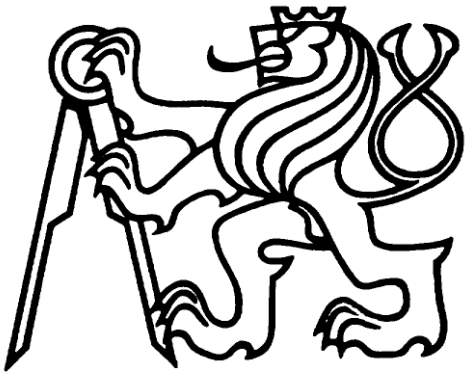
CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF MECHANICAL ENGINEERING

Department of Instrumentation and Control Engineering



Master's Thesis

MURAT ÜNVER

Academic Year: 2017/2018

CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF MECHANICAL ENGINEERING

Department of Instrumentation and Control Engineering

Master's Thesis

Study of Incremental Machine Learning Algorithms for 2-2 Linear MIMO system with Prospects to Control of Actively Actuated Double-Wheel-Set Roller Rig

Author: MURAT ÜNVER

Academic Year: 2017/2018

Supervisor: doc. Ing. Ivo Bukovsky, Ph.D.

Czech Technical University in Prague Faculty of Mechanical Engineering

Department of Instrumentation and Control Engineering *School year:* 2017/2018

MASTER THESIS ASSIGNMENT

for Murat ÜNVER

Field of study *Instrumentation and Control Engineering*

Study of Incremental Machine Learning Algorithms for 2-2 Linear MIMO system with Prospects to Control of Actively Actuated Double-Wheel-Set Roller Rig

The thesis will be concerned with the following items:

1) Review recently published works on adaptive algorithms for identification and control of multi-input multi-output systems with supervised learning algorithms; focus on sample-by-sample adaptation rules (GD, RLS, ADAM,…).
2) Propose a possible adaptive identification and control scheme for Linear MIMO system with two inputs and two outputs (2-2), derive the learning rules and carry out the identification and control experimental analysis with three selected learning rules from task 1) for:
    a. Stable linear 2-2 Linear MIMO system of 2^{nd} order of dynamics
    b. Stable oscillating 2-2 Linear MIMO system of 4^{th} order of dynamics
3) Derive a simplified dynamical model of actively actuated double-wheel-set roller rig that is researched at CTU in Prague.
4) Based on results of task 2), choose the suitable learning rules and carry out the adaptive control of the derived model obtained in task 3).
5) Document your work with proper technical quality and summarize your achievements and results.

Graphical extent: max 50 %

Thesis extent: min 50 pages + appendix

Specialized literature list:

[1] "IEEE Xplore Abstract - Adaptive control: The model reference approach." [Online]. Available: http://ieeexplore.ieee.org.ezproxy.techlib.cz/xpls/abs_all.jsp?arnumber=6313284. [Accessed: 23-Jan-2016].

[2] A.-O. Ilia, "*Solution for Adaptive Control of Coupled Motor System with Flexible Joint*" Master's Thesis, FME, CTU in Prague , Jun. 2016 (https://dspace.cvut.cz/handle/10467/66487)

[3] I. Bukovsky, P. Benes, and M. Slama, "Laboratory Systems Control with Adaptively Tuned Higher Order Neural Units," in *Intelligent Systems in Cybernetics and Automation Theory, Vol 2*, vol. 348, R. Silhavy, R. Senkerik, Z. K. Oplatkova, Z. Prokopova, and P. Silhavy, Eds. Berlin: Springer-Verlag Berlin, 2015, pp. 275–284.

[4] P. M. Benes, M. Cejnek, J. Kalivoda, and I. Bukovsky, "Neural Network Approach to Railway Stand Lateral SKEW Control," 2014, pp. 327–339.

[5] more shall be found as a review task 1)

Supervisor: Doc. Ing. Ivo Bukovský, Ph.D.

Date of thesis assignment: 28.3.2017

Deadline of submission: 15.6.2017

Reviewer (tentative): Ing. Peter Mark Beneš

doc. Ing. Jan Chyský, CSc. Prof. Ing. Michael Valášek, DrSc.

Head of Department Dean

Prague, 19.3.2016

If the student does not submit his thesis in time he must justify in advance this fact in writing. If the dean recognizes this excuse (handed to the Dean via the Study Department) he will grant the student an alternative term for his State Final Exam (two terms still remain). If the student does not present a relevant excuse or if the dean does not recognize the excuse, he will also grant the student an alternative term for his State Final Exam. In this case, however, only one term of the State Final Exam remains (Student Examination Code, Art.22, par.3, 4.)

The student acknowledges that he must elaborate the project by himself, without any help except consultations with his supervisor. A list of used literature, other sources and names of consultants must be listed in the thesis.

I received the assignment on (date):

.....

student (signature)

Annotation List

Authors Name:	Murat Ünver
Name of Master's Thesis:	Study of Incremental Machine Learning Algorithms for 2-2 Linear MIMO system with Prospects to Control of Actively Actuated Double-Wheel-Set Roller Rig
Year:	2018
Field of Study:	Instrumentation and Control Engineering
Department:	Department of Instrumentation and Control Engineering
Supervisor:	Doc. Ing. Ivo Bukovsky, Ph.D.
Bibliographical Data:	Number of pages: 88 Number of figures: 42 Number of tables: 17 Number of attachments: 0
Keywords:	machine learning, gradient descent, recursive least square, MIMO dynamical systems, Python, MATLAB, Simulink, training, controller tuning strojové učení, gradientní sestup, rekurzivní nejmenší čtverec, dynamické systémy MIMO, Python, MATLAB, Simulink, trénink, ladění řadiče

Statement

I declare that I have worked out this thesis independently assuming that results of the thesis can also be used at the discretion of the supervisor of the thesis as its co-author. I also agree with the potential publication of the results of the thesis or its substantial part, provided I will be listed as co-author

In Prague:..... **Signature:**.....

Acknowledgments

I would like to express great thanks to my supervisor **doc. Ing. Ivo Bukovsky, Ph. D**, whose knowledge and guidance has helped me by continuous support and professional advices to get through the hard process while preparing this thesis. I would like to also thank **Ing. Peter M. Benes, Ph. D**. For his valuable advice and expertise.

Abstract

This master's thesis main goal is to make research about model reference adaptive control for linear MIMO systems. For these purposes theoretical models are used to see possible implementation of neural algorithms for Linear MIMO system. There are two different algorithms applied; GD and RLS, most effective algorithm is finally adopted to 20th order theoretical MIMO model. All of these analysis, design and implementation are made via MATLAB Simulink and Python 2.7.

Abstract

Cílem této diplomové práce je provést výzkum adaptivního řízení s referenčním modelem pro lineární MIMO systémy. Pro tyto účely se teoretické modely používají k posouzení možnosti implementace neurálních algoritmů pro dynamické MIMO systémy. Jsou použity dva různé algoritmy; GD a RLS, nejúčinnější algoritmus, je nakonec přijat do teoretického modelu MIMO 20. řádu. Všechny tyto analýzy, návrhy a implementace jsou realizovány pomocí MATLABu Simulink a Pythonu 2.7.

Table of Contents

List of Figures

List of Tables

1. Introduction

In last decades, human and machine interaction increased significantly. This pushed machines to become smarter for better use by human in many different fields. As a result, concept of machine learning was born. Nowadays, there are variety usage of machine learning applications with different algorithms in industries. For instance, numerous industrial process are demanding automatic control for efficient production or management and machine learning controller applications are requested instead of conventional controller to ensure high quality control. Therefore, objective of this thesis to make research about adaptive identification and control algorithms and implementation of these neural algorithms on example theoretical plants.There are many studies about neural algorithms and their application however many of them are focused on simple-input simple-output system (SISO) as implementation [1]. In this thesis, multiple-input multiple-output system (MIMO) is chosen for implementation to test performance of neural algorithms. Thus, three linear MIMO system are proposed as theoretical models for testing; stable linear 2-2 linear MIMO system of 2^{nd} order of dynamics and stable oscillating 2-2 linear MIMO system of 4^{th} order of dynamics [2]. These models are built and simulated in MATLAB & Simulink and Python 2.7 environment. Further recently published works on adaptive algorithms for identification and control of linear MIMO system with supervised learning algorithms are reviewed, and sample-by-sample adaptation rules are mainly focused such as Gradient Descent (GD), and Recursive Least Square (RLS) algorithms[3][4]. These adaptation rules are derived and carried out for identification and control experimental analysis on linear MIMO system via Python 2.7. There are different control approach such as model predictive adaptive control and model reference adaptive control (MRAC) [5;6]. For this thesis work, MRAC approach is used. As final, theoretical 20^{th} order model is created for derived learning algorithms because given actively actuated double-wheel-set roller rig dynamic model was not stable so it was not suitable for the objective of this thesis as the main objective was the investigation of MRAC for (stabilized) MIMO systems. Therefore to follow main goal of this thesis, similar theoretical model is created to show how these derived neuro controller works with

same order plant with rig model. All of these works are provided with schemes, figures, tables and codes in thesis work. At the end, results are shared in conclusion part.

2. Linear MIMO system Description

Linear MIMO system can be empirically described by several linear dynamic system models. These system are more complicated than SISO system due to several factors which are multivariable interaction, potential co-linearity of inputs, and large data processing requirements. MIMO dynamical models are presented in;

- Ordinary differential equations
- Continuous transfer functions
- State space presentation.

General look to linear MIMO system is illustrated in Figure 1 below. For sake of this thesis work, 2 inputs and 2 outputs linear MIMO system are used.

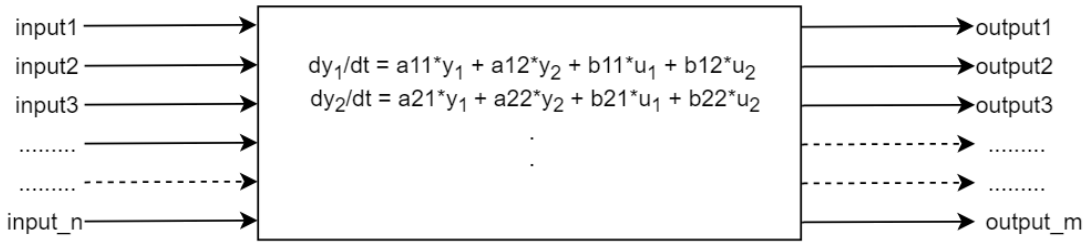


Figure 1:Block diagram of the structure to represent a linear linear MIMO system.

There are two main representations of linear MIMO system; frequency domain and state space model. These representations are most used in this work therefore their form are illustrated as 2-inputs 2-outputs models also any *"u"* presents input and any *"y"* presents output of the system. Frequency domain solution of linear MIMO system in matrix transfer function form is as it follows;

$$\begin{bmatrix} y_1(s) \\ y_2(s) \end{bmatrix} = \begin{bmatrix} g_{11}(s) & g_{12}(s) \\ g_{21}(s) & g_{22}(s) \end{bmatrix} * \begin{bmatrix} u_1(s) \\ u_2(s) \end{bmatrix} \quad (1)$$

The block diagram of transfer function implementation can be shown as in Figure 2.

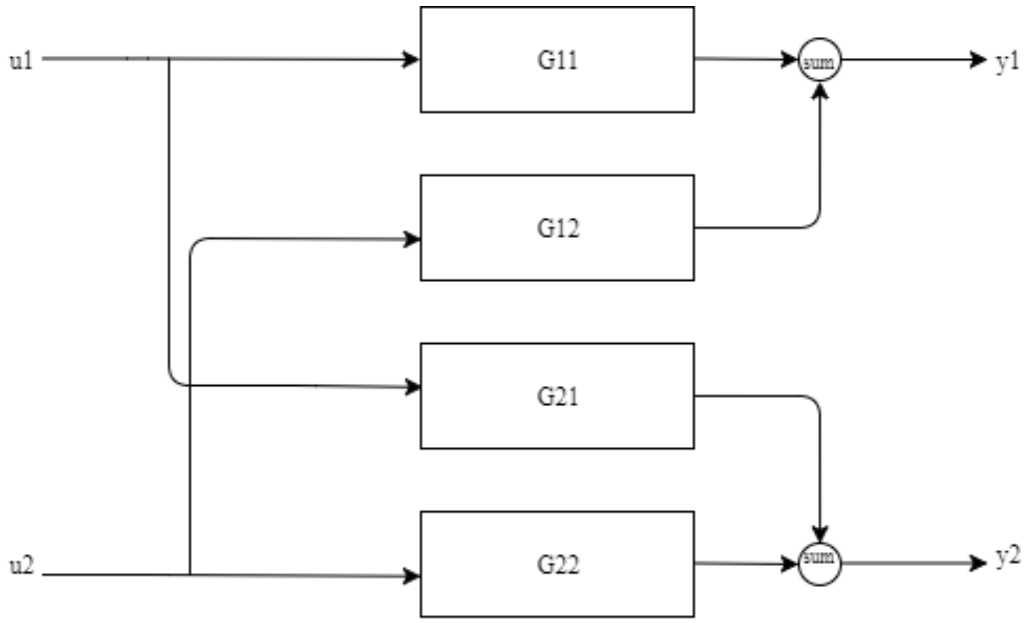


Figure 2: Block diagram of transfer function set for linear MIMO system.

2-inputs and 2-outputs linear MIMO system yield 4 transfer functions. This block diagram shows relation between inputs and outputs as it is in matrix form also explains how system should be simulated in Simulink. As last representation state space form for 2nd order 2-inputs and 2-outputs MIMO [7] system is;

$$\dot{x} = A * x + B * u$$
$$y = C * x + D * u \quad (2)$$

where equation has variables;

- $\dot{x}$ ⇢ the state process, includes states according to order of system,
- u ⇢ the input process, includes inputs for this case in 2x1 dimension,
- y ⇢ the output proces, includes outputs for this case in 2x1 dimension,
- A ⇢ the system matrix,
- B ⇢ the input matrix,
- C ⇢the output matrix,
- D ⇢ the feed-through matrix.

In this part, linear MIMO system is represented simply for better understanding of this thesis work. Next section is about stable linear 2-2 linear MIMO system of 2nd order of dynamics.

2.1 Theoretical Linear Stable 2nd Order Dynamic System

System of second order is used as a test model to see adaptation of learning rules for linear MIMO system. It is created as random model, it has no relation with real plant however because of its structure, it leads better construction of neural learning rules for roller rig. This system is created in MATLAB[8] with simple code as;

```
rss(2,2,2)
```

first element in array defines order of system, second element defines number of inputs, last element defines number of outputs. As result, created system is;

$$\begin{aligned} \dot{x}_1 &= -1.0 * x_1 + 0.5 * x_2 + 3.0 * u_1 + 1.5u_2 \\ \dot{x}_2 &= -0.5 * x_1 - 2.0 * x_2 + 0.0 * u_1 - 1.0 * u_2 \\ y_1 &= 0.4 * x_1 - 0.1 * x_2 \\ y_2 &= -0.98 * x_2 \end{aligned} \tag{3}$$

where system order is reduced to first order and presented as ordinary differential equation (ODE). Yielded continuous time transfer function of system is created in MATLAB & Simulink for simulation as it is below;

$$\begin{aligned} G_{11} &= \frac{1.2s + 3.9}{s^2 + 3 * s + 2.25} \\ G_{12} &= \frac{1.47}{s^2 + 3 * s + 2.25} \\ G_{21} &= \frac{-0.4s + 1.15}{s^2 + 3 * s + 2.25} \\ G_{22} &= \frac{-0.98s - 0.245}{s^2 + 3 * s + 2.25} \end{aligned} \tag{4}$$

This theoretical plant is fed with step signal for inputs in different period and signal width. It is simulated in Python 2.7 via scipy.integrate.odeint[9] to see response of system under different input signals. Also, variable step signals is inserted to system to analyze response of system under different type of inputs signal. Properties of step signal as follows;

- For u1; T = 40 sec., W = %50 * T
- For u2; T = 50 sec., W =%50* T
- T = 200 sec.
- dT = 0.1

under these conditions system response is illustrated as in Figure 3.

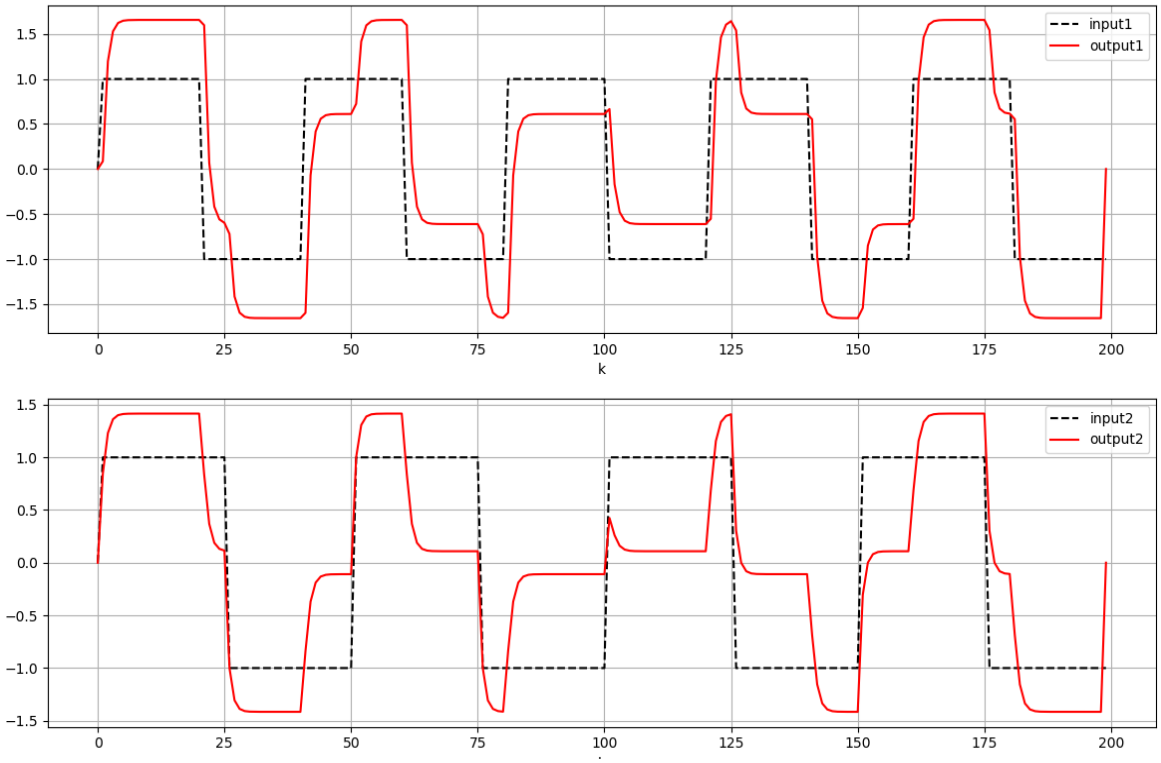


Figure 3: Preview of process data as 2nd order system inputs and outputs.

As it is mentioned before there is an another illustration in Figure 4. Shows theoretical system response to variable input step signals.

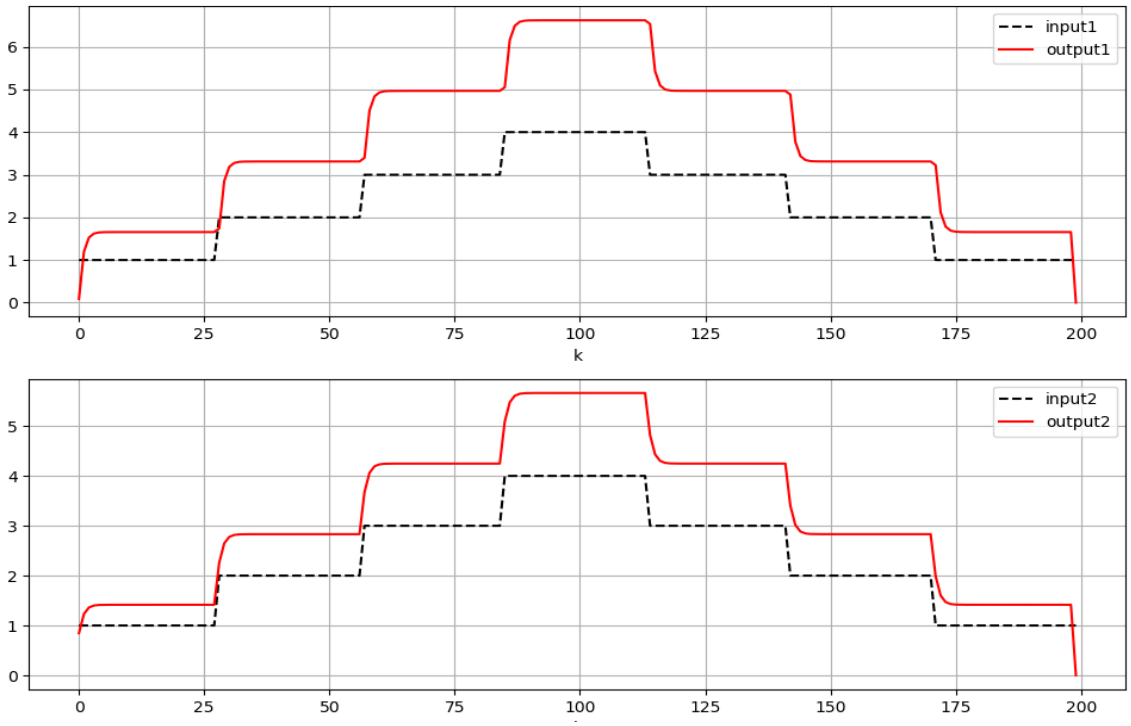


Figure 4: Theoretical 2nd order system response to variable input signals.

2.2 Theoretical Oscillating Stable 4th Order Dynamic System

System of fourth order is used as a test model to see adaptation of learning rules for linear MIMO oscillating system. It is created as random model, it has no relation with real plant however because of its structure, it leads better construction for understanding neural learning rules. This system is created in MATLAB with simple code as;

```
rss(4,2,2)
```

first element in array defines order of system, second element defines number of inputs, last element defines number of outputs. As result, created system is;

$$\begin{aligned}
\dot{x}_1 &= -0.8 * x_1 + 4.0 * x_2 - 5.0 * x_3 - 2.0 * x_4 + 1.0 * u_1 + 0.1u_2 \\
\dot{x}_2 &= -4.0 * x_1 - 1.0 * x_2 - 3 * x_3 - 1.0 * x_4 - 0.1 * u_1 + 0.5 * u_2 \\
\dot{x}_3 &= 5.0 * x_1 + 1 * x_2 - 2.0 * x_3 + 4.0 * x_4 + 0.5 * u_1 \\
\dot{x}_4 &= 3.0 * x_1 + 1.5 * x_2 + 0.5 * x_3 - 5.0 * x_4 - 1.0 * u_1 \\
y_1 &= 1.0 * x_1 - 1.2 * x_2 + 0.96 * x_3 - 1.2 * x_4 \\
y_2 &= 3.4 * x_2
\end{aligned} \tag{5}$$

where system order is reduced to first order and presented as ordinary differential equation (ODE). Yielded continuous time transfer function of system is created in MATLAB & Simulink for simulation as it is below;

$$G_{11} = \frac{2.8\,s^3 + 14.66\,s^2 + 163.1\,s + 285.2}{s^4 + 7\,s^3 + 65\,s^2 + 306\,s + 234}$$

$$G_{12} = \frac{1.47 - 0.34\,s^3 - 14.79\,s^2 - 90.95\,s - 428.7}{s^4 + 7\,s^3 + 65\,s^2 + 306\,s + 234}$$

$$G_{21} = \frac{-0.5\,s^3 - 0.82\,s^2 - 6.912\,s - 84.82}{s^4 + 7\,s^3 + 65\,s^2 + 306\,s + 234} \tag{6}$$

$$G_{22} = \frac{1.7\,s^3 + 8.84\,s^2 + 57.63\,s + 243.1}{s^4 + 7\,s^3 + 65\,s^2 + 306\,s + 234}$$

This theoretical plant is fed with step signal for inputs in different period and signal width. It is simulated in Python 2.7 via scipy.integrate.odeint to see response of system under different input signals. Also, variable step signals is inserted to system to analyze response of system under different type of inputs signal. Properties of step signal as follows;

- For u1; T = 40 sec., W = %50 * T
- For u2; T = 50 sec., W =%50* T
- T = 200 sec.
- dT =0.1

under these conditions system step response is illustrated as in Figure 5.

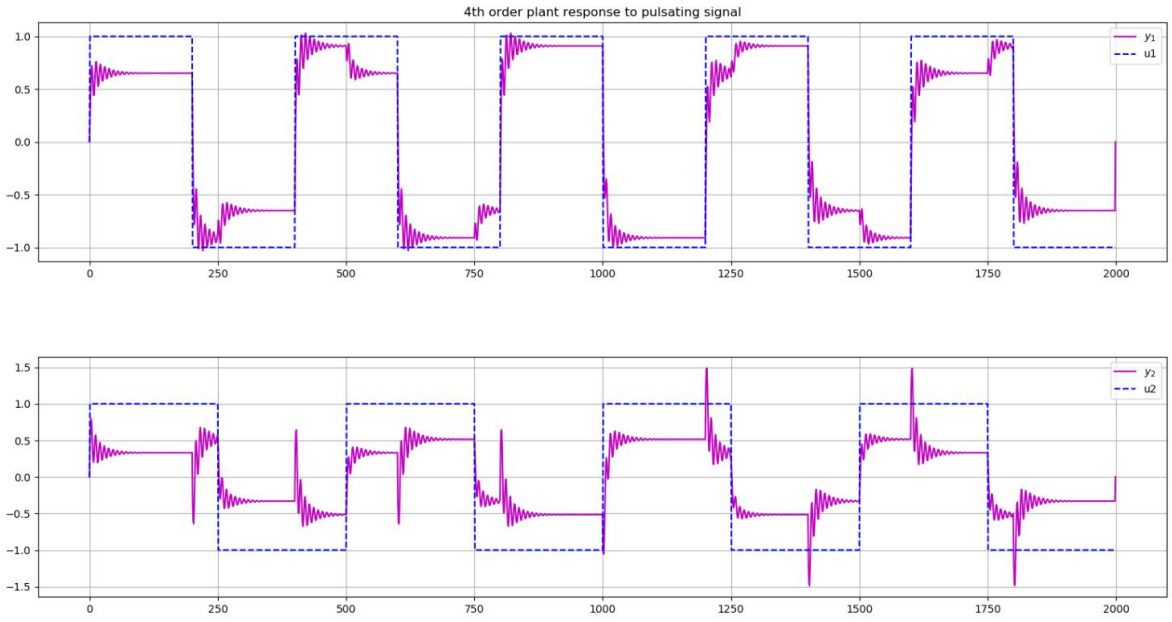


Figure 5: Preview of process data as 4th order system inputs and outputs.

As it is mentioned before there is another illustration in Figure 6. Shows theoretical system response to variable input step signals.

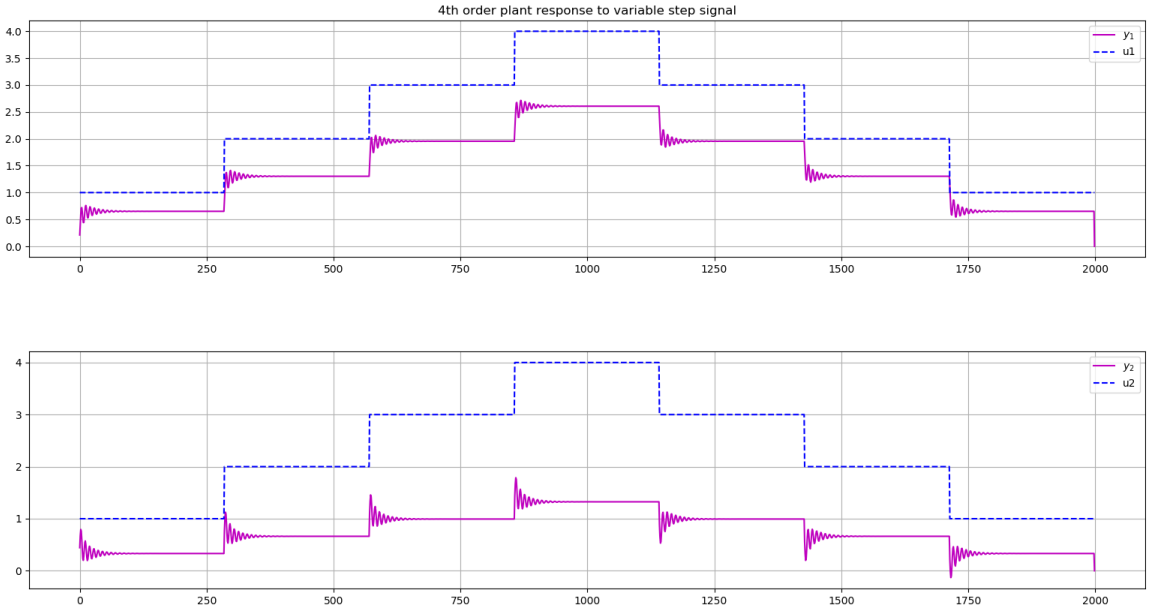


Figure 6: Theoretical 4th order system response to variable input signals.

These are theoretical systems as example to apply identification and control algorithms. In next chapter, these algorithms are derived according to linear MIMO system, and these two theoretical plant are used to see results.

3. Implemented Identification and Control Algorithms

In this chapter, as solution of identification and control, discrete-time adaptive approaches are adopted to neural unit or high order neural units (HONU)[10]. These neural units are used as a recurrent models for dynamic system identification and later as controller.

For linear stable 2nd order theoretical plant linear neural unit (LNU) is used however derivation of high order neural unit (HONU) is also revealed. All applications are based on dynamic neural units which means feedbacks of system also supplied by neural units. This term of dynamic neural units are discussed further. These are two main algorithms are derived; GD and RLS. These both learning rules are based on real time recurrent learning (RTRL) technique[11].

3.1 Sample by Sample Learning Plant Identification

The GD is fundamental learning rule behind neural units such as LNU and quadratic neural units (QNU). General polynomial form of the LNU is to be described as by following;

$$y_n = \sum_{i=0}^{n} w_i * x_i = w_0 * x_0 + w_1 * x_1 + w_2 * x_2 + w_3 * x_3 + .. + w_n * x_n \quad (7)$$

where w is updatable vector of neural weights and x vector of neuron inputs which includes plant inputs and outputs in case of static model, in dynamic model it includes neural outputs instead of system outputs. General polynomial form of the QNU can be derived as below;

$$y_n = \sum_{i=0}^{n}\sum_{j=0}^{n} w_{i,j}\, x_i x_j = w_{0,0}x_0x_0 + w_{0,1}x_0\, x_1 + .. + w_{n,n} * x_n x_n$$
$$y_n = row\boldsymbol{x}.col\boldsymbol{w} \quad (8)$$

Where *"row**x**"* and *"col**w**"* are long vectors representation of input vector and weight matrix of QNU in general.

Now, these neural weights general form is reconstructed according to linear MIMO system also input vector of neuron *"**x**"* includes neuron outputs since system is dynamic. To identify linear MIMO system two linear neural units are set, therefore there are two *"**x**"* vectors are derived for each neural units. System inputs and outputs connect through both neural units since system depends on both. So definition for first neural unit *"**x₁**"* vector and second neural unit *"**x₂**"* vector are as below;

$$x_1 = \begin{bmatrix} 1 \\ y_{n1}[k] \\ y_{n1}[k-1] \\ y_{n1}[k-2] \\ y_{n1}[k-3] \\ y_{n2}[k] \\ y_{n2}[k-1] \\ y_{n2}[k-2] \\ u_1[k] \\ u_1[k-1] \\ u_1[k-2] \\ u_1[k-3] \\ u_2[k] \\ u_2[k-1] \\ u_2[k-2] \end{bmatrix} \qquad x_2 = \begin{bmatrix} 1 \\ y_{n2}[k] \\ y_{n2}[k-1] \\ y_{n2}[k-2] \\ y_{n2}[k-3] \\ y_{n1}[k] \\ y_{n1}[k-1] \\ y_{n1}[k-2] \\ u_2[k] \\ u_2[k-1] \\ u_2[k-2] \\ u_2[k-3] \\ u_1[k] \\ u_1[k-1] \\ u_1[k-2] \end{bmatrix} \tag{9}$$

Where *"y_{n1}"* is first neuron output and *"y_{n2}"* is second neuron output. Also *"u_1"*is first input, *"u_2"* is second output to linear MIMO system. Amount of *"$y_{n1}[k]$"* s, *"$y_{n2}[k]$"* s , *"$u_1[k]$"* s and *"$u_2[k]$"* s correspond to components of input vector. These all components are calculated using system equation at different time instances of *"k"*. Size of *"x"* vector is 15, as result *"**w_1** and **w_2** "* are to be in same size.

3.1.1 Gradient Descent Algorithm Adaptation

The main purpose behind all of these to adapt neural weights and it is a crucial factor for identification process of linear MIMO system. Adaptation of neural weights are succeed by derivation of GD formula [12]. Adaptation of neural weights is same process for both neural units therefore general derivation of GD is given as it follows;

$$w_{i+1} = w_i - \frac{1}{2}\mu * \frac{\partial e^2(k)}{\partial w_i} \tag{10}$$

where "μ" is learning rate of the weight adaptation and *"$e_{(k)}$"* is error value between system or plant and neural output. It can be represented as;

$$e(k) = y_r(k) - y_n(k) \tag{11}$$

where *"$y_{n(k)}$"* represents plant or system output. Rest of the derivation of GD is shown as following;

$$\begin{aligned} w_{i+1} &= w_i - \frac{1}{2}\mu * 2e(k) * \frac{\partial e(k)}{\partial w_i} \\ w_{i+1} &= w_i - \mu * e(k)\left(\frac{\partial y_r(k)}{\partial w_i} - \frac{\partial y_n(k)}{\partial w_i}\right) \\ w_{i+1} &= w_i + \mu * e(k) * \frac{\partial y_n(k)}{\partial w_i} \end{aligned} \tag{12}$$

where " $\frac{\partial y_n(k)}{\partial w_i}$ " represents partial derivatives of the neural inputs respective to each neural weights.

Modification of GD for QNU is same derivation as LNU and it is final formula as follows;

$$colw(k+1) = colw(k) + \mu * e(k) * \frac{\partial y_n(k+1)}{\partial colw(k)} \tag{13}$$

3.1.2 Recursive Least Square Algorithm Adaptation

In this method, main goal is to adapt neural weights again. Structure of linear neural unit *"x"* vector is same as in previous chapter. Adaptation of neural weights are succeed by derivation of RLS formula. Adaptation of neural weights is same process for both neural units therefore general derivation of RLS [13;14] is given as it follows;

$$
\begin{gathered}
w_{(k+1)} = w_{(k)} + \Delta w_{(k)} \\
\Delta w_{(k)} = R_{(k)} * x_{(k)} * e_{(k)} \\
e_{(k)} = y_{r_{(k)}} - y_{n_{(k)}} \\
R_{(k)} = \frac{1}{\mu} * \left(R_{(k-1)} - \frac{R_{(k-1)} * x_{(k)} * x_{(k)}^{T} * R_{(k-1)}}{\mu + x_{(k)}^{T} * R_{(k-1)} * x_{(k)}} \right) \\
R_{(0)} = \frac{1}{\delta} * I
\end{gathered}
\tag{14}
$$

where *"$e_{(k)}$"* is error between neural unit output and real system output, w is weight of neural units. *"$R_{(k)}$"*is inverse autocorrelation matrix, *"I"* is identity matrix, *"δ"* is small positive constant.

3.2 Procedure of Plant Identification Algorithm Application

In previous parts, structure of neural units, and adaptation of algorithms to identification process are explained with formulations. In this section, scheme of MIMO adapted plant identification is shared with implementation of adaptive identification of theoretical models.

The principal schematics of linear MIMO system adaptive identification with neural units can be shown as in Figure 7. Both neural units are fed with MIMO s both inputs and outputs to create input vector for neural units. Their feedback is supported by individual difference of related output and neural output.

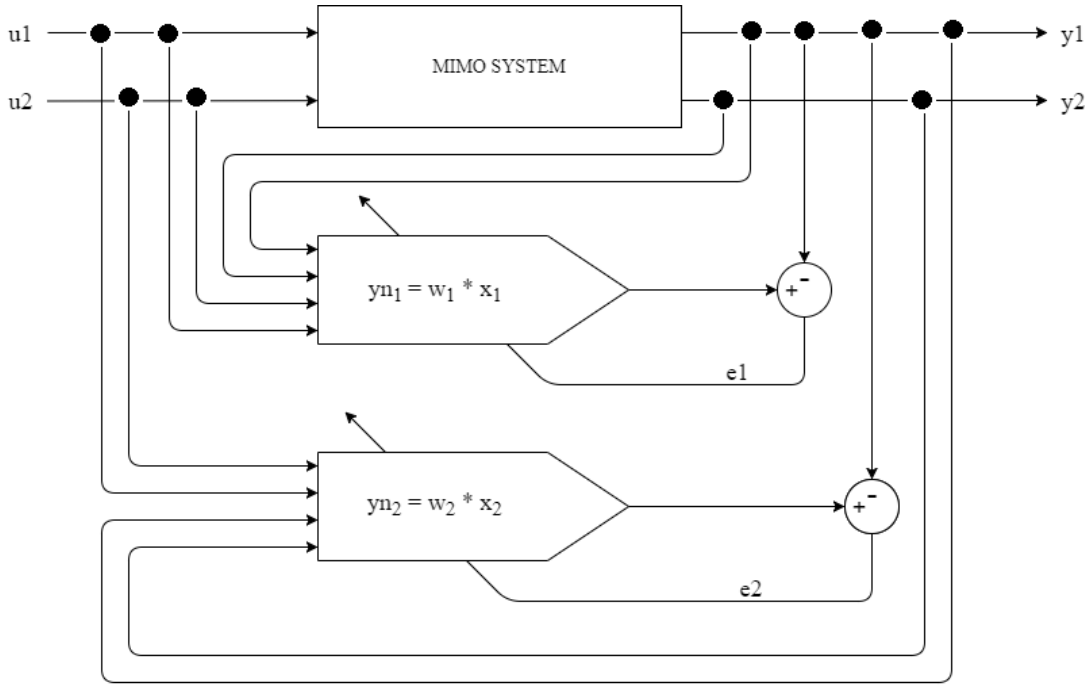


Figure 7: Scheme of adaptive identification for linear MIMO system.

For dynamical models, error value can feed neural unit with difference of neural unit input and output instead of difference of system output and neural unit output.

System identification requires plant data (input and output). This algorithm application is realized on Python 2.7. Therefore theoretical model is inserted to identification

algorithm as online plant. Program works as once for both theoretical plant simulation and identification in terms of neuron long *"**x**"* vector of weights *"**w**"*.

Sampling rate is significant for system identification also for controller, because it has important role on defining size of neuron input vector. Large sampling rates as 200 samples for a second requires 120 previous inputs and outputs to neural unit. Therefore it is not efficient, re-sampling or keeping sample rate low increases efficiency. In this work, sampling rate for plant is kept as 10 samples for per second.

Final important point is learning rate "$\boldsymbol{\mu}$". It corresponds to the speed of learning. High value of "$\boldsymbol{\mu}$" corresponds faster learning process of (identifying) system, lower rate corresponds slower learning process. However, it does not mean for all cases usage of high rate can be efficient. Higher rate of learning can result with instability in the system, so it can be chosen by testing according to algorithm and system. This difference will be also revealed in next sections, because even for same algorithm and system, learning rate must be set because of different system input.

3.3 Plant Identification Algorithm Application

For identification methods, different number of epochs, learning rate but same sampling rate and, same amount of input and output values for neuron long *"x"* vector are used. Because main purpose of this work to obtain best possible results with given algorithms. These epochs and different learning rate values are obtained experimentally. As theoretical models and roller rig models are linear, LNU is used in learning algorithms.

For both learning algorithms, two different inputs are given to system; step and variable step inputs for observation of system under different loads. And for these two different inputs, two different graphs are created by program which illustrates neural output with comparison system input and output, also error values.

In next parts, there will be controller adaptation algorithm. For that purpose, variable step input will be used as system input for simulating real plant behavior realistically however to see identification algorithm performance, step input signal illustrates more observable response in the system.

3.3.1 LNU Algorithm Trained with the GD Method

First illustration in Figure 8 shows 2nd order stable linear theoretical plant response to step input signal.

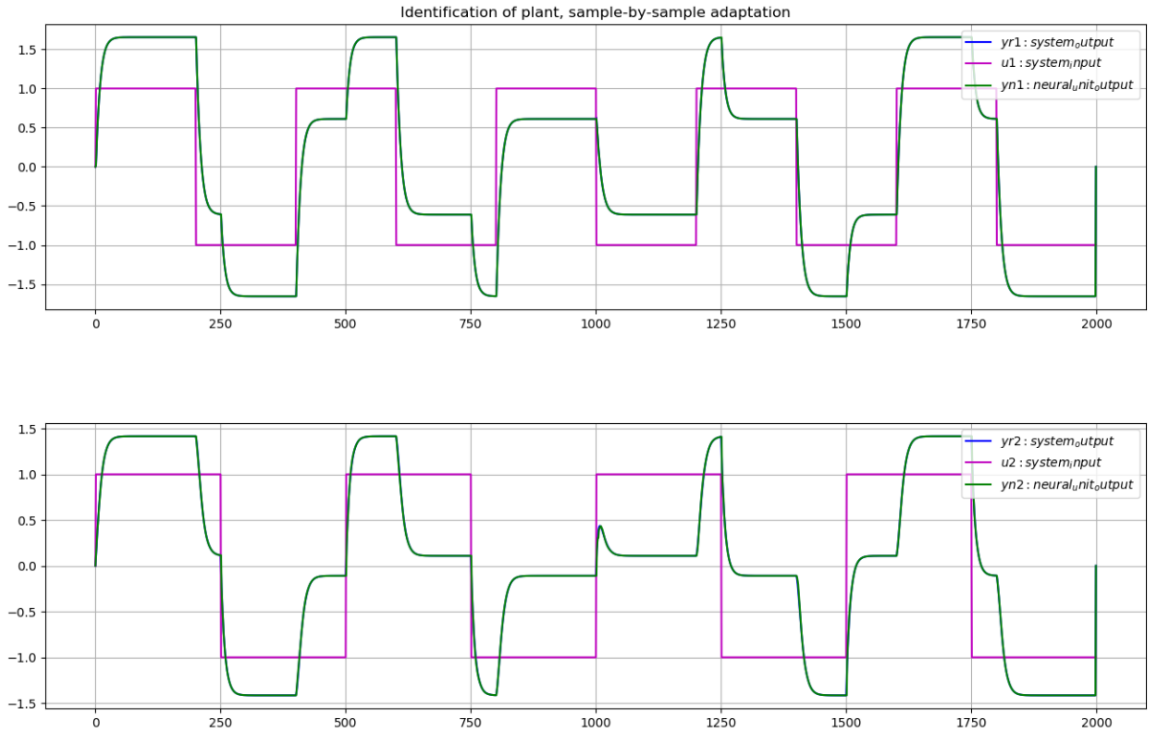


Figure 8: 2nd order plant identification with LNU using GD as response to step input signal.

**Original plant can be seen in Figure 3.*

As if can be seen in Figure 8, system output and neural input are nearly perfectly in same line which means neural-model step by step training works efficiently. Input signal properties are given below;

- For u1; T = 40 sec., W = %50*T
- For u2; T = 50 sec., W = %50*T
- T = 200 sec.
- dT = 0.1

these values are valid for both theoretical models, and their step and variable step input signals.

Error states of this identification process is shown in Figure 9.

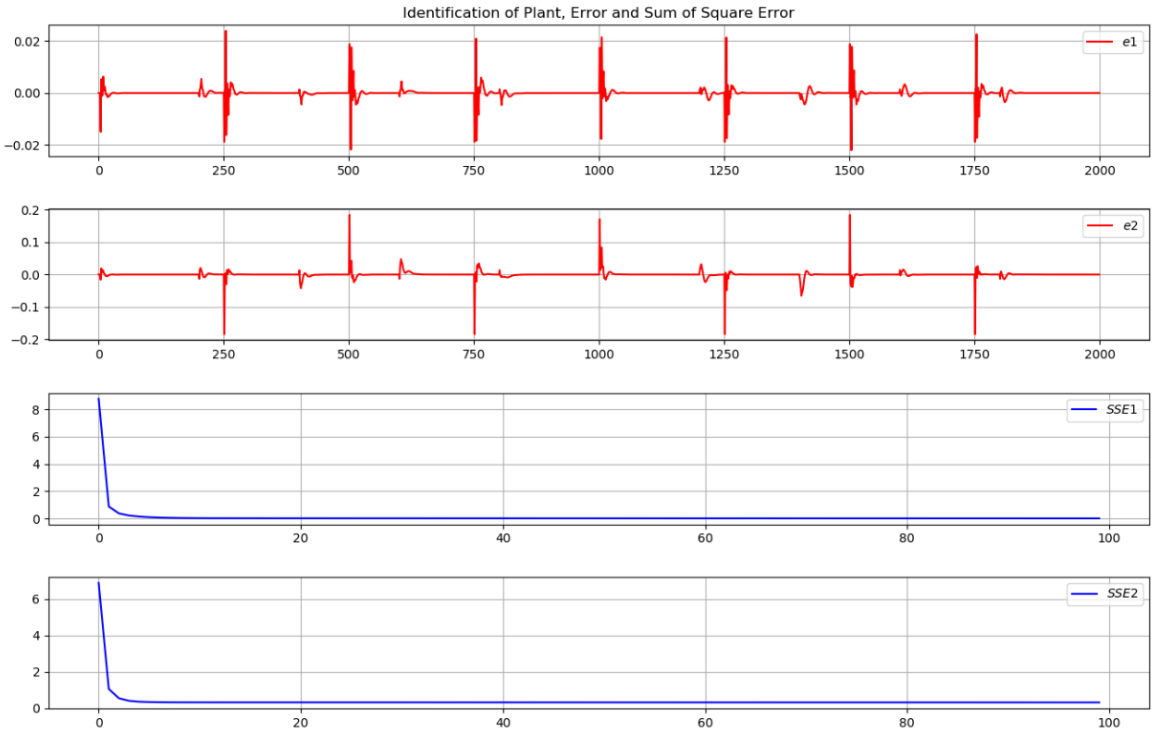


Figure 9: 2nd order plant error of identification with LNU using GD as response to step signal.

**Original plant can be seen in Figure 3.*

Last trained epoch results are given in Table 1 below;

GD	**SSE1**	**SSE2**	**μ**	**Epochs**
LNU	0.0085	0.32	.4	100

Table 1: Details of simulated system of 2nd order with step signal.

As it is visible in Figure 9 and Table 1, SSE results are very small. Learning rate and Epochs are defined by many experiments for best results.

In Figure 10 shows 2nd order stable linear theoretical plant response to variable step signal. Identification of system is even better in comparison with step signal, variable step input response stability is better than step signal response.

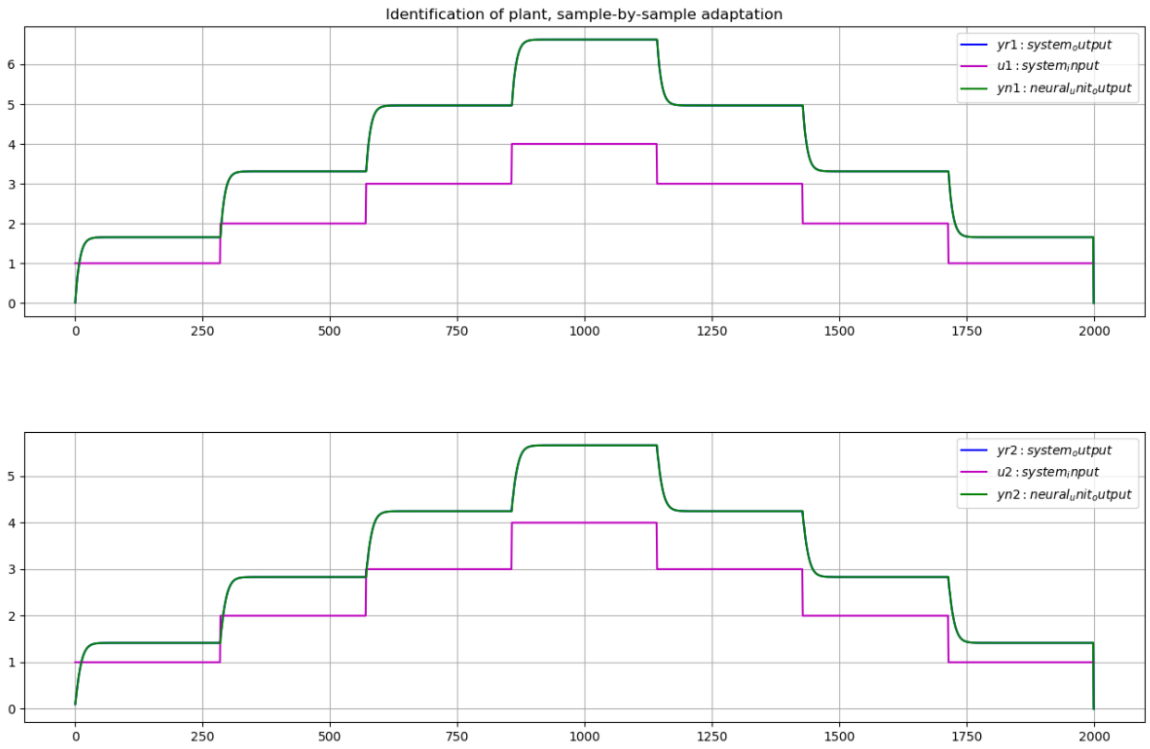


Figure 10: 2nd order plant identification with LNU using GD as response to variable step signal.

**Original plant can be seen in Figure 4.*

Error states of this identification process is shown in Figure 11.

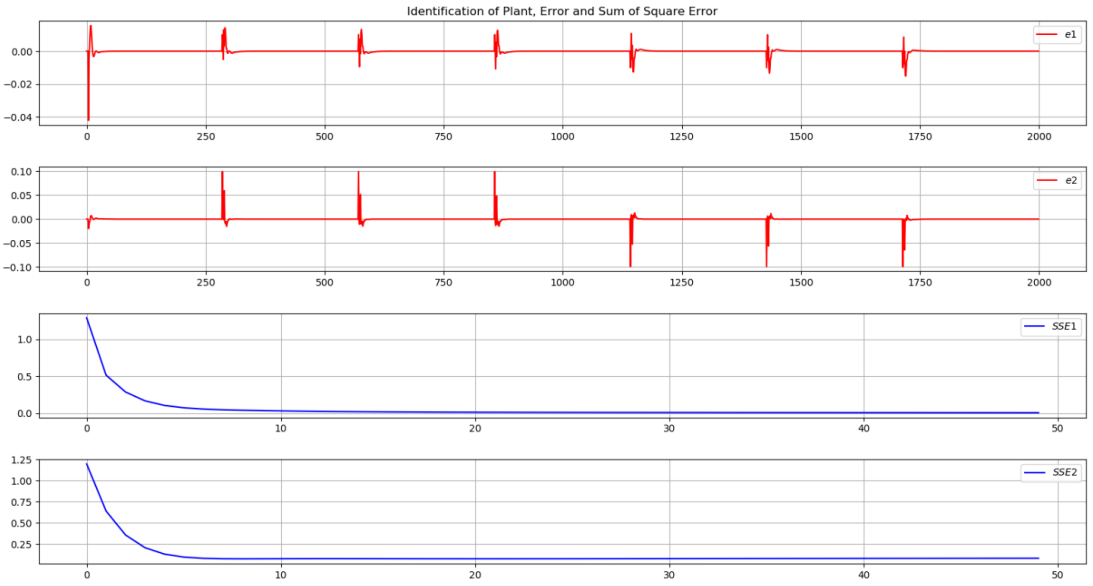


Figure 11: 2nd order plant error of identification with LNU using GD as response to variable step signal.

**Original plant can be seen in Figure 4.*

Last trained epoch results are given in Table 2 below;

GD	SSE1	SSE2	μ	Epochs
LNU	0.0067	0.08	.8	50

Table 2: Details of simulated system of 2nd order with variable signal.

As it is clear in Table 2, variable input signal is better option for system simulation.

So far plant identification for linear MIMO systems results are shown for second order plant. Next illustration Figure 12 represents 4th order oscillating linear theoretical plant response to step input signal.

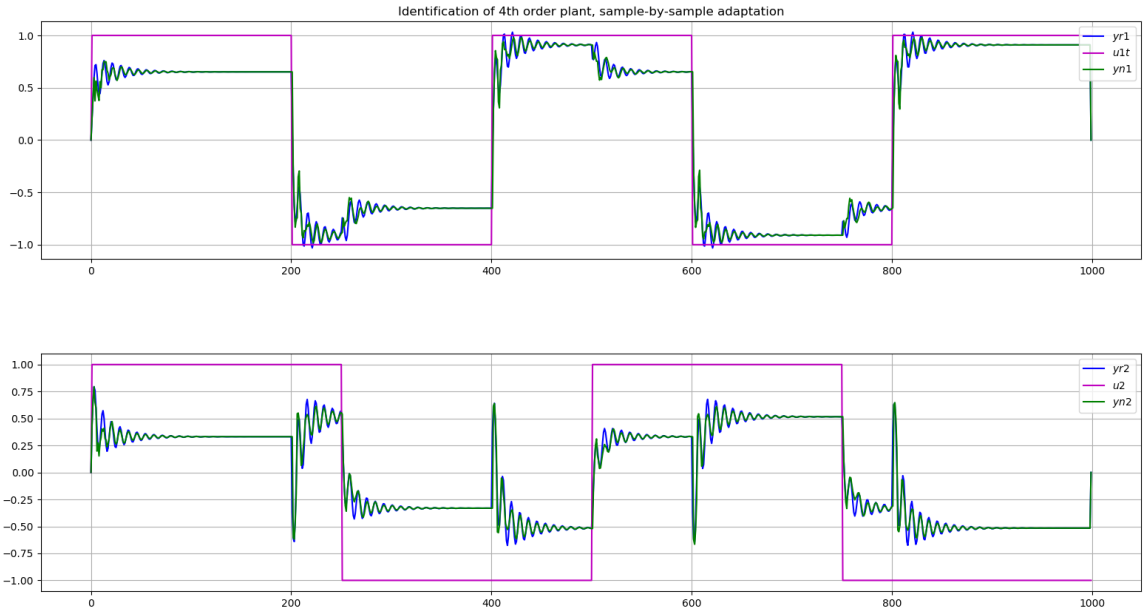


Figure 12: 4th order plant identification with LNU using GD as response to step input signal.

**Original plant can be seen in Figure 5.*

For oscillating systems identification process does not perform good as stable systems. Order of the system and MIMO also effects performance of identification process.

Error states of this identification process is shown in Figure 13.

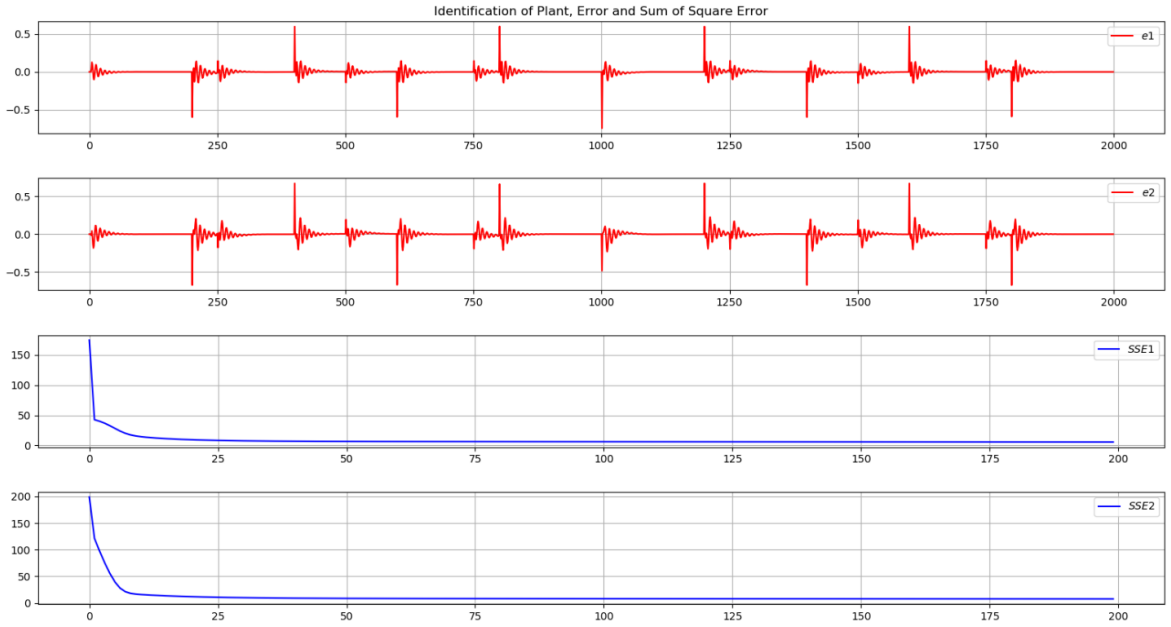


Figure 13:4th order plant error of identification with LNU using GD as response to step signal.

**Original plant can be seen in Figure 5.*

Last trained epoch results are given in Table 3 below;

GD	**SSE1**	**SSE2**	**μ**	**Epochs**
LNU	5.27	7.21	.01	200

Table 3: Details of simulated system of 4th order with step signal.

SSE values are higher in comparison with second order plant, it is due to complexity of plant.

In Figure 14 shows 4th order oscillating linear theoretical plant response to variable step signal. Identification of system is even better in comparison with step signal, variable step input response stability is better than step signal response.

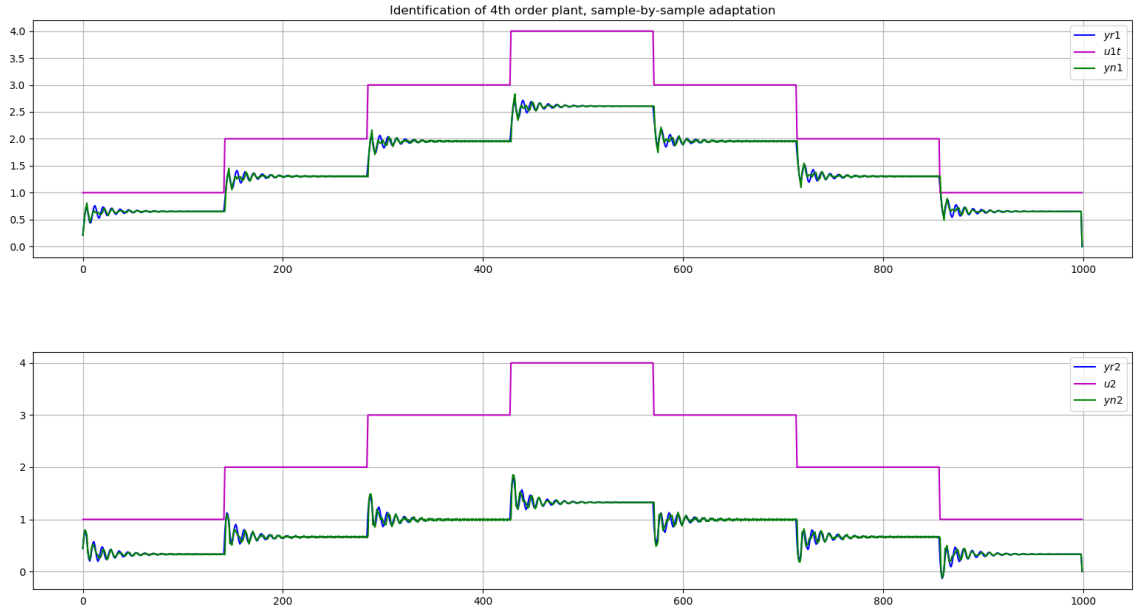


Figure 14:4th order plant identification with LNU using GD as response to variable step signal.

**Original plant can be seen in Figure 6.*

Error states of this identification process is shown in Figure 15.

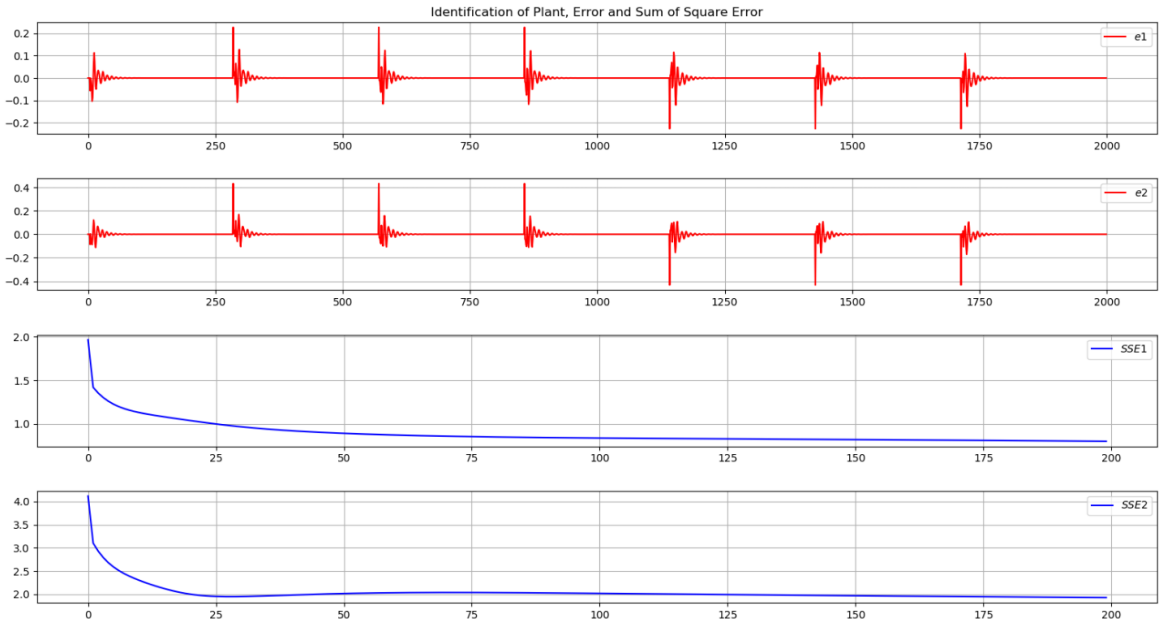


Figure 15: 4th order plant error of identification with LNU using GD as response to variable step signal.

Last trained epoch results are given in Table 4 below;

GD	SSE1	SSE2	μ	Epochs
LNU	0.78	1.91	.3	200

Table 4: Details of simulated system of 4th order with variable step signal.

In comparison with step signal input, learning rate is higher and SSE values are smaller.

3.3.2 LNU Algorithm Trained with the RLS Method

First illustration in Figure 16 shows 2nd order stable linear theoretical plant response to step input signal.

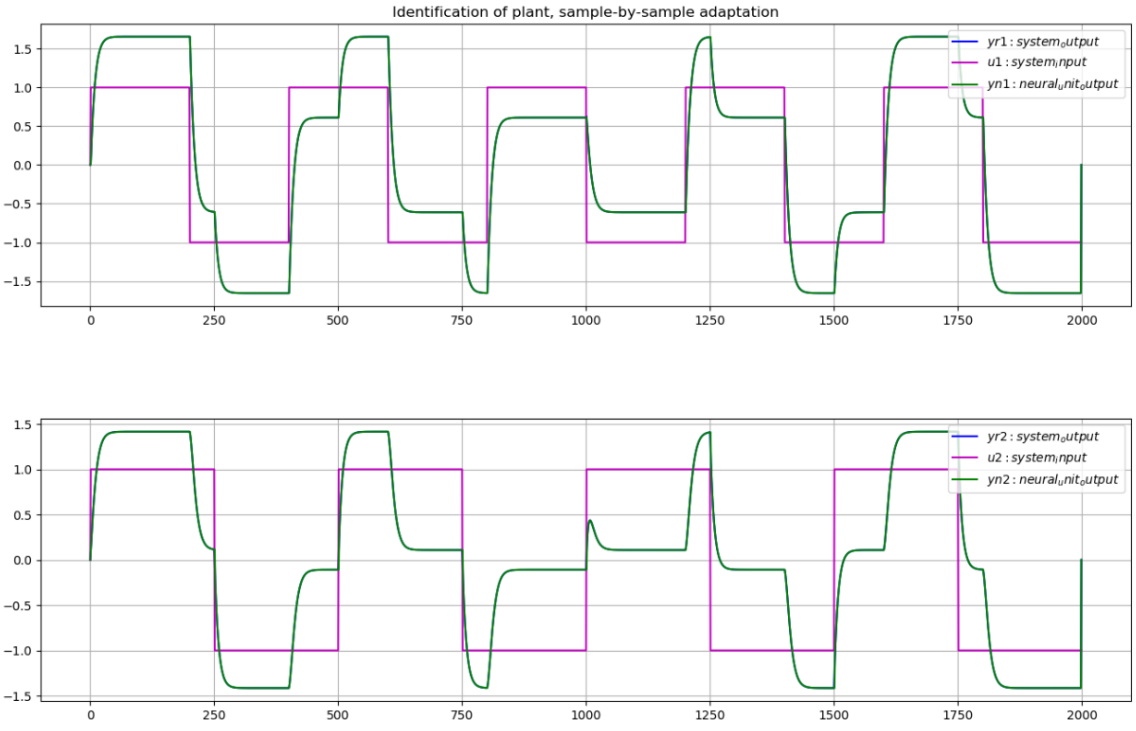


Figure 16: 2nd order plant identification with LNU using RLS as response to step input signal.

**Original plant can be seen in Figure 3.*

As if can be seen in Figure 8, system output and neural input are nearly perfectly in same line which means neural-model step by step training works efficiently.

Error states of this identification process is shown in Figure 17.

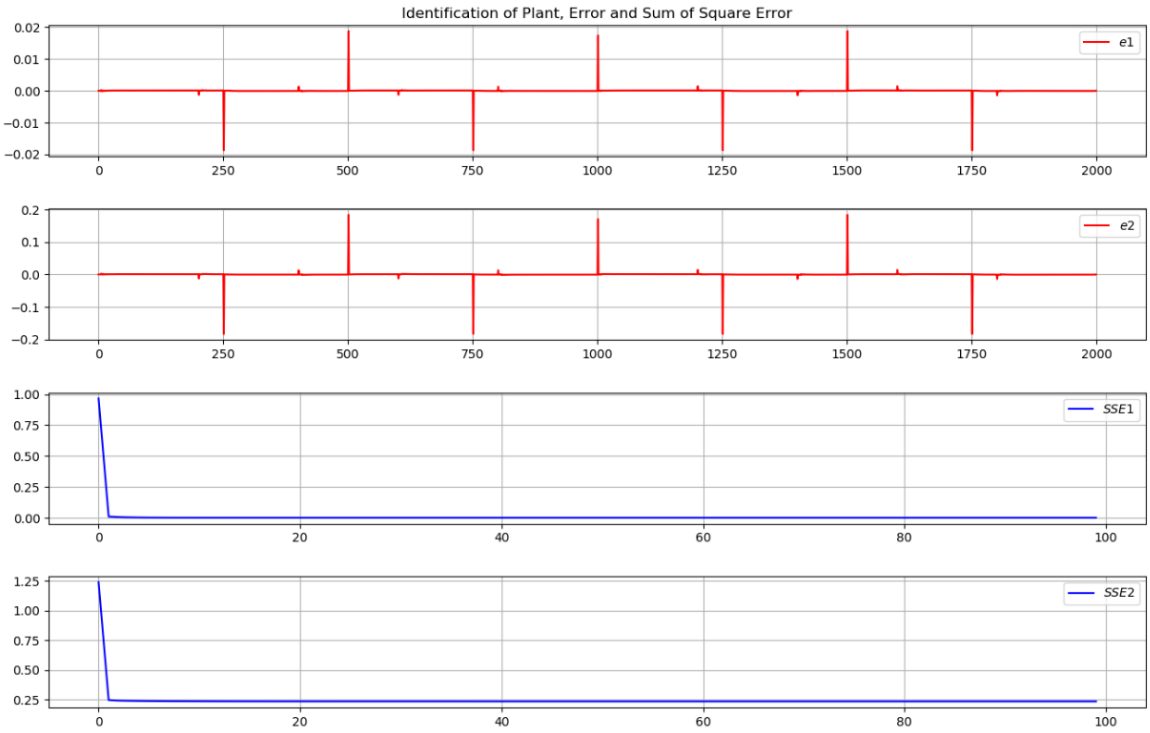


Figure 17: 2nd order plant error of identification with LNU using RLS as response to step signal.

Last trained epoch results are given in Table 5 below;

GD	**SSE1**	**SSE2**	**μ**	**Epochs**
LNU	0.002	0.091	.9998	100

Table 5: Details of simulated system of 2nd order with step signal via RL.

For same conditions in GD algorithm, it can be easily said that for adaptive identification purposes RLS algorithm performs better than GD algorithm.

In Figure 18 shows 2nd order stable linear theoretical plant response to variable step signal.

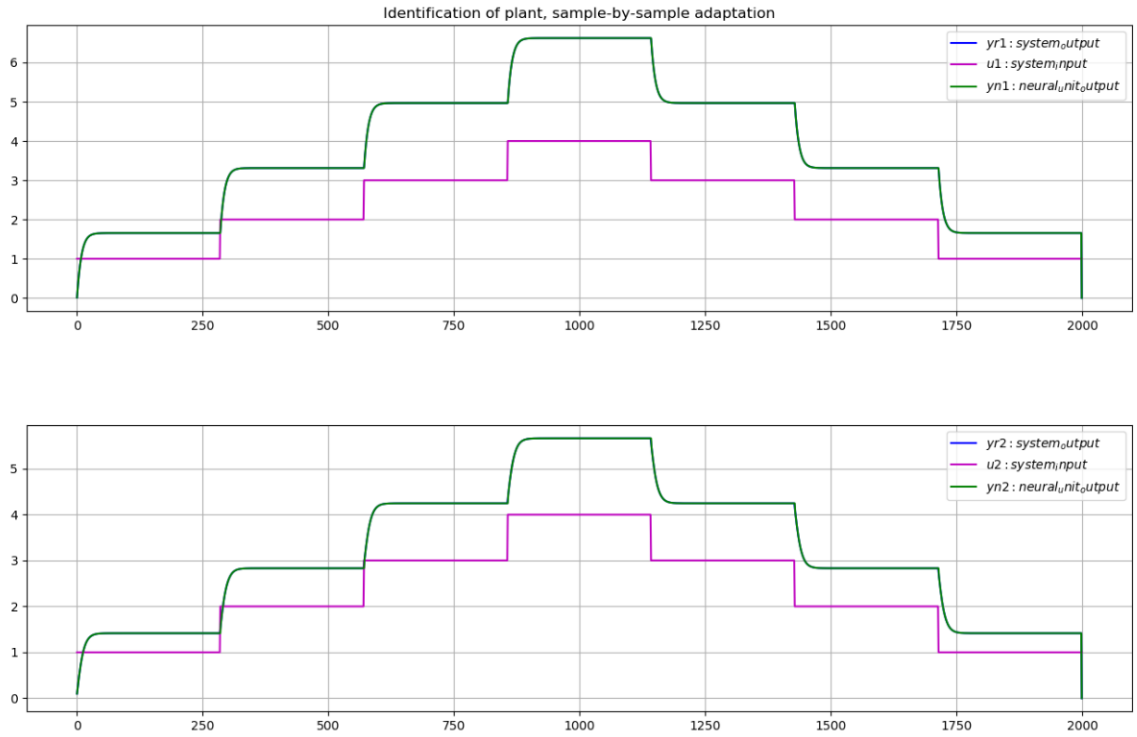


Figure 18: 2nd order plant identification with LNU using RLS as response to variable step signal.

**Original plant can be seen in Figure 4.*

Error states of this identification process is shown in Figure 19.

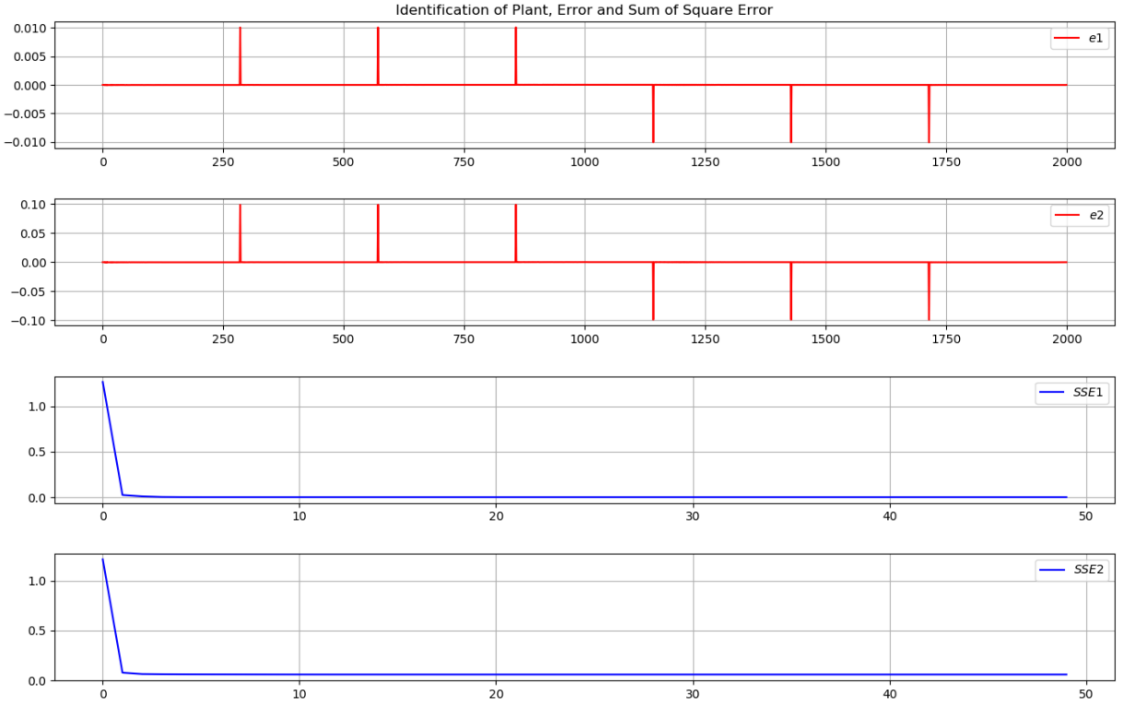


Figure 19: 2nd order plant error of identification with LNU using RLS as response to variable step signal.

Last trained epoch results are given in Table 6 below;

GD	SSE1	SSE2	μ	Epochs
LNU	0.0006	0.038	.9998	50

Table 6: Details of simulated system of 2nd order with variable step signal via RLS.

So far plant identification for linear MIMO systems results are shown for second order plant. Next illustration Figure 20 represents 4th order oscillating linear theoretical plant response to step input signal.

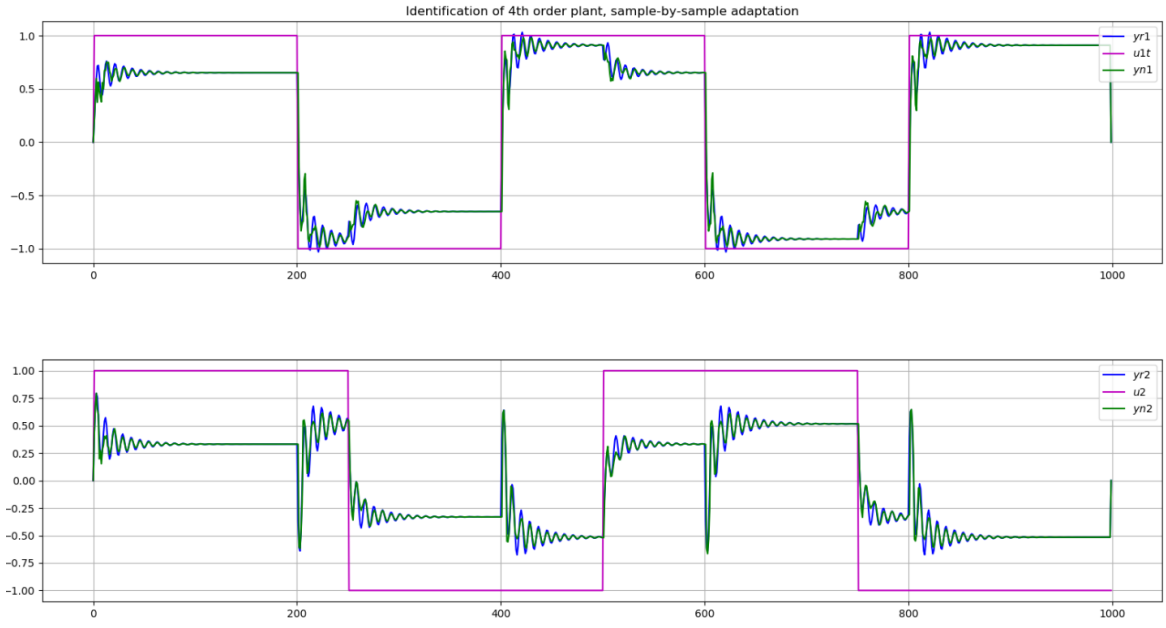


Figure 20: 4th order plant identification with LNU using RLS as response to step input signal.

**Original plant can be seen in Figure 5.*

Error states of this identification process is shown in Figure 21.

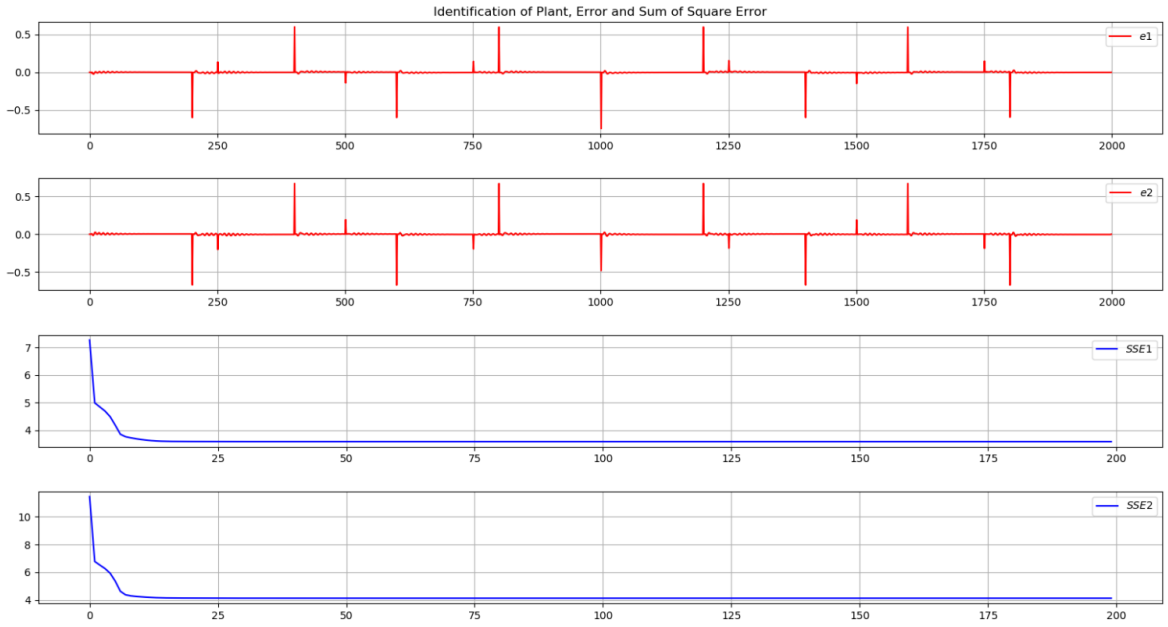


Figure 21:4th order plant error of identification with LNU using RLS as response to step signal.

Last trained epoch results are given in Table 7 below;

GD	**SSE1**	**SSE2**	**μ**	**Epochs**
LNU	3.52	4.12	.9998	200

Table 7: Details of simulated system of 4th order with step signal via RLS.

For same conditions in GD algorithm, it can be easily said that RLS performs better than GD for identification approach. As last identification results, in Figure 22 illustrates 4th order oscillating linear theoretical plant response to variable step signal.

Last trained epoch results for variable step response are given in Table 8 below;

GD	**SSE1**	**SSE2**	**μ**	**Epochs**
LNU	0.30	1.12	0.9998	50

Table 8: Details of simulated system of 4th order with variable signal via RLS.

From all of the results, it can be easily said that RLS algorithm performs better than GD for adaptive identification process.

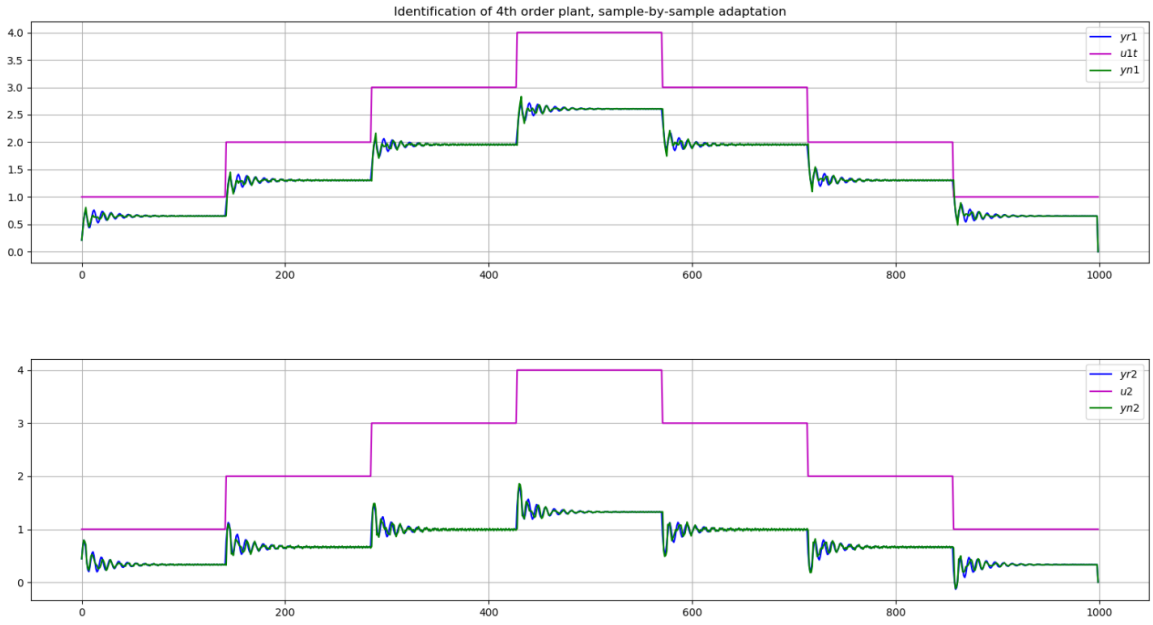


Figure 22: 4th order plant identification with LNU using RLS as response to variable step signal.

**Original plant can be seen in Figure 6.*

Error states of this identification process is shown in Figure 23.

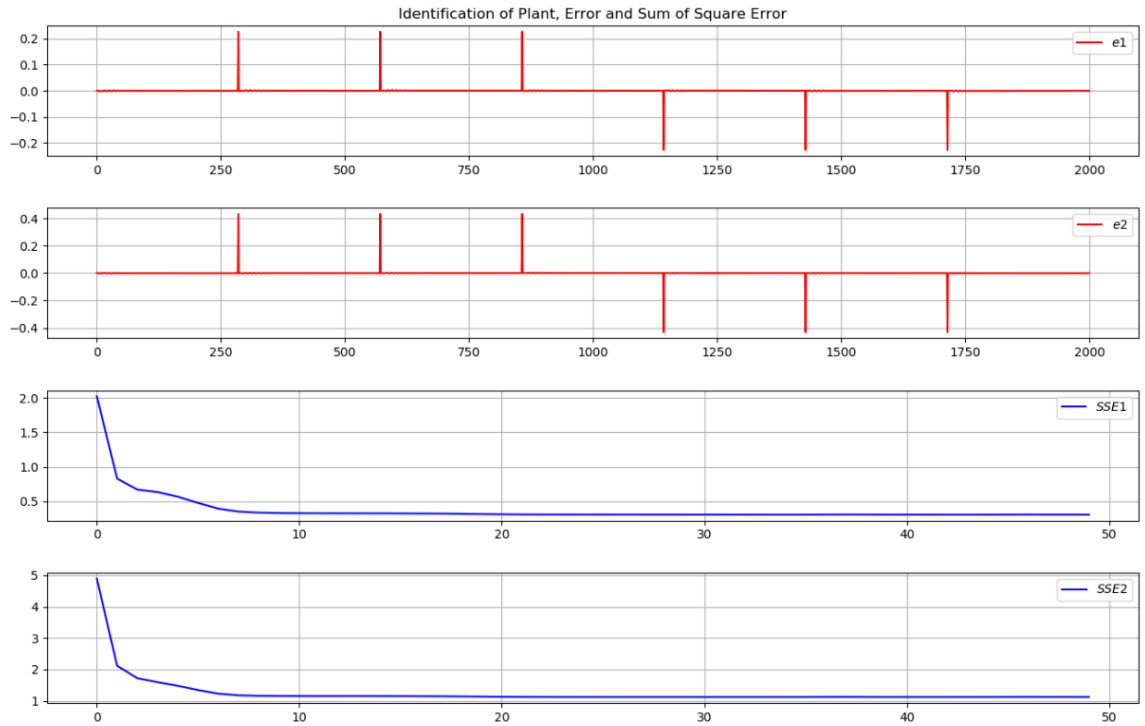


Figure 23: 4th order plant error of identification with LNU using RLS as response to variable step signal.

4. Neuro Controller

Concept of model reference adaptive controller [15] is same as plant identification algorithm. Main difference is neuro controller is used for manipulating newly feed input into neural unit for control. But here neural unit is used as a model itself. For linear MIMO systems, complete scheme of neural controller can be illustrated as in Figure 24. This scheme included all process for this work.

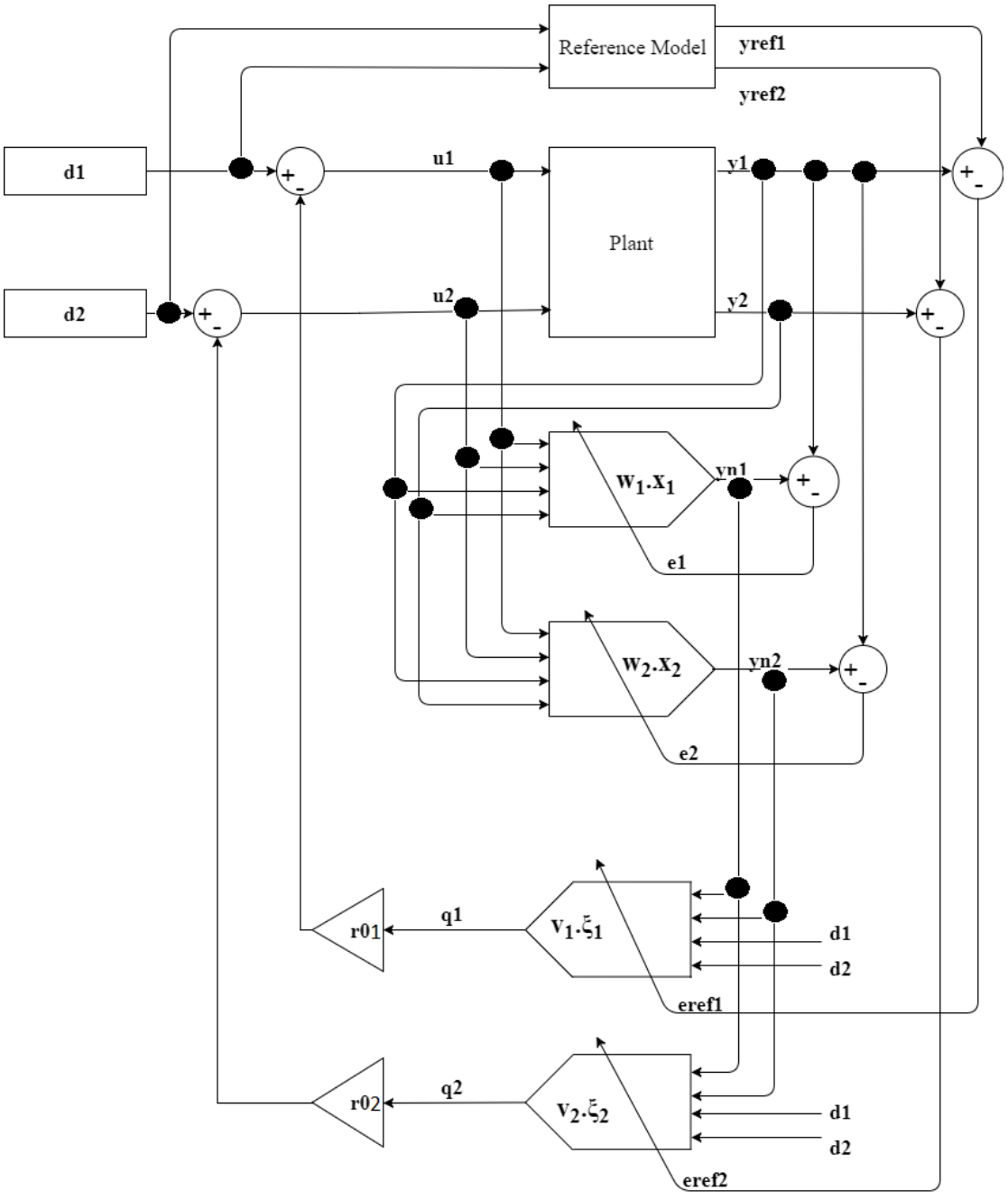


Figure 24: Scheme of neuro controller with all other system parts.

where "q" is neuro controller output, its structure same as "y_n", "v" is neural weights to show difference is denoted with different letter. "ξ" is input vector for neural unit its structure is similar with "x" vector. Neuro controller equation yields as;

$$q = v_0 * \xi_0 + v_1 * \xi_1 + v_2 * \xi_2 + \cdots + v_n * \xi_n = v * \xi \tag{15}$$

Update rule for neural weights are as following;

$$v_{i+1} = v_i + \mu_c * e_{ref}(k) * \frac{\partial y_n(k)}{\partial v_i} \tag{16}$$

where "e_{ref}" is an error value between reference model and plant, "r_0" is neuro controller gain value. For this value adaptation also possibly can be used but in this work, ideal gain value is obtained after multiple test run. Key variable of the equation is "$\frac{\partial y_n(k)}{\partial v_i}$". Since it provides dependence of neural model to output of controller[12]. As it follows;

$$\frac{\partial y_n(k)}{\partial v_i} = \frac{\partial (w.x(k))}{\partial v_i} = w * \frac{\partial x(k)}{\partial v_i} \tag{17}$$

$$\frac{\partial x(k)}{\partial v_i} = \begin{bmatrix} 1 \\ y(k) \\ y(k-1) \\ \dots \\ \big(d(k) - q(k)\big) * r0 \\ \big(d(k-1) - q(k-1)\big) * r0 \\ \dots \end{bmatrix} = -r0 * \frac{\partial}{\partial v_i} * \begin{bmatrix} 0 \\ 0 \\ 0 \\ \dots \\ q(k) \\ q(k-1) \\ \dots \end{bmatrix} \tag{18}$$

$$= -r0 * \begin{bmatrix} 0 \\ 0 \\ 0 \\ \dots \\ \xi(k) \\ \xi(k-1) \\ \dots \end{bmatrix}$$

Structure of "ξ" vector is;

$$\xi = [yn(k) \quad yn(k-1) \quad \ldots \quad d(k) \quad d(k-1) \quad \ldots. \quad] \tag{19}$$

Size of "ξ" array is same as "*x"* vector.

4.1 Procedure of Controller Implementation

Implementation of control algorithm is quite same as identification process however there are still few difference. Main idea behind neuro controller is to manipulate real plant to act as reference model. In this case, reference model can be declared as ideal response of plant to input signal.

If it is explained in detail, real plant responses to given input, these output and input values are inserted to create input vector of neural unit. This process allows neural unit to define proper weights to act as real plant, in this point, neural unit becomes model which means simulated plant. After neural model output feeds neuro controller unit with desired value which is same as input value in many cases. Neuro controller update rule is provided by "e_{ref}*"* which defines error value between plant and reference model. Finally neuro controller yields its output value "*q"*. This value is multiplied by gain than it is subtracted by desired value to control system.

Important point all of these processes is that; both neural unit and neuro controller is fed by both input and output values. Although there are double neural units and neuro controller, system behavior depends on both inputs and output value, so without them, identification and controller process can not perform. This specifies MIMO plants systems need for adaptive identification and control with neural units.

As final, importance of reference model is explained above. Here reference model for precise controller tuning is given as a filter to desirable value. it can be seen in Figure 25.

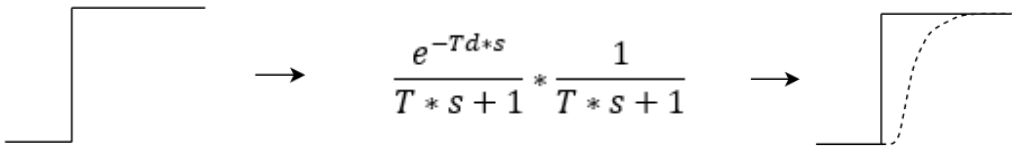


Figure 25: Principal of reference model

4.2 Neuro Controller Application

In this section, neuro controller algorithm is applied to theoretical models which are created in previous part. All system works dynamically, weights for neural unit are not set as constant. Algorithm is implemented with continuous training of neural weights for all process. This method lets controller fast adaptation to changes in the system behavior, system simulation identification and control occur in same sampling rate, input values and sampling rate are same as previous parts only difference is variable step signal used as system input. Both GD and RLS algorithms are used for neuro controller[16]. After implementation of these algorithms, the one yields better results is elected to use for 20th order theoretical plant.

4.2.1 2nd Order Theoretical Plant Neuro Controller Application

2nd order stable linear theoretical model is controlled by neuro controller. Result is shown in Figure 26.

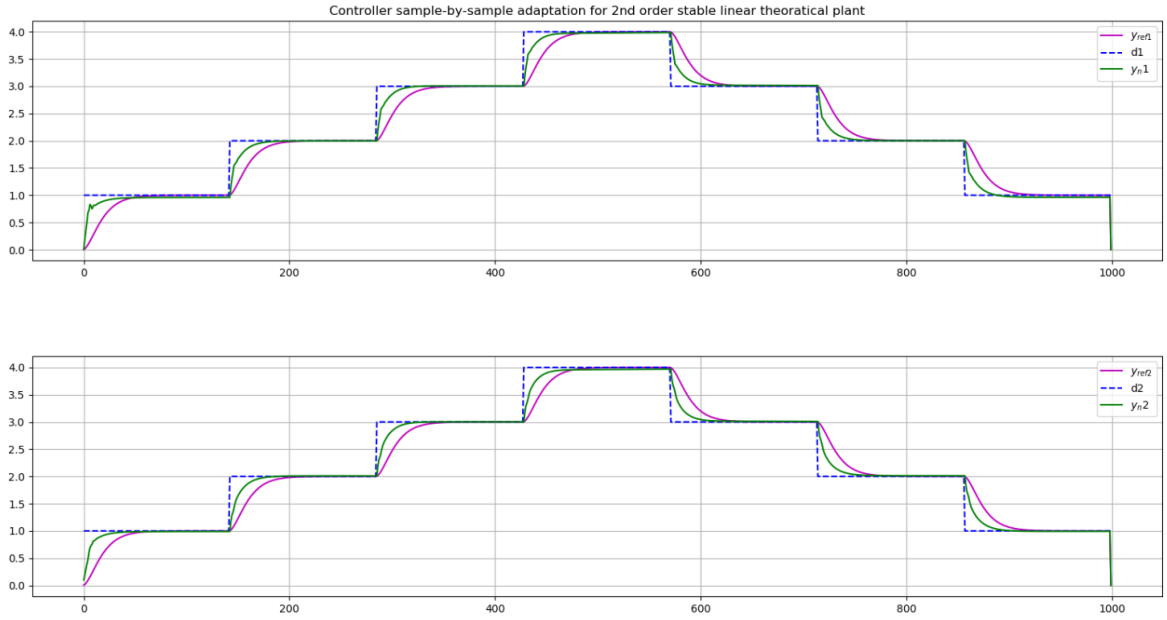


Figure 26: 2nd order theoretical plant controller with continues weights training via GD.(Original plant can be seen in Figure 4.)

System initial conditions and error values are given in Table 9. Magnified look to controller results are shown in Figure 27.

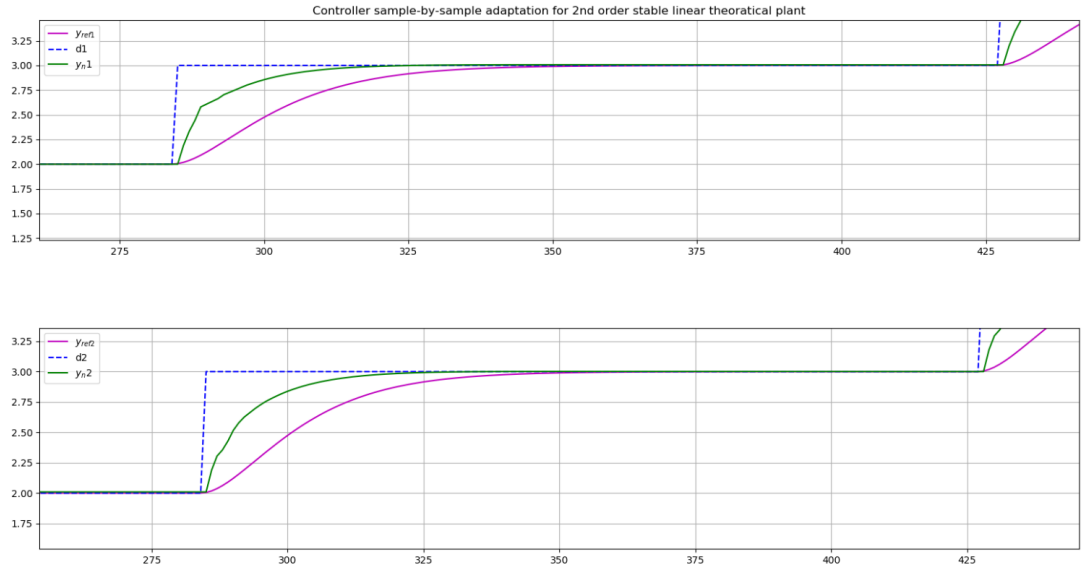


Figure 27: Magnified view of Figure 26.(Original plant can be seen in Figure 4.)

GD	**SSE1**	**SSE2**	**μ**	**Epochs**	**μ_c**	**r01**	**r02**
LNU	2,32	4,28	0.5	2000	0.3	0.009	0.015

Table 9: Details of controlled system of 2nd order with variable signal via GD.

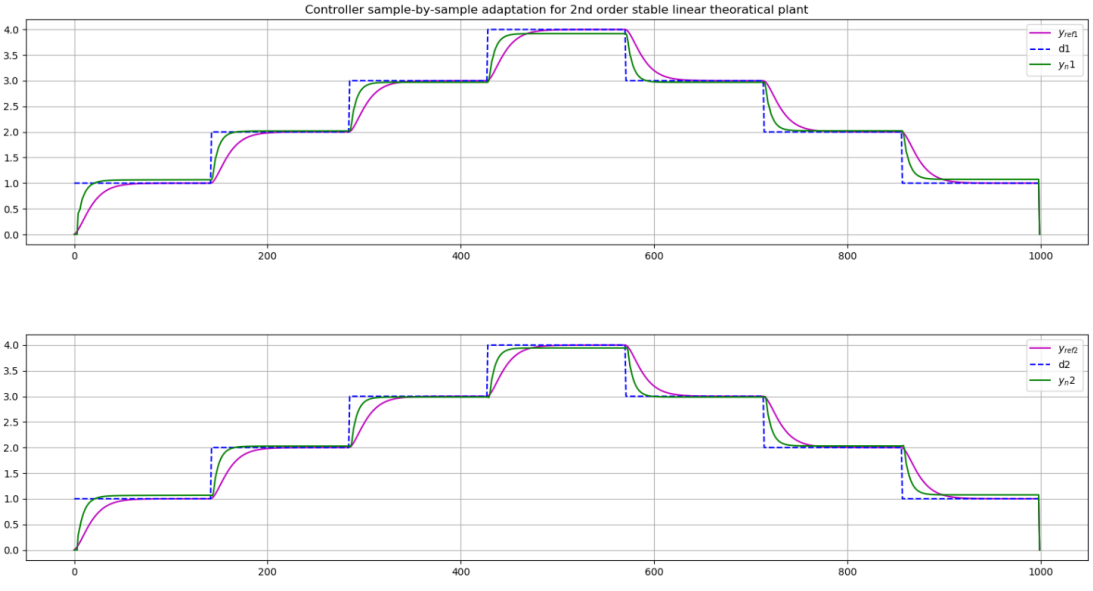


Figure 28: 2nd order theoretical plant controller with continues weights training via RLS. (Original plant can be seen in Figure 4.)

System initial conditions and error values are given in Table 10. Magnified look to controller results are shown in Figure 29.

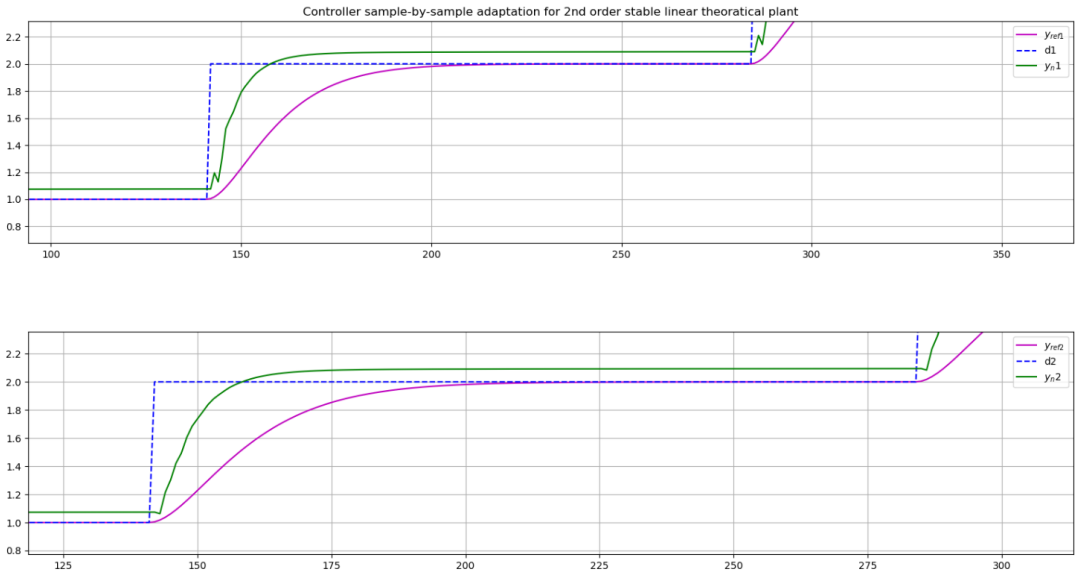


Figure 29: Magnified view of Figure 28. (Original plant can be seen in Figure 4.)

RLS	**SSE1**	**SSE2**	**μ**	**Epochs**	**Epochsc**	**μ_c**	**r01**	**r02**
LNU	37,92	36,58	0.5	200	2	0.9998	0.05	0.05

Table 10: Details of controlled system of 2nd order with variable signal via RLS.

It is clearly seen that GD algorithm for linear MIMO system controller implementation yields better result. RLS algorithm did not perform as good as GD, results can be read in Table 10. Especially oscillating system results were extremely unstable that is the reason they are not add in thesis work however implementation of it is add to Appendix. Main goal was so far to identify ideal learning and controller rule for linear MIMO system, GD algorithm is carried out for next section.

In addition, as it can be seen in Table 9, SSE values are nice. Reference model is chosen as second order linear system. Choosing second order linear system as system reference model yields better result because of its smooth transition. In addition, due to system complexity, it is complicated to create reference model. According to gained experience of this work, linear MIMO systems reference model could be plant itself but manipulated version by conventional controller or it could be desired value itself. In other word, there is no need for reference value in case of feeding neuro controller with desired value and neural unit difference. But in such case transition is not smooth so it

can cause instability in controller. To avoid all possible problems, for this work second order plant is chosen as reference model after testing performance of all mentioned reference model implementation.

4.2.2 4th Order Theoretical Plant Neuro Controller Application

4th order oscillating linear theoretical model is controlled by neuro controller. Result is shown in Figure 28.

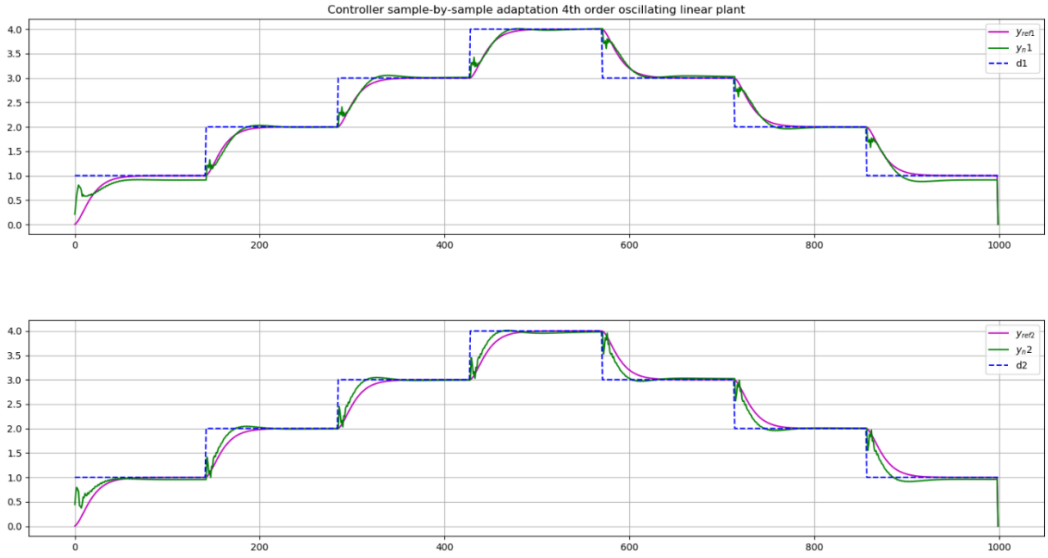


Figure 30: 4th order Theoretical plant controller with continues weights training via GD. (Original plant can be seen in Figure 6.)

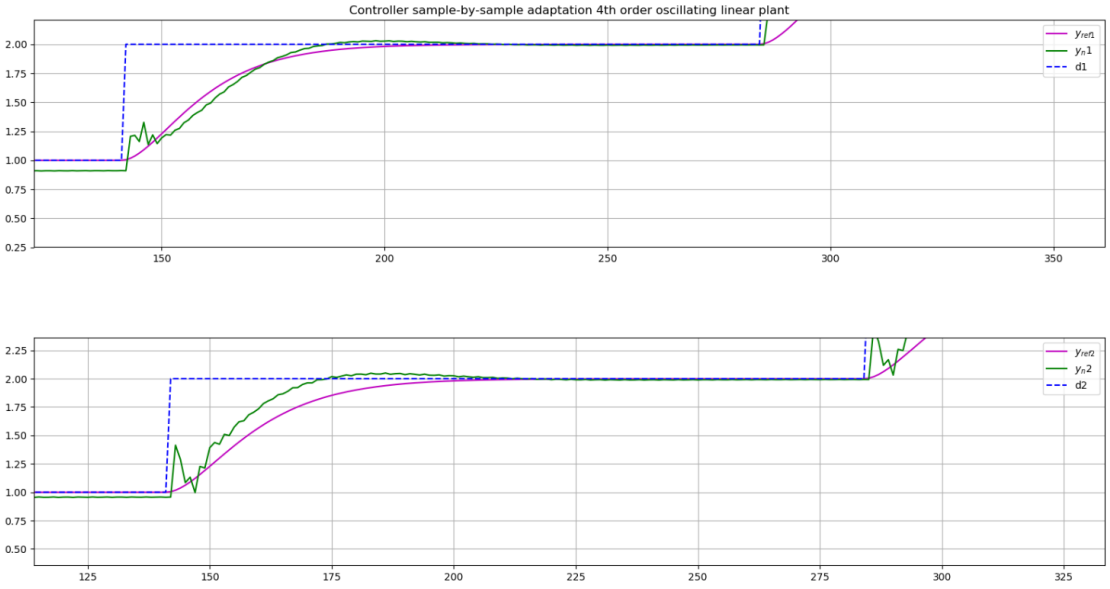


Figure 31: Magnified view of Figure 30. (Original plant can be seen in Figure 6.)

System initial conditions and error values are given in Table 10. Magnified look to controller results are shown in Figure 29.

GD	**SSE1**	**SSE2**	**μ**	**Epochs**	**μ_c**	**r01**	**r02**
LNU	4,32	2,28	0.5	2000	0.5	0.013	0.016

Table 10: Details of controlled system of 4th order with variable signal via GD.

Magnitude of errors are also good. As it is explained in previous section, second order plant is used as reference model. Real plant is oscillating so it results with overshooting in plant.

4.3 Discussion

For better achievements, it is observed that using different learning rates for different neural units could be the key point, however it can cause instability of the system. So relation between different learning rates must set with numerical relation between them. In addition more efficient sampling interval, and length of neural inputs. Also different learning algorithms can be used such as batch training approach Levenberg-Marquardt (LM) especially for offline plant tuning.

Another important point is defining neuro controller gain *"r_0"* value, it is chosen in this thesis work according to theoretical system gains, however for better controller achievements it is possible to derive adaptation rule for *"r_0"*.

The last point is about weights, GD algorithm can be used for both online and offline tuning of neuro controller. The best way of it, pre-training weights before tuning controller however this is possible for offline plant controller.

In next section, discussed algorithms for identification and control process are applied to 20th order theoretical model. According to experience of this work so far, it is chosen that GD algorithm will be used for training weights continuously.

5. Actively Actuated Double Wheel Set Roller Rig

Actively actuated double wheel set roller rig is one of the project which has studied in Czech Technical University in Prague (CTU). It is basically explained in source [17] as;

" Active control of wheelset guidance is regarded as a promising solution for railway vehicles of the future. Although computational simulations published in number of studies, show that active wheelset guidance offers superior properties in comparison with conventional, passive designs, track tests of such a system on vehicles are rather rare. On the other hand on a roller rig it is possible to test railway vehicle running dynamics."

One of the main goal for control process is to prevent wheel flange and rail head contact on railway carriages, with independently rotating wheel sets. System can be seen as in Figure 30.

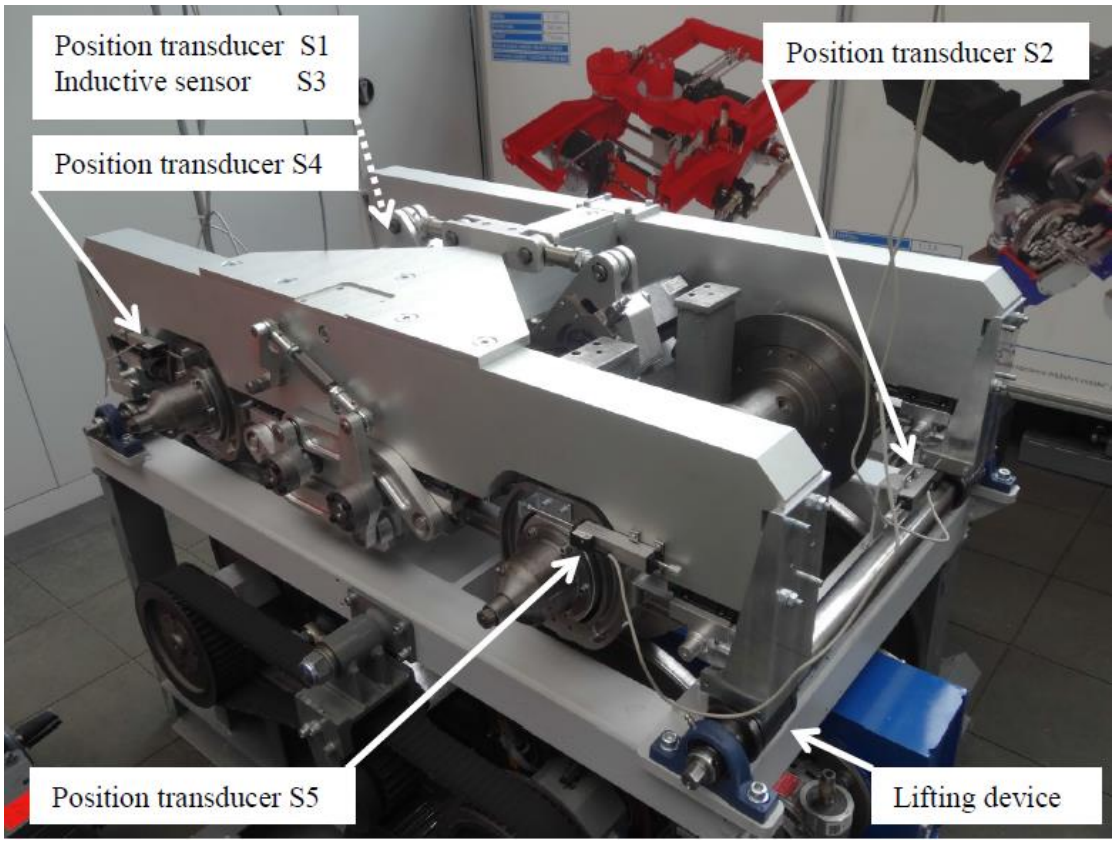


Figure 32: Actively actuated double wheel set roller rig (courtesy of [17]).

Unfortunately, system was in repair so it was not possible to observe system properly, none of datum are taken from the roller rig. Instead of real data, mathematical model of system is given from the source [17]. Model of the system is given as 20^{th} order in space state form. However system was extremely unstable. And it was not possible to stabilize system with conventional controller techniques. Roller rig model space state form eigen values of matrix A is given in Table11. Unstable parts are written in red color.

-2.20385581e+02+1521.89549136j	-2.20385581e+02-1521.89549136
-2.43060266e+02+1655.83731627j	-2.43060266e+02-1655.83731627j
-3.01146497e+02+1858.81351718j	-3.01146497e+02-1858.81351718j
-2.85457410e+02+1825.78333622j	-2.85457410e+02-1825.78333622j
-1.04138625e+03 +0.j	-1.04138625e+03 -0.j
-1.58218868e+02 +24.07876876j	-1.58218868e+02 -24.07876876j
-1.38241861e+02 +21.6711047j	-1.38241861e+02 -21.6711047j
4.86260449e+00 +27.82821513j	4.86260449e+00 -27.82821513j
5.40308376e+00 +27.36325273j	5.40308376e+00 -27.36325273j
-1.06718544e-04 +0.j	9.42039509e-05 -0.j

Table 11. Eigen values of roller rig space state model.

Step response to system can be seen in Appendix, in “unstable responses part”. Poles, zeros graph of eigen values are shown in Figure 33.

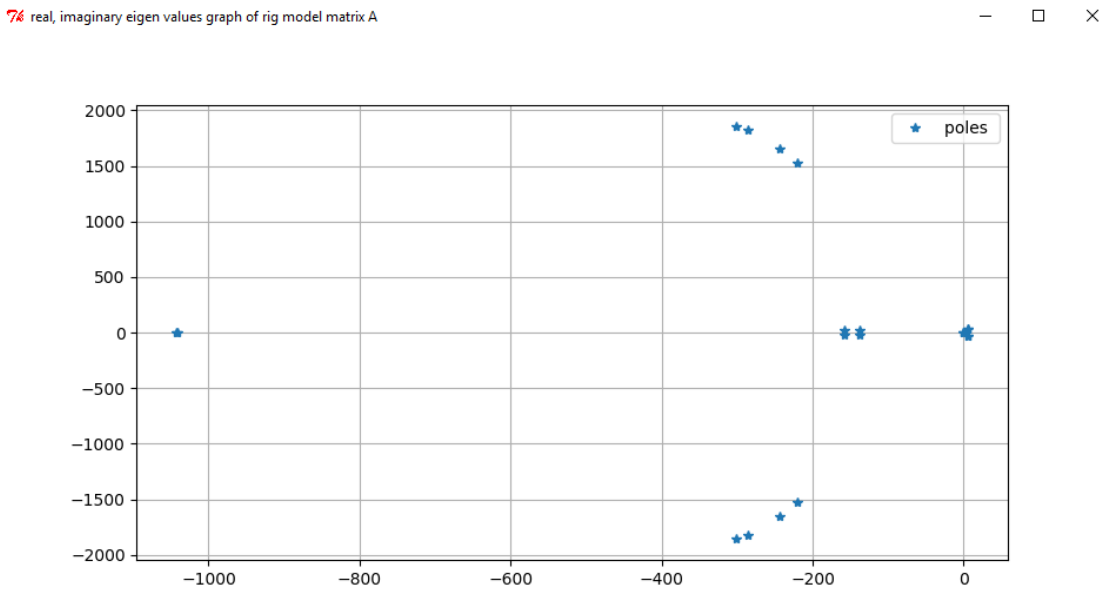


Figure 33: Poles, zeros graph of eigen values of rig model.

System is tried to be stabilize by changing unstable parts, and new eigen values of system is calculated as in Table12.

-6.26410962e+03 +0.j	-6.26410962e+03 +0.j
-3.33756797e+02+1403.70483194j	-3.33756797e+02-1403.70483194j
-3.02564040e+02+1612.04266345j	-3.02564040e+02-1612.04266345j
-3.06665307e+02+1846.61822734j	-3.06665307e+02-1846.61822734j
-2.88776050e+02+1824.6965927j	-2.88776050e+02-1824.6965927j
-7.25051377e+02 +0.j	-7.25051377e+02 +0.j
-9.19880084e+02 +0.j	-9.19880084e+02 +0.j
-8.37749418e-01 +5.43925808j	-8.37749418e-01 -5.43925808j
-1.66984168e+00 +4.49961517j	-1.66984168e+00 -4.49961517j
-5.65632980e-07 +0.j	-5.65632980e-07 +0.j

Table 12. Eigen values of roller rig space state model after replacement of poles.

Step response to system can be seen in Appendix, in “unstable responses part”. Poles, zeros graph of eigen values are shown in Figure 34.

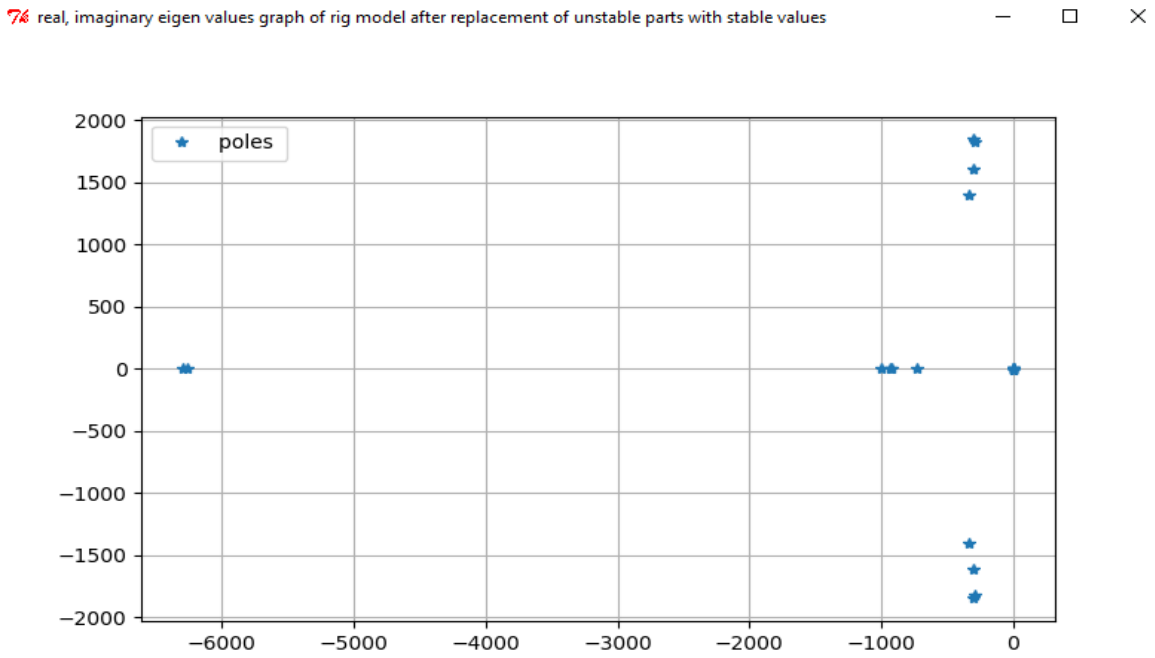


Figure 33: Poles, zeros graph of eigen values of rig model after replacement of poles.

Although, eigen values are in stable region after poles replacement system was still unstable. My main task was not try to stabilize such kind of system with neural

algorithms and also due to goal of this thesis and time limitation, after discussion with my supervisor, I decided to use 20^{th} order theoretical plant to show performance of linear MIMO model reference adaptive control and its possibility for implementing on roller rig.

6. 20th Order Theoretical Plant

This plant is created to illustrate linear MIMO model reference adaptive control performance, its degree is chosen to approach 20th order roller rig model, due to stability issues, roller rig was not able to be used. For simulation of system, variable step signal is used as inputs value, system response is given in Figure 34.

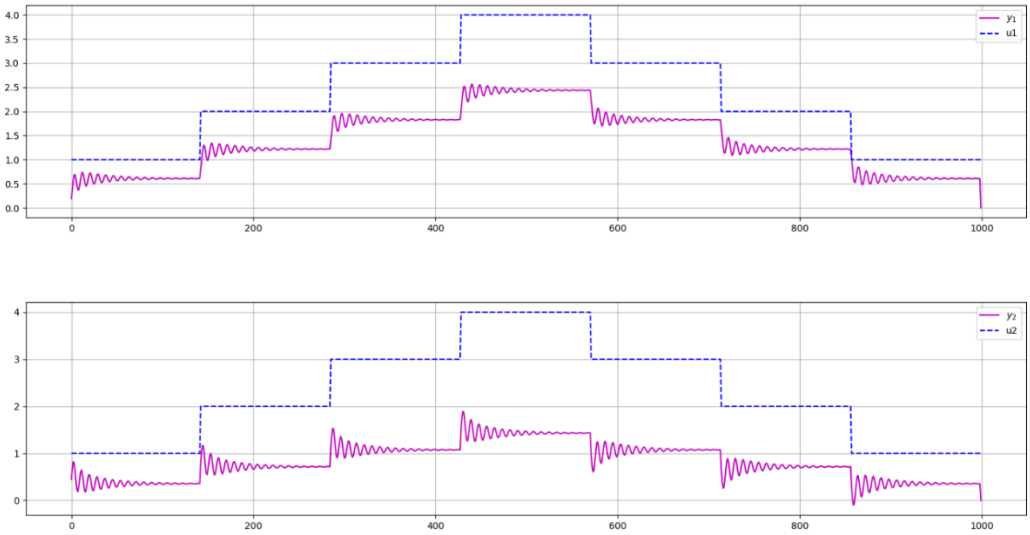


Figure 34: Theoretical 20th order system response to variable input signals.

Eigen values of plant is given in Table 13.

-0.33645671+14.05255636j	-0.33645671-14.05255636j
-3.05654692 +9.01803421j	-3.05654692 -9.01803421j
-10.49059971 +0.j	-0.79787216 +5.21286789j
-0.79787216 -5.21286789j	-1.44958056 +3.099382j
-1.44958056 -3.099382j	-4.63026689 +0.j
-4.63055112 +0.j	-2.11293254 +0.j
-1.08954236 +0.j	-0.82498758 +0.j
-0.22529631 +0.j	-0.34602196 +0.j
-0.54032827 +0.j	-0.47590645 +0.j
-0.46316735 +0.j	-0.46316735 +0.j

Table 13. Eigen values of 20th order theoretical plant.

Plant poles and zeros graph is shown in Figure 35.

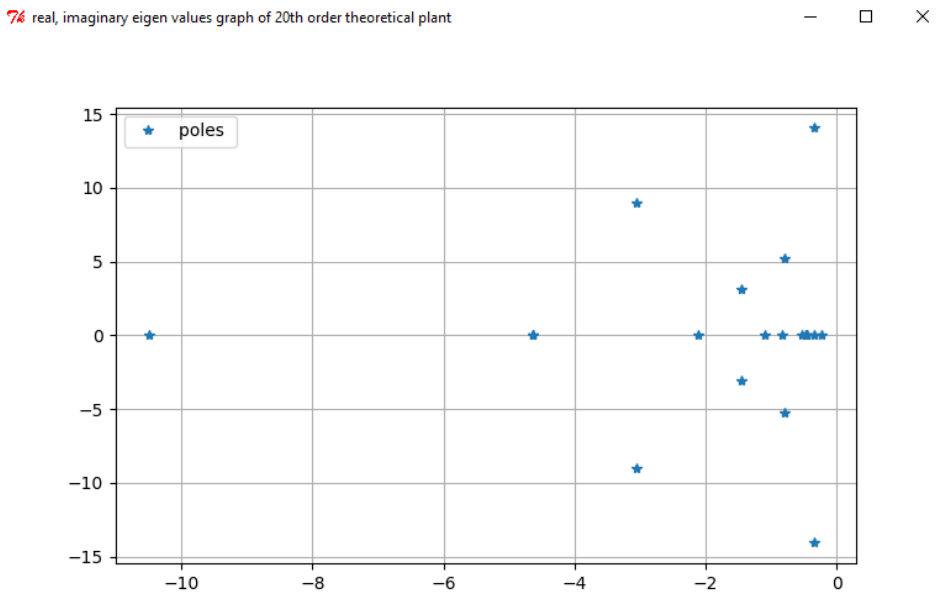


Figure 35: Poles, zeros graph of 20th order theoretical plant.

Eigen values of this plant is way smaller than rig model however system is stable.

6.1 20th Order Theoretical Plant Control

In this section, results of control and identification process are shared. Identification of system is achieved via GD algorithm and results is illustrated in Figure 36.

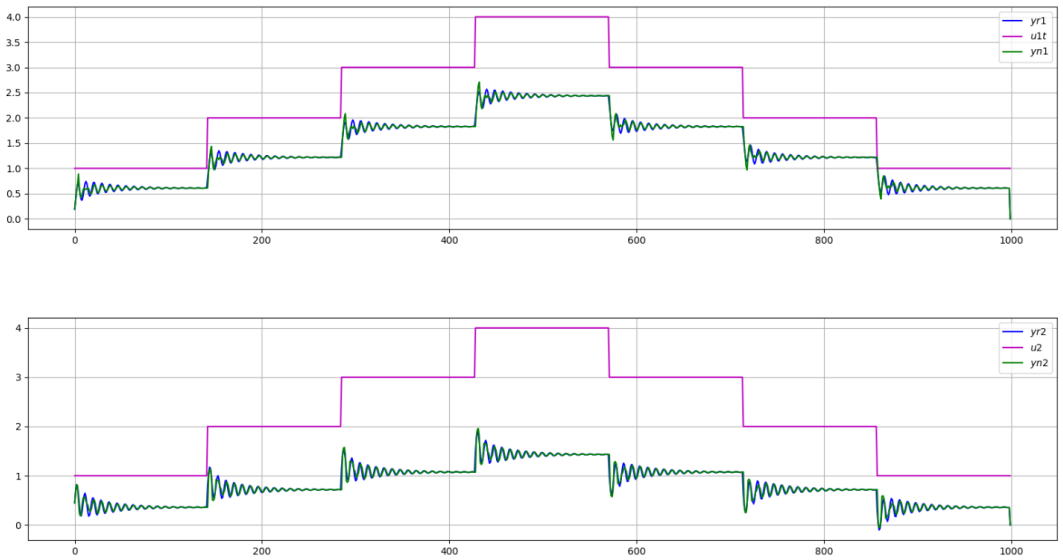


Figure 36: 20th order plant identification with LNU using GD as response to step input signal.

Initial set values and error values are given in Table 11.

GD	SSE1	SSE2	μ_1	Epochs	μ_2
LNU	1,32	1,28	0.5	200	0.7

Table 14: Details of simulated system of 20th order with variable step signal.

Identification of system is pretty good, in next part neuro controller is applied to model, as reference model second order plant is used as in previous chapters because of its smooth transition.

Neuro controller implementation is made under different conditions to compare effects of variables in implemented algorithm. Variables in identification process are not changed. First initial conditions and error results are given in Table 13. According to first initial conditions, neuro controller performance is given in Figure 37.

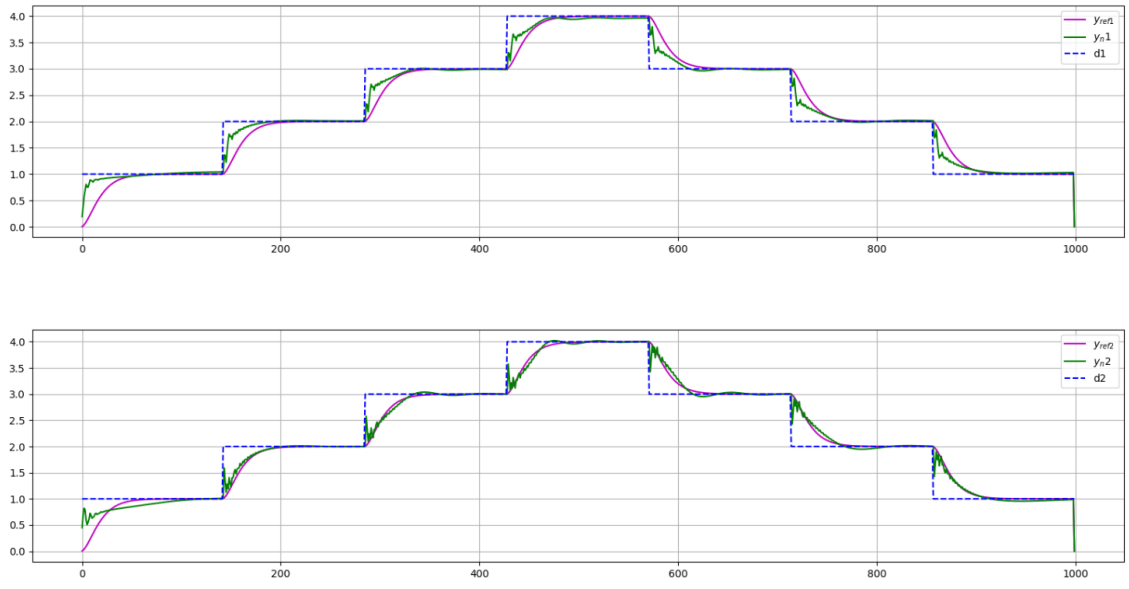


Figure 37: Roller-Rig tuned controller for first trial.(Original plant can be seen in Figure 34.)

Magnified version of Figure 37 is given in Figure 38.

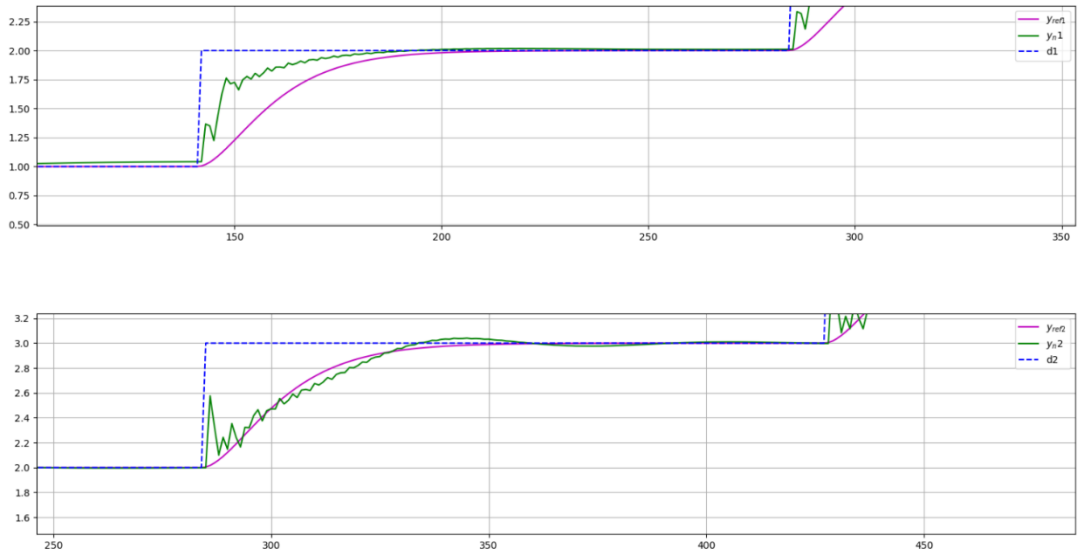


Figure 38: Magnified view of Figure 37.

**Original plant can be seen in Figure 34.*

GD	**SSE1**	**SSE2**	μ_{1c}	μ_{2c}	**Epochs**	**r01**	**r02**
LNU	15,12	10,64	0.4	0.3	200	0.001	0.01

Table 15: First initial conditions and error results.

Results are under second initial conditions given in Figure 39 and Table 14.

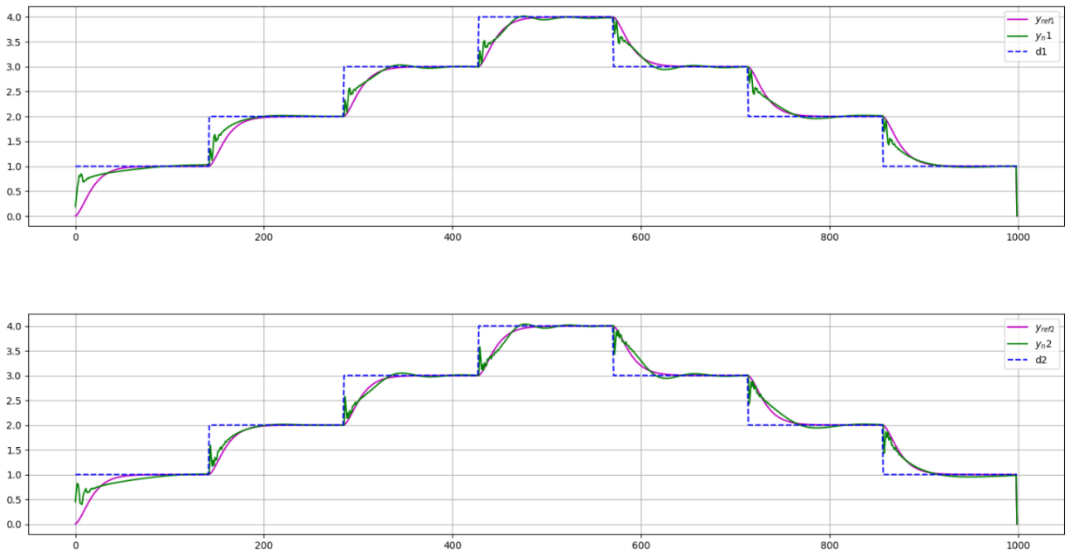


Figure 39: Roller-Rig tuned controller for second trial.

**Original plant can be seen in Figure 34.*

Magnified version of Figure 39 is given in Figure 40.

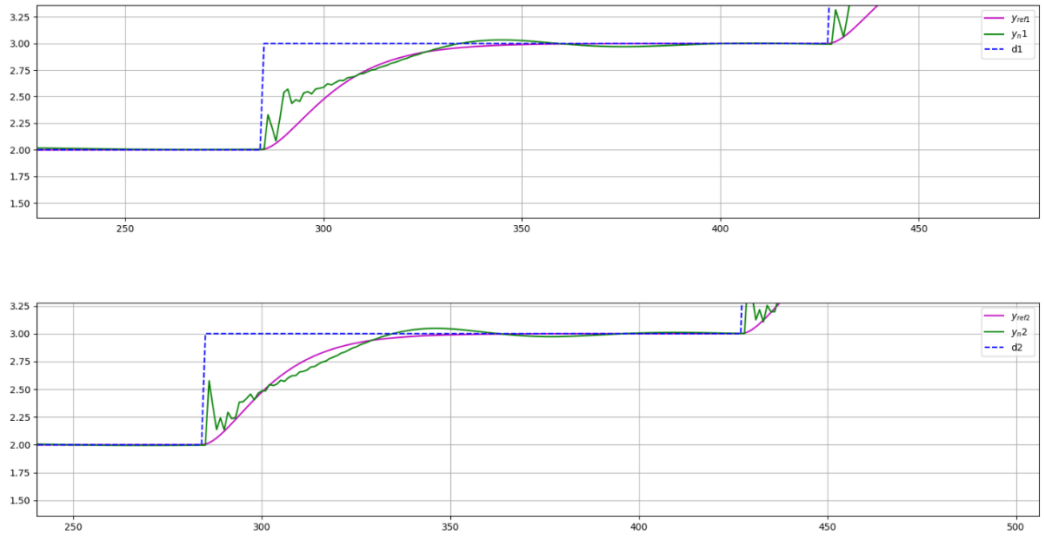


Figure 40: Magnified view of Figure 37.

**Original plant can be seen in Figure 34.*

GD	**SSE1**	**SSE2**	μ_{1c}	μ_{2c}	**Epochs**	**r01**	**r02**
LNU	12,51	7,82	0.2	0.15	200	0.006	0.04

Table 16: Second initial conditions and error results.

Final trial is shown in Figure 41 and Table 15.

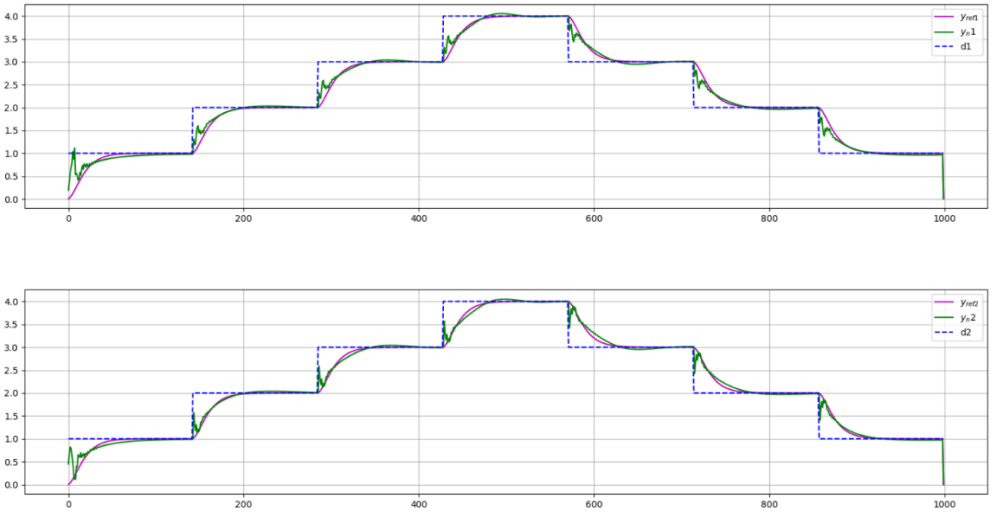


Figure 41: Roller-Rig tuned controller for second trial.(Original plant can be seen in Figure 34.)

Magnified version of Figure 41 is given in Figure 42.

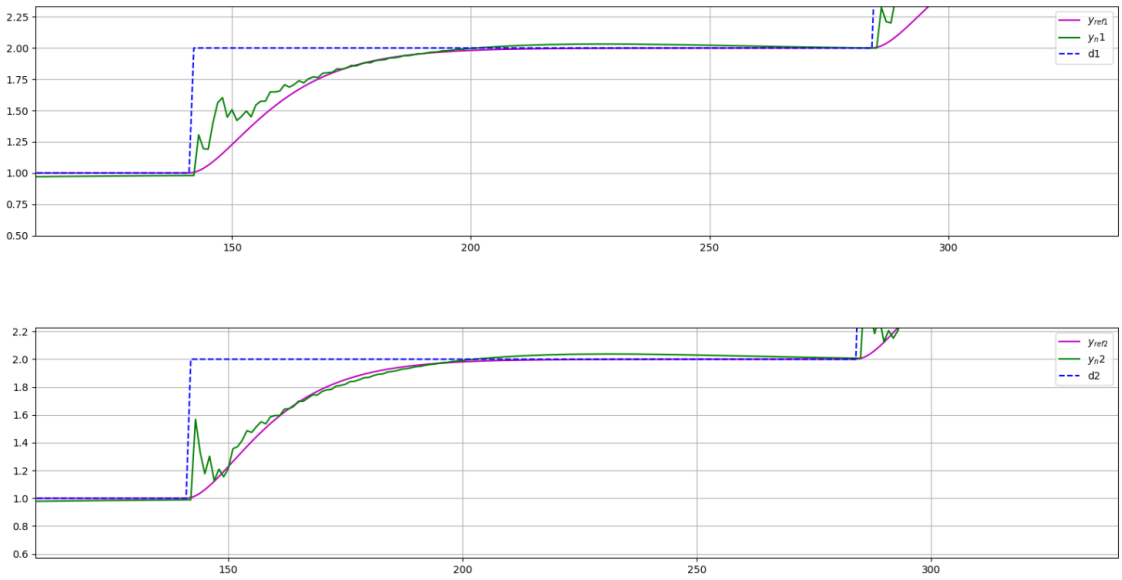


Figure 42: Magnified view of Figure 39.(Original plant can be seen in Figure 34.)

GD	**SSE1**	**SSE2**	**μ_{1c}**	**μ_{2c}**	**Epochs**	**r01**	**r02**
LNU	6,51	5,82	0.23	0.19	200	0.004	0.09

Table 17: Final initial conditions and error results.

Different initial conditions and variable values effect performance of neuro controller, so for better performance, learning rate values and system gain values could be tuned precisely. After 3 different sets of initial conditions and different variable values, results are shared. In final trial, SSE values are less than other trial and it can be seen plant is controlled nicely except overshooting in first phase of transition. It is also possible to gain better results with different reference model.

7. Conclusion

In this thesis, multiple-input multiple-output system (MIMO) is chosen for implementation to test performance of neural algorithms. Thus, two linear MIMO system are proposed as theoretical models for testing; stable linear 2-2 linear MIMO system of 2^{nd} order of dynamics and stable oscillating 2-2 linear MIMO system of 4^{th} order of dynamics. These models are built and simulated in MATLAB & Simulink and Python 2.7 environment. Further recently published works on adaptive algorithms for identification and control of linear MIMO system with supervised learning algorithms are reviewed, and sample-by-sample adaptation rules are mainly focused such as Gradient Descent (GD), and Recursive Least Square (RLS) algorithms. These adaptation rules are derived and carried out for identification and control experimental analysis on linear MIMO system via Python 2.7. For this thesis work, MRAC approach is used. All key objectives are completed within scope of the thesis, there are main accomplished points should be considered. One of them is system simulation in Python 2.7, this allow me to tune controller same time with system run. Another one is neuro controller implementation on linear MIMO system, because of linear MIMO system complexity, it required complex neural units formation. Next point to mention is algorithms, GD algorithm performed very well however RLS algorithm did not perform as good as GD especially for neuro controller. The last point, roller rig model could not be observed properly due to maintenance process, its mathematical model was not stable therefore it is tried to be stabilized. However system was not stabilized, and main goal of the thesis was not try to stabilize such system with neural algorithms. So 20^{th} order theoretical model is created to simulate similar system and observe its performance, implementation of algorithm on such complex structure (20^{th} order system) gives idea about possible implementation of neuro controller. This is thesis could be first step for those who will make a further search in this field.

Reference

[1] “IEEE Xplore Abstract - Adaptive control: The model reference approach.” [Online].Available: http://ieeexplore.ieee.org.ezproxy.techlib.cz/xpls/abs_all.jsp?arnumber=6313284.

[2] Henry Howards, "Oscillation Criteria for Fourth-Order Linear Differential Equations."[Online]. Available: https://www.ams.org/journals/tran/1960-096-02/S0002-9947-1960-0117379-X/S0002-9947-1960-0117379-X.pdf. [Accessed: 12-Apr-2018].

[3] Peter Mark Beneš and, Ivo Bukovský, " Software Application for Adaptive Identification and Controller Tuning".[Online] http://stc.fs.cvut.cz/pdf13/2611.pdf.[Accessed: 10-Jan-2018].

[4] LifuWu, Xiaojun Qiu, Ian S.Burnett, Yecai Guo. “A recursive least square algorithm for active control of mixed noise.”. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022460X14008773. [Accessed: 1-Jun-2018].

[5] "Adaptive MPC Design - MATLAB & Simulink." [Online]. Available: https://www.mathworks.com/help/mpc/adaptive-mpc-design.html.[Accessed: 21-Mar-2018].

[6] Vasiliki Koropouli, Azwirman Gusrialdi and Dongheui Lee, "ESC-MRAC of Linear MIMO systems for Constrained Robotic Motion Tasks in Deformable Environments". 2014 European Control Conference (ECC) June 24-27, 2014. Strasbourg, France.

[7] "MIMO State-Space Models - MATLAB & Simulink."[Online]. Available: https://www.mathworks.com/help/control/ug/MIMO-state-space-models.html. [Accessed: 8-Apr-2018].

[8] "Generate random continuous test model - MATLAB & Simulink."[Online]. Available: https://www.mathworks.com/help/control/ref/rss.html. [Accessed: 8-Apr-2018].

[9] "scipy.integrate.odeint." [Online]. Available: https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.integrate.odeint.html [Accessed: 6-May-2018].

[10] Ivo Bukovsky, Noriyasu Homma, "An Approach to Stable Gradient Descent Adaptation of Higher-Order Neural Units". [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1606/1606.07149.pdf. [Accessed: 25-Mar-2018].

[11] "Real rime recurrent learning". [Online]. Available: http://www.dlsi.ua.es/~mlf/nnafmc/pbook/node29.html.

[12] Ilia Ali Ogly, "Solution for Adaptive Control of Coupled Motor System with Flexible Joint".
Master Thesis, Academic Year 2015/2016, Czech Technical University in Prague, Prague.

[13] Murat Üney, "Optimal and Adaptive Filtering".Institute for Digital Communications (IDCOM), University of Edinburgh, 20-07-2015.

[14] "Recursive Least Squares (RLS)." [Online]. Available: http://matousc89.github.io/padasip/sources/filters/rls.html [Accessed: 23-May-2018].

[15] Vasiliki Koropouli, Azwirman Gusrialdi and Dongheui Lee, "ESC-MRAC of Linear MIMO systems for Constrained Robotic Motion Tasks in Deformable Environments".
2014 European Control Conference (ECC) June 24-27, 2014. Strasbourg, France.

[16] Scott C. Bonnen and, L. Wozniak, P.E., "Adaptive Speed Control of Hydrogenerators by Recursive Least Square Idenficiation Algorithm".
IEEE Transactions on Energy Conversion, Vol. 10, No. 1, March 1995.

[17] J. Kalivoda and P. Bauer, "Active Stabilization of a 2-axle Railway Bogie – Roller Rig Tests".
Department of Automotive, Combustion Engine and Railway Engineering, Faculty of Mechanical Engineering, Czech Technical University in Prague, Czech Republic

Appendix

Code for 2nd Order Theoratical Plant Neuro Controller Implementation

```
from numpy import*
from matplotlib.pyplot import*
from scipy.integrate import odeint
from numpy.random import randn

#Input signals
t=arange(0,200,.1)
N = len(t)
u1=sign(sin(2*pi*t/40))
u2=sign(sin(2*pi*t/50))

u1[0:int(N/7)]=1;                              u1[int(N/7):int(2*N/7)]=2;
u1[int(2*N/7):int(3*N/7)]=3
u1[int(3*N/7):int(4*N/7)]=4;u1[int(4*N/7):int(5*N/7)]=3
u1[int(5*N/7):int(6*N/7)]=2; u1[int(6*N/7):int(7*N/7)]=1

u2[0:int(N/7)]=1;                              u2[int(N/7):int(2*N/7)]=2;
u2[int(2*N/7):int(3*N/7)]=3
u2[int(3*N/7):int(4*N/7)]=4; u2[int(4*N/7):int(5*N/7)]=3
u2[int(5*N/7):int(6*N/7)]=2; u2[int(6*N/7):int(7*N/7)]=1
#Theoretical Second Order Plant Simulation
y1 = zeros(N); y2 = zeros(N); x0 = [0,0]

a1 = -1.0; a2 = 0.5; a3 = 3.0; a4 = 1.5
b1 = -0.5; b2 = -2.0; b3 = 0; b4 = -1.0

def dfdt(x,t,u1,u2,a1,a2,a3,a4,b1,b2,b3,b4):
    dx1dt = a1*x[0] + a2*x[1] + a3*u1 + a4*u2
    dx2dt = b1*x[0] + b2*x[1] + b3*u1 + b4*u2
    return (dx1dt, dx2dt)

for i in range(0,N-1):
    tt = [t[i],t[i+1]]
    x = odeint(dfdt,x0,t,(u1[i],u2[i],a1,a2,a3,a4,b1,b2,b3,b4))
    y1[i] = .4*x[0,0] - .1*x[1,1]
    y2[i] = -.98*x[1,1]
    x0 = x[1,:]
#Reference model
yref1 =zeros(N);yref2 = zeros(N); k= -2
yref1 = zeros(N); x0r1 = [0,0]
def ref1(x,t,u1,k):
    dxdt1 = k*x[0] + -1*x[1] + u1
    dxdt2 = 1*x[0]
    return(dxdt1,dxdt2)
for i in range(0,N-1):
    tt = [t[i],t[i+1]]
    x = odeint(ref1,x0r1,t,(u1[i],k))
    yref1[i] = x[1,1]*1
    x0r1 = x[1,:]
m= -2
yref2 = zeros(N); x0r2 = [0,0]
def ref2(x,t,u2,m):
    dxdt1 = m*x[0] + -1*x[1] + u2
```

```
    dxdt2 = 1*x[0]
    return(dxdt1,dxdt2)
for i in range(0,N-1):
    tt = [t[i],t[i+1]]
    x = odeint(ref2,x0r2,t,(u2[i],m))
    yref2[i] = x[1,1]*1
    x0r2 = x[1,:]
#) Parameter Definition for Adaptive Identification
#first neuron
nu1 = 4 ;nu12 = 3; ny1 = 4; ny12 = 3; nx1 = 1 + nu1+nu12 +ny1 +ny12
nw1 = nx1
#second neuron
nu2 = 4; nu21 = 3; ny2 = 4; ny21 = 3; nx2 = 1 + nu1+nu21 +ny1 + ny21
nw2 = nx2
#common variables
nx = nx1 = nx2; nw = nw1 = nw2
##common values
mu = .5; Epochs =2000
##first neuron
w1 = randn(nw)/nw; ynn1 = y1.copy(); e1 = zeros(N); x1 = zeros(nx)
SSE1 = zeros(Epochs); dydw1 = zeros((N,nw))

##second neuron
w2 = randn(nw)/nw; ynn2 = y2.copy();e2 = zeros(N);x2 = zeros(nx)
SSE2 = zeros(Epochs)  ; dydw2 = zeros((N,nw))

#6) Adaptive Identification
for epoch in range(0,Epochs):
    for k in range(max(ny1,nu1,nu12,ny12),N-1):
        ##first neuron x vector
        x1[0] = 1
        x1[1:(ny1+1)] =ynn1[range(k-1,k-(ny1+1),-1)]
        x1[(ny1+1):(ny1+ny12+1)] =ynn2[range(k-1,k-(ny12+1),-1)]
        x1[(ny1+ny12+1):(ny1+ny12+nu1+1)]=u1[range(k-1,k-(nu1+1),-1)]
        x1[(ny1+ny12+nu1+1):]=[range(k-1,k-(nu12+1),-1)]

        ##second neuron x vector
        x2[0] = 1
        x2[1:(ny2+1)]= ynn2[range(k-1,k-(ny2+1),-1)]
        x2[(ny2+1):(ny2+ny21+1)]= ynn1[range(k-1,k-(ny21+1),-1)]
        x2[(ny2+ny21+1):(ny2+ny21+nu2+1)]=u2[range(k-1,k-(nu2+1),-1)]
        x2[(ny2+ny21+nu2+1):]= u1[range(k-1,k-(nu21+1),-1)]
        #update rule
        ynn1[k] = dot(w1,x1)
        e1[k] = y1[k] - ynn1[k]
        dydw1[k,:] = x1

        #gd
        dw1= mu/(1+dot(x1,x1))*e1[k]*dydw1[k,:]
        w1 = w1+dw1
        #update rule
        ynn2[k] = dot(w2,x2)
        e2[k] = y2[k] - ynn2[k]
        dydw2[k,:] = x2

       #gd
        dw2= mu/(1+dot(x2,x2))*e2[k]*dydw2[k,:]
        w2 = w2+dw2


#Desired Values
```

```
d1 = u1.copy(); d2 = u2.copy()
#Set points
setpoint1 = d1.copy(); setpoint2 = d2.copy()
#Parameter Definitions for controller
muv = .3; r01 = 0.009; r02=0.015; v1 = zeros(nw); v2=zeros(nw)
q1 = zeros(N) ; q2 = zeros(N); eref1=zeros(N) ; eref2=zeros(N)
xi1 = ones(nw) ; xi2 = ones(nw); dxdv1 = zeros((nw,nw)) ; dxdv2 =
zeros((nw,nw))
dydv1 = zeros(nw) ; dydv2 = zeros(nw); nd = nu1 + nu12 ;ny= ny1+ny12
yn1 = y1.copy(); yn2 = y2.copy()
for epoch in range(0,Epochs):
    for k in range(max(ny1,nu1,nu12,ny12),N-1):
        xi1[0] = 1
        xi1[1:(ny1+1)] = yn1[range(k-1,k-(ny1+1),-1)]
        xi1[(ny1+1):(ny1+ny12+1)] = yn2[range(k-1,k-(ny12+1),-1)]
        xi1[(ny1+ny12+1):(ny1+ny12+nu1+1)]=d1[range(k-1,k-(nu1+1),-
1)]
        xi1[(ny1+ny12+nu1+1):] = d2[range(k-1,k-(nu12+1),-1)]
        q1[k] = dot(v1,xi1)
        setpoint1[k]= d1[k] - r01*q1[k]
        xi2[0] = 1
        xi2[1:(ny2+1)] =                    yn2[range(k-1,k-(ny2+1),-
1)]
        xi2[(ny2+1):(ny2+ny21+1)] =         yn1[range(k-1,k-(ny21+1),-
1)]
        xi2[(ny2+ny21+1):(ny2+ny21+nu2+1)] = d2[range(k-1,k-(nu2+1),-
1)]
        xi2[(ny2+ny21+nu2+1):] =d1[range(k-1,k-(nu21+1),-1)]
        q2[k] = dot(v2,xi2)
        setpoint2[k]= d2[k] - r02*q2[k]
        #first neuron as model
        x1[0] = 1
        x1[1:(ny1+1)] =yn1[range(k-1,k-(ny1+1),-1)]
        x1[(ny1+1):(ny1+ny12+1)] =yn2[range(k-1,k-(ny12+1),-1)]
        x1[(ny1+ny12+1):(ny1+ny12+nu1+1)]=setpoint1[range(k-1,k-
(nu1+1),-1)]
        x1[(ny1+ny12+nu1+1):] =setpoint2[range(k-1,k-(nu12+1),-1)]
       #second neuron as model
        x2[0] = 1
        x2[1:(ny2+1)] =yn2[range(k-1,k-(ny2+1),-1)]
        x2[(ny2+1):(ny2+ny21+1)] =yn1[range(k-1,k-(ny21+1),-1)]
        x2[(ny2+ny21+1):(ny2+ny21+nu2+1)=setpoint2[range(k-1,k-
(nu2+1),-1)]
        x2[(ny2+ny21+nu2+1):] =setpoint1[range(k-1,k-(nu21+1),-1)]

        yn1[k] = dot(w1,x1)
        eref1[k] = yref1[k] - yn1[k]
        dxdv1[1:ny,:]=dxdv1[2:1+ny,:];dxdv1[ny,:]=dydv1
        dxdv1[1+ny:-1,:]=dxdv1[2+ny:,:];dxdv1[-1,:]=-r01*xi1
        dydv1 = dot(w1,dxdv1)
        #dxdv[-1,:] = -xi

        dv1 = muv*eref1[k]*dydv1 #dot(w1,dxdv1)
        v1 = v1 +dv1

        yn2[k] = dot(w2,x2)
        eref2[k] = yref2[k] - yn2[k]
        dxdv2[1:ny,:]=dxdv2[2:1+ny,:];dxdv2[ny,:]=dydv2
        dxdv2[1+ny:-1,:]=dxdv2[2+ny:,:];dxdv2[-1,:]=-r02*xi2
        dydv2 = dot(w2,dxdv2)
        #dxdv[-1,:] = -xi
```

```
        dv2 = muv*eref2[k]*dydv2 #dot(w2,dxdv2)
        v2 = v2 +dv2
    SSE2[epoch] = sum(eref2*eref2)
    SSE1[epoch] = sum(eref1*eref1)
    print(SSE1[epoch]),(SSE2[epoch])

figure(figsize=(14,9))
subplots_adjust(hspace=.4)
subplot(211)
title("Controller sample-by-sample adaptation for 2nd order stable
linear theoratical plant")
plot(yref1,'m',label="$y_{ref1}$")
plot(d1,"b--", label="d1")
plot(yn1,'g',label="$y_n1$")
grid();legend()
subplot(212)
plot(yref2,'m',label="$y_{ref2}$")
plot(d2,"b--", label="d2")
plot(yn2,'g',label="$y_n2$")
grid();legend()
show()
```

Code for 4th Order Theoratical Plant Neuro Controller Implementation

```
from numpy import*
from matplotlib.pyplot import*
from scipy.integrate import odeint
from numpy.random import randn

#Input signals
t=arange(0,100,.1)
N = len(t)
u1=sign(sin(2*pi*t/40))
u2=sign(sin(2*pi*t/50))

u1[0:int(N/7)]=1;                              u1[int(N/7):int(2*N/7)]=2;
u1[int(2*N/7):int(3*N/7)]=3
u1[int(3*N/7):int(4*N/7)]=4;u1[int(4*N/7):int(5*N/7)]=3
u1[int(5*N/7):int(6*N/7)]=2; u1[int(6*N/7):int(7*N/7)]=1

u2[0:int(N/7)]=1;                              u2[int(N/7):int(2*N/7)]=2;
u2[int(2*N/7):int(3*N/7)]=3
u2[int(3*N/7):int(4*N/7)]=4; u2[int(4*N/7):int(5*N/7)]=3
u2[int(5*N/7):int(6*N/7)]=2; u2[int(6*N/7):int(7*N/7)]=1
# 4th order system
y1 = zeros(N); y2 = zeros(N); x0 = [0,0,0,0]
a1=-0.8  ;a2=4      ;a3=-5     ;a4=-2
b1=-4    ;b2=-1     ;b3=-3     ;b4=-1
c1=5     ;c2=1      ;c3=-2     ;c4= 4
d1=3     ;d2= 1.5   ;d3=0.5    ;d4=-5
def dfdt(x,t,u1,u2,a1,a2,a3,a4,b1,b2,b3,b4,c1,c2,c3,c4,d1,d2,d3,d4):
    dx1dt = a1*x[0] + a2*x[1] + a3*x[2] + a4*x[3] +     1*u1 +   0.1*u2
    dx2dt = b1*x[0] + b2*x[1] + b3*x[2] + b4*x[3] + -0.1*u1  +  0.5*u2
    dx3dt = c1*x[0] + c2*x[1] + c3*x[2] + c4*x[3] +    0.5*u1
    dx4dt = d1*x[0] + d2*x[1] + d3*x[2] + d4*x[3] +    -1*u1
    return (dx1dt, dx2dt,dx3dt,dx4dt)
for i in range(0,N-1):
    tt = [t[i],t[i+1]]
    x                                                              =
odeint(dfdt,x0,t,(u1[i],u2[i],a1,a2,a3,a4,b1,b2,b3,b4,c1,c2,c3,c4,d1,
d2,d3,d4))
    y1[i] =( 0.8*x[0,0]  - 1.5*x[1,1]    +1.2*x[2,2] - 1.5*x[3,3])*0.8
```

```
    y2[i] =  x[2,2] *3.4
    x0 = x[1,:]
yref1 = zeros(N); yref2= zeros(N)

# Reference Model
k= -2
yref1 = zeros(N); x0r1 = [0,0]
def ref1(x,t,u1,k):
    dxdt1 = k*x[0] + -1*x[1] + u1
    dxdt2 = 1*x[0]
    return(dxdt1,dxdt2)

for i in range(0,N-1):
    tt = [t[i],t[i+1]]
    x = odeint(ref1,x0r1,t,(u1[i],k))
    yref1[i] = x[1,1]*1
    x0r1 = x[1,:]

m= -2
yref2 = zeros(N); x0r2 = [0,0]
def ref2(x,t,u2,m):
    dxdt1 = m*x[0] + -1*x[1] + u2
    dxdt2 = 1*x[0]
    return(dxdt1,dxdt2)

for i in range(0,N-1):
    tt = [t[i],t[i+1]]
    x = odeint(ref2,x0r2,t,(u2[i],m))
    yref2[i] = x[1,1]*1
    x0r2 = x[1,:]

#) Defining parameters for identification
##first neuron
nu1 = 4 ;nu12 = 3; ny1 = 4; ny12 = 3; nx1 = 1 + nu1+nu12 +ny1 +ny12
nw1 = nx1
##second neuraon
nu2 = 4; nu21 = 3; ny2 = 4; ny21 = 3; nx2 = 1 + nu1+nu21 +ny1 + ny21
nw2 = nx2
##common variables
nx = nx1 = nx2; nw = nw1 = nw2
##common values
mu = .5; Epochs =2000
##first neuron
w1 = randn(nw)/nw; ynn1 = y1.copy(); e1 = zeros(N); x1 = zeros(nx)
SSE1 = zeros(Epochs); dydw1 = zeros((N,nw))
##second neuron
w2 = randn(nw)/nw ;ynn2 = y2.copy(); e2 = zeros(N); x2 = zeros(nx)
SSE2 = zeros(Epochs); dydw2 = zeros((N,nw))
#6) Adaptive Identification
for epoch in range(0,Epochs):
    for k in range(max(ny1,nu1,nu12,ny12),N-1):
        ##first neuron x vector
        x1[0] = 1
        x1[1:(ny1+1)] =                    ynn1[range(k-1,k-(ny1+1),-
1)]
        x1[(ny1+1):(ny1+ny12+1)] =         ynn2[range(k-1,k-(ny12+1),-
1)]
        x1[(ny1+ny12+1):(ny1+ny12+nu1+1)]  =  u1[range(k-1,k-(nu1+1),-
1)]
        x1[(ny1+ny12+nu1+1):] =              u2[range(k-1,k-(nu12+1),-
1)]
```

```
        ##second neuron x vector
        x2[0] = 1
        x2[1:(ny2+1)] =                     ynn2[range(k-1,k-(ny2+1),-
1)]
        x2[(ny2+1):(ny2+ny21+1)] =         ynn1[range(k-1,k-(ny21+1),-
1)]
        x2[(ny2+ny21+1):(ny2+ny21+nu2+1)]  =  u2[range(k-1,k-(nu2+1),-
1)]
        x2[(ny2+ny21+nu2+1):] =              u1[range(k-1,k-(nu21+1),-
1)]
        #update rule
        ynn1[k] = dot(w1,x1)
        e1[k] = y1[k] - ynn1[k]
        dydw1[k,:] = x1
        #gd
        dw1= mu/(1+dot(x1,x1))*e1[k]*dydw1[k,:]
        w1 = w1+dw1
         #update rule
        ynn2[k] = dot(w2,x2)
        e2[k] = y2[k] - ynn2[k]
        dydw2[k,:] = x2
        #gd
        dw2= mu/(1+dot(x2,x2))*e2[k]*dydw2[k,:]
        w2 = w2+dw2

    #
#Desired Values
d1 = u1.copy(); d2 = u2.copy()
#Set points
setpoint1 = d1.copy();setpoint2 = d2.copy()
#Controller Parameters ;muv = .5 ; r01 = .013 ; r02 = .016
v1 = zeros(nw); v2=zeros(nw)
q1 = zeros(N) ; q2 = zeros(N); eref1=zeros(N) ; eref2=zeros(N)
xi1 = ones(nw) ; xi2 = ones(nw); dxdv1 = zeros((nw,nw)) ; dxdv2 =
zeros((nw,nw))
dydv1 = zeros(nw) ; dydv2 = zeros(nw); nd = nu1 + nu12 ;ny= ny1+ny12
yn1 = y1.copy(); yn2 = y2.copy()
for epoch in range(0,Epochs):
    for k in range(max(ny1,nu1,nu12,ny12),N-1):
        xi1[0] = 1
        xi1[1:(ny1+1)] =                     yn1[range(k-1,k-(ny1+1),-
1)]
        xi1[(ny1+1):(ny1+ny12+1)] =         yn2[range(k-1,k-(ny12+1),-
1)]
        xi1[(ny1+ny12+1):(ny1+ny12+nu1+1)] = d1[range(k-1,k-(nu1+1),-
1)]
        xi1[(ny1+ny12+nu1+1):] =             d2[range(k-1,k-(nu12+1),-
1)]

        q1[k] = dot(v1,xi1)
        setpoint1[k]= d1[k] - r01*q1[k]

        xi2[0] = 1
        xi2[1:(ny2+1)] =                     yn2[range(k-1,k-(ny2+1),-
1)]
        xi2[(ny2+1):(ny2+ny21+1)] =         yn1[range(k-1,k-(ny21+1),-
1)]
        xi2[(ny2+ny21+1):(ny2+ny21+nu2+1)] = d2[range(k-1,k-(nu2+1),-
1)]
```

```
        xi2[(ny2+ny21+nu2+1):] =            d1[range(k-1,k-(nu21+1),-
1)]

        q2[k] = dot(v2,xi2)
        setpoint2[k]= d2[k] - r02*q2[k]

        #first neuron
        x1[0] = 1
        x1[1:(ny1+1)] =                     yn1[range(k-1,k-(ny1+1),-
1)]
        x1[(ny1+1):(ny1+ny12+1)] =         yn2[range(k-1,k-(ny12+1),-
1)]
        x1[(ny1+ny12+1):(ny1+ny12+nu1+1)]   =   setpoint1[range(k-1,k-
(nu1+1),-1)]
        x1[(ny1+ny12+nu1+1):] =                 setpoint2[range(k-1,k-
(nu12+1),-1)]

        #second neuron
        x2[0] = 1
        x2[1:(ny2+1)] =                     yn2[range(k-1,k-(ny2+1),-
1)]
        x2[(ny2+1):(ny2+ny21+1)] =         yn1[range(k-1,k-(ny21+1),-
1)]
        x2[(ny2+ny21+1):(ny2+ny21+nu2+1)]   =   setpoint2[range(k-1,k-
(nu2+1),-1)]
        x2[(ny2+ny21+nu2+1):] =                 setpoint1[range(k-1,k-
(nu21+1),-1)]

        yn1[k] = dot(w1,x1)
        eref1[k] = yref1[k] - yn1[k]
        dxdv1[1:ny,:]=dxdv1[2:1+ny,:];dxdv1[ny,:]=dydv1
        dxdv1[1+ny:-1,:]=dxdv1[2+ny:,:];dxdv1[-1,:]=-r01*xi1
        dydv1 = dot(w1,dxdv1)
        #dxdv[-1,:] = -xi

        dv1 = muv*eref1[k]*dydv1 #dot(w1,dxdv1)
        v1 = v1 +dv1

        yn2[k] = dot(w2,x2)
        eref2[k] = yref2[k] - yn2[k]
        dxdv2[1:ny,:]=dxdv2[2:1+ny,:];dxdv2[ny,:]=dydv2
        dxdv2[1+ny:-1,:]=dxdv2[2+ny:,:];dxdv2[-1,:]=-r02*xi2
        dydv2 = dot(w2,dxdv2)
        #dxdv[-1,:] = -xi
        dv2 = muv*eref2[k]*dydv2 #dot(w2,dxdv2)
        v2 = v2 +dv2
    SSE2[epoch] = sum(eref2*eref2)
    SSE1[epoch] = sum(eref1*eref1)
    print(SSE1[epoch]),(SSE2[epoch])

figure(figsize=(14,9))
subplots_adjust(hspace=.4)
subplot(211)
title("Controller sample-by-sample adaptation 4th order oscillating
linear plant")
#plot(y1,'b',label="$y_r$")
plot(yref1,'m',label="$y_{ref1}$")
plot(yn1,'g',label="$y_n1$")
plot(d1, "b--", label="d1")
grid();legend()
subplot(212)
```

```
#plot(y2,'b',label="$y_r$")
plot(yref2,'m',label="$y_{ref2}$")
plot(yn2,'g',label="$y_n2$")
plot(d2, "b--", label="d2")
grid();legend()
show()
```

Code for 20th Order Plant Neuro Controller Implementation

```
from numpy import*
from matplotlib.pyplot import*
from scipy.integrate import odeint
from numpy.random import randn

#Input signals
t=arange(0,100,.1)
N = len(t)
u1=sign(sin(2*pi*t/40))
u2=sign(sin(2*pi*t/50))

u1[0:int(N/7)]=1;                              u1[int(N/7):int(2*N/7)]=2;
u1[int(2*N/7):int(3*N/7)]=3
u1[int(3*N/7):int(4*N/7)]=4;u1[int(4*N/7):int(5*N/7)]=3
u1[int(5*N/7):int(6*N/7)]=2; u1[int(6*N/7):int(7*N/7)]=1

u2[0:int(N/7)]=1;                              u2[int(N/7):int(2*N/7)]=2;
u2[int(2*N/7):int(3*N/7)]=3
u2[int(3*N/7):int(4*N/7)]=4; u2[int(4*N/7):int(5*N/7)]=3
u2[int(5*N/7):int(6*N/7)]=2; u2[int(6*N/7):int(7*N/7)]=1

#System Definition
a = -5
x0 = zeros(20)
ef dfdt(x,t,u1,u2,a):
    dx1dt =a*x[0] +1.985*x[1] +0.4971*x[2]      -0.2271*x[3]      -
1.784*x[4]  +0.4091*x[5]      +1.475*x[6] +0.7547*x[7]      -
1.416*x[8]  +0.1734*x[9]      +2.591*x[10]      +2.573*x[11]
      +1.047*x[12]      +1.125*x[13]      -2.061*x[14]      -
0.3782*x[15]      -0.1733*x[16]     -0.04207*x[17]    -1.476*x[18]
      -0.3628*x[19]   +0.1202*u1    +0.09231*u2
    dx2dt =-1.477*x[0]  -2.189*x[1] -2.167*x[2] +0.3972*x[3]
      +1.949*x[4] +0.647*x[5] -2.259*x[6] +0.3126*x[7]
      +1.619*x[8] -2.4*x[9]   -0.9923*x[10]     +0.1743*x[11]
      +2.16*x[12] -0.7117*x[13]     -0.5006*x[14]     +0.7109*x[15]
      -0.3583*x[16]     +1.901*x[17]      -0.4519*x[18]     -
0.5639*x[19]   +0*u1    +1.73*u2
    dx3dt =-1.08*x[0]   +1.289*x[1] -1.604*x[2] +4.064*x[3] -
2.64*x[4]   +2.154*x[5] -1.563*x[6] +3.149*x[7] -3.44*x[8]  -
1.697*x[9]  -1.075*x[10]      +2.627*x[11]      +2.22*x[12] -
2.204*x[13] -1.557*x[14]      -1.608*x[15]      +0.2274*x[16]     -
0.4241*x[17]      -1.942*x[18]      +1.295*x[19]    +0*u1   +0*u2
    dx4dt= 0.6845*x[0]  -0.09759*x[1]     -3.177*x[2] -1.92*x[3]  -
1.175*x[4]  +3.755*x[5] +0.1581*x[6]      -0.6664*x[7]
      +0.09002*x[8]     -1.107*x[9] -1.084*x[10]      -0.717*x[11]
      +0.2265*x[12]     -0.7542*x[13]     +0.5366*x[14]     -
1.355*x[15] -0.1324*x[16]     -2.146*x[17]      -0.6385*x[18]     -
0.6902*x[19]   -0.987*u1      +0*u2
    dx5dt =1.702*x[0]   -1.304*x[1] +2.491*x[2] +0.9819*x[3]      -
1.366*x[4]  +1.515*x[5] +0.0522*x[6]      +1.627*x[7] -2.094*x[8]
      +0.6409*x[9]      +1.647*x[10]      +1.567*x[11]
      +0.4101*x[12]     -0.1509*x[13]     -1.409*x[14]      -
```

```
0.1952*x[15]      +1.958*x[16]      -2.058*x[17]      -1.683*x[18]
      +2.367*x[19]    +0.7596*u1    -1.75*u2
    dx6dt =-0.8476*x[0] -0.0988*x[1]      -3.141*x[2] -2.606*x[3] -
2.392*x[4]  -1.295*x[5] +2.737*x[6] -0.9228*x[7]      +1.403*x[8]
      +1.182*x[9] +2.259*x[10]      -3.268*x[11]      -0.9255*x[12]
      +2.322*x[13]      +1.864*x[14]      -0.2566*x[15]     -
1.642*x[16] +0.1723*x[17]     +2.443*x[18]      -1.219*x[19]    +0*u1
      +0.9105*u2
    dx7dt =-1.276*x[0]  +1.74*x[1]  +0.9557*x[2]      -0.7484*x[3]
      +0.7399*x[4]      -2.336*x[5] -1.356*x[6] -1.325*x[7] -
0.6889*x[8] +0.2893*x[9]      -0.6938*x[10]     +0.05672*x[11]
      +0.2945*x[12]     +1.36*x[13] -0.9224*x[14]     +1.326*x[15]
      -0.2123*x[16]     +1.824*x[17]      +0.136*x[18]      -
1.526*x[19]    +0*u1    +0.8671*u2
    dx8dt = -0.3924*x[0]      +0.9739*x[1]      -1.57*x[2]  -
0.5995*x[3] -1.417*x[4] +1.454*x[5] +1.108*x[6] -2.047*x[7]
      +0.2119*x[8]      -2.016*x[9] -0.7274*x[10]     +0.5994*x[11]
      -0.5053*x[12]     -0.6632*x[13]     +0.8841*x[14]     -
0.9682*x[15]      +0.496*x[16]      -1.68*x[17] -0.5078*x[18]     -
2.292*x[19]    +0.1769*u1     -0.07989*u2
    dx9dt = 1.588*x[0]  -1.835*x[1] +2.778*x[2] -1.045*x[3]
      +0.8129*x[4]      -1.532*x[5] +0.2996*x[6]      +0.4092*x[7]
      -1.635*x[8] +2.07*x[9]  +0.811*x[10]      +0.5032*x[11]
      +0.09832*x[12]    +1.696*x[13]      -0.9083*x[14]
      +1.672*x[15]      -0.1761*x[16]     -0.1182*x[17]     -
0.9934*x[18]      +1.224*x[19]    -0.3075*u1    +0.8985*u2
    dx10dt = -0.05716*x[0]    +3.078*x[1] +2.561*x[2] +0.9238*x[3]
      -0.08049*x[4]     -1.83*x[5]  -0.5941*x[6]      +0.7435*x[7]
      -0.8183*x[8]      -1.579*x[9] -0.7263*x[10]     +1.954*x[11]
      -0.8051*x[12]     +0.05833*x[13]    -0.5489*x[14]     -
1.409*x[15] +2.323*x[16]      +1.979*x[17]      -1.868*x[18]      -
0.2734*x[19]    -0.1318*u1    +0.1837*u2
    dx11dt = -1.534*x[0]      +1.186*x[1] +1.843*x[2] -0.2953*x[3]
      -0.79*x[4]  -2.037*x[5] -0.8962*x[6]      -0.1266*x[7]      -
1.374*x[8]  -0.6778*x[9]      -2.54*x[10] -1.148*x[11]      -
1.192*x[12] +1.533*x[13]      +0.9508*x[14]     -0.71*x[15]
      +0.4906*x[16]     +1.348*x[17]      -0.3508*x[18]     -
1.381*x[19]     +0*u1   +0.2908*u2
    dx12dt = -1.334*x[0]      -0.05293*x[1]     -2.141*x[2]
      +0.1946*x[3]      -1.515*x[4] +3.373*x[5] -0.7687*x[6]      -
0.6886*x[7] -0.7571*x[8]      -2.551*x[9] -0.3203*x[10]     -
1.129*x[11] +1.459*x[12]      +0.03717*x[13]    -0.8015*x[14]     -
0.3177*x[15]      +1.898*x[16]      -2.641*x[17]      +0.8335*x[18]
      -1.833*x[19]     +0*u1  +0.1129*u2
    dx13dt = 0.1065*x[0]      -0.5554*x[1]      +0.4454*x[2]      -
0.9644*x[3] -0.4928*x[4]      +1.526*x[5] +0.2531*x[6]      -
1.441*x[7]  +0.966*x[8] -0.5905*x[9]      +0.1721*x[10]     -
2.057*x[11] -2.132*x[12]      +0.01619*x[13]    +1.713*x[14]      -
0.915*x[15] +1.545*x[16]      -1.891*x[17]      +1.21*x[18] -
0.7068*x[19]    -0.198*u1     +0.44*u2
    dx14dt = -1.991*x[0]      +0.9856*x[1]      +2.056*x[2]
      +1.291*x[3] +0.796*x[4] -2.359*x[5] -0.8435*x[6]      -
0.9635*x[7] -0.1495*x[8]      -0.469*x[9] -0.2485*x[10]
      +0.9014*x[11]     -0.1367*x[12]     -1.97*x[13] +0.495*x[14]
      +1.203*x[15]      -0.2256*x[16]     +1.347*x[17]      -
0.6591*x[18]      +0.3882*x[19]     +0*u1 +0.1017*u2
    dx15dt = 1.689*x[0] -0.2474*x[1]      +0.15*x[2]  +0.01819*x[3]
      +1.268*x[4] -1.876*x[5] +0.632*x[6] +0.7666*x[7]
      +0.1137*x[8]      +1.873*x[9] -0.5716*x[10]     +0.91*x[11]
      +0.206*x[12]      +0.5918*x[13]     -1.622*x[14]
```

```
      +0.9979*x[15]     -1.913*x[16]      +1.177*x[17]
      +0.01638*x[18]    +0.738*x[19]     +0*u1  +2.787*u2
    dx16dt = 0.9128*x[0]      +0.5218*x[1]      +3.217*x[2]
      +0.8333*x[3]      +0.04456*x[4]     +0.7694*x[5]      -
1.065*x[6]  +0.3341*x[7]      -1.212*x[8] +0.7797*x[9]      -
0.008353*x[10]    -0.2017*x[11]     -1.146*x[12]      -0.3141*x[13]
      -0.1908*x[14]     -1.399*x[15]      +2.076*x[16]      -
0.7609*x[17]      -0.2437*x[18]     +1.52*x[19]            +0.2296*u1
      +0*u2
    dx17dt = -2.833*x[0]      -1.455*x[1] -2.7*x[2]   +0.561*x[3] -
2.142*x[4]  +1.927*x[5] +0.304*x[6] +1.295*x[7] -1.311*x[8]
      +0.2297*x[9] +1.101*x[10]     +0.18*x[11] +1.945*x[12]      -
0.3131*x[13]      -0.3687*x[14]     +0.05187*x[15]    -4.041*x[16]
      -0.2862*x[17]     -0.6418*x[18]     +0.9395*x[19]     +0*u1 -
1.854*u2
    dx18dt = 1.463*x[0] -1.824*x[1] +2.102*x[2] +1.508*x[3]
      +2.293*x[4] +0.4533*x[5]      -1.03*x[6]  +0.5963*x[7]
      +0.5306*x[8]      -2.19*x[9]  -1.827*x[10]      +2.206*x[11]
      +0.1556*x[12]     -1.634*x[13]      -0.1956*x[14]
      +0.413*x[15]      +2.192*x[16]      -1.647*x[17]      -
1.647*x[18] +0.4107*x[19]    -0.6169*u1   +0*u2
    dx19dt = 0.5868*x[0]      +0.1263*x[1]      +1.82*x[2]  -
2.001*x[3]  +1.134*x[4] -1.784*x[5] -0.4244*x[6]      -0.7385*x[7]
      +0.04053*x[8]     +1.995*x[9] +0.1191*x[10]     -0.4276*x[11]
      -0.8157*x[12]     +0.1027*x[13]     +0.533*x[14]
      +0.654*x[15]      -1.307*x[16]      +1.901*x[17]      -
2.327*x[18] +1.74*x[19]      +0.2748*u1   -1.093*u2
    dx20dt = 0.8673*x[0]      +1.847*x[1] -0.04373*x[2]
      +0.5824*x[3]      -2.139*x[4] +1.023*x[5] +1.119*x[6]
      +1.221*x[7] -1.137*x[8] -1.165*x[9] -0.8055*x[10]
      +1.374*x[11]      -1.111*x[12]      +0.7632*x[13]
      +0.2468*x[14]     -2.771*x[15]      +1.729*x[16]      -
0.5774*x[17]      -0.8746*x[18]     -1.772*x[19]     +0.6011*u1   -
0.4336*u2

return(dx1dt,dx2dt,dx3dt,dx4dt,dx5dt,dx6dt,dx7dt,dx8dt,dx9dt,dx10dt,d
x11dt,dx12dt,dx13dt,dx14dt,dx15dt,dx16dt,dx17dt,dx18dt,dx19dt,dx20dt)
y1 = zeros(N); y2 = zeros(N)
for i in range(0,N-1):
    tt = [t[i],t[i+1]]
    x = odeint(dfdt,x0,t,(u1[i],u2[i],a))
    y1[i] = x[0,0]
    y2[i] = x[4,4]
    x0 = x[1,:]
yref1 = zeros(N); yref2= zeros(N)
# Reference Model
k= -2
yref1 = zeros(N); x0r1 = [0,0]
def ref1(x,t,u1,k):
    dxdt1 = k*x[0] + -1*x[1] + u1
    dxdt2 = 1*x[0]
    return(dxdt1,dxdt2)

for i in range(0,N-1):
    tt = [t[i],t[i+1]]
    x = odeint(ref1,x0r1,t,(u1[i],k))
    yref1[i] = x[1,1]*1
    x0r1 = x[1,:]


m= -2
```

```
yref2 = zeros(N); x0r2 = [0,0]
def ref2(x,t,u2,m):
    dxdt1 = m*x[0] + -1*x[1] + u2
    dxdt2 = 1*x[0]
    return(dxdt1,dxdt2)

for i in range(0,N-1):
    tt = [t[i],t[i+1]]
    x = odeint(ref2,x0r2,t,(u2[i],m))
    yref2[i] = x[1,1]*1
    x0r2 = x[1,:]

#) Defining parameters for identification
##first neuron
nu1 = 4 ;nu12 = 3; ny1 = 4; ny12 = 3 ;nx1 = 1 + nu1+nu12 +ny1 +ny12
nw1 = nx1
##second neuraon
nu2 = 4; nu21 = 3; ny2 = 4; ny21 = 3 ;nx2 = 1 + nu1+nu21 +ny1 + ny21
nw2 = nx2
##common variables
nx = nx1 = nx2; nw = nw1 = nw2
##common values
mu = .5; Epochs =2000
##first neuron
w1 = randn(nw)/nw; ynn1 = y1.copy(); e1 = zeros(N); x1 = zeros(nx)
SSE1 = zeros(Epochs); dydw1 = zeros((N,nw))
##second neuron
w2 = randn(nw)/nw; ynn2 = y2.copy(); e2 = zeros(N); x2 = zeros(nx)
SSE2 = zeros(Epochs); dydw2 = zeros((N,nw))
#6) Adaptive Identification
for epoch in range(0,Epochs):
    for k in range(max(ny1,nu1,nu12,ny12),N-1):
        ##first neuron x vector
        x1[0] = 1
        x1[1:(ny1+1)] =                    ynn1[range(k-1,k-(ny1+1),-
1)]
        x1[(ny1+1):(ny1+ny12+1)] =        ynn2[range(k-1,k-(ny12+1),-
1)]
        x1[(ny1+ny12+1):(ny1+ny12+nu1+1)]  =  u1[range(k-1,k-(nu1+1),-
1)]
        x1[(ny1+ny12+nu1+1):] =             u2[range(k-1,k-(nu12+1),-
1)]

        ##second neuron x vector
        x2[0] = 1
        x2[1:(ny2+1)] =                    ynn2[range(k-1,k-(ny2+1),-
1)]
        x2[(ny2+1):(ny2+ny21+1)] =        ynn1[range(k-1,k-(ny21+1),-
1)]
        x2[(ny2+ny21+1):(ny2+ny21+nu2+1)]  =  u2[range(k-1,k-(nu2+1),-
1)]
        x2[(ny2+ny21+nu2+1):] =             u1[range(k-1,k-(nu21+1),-
1)]
        #update rule
        ynn1[k] = dot(w1,x1)
        e1[k] = y1[k] - ynn1[k]
        dydw1[k,:] = x1
        #gd
        dw1= mu/(1+dot(x1,x1))*e1[k]*dydw1[k,:]
        w1 = w1+dw1
```

```
        #update rule
        ynn2[k] = dot(w2,x2)
        e2[k] = y2[k] - ynn2[k]
        dydw2[k,:] = x2
        #gd
        dw2= mu/(1+dot(x2,x2))*e2[k]*dydw2[k,:]
        w2 = w2+dw2
#Desired Values
d1 = u1.copy()
d2 = u2.copy()
#Set points
setpoint1 = d1.copy()
setpoint2 = d2.copy()
#Controller Parameters
muv1 = .23 ; muv2 = .19 ;  r01 = .004 ; r02 = .09
v1 = zeros(nw); v2=zeros(nw); q1 = zeros(N) ; q2 = zeros(N)
eref1=zeros(N) ; eref2=zeros(N); xi1 = ones(nw) ; xi2 = ones(nw)
dxdv1 = zeros((nw,nw)) ; dxdv2 = zeros((nw,nw))
dydv1 = zeros(nw) ; dydv2 = zeros(nw); nd = nu1 + nu12 ;ny= ny1+ny12
yn1 = y1.copy(); yn2 = y2.copy()
for epoch in range(0,Epochs):
    for k in range(max(ny1,nu1,nu12,ny12),N-1):
        xi1[0] = 1
        xi1[1:(ny1+1)] =                    yn1[range(k-1,k-(ny1+1),-
1)]
        xi1[(ny1+1):(ny1+ny12+1)] =        yn2[range(k-1,k-(ny12+1),-
1)]
        xi1[(ny1+ny12+1):(ny1+ny12+nu1+1)] = d1[range(k-1,k-(nu1+1),-
1)]
        xi1[(ny1+ny12+nu1+1):] =            d2[range(k-1,k-(nu12+1),-
1)]

        q1[k] = dot(v1,xi1)
        setpoint1[k]= d1[k] - r01*q1[k]

        xi2[0] = 1
        xi2[1:(ny2+1)] =                    yn2[range(k-1,k-(ny2+1),-
1)]
        xi2[(ny2+1):(ny2+ny21+1)] =        yn1[range(k-1,k-(ny21+1),-
1)]
        xi2[(ny2+ny21+1):(ny2+ny21+nu2+1)] = d2[range(k-1,k-(nu2+1),-
1)]
        xi2[(ny2+ny21+nu2+1):] =            d1[range(k-1,k-(nu21+1),-
1)]

        q2[k] = dot(v2,xi2)
        setpoint2[k]= d2[k] - r02*q2[k]

        #first neuron
        x1[0] = 1
        x1[1:(ny1+1)] =                     yn1[range(k-1,k-(ny1+1),-
1)]
        x1[(ny1+1):(ny1+ny12+1)] =         yn2[range(k-1,k-(ny12+1),-
1)]
        x1[(ny1+ny12+1):(ny1+ny12+nu1+1)]   =   setpoint1[range(k-1,k-
(nu1+1),-1)]
        x1[(ny1+ny12+nu1+1):] =                setpoint2[range(k-1,k-
(nu12+1),-1)]

        #second neuron
        x2[0] = 1
```

```
        x2[1:(ny2+1)] =                      yn2[range(k-1,k-(ny2+1),-
1)]
        x2[(ny2+1):(ny2+ny21+1)] =          yn1[range(k-1,k-(ny21+1),-
1)]
        x2[(ny2+ny21+1):(ny2+ny21+nu2+1)]   =   setpoint2[range(k-1,k-
(nu2+1),-1)]
        x2[(ny2+ny21+nu2+1):] =                 setpoint1[range(k-1,k-
(nu21+1),-1)]

        yn1[k] = dot(w1,x1)
        eref1[k] = yref1[k] - yn1[k]
        dxdv1[1:ny,:]=dxdv1[2:1+ny,:];dxdv1[ny,:]=dydv1
        dxdv1[1+ny:-1,:]=dxdv1[2+ny:,:];dxdv1[-1,:]=-r01*xi1
        dydv1 = dot(w1,dxdv1)
        #dxdv[-1,:] = -xi

        dv1 = muv1*eref1[k]*dydv1 #dot(w1,dxdv1)
        v1 = v1 +dv1

        yn2[k] = dot(w2,x2)
        eref2[k] = yref2[k] - yn2[k]
        dxdv2[1:ny,:]=dxdv2[2:1+ny,:];dxdv2[ny,:]=dydv2
        dxdv2[1+ny:-1,:]=dxdv2[2+ny:,:];dxdv2[-1,:]=-r02*xi2
        dydv2 = dot(w2,dxdv2)
        #dxdv[-1,:] = -xi
        dv2 = muv2*eref2[k]*dydv2 #dot(w2,dxdv2)
        v2 = v2 +dv2
    SSE2[epoch] = sum(eref2*eref2)
    SSE1[epoch] = sum(eref1*eref1)
    print(SSE1[epoch]),(SSE2[epoch])

figure(figsize=(14,9))
subplots_adjust(hspace=.4)
subplot(211)
title("Controller sample-by-sample adaptation roller rig model")
#plot(y1,'b',label="$y_r$")
plot(yref1,'m',label="$y_{ref1}$")
plot(yn1,'g',label="$y_n1$")
plot(d1, "b--", label="d1")
grid();legend()
subplot(212)
#plot(y2,'b',label="$y_r$")
plot(yref2,'m',label="$y_{ref2}$")
plot(yn2,'g',label="$y_n2$")
plot(d2, "b--", label="d2")
grid();legend()
show()
```

Code for Second Order RLS Neuro Controller Implementation

```
from numpy import*
from matplotlib.pyplot import*
from scipy.integrate import odeint
from numpy.random import randn

#Input signals
t=arange(0,200,.1)
N = len(t)
u1=sign(sin(2*pi*t/40))
u2=sign(sin(2*pi*t/50))
```

```
u1[0:int(N/7)]=1;                              u1[int(N/7):int(2*N/7)]=2;
u1[int(2*N/7):int(3*N/7)]=3
u1[int(3*N/7):int(4*N/7)]=4;u1[int(4*N/7):int(5*N/7)]=3
u1[int(5*N/7):int(6*N/7)]=2; u1[int(6*N/7):int(7*N/7)]=1

u2[0:int(N/7)]=1;                              u2[int(N/7):int(2*N/7)]=2;
u2[int(2*N/7):int(3*N/7)]=3
u2[int(3*N/7):int(4*N/7)]=4; u2[int(4*N/7):int(5*N/7)]=3
u2[int(5*N/7):int(6*N/7)]=2; u2[int(6*N/7):int(7*N/7)]=1


y1 = zeros(N); y2 = zeros(N); x0 = [0,0]
a1 = -1.0; a2 = 0.5; a3 = 3.0; a4 = 1.5
b1 = -0.5; b2 = -2.0; b3 = 0; b4 = -1.0

def dfdt(x,t,u1,u2,a1,a2,a3,a4,b1,b2,b3,b4):
    dx1dt = a1*x[0] + a2*x[1] + a3*u1 + a4*u2
    dx2dt = b1*x[0] + b2*x[1] + b3*u1 + b4*u2
    return (dx1dt, dx2dt)

for i in range(0,N-1):
    tt = [t[i],t[i+1]]
    x = odeint(dfdt,x0,t,(u1[i],u2[i],a1,a2,a3,a4,b1,b2,b3,b4))
    y1[i] = .4*x[0,0] - .1*x[1,1]
    y2[i] = -.98*x[1,1]
    x0 = x[1,:]


#reference model
yref1 =zeros(N); yref2 = zeros(N); k= -2; yref1 = zeros(N); x0r1 =
[0,0]
def ref1(x,t,u1,k):
    dxdt1 = k*x[0] + -1*x[1] + u1
    dxdt2 = 1*x[0]
    return(dxdt1,dxdt2)

for i in range(0,N-1):
    tt = [t[i],t[i+1]]
    x = odeint(ref1,x0r1,t,(u1[i],k))
    yref1[i] = x[1,1]*1

    x0r1 = x[1,:]

m= -2
yref2 = zeros(N); x0r2 = [0,0]
def ref2(x,t,u2,m):
    dxdt1 = m*x[0] + -1*x[1] + u2
    dxdt2 = 1*x[0]
    return(dxdt1,dxdt2)

for i in range(0,N-1):
    tt = [t[i],t[i+1]]
    x = odeint(ref2,x0r2,t,(u2[i],m))
    yref2[i] = x[1,1]*1
    x0r2 = x[1,:]
y1 = y1; y2 = y2
#) Defining parameters
#first neuron
nu1 = 4 ;nu12 = 3; ny1 = 4; ny12 = 3; nx1 = 1 + nu1+nu12 +ny1 +ny12
nw1 = nx1
#second neuron
```

```
nu2 = 4; nu21 = 3; ny2 = 4; ny21 = 3; nx2 = 1 + nu1+nu21 +ny1 + ny21
nw2 = nx2
#common variables
nx = nx1 = nx2; nw = nw1 = nw2
##common values
mu = 0.15
Epochs =5; Epochsc = 1
##first neuron
w1 = randn(nw)/nw; ynn1 =zeros(N) #y1.copy(); e1 = zeros(N); x1 =
zeros(nx)
SSE1i = zeros(Epochs); SSE1 = zeros(Epochsc); dydw1 = zeros((N,nw))
##second neuron
w2 = randn(nw)/nw; ynn2 = zeros(N) #y2.copy();e2 = zeros(N);x2 =
zeros(nx)
SSE2i = zeros(Epochs); SSE2 = zeros(Epochsc); dydw2 = zeros((N,nw))
#rls variables
R1=1/delta*identity(nw)
R2=1/delta*identity(nw)
#6) Adaptive Identification
for epoch in range(0,Epochs):
    for k in range(max(ny1,nu1,nu12,ny12),N-1):
        ##first neuron x vector
        x1[0] = 1
        x1[1:(ny1+1)] =                    ynn1[range(k-1,k-(ny1+1),-
1)]
        x1[(ny1+1):(ny1+ny12+1)] =        ynn2[range(k-1,k-(ny12+1),-
1)]
        x1[(ny1+ny12+1):(ny1+ny12+nu1+1)] = u1[range(k-1,k-(nu1+1),-
1)]
        x1[(ny1+ny12+nu1+1):] =             u2[range(k-1,k-(nu12+1),-
1)]

        ##second neuron x vector
        x2[0] = 1
        x2[1:(ny2+1)] =                    ynn2[range(k-1,k-(ny2+1),-
1)]
        x2[(ny2+1):(ny2+ny21+1)] =        ynn1[range(k-1,k-(ny21+1),-
1)]
        x2[(ny2+ny21+1):(ny2+ny21+nu2+1)] = u2[range(k-1,k-(nu2+1),-
1)]
        x2[(ny2+ny21+nu2+1):] =             u1[range(k-1,k-(nu21+1),-
1)]
        #update rule
        ynn1[k] = dot(w1,x1)
        e1[k] = y1[k] - ynn1[k]
        dydw1[k,:] = x1
        # rls
        R11 = dot(dot(dot(R1,x1),x1.T),R1)
        R12 = mu + dot(dot(x1,R1),x1.T)
        R1 = 1/mu * (R1 - R11/R12)
        dw1 = dot(R1,x1.T)*e1[k]
        w1 = w1+dw1
        #update rule
        ynn2[k] = dot(w2,x2)
        e2[k] = y2[k] - ynn2[k]
        dydw2[k,:] = x2
        #rls
        R21 = dot(dot(dot(R2,x2),x2.T),R2)
        R22 = mu + dot(dot(x2,R2),x1.T)
        R2 = 1/mu * (R2 - R21/R22)
        dw2 = dot(R2,x2.T)*e2[k]
```

```
        w2 = w2+dw2
d1 = u1.copy(); d2 = u2.copy();setpoint1 = d1.copy(); setpoint2 =
d2.copy()
#rls variables
deltac = 1
R1c=1/deltac*identity(nw); R2c=1/deltac*identity(nw)
muv = 0.9998; r01 = 0.05; r02=0.05; v1 = zeros(nw); v2=zeros(nw)
q1 = zeros(N) ; q2 = zeros(N); eref1=zeros(N) ; eref2=zeros(N)
xi1 = ones(nw) ; xi2 = ones(nw)
dxdv1 = zeros((nw,nw)) ; dxdv2 = zeros((nw,nw))
dydv1 = zeros(nw) ; dydv2 = zeros(nw)
nd = nu1 + nu12 ;ny= ny1+ny12 ; yn1 = zeros(N) #y1.copy();
yn2 = zeros(N) #y2.copy()
for epoch in range(0,Epochsc):
    for k in range(max(ny1,nu1,nu12,ny12),N-1):
        xi1[0] = 1
        xi1[1:(ny1+1)] =                    yn1[range(k-1,k-(ny1+1),-
1)]
        xi1[(ny1+1):(ny1+ny12+1)] =        yn2[range(k-1,k-(ny12+1),-
1)]
        xi1[(ny1+ny12+1):(ny1+ny12+nu1+1)] = d1[range(k-1,k-(nu1+1),-
1)]
        xi1[(ny1+ny12+nu1+1):] =            d2[range(k-1,k-(nu12+1),-
1)]

        q1[k] = dot(v1,xi1)
        setpoint1[k]= d1[k] - r01*q1[k]

        xi2[0] = 1
        xi2[1:(ny2+1)] =                    yn2[range(k-1,k-(ny2+1),-
1)]
        xi2[(ny2+1):(ny2+ny21+1)] =        yn1[range(k-1,k-(ny21+1),-
1)]
        xi2[(ny2+ny21+1):(ny2+ny21+nu2+1)] = d2[range(k-1,k-(nu2+1),-
1)]
        xi2[(ny2+ny21+nu2+1):] =            d1[range(k-1,k-(nu21+1),-
1)]

        q2[k] = dot(v2,xi2)
        setpoint2[k]= d2[k] - r02*q2[k]

        #first neuron identification
        x1[0] = 1
        x1[1:(ny1+1)] =               yn1[range(k-1,k-(ny1+1),-1)]
        x1[(ny1+1):(ny1+ny12+1)] =     yn2[range(k-1,k-(ny12+1),-1)]
        x1[(ny1+ny12+1):(ny1+ny12+nu1+1)]=     setpoint1[range(k-1,k-
(nu1+1),-1)]
        x1[(ny1+ny12+nu1+1):] =      setpoint2[range(k-1,k-(nu12+1),-
1)]

        #second neuron identification
        x2[0] = 1
        x2[1:(ny2+1)] =                    yn2[range(k-1,k-(ny2+1),-
1)]
        x2[(ny2+1):(ny2+ny21+1)] =         yn1[range(k-1,k-(ny21+1),-
1)]
        x2[(ny2+ny21+1):(ny2+ny21+nu2+1)]=     setpoint2[range(k-1,k-
(nu2+1),-1)]
        x2[(ny2+ny21+nu2+1):] =      setpoint1[range(k-1,k-(nu21+1),-
1)]
```

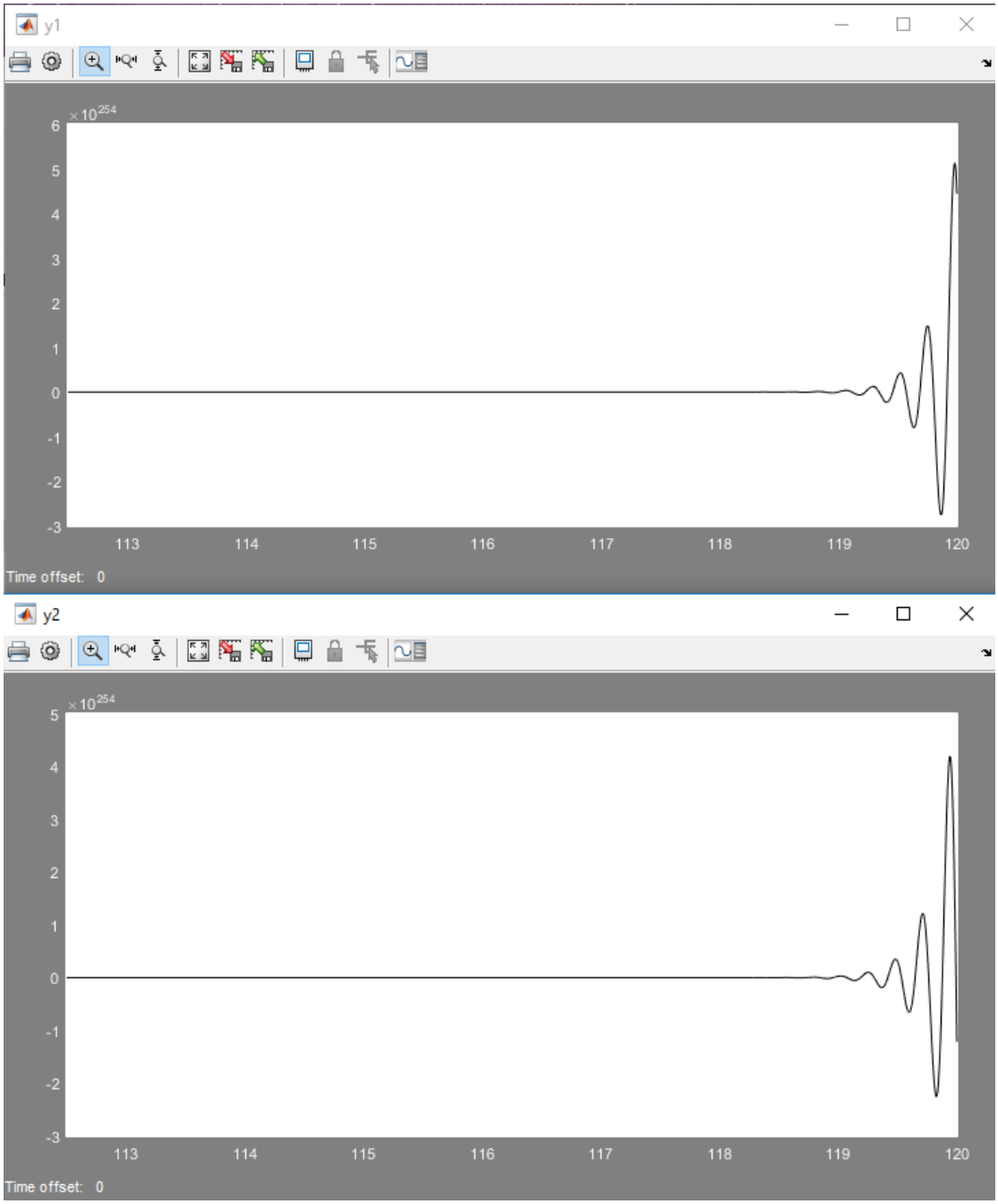
```
            yn1[k] = dot(w1,x1)
            eref1[k] = yref1[k] - yn1[k]
            dxdv1[1:ny,:]=dxdv1[2:1+ny,:];dxdv1[ny,:]=dydv1
            dxdv1[1+ny:-1,:]=dxdv1[2+ny:,:];dxdv1[-1,:]=-r01*xi1
            dydv1 = dot(w1,dxdv1)
            #dxdv1[-1,:] = -xi1
            #rls
            R11c = dot(dot(dot(R1c,xi1),xi1.T),R1c)
            R12c = muv + dot(dot(xi1,R1c),xi1.T)
            R1c = 1/muv * (R1c - R11c/R12c)
            dv1 = dot(R1c,xi1.T)*eref1[k]
            v1 = v1+dv1

            yn2[k] = dot(w2,x2)
            eref2[k] = yref2[k] - yn2[k]
            dxdv2[1:ny,:]=dxdv2[2:1+ny,:];dxdv2[ny,:]=dydv2
            dxdv2[1+ny:-1,:]=dxdv2[2+ny:,:];dxdv2[-1,:]=-r02*xi2
            dydv2 = dot(w2,dxdv2)
            # dxdv2[-1,:] = -xi2
            #rls
            R21c = dot(dot(dot(R2c,xi2),xi2.T),R2c)
            R22c = muv + dot(dot(xi2,R2c),xi2.T)
            R2c = 1/muv * (R2c - R21c/R22c)
            dv2 = dot(R2c,xi2.T)*eref2[k]
            v2 = v2+dv2
        SSE2[epoch] = sum(eref2*eref2)
        SSE1[epoch] = sum(eref1*eref1)
        print(SSE1[epoch]),(SSE2[epoch])

figure(figsize=(14,9))
subplots_adjust(hspace=.4)
subplot(211)
title("Controller sample-by-sample adaptation for 2nd order stable
linear theoratical plant")
#plot(y1,'b',label="$y_r$")
plot(yref1,'m',label="$y_{ref1}$")
plot(d1,"b--", label="d1")
plot(yn1,'g',label="$y_n1$")
grid();legend()
subplot(212)
#plot(y2,'b',label="$y_r$")
plot(yref2,'m',label="$y_{ref2}$")
plot(d2,"b--", label="d2")
plot(yn2,'g',label="$y_n2$")
grid();legend()
show()
```

Unstable System Responses

Roller rig model step response



Step response of roller-rig after poles replacement