Chapter 2
Neural Network-Assisted $PI^{\lambda}D^{\mu}$ Control

Mehmet Önder Efe

1 Introduction

The need to handle the computational intensity of fractional order differintegration operators was an obstacle in between useful applications and theory. Rapid growth in the technology of fast computation platforms has made it possible to offer versatile design and simulation tools, from which the field of control engineering has benefited remarkably.

In [1–3], fundamental issues regarding the fractional calculus, fractional differential equations, and a viewpoint from the systems and control engineering are elaborated, and several exemplar cases are taken into consideration. One such application area focuses on PID control with derivative and integral actions having fractional orders, i.e., $PI^{\lambda}D^{\mu}$ control is implemented. In the literature, several applications of $PI^{\lambda}D^{\mu}$ controllers have been reported. The early notion of the scheme is reported by [3, 4]. In [5] and [6], tuning of the controller parameters is considered when the plant under control is a fractional order one. Ziegler–Nichols type tuning rules are derived in [7], and rules for industrial applications are designed in [8]. The application of fractional order PID controllers in chemical reaction systems is reported in [9], and the issues regarding the frequency domain are considered in [10]. Tuning based on genetic algorithms is considered in [11], where the best parameter configuration is coded appropriately and a search algorithm is executed to find a parameter set that meets the performance specifications. A similar approach exploiting the particle swarm optimization for finding a good set of gains and differintegration orders isin [12]. Clearly, the cited volume of works

M.Ö. Efe (✉)
Department of Pilotage, University of Turkish Aeronautical Association,
Akköprü, Ankara, Turkey
e-mail: onderefe@ieee.org

D. Baleanu et al. (eds.), *Fractional Dynamics and Control*,
DOI 10.1007/978-1-4614-0457-6_2,

demonstrates that the interest to PID control is growing also in the direction of fractional order versions. Unsurprisingly, the reason for this is the widespread use of the variants of PID controller and the confidence of the engineers in industry.

The idea of approximating the fractional order operators has been considered in [13], where a fractional order integrator is generalized by a neural network observing some history of the input and the output. The fundamental advancement introduced here is to generalize a PID controller using a neural structure with a similar network structure.

This chapter is organized as follows: Sect. 2 briefly gives the definitions of widely used fractional differintegration formulas and basics of fractional calculus; Sect. 3 describes the Levenberg–Marquardt training scheme and neural network structure; Sect. 4 presents a set of simulation studies, and the concluding remarks are given in Sect. 5 at the end of the chapter.

2 Fundamental Issues in Fractional Order Systems and Control

Let $\mathbf{D}^{\beta}$ denote the differintegration operator of order β, where $\beta \in \Re$. For positive values of β, the operator is a differentiator whereas the negative values of β correspond to integrators. This representation lets $\mathbf{D}^{\beta}$ to be a differintegration operator whose functionality depends upon the numerical value of β. With n being an integer and $n-1 \leq \beta < n$, Riemann–Liouville definition of the β-fold fractional differintegration is defined by (2.1) where Caputo's definition for which is in (2.2).

$$\mathbf{D}^{\beta} f(t) = \frac{1}{\Gamma(n-\beta)} \left(\frac{\mathrm{d}}{\mathrm{d}t}\right)^{n} \int_{0}^{t} \frac{f(\tau)}{(t-\tau)^{\beta-n+1}} \mathrm{d}\tau \tag{2.1}$$

$$\mathbf{D}^{\beta} f(t) = \frac{1}{\Gamma(n-\beta)} \int_{0}^{t} \frac{f^{(n)}(\tau)}{(t-\tau)^{\beta-n+1}} \mathrm{d}\tau \tag{2.2}$$

where $\Gamma(\beta) = \int_0^{\infty} e^{-t} t^{\beta-1} \mathrm{d}t$ is the well-known Gamma function. In both definitions, we assumed the lower terminal zero and the integrals start from zero. Considering $a_k, b_k \in \Re$ and $\alpha_k, \beta_k \in \Re^{+}$, one can define the following differential equation:

$$(a_n \mathbf{D}^{\alpha_n} + a_{n-1}\mathbf{D}^{\alpha_{n-1}} + \cdots + a_0 \mathbf{D}^{\alpha_0}) y(t) = (b_m \mathbf{D}^{\beta_m} + b_{m-1}\mathbf{D}^{\beta_{m-1}} + \cdots + b_0 \mathbf{D}^{\beta_0}) u(t) \tag{2.3}$$

and with the assumption that all initial conditions are zero, obtain the transfer function given by (2.4).

$$\frac{Y(s)}{U(s)} = \frac{b_m s^{\beta_m} + b_{m-1} s^{\beta_{m-1}} + \cdots + b_0 s^{\beta_0}}{a_n s^{\alpha_n} + a_{n-1} s^{\alpha_{n-1}} + \cdots + a_0 s^{\alpha_0}} \tag{2.4}$$

Denoting frequency by ω and substituting $s = j\omega$ in (2.4), one can exploit the techniques of frequency domain. A significant difference in the Bode magnitude plot is to observe that the asymptotes can have any slope other that the integer multiples of 20 dB per decade, and this is a substantially important flexibility for modeling and identification research. When the state space models are taken into consideration, we have

$$\begin{aligned} \mathbf{D}^{\beta}\mathbf{x} &= \mathbf{A}\mathbf{x} + \mathbf{B}u \\ y &= \mathbf{C}\mathbf{x} + Du \end{aligned} \tag{2.5}$$

and we obtain the transfer function via taking the Laplace transform in the usual sense, i.e.,

$$H(s) = \mathbf{C}\left(s^{\beta}\mathbf{I} - \mathbf{A}\right)^{-1}\mathbf{B} + D \tag{2.6}$$

For the state space representation in (2.5), if λ_i is an eigenvalue of the matrix $\mathbf{A}$, the condition

$$|\arg(\lambda_i)| > \beta\frac{\pi}{2} \tag{2.7}$$

is required for stability. It is possible to apply the same condition for the transfer function representation in (2.4), where λ_is denotes the roots of the expression in the denominator.

The implementation issues are closely related to the numerical realization of the operators defined in (2.1) and (2.2). There are several approaches in the literature and Crone is the most frequently used scheme in approximating the fractional order differintegration operators [1]. More explicitly, the algorithm determines a number of poles and zeros and approximates the magnitude plot over a predefined range of the frequency spectrum. In (2.8), the expression used in Crone approximation is given and the approximation accuracy is depicted for $N = 3$ and 9 in Fig. 2.1. According to the approximates shown, it is clearly seen that the accuracy is improved as N gets larger, yet the price paid for this is the complexity and the technique presented next is a remedy to handle the difficulties stemming from the implementation issues.

$$s^{\beta} \approx K\frac{\prod_{k=1}^{N} 1 + s/w_{pk}}{\prod_{k=1}^{N} 1 + s/w_{zk}} \tag{2.8}$$

The PI$^{\lambda}$D$^{\mu}$ controller with the operator described above has the transfer function given by (2.9), where $E(s)$ is the error entering the controller and $U(s)$ stands for the output.

$$\frac{U(s)}{E(s)} = K_p + \frac{K_i}{s^{\lambda}} + K_d s^{\mu} \tag{2.9}$$

In Fig. 2.2, it is illustrated that the classical PID controller variants correspond to a subset in the λ–μ coordinate system, and there are infinitely many parameter configurations that may lead to different performance indications.

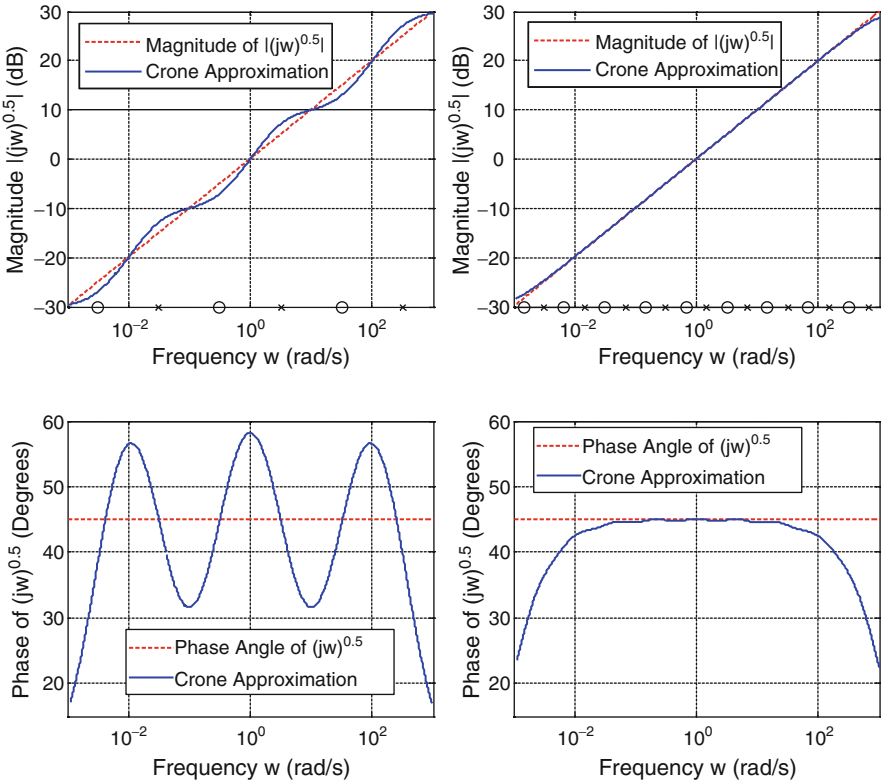

Fig. 2.1 Crone approximation to the operator $s^{0.5}$ with $\omega_{\min} = 1e-3\,\text{rad/s}$, $\omega_{\max} = 1e+3\,\text{rad/s}$. Left column: $N = 3$, Right column: $N = 9$

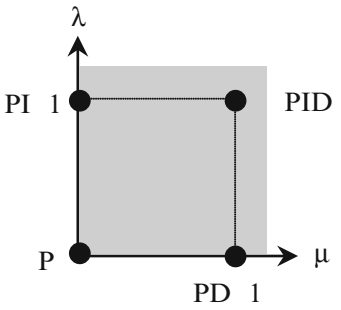

Fig. 2.2 Continuous values of the differintegration orders λ and μ enables to obtain infinitely many configurations of $\text{PI}^{\lambda}\text{D}^{\mu}$ controller where the variants of the classical PID controller correspond to a subset of the domain

3 Neural Network-Based Modeling and Levenberg–Marquardt Training Scheme

In this work, we consider the feedforward neural network structure shown in Fig. 2.3, where there are m inputs, R neurons in the first hidden layer, and Q hidden layer in the second hidden layer. Since the neural structure is aimed to imitate a

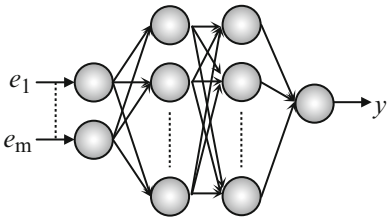

Fig. 2.3 Feedforward neural network structure with R neurons in the first, Q neurons in the second hidden layer

PI$^{\lambda}$D$^{\mu}$ controller, the model has a single output. The hidden layers have hyperbolic tangent-type nonlinear activation while the output layer neuron is linear.

The powerful mapping capabilities of neural networks have made them useful tools of modeling research especially when the entity to be used is in the form of raw data. This particular property is mainly because of the fact that real systems have many variables, the variables involved in the modeling process are typically noisy, and the underlying physical phenomenon is sometimes nonlinear. Due to the inextricably intertwined nature of the describing differential (or difference) equations, which are not known precisely, it becomes a tedious task to see the relationship between the variables involved. In such cases, black box models such as neural networks, fuzzy logic, or the methods adapted from the artificial intelligence come into the picture as tools representing the input/output behavior accurately. In what follows, we describe briefly the Levenberg–Marquardt training scheme for adjusting the parameters of a neural structure [14]. Since the algorithm is a soft transition in between the Newton's method and the standard gradient descent, it very quickly locates the global minimum (if achievable) of the cost hypersurface, which is denoted by J in (2.10).

$$J = \frac{1}{2}\sum_{p=1}^{P}\left(d_p - y_p(e,\phi)\right)^2 \tag{2.10}$$

where y_p denotes the response of the single output neural network, and d_p stands for the corresponding target output. In (2.10), ϕ is the set of all adjustable parameters of the neural structure (weights and the biases), and u is the vector of inputs which are selected according to the following procedure:

$$\phi(t+1) = \phi(t) - \left(\mu I + \Phi(t)^{\mathrm{T}}\Phi(t)\right)^{-1}\Phi(t)^{\mathrm{T}}F(t) \tag{2.11}$$

where μ is the regularization parameter, $F(t) = [f_1\, f_2 \ldots f_P]^T$ is the vector of errors described as $f_i = d_i - y_i(e,\phi) i = 1,2,\ldots,P$, where P is the number of training pairs and Φ is the Jacobian given explicitly by (2.12)

$$\Phi = \begin{bmatrix} \frac{\partial f_1}{\partial \phi_1} & \frac{\partial f_1}{\partial \phi_2} & \cdots & \frac{\partial f_1}{\partial \phi_H} \\ \frac{\partial f_2}{\partial \phi_1} & \frac{\partial f_2}{\partial \phi_2} & \cdots & \frac{\partial f_2}{\partial \phi_H} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_P}{\partial \phi_1} & \frac{\partial f_P}{\partial \phi_2} & \cdots & \frac{\partial f_P}{\partial \phi_H} \end{bmatrix} \tag{2.12}$$

where there are H adjustable parameters within the vector ϕ. In the application of the tuning law in (2.11), if μ is large, the algorithm behaves more like the gradient descent; conversely, if μ is small, the prescribed updates are more like the Gauss–Newton updates. The algorithm removes the problem of rank deficiency in (2.11) and improves the performance of gradient descent significantly.

4 Simulation Studies

The first stage of emulating the response of a $\mathrm{PI}^{\lambda}\mathrm{D}^{\mu}$ controller is to select a representative set of inputs to be applied to the $\mathrm{PI}^{\lambda}\mathrm{D}^{\mu}$ controller and to collect the response. We have set $N = 9$ and follow the procedure described below.

For $n = 1$ to #experiments
  Set a random $K_p \in (0,2)$
  Set a random $K_d \in (0,1)$
  Set a random $K_i \in (0,1)$
  Set a random $\mu \in (0,1)$
  Set a random $\lambda \in (0,1)$
  Apply $u(t)$ and obtain $y(t)$ for $t \in [0,3]$
  Store $u(t)$, $y(t)$, K_p, K_d, K_i, μ, λ
End

A total of 200 experiments with step size 1 ms have been carried out to obtain the data to be used for training data. Once the set of all responses are collected, a matrix is formed, a generic row of which has the following structure:

$$[y(k), y(k-1), \cdots, y(k-d), K_p(k), K_d(k), K_i(k), \lambda(k), \mu(k)] \tag{2.13}$$

where k is the time index indicating $y(k) = y(kT)$ and $T = 1$ ms, and there are $d+6$ columns in each row and the delay depth d is a user-defined parameter. Denote the matrix, whose generic row is shown above, by Ω. In order to obtain the training data set, we downsample the matrix Ω by selecting the first row of every 100 consecutive row blocks. This significantly reduces the computational load of the training scheme, and according to the given procedure, 60,000 pairs of training data are generated and a neural network having $m = 16$ inputs is constructed. In Fig. 2.4, the evolution for the training data is shown with that obtained for the checking data, which is obtained by running 15 experiments and the same procedure of downsampling.

At 128th epoch, the best set network parameters is obtained, and after this time the checking error for the neural model starts increasing and the training scheme stops the parameter tuning when $J = 0.01778$. In what follows, we discuss the performance of the neural model as a $\mathrm{PI}^{\lambda}\mathrm{D}^{\mu}$ controller.

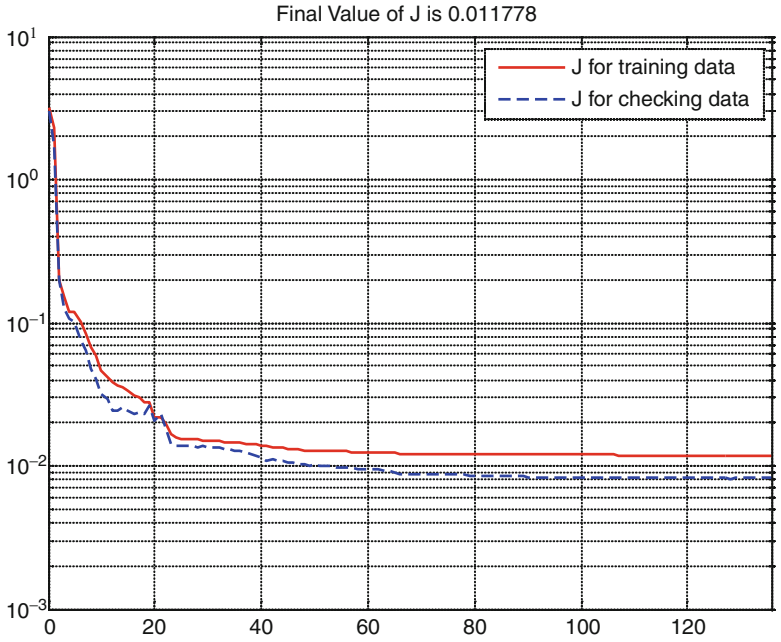

Fig. 2.4 Feedforward neural network structure with R neurons in the first, Q neurons in the second hidden layer

As an illustrative example, we consider the following control problem, which is simple yet our goal is to compare the responses of two controllers, namely, PI$^{\lambda}D^{\mu}$ controller and its neural network-based approximate. The plant dynamics is given below:

$$\frac{Y(s)}{U(s)} = \frac{1}{s(s+1)} \tag{2.14}$$

where Y is the plant output and U is the control input. We choose $K_p = 2.5$, $K_d = 0.9$, $K_i = 0.1$, $\mu = 0.02$, $\lambda = 0.7$ and apply a step command that rises when $t = 1$ s. The command signal, the response obtained with the PI$^{\lambda}$D$^{\mu}$ controller exploiting the above parameters, and the result obtained with the trained neural network emulator are shown on the top row of Fig. 2.5, where the response of PI$^{\lambda}$D$^{\mu}$ controller is obtained using the toolbox described in [15]. For a better comparison, the bottom row depicts the difference in between the plant responses obtained for both controllers individually. Clearly the results suggest that the neural network-based controller is able to imitate the PI$^{\lambda}$D$^{\mu}$ controller to a very good extent as the two responses are very close to each other.

A better comparison is to consider the control signals that are produced by the PI$^{\lambda}$D$^{\mu}$ controller (u_{FracPID}) and the neural network controller (u_{NNPID}). The

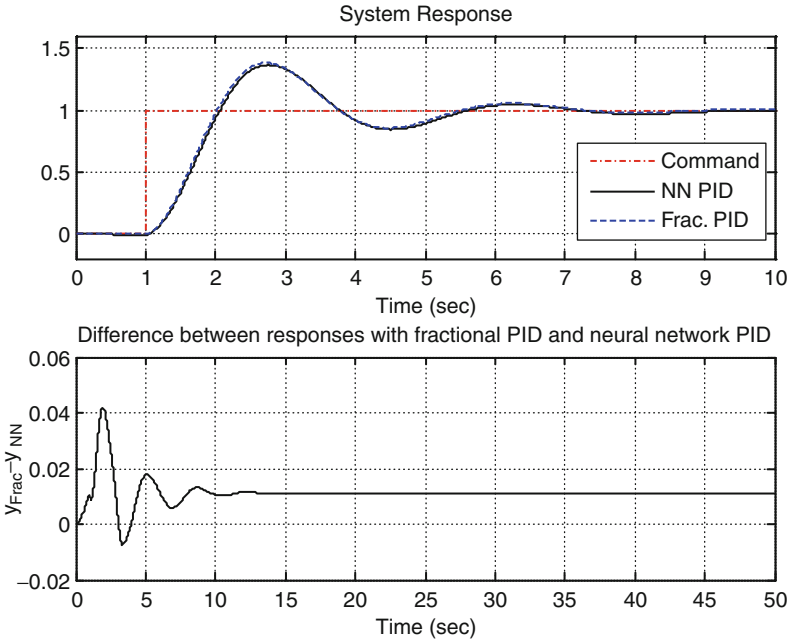

Fig. 2.5 For the first example, system response and the difference in between the two responses obtained with the $PI^{\lambda}D^{\mu}$ controller and its neural network-based substitute

results are seen in Fig. 2.6, where the two control signals are shown together on the top subplot, whereas the difference between them is illustrated in the bottom subplot. Clearly the two control signals are very close to each other; furthermore, the signal generated by the neural network is smoother than its alternative when $t = 1$. This particular example demonstrates that the neural network-based realization can be a good candidate for replacing the $PI^{\lambda}D^{\mu}$ controller.

Define the following relative error as given in (2.15), where T denotes the final time. For the results seen above, we obtain $e_{\text{rel}} = 0.1091$, which is an acceptably small value indicating the similarity of the two control signals seen in Fig. 2.6.

$$e_{\text{rel}} := \frac{\frac{1}{T}\int_0^T |u_{\text{FracPID}} - u_{\text{NNPID}}|\,\mathrm{d}t}{\frac{1}{T}\int_0^T |u_{\text{FracPID}}|\,\mathrm{d}t} \tag{2.15}$$

In Table 2.1, we summarize a number of test cases with corresponding relative error values. The data presented in the table indicate that the proposed controller is able to perform well for a wide range of controller gains and for small values of λ and μ. However, for another control problem, the proposed scheme may perform

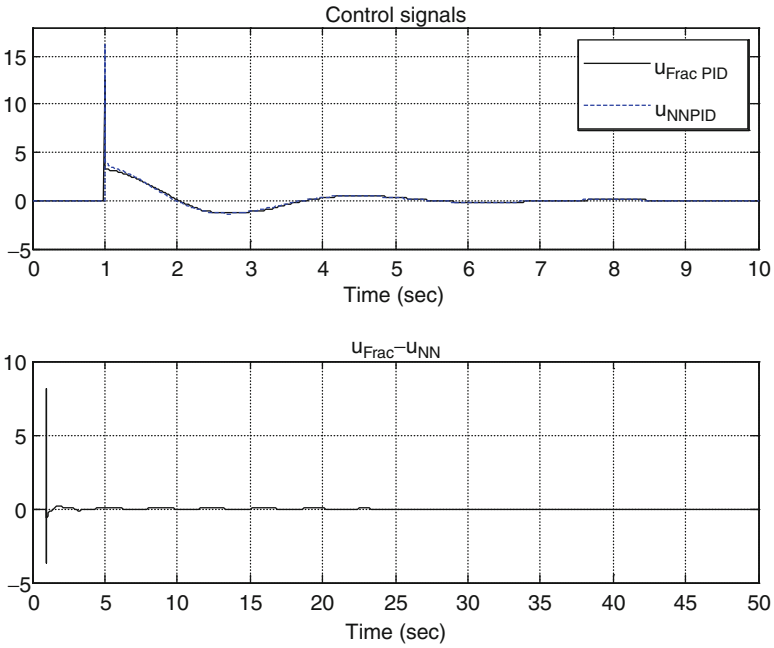

Fig. 2.6 The control signals generated by the PI$^{\lambda}$D$^{\mu}$ controller and its neural network-based substitute. The *bottom row* shows the difference between the two signals

Table 2.1 Performance of the proposed controller for a number of different parameter configurations

K_p	K_i	K_d	μ	λ	$e_{\text{rel.}}$
1.3000	0.9000	0.7000	0.0200	0.0900	0.0897
2.1000	0.9000	0.1000	0.0200	0.3900	0.0938
1.7000	0.7000	0.4000	0.0200	0.0900	0.0965
2.5000	0.7000	0.1000	0.0200	0.3900	0.0982
2.5000	0.7000	0.1000	0.0200	0.6900	0.0991
1.7000	0.3000	1.0000	0.0200	0.0900	0.1005
0.9000	0.9000	0.7000	0.0200	0.0900	0.1014
2.5000	0.9000	0.1000	0.0200	0.6900	0.1018
1.3000	0.7000	0.4000	0.0200	0.0900	0.1028
1.7000	0.1000	1.0000	0.0200	0.0900	0.1035
2.1000	0.9000	0.1000	0.0200	0.6900	0.1038
1.3000	0.7000	1.0000	0.0200	0.0900	0.1052
1.3000	0.9000	0.4000	0.0200	0.0900	0.1073
1.7000	0.5000	0.7000	0.0200	0.0900	0.1081
1.3000	0.9000	1.0000	0.0200	0.0900	0.1090
0.9000	0.9000	1.0000	0.0200	0.0900	0.1094
0.9000	0.7000	0.7000	0.0200	0.0900	0.1108
2.1000	0.7000	0.1000	0.0200	0.9900	0.1112
1.7000	0.5000	1.0000	0.0200	0.0900	0.1113

better for larger values of differintegration orders. To see this, as a second example, we consider the following plant dynamics:

$$\begin{aligned} x_1^{(0.1)} &= x_2 \\ x_1^{(0.4)} &= x_3 \\ x_3^{(0.8)} &= f(x_1,x_2,x_3) + \Delta(x_1,x_2,x_3,t) + g(t)x_4 + \xi(t) \\ x_4^{(0.5)} &= u \end{aligned} \tag{2.16}$$

where $\Delta(x_1,x_2,x_3)$ and $\xi(t)$ are uncertainties and disturbance terms that are not available to the designer. In the above equation, we have

$$f(x_1,x_2,x_3) = -0.5x_1 - 0.5x_2^3 - 0.5x_3|x_3| \tag{2.17}$$

$$g(t) = 1 + 0.1\sin\left(\frac{\pi t}{3}\right) \tag{2.18}$$

$$\begin{aligned} \Delta(x_1,x_2,x_3,t) = &(-0.05 + 0.25\sin(5\pi t))x_1 + (-0.03 + 0.3\cos(5\pi t))x_2^3 \\ &+ (-0.05 + 0.25\sin(7\pi t))x_3|x_3| \end{aligned} \tag{2.19}$$

$$\xi(t) = 0.2\sin(4\pi t) \tag{2.20}$$

The plant considered is a nonlinear one having four states, disturbance terms, and uncertainties. The time-varying gain multiplying the state x_4 in (2.14) makes the problem further complicated, and we compare the neural network substitute of the $\mathrm{PI}^{\lambda}\mathrm{D}^{\mu}$ controller given by

$$\frac{U(s)}{E(s)} = 2 + \frac{0.7}{s^{0.9}} + 0.6s^{0.75} \tag{2.21}$$

The results are illustrated in Figs. 2.7 and 2.8. The responses of the system for both controllers are depicted in Fig. 2.7, where we see that the two responses are very close to each other. The similarity in the fluctuations around the setpoint is another result to emphasize. The outputs of the controllers are analyzed in Fig. 2.8, where we see that the $\mathrm{PI}^{\lambda}\mathrm{D}^{\mu}$ controller generates a very large magnitude spike when the step change in the command signal occurs, whereas the neural network-based substitute produces a smoother control signal, and this is reflected as a slight difference in between the plant responses to controllers being compared. The two controllers produce similar signals when the plant output is forced to lie around unity, which is seen in the middle subplot of Fig. 2.8, and the difference between the two control signals is seen to be bounded by 0.05 during this period. The value of e_{rel} for this case is equal to 20.3283, which seems large but noticing the peak in the top subplot of Fig. 2.8; this could be seen tolerable as the $\mathrm{PI}^{\lambda}\mathrm{D}^{\mu}$ controller requests high magnitude control signals when there is a step change in the command.

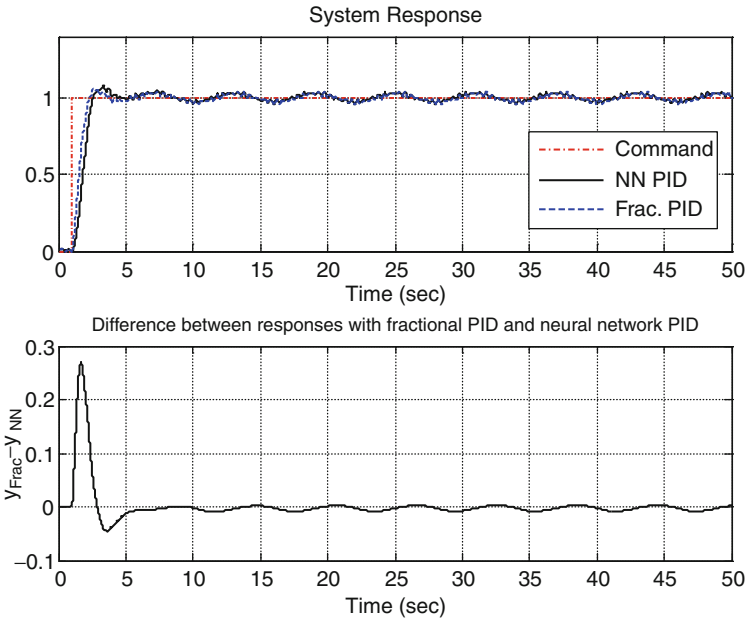

Fig. 2.7 For the second example, system response and the difference in between the two responses obtained with the PI$^{\lambda}$D$^{\mu}$ controller and its neural network-based substitute

A last issue to consider here is the possibility of increasing the performance obtained by the chosen neural network structure, which is 16-25-10-1. One can argue that the neural network could be realized as a single hidden layer one, or with two hidden layers with less number of neurons in each. In obtaining the neural model, whose results are discussed, many trials have been performed, and it is seen that the approximation performance could be increased if there are more neurons in the hidden layers. In a similar fashion, a better map could be constructed if earlier values of the incoming error signal are taken into consideration. This enlarges the network size and makes it more intense computationally to train the model. Depending on the problem in hand, the goal of this chapter is to demonstrate that a fractional order PI$^{\lambda}$D$^{\mu}$ control could be replicated to a certain extent using neural network models, and the findings of the chapter support these claims thoroughly.

5 Conclusions

This chapter discusses the use of standard neural network models for imitating the behavior of a PI$^{\lambda}$D$^{\mu}$ controller, whose parameters are provided explicitly as the inputs to the neural network. The motivation in focusing this has been the difficulty of realizing fractional order controllers requiring high orders of approximation for

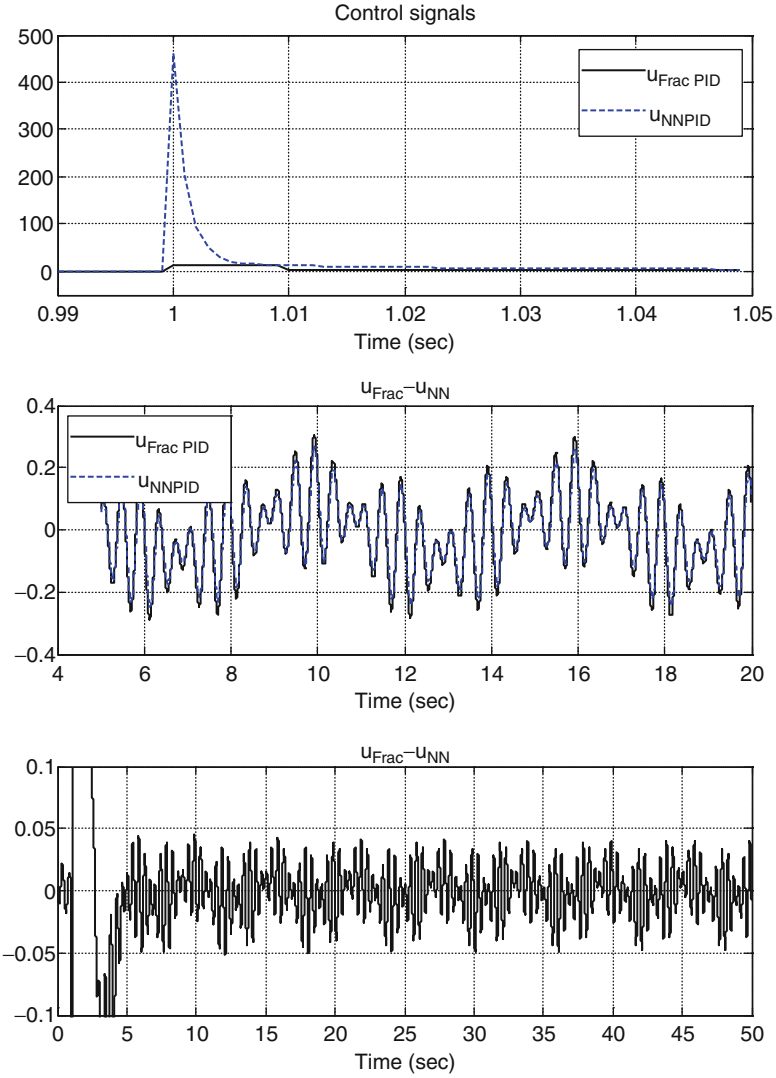

Fig. 2.8 The control signals generated by the $PI^{\lambda}D^{\mu}$ controller and its neural network-based substitute. The *top row* illustrates the two signals when the step change occurs. The *middle row* depicts the closeness of the two signals when $t > 5$ s, and the *bottom row* shows the difference in between the two signals

accuracy. The method followed here is to collect a set of data and to optimize the set of parameters to obtain an emulator of the $PI^{\lambda}D^{\mu}$ controller. Aside from the parameters of the $PI^{\lambda}D^{\mu}$ controller, the neural model observed some history of the input and outputs a value approximating the response of the $PI^{\lambda}D^{\mu}$ controller. Several exemplar cases are presented, and it is seen that the use of neural network

models is a practical alternative in realizing the $PI^{\lambda}D^{\mu}$ controllers. Furthermore, the developed neural model allows modifying the controller parameters online as those parameters are supplied as eternal inputs to the network.

Acknowledgment This work is supported in part by Turkish Scientific Council (TÜBİTAK) Contract 107E137.

References

1. Das S (2008) Functional fractional calculus for system identification and controls, 1st edn. Springer, New York
2. Oldham KB, Spanier J (1974) The fractional calculus. Academic, London
3. Podlubny I (1998) Fractional differential equations, 1st edn. Elsevier Science & Technology Books, Amsterdam
4. Podlubny I (1999) Fractional-order systems and (PID mu)-D-lambda-controllers. IEEE Trans Automatic Control 44(1):208–214
5. Zhao C, Xue D, Chen Y-Q (2005) A fractional order PID tuning algorithm for a class of fractional order plants. Proceedings of the IEEE international conference on mechatronics & automation, Niagara Falls, Canada
6. Caponetto R, Fortuna L, Porto D (2002) Parameter tuning of a non integer order PID controller. In: Proceedings of the fifteenth international symposium on mathematical theory of networks and systems, Notre Dame, Indiana
7. Valerio D, Sa Da Costa L (2006) Tuning of fractional PID controllers with Ziegler–Nichols-type rules. Signal Process 86:2771–2784
8. Monje CA, Vinagre BM, Feliu V, Chen Y-Q (2006) Tuning and auto-tuning of fractional order controllers for industry applications. Control Eng Pract 16:798–812
9. Leu JF, Tsay SY, Hwang C (2002) Design of optimal fractional-order PID controllers. J Chinese Institute Chem Eng 33(2):193–202
10. Vinagre BM, Podlubny I, Dorcak L, Feliu V (2000) On fractional PID controllers: a frequency domain approach. In: IFAC workshop on digital control. past, present and future of PID control. Terrasa, Spain, pp 53–58
11. Cao J-Y, Liang J, Cao B-G (2005) Optimization of fractional order pid controllers based on genetic algorithms. In: Proceedings of the fourth international conference on machine learning and cybernetics, Guangzhou, 18–21 August
12. Maiti D, Biswas S, Konar A (2008) Design of a fractional order PID controller using particle swarm optimization technique. In: 2nd national conference on recent trends in information systems (ReTIS-08), February 7–9, Kolkata, India
13. Abbisso S, Caponetto R, Diamante O, Fortuna L, Porto D (2001) Non-integer order integration by using neural networks. The 2001 IEEE International Symposium on Circuits and Systems (ISCAS 2001), 6–9 May, vol. 2, pp 688–691
14. Hagan MT, Menhaj MB (1994) Training feedforward networks with the Marquardt algorithm. IEEE Trans Neural Network 5(6):989–993
15. Valerio D (2005) Ninteger v.2.3 fractional control toolbox for MatLab