85.36s. Increasing the reference to 0.1m settled after 16.93s.

Single setpoint + on the spot controller parameter change

The first controller parameters were set as follows:

$$Kp: -3.38322,\ Ki: 5.36875,\ \lambda: 0.7,\ Kd: 21.73804,\ \mu: 0.5.$$

The set point was set to 0.13m using the manual switch in Fig. 31. The process was started and after 39.29s, the set-point was reached as shown in Fig. 34.

When the process had ran for 76s, a second controller parameters as below

$$Kp: -9.04369,\ Ki: 6.11986,\ \lambda: 0.7,\ Kd: 31.72628,\ \mu: 0.5$$

was loaded. It is noticed in 31 that the system set-point was achieved after 68.36s.

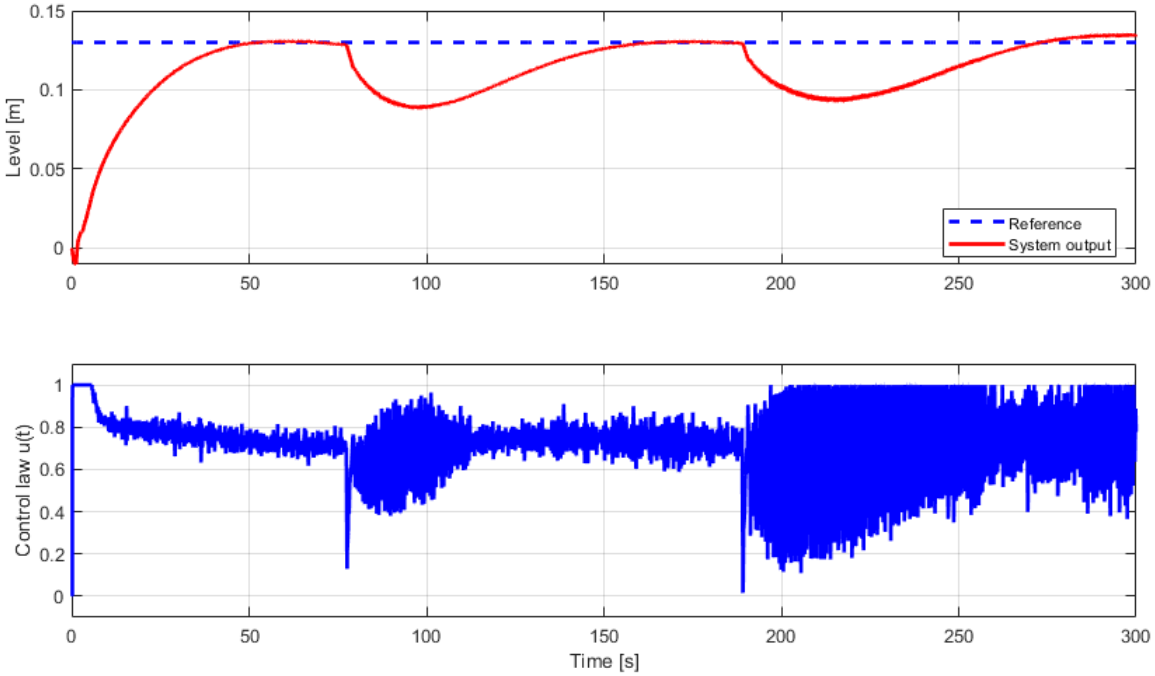


Figure 34. Multi-Tank output with on the spot Controller changed

When the process had run for 190s, a third controller parameter as shown below

$$Kp: -16.11473,\ Ki: 7.04386,\ \lambda: 0.7,\ Kd: 45.85597,\ \mu: 0.5$$

was loaded and the set-point was reached again after 70.7s. In Fig .34, it was observed that the control law was more noisy when the third controller was set which is because the differential components increased, thus emphasizes the noise. It can also be seen in the control law that the emphasis on noise increased progressively as the derivative components of the controllers increased from the first to third controller.

6. Conclusions

In this chapter, the studied problems, achieved results and further topic development perspectives is presented by the author.

The reader was introduced to system modelling, identification and control methods within the fractional-order calculus context. FOMCONpy, a new *Python* fractional-order system toolbox, in which these discussed methods have been implemented, was presented and example were given on how to use the library. The examples in this thesis can be said to be limited to relatively simple models. However, it is believed that the proposed methods will be effective for much more complicated cases if the processing speed of such complicated case is not paramount. Some advantages and drawbacks of each module is provided.

6.1 Fractional-order System Analysis Module

6.1.1 Advantages

- Provides sophisticated methods for fractional-order system modelling.
- Fully-featured for fractional-order transfer function analysis.
- Provides convenience methods, e.g. a graphical user interface for effective workflow.
- Integrated with the numpy, scipy and pandas which means highly effective algorithms are used for data processing.
- Block modules can be constructed using common operators (+,**,-,*)
- It is open source, so no need for premium licenses.
- Compatible with IOT devices and
- Supports object oriented programming

6.1.2 Drawbacks

- Limited to the single input-single output (SISO) linear time-invariant dynamic system case
- Currently has no support for direct fractional-order state-space system analysis
- There is **little** support for fractional-order system initial conditions (Using the commensurate generator button)
- Time-domain simulation may require a lot of computational resources depending on data size.
- No proper discretization method is implemented.
- Stability analysis has a fixed minimum precision 0.01 to solve the associated computational effort problem.

6.2 Fractional-order System Identification Module

6.2.1 Advantages

- Offers more accurate identification than the integer-order identification.
- Encompasses time-domain methods.
- Provides GUIs for all identification tools facilitating the initial guess model design, identification process and effective workflow.
- Provides 9 options to achieve the best possible results with the proposed methods.
- It is open source, so no need for premium licenses.
- Compatible with IOT devices

6.2.2 Drawbacks

- No frequency-domain identification methods.
- Time-domain identification may be slow and require a lot of computational resources depending on selected identification options.

- No automatic signal filtering feature is currently available, signal pre-filtering should be done or specified by the user.
- Time-domain simulation may require a lot of computational resources depending on data size.
- setting bounds usinf Levenberg Marquardt algorithm is not available
- Stability analysis has a fixed minimum precision 0.01 to solve the associated computational effort problem.

6.3 Fractional-order Control module

6.3.1 Advantages

- Offers methods for fractional-order PID controller design, tuning and optimization according to given specifications.
- Fast analytic tuning method for the fractional PID controller based on F-MIGO and the approach in Tepljakov et al., 2015 [44].
- Provides GUIs for Tuning, simulation and Distributed control for effective workflow.
- Tuned controller can handle fractional-order and integer-order systems.
- Controller tuning does not require additional *Python* library.

6.3.2 Drawbacks

- Controller optimization has limited support for all performance specifications.
- Security vulnerabilities between the server and client communication wasn't consider.
- Control system library for *Python* is not so robust like that of MATLAB as it is still in its alpha development phase

6.4 Research perspectives

With the above considerations, further research directions, involving the development of the FOMCONpy library, can be outlined:

- Possibility for fractional-order MIMO system modelling and control.
- Implement a tool for working with fractional-order state-space models.
- Search for a way to effectively implement the initial conditions problem.
- Investigate the implementation of security in real time control.
- 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 0.001. (*FOMCONpy* already can, but computation time is longer).
- Research the contribution of noise and disturbances to the identified model.
- Research on fractional-order PID tuning methods that takes less time.
- Porting the FOMCON toolbox to native C++ computing platform should also be considered for processing speed

6.5 Concluding comments

This thesis has illustrated the use of fractional-order calculus in dynamic system modelling,identification and control with application to distributed systems. The benefits of 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 fractional-order calculus will continue to gain more attention in the coming years, thus more efficient computational and analytical methods will be developed and the use of non-integer calculus will become standard practice in the industry come Industry 4.0.

Bibliography

[1] Y. Chen, I. Petras, and D. Xue, "Fractional order control - a tutorial", in *2009 American Control Conference*, IEEE, 2009.

[2] A. Tepljakov, "Fractional-order modeling and control of dynamic systems", PhD thesis, 2017.

[3] F. Padula and A. Visioli, *Advances in Robust Fractional Control*. Springer International Publishing, 2015.

[4] "Chapter 3 ordinary fractional differential equations. existence and uniqueness theorems", in *Theory and Applications of Fractional Differential Equations*, ser. North-Holland Mathematics Studies, A. A. Kilbas, H. M. Srivastava, and J. J. Trujillo, Eds., vol. 204, North-Holland, 2006, pp. 135–219. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304020806800046.

[5] F. Liu, O. P. Agrawal, S. Momani, N. N. Leonenko, and W. Chen, "Fractional differential equations 2012", *International Journal of Differential Equations*, vol. 2013, pp. 1–2, 2013.

[6] I. Podlubny, *Fractional Differential Equations*. Elsevier Science Publishing Co Inc, Oct. 27, 1998, 340 pp., ISBN: 0125588402. [Online]. Available: https://www.ebook.de/de/product/3646779/igor_technical_university_of_kosice_slovak_republic_podlubny_fractional_differential_equations.html.

[7] "Chapter 2 fractional integrals and fractional derivatives", in *Theory and Applications of Fractional Differential Equations*, ser. North-Holland Mathematics Studies, A. A. Kilbas, H. M. Srivastava, and J. J. Trujillo, Eds., vol. 204, North-Holland,

2006, pp. 69–133. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304020806800034.

[8] M. D. Ortigueira, *Fractional Calculus for Scientists and Engineers*. Springer Netherlands, 2011.

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

[10] "Chapter 4 methods for explicitly solving fractional differential equations", in *Theory and Applications of Fractional Differential Equations*, ser. North-Holland Mathematics Studies, A. A. Kilbas, H. M. Srivastava, and J. J. Trujillo, Eds., vol. 204, North-Holland, 2006, pp. 221–277. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304020806800058.

[11] K. Liu, Y. Chen, P. Domański, and X. Zhang, "A novel method for control performance assessment with fractional order signal processing and its application to semiconductor manufacturing", *Algorithms*, vol. 11, no. 7, p. 90, Jun. 2018.

[12] Z. Li, Y. Lan, G. He, S. He, and S. Wang, "Chipless RFID tag anti-collision algorithm based on FRactional fourier transform", in *2018 Chinese Automation Congress (CAC)*, IEEE, Nov. 2018.

[13] B. Meghni, D. Dib, A. T. Azar, S. Ghoudelbourk, and A. Saadoun, "Robust adaptive supervisory fractional order controller for optimal energy management in wind turbine with battery storage", in *Fractional Order Control and Synchronization of Chaotic Systems*, Springer International Publishing, 2017, pp. 165–202.

[14] A. Tepljakov, E. Petlenkov, and J. Belikov, "A flexible MATLAB tool for optimal fractional-order PID controller design subject to specifications", *Proceedings of the 31st Chinese Control Conference*, pp. 4698–4703, 2012.

[15] B. B. Alagoz, A. Tepljakov, C. Yeroglu, E. Gonzalez, S. H. HosseinNia, and E. Petlenkov, "A numerical study for plant-independent evaluation of fractional-order pid controller performance11this study is based upon works from cost action

ca15225, a network supported by cost (european cooperation in science and technology).", *IFAC-PapersOnLine*, vol. 51, no. 4, pp. 539–544, 2018, 3rd IFAC Conference on Advances in Proportional-Integral-Derivative Control PID 2018, ISSN: 2405-8963. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405896318304488.

[16] Q. Zou, J. Zhang, R. Lu, and R. Zhang, "Design of fractional order control using predictive functional control structure", in *2016 35th Chinese Control Conference (CCC)*, IEEE, Jul. 2016.

[17] N. Liu and J. Fei, "Adaptive fractional sliding mode control of active power filter based on dual RBF neural networks", *IEEE Access*, vol. 5, pp. 27 590–27 598, 2017.

[18] Y. Zhang and J. Li, "Fractional-order PID controller tuning based on genetic algorithm", in *2011 International Conference on Business Management and Electronic Information*, IEEE, May 2011.

[19] X. Dingyü, *Fractional-Order Control Systems: Fundamentals and Numerical Implementations*, ser. Fractional Calculus in Applied Sciences and Engineering. De Gruyter, 2017, ISBN: 9783110497977.

[20] D. Valerio. (Sep. 2005). Ninteger toolbox for matlab, [Online]. Available: http://web.ist.utl.pt/duarte.valerio/ninteger/ninteger.htm.

[21] A. Tepljakov, "Fractional-order calculus based identification and control of linear dynamic systems", Master's thesis, Tallinn University of Technology, 2011.

[22] C. Kuhlins, B. Rathonyi, A. Zaidi, and M. Hogan, "Cellular networks for massive iot", Ericsson, Tech. Rep., Jan. 2020. [Online]. Available: https://www.ericsson.com/en/reports-and-papers/white-papers/cellular-networks-for-massive-iot--enabling-low-power-wide-area-applications.

[23] J. White. (Apr. 2019). One year in: How our $5b investment in iot and intelligent edge is accelerating customer, partner and solution innovation. Last accessed:10-Feb-2020, [Online]. Available: https://blogs.microsoft.com/blog/

2019/04/04/one-year-in-how-our-5b-investment-in-iot-and-intelligent-edge-is-accelerating-customer-partner-and-solution-innovation/.

[24] Avnet. (Mar. 2017). Study: Digital intelligence driving massive business transformation. Last accessed: 09-Feb-2020, [Online]. Available: https://www.avnet.com/wps/portal/us/resources/article/study-digital-intelligence-driving-massive-business-transformation/.

[25] C. Ozgur, T. Colliau, G. Rogers, Z. Hughes, and B. Myer-Tyson, "C. ozgur, t. colliau, g. rogers, z.hughes & b. myer-tyson "matlab vs. python vs. r" journal of data science vol. 15 no. 3 pp. 355-372", *Journal of data science: JDS*, vol. Vol 15, pp. 355–372, Jul. 2017.

[26] A. Oustaloup, P. Melchior, P. Lanusse, O. Cois, and F. Dancla, "The CRONE toolbox for Matlab", in *CACSD. Conference Proceedings. IEEE International Symposium on Computer-Aided Control System Design (Cat. No.00TH8537)*, IEEE, Sep. 2000, pp. 190–195.

[27] L. Ljung, *System Identification: Theory for the User, 2nd Edition*. Prentice Hall PTR, Upper Saddle River, NJ 07458, USA, Sep. 1, 1999, 658 pp., ISBN: 0136566952. [Online]. Available: https://www.ebook.de/de/product/3240787/lennart_ljung_system_identification.html.

[28] R. Magin, M. D. Ortigueira, I. Podlubny, and J. Trujillo, "On the fractional signals and systems", *Signal Processing*, vol. 91, no. 3, pp. 350–371, Mar. 2011.

[29] B. Vinagre, I. Podlubny, A. Hernández, and V. Feliu, "Some approximations of fractional order operators used in control theory", *Fractional Calculus Applied Analysis (FCAA)*, vol. 3, Jan. 2000.

[30] A. Oustaloup, F. Levron, B. Mathieu, and F. Nanot, "Frequency-band complex noninteger differentiator: Characterization and synthesis", *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 1, pp. 25–39, 2000.

[31] D. Xue, Y. Chen, and D. P. Atherton, *Linear Feedback Control-Analysis and Design, Ch 8*. Society for Industrial and Applied Mathematics, Jan. 2007.

[32] S. Library. (Apr. 2020). Scipy.optimize.least_squares. LeastSquares function in scipy library, [Online]. Available: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html`.

[33] J. J. Moré and D. C. Sorensen, "Computing a trust region step", *SIAM Journal on Scientific and Statistical Computing*, vol. 4, no. 3, pp. 553–572, Sep. 1983.

[34] T. F. Coleman and Y. Li, "An interior trust region approach for nonlinear minimization subject to bounds", *SIAM Journal on Optimization*, vol. 6, no. 2, pp. 418–445, May 1996.

[35] M. A. Branch, T. F. Coleman, and Y. Li, "A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems", *SIAM Journal on Scientific Computing*, vol. 21, no. 1, pp. 1–23, Jan. 1999.

[36] J. J. Moré, "The levenberg-marquardt algorithm: Implementation and theory", in *Lecture Notes in Mathematics*, Springer Berlin Heidelberg, 1978, pp. 105–116.

[37] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters", *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963, ISSN: 03684245. [Online]. Available: `http://www.jstor.org/stable/2098941`.

[38] R. Malti, S. Victor, and A. Oustaloup, "Advances in system identification using fractional models", *Journal of Computational and Nonlinear Dynamics*, vol. 3, no. 2, Jan. 2008.

[39] A. Khadhraoui, K. Jelassi, J.-C. Trigeassou, and P. Melchior, "Identification of fractional model by least-squares method and instrumental variable", *Journal of Computational and Nonlinear Dynamics*, vol. 10, no. 5, Sep. 2015.

[40] N. S. Nise, *Control System Engineering, 7th Edition*. John Wiley Sons, 2015, ISBN: 978-1-118-80063-8.

[41] I. Podlubny, “Fractional-order systems and PI/sup /spl lambda//d/sup /spl mu//-controllers”, *IEEE Transactions on Automatic Control*, vol. 44, no. 1, pp. 208–214, Jan. 1999.

[42] M. Čech and M. Schlegel, “The fractional-order pid controller outperforms the classical one”, Pardubice: Technical University, 2006, pp. 1–6, ISBN: 80-7194-860-8. [Online]. Available: `http://www.kky.zcu.cz/en/publications/CechM_2006_Thefractional-order`.

[43] S. Zheng, X. Tang, and B. Song, “A graphical tuning method of fractional order proportional integral derivative controllers for interval fractional order plant”, *Journal of Process Control*, vol. 24, no. 11, pp. 1691–1709, Nov. 2014.

[44] A. Tepljakov, E. Petlenkov, and J. Belikov, “FOPID controller tuning for fractional FOPDT plants subject to design specifications in the frequency domain”, in *2015 European Control Conference (ECC)*, IEEE, Jul. 2015, pp. 3502–3507.

[45] A. Tepljakov, E. Petlenkov, and J. Belikov, “Robust fopi and fopid controller design for ffopdt plants in embedded control applications using frequency-domain analysis”, in *2015 American Control Conference (ACC)*, 2015, pp. 3868–3873.

[46] A. Tepljakov, E. Petlenkov, and J. Belikov, “FOMCON: Fractional-order modeling and control toolbox for MATLAB”, in *Proceedings of the 18th International Conference Mixed Design of Integrated Circuits and Systems - MIXDES 2011*, Jun. 2011, pp. 684–689.

[47] H. van der Bent, “Wireless technology in industrial automation”, Yokogawa Europe BV, Tech. Rep., 2015. [Online]. Available: `https://www.yokogawa.com/eu/library/resources/white-papers/wireless-technology-in-industrial-automation/`.

[48] B. Holfeld, D. Wieruch, T. Wirth, L. Thiele, S. A. Ashraf, J. Huschke, I. Aktas, and J. Ansari, “Wireless communication for factory automation: An opportunity for LTE and 5g systems”, *IEEE Communications Magazine*, vol. 54, no. 6, pp. 36–43, Jun. 2016.

[49] M. J. Olsen. (). Addict repository. addict python library, used for ease creation of dictionary in FOPID Tuning, [Online]. Available: https://github.com/mewwts/addict.

Appendix

1. FOMCON vs FOMCONpy

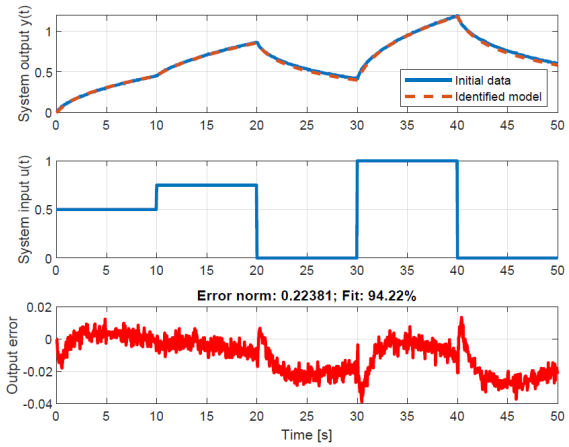
To certify that *FOMCONpy* performance is as good as its counterpart FOMCON, The author did performance tests with time domain identification. Using same data as used in section 5.3. The performance criteria was percentage fitness (3.8) discussed in section 3.3. Using Grunwald Letnikov simulation ,comparison was done for Levenberg Marquardt and Trust Region Reflective optimization method. For each method we compared the outputs percentage fitness for different optimized parameter: Coefficients only, Exponents only and Both (Coefficients + Exponents).

To obtain the FOMCONpy identification results and plots, the user can run the `test()` function in the `fomconoptimize` module which has the following commands:

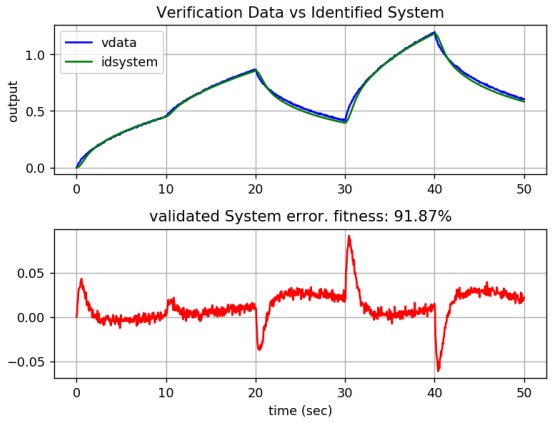
```
def test():
    result, counter = [], 1
    guessset = newfotf('-2s^{0.63}+4', '2s^{3.501}+3.8s^{2.42}+2.6s
   ^{1.798}+2.5s^{1.31}+1.5', 0)
    guessset.numberOfDecimal = 5
    for j in [optAlgo.LevenbergMarquardt]:
        for k in [optFix.Free, optFix.Coeff,optFix.Exp]:
            for l in [simMethod.grunwaldLetnikov]:
                polyfixset = [0, 0]
                optiset = opt(guessset, l, j, k, polyfixset)
                print('\n{0}: Computing settings: {1}, {4}, {2}, {3}\n'
   .format(counter,  j, k, polyfixset,l))
```

```
                res = fid('dataFiles\idenData.xlsx', 'dataFiles\
ValiData.xlsx', optiset, plot=[False, False], plotid=[False, True],
 cleanDelay=[True,2.5])
                res.G.numberOfDecimal = 5
                result.append(res)
                print(res.G, "\n\n")
                counter+=1

 guessset = newfotf('2s^{0.63}+4', '2s^{3.501}+3.8s^{2.42}+2.6s
^{1.798}+2.5s^{1.31}+1.5', 0)
 guessset.numberOfDecimal = 5
 for j in [optAlgo.TrustRegionReflective]:
     for k in [optFix.Free, optFix.Coeff,optFix.Exp]:
         for l in [simMethod.grunwaldLetnikov]:
             polyfixset = [0, 0]
             optiset = opt(guessset, simMethod.grunwaldLetnikov, j,
k, polyfixset)
             print('\n{0}: Computing settings: {1}, {4}, {2}, {3}\n'
.format(counter,  j, k, polyfixset,l))
             res = fid('dataFiles\idenData.xlsx', 'dataFiles\
ValiData.xlsx', optiset, plot=[False, False], plotid=[False, True],
 cleanDelay=[True,2.5])
             res.G.numberOfDecimal = 5
             result.append(res)
             print(res.G, "\n\n")
             counter+=1

 return result
```

Listing 1.1. Using the `fid()` function to generate identification models and plots for comparison with MATLAB



(a) Matlab LM Both: 94.22% fitness (better).



(b) Python LM Both: 91.87% fitness.

1.1 Identification using Levenberg Marquardt method

Initial Guess used for bench-marking purpose:

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

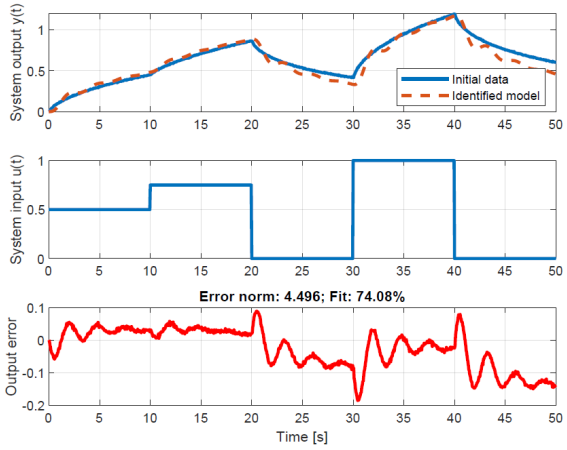
1.1.1 Both – Coefficients + Exponents

1. Matlab LM Both: 94.22% fitness (better).

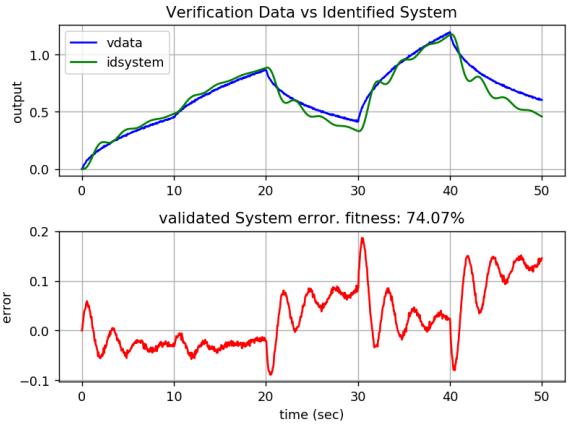
$$\frac{4.1559s^{0.050734} - 2.0996s^{1.2942\times10^{-07}}}{2.256s^{1.6514} + 2.3271s^{0.8797} + 3.7516s^{0.6457} + 2.8353s^{0.6421} + 0.22734s^{5.2447\times10^{-10}}}$$

2. Python LM Both: 91.87% fitness.

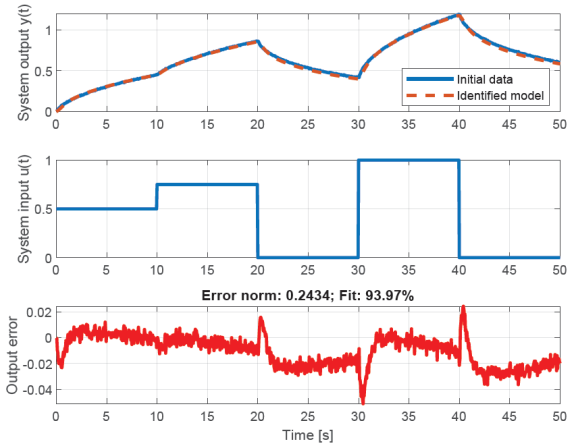
$$\frac{-2.91799s^{-0.04681} + 5.35592}{1.15126s^{2.5615} + 4.74487s^{1.59725} + 3.97537s^{0.62262} + 5.76595s^{0.61868} + 0.30441}$$



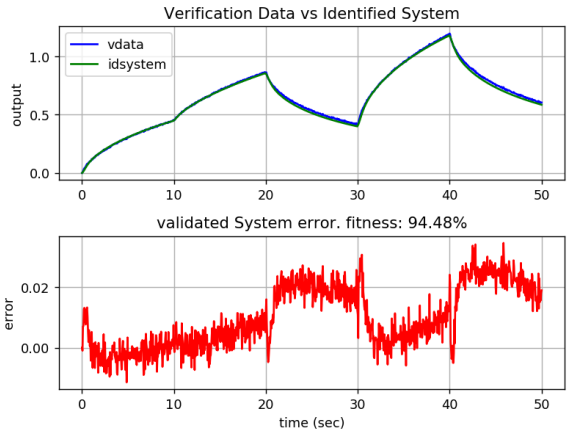
(a) Matlab LM Exponents only: 74.08% fitness (better).



(b) Python LM Exponents only: 74.07% fitness.



(a) Matlab LM Coefficients only: 93.97% fitness.



(b) Python LM Coefficients only: 94.48% fitness (better).

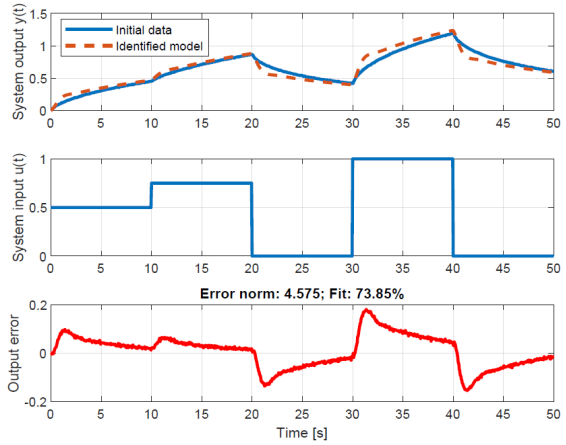
1.1.2 Exponents only

1. Matlab LM Exponents only: 74.08% fitness better.

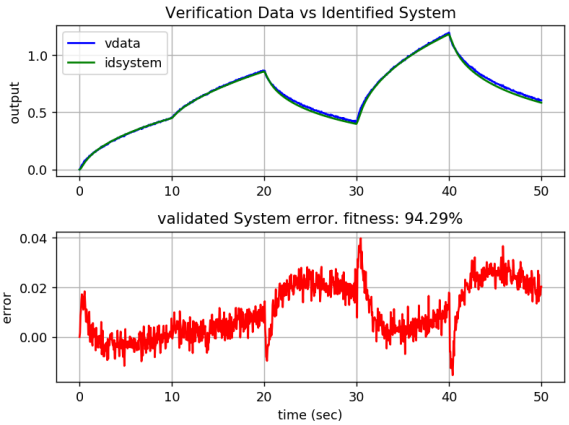
$$\frac{2s^{0.24299} + 4s^{6.2041\times10^{-17}}}{2s^{2.4914} + 3.8s^{0.63126} + 2.5s^{0.63125} + 2.6s^{0.63123} + 1.5}$$

2. Python LM Exponents only: 74.07% fitness.

$$\frac{-2.0s^{0.24309} + 4.0}{2.0s^{2.49852} + 3.8s^{0.63103} + 2.6s^{0.63134} + 2.5s^{0.63131} + 1.5}$$



(a) Matlab TRR Both: 73.85% fitness.



(b) Python TRR Both: 94.29% fitness (better).

1.1.3 Coefficients only

1. Matlab LM Coefficients only: 93.97% fitness.

$$\frac{1.0406s^{0.63} - 0.080038}{-0.024459s^{3.501} + 1.3468s^{2.42} - 0.90976s^{1.798} + 5.0451s^{1.31} - 0.024465}$$

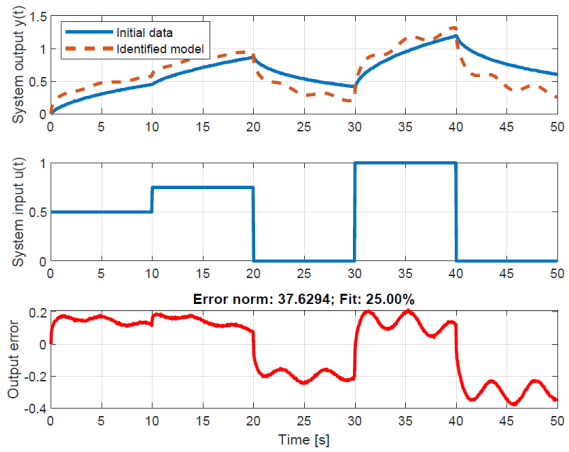
2. Python LM Coefficients only: 94.48% fitness (better).

$$\frac{206.27943s^{0.63} - 16.53132}{-2.95129s^{3.501} + 159.90228s^{2.42} - 107.60438s^{1.798} + 972.42517s^{1.31} - 5.20082}$$

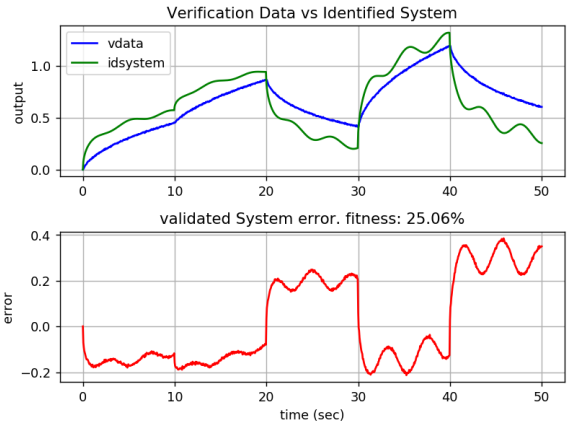
1.2 Identification using Trust Region Reflective method

Initial Guess used for bench-marking purpose:

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



(a) Matlab TRR Exponent only: 25.00% fitness.



(b) Python TRR Exponents only: 25.06% fitness (better).

1.2.1 Both – Coefficients + Exponents

1. Matlab: 73.85% fitness (but does not have a steady state, because the last exponent of s that is, $\alpha_0 \neq \beta_0 \neq 0$).

$$\frac{8.4326s^{1.0176} + 2.2713s^{0.43631}}{7.9863s^{2.6433} + 30.791s^{1.1911} - 2.3643s^{0.5283} + 1.5097s^{0.35477} - 0.027909s^{0.25957}}$$

2. Python: 94.29% fitness (better).

$$\frac{-0.53277s^{0.4587} + 2.58579}{5e-05s^{3.71577} - 0.13814s^{2.40318} + 1.25029s^{1.76607} + 9.66293s^{0.59257} + 0.63314}$$

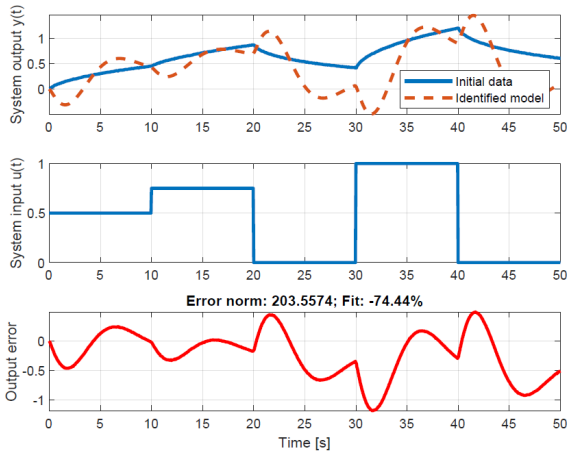
1.2.2 Exponents only

1. Matlab: 25.00% fitness.

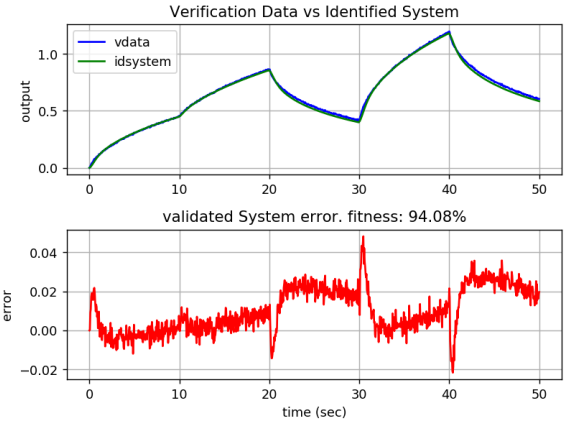
$$\frac{2s^{1.9976} + 4s^{2.4281\times10^{-14}}}{3.8s^{2.4129} + 2.5s^{0.44366} + 2.6s^{0.44362} + 2s^{0.4436} + 1.5s^{9.9998\times10^{-10}}}$$

2. Python: 25.06% fitness (better).

$$\frac{2.0s^{1.99175} + 4.0}{2.0s^{0.44364} + 3.8s^{2.41274} + 2.6s^{0.44364} + 2.5s^{0.44364} + 1.5}$$



(a) MATLAB TRR Coefficients only: -74.44% fitness.



(b) Python TRR Coefficients only: 94.08% fitness (better).

1.2.3 Coefficients only

1. Matlab: -74.4% fitness.

$$\frac{-6.6881s^{0.63}+3.3329}{-0.029477s^{3.501}+1.2118s^{2.42}+3.4482s^{1.798}+3.0363s^{1.31}+3.1005}$$

2. Python: 94.08% fitness (better).

$$\frac{2.70797s^{0.63}-0.21049}{-0.05962s^{3.501}+3.26931s^{2.42}-2.17784s^{1.798}+13.04912s^{1.31}-0.0649}$$

1.3 Remarks

From the comparison, similar results are obtained from Python as compared to MATLAB with variance in some cases possible dues to allowed number of iterations and no bound restriction was applied.