Use Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at Idaho State University, I agree that the library shall make it freely available for inspection. I further state that permission to download and/or print my thesis for scholarly purposes may be granted by the Dean of the Graduate School, Dean of my academic division, or by the University Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Signature _____

Date _____

SYSTEM IDENTIFICATION USING NUCLEAR NORM AND TABU SEARCH OPTIMIZATION

By

Asif Ahmed

A thesis submitted in partial fulfillment of the requirements for the degree of

Masters of Science in Measurement & Controls Engineering

College of Science and Engineering, Idaho State University

May, 2016

RESPECTED GRADUATE COMITTEE

To the Graduate Faculty:

The members of the committee appointed to examine the thesis of ASIF AHMED find it satisfactory and recommend that it be accepted.

Dr. Marco P. Schoen, Ph.D., P.E.
[Major Advisor]

Dr. Ken W. Bosworth, Ph.D.
[Co Advisor]

Dr. Bruce Savage, Ph.D., P.E.
[Graduate Faculty Representative]

ACKNOWLEDGEMENTS

I cordially thank Dr. Marco P. Schoen, Major Advisor, and Dr. Ken W. Bosworth, Co Advisor of this research for their invaluable guidance and support which was instrumental in achieving the goals for this research. Dr. Ken W. Bosworth's thoughtful and patient advice was the key to achieving results for this research, and Dr. Marco P. Schoen's instruction and advice was a source of great motivation and strength for me as an engineering graduate student.

I cordially thank Dr. Gene Stuffle for providing me with an opportunity to serve as Electrical Engineering Graduate Teaching Assistant during my first year as an engineering graduate student, and Dr. Steve Chiu and Dr. Mikle Ellis for allowing me to serve as Teaching Assistant for their courses. I thank the engineering administrative staff including Ms. Ellen Combs, and Ms. Jennilee Overocker for their support throughout my work as an engineering graduate student at Idaho State University. Special thank you to Mr. Mark Cook for his hard work in ensuring proper maintenance of Lillibridge Engineering Building. I cordially thank Mr. Jesse Kiboko for all his support and advice. I thank Dr. Bennet Palmer, and all the professors that I had an opportunity to learn from.

It has been my honor and pleasure to be a graduate student at Idaho State University, and express deep gratitude to all the respective faculty, staff, and students.

Table of Contents

List of Figures

List of Tables

ABSTRACT

In recent years, subspace System Identification (SI) algorithms have seen increased research, stemming from advanced minimization methods being applied to the Nuclear Norm (NN) approach in system identification. These minimization algorithms are based on hard computing methodologies. To the authors knowledge, as of now, there has been no work reported that utilizes soft computing algorithms to address the minimization problem within the nuclear norm SI algorithm. A linear, time-invariant, discrete time system is used in this work as the basis model for characterizing a dynamical system to be identified. The main objective is to extract a mathematical model from collected experimental input-output data. Hankel matrices are constructed from experimental data, and the extended observability matrix is employed to define an estimated output of the system. This estimated output and the actual – measured – output are utilized to construct a minimization problem. An embedded rank measure assures for minimum state realization outcomes. Current NN-SI algorithms employ hard computing algorithms for minimization. In this work, we propose a simple Tabu Search (TS) algorithm for minimization. TS algorithm based SI is compared with the iterative Alternating Direction Method of Multipliers (ADMM) line search optimization based NN-SI. For comparison, several different benchmark system identification problems are used and tabulated. Results show improved performance of the proposed SI-TS algorithm compared to the NN-SI ADMM algorithm.

CHAPTER 1.0 - INTRODUCTION

1.1 - Overview

Nuclear norm by definition is the sum of singular values of a matrix [1]. It involves estimating system matrices A, B, C, D of a state –space realization from given input-output data that is contaminated with noise [1]. The method is aimed at utilizing a discrete-time state space model, as well as the properties of observability and controllability matrices to develop corresponding Hankel and Toeplitz matrices respectively [1]. Amongst the benefits of this method is that of minimizing the nuclear norm of a given matrix and thereby minimizing its rank, which would represent the number of linearly independent rows or columns of the matrix [5, 6].

This thesis presents application, proof, and validation of Subspace Identification of various dynamic benchmark models through Nuclear Norm and Tabu Search optimization. In the past, significant research has been conducted in the area of system identification using nuclear norm for systems with missing inputs and outputs. It has been proven that as the amount of missing input-output data increases, the performance of nuclear norm system identification method slowly degrades with it [6].

Following the mathematical derivations, Matlab™ code and corresponding Simulink™ models are developed to apply and analyze the associated system identification procedures, in turn providing a platform to validate the derived method.

$$\left|\left|\boldsymbol{A}\right|\right|_{\mathbf{1}} \triangleq \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{n}} \boldsymbol{\sigma}_{\boldsymbol{i}}\left(\boldsymbol{A}\right) \qquad (\mathbf{1.1})$$

$$\left|\left|\boldsymbol{A}\right|\right|_{\mathbf{1}} = \boldsymbol{tr}\left(\sqrt{\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}}\right) \qquad (\mathbf{1.2})$$

Nuclear Norm is computed by using definition of Schatten Norm as the starting point, as shown in Equations *(1.1)* and *(1.2)* [7]. σ_i represents the ith singular value [7].

1.2 – Objective

- First objective of this research work is to mathematically derive proofs for each step of Harshad Deshmane's Iterative ADMM system identification method, and to describe Tabu Search system identification method in detail.
- Second objective of this work is to programmatically engineer a system using Matlab™ and Simulink™ software for applying, simulating, and analyzing the two optimization methods of Nuclear Norm and Tabu Search, thus validating mathematical foundations of each method.
- Third objective is to analyze and compare performance of both methods through rigorous testing and simulation on each of twelve benchmark systems, thereby examining the performance of the two optimization techniques.

1.3 – Organization of Thesis

This thesis is organized as follows:

- Chapter 1.0 provides an overview and objective of the research problem.
- Chapter 2.0 details system identification mathematical derivations and proofs.
- Chapter 3.0 describes application of Iterative ADMM method using Matlab™ and Simulink™ software.
- Chapter 4.0 describes application of Tabu Search method using Matlab™ and Simulink™ software.
- Chapter 5.0 compares and analyzes the two optimization techniques.
- Chapter 6.0 provides conclusion and future work.

CHAPTER 2.0 – SYSTEM IDENTIFICATION THEORY

2.1 - Basis System

A linear, time-invariant, discrete time system is considered as the basis system. The system description involves the state vector *x(k)*, input *u(k)*, process noise *w(k)*, measurement noise *v(k)*, output *y(k)*, and system matrices *A*, *B*, *C*, and *D*. State vector for *(k+1)* element is defined as follows:

$$\boldsymbol{x}(\boldsymbol{k}+\mathbf{1}) = \boldsymbol{A}\boldsymbol{x}(\boldsymbol{k}) + \boldsymbol{B}\boldsymbol{u}(\boldsymbol{k}) + \boldsymbol{w}(\boldsymbol{k}) \tag{2.1}$$

The output y(k), is defined as follows:

$$\boldsymbol{y}(\boldsymbol{k}) = \boldsymbol{C}\boldsymbol{x}(\boldsymbol{k}) + \boldsymbol{D}\boldsymbol{u}(\boldsymbol{k}) + \boldsymbol{v}(\boldsymbol{k}) \tag{2.2}$$

In the above system, $\{y\} \in \mathbb{R}^{n_y \mathrm{x} L}$, and $\{u\} \in \mathbb{R}^{n_u \mathrm{x} L}$, represent the output data sequence and input data sequence, respectively. These sequences are recorded during a system experimentation with a sample frequency f_s. This yields *L* discrete data points for the input and output sequence, respectively. Here, n_y represents the number of outputs, and n_u represents the number of inputs. Furthermore, $\{w\} \in \mathbb{R}^{n \mathrm{x} L}$ is representative of process noise, and $\{v\} \in \mathbb{R}^{n_y \mathrm{x} L}$ represents measurement noise. $\{x\} \in \mathbb{R}^{n \mathrm{x} L}$ gives the state vector.

2.2 - Kalman Filter

State estimation of the variable *x(k)* involves the use of a Kalman filter *K*. By definition, a Kalman filter is used to recursively and optimally estimate concerned parameters while disregarding associated inaccuracies [8]. System represented by

Equations *(2.1)* and *(2.2)* are reformulated using the gain K of the Kalman filter as follows:

$$\boldsymbol{K} = \boldsymbol{P_f C^T}\left[\boldsymbol{R_f} + \boldsymbol{C P_f C^T}\right] \in \mathbb{R}^{nxn_y} \qquad (\mathbf{2.3})$$

In the above representation of the Kalman filter, $P_f \in \mathbb{R}^{nxn}$ is a solution to the steady-state algebraic Riccati equation. A Riccati equation by definition is any first order nonlinear ordinary differential equation [9]. R_f in Equation *(2.3)* is defined as $R_f = E[v(k)' * v(k)]$, where E is the expectation operator and *v(k)* is the measurement noise.

<u>2.3 – Estimated System Representation</u>

The system representation, as given in Equation *(2.1)* is now represented as:

$$\boldsymbol{\hat{x}(k+1) = A\hat{x}(k) + Bu(k) + K\varepsilon(k)} \qquad (\mathbf{2.4})$$

$$\boldsymbol{\hat{y}(k) = C\hat{x}(k) + Du(k)} \qquad (\mathbf{2.5})$$

where $\varepsilon(k) = y(k) - C\hat{x}(k) - Du(k)$ (2.6)

In Equation *(2.1)*, with the process noise represented as *w(k)*, the system is said to be in process form. In Equation *(2.4)*, with *w(k)* being replaced by *Kε(k)*, the system is now in innovation form. Substitution of Equation *(2.6)* into Equation *(2.4)* gives the following:

$$\boldsymbol{\hat{x}(k+1) = A\hat{x}(k) + Bu(k) + Ky(k) - KC\hat{x}(k) - KDu(k)} \qquad (\mathbf{2.7})$$

$$\boldsymbol{\hat{x}(k+1) = [A - KC]\hat{x}(k) + [B - KD]u(k) + Ky(k)} \qquad (\mathbf{2.8})$$

$$\boldsymbol{\hat{y}(k) = C\hat{x}(k) + Du(k)} \qquad (\mathbf{2.9})$$

If we define $\bar{A} = [A - KC]$ & $\bar{B} = [B - KD]$, the predictor form of the given

system is given as follows:

$$\hat{\boldsymbol{x}}(\boldsymbol{k}+\mathbf{1}) = \bar{\boldsymbol{A}}\hat{\boldsymbol{x}}(\boldsymbol{k}) + \bar{\boldsymbol{B}}\boldsymbol{u}(\boldsymbol{k}) + \boldsymbol{K}\boldsymbol{y}(\boldsymbol{k}) \qquad (\mathbf{2.10})$$

$$\hat{\boldsymbol{y}}(\boldsymbol{k}) = \boldsymbol{C}\hat{\boldsymbol{x}}(\boldsymbol{k}) + \boldsymbol{D}\boldsymbol{u}(\boldsymbol{k}) \qquad (\mathbf{2.11})$$

2.4 – Hankel & Observability Matrices

A Hankel matrix by definition is one in which the elements are the same along all the reverse diagonals [10]. The Hankel matrices for the input and output respectively are represented as follows:

$$Us = \begin{pmatrix} u(1) & u(2) & \cdots & u(L-s+1) \\ u(2) & u(3) & \cdots & u(L-s+2) \\ \vdots & \vdots & \ddots & \vdots \\ u(s) & u(s+1) & \cdots & u(L) \end{pmatrix}, \qquad \textbf{(2.12)}$$

$$Y_s = \begin{pmatrix} y(1) & y(2) & \cdots & y(L-s+1) \\ y(2) & y(3) & \cdots & y(L-s+2) \\ \vdots & \vdots & \ddots & \vdots \\ y(s) & y(s+1) & \cdots & y(L) \end{pmatrix}. \qquad \textbf{(2.13)}$$

An observability matrix is defined to represent the internal functioning of a system [11]. The observability matrix for the system is given as follows:

$$Os = \begin{pmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{s-1} \end{pmatrix} \qquad \textbf{(2.14)}$$

2.5 – Derivation of Hankel and Toeplitz Matrices

Mathematically, a Hankel matrix is defined as follows [2]:

$$\boldsymbol{H} = \Gamma_{s+1}\boldsymbol{\Omega}_{s+1} \tag{2.15}$$

In the above equation, $\Gamma_{s+1}$is the observability matrix, and Ω_{s+1} is the controllability matrix. The goal is to find a controllability matrix that can yield specific Toeplitz and Hankel structures as given in matrices *(2.16)* and *(2.17)*:

$$T_{u,s} = \begin{pmatrix} D & 0 & \cdots & 0 \\ CB & D & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{s-2}B & CA^{s-3}B & \cdots & D \end{pmatrix} \tag{2.16}$$

$$H_{u,s} = \begin{pmatrix} 0 & \cdots & 0 & D \\ 0 & \cdots & D & CB \\ \vdots & \iddots & \vdots & \vdots \\ D & \cdots & CA^{s-3}B & CA^{s-2}B \end{pmatrix} \tag{2.17}$$

In order to find the controllability matrix from Equation *(2.15)*, we use the following formula:

$$\boldsymbol{\Omega}_{s+1} = \Gamma_{s+1}^{-1}\boldsymbol{H}_{u,s} \tag{2.18}$$

$$\Omega_{s+1} = \begin{pmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{s-1} \end{pmatrix}^{-1} \begin{pmatrix} 0 & \cdots & 0 & D \\ 0 & \cdots & D & CB \\ \vdots & \iddots & \vdots & \vdots \\ D & \cdots & CA^{s-3}B & CA^{s-2}B \end{pmatrix} \tag{2.19}$$

According to matrix multiplication properties, a matrix must be square and the determinant of the matrix must not be zero in order for the inverse of a matrix to exist

[13]. In this case, as is evident from Equation *(2.14)*, the observability matrix is not a square matrix. To compute the best approximation of the inverse, it is possible to compute the Moore Penrose Pseudo Inverse as follows:

$$\Gamma^{+}_{\mathbf{1}:\boldsymbol{s}} = (\Gamma^{\boldsymbol{T}}_{\mathbf{1}:\boldsymbol{s}}\,\Gamma_{\mathbf{1}:\boldsymbol{s}})^{-\mathbf{1}}\Gamma^{\boldsymbol{T}}_{\mathbf{1}:\boldsymbol{s}} \tag{2.20}$$

$$\Gamma^{+}_{\mathbf{1}:\boldsymbol{s}} = [\begin{pmatrix} C & CA & CA^2 \cdots CA^{s-1} \end{pmatrix} \begin{pmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{s-1} \end{pmatrix}]^{-1} \begin{pmatrix} C & CA & CA^2 \cdots CA^{s-1} \end{pmatrix} \tag{2.21}$$

$$\Gamma^{+}_{\mathbf{1}:\boldsymbol{s}} = \begin{pmatrix} CC & CACA+ & \cdots + CA^{s-1}CA^{s-1} \end{pmatrix}^{-1} \begin{pmatrix} C & CA & CA^2 \cdots CA^{s-1} \end{pmatrix} \tag{2.22}$$

$$\Gamma^{+}_{\mathbf{1}:\boldsymbol{s}} = \left(\frac{1}{CC \quad +CACA+ \quad \cdots + CA^{s-1}CA^{s-1}} \right) \begin{pmatrix} C & CA & CA^2 \cdots CA^{s-1} \end{pmatrix} \tag{2.23}$$

Through the computations presented in Equations *(2.20)* through *(2.23)*, the Moore Penrose Pseudo Inverse is going to yield a 1xs matrix, having 1 row and s columns. Hence, we obtain the controllability matrix as follows:

$$\boldsymbol{\Omega}_{\boldsymbol{s}+\mathbf{1}} = \Gamma^{+}_{\mathbf{1}:\boldsymbol{s}} \begin{pmatrix} 0 & \cdots & 0 & D \\ 0 & \cdots & D & CB \\ \vdots & \iddots & \vdots & \vdots \\ D & \cdots & CA^{s-3}B & CA^{s-2}B \end{pmatrix} \tag{2.24}$$

$$\boldsymbol{\Omega}_{\boldsymbol{s}+\mathbf{1}} = \left(\frac{1}{CC \quad +CACA+ \quad \cdots + CA^{s-1}CA^{s-1}} \right) \begin{pmatrix} C & CA & CA^2 \cdots CA^{s-1} \end{pmatrix} \begin{pmatrix} 0 & \cdots & 0 & D \\ 0 & \cdots & D & CB \\ \vdots & \iddots & \vdots & \vdots \\ D & \cdots & CA^{s-3}B & CA^{s-2}B \end{pmatrix} \tag{2.25}$$

Through the above computations, the controllability matrix will result in a 1xs

matrix, having 1 row and s columns. Therefore, now that the controllability matrix is known, the Hankel matrix can be computed using Equation *(2.15)* to yield an sxs matrix. From it, the respective Toeplitz matrix can be obtained through the property that relates it to the respective Hankel matrix. In that, a Toeplitz matrix has constant entries along the forward diagonals, and a Hankel matrix is one in which entries are constant along the reverse diagonals [12]. The controllability matrix for this system is given as follows:

$$\boldsymbol{\Omega_{s+1}} = \begin{pmatrix} B & AB & A^2B \cdots A^{s-1}B \end{pmatrix} \tag{2.26}$$

2.6 – State Sequence, Output, Derivation of Estimated Output Function

For the system being considered, the state sequence is defined to be:

$$\boldsymbol{X} = \begin{pmatrix} x(1) & x(2) & \cdots x(L-s+1) \end{pmatrix} \tag{2.27}$$

The system output, computed as a function of the matrices presented and derived in the preceding sections is as follows:

$$\boldsymbol{Y_s} = \boldsymbol{O_s X} + \boldsymbol{T_{u,s} U_s} + \boldsymbol{T_k E_s} \tag{2.28}$$

The actual output $\boldsymbol{Y_s}$ is represented as follows in *(2.29):*

$$\boldsymbol{Y_s} = \begin{pmatrix} C \\ CA \\ \vdots \\ CA^{s-1} \end{pmatrix} \begin{pmatrix} x(1) & x(2) & \cdots & x(L-s+1) \end{pmatrix}$$

$$+\begin{pmatrix} D & 0 & \cdots & 0 \\ CB & D & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{s-2}B & CA^{s-3}B & \cdots & D \end{pmatrix}\begin{pmatrix} u(1) & u(2) & \cdots & u(L-s+1) \\ u(2) & u(3) & \cdots & u(L-s+2) \\ \vdots & \vdots & \ddots & \vdots \\ u(s) & u(s+1) & \cdots & u(L) \end{pmatrix}$$

$$+\begin{pmatrix} 1 & 0 & \cdots & 0 \\ CK & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{s-2}K & CA^{s-3}K & \cdots & 1 \end{pmatrix}\begin{pmatrix} e(1) & e(2) & \cdots & e(L-s+1) \\ e(2) & e(3) & \cdots & e(L-s+2) \\ \vdots & \vdots & \ddots & \vdots \\ e(s) & e(s+1) & \cdots & e(L) \end{pmatrix} \qquad (\mathbf{2.29})$$

Equation *(2.28)* represents the fundamental structure for subspace identification utilizing the extended observability matrix form. Through this structure, it is possible to determine the columns of the observability matrix given in *(2.14)*, as well as the state sequence given in*(2.27)*. In order to estimate the output, the innovation term is considered to be zero, thereby yielding the following equation:

$$\widehat{\boldsymbol{Y}}_{\boldsymbol{s}} = \boldsymbol{O}_{\boldsymbol{s}}\boldsymbol{X} + \boldsymbol{T}_{\boldsymbol{u},\boldsymbol{s}}\boldsymbol{U}_{\boldsymbol{s}} \qquad (\mathbf{2.30})$$

In order to derive the estimated output, the following steps are taken:

$$\widehat{\boldsymbol{Y}}_{\boldsymbol{s}} = \begin{pmatrix} C \\ CA \\ \vdots \\ CA^{s-1} \end{pmatrix}\begin{pmatrix} x(1) & x(2) & \cdots & x(L-s+1) \end{pmatrix}$$

$$+\begin{pmatrix} D & 0 & \cdots & 0 \\ CB & D & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{s-2}B & CA^{s-3}B & \cdots & D \end{pmatrix}\begin{pmatrix} u(1) & u(2) & \cdots & u(L-s+1) \\ u(2) & u(3) & \cdots & u(L-s+2) \\ \vdots & \vdots & \ddots & \vdots \\ u(s) & u(s+1) & \cdots & u(L) \end{pmatrix} \qquad (\mathbf{2.31})$$

$$\widehat{\boldsymbol{Y}}_{\boldsymbol{s}} = \begin{pmatrix} Cx(1) & Cx(2) & \cdots & Cx(L-s+1) \\ CAx(1) & CAx(2) & \cdots & CAx(L-s+1) \\ \vdots & \vdots & \ddots & \vdots \\ CA^{s-1}x(1) & CA^{s-1}x(2) & \cdots & CA^{s-1}x(L-s+1) \end{pmatrix}$$

$$+ \begin{pmatrix} Du(1) & Du(2) & \cdots & Du(L-s+1) \\ CBu(1)+Du(2) & CBu(2)+Du(3) & \cdots & CBu(L-s+1)+Du(L-s+2) \\ \vdots & \vdots & \ddots & \vdots \\ \begin{matrix} CA^{s-2}Bu(1) \\ +CA^{s-3}Bu(2) \\ +\cdots+Du(s) \end{matrix} & \begin{matrix} CA^{s-2}Bu(2) \\ +CA^{s-3}Bu(3) \\ +\cdots+Du(s+1) \end{matrix} & \cdots & \begin{matrix} CA^{s-2}Bu(L-s+1) \\ +CA^{s-3}Bu(L-s+2) \\ +\cdots+Du(L) \end{matrix} \end{pmatrix} \tag{2.32}$$

2.7 – Validation of Estimated Model

In order to validate the estimated output, the following approaches may be used [14]:

- Compare the estimated output to the actual output [14]. It is known that the difference between the estimated output and actual output would be the innovation term E_s, as outlined in Equations *(2.28)* and *(2.30)*.
- Analysis of autocorrelation and cross-correlation of residuals with input.
- Analysis of model response in terms of impulse, step, or frequency response plots [14].
- Considering plots of poles and zeros of linear parametric model [14].
- Comparison between non-parametric and parametric models [14]. Non-parametric models would include impulse, step, and frequency response models; whereas parametric models would include linear polynomial models, state-space models, and non-linear parametric models [14].

- Comparison between models through Akaike Information Criterion, or Akaike Final Prediction Error [14].
- Making linear and non-linear plots for Hammerstein-Wiener and non-linear ARX models [14].

In order to validate the model, the block diagram representation given for a linear time-invariant system is used, as shown in Figure-2.1 [15]:

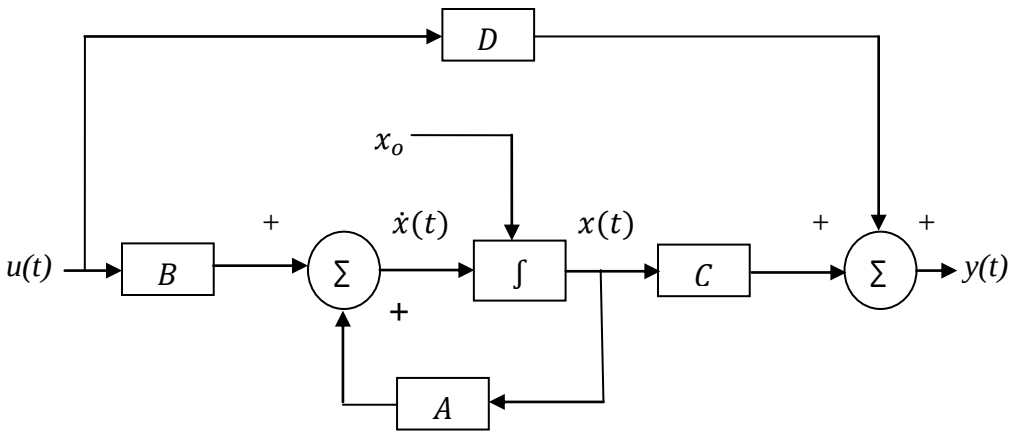

Figure-2.1: State Equation Block Diagram Representation [15]

For the basis system shown in Equations *(2.1)* and *(2.2)*, the block diagram representation is derived to be as shown in Figure-2.2, to validate the result for estimation of $\widehat{y_s}$.

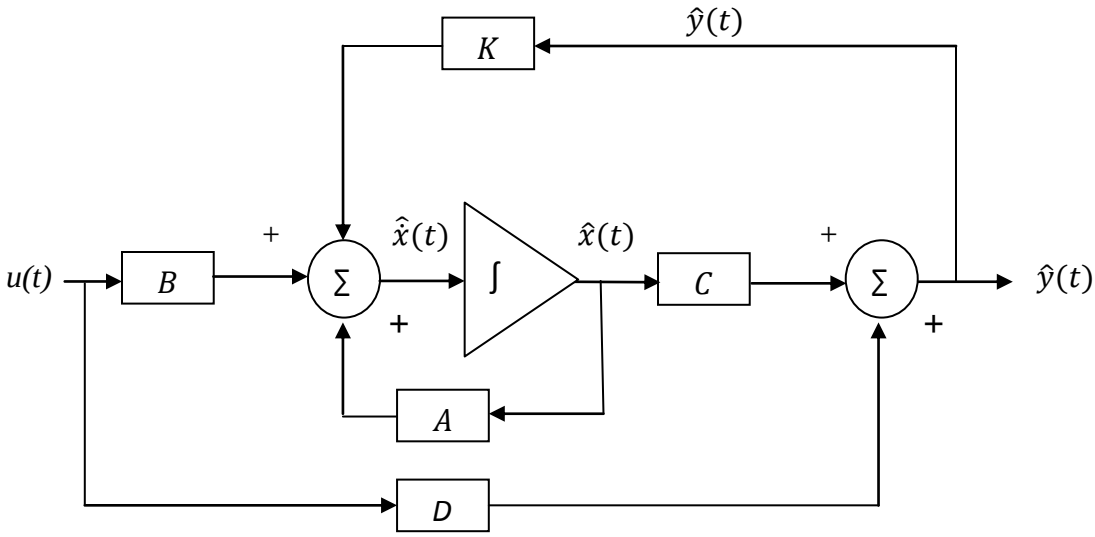

Figure-2.2: Block Diagram Representation for the system in Equations *(2.1)* & *(2.2)* in continuous time

2.8 – Rank & Error Squared Minimization

In order to minimize the rank, Equation *(2.30)* is used by evaluating rank of $O_s X$ term, which is the same as rank of $(\hat{Y}_s - T_{u,s}U_s)$ term:

$$\min_{\hat{\boldsymbol{Y}}_s \epsilon \boldsymbol{\pi}_p, \boldsymbol{T}_{u,s} \epsilon \boldsymbol{T}_{p,m}, \boldsymbol{T}_{y,s} \epsilon \boldsymbol{T}_{p,m}} \boldsymbol{rank}\,(\hat{\boldsymbol{Y}}_s - \boldsymbol{T}_{u,s}\boldsymbol{U}_s) \qquad (\mathbf{2.33})$$

In the above equation, $T_{p,m}$ is a lower triangular block-Toeplitz matrix, meaning that all elements in the matrix above the main diagonal are zero. π_p is a block-Hankel matrix having p column vectors.

To obtain the error squared, the matrix obtained through computing the difference between the actual output and the estimated output is multiplied by the transpose matrix obtained from computing the transpose of the difference between the actual output and the estimated output. The expectation operator is applied on the resulting product to yield

the mean value. Lastly, the minimum is computed to get the smallest value of the set.

$$\min \boldsymbol{E}[(\boldsymbol{y}(\boldsymbol{k}) - \widehat{\boldsymbol{y}}(\boldsymbol{k}))(\boldsymbol{y}(\boldsymbol{k}) - \widehat{\boldsymbol{y}}(\boldsymbol{k}))^{\boldsymbol{T}}] \tag{2.34}$$

2.9 – Null Space

Consider *T* as a linear transformation of $\mathbb{R}^n$ [16]. The nullspace of *T* is represented as the set of all vectors *X* such that *T(X)*=0 [16]. In Matlab™, the 'null' command provides an orthonormal basis for null space of the respective matrix being considered [17].

A matrix $L^{\perp}$ is defined to be the orthogonal complement of *L* [15]. It is considered that the columns of $L^{\perp}$ span the null space of U_s. Therefore, upon multiplication, the $T_{u,s}U_s$ term goes to zero. The resulting equation after the multiplication is as follows:

$$\boldsymbol{Y_s L^{\perp} = O_s X L^{\perp}} \tag{2.35}$$

2.10 – Nuclear Norm System Identification

In order to obtain a solution to the system by utilizing nuclear norm minimization, a technique presented by Liu and Vandenberghe in the paper titled Interior-point method for nuclear norm approximation with application to system identification is used as shown below [18]:

$$\underset{\boldsymbol{y_m}}{\boldsymbol{minimize}} \sum_{\boldsymbol{k=1}}^{\boldsymbol{L}} \{\boldsymbol{y}(\boldsymbol{k}) - \boldsymbol{y_m}(\boldsymbol{k})\}^2 + \boldsymbol{\lambda}\, ||\widehat{\boldsymbol{Y}}_{\boldsymbol{s}} \boldsymbol{L}^{\perp}||_* \tag{2.36}$$

In the above equation, λ is the Lagrange Multiplier.

2.11 – Proposed Iterative ADMM Algorithm

The first approach considered is the one presented in the thesis titled "System Identification via Nuclear Norm Regularization" by Harshad Deshmane [3]. In this approach, a deterministic discrete-time linear time-invariant state-space model of the following form is considered as the basis model [3]:

$$\boldsymbol{x(k+1) = Ax(k) + Bu(k)} \tag{2.37}$$

$$\boldsymbol{y(k) = Cx(k) + Du(k)} \tag{2.38}$$

It is evident from Equations *(2.37)* and *(2.38)* that the stochastic model containing process and measurement noise as given in Equations *(2.1)* and *(2.2)*is altered to be a deterministic model as shown in Equations *(2.37)* and *(2.38)* in order to implement the Iterative ADMM method [3]. This approach is outlined in Harshad Deshmane's thesis [3].

From this thesis, derivations are made to prove the different steps of the Iterative ADMM algorithm [3]. Firstly, the minimization problem is shown to lead to a local minimum point [21, 22]. From Deshman's thesis, H(y) is equal to Hankel of output matrix Y [3]. Therefore, in all equations, formulae, and matrices that are presented from here on forward; each instance of H(y) is changed to Y [3]. Using the Lagrange Multiplier Theorem, the minimization problem is given as [21, 22]:

$$\boldsymbol{\min f(\bar{x}) \;\; subjected\ to\ YU^{\perp} - Z = 0} \tag{2.39}$$

In Equation *(2.39)*, U is represented as follows [3]:

$$U = \begin{pmatrix} u(0) & u(1) & u(2) & \cdots & u(N-r) \\ u(1) & u(2) & u(3) & \cdots & u(N-r+1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u(r-1) & u(r) & u(r+1) & \cdots & u(N-1) \end{pmatrix} \tag{2.40}$$

In Equation *(2.39)*, Y: $\mathbb{R}^{\boldsymbol{pxn}} \rightarrow \mathbb{R}^{\boldsymbol{prxN-r+1}}$; and Z is represented as a substitution of $Z = YU^{\perp}$ [3].

Let $v = YU^{\perp} - Z$, and let $\bar{x}^*$ be a local minimum point [21, 22]. Then there exist a Lagrange Multiplier Λ such that [21, 22]:

$$\frac{\boldsymbol{\partial f}(\overline{\boldsymbol{x}}^*)}{\boldsymbol{\partial x}} + \Lambda \frac{\boldsymbol{\partial}(\boldsymbol{YU}^{\perp} - \boldsymbol{Z})}{\boldsymbol{\partial y}} = \mathbf{0} \tag{2.41}$$

A Lagrange function is defined as [21, 22]:

$$\boldsymbol{L}(\overline{\boldsymbol{x}}, \Lambda) = \boldsymbol{f}(\overline{\boldsymbol{x}}) + \Lambda(\boldsymbol{Y'U}^{\perp} - \boldsymbol{Z'}) \tag{2.42}$$

Equation *(2.41)* leads to the following results [21, 22]:

$$\overline{\boldsymbol{\nabla}}\boldsymbol{L}(\overline{\boldsymbol{x}}^*, \Lambda) = \mathbf{0} \tag{2.43}$$

$$\frac{\boldsymbol{\partial}\overline{\boldsymbol{L}}(\overline{\boldsymbol{x}}^*, \Lambda)}{\boldsymbol{\partial}\overline{\boldsymbol{x}}} = \mathbf{0} \tag{2.44}$$

$$\frac{\boldsymbol{\partial L}(\overline{\boldsymbol{x}}^*, \Lambda)}{\boldsymbol{\partial}\Lambda} = \boldsymbol{v} = \mathbf{0} \tag{2.45}$$

Equations *(2.43)*, *(2.44)* and *(2.45)* represent that L is stationary with respect to variables $\bar{x}$ and Λ [21, 22]. This can be regarded as an unconstrained function $f(\bar{x}, \Lambda)$ [21, 22]. All points that satisfy conditions of the theorem presented in Equations *(2.39)* through *(2.45)* are candidate minimum points, whereas those that don't satisfy these conditions are not local minimum points [21, 22].

Through use of Lagrange multipliers, the minimization problem can be solved with the following necessary conditions [21, 22]:

$$\frac{\partial \mathcal{L}_\rho}{\partial y} = \frac{\partial\left(\frac{1}{2}\left|\left|y-b\right|\right|^2\right)}{\partial y} + \frac{\partial\left(tr\left(\Lambda^T(YU^\perp - Z)\right)\right)}{\partial y} + \frac{\partial\left(\frac{\rho}{2}\left|\left|YU^\perp - Z\right|\right|_F^2\right)}{\partial y} = 0 \tag{2.46}$$

$$\frac{\partial \mathcal{L}_\rho}{\partial z} = \frac{\partial(\gamma\left|\left|Z\right|\right|_*)}{\partial z} + \frac{\partial\left(tr\left(\Lambda^T(YU^\perp - Z)\right)\right)}{\partial z} + \frac{\partial\left(\frac{\rho}{2}\left|\left|YU^\perp - Z\right|\right|_F^2\right)}{\partial z} = 0 \tag{2.47}$$

$$\frac{\partial \mathcal{L}_\rho}{\partial \Lambda} = \frac{\partial\left(tr\left(\Lambda^T(YU^\perp - Z)\right)\right)}{\partial \Lambda} = 0 \tag{2.48}$$

The three non-linear equations cannot be solved for y^*, z^*, and Λ^*, since the respective dimensions are too large for that purpose.

2.12 - Detailed Iterative ADMM Algorithm

Following the proof from the preceding section for the minimization problem given in the thesis titled "System Identification via Nuclear Norm Regularization" by Harshad Deshmane, the Iterative ADMM algorithm from that thesis is now discussed [3].

The algorithm is divided into three steps as follows [3]:

Initialize values of y^0, Z^0, Λ^0 [3]. Choose ρ to be positive [3].

Step1:

Let $y^{(k+1,0)} = y^k$ and $y^{(k+1,-1)} = y^k$ (if applicable) (k and l are outer & inner line search iteration counters respectively)

In the case that there is no fixed termination criteria, compute the following [3]:

$$y^{(k+1,l+1)} = y^{(k+1,l)} - \alpha^{(l)}(D^{(l)})^{-1}\nabla_y \mathcal{L}_\rho\left(y^{k+1,l}\right) \tag{2.49}$$

In Equation *(2.49)*, $\alpha^{(l)}$ is calculated based on the Barzilai-Borwein method:

$$\boldsymbol{\alpha}^{(l)} = \frac{< \boldsymbol{x}^{l} - \boldsymbol{x}^{l-1}, \ \boldsymbol{\nabla f}^{l} - \boldsymbol{\nabla f}^{l-1} >}{< \boldsymbol{\nabla f}^{l} - \boldsymbol{\nabla f}^{l-1}, \ \boldsymbol{\nabla f}^{l} - \boldsymbol{\nabla f}^{l-1} >} \tag{2.50}$$

where $< \boldsymbol{x}^{l} - \boldsymbol{x}^{l-1}, \ \boldsymbol{\nabla f}^{l} - \boldsymbol{\nabla f}^{l-1} >$ is dot product of $\boldsymbol{x}^{l} - \boldsymbol{x}^{l-1}$ and $\boldsymbol{\nabla f}^{l} - \boldsymbol{\nabla f}^{l-1}$,

$\boldsymbol{x}^{l}$ represents current output, $\boldsymbol{x}^{l-1}$represents output from previous iteration,

$\boldsymbol{\nabla f}^{l}$ represents current gradient, $\boldsymbol{\nabla f}^{l-1}$represents gradient from previous iteration

$$\boldsymbol{D}^{(l)} = \boldsymbol{identity\ matrix} \tag{2.51}$$

$\mathcal{L}_{\rho}(y, Z, \Lambda)$ can be derived from the augmented Lagrangian presented in Deshmane's thesis as follows [3]:

$$\boldsymbol{\mathcal{L}_{\rho}}(\boldsymbol{y}, \boldsymbol{Z}, \Lambda) = \boldsymbol{\gamma}||\boldsymbol{Z}||_{*} + \frac{\boldsymbol{1}}{\boldsymbol{2}}\left|\left|\boldsymbol{y} - \boldsymbol{b}\right|\right|^{\boldsymbol{2}} + \boldsymbol{tr}\left(\Lambda^{\boldsymbol{T}}(\boldsymbol{Y}\boldsymbol{U}^{\perp} - \boldsymbol{Z})\right) + \frac{\boldsymbol{\rho}}{\boldsymbol{2}}\left|\left|\boldsymbol{Y}\boldsymbol{U}^{\perp} - \boldsymbol{Z}\right|\right|_{\boldsymbol{F}}^{\boldsymbol{2}} \tag{2.52}$$

$\nabla_{y}\ \mathcal{L}_{\rho}(y^{k+1,l})$ can be taken from the gradient of the augemented Lagrangian presented in Deshmane's thesis as follows [3]:

$$\boldsymbol{\nabla_{y}\mathcal{L}_{\rho}}\left(\boldsymbol{y}^{\boldsymbol{k+1,l}}\right) = \boldsymbol{\rho H}^{*}\left(\left(\boldsymbol{Y}\left(\boldsymbol{y}^{\boldsymbol{k+1,l}}\right)\boldsymbol{U}^{\perp} - \left(\boldsymbol{Z}^{\boldsymbol{k}} - (\boldsymbol{\rho}^{-\boldsymbol{1}})\Lambda^{\boldsymbol{k}}\right)\right)(\boldsymbol{U}^{\perp})^{\boldsymbol{T}}\right) + \left(\boldsymbol{y}^{\boldsymbol{k+1,l}} - \boldsymbol{b}\right) \tag{2.53}$$

Step2:

Compute the singular-value-decomposition through the following steps [3]:

$$\boldsymbol{W}\sum\boldsymbol{V}^{\boldsymbol{T}} = \boldsymbol{Y}\left(\boldsymbol{y}^{\boldsymbol{k+1}}\right)\boldsymbol{U}^{\perp} + (\boldsymbol{\rho})^{-\boldsymbol{1}}\Lambda^{\boldsymbol{k}} \tag{2.54}$$

For incrementing Z, do the following [3]:

$$\boldsymbol{Z}^{\boldsymbol{k+1}} = \boldsymbol{W}\ \mathbf{max}\{\sum - \frac{\boldsymbol{\gamma}}{\boldsymbol{\rho}}\boldsymbol{I}, \boldsymbol{0}\}\boldsymbol{V}^{\boldsymbol{T}} \tag{2.55}$$

In Equation *(2.54)* and *(2.55)*, variables are defined as follows [3]:

U: Hankel of input matrix [3]

W, $\sum$, V: can be found from svd command in Matlab™ through singular-value-decomposition of right-side of Equation *(2.54)* [3]

$H(y^{k+1})$: block Hankel matrix is given as H(y) and is substituted with Y [3]

ρ: positive scalar, kept constant at 1 [3]

Λ: Lagrange multiplier or dual variable [3]

Z: $Z = YU^{\perp}$ is used as substitution in minimization problem [3]

I: identity matrix [3]

γ**:** regularization parameter that is varied to compute the results of the Iterative ADMM algorithm for various values of this parameter, in order to arrive at a value that gives best estimation results for $\boldsymbol{y}$ having least error when compared to actual output data [3]

*Step3***:**

The Lagrange multiplier is incremented as follows [3]:

$$\Lambda^{\boldsymbol{k+1}} = \Lambda^{\boldsymbol{k}} + \boldsymbol{\rho}(\boldsymbol{Y}(\boldsymbol{y}^{\boldsymbol{k+1}})\boldsymbol{U}^{\perp} - \boldsymbol{Z}^{\boldsymbol{k+1}}) \tag{2.56}$$

In Equation *(2.56)*, variables are defined as follows [3]:

Λ: Lagrange multiplier or dual variable [3]

ρ: positive scalar [3]

$H(y^{k+1})$: block Hankel matrix is given as H(y) and is substituted with Y [3]

Z^{k+1}: $Z = Y(y^{k+1})U^{\perp}$ is used as substitution in minimization problem [3]

2.13 – Tabu Search Optimization

Aside from the Iterative ADMM method discussed in Sections 2.11 and 2.12, Tabu Searh Optimization is used as the second method for minimization and System Identification. A deterministic system, as represented in Equations *(2.37)* and *(2.38)* is considered as the basis system, in order to apply Tabu Search Optimization and Iterative ADMM, and thereby provide for possibility of analyzing and comparing performance of

both methods.

Tabu Search aims at finding the best possible solution through iteratively searching and moving from a current solution to a better solution [19]. Through the search, paths taken are stored in the tabu list [19]. With this list, directions for new moves are determined that would yield a better local optimum solution, in turn leading to the global optimum solution [19]. Tabu Search can be broken down into the following steps [19]:

1) Choose a random initial solution *So* from a search area having radius *R* [19]. Solution *So* would be the current local optimum solution [19].
2) By random choice, *N* new solutions are selected through moving the current solution around *So* in the search space. Set $S_1(r)$ would be comprised of *N* solutions [19].
3) Utilize the objective function to evaluate the cost value of each member in $S_1(r)$ [19]. From these, choose the best solution which would be the one with minimum cost [19, 20].
4) Compare the cost of the best solution obtained in step (3) to the cost of the solution obtained in step (1) [19, 20].
5) If the cost of the best solution from step (3) is greater than or equal to the cost of the solution from step(1), then leave *So* unchanged [19, 20]. Else, if the cost of the solution from step (3) is less than the cost of the solution from step (1), store the

best solution from step(3) in *So* [19, 20]. The solution with the minimum cost is the best solution [19, 20].

6) Store the best solution from step (5) in the tabu list [19, 20].
7) Repeat the process from step (2) through step (6) until the maximum number of iterations are reached [19, 20].

2.14 – Eigen-System Realization

Eigen-System Realization Algorithm (ERA) is utilized in order to identify the system using estimated output from the two estimation techniques discussed in the preceding Sections of 2.12 and 2.13 [23]. This algorithm aims to use the discrete impulse response h for constructing a square block Hankel matrix $H(0)$ as shown in *(2.57)* [23, 24]. A second square block Hankel matrix $H(1)$ is obtained by shifting every column in the original block hankel matrix $H(0)$ by one unit to the left [23, 24]. Singular-value-decomposition is performed on $H(0)$, the result of which are the matrices P_n, D_n, and Q_n^T [23]. From the results of singular-value-decomposition, a matrix $D_{nn} = D_n(1{:}n, 1{:}n)$ is obtained, where n represents the order of the system; that is for a second-order system, n is equal to 2 [23]. Observability and Controllability matrices are calculated through using Equations *(2.58)*, and *(2.59)* shown below with respect to order of system being n [23]. System matrices A, B, C, and D are calculated through use of Equations *(2.60)* through *(2.63)* [23]. Once the system matrices are known, it is possible to calculate a discrete-time or a continuous-time transfer function using Matlab™ [23].

$$H(0)=\begin{pmatrix} h(1) & h(2) & \cdots & h(n) \\ h(2) & h(3) & \cdots & h(n+1) \\ \vdots & \vdots & \iddots & \vdots \\ h(n) & h(n+1) & \cdots & h(2n-1) \end{pmatrix} \tag{2.57}$$

$$\boldsymbol{W_o = P_n(:, 1{:}n) * (D_{nn}^{0.5})} \tag{2.58}$$

$$\boldsymbol{C_o = (D_{nn}^{0.5}) * (Q_n^T(:, 1{:}n))^T} \tag{2.59}$$

$$\boldsymbol{A = (D_{nn}^{-0.5}) * (P_n(:, 1{:}n))^T * H(1) * (Q_n^T(:, 1{:}n)) * (D_{nn}^{-0.5})} \tag{2.60}$$

$$\boldsymbol{B = C_0(:, 1)} \tag{2.61}$$

$$\boldsymbol{C = W_0(1, :)} \tag{2.62}$$

$$\boldsymbol{D = h(1)} \tag{2.63}$$

CHAPTER 3.0 – IMPLEMENTATION OF ITERATIVE ADMM METHOD

Matlab™ and Simulink™ software are used in order to programmatically implement the Iterative ADMM method discussed in the preceding chapter [3]. Steps involved in implementing this method are described in this chapter, and associated Matlab™ code can be found in Appendix A [3]. For the purpose of explaining the procedure, the benchmark system 1 shown in equation (3.1) will be utilized [25].

3.1 - Obtaining Discrete-Time Transfer Function

The first step is to analyze a benchmark research paper, and obtain a system transfer function from it. If the transfer function is in discrete-time, then it is used as is. If the transfer function is in continuous-time, then it is converted to its discrete-time equivalent using Matlab™ with a sample time of 0.1 seconds. The corresponding discrete-time transfer function for system 1 is as follows [25]:

$$\boldsymbol{G(z) = \frac{0.7071z + 4.2426}{z^2 + 0.750z + 0.125}} \qquad (\mathbf{3.1})$$

3.2 – System Matrices & Eigen-Values of Discrete-Time Transfer Function

The second step is to obtain the system matrices A, B, C, D, and the eigen-values of A matrix for the discrete-time system transfer function using Matlab™. Steps outlined in Sections 3.1 and 3.2 are achieved through Matlab™ code for Main Program shown in Appendix A Section A1. A screen-shot of Matlab™ is shown in Figure-3.1 below. Part of the script of Main.m that performs steps in Sections 3.1 and 3.2 is shown in the center,

part of the output of running the script is shown in the Command Window at the bottom, and resulting variables including discrete-time transfer function model Hd, system matrices A, B, C, D, and eigen-values of A matrix A_eig are shown in the Workspace section in the bottom left.

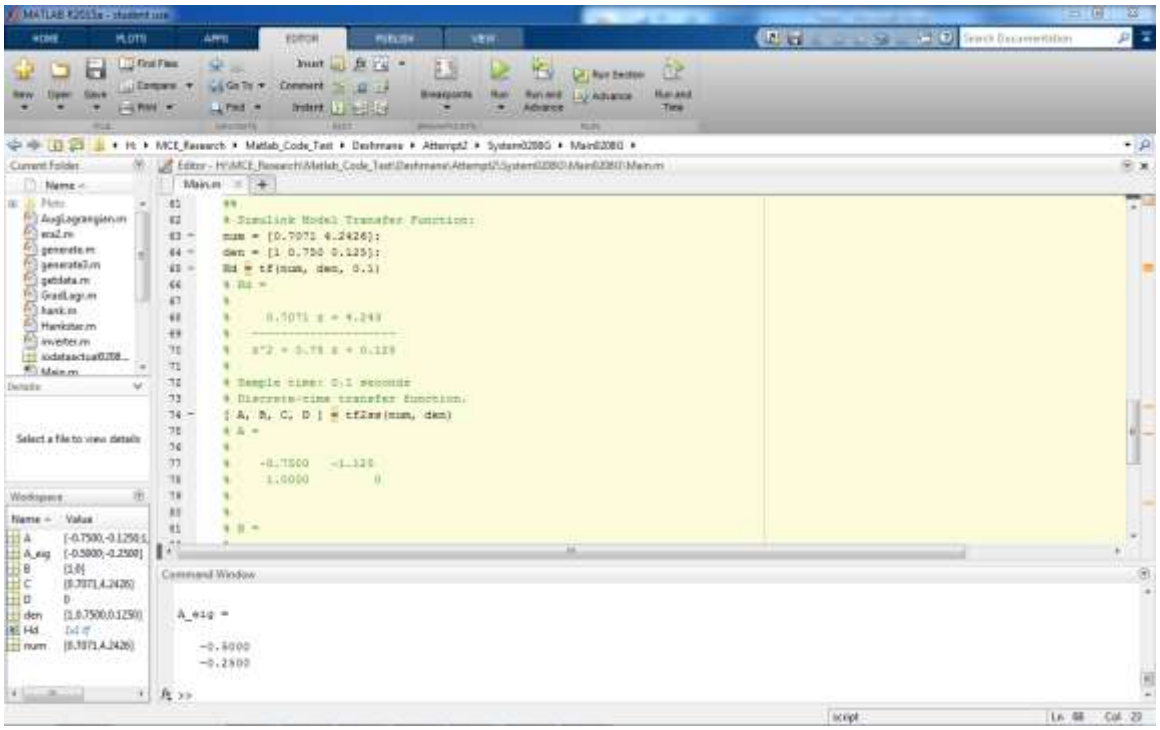

Figure-3.1: Matlab™ Window showing Main.m

3.3 – Input, Output, & Measured Output Data Matrices from Simulink™

The discrete-time transfer function is described as a block in Simulink™. Due to not having measured data readily available, the measured output is obtained by using a transfer function that is a slight modification (coefficients for respective powers of z are close but not exactly equal) of the original discrete-time transfer function, to simulate the measured output data that is close to but not exactly equal to the actual ouput. A pseudo-random signal with sample time of 0.1 seconds is used as the input. Upon running the

Simulink™ model, the input, output, and measured output data matrices, all being 1001x1 in size are obtained in Matlab™ workspace. These are saved with appropriate naming convention in the current directory, eg: Input0208 and Output0208 are saved as 'iodataactual0208.mat,' and the measured output is saved as 'OutputMeasured0208.mat,' where '0208' is the system identifier for the benchmark system being considered.

As shown in Figure-3.2 below, the Band-Limited White Noise and Sign blocks comprise the pseudo-random input with sample time of 0.1 seconds. The system transfer function is seen in the block titled Model (Actual Output without noise – Discretized). Transfer function used for generating the measured data close to actual output data is shown in block titled Model (Estimated by slightly modifying original transfer function above, used to generate measured output that is close to but not exactly equal to the actual output). Block Output0208G performs the function of placing actual output of the system in Matlab™ workspace to be used by programs contained therein, and similarly, block for OutputMeasured0208G and block Input0208G place measured output and input of the system in the workspace. The button with a green arrow symbol in the top-left can be clicked to run the model, and the scope seen in the center right titled Output can be double-clicked to view graphical representation of the behavior of all four models versus time.

Estimated transfer functions through Iterative ADMM and Tabu Search are shown in the last two blocks in the center respectively. The block OutputEstimated0208G places

the estimated output in the Matlab™ workspace. Details of arriving at the estimated transfer functions using Iterative ADMM and Tabu Search will be explained in more detail in the following Sections.

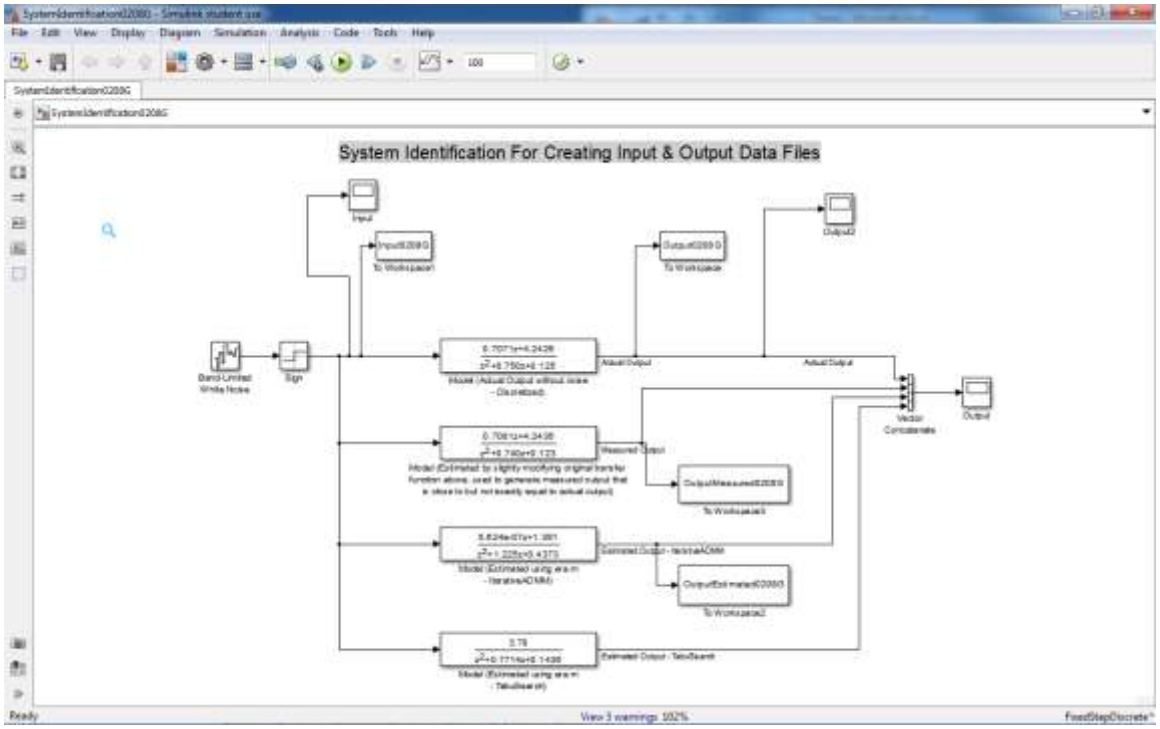

Figure-3.2: Simulink™ Window showing Block model for System0208G

3.4 – Initializations

Initialization operations are performed including variable declarations, and construction of appropriate matrices to be used in inner and outer line-search routines. Matlab™ code for Get Data Function and Main Program shown in Appendix A Sections A7 and A1 respectively perform these functions, which are as follows:

- Output data matrix is initialized as global variable y of size 1001x1.
- Input data matrix is initialized as global variable u of size 1001x1.
- Measured data matrix is initialized as global variable y_hat of size 1001x1.

- Number of rows of input data matrix is stored as m.
- Number of rows of output data matrix is stored as p.
- Number of columns of input and output data matrices, both being equal, is stored as N.
- Block row dimension for construction of Hankel matrices is initialized as r=2, to construct Hankel matrices having two rows.
- Measured data from y_hat is stored in variable b.
- Positive scalar rho is initialized with value of 1.
- Uperp is calculated by first building Hankel matrix from input matrix u, and taking null-space of the result.
- Regularization parameter Gamma is initialized with value of 1.
- Maximum possible number of iterations for inner-line-search is initialized as 20.
- Maximum possible number of iterations for outer-line-search is initialized as 20.
- Stopping tolerance for inner-line-search is initialized as 0.001.
- Y is built as hankel matrix of output data matrix.
- Z is initialized as a matrix of zeros with proper dimensions of 2x998 for the algorithm and the system.
- lagrangemultiplier is initialized as a matrix of zeros with proper dimensions of

2x998 for the algorithm and the system.

3.5 – Inner & Outer Line Search Routine

Following steps of preceding Sections 3.1 – 3.4, a basis has been established to execute inner and outer line-search routines for the system. This is done through the Matlab™ code shown in Appendix A Section A8. This function is designed to take in as inputs the following: output data matrix y as yinit, Z matrix, and lagrangemultiplier matrix. Outputs of the function could be any of the calculated parameters from within the function. The collective functioning of the inner and outer loops is described in Figure-3.3 flow diagram on the following page.

A factor of 25 is multiplied into the ratio of Gamma/rho in the Matlab™ code show in Appendix A Section A8. Having rho fixed at 1, this factor could be regarded as the value of Gamma that would yield a reasonable estimate of the original system. Value of 25 for this factor was determined through executing the inner and outer line-search function for various values, and observing how close the estimated output ynew was to the actual output yinit for each value of this factor by calculating estimation error for each value of gamma [3]. This value of 25 is not the optimal value of Gamma for this system, as it doesn't yield perfect estimation results, and further investigation in terms of the range of values of Gamma could be done to determine Gamma that would give optimal estimation results [3].

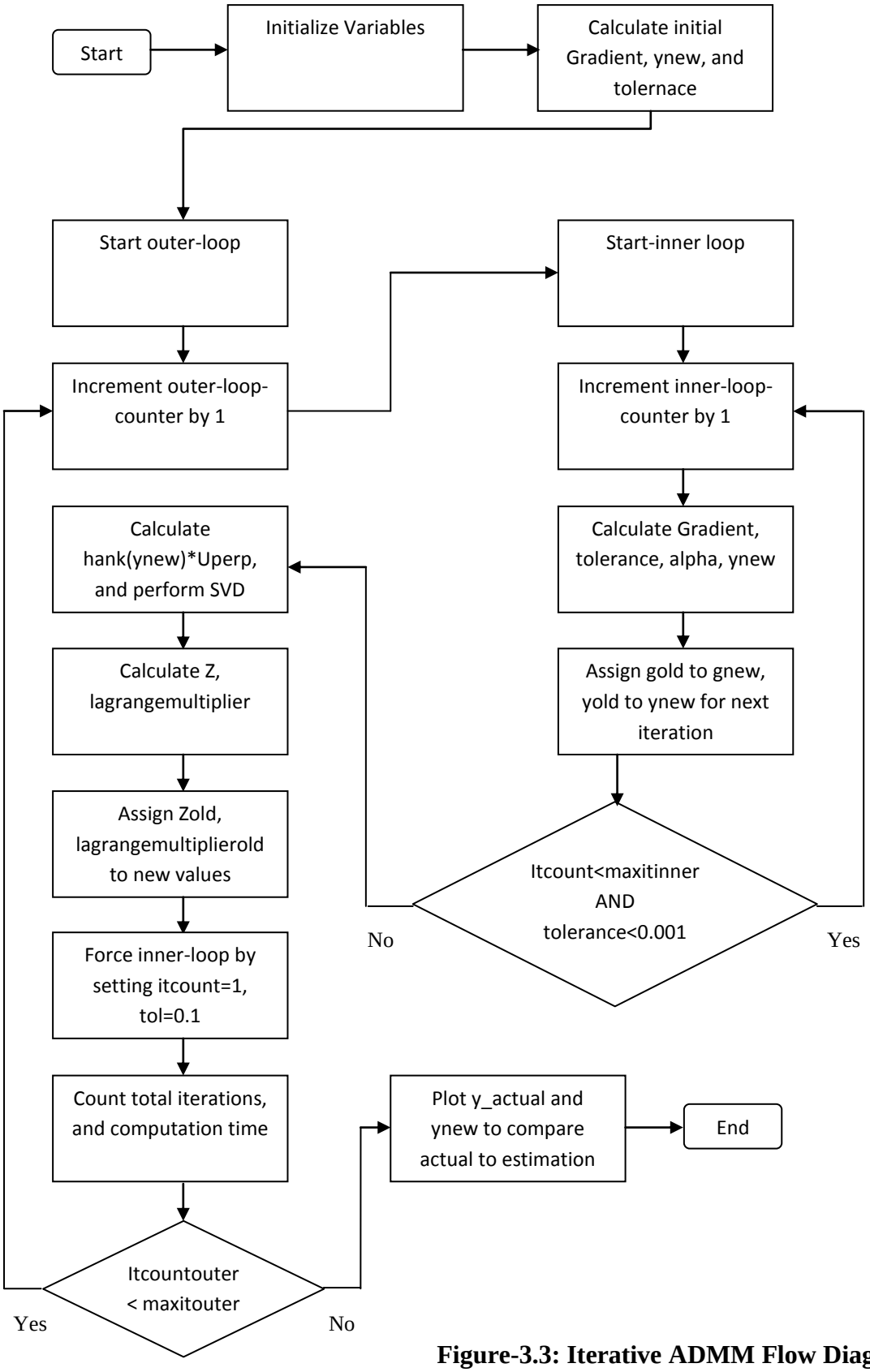

Figure-3.3: Iterative ADMM Flow Diagram

3.6 – Eigen System Realization for System Identification

Following execution of inner and outer line-search routines, an estimation result has been obtained in the form of a matrix of the same dimensions as the original output data matrix of the system simulated in Simulink™. Using this estimated result, it is desired to identify the estimated system by finding associated system matrices A, B, C, and D.

Eigen-system realization is used for identification of estimated system. First step is to obtain a discrete-time impulse response. Matlab™ code for obtaining a discrete-time impulse response using estimated output ynew and original input Input0208G is shown in Appendix A Section A9. Firstly, an iddata object is created using ynew, Input0208G, and sample time of 1 second. Second, a transfer function estimate with 2 poles, 1 zero, io-delay of 1, and sample-time of 1 second is made. Using this estimated transfer function, a discrete-time impulse response is obtained to be used in eigen-system realization algorithm.

Using the discrete-time impulse response, eigen-system realization algorithm is computed through the Matlab™ code in Appendix A Section A10. The order of the model is 2 since it is desired to estimate a system with two poles, so as to estimate a model of the same order as the original system. The number of samples to construct the Hankel matrix can be any number that is at least two units less than number of columns of the impulse response matrix. The sample time is taken as 0.1 seconds, and def is set to

1 to yield estimation in discrete-time. The developed Matlab™ code implements the procedure outlined for the eigen-system realization introduced in Section 2.14, and performs the following steps:

- Construction of hankel matrix from discrete-time impulse response.
- Construction of second hankel matrix by shifting each column of original hankel matrix by one unit to the left.
- Singular-Value-Decomposition, and computation of observability and controllability matrices with respect to the model order of 2.
- Evaluation of system matrices A, B, C, and D.
- Determination of discrete-time transfer function, and eigen-values of A matrix, to compare to the transfer function and eigen-values of original system.

CHAPTER 4.0 – IMPLEMENTATION OF TABU SEARH METHOD

Matlab™ software was used in order to programmatically implement the Tabu Search method discussed in the Chapter 2.0 [3]. Steps involved in implementing this method are described in this chapter, and the associated Matlab™ code can be found in Appendix B [3]. For the purpose of explaining the procedure, benchmark system 1 given in equation (3.1) will be utilized [25].

4.1 - Initializations

Initialization operations are performed including variable declarations, and construction of appropriate matrices to be used in searching for best solution with minimum cost. The developed Matlab™ code for Simple Tabu Search Main Program is shown in Appendix B Section B1 to perform the following initializations:

- Dimension of solution nx is initialized as 1001, due to 1001 data points.
- Maximum maxx and minimum minx of search space are initialized as 100 and -100 respectively, as this is sufficient space for optimally searching and arriving at minimum solution.
- Length of tabu-list tll is initialized as 5.
- Length of promising-list pll is initialized as 5.
- Tabu ball radius is initialized as r = 2.5/100*(maxx-minx) = 62.5x10^-6
- Radius of neighborhood is initialized as r1 = r/2 = 31.25x10^-6
- Promising ball radius is initialized as r3 = r1/2 = 15.625x10^-6

- Number of neighborhoods n1 is initialized as 25.
- Number of iterations nd is initialized as 100.
- x is assigned the value of Output0208G transposed, for it to be a row-vector.

4.2 – Cost Function

Cost function to be minimized is implemented from Deshmane thesis, and is shown in Equation *(4.1)* [3].

$$\underset{\boldsymbol{y},\boldsymbol{Z}}{\boldsymbol{minimize}} \quad \boldsymbol{\gamma} \,||\, \boldsymbol{H}(\boldsymbol{y})\, \boldsymbol{U}^{\perp} \,||_{*} + \frac{\mathbf{1}}{\mathbf{2}} \,||\, \boldsymbol{y} - \boldsymbol{b} \,||^{\mathbf{2}} \qquad (\mathbf{4.1})$$

This function is programmatically implemented using Matlab™, the code for which is shown in Appendix B Section B2. For system 1, value of gamma that yields a somewhat close estimation to the original output is 25. $H(y)$ is the Hankel matrix of the output data matrix y having 2 block rows. $\boldsymbol{U}^{\perp}$is the matrix of null-space of the Hankel of input data matrix u, and b is measured output generated from Simulink™ as described in Sections 3.3 and 3.4.

4.3 – Creation of Neighbors

Creation of neighbors is a necessary step in implementation of Tabu Search algorithm. The developed Matlab™ code for creating neighbors is given in Appendix B Section B4. The code aims at computing difference between outer and inner diameters, randomizing the difference, evaluating diameter for each neighbor, and evaluating theta and xcomp. This is done for each dimension, and using the final result for xcomp,

xneighbor is computed for each neighborhood as candidate element from the neighborhood.

4.4 – Tabu Ball Verification

In order to verify that position of candidate element is contained within the Tabu ball, the Matlab™ code given for Tabu Ball Verification Function in Appendix B Section B3 is utilized. This function checks whether a given element from Tabu-list is contained within the pre-specified Tabu-ball radius. If the verification succeeds, then the flag is set to zero, and position of element is within Tabu ball. If verification doesn't succeed, the flag is set to one, and position of element is not within Tabu ball.

4.5 – Main Loop for Tabu Search Algorithm

Through sub-blocks described in Sections 4.1-4.4, it is possible to implement Tabu search algorithm for computing solution with minimum cost. This is performed in the main loop section of Matlab™ code shown in Appendix B Section B1. Following are the steps carried out in main loop:

- The main loop is executed for k ranging from 1 till 100.
- The first step is to create neighbors as described in Section 4.3, based on radius of neighborhood, number of neighborhoods, dimension of solution, and candidate elements given by x.
- Secondly, all candidate elements in the given neighborhood are selected one at a time, with their corresponding cost also being evaluated. The element with

the least cost is stored as bestelement, and its cost as minimumcost.

- Using the determined bestelemnt from the previous step, it is checked if this element is within the pre-defined Tabu ball of given radius using the Tabu Ball Verification method in Section 4.4. If bestelement is within Tabu ball, and if the cost of bestelement is greater than initially determined globalbestcost, then candidate element is moved by assigning it value of the 25th element in neighborhood. The minimum cost is set to be cost of this candidate element with respect to the considered cost function.
- It is checked if minimumcost is less than globalbestcost. If this is the case, the value of xtest is assigned to candidatex, and cost of candidate for specific cost function considered is assigned to minimum cost. If minimumcost is not greater than globalbestcost, then candidatex is assigned the value of x(k,:), where k would be the current iteration value, and cost of candidate for specific cost function considered is assigned to minimum cost.
- The search limit set for the Tabu Search algorithm is checked, in that, it is checked if the position of candidate element is within the bounds specified for searching. Maximum possible value for search space is maxx and least possible value for it is minx. If position of candidate element is greater than maxx, then the position of the element is set to maxx, and minimumcost is set to the cost of the candidate element through the respective cost function.

Similarly, if position of candidate element is lesser than minx, then the position of the element is set to minx, and minimumcost is set to the cost of the candidate element through the respective cost function.

- After the steps outlined above, the position of (k+1) element is assigned the value of candidatex. The value of mincost for the kth element is assigned to the cost of the (k+1) element for the respective cost function being considered.
- It is checked if the least cost of the kth element is less than globalbestcost computed in an earlier step in main loop. If that is the case, then globalbestcost is assigned value of mincost of kth element. Counter is incremented, and the counter value element of globalbestx is assigned the value of (k+1) element of x.
- The Tabu list is updated by moving each element one position down, and the top position is filled with (k+1) element of x.
- If the dimension of solution being considered is less than 3, then two plots are generated, first plot being shown is the position of the agent as it moves to search for the global minimum solution, and second, a plot for the minimum cost with respect to the iteration index is shown.
- The last step is to output the global best solution for the position that gives globally the most minimum cost, the value for global best cost, and computation-time for executing Tabu Search algorithm.

CHAPTER 5.0 – OPTIMIZATION METHODS COMPARED

After results have been obtained from Iterative ADMM and Tabu Search alogorithms, it is desired to analyze the results in comparison to the true system. For this purpose, twelve systems from benchmark research papers presented at the 17th IFAC Symposium on System Identification (Beijing International Convention Center, October 19-21, 2015, Beijing, China) are considered, and key metrics for each system are individually computed and compared [25 - 29]. Results for Iterative ADMM and Tabu Search optimization are presented and compared to the true system as shown in Equations *(5.2)* through *(5.37)*,Tables 5.1 through 5.12, and Figures 5.1 through 5.24. Key findings from the results are summarized as follows:

Number of Iterations:

Iterative ADMM uses more iterations than Tabu Search, due to Tabu Search method arriving at an optimal solution with minimum cost, and Iterative ADMM method not being able to arrive at fully optimal solution with minimum cost, thereby requiring more number of iterations for computation of minimal solution.

Computation Time:

Iterative ADMM is faster using less computation-time than Tabu Search.

Error:

Iterative ADMM has certain spikes of very high percent error caused due to a non-optimal value of gamma being used. This could be improved by testing the Iterative

ADMM method for a broader range of regularization parameter gamma, to arrive at the value that would yield better results [3]. Tabu Sarch, being fully optimized, is able to give zero percent error for all systems. Identification error for Iterative ADMM gives a measure of how close the estimation is to the actual output, and is calculated as follows [3]:

$$\boldsymbol{e_I = 100 \times \frac{||\hat{y} - \tilde{y}||_F}{||\tilde{y}||_F}} \qquad (\mathbf{5.1})$$

Characteristic:

Characteristic behavior of Iterative ADMM estimation is close for some systems, and not so close for others, due to optimal value of gamma not being known for all individual systems. The behavior of Tabu Search estimation result is such that it always perfectly matches to the true system due to it being fully optimized.

Eigen-Values of A Matrix:

The extracted eigen-values of A matrix of Iterative ADMM estimation are close to the true system A matrix eigen-values for some systems, but not for all systems. This is caused by a non-optimal value of gamma being used, thereby decreasing the accuracy of discrete-time impulse response used for eigen-system realization to obtain system matrices. Eigen-values of the A matrix of Tabu Search estimation are closer - but not exactly equal for all systems - to those of the true system. Slight inaccuracy in this case is caused by the chosen sample time and number of samples used for obtaining discrete-time impulse response and eigen-system realization for obtaining system matrices.

Recovering Wiener-Hammerstein Nonlinear State-Space Models Using Liner Algebra (P. Dreesen, M. Ishteva, and J. Schoukens, 2015)-System 1 [25]

$$G\_true(z) = \frac{0.7071z+4.2426}{z^2+0.750z+0.125} \quad (5.2)$$

$$G\_IterADMM(z) = \frac{1.717}{z^2+0.7677z+6.903e-07} \quad (5.3)$$

$$G\_TabuSearch(z) = \frac{3.748}{z^2+0.7714z+0.1436} \quad (5.4)$$

Table-5.1: Iterative ADMM & Tabu Search Results – System 1 [25]

	Iterative ADMM	Tabu Search	True System
# of iterations	151	100	-
Computation Time	0.3828	136.4569	-
Gamma Value	25	25	-
Error	Identification Error=58.0721%	Percent Error=0%	-
Characteristic	Very Close, Needs Optimization	Exact Match	-
A Eigenvalue 1	-0.7677	-0.4576	-0.5
A Eigenvalue 2	0	-0.3138	-0.25

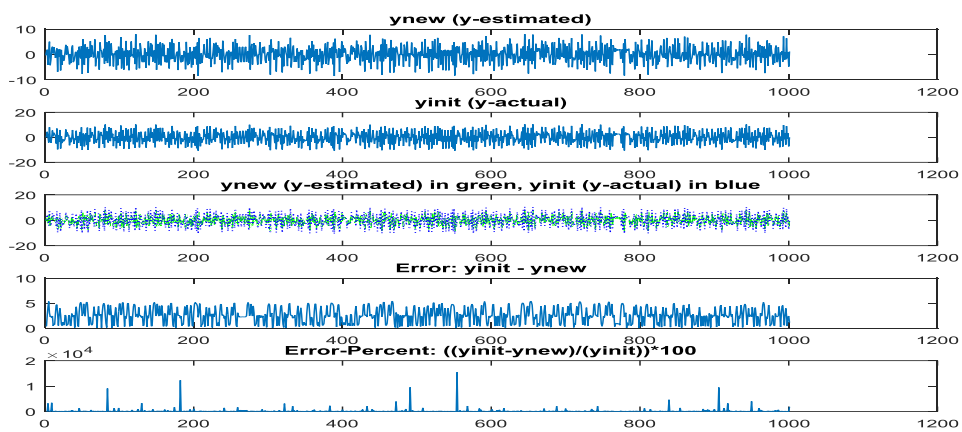

Figure-5.1: Iterative ADMM Output & Actual Output Compared – System 1 [25]

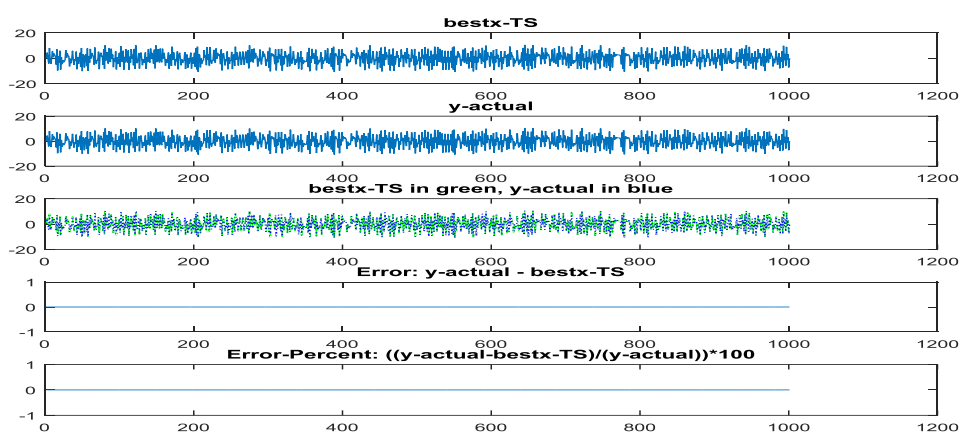

Figure-5.2: Tabu Search Output & Actual Output Compared – System 1 [25]

Recovering Wiener-Hammerstein Nonlinear State-Space Models Using Liner Algebra (P. Dreesen, M. Ishteva, and J. Schoukens, 2015)-System 2 [25]

$$G\_true(z) = \frac{8.4853z^2+25.4558z+16.9706}{z^3-1.5z^2+z-0.25} \quad (5.5)$$

$$G\_IterADMM(z) = \frac{0.0004246z^2+14.16z+13}{z^3-1.682z^2+1.19z-0.3228} \quad (5.6)$$

$$G\_TabuSearch(z) = \frac{2.203e-05z^2+38.14z+7.02}{z^3-1.548z^2+1.049z-0.2695} \quad (5.7)$$

Table-5.2: Iterative ADMM & Tabu Search Results – System 2 [25]

	Iterative ADMM	Tabu Search	True System
# of iterations	201	100	-
Computation Time	0.8775	128.7338	-
Gamma Value	427.3409	427.3409	-
Error	Identification Error=42.1145%	Percent Error=0%	-
Characteristic	Very Close, Needs Optimization	Exact Match	-
A Eigenvalue 1	0.5445 + 0.4982i	0.5096 + 0.4997i	0.5000 + 0.5000i
A Eigenvalue 2	0.5445 – 0.4982i	0.5096 - 0.4997i	0.5000 - 0.5000i
A Eigenvalue 3	0.5927 + 0.0000i	0.5289 + 0.0000i	0.5000 + 0.0000i

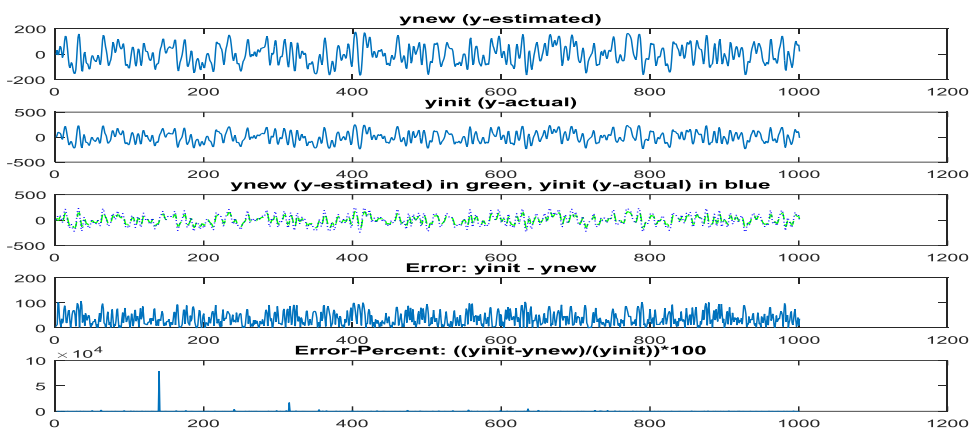

Figure-5.3: Iterative ADMM Output & Actual Output Compared – System 2 [25]

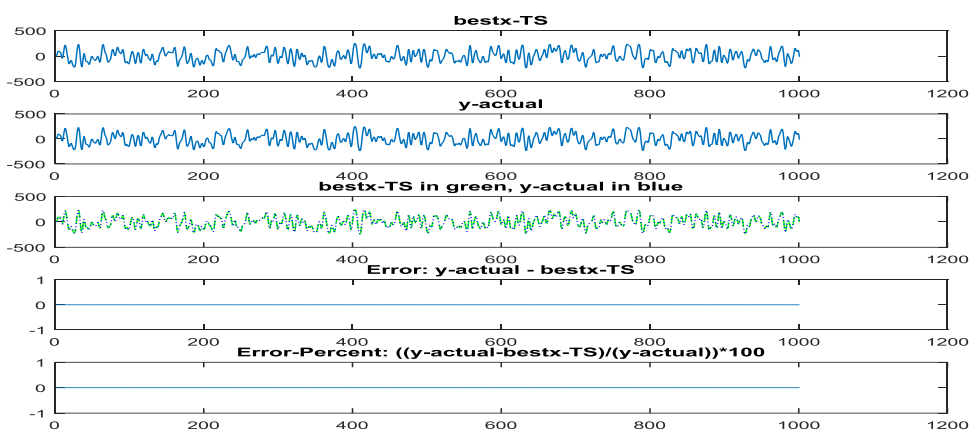

Figure-5.4: Tabu Search Output & Actual Output Compared – System 2 [25]

Version 7.0 of the CONTSID Toolbox (A. Padilla, H. Garnier, and M. Gilson, 2015)-System 1 [26]

$$G\_true(z) = \frac{-0.7757z^3 - 1.367z^2 + 1.624z + 0.6028}{z^4 - 1.198z^3 + 0.3238z^2 - 0.6383z + 0.6015} \quad (5.8)$$

$$G\_IterADMM(z) = \frac{-0.000235z^3 - 1.821z^2 + 2.043z - 0.1704}{z^4 - 1.493z^3 + 0.7493z^2 - 0.6952z + 0.5122} \quad (5.9)$$

$$G\_TabuSearch(z) = \frac{-0.0001543z^3 - 2.381z^2 + 1.817z + 0.7007}{z^4 - 1.148z^3 + 0.1312z^2 - 0.4208z + 0.5251} \quad (5.10)$$

Table-5.3: Iterative ADMM & Tabu Search Results – System 1 [26]

	Iterative ADMM	Tabu Search	True System
# of iterations	179	100	-
Computation Time	1.0068	173.6016	-
Gamma Value	8.9437	8.9437	-
Error	Identification Error=36.7502%	Percent Error=0%	-
Characteristic	Very Close, Needs Optimization	Exact Match	-
A Eigenvalue 1	0.9398 + 0.1843i	0.9363 + 0.1839i	0.9328 + 0.1834i
A Eigenvalue 2	0.9398 – 0.1843i	0.9363 – 0.1839i	0.9328 – 0.1834i
A Eigenvalue 3	-0.1932 + 0.7219i	-0.3621 + 0.6676i	-0.3338 + 0.7444i
A Eigenvalue 4	-0.1932 – 0.7219i	-0.3621 – 0.6676i	-0.3338 – 0.7444i

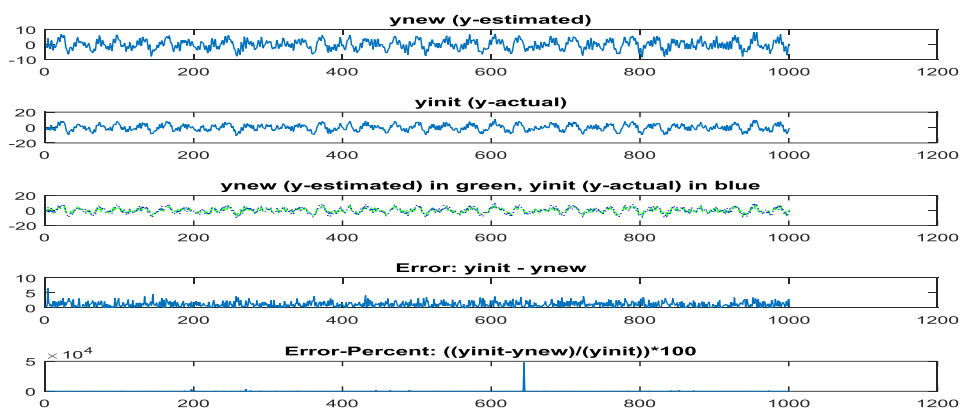

Figure-5.5: Iterative ADMM Output & Actual Output Compared – System 1 [26]

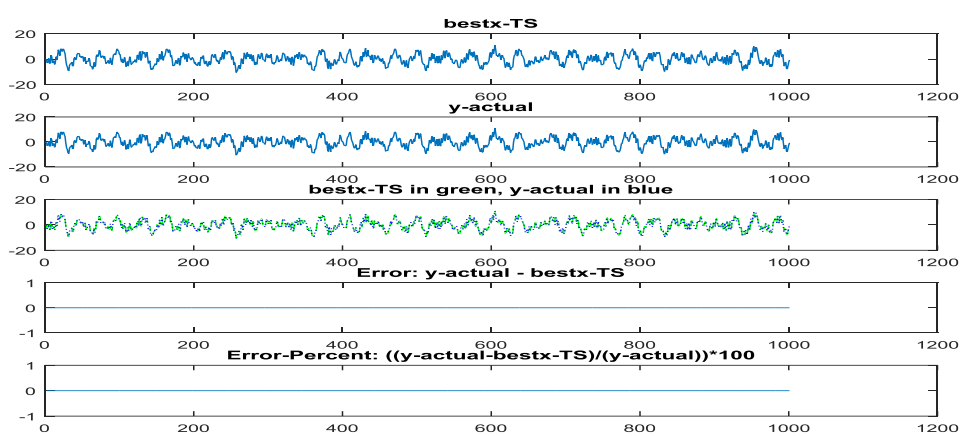

Figure-5.6: Tabu Search Output & Actual Output Compared – System 1 [26]

Version 7.0 of the CONTSID Toolbox (A. Padilla, H. Garnier, and M. Gilson, 2015)-System 2 [26]

$$G\_true(z) = \frac{-0.7526z^3 - 1.329z^2 + 1.58z + 0.5876}{z^4 - 1.194z^3 + 0.3258z^2 - 0.6458z + 0.6059} \quad (5.11)$$

$$G\_IterADMM(z) = \frac{-3.801e{-}06z^3 - 1.63z^2 + 0.6493z - 0.9706}{z^4 - 0.08649z^3 + 0.3589z^2 - 0.2013z + 0.2105} \quad (5.12)$$

$$G\_TabuSearch(z) = \frac{-0.000158z^3 - 2.31z^2 + 1.77z + 0.6786}{z^4 - 1.145z^3 + 0.1359z^2 - 0.4308z + 0.5303} \quad (5.13)$$

Table-5.4: Iterative ADMM & Tabu Search Results – System 2 [26]

	Iterative ADMM	Tabu Search	True System
# of iterations	187	100	-
Computation Time	0.9257	135.1128	-
Gamma Value	17.6022	17.6022	-
Error	Identification Error=69.9599%	Percent Error=0%	-
Characteristic	Moderately Close, Needs Optimization	Exact Match	-
A Eigenvalue 1	-0.4482 + 0.000i	0.9354 + 0.1865i	0.9319 + 0.1861i
A Eigenvalue 2	-0.0708 + 0.8303i	0.9354 – 0.1865i	0.9319 – 0.1861i
A Eigenvalue 3	-0.0708 - 0.8303i	-0.3626 + 0.6719i	-0.3349 + 0.7476i
A Eigenvalue 4	0.6764 + 0.0000i	-0.3626 - 0.6719i	-0.3349 – 0.7476i

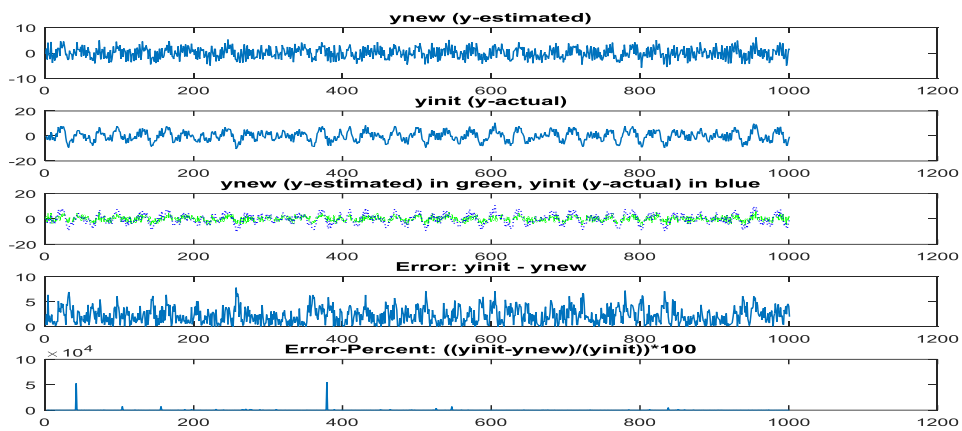

Figure-5.7: Iterative ADMM Output & Actual Output Compared – System 2 [26]

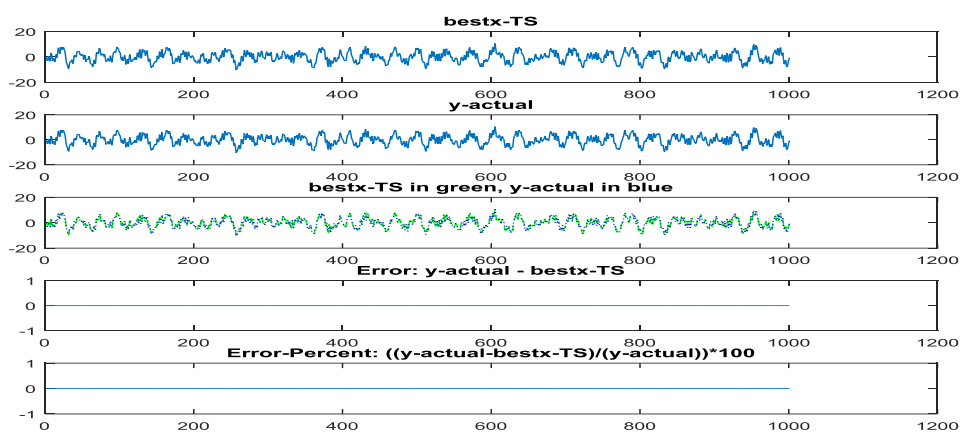

Figure-5.8: Tabu Search Output & Actual Output Compared – System 2 [26]

Version 7.0 of the CONTSID Toolbox (A. Padilla, H. Garnier, and M. Gilson, 2015)-System 3 [26]

$$G\_true(z) = \frac{0.05257z+0.04601}{z^2 - 1.637z + 0.6703} \tag{5.14}$$

$$G\_IterADMM(z) = \frac{0.04767z^2 - 4.084e{-}05z + 5.438e{-}05}{z^2 - 1.732z + 0.7571} \tag{5.15}$$

$$G\_TabuSearch(z) = \frac{0.08385z+2.908e{-}05}{z^2 - 1.69z + 0.7183} \tag{5.16}$$

Table-5.5: Iterative ADMM & Tabu Search Results – System 3 [26]

	Iterative ADMM	Tabu Search	True System
# of iterations	149	100	-
Computation Time	0.7612	135.1274	-
Gamma Value	0.9346	0.9346	-
Error	Identification Error=43.3266%	Percent Error=0%	-
Characteristic	Very Close, Needs Optimization	Exact Match	-
A Eigenvalue 1	0.8662 + 0.0820i	0.8449 + 0.0665i	0.8185 + 0.0189i
A Eigenvalue 2	0.8662 – 0.0820i	0.8449 – 0.0665i	0.8185 – 0.0189i

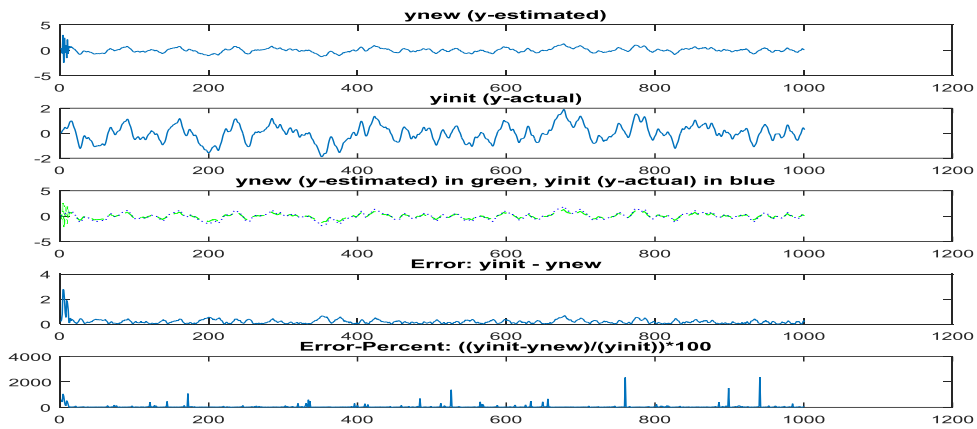

Figure-5.9: Iterative ADMM Output & Actual Output Compared – System 3 [26]

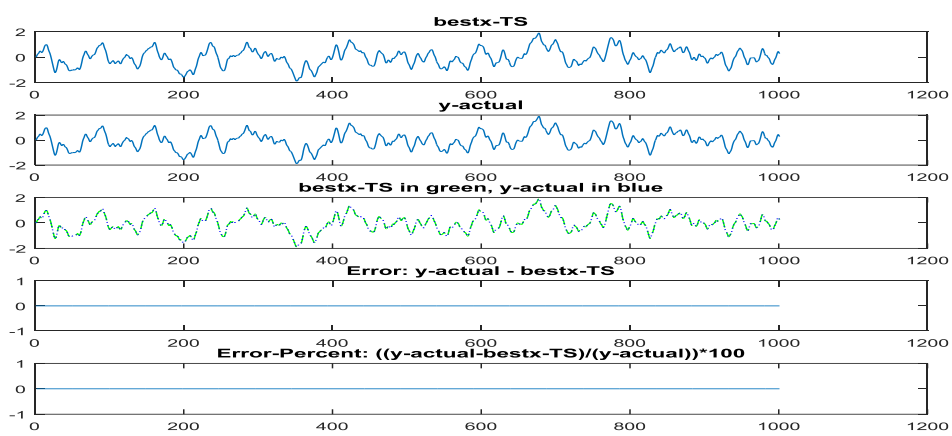

Figure-5.10: Tabu Search Output & Actual Output Compared – System 3 [26]

Identification of block-oriented systems with rate saturation nonlinearity (A. Y. K. Yong, A. H. Tan, and C. L. Cham, 2015)-System 1 [27]

$$G\_true(z) = \frac{z+0.5}{z^2 - 1.5z + 0.7} \tag{5.17}$$

$$G\_IterADMM(z) = \frac{-2.751e-05z+0.1489}{z^2 - 1.58z + 0.6947} \tag{5.18}$$

$$G\_TabuSearch(z) = \frac{-1.384e-05z+1.66}{z^2 - 1.411z + 0.6503} \tag{5.19}$$

Table-5.6: Iterative ADMM & Tabu Search Results – System 1 [27]

	Iterative ADMM	Tabu Search	True System
# of iterations	153	100	-
Computation Time	0.7766	136.7983	-
Gamma Value	75	75	-
Error	Identification Error=89.4514%	Percent Error=0%	-
Characteristic	Moderately Close, Needs Optimization	Exact Match	-
A Eigenvalue 1	0.7898 + 0.2662i	0.7054 + 0.3907i	0.7500 + 0.3708i
A Eigenvalue 2	0.7898 - 0.2662i	0.7054 - 0.3907i	0.7500 - 0.3708i

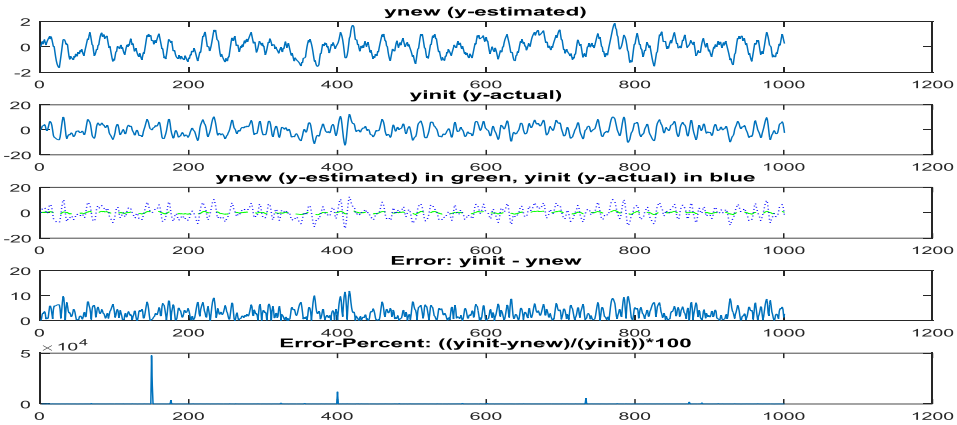

Figure-5.11: Iterative ADMM Output & Actual Output Compared – System 1 [27]

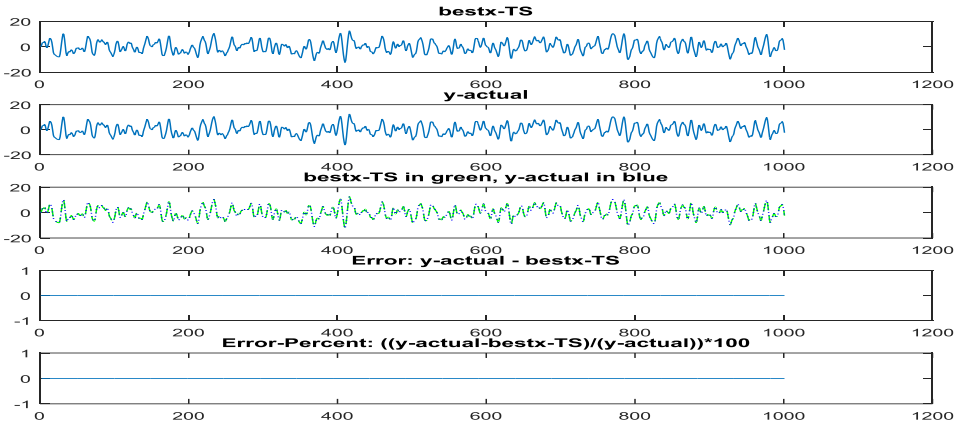

Figure-5.12: Tabu Search Output & Actual Output Compared – System 1 [27]

Identification of block-oriented systems with rate saturation nonlinearity (A. Y. K. Yong, A. H. Tan, and C. L. Cham, 2015)-System 2 [27]

$$G\_true(z) = \frac{0.0644z^2+0.0249z-0.0045}{z^3-2.451z^2+2.129z-0.6666} \quad (5.20)$$

$$G\_IterADMM(z) = \frac{-1.498e-07z^2+0.004713z+0.001619}{z^3-2.517z^2+2.202z-0.6771} \quad (5.21)$$

$$G\_TabuSearch(z) = \frac{1.727e-06z^2+0.1604z-0.08228}{z^3-2.43z^2+2.077z-0.6368} \quad (5.22)$$

Table-5.7: Iterative ADMM & Tabu Search Results – System 2 [27]

	Iterative ADMM	Tabu Search	True System
# of iterations	125	100	-
Computation Time	0.5013	142.2242	-
Gamma Value	25	25	-
Error	Identification Error=90.3023%	Percent Error=0%	-
Characteristic	Moderately Close, Needs Optimization	Exact Match	-
A Eigenvalue 1	0.9461 + 0.0000i	0.9438 + 0.0000i	0.9427 + 0.0000i
A Eigenvalue 2	0.7854 + 0.3142i	0.7431 + 0.3500i	0.7542 + 0.3720i
A Eigenvalue 3	0.7854 – 0.3142i	0.7431 - 0.3500i	0.7542 – 0.3720i

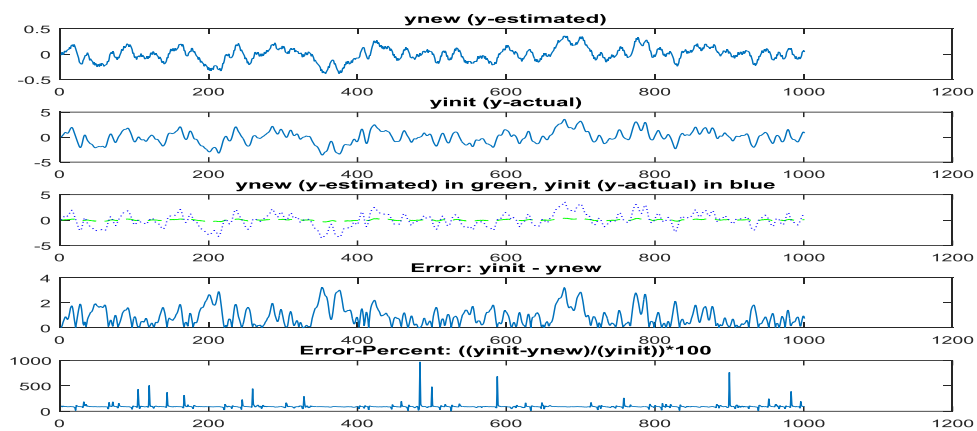

Figure-5.13: Iterative ADMM Output & Actual Output Compared – System 2 [27]

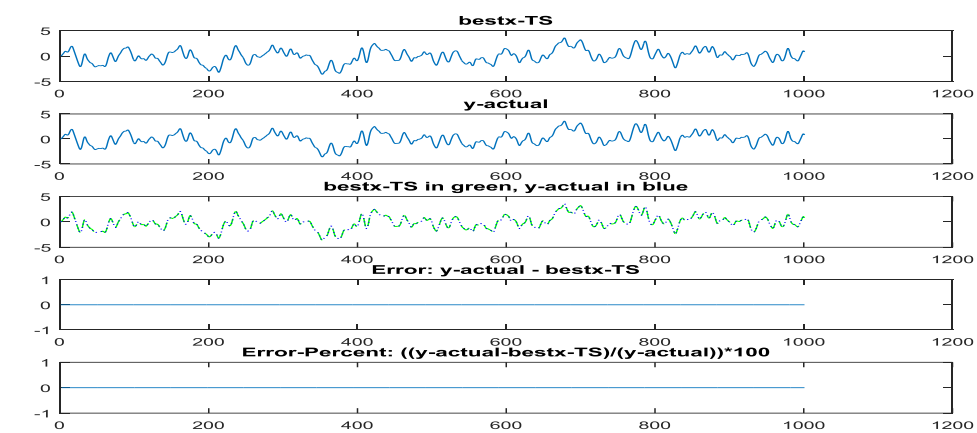

Figure-5.14: Tabu Search Output & Actual Output Compared – System 2 [27]

Identification of block-oriented systems with rate saturation nonlinearity (A. Y. K. Yong, A. H. Tan, and C. L. Cham, 2015)-System 3 [27]

$$G\_true(z) = \frac{0.4479z^2+0.1254z-0.0076}{z^3-2.114z^2+1.622z-0.4319} \quad (5.23)$$

$$G\_IterADMM(z) = \frac{8.524e-06z^2+0.04927z+0.009867}{z^3-2.474z^2+2.156z-0.6554} \quad (5.24)$$

$$G\_TabuSearch(z) = \frac{-2.562e05z^2+0.9902z-0.7127}{z^3-2.3z^2+1.888z-0.5486} \quad (5.25)$$

Table-5.8: Iterative ADMM & Tabu Search Results – System 3 [27]

	Iterative ADMM	Tabu Search	True System
# of iterations	159	100	-
Computation Time	0.7058	134.9639	-
Gamma Value	46.4	46.4	-
Error	Identification Error=87.1189%	Percent Error=0%	-
Characteristic	Moderately Close, Needs Optimization	Exact Match	-
A Eigenvalue 1	0.8275 + 0.3400i	0.7574 + 0.3540i	0.7452 + 0.3705i
A Eigenvalue 2	0.8275 - 0.3400i	0.7574 - 0.3540i	0.7452 - 0.3705i
A Eigenvalue 3	0.8189 – 0.0000i	0.7849 + 0.0000i	0.6236 + 0.0000i

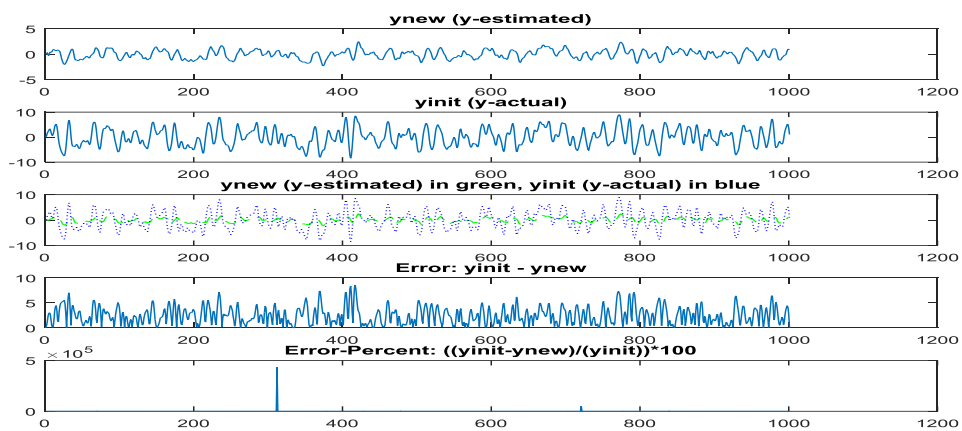

Figure-5.15: Iterative ADMM Output & Actual Output Compared – System 3 [27]

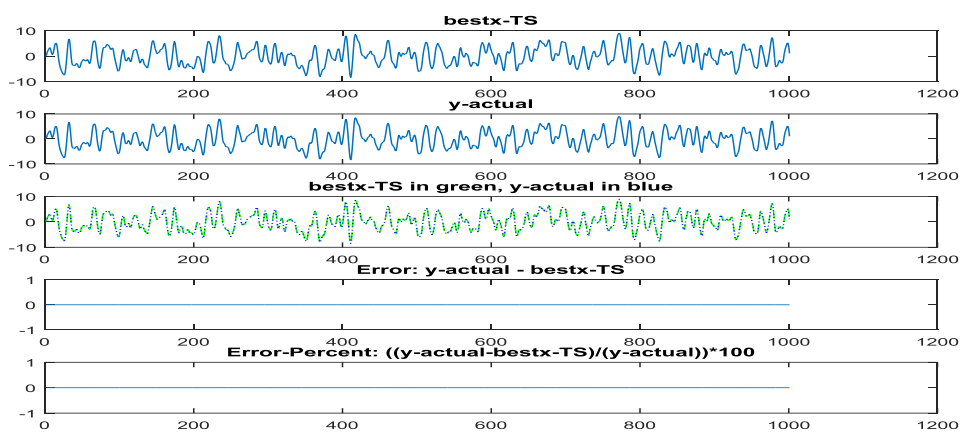

Figure-5.16: Tabu Search Output & Actual Output Compared – System 3 [27]

A subspace-based identification of two-channel wiener systems (H. Ase, and T. Katayama, 2015)-System 1 [28]

$$G\_true(z) = \frac{-0.03z+0.1}{z^2-1.8z+0.85} \tag{5.26}$$

$$G\_IterADMM(z) = \frac{-3.009e-06z+0.02164}{z^2-1.791z+0.8437} \tag{5.27}$$

$$G\_TabuSearch(z) = \frac{-1.278e-05z+0.06723}{z^2-1.811z+0.8585} \tag{5.28}$$

Table-5.9: Iterative ADMM & Tabu Search Results – System 1 [28]

	Iterative ADMM	Tabu Search	True System
# of iterations	152	100	-
Computation Time	0.7088	145.6996	-
Gamma Value	7.02	7.02	-
Error	Identification Error=71.2422%	Percent Error=0%	-
Characteristic	Very Close, Needs Optimization	Exact Match	-
A Eigenvalue 1	0.8956 + 0.2038i	0.9056 + 0.1959i	0.9000 + 0.2000i
A Eigenvalue 2	0.8956 – 0.2038i	0.9056 - 0.1959i	0.9000 - 0.2000i

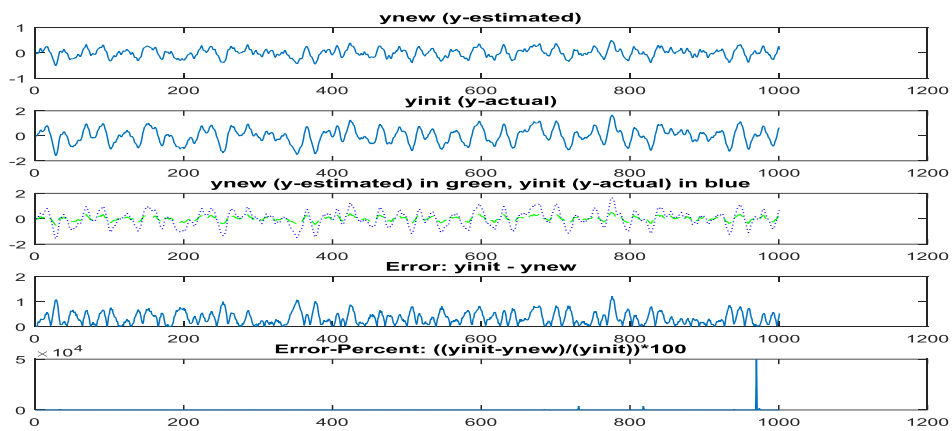

Figure-5.17: Iterative ADMM Output & Actual Output Compared – System 1 [28]

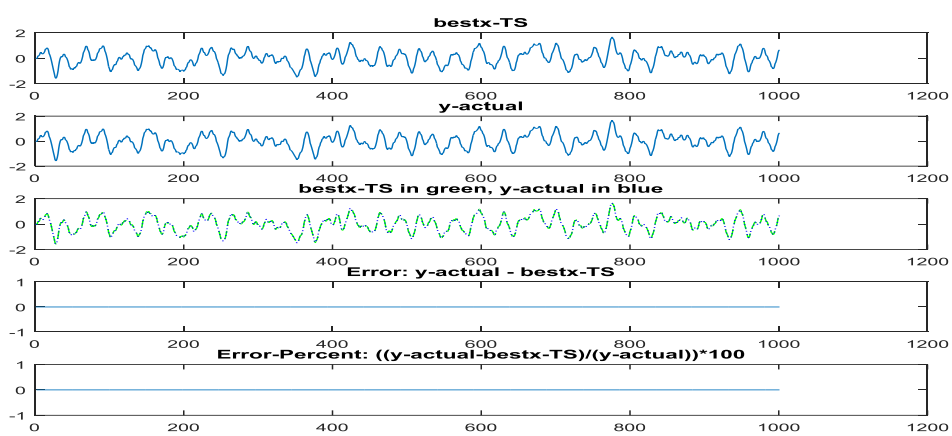

Figure-5.18: Tabu Search Output & Actual Output Compared – System 1 [28]

A subspace-based identification of two-channel wiener systems (H. Ase, and T. Katayama, 2015)-System 2 [28]

$$G\_true(z) = \frac{0.06875z+0.13775}{z^5 - 2.34z^4 + 3.08z^3 - 2.53z^2 + 1.24z - 0.368} \quad (5.29)$$

$$G\_IterADMM(z) = \frac{-2.547e-06z^4 - 0.01145z^3 + 0.09295z^2 - 0.1272z + 0.1945}{z^5 - 2.112z^4 + 2.747z^3 - 2.287z^2 + 1.149z - 0.4002} \quad (5.30)$$

$$G\_TabuSearch(z) = \frac{2.052e-09z^4 - 1.44e-09z^3 + 4.646e-09z^2 + 0.06875z + 0.1378}{z^5 - 2.34z^4 + 3.08z^3 - 2.53z^2 + 1.24z - 0.368} \quad (5.31)$$

Table-5.10: Iterative ADMM & Tabu Search Results – System 2 [28]

	Iterative ADMM	Tabu Search	True System
# of iterations	144	100	-
Computation Time	0.7452	133.8792	-
Gamma Value	4.4725	4.4725	-
Error	Identification Error=51.4888%	Percent Error=0%	-
Characteristic	Very Close, Needs Optimization	Exact Match	-
A Eigenvalue 1	0.9131 + 0.0000i	0.9012 + 0.0000i	0.9012 + 0.0000i
A Eigenvalue 2	0.2068 + 0.7576i	0.3037 + 0.7491i	0.3037 + 0.7491i
A Eigenvalue 3	0.2068 - 0.7576i	0.3037 - 0.7491i	0.3037 - 0.7491i
A Eigenvalue 4	0.3926 + 0.7460i	0.4157 + 0.6724i	0.4157 + 0.6724i
A Eigenvalue 5	0.3926 - 0.7460i	0.4157 - 0.6724i	0.4157 - 0.6724i

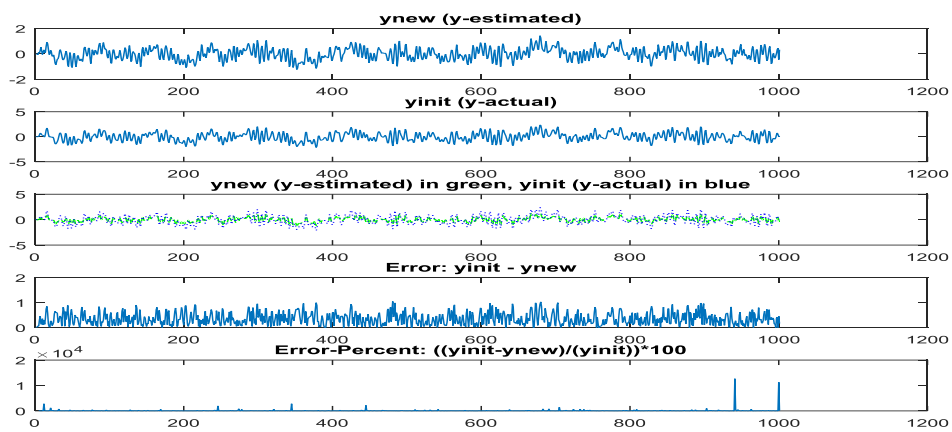

Figure-5.19: Iterative ADMM Output & Actual Output Compared – System 2 [28]

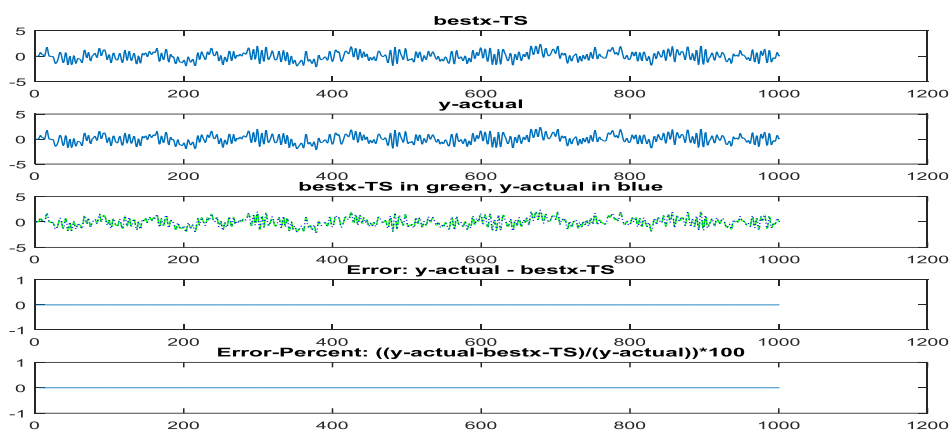

Figure-5.20: Tabu Search Output & Actual Output Compared – System 2 [28]

A least squares method for identification of feedback cascade systems (M. Galrinho, C. R. Rojas, and H. Hjalmarsson, 2015)-System 1 [29]

$$G\_true(z) = \frac{1}{z^2 - 0.2z + 0.5} \tag{5.32}$$

$$G\_IterADMM(z) = \frac{-1.43e-07z+0.6673}{z^2 - 0.5887z + 0.4752} \tag{5.33}$$

$$G\_TabuSearch(z) = \frac{-1.623e-08z+1}{z^2 - 0.2z + 0.5} \tag{5.34}$$

Table-5.11: Iterative ADMM & Tabu Search Results – System 1 [29]

	Iterative ADMM	Tabu Search	True System
# of iterations	175	100	-
Computation Time	0.7678	159.8861	-
Gamma Value	1.677	1.677	-
Error	Identification Error=67.9169%	Percent Error=0%	-
Characteristic	Very Close, Needs Optimization	Exact Match	-
A Eigenvalue 1	0.2944 + 0.6233i	0.1000 + 0.7000i	0.1000 + 0.7000i
A Eigenvalue 2	0.2944 – 0.6233i	0.1000 - 0.7000i	0.1000 - 0.7000i

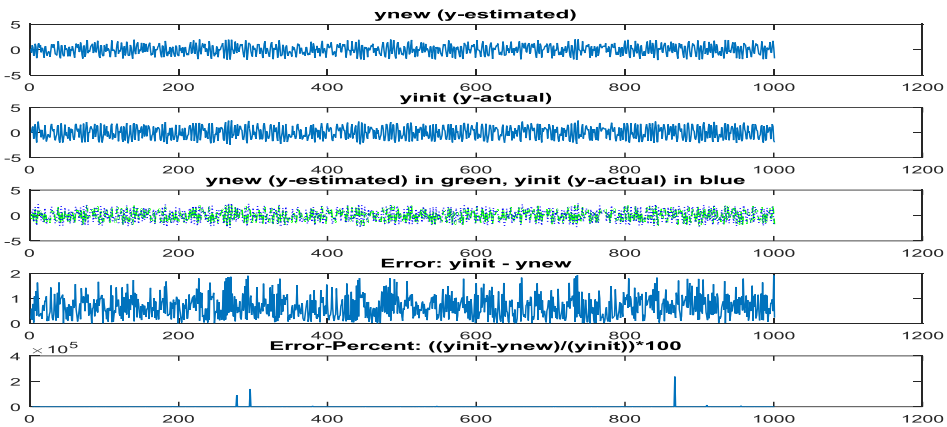

Figure-5.21: Iterative ADMM Output & Actual Output Compared – System 1 [29]

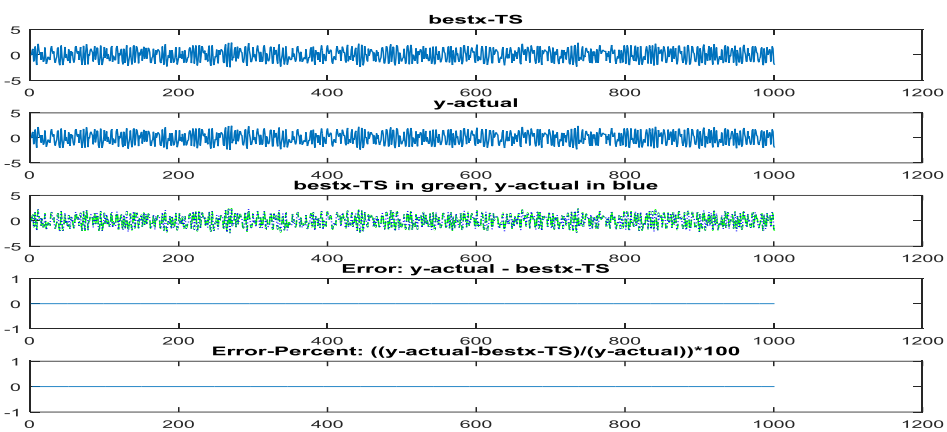

Figure-5.22: Tabu Search Output & Actual Output Compared – System 1 [29]

Version 7.0 of the CONTSID Toolbox (A. Padilla, H. Garnier, and M. Gilson, 2015)-System 4 [26]

$$G\_true(z) = \frac{-0.7691z^3 - 1.359z^2 + 1.616z + 0.6008}{z^4 - 1.2z^3 + 0.3309z^2 - 0.6484z + 0.6065} \quad (5.35)$$

$$G\_IterADMM(z) = \frac{-3.257e{-}06z^3 - 0.3884z^2 + 0.3775z + 9.629e{-}06}{z^4 - 3.022z^3 + 3.712z^2 - 2.255z + 0.5847} \quad (5.36)$$

$$G\_TabuSearch(z) = \frac{-0.0001684z^3 - 2.366z^2 + 1.813z + 0.6935}{z^4 - 1.151z^3 + 0.1404z^2 - 0.4324z + 0.5304} \quad (5.37)$$

Table-5.12: Iterative ADMM & Tabu Search Results – System 4 [26]

	Iterative ADMM	Tabu Search	True System
# of iterations	182	100	-
Computation Time	0.9399	134.3384	-
Gamma Value	23.351	23.351	-
Error	Identification Error=72.2935%	Percent Error=0%	-
Characteristic	Very Close, Needs Optimization	Exact Match	-
A Eigenvalue 1	0.5818 + 0.5584i	-0.3611 + 0.6721i	-0.3333 + 0.7479i
A Eigenvalue 2	0.5818 – 0.5584i	-0.3611 – 0.6721i	-0.3333 – 0.7479i
A Eigenvalue 3	0.9292 + 0.1889i	0.9367 + 0.1838i	0.9333 + 0.1834i
A Eigenvalue 4	0.9292 - 0.1889i	0.9367 - 0.1838i	0.9333 - 0.1834i

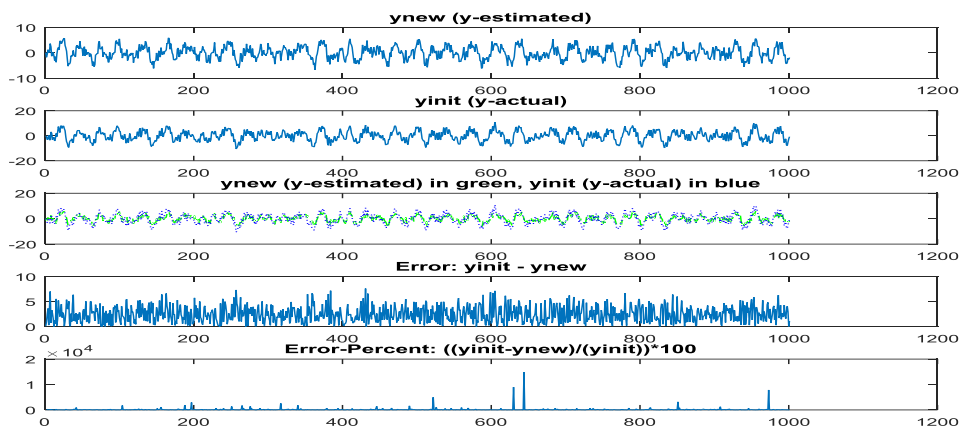

Figure-5.23: Iterative ADMM Output & Actual Output Compared – System 4 [26]

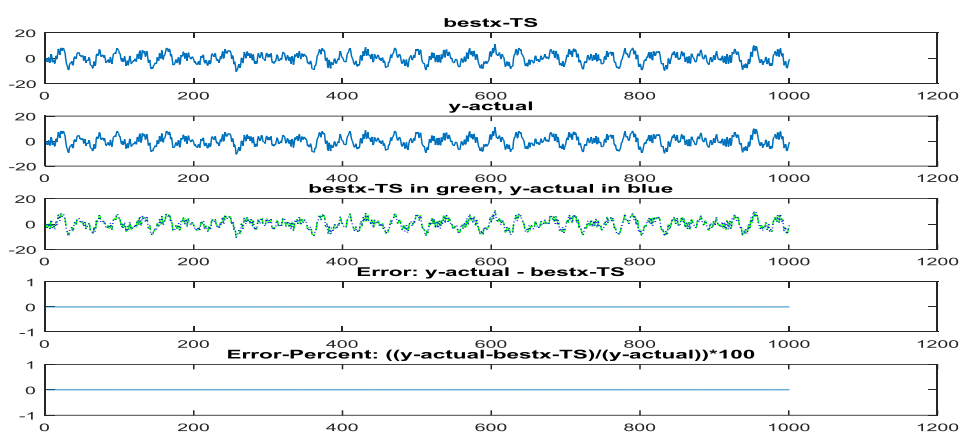

Figure-5.24: Tabu Search Output & Actual Output Compared – System 4 [26]

CHAPTER 6.0 – CONCLUSION & FUTURE WORK

6.1 – CONCLUSION

In this thesis, the author has described the use of Matlab™ and Simulink™ to build, test, validate, and analyze a tool box for performing Iterative ADMM method that uses nuclear norm technique for system identification, as outlined by Harshad Deshmane in his thesis [3]. Secondly, the incorporation and adaption of an existing programmatic tool to perform Tabu Search optimization for the same system for system identification has been presented. Matlab™ version 8.5.0.197613 (R2015a) was used for programming and analysis. Experiments were executed on a 2.40 GHz Intel® Core™ 2 Duo laptop computer with 4.00 GB of memory, and 64 bit operating system. Through metrics presented in Chapter 5.0, it is concluded that Tabu Search optimization program achieves optimal results for system identification, whereas Iterative ADMM program is able to perform estimation through varying value of gamma to some accuracy, but is not able to produce optimal results.

6.2 – FUTURE WORK

Iterative ADMM program could be enhanced to perform estimation for a broad range of values for regularization parameter gamma, while observing the identification error. Further automation and adaptation could be performed to the existing program to yield an optimal solution for a given system. A higher order system could be investigated, by increasing block-row dimension for construction of Hankel matrices.

In this research work, the gradient descent method is chosen in development of Iterative ADMM method involving inner and outer line search [3]. Other methods that could be investigated for development of Iterative ADMM solution include Newton and Quasi-Newton methods [3]. Newton's method involves using D^l as Hessian $\nabla^2 \mathcal{L}_\rho(y)$ [3]. Quasi-Newton methods use D^l which is an approximation to the Hessian, evaluated from information about gradient computed at present and past iterates [3].

This research utilizes the Barzilai – Borwein method for evaluation of step size [3]. Two other methods that could be researched that involve different methods for calculation of step-size are Backtracking line search method and Wolfe line search [3]. Backtracking line search is based on the requirement that step-size α^l meets the Armijo condition, while Wolfe line search is based on the requirement that step-size α^l meets the curvature condition and the Armijo condition [3].

REFERENCES

[1] D. Sadigh, H. Ohlsson, S. S. Sastry, and S. A. Seshia, "Robust subspace system identification via weighted nuclear norm optimization," in *Proc. 19th IFAC World Congress*, Cape Town, South Africa, 2014, pp. 9510-9515.

[2] K. J. Keesman, *System identification an introduction*. New York: Springer, 2011.

[3] H. Deshmane, "System identification via nuclear norm regularization," M. S. thesis, Dept. Elect. Eng., Dept. Aeros. Eng. and Mecha., Dept. Contr. Scien. and Dynam. Syste., Univ. of Minnesota, Minneapolis, MN, 2014.

[4] M. Fazel, H. Hindi, and S. Boyd. (2001, June). A rank minimization heuristic with application to minimum order system approximation. Presented at American Control Conference, 2001. [Online]. Available: https://faculty.washington.edu/mfazel/nucnorm_acc_final.pdf

[5] M. Fazel, "Matrix rank minimization with applications," Ph.D. dissertation, Dept. Elect. Eng., Comm. Grad. Stud., Standford Univ., Stanford, CA, 2002. [Online]. Available: https://faculty.washington.edu/mfazel/thesis-final.pdf

[6] Z. Liu, A. Hansson, and L. Vandenberghe. (2013, Aug.). Nuclear norm system identification with missing inputs and outputs. *Systems & Control Lett*. [Online]. 62(8), pp. 605-612. Available: http://www.seas.ucla.edu/~vandenbe/publications/subspace.pdf

[7] A. Krishnamurthy. Some properties of matrix norms. [Online]. Available: http://www.cs.cmu.edu/~akshaykr/files/matrix_norms.pdf

[8] Department of Electrical and Computer Systems Engineering, Monash University, Clayton. Understanding and applying kalman filtering. [Online]. Available: http://biorobotics.ri.cmu.edu/papers/sbp_papers/integrated3/kleeman_kalman_basics.pdf

[9] A. A. Bastmi, M. R. Belić, and N. Z. Petrović. (2010). Special solutions of the riccati equation with applications to the gross-pitaevskii nonlinear pde. *Electr. Jour. Of Diff. Eq.* [Online]. 2010 (66), pp. 1-10. Available: http://ejde.math.txstate.edu/Volumes/2010/66/albastami.pdf

[10] Mathworks. Hankel. [Online]. Available: http://www.mathworks.com/help/matlab/ref/hankel.html

[11] Electrical & Computer Engineering, Rutgers University. Controllability and obserbvability. [Online]. Available: http://www.ece.rutgers.edu/~gajic/psfiles/chap5traCO.pdf

[12] K. Ye, and L. Lim. (2015, Mar.). Every matrix is a product of toeplitz matrices. *Found. Of Compu. Math.*. [Online]. , pp. 1-22. Available: http://www.stat.uchicago.edu/~lekheng/work/toeplitz.pdf

[13] Wolfram MathWorld. Matrix inverse. [Online]. Available: http://mathworld.wolfram.com/MatrixInverse.html

[14] Mathworks. Validating models after estimation. [Online]. Available: http://www.mathworks.com/help/ident/ug/validating-models-after-estimation.html

[15] R. L. Williams II, and D. A. Lawrence, *Linear state-space control systems*. Hoboken, NJ: John Wiley & Sons, 2007.

[16] Wolfram MathWorld. Null space. [Online]. Available: http://mathworld.wolfram.com/NullSpace.html

[17] Mathworks. Null. [Online]. Available: http://www.mathworks.com/help/matlab/ref/null.html

[18] Z. Liu, and L. Vandenberghe (2010). Interior-point method for nuclear norm approximation with application to system identification. *SIAM J. on Matrix Ana. and Applic.* [Online]. 31(3), pp. 1235 – 1256. Available: http://www.seas.ucla.edu/~vandenbe/publications/nucnrm.pdf

[19] D. Puangdownreong, K. N. Areerak, A. Srikaew, S.Sujitjorn, and P. Totarong. (2002). System identification via adaptive tabu search. Presented at 2002 Intern. Conf. on Indus. Tech., IEEE ICIT 2002. [Online]. Available: http://203.158.6.11:8080/sutir/bitstream/123456789/1975/2/bib1488_F.pdf

[20] S. Sujitjorn, J. Kluabwang, D. Puangdownreong, and N. Sarasiri. (2010, Jan.). Adaptive tabu search and management agent. *ECTI Trans..on Elec. Eng., Electronics., and Communications* [Online]. 8(1), pp. 1-10. Available: http://www.ecti-thailand.org/assets/papers/811_pub_26.pdf

[21] J. Juang, and M. Q. Phan, *Identification and Control of Mechanical Systems*. Cambridge, NY: Cambridge University Press, 2001.

[22] J. S. Arora, *Introduction to Optimum Design*. Waltham, MA: Elsevier, 2012.

[23] J. N. Juang, and H. Suzuki (1988, Jan.). An eigensystem realization algorithm in frequency domain for modal parameter identification. *ASME J. of Vibr., Acous, Stress, and Reliab. in Design* [Online]. 110(1), pp. 24-29. Available: http://aero.tamu.edu/sites/default/files/jnj_symposium/JVM%2088.pdf

[24] W. Xu, and S. Qiao, (2006). A fast symmetric svd algorithm for square hankel matrices. Presented at The 2nd Inter. Conf. on Struct. Matr. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0024379507002443

[25] P. Dreesen, M. Ishteva, and J. Schoukens, "Recovering wiener-hammerstein nonlinear state-space models using liner algebra," *in Proc. 17th IFAC Symp. on System Identification,* Beijing, China, 2015, pp. 951-956.

[26] A. Padilla, H. Garnier, and M. Gilson, "Version 7.0 of the CONTSID toolbox," *in Proc. 17th IFAC Symp. on System Identification,* Beijing, China, 2015, pp. 757-762.

[27] A. Y. K. Yong, A. H. Tan, and C. L. Cham, "Identification of block-oriented systems with rate saturation nonlinearity," *in Proc. 17th IFAC Symp. on System Identification,* Beijing, China, 2015, pp. 939-944.

[28] H. Ase, and T. Katayama, "A subspace-based identification of two-channel wiener systems," *in Proc. 17th IFAC Symp. on System Identification,* Beijing, China, 2015, pp. 638-643.

[29] M. Galrinho, C. R. Rojas, and H. Hjalmarsson, "A least squares method for identification of feedback cascade systems," *in Proc. 17th IFAC Symp. on System Identification*, Beijing, China, 2015, pp. 98-103.

APENDIX A

A1 –Main Program

```
% File Name: Main.m
% Date Created: 10/14/2015
% Author: Asif Ahmed
% Description: Harshad Deshmane Thesis, Iterative ADMM Algorithm,
%            Section 3.3
% Reference1: Harshad Deshmane, “System Identification via Nuclear
%            Norm Regularization,” 2014.
% Reference2: Dr. Kenneth Bosworth, Notes, Idaho State University,
%            Fall 2015.
% Reference3: Mathworks, "http://www.mathworks.com/matlabcentral/answers/64384-subtracting-two-
matrices-of-different-size-element-by-element," 2013
% Reference4: Stackoverflow, "http://stackoverflow.com/questions/738438/removing-zeros-from-a-matrix-
not-sparse," 2013
% Reference5: Stanford University, "https://see.stanford.edu/materials/lsoeldsee263/05-ls.pdf," 2007
% Reference6: Mathworks, "http://www.mathworks.com/help/control/ref/impulse.html," 2016
% ____________________________________________________________________________

%%
% Define variables:
global m
global p
global N
global u
global y_hat
global y
global r
r = 2; % Define r=2 to be used in hank.m
global b
% obtain b from y with number of rows = number of elements in y matrix,
% and 1 colum, to be used in AugLagrangian.m :
b = reshape(y_hat, [numel(y_hat), 1]);
global rho
rho = 1; % Define rho=1, to be used in AugLagrangian.m
global Uperp
global Gamma
Gamma = 1; % Define Gamma=1, to be used in AugLagrangian.m
global maxitinner %maxitinner for inner iteration inverter.m function
maxitinner = 20;
global stoptol %stoptol for inner iteration inverter.m function
stoptol = 0.001;
global maxitouter % maxitoutter for outer iteration
maxitouter = 20;

% Developing U as hankel matrix from u:
U = hank(u);

% Obtaining orthogonal complement of U:
Uperp = null(U);

% Developing Y as hankel matrix from y:
```

```
global Y
Y = hank(y);

%-----------------------------------------------------------------------
% Example computation to make matrix dimensions work for the
% 'AugLagrangian.m' and 'GradLagr.m' functions:
% Z is generated randomnly with proper dimensions:
Z = zeros(2, 998);
% lagrangemultiplier is generated randomnly with proper dimensions:
lagrangemultiplier = zeros(2, 998);
%-----------------------------------------------------------------------

%%
% Simulink Model Transfer Function:
num = [0.7071 4.2426];
den = [1 0.750 0.125];
Hd = tf(num, den, 0.1)
% Hd =
%
%     0.7071 z + 4.243
%   --------------------
%   z^2 + 0.75 z + 0.125
%
% Sample time: 0.1 seconds
% Discrete-time transfer function.
[ A, B, C, D ] = tf2ss(num, den)
% A =
%
%    -0.7500   -1.125
%     1.0000         0
%
%
% B =
%
%      1
%      0
%
%
% C =
%
%     0.7071    4.2426
%
%
% D =
%
%      0

% Find eigen-values of A matrix:
A_eig = eig(A)
% A_eig =
%
%    -0.5000
%    -0.2500
```

A2 –Hankel Function

```
% File Name: hank.m
% Date Created: 10/19/2015
% Author: Asif Ahmed
% Description: Harshad Deshmane Thesis, Iterative ADMM Algorithm,
%            Section 3.3
% Reference1: Harshad Deshmane, “System Identification via Nuclear
%            Norm Regularization,” 2014.
% Reference2: Dr. Kenneth Bosworth, Notes, Idaho State University,
%            Fall 2015.
% Reference3: Mathworks, "http://www.mathworks.com/matlabcentral/fileexchange/2290-subspace-
identification-for-linear-systems/content/vanoverschee/SUBFUN/blkhank.m", 2002.
% ____________________________________________________________________

function [ A ] = hank( z )

% r is number of block rows of resulting hankel matrix
% jj is number of colums of resulting hankel matrix

% Define variables:
global r;

% assign size of z matrix to k rows and M colums:
[k, M] = size(z);

% If z is colum vector, obtain z as row vector:
if M == 1
% obtain z as row vector with number of columns = number of elements in
% z matrix, and 1 row:
z = reshape(z, [1, numel(z)]);
% Obtain number of rows, and number of colums again:
[k, M] = size(z);
end

% Allocate storage for Hankel matrix:
A = zeros(k*r, M-r+1);

% Compute Number of colums for Hankel Matrix:
jj = M-r+1;

% Build Hankel matrix based on given z matrix:
for ii = 1:r
    A((ii-1)*k+1:ii*k, :) = z(:, ii:ii+jj-1);
end

return

end
```

A3 –Hankstar Function

```
% File Name: Hankstar.m
% Date Created: 10/19/2015
% Author: Asif Ahmed
```

```
% Description: Harshad Deshmane Thesis, Iterative ADMM Algorithm,
%             Section 3.3
% Reference1: Harshad Deshmane, “System Identification via Nuclear
%            Norm Regularization,” 2014.
% Reference2: Dr. Kenneth Bosworth, Notes, Idaho State University,
%            Fall 2015.
% Reference3: Stackoverflow, "http://stackoverflow.com/questions/18816841/obtaining-opposite-diagonal-
of-a-matrix-in-matlab," 2013.
% Reference4: Mathworks, "http://www.mathworks.com/matlabcentral/answers/48938-delete-element-
from-vector," 2012.
% Reference5: Mathworks, "http://www.mathworks.com/matlabcentral/newsreader/view_thread/237551,"
2008.
%_____________________________________________________________________

function [ A ] = Hankstar( Q )

% assign size of Q matrix to k rows and M colums:
[k, M] = size(Q);

% Allocate storage for row vector:
% If Q contains only 1 row, then A is size 1xM
if k==1
   A = zeros(1,M);
% Elseif Q contains only 1 column, then A is size 1xk
elseif M==1
   A = zeros(1,k);
% ElseIf Q contains only >1 rows and/or >1 columns, then A is size 1x(M+1)
else
   A = zeros(1, M+1);
end

% Initialize number of colums of row vector as 1:
jj = 1;

% Initialize forward & reverse diagonal counters as zero:
fwd_diag_counter = 0;
rev_diag_counter = 0;

% Count number of diagonals above main diagonal in Q matrix:
for ff=1:1000
   diag(Q,ff);
   fwd_diag_counter = fwd_diag_counter +1;
end

% Count number of diagonals below main diagonal in Q matrix:
for rr=-1000:-1
   diag(Q,rr);
   rev_diag_counter = rev_diag_counter +1;
end

% Build row vector based on given Q matrix:

% Exception for case when k=1 or M=1, then assign all elements of Q to
% respective elements of A, since sum of reverse diagonals will yield same
% A matrix in this case:
if k==1
```

```
        A(1,1:M) = Q(1:k,1:M);

        % Convert A from column vector to row vector:
        %A = transpose(A);

elseif M==1
        A(1,1:k) = Q(1:k,1:M);

        % Convert A from column vector to row vector:
        %A = transpose(A);

else
    for ii=(-rev_diag_counter):fwd_diag_counter
        A(1,jj) = Sum(ii,Q);
        jj = jj + 1;
    end
end

% Eliminate entries from A matrix where sum of diagonal doesn't exist:
A = A(A~='E');

% Obtain A as column vector:
A = reshape(A, [numel(A), 1]);

return

end

% File Name: Sum.m
% Date Created: 10/19/2015
% Author: Asif Ahmed
% Description: Harshad Deshmane Thesis, Iterative ADMM Algorithm,
%              Section 3.3
% Reference1: Harshad Deshmane, “System Identification via Nuclear
%             Norm Regularization,” 2014.
% Reference2: Dr. Kenneth Bosworth, Notes, Idaho State University,
%             Fall 2015.
% Reference3: Stackoverflow, "http://stackoverflow.com/questions/18816841/obtaining-opposite-diagonal-
of-a-matrix-in-matlab," 2013.
% ____________________________________________________________________

function [ B, R ] = Sum( ii, Q )

% Compute sum of reverse diagonals:
B = sum(diag(flipud(Q),ii));

% Compute value of 'diag' function:
R = (diag(flipud(Q),ii));
% If 'diag' returns 'Empty' then assign B=E to eliminate these un-wanted
% entries in Hankstar.m:
F = isempty(R);
if F == 1
    B = 'E';
end

return
```

```
end
```

A4 –Nuclear Norm Function

```
% File Name: nucnorm.m
% Date Created: 10/21/2015
% Author: Asif Ahmed
% Description: Harshad Deshmane Thesis, Iterative ADMM Algorithm,
%            Section 3.3
% Reference1: Harshad Deshmane, "System Identification via Nuclear
%            Norm Regularization," 2014.
% Reference2: Dr. Kenneth Bosworth, Notes, Idaho State University,
%            Fall 2015.
% ____________________________________________________________________

function [ r ] = nucnorm( A )

% compute nuclear norm as sum of singular values of matrix A:
r = sum(svd(A));

return

end
```

A5 –Augmented Lagrangian Function

```
% File Name: AugLagrangian.m
% Date Created: 10/21/2015
% Author: Asif Ahmed
% Description: Harshad Deshmane Thesis, Iterative ADMM Algorithm,
%            Section 3.3
% Reference1: Harshad Deshmane, "System Identification via Nuclear
%            Norm Regularization," 2014.
% Reference2: Dr. Kenneth Bosworth, Notes, Idaho State University,
%            Fall 2015.
% ____________________________________________________________________

function [ L ] = AugLagrangian( y, Z, lagrangemultiplier )

% Define variables:
global Gamma %Constant
global b % vec(y), meaning y as colum vector
global rho % Constant
global Uperp % orhtogonal basis of U

% Performing part of the computation to substitute in main function below:
T = hank(y)*Uperp;

% Compute Augmented Lagrangian:
L = Gamma*nucnorm(Z) ...
   + 0.5*(norm((y-b), 2)) ...
   + trace(transpose(lagrangemultiplier)*(T - Z))...
   + (rho/2)*(norm((T - Z),'fro'));
```

```
return

end
```

A6 –Gradient of Lagrangian Function

```
% File Name: GradLagr.m
% Date Created: 10/26/2015
% Author: Asif Ahmed
% Description: Harshad Deshmane Thesis, Iterative ADMM Algorithm,
%              Section 3.3
% Reference1: Harshad Deshmane, "System Identification via Nuclear
%             Norm Regularization," 2014.
% Reference2: Dr. Kenneth Bosworth, Notes, Idaho State University,
%             Fall 2015.
% ____________________________________________________________________

function [ G ] = GradLagr( y, Z, lagrangemultiplier )

% Define variables:
global b % vec(y), meaning y as colum vector
global rho; % Constant
global Uperp % orhtogonal basis of U

% Compute Gradient of Lagrangian:
G = rho*Hankstar((((hank(y)*(Uperp)) - ...
   (Z-((rho^-1)*lagrangemultiplier)))*(Uperp'))) + (y-b);

return

end
```

A7 –Get Data Function

```
% File Name: getdata.m
% Date Created: 10/14/2015
% Author: Asif Ahmed
% Description: Harshad Deshmane Thesis, Iterative ADMM Algorithm,
%              Section 3.3
% Reference1: Harshad Deshmane, "System Identification via Nuclear
%             Norm Regularization," 2014.
% Reference2: Dr. Kenneth Bosworth, Notes, Idaho State University,
%             Fall 2015.
% ____________________________________________________________________

function [ getdata_out ] = getdata()

   % Define variables:
   global m
   global p
   global N
   global u
   global y_hat
   global y
```

```
    % Using pseudo-random input from Simulink:
    % [ If Simulink Model is changed, can update the .mat files by using
    % 'save' eg: save iodataactual.mat ]

    %-----------------------------------------------------------------------
    % SystemIdentification0208G Simulink
    % ([0.7071s + 4.2426] / [s^2 + 0.750s + 0.125])
    load iodataactual0208G.mat;
    load OutputEstimated0208G.mat;
    load OutputMeasured0208G.mat;
    y = Output0208G; % actual output
    u = Input0208G; % actual input
    y_hat = OutputMeasured0208G; % measured output - y_bar
    %-----------------------------------------------------------------------

    % assign size of u matrix to m rows and N1 colums:
    [m,N1] = size(u);
    % assign size of y matrix to p rows and N2 colums:
    [p,N2] = size(y);

    % Check if number of columns of u is not equal to y
    if N1 ~= N2
        error('error - # of y and u observed must agree')
    else
        N = N1
    end

    return
end
```

A8 –Inner & Outer Line Search Function

```
% File Name: OuterLineSearch2.m
% Date Created: 11/16/2015
% Author: Asif Ahmed
% Description: Harshad Deshmane Thesis, Iterative ADMM Algorithm,
%              Section 3.3
% Reference1: Harshad Deshmane, "System Identification via Nuclear
%             Norm Regularization," 2014.
% Reference2: Dr. Kenneth Bosworth, Notes, Idaho State University,
%             Fall 2015.
% ____________________________________________________________________

function [ e_i, ynew, iter_total ] = OuterLineSearch2( yinit, Z, lagrangemultiplier )
%       [ e_i, ynew, iter_total ] = OuterLineSearch2( y, Z, lagrangemultiplier )

% This function is used to perform steps 1, 2 and 3 of Iterative ADMM
% Algorithm

% Define Variables:
global maxitinner
global stoptol
global rho
global Uperp
```

```
global Gamma
global maxitouter

% counter for total number of inner-loop iterations:
count_inner_total = 0;
% counter for total number of inner-loop and outer-loop iterations:
iter_total = 0;

% initialize computation_time for inner & outer line search as zero:
computation_time = 0;

% Initializations:
Z_old = Z;
lagrangemultiplier_old = lagrangemultiplier;

% Defining yold, gold, ynew, itcount, and tol for inner-linesearch:
yold = yinit;
gold = GradLagr( yinit, Z_old, lagrangemultiplier_old );
ynew = yold - (0.1*gold);
itcount = 1;
itcount_outer = 1;
tol = norm(gold,2);

% Outer While Loop is for Outer-ADMM-Iteration, and inner While Loop is for
% Inner-Line-Search_Iteration:
while (itcount_outer < maxitouter)

    % Start stopwatch timer:
    tic

    % increment outer-loop counter:
    itcount_outer = itcount_outer + 1;

    %-----------------------------------------------------------------
    % Inner (l+1) loop:
    while (itcount < maxitinner) && (tol > stoptol)

      % increment inner-loop counter:
      itcount = itcount + 1;
      % calculate gnew:
      gnew = GradLagr( ynew, Z_old, lagrangemultiplier_old );
      % calculate tolerance:
      tol = norm(gnew, 2)
      % calculate alpha:
      alpha = dot(ynew - yold, gnew - gold) / ...
            dot(gnew - gold, gnew - gold);
      % assign gold=gnew for next inner-loop computation:
      gold = gnew;
      % assign yold=ynew for next inner-loop computation:
      yold = ynew;
      % Calculate ynew:
      ynew = ynew - alpha*gnew;

    end

    % increment count_inner_total used for computing total
```

```
    % number of iterations:
    count_inner_total = count_inner_total + itcount;

    %-----------------------------------------------------------------
    % Outer (k+1) loop:

    % Performing part of the computation to substitute in main function
    % below:
    T = hank(ynew)*Uperp;

    % Singular-Value-Decomposition:
    [W, Sgma, Vtrnspse] = svd(T + (rho^-1)*lagrangemultiplier_old);

    %-----------------------------------------------------------------
    % Calculate Z_new by including a factor of 25 times (Gamma/rho).
    % It is necessary to have Z_new calculate a non-zero result.
    % The value of 25 was determined through running the algorithm
    % for varying values of this factor. Keeping rho fixed at 1, this
    % factor could be regarded as value of Gamma in the algorithm that
    % gives best results for estimate of y compared to actual y from
    % Simulink. This is for SystemIdentification Simulink System0208G:
    % ([0.7071z + 4.2426] / [z^2 + 0.750z + 0.125]):
    Z_new = W*max(Sgma - (25*(Gamma/rho)), 0)*Vtrnspse;
    %-----------------------------------------------------------------

    % lagrangemultiplier:
    lagrangemultiplier_new = lagrangemultiplier_old + rho*(T - Z_new);

    % Assign lagrangemultiplier_old and Z_old to be used in next
    % computation of inner (l+1) above:
    lagrangemultiplier_old = lagrangemultiplier_new;
    Z_old = Z_new;

    % Set itcount=1 and tol=0.1 to force inner loop to execte again:
    itcount=1;
    tol=0.1;

    % Count total number of iterations:
    iter_total = count_inner_total + itcount_outer;

end
% Stop stopwatch timer:
computation_time = toc

% Build hankel matrix for ynew:
% global Y_new
Y_new = hank(ynew);

% Identification Error:
e_i = 100*((norm((ynew - yinit), 'fro'))/(norm((yinit), 'fro')));

% Plotting ynew and yinit to compare y before iterations to y after
% iterations:
subplot(5,1,1)
plot(ynew)
title('ynew (y-estimated)')
```

```
subplot(5,1,2)
plot(yinit)
title('yinit (y-actual)')
y1=ynew;
y2=yinit;
subplot(5,1,3)
plot(y1, 'g--')
hold on
plot(y2, 'b:')
hold off
title('ynew (y-estimated) in green, yinit (y-actual) in blue')
subplot(5,1,4)
plot(abs((y2) - (y1)))
title('Error: yinit - ynew')
subplot(5,1,5)
plot(((abs((y2) - (y1)))./(abs(y2)))*100)
title('Error-Percent: ((yinit-ynew)/(yinit))*100')

return

end
```

A9 –Simulation & Estimation Program

```
% File Name: SimulationEstimation.m
% Date Created: 01/22/2016
% Author: Asif Ahmed
% Description: Harshad Deshmane Thesis, Iterative ADMM Algorithm,
%              Section 3.3
% Reference1: Harshad Deshmane, "System Identification via Nuclear
%             Norm Regularization," 2014.
% ____________________________________________________________________

%% ITERATIVE-ADMM & ACTUAL-SIMULINK-OUTPUT COMPARED:
%  System0208G: ([0.7071z + 4.2426] / [z^2 + 0.750z + 0.125])

% Need to run SIMULINK model, and save variables in order to compare
% actual output to estimated output using era2.m function. Results from
% OuterlineSearch2.m and era2.m appear to be closely matched, but may not
% be exactly the same. Difference is caused by using sample time (Ts) of
% 0.1 in era2.m function to give transfer function closest to actual
% transfer function.

load iodataactual0208G.mat;
load OutputEstimated0208G.mat;

% Plotting y and IterativeADMM estimated y:
subplot(5,1,1)
plot(OutputEstimated0208G)
title('y-estimated')
subplot(5,1,2)
plot(Output0208G)
title('y-actual')
y1=OutputEstimated0208G;
y2=Output0208G;
```

```
subplot(5,1,3)
plot(y1, 'g--')
hold on
plot(y2, 'b:')
hold off
title('y-estimated in green, y-actual in blue')
subplot(5,1,4)
plot(abs((y2) - (y1)))
title('Error: y-actual - y-estimated')
subplot(5,1,5)
plot(((abs((y2) - (y1)))./(abs(y2)))*100)
title('Error-Percent: ((y-actual-y-estimated)/(y-actual))*100')

%% TABU-SEARCH & ITERATIVE-ADMM COMPARED:
%  System0208G: ([0.7071z + 4.2426] / [z^2 + 0.750z + 0.125])

load TS0208G_&_ITADMM_01-22-2016.mat;

% Plotting ynew (from OuterLineSearch2.m) and bestx (from Tabu Search):
subplot(5,1,1)
plot(bestx')       % bestx is transposed to obtain it as column vector
title('bestx-TS')
subplot(5,1,2)
plot(ynew)
title('ynew-IterativeADMM')
y1=bestx';
y2=ynew;
subplot(5,1,3)
plot(y1, 'g--')
hold on
plot(y2, 'b:')
hold off
title('bestx-TS in green, ynew-IterativeADMM in blue')
subplot(5,1,4)
plot(abs((y2) - (y1)))
title('Error: ynew-IterativeADMM - bestx-TS')
subplot(5,1,5)
plot(((abs((y2) - (y1)))./(abs(y1)))*100)
title('Error-Percent: ((IterativeADMM - bestx-TS)/(bestx-TS))*100')

%Create hankel matrix for bestx to be used in era.m:
Bestx_new = hank(bestx);

%% TABU-SEARCH & ACTUAL-SIMULINK-OUTPUT COMPARED:
%  System0208G: ([0.7071z + 4.2426] / [z^2 + 0.750z + 0.125])

load TS0208G_&_ITADMM_01-22-2016.mat;
load iodataactual0208G.mat;

% Plotting y (from Simulink) and bestx (from Tabu Search):
subplot(5,1,1)
plot(bestx')       % bestx is transposed to obtain it as column vector
title('bestx-TS')
subplot(5,1,2)
plot(Output0208G)
title('y-actual')
```

```
y1=bestx';
y2=Output0208G;
subplot(5,1,3)
plot(y1, 'g--')
hold on
plot(y2, 'b:')
hold off
title('bestx-TS in green, y-actual in blue')
subplot(5,1,4)
plot(abs((y2) - (y1)))
title('Error: y-actual - bestx-TS')
subplot(5,1,5)
plot(((abs((y2) - (y1)))./(abs(y2)))*100)
title('Error-Percent: ((y-actual-bestx-TS)/(y-actual))*100')

%% Actual input-output data converted to impulse response for era.m
%  System0208G: ([0.7071z + 4.2426] / [z^2 + 0.750z + 0.125])

% Need to input a discrete time impulse response h for era.m:

% load variables:
load iodataactual0208G.mat;

% Create iddata object:
z = iddata(Output0208G, Input0208G, 1);

% Create transfer function estimate with 2 poles and 1 zeros, with
% io-delay=1, Ts=1:
model = tfest(z, 2, 1, 1, 'Ts', 1);

% Convert model to discrete:
% model = c2d(model, 1);

% Obtain impulse response:
[y_impulse, t] = impulse(model);

% y_impulse is column vector. Transpose to get as row vector:
y_impulse = y_impulse';

%% TS results converted to impulse respose to be used in era function:
%  System0208G: ([0.7071z + 4.2426] / [z^2 + 0.750z + 0.125])

% Need to input a discrete time impulse response h for era.m:

% load variables:
load iodataactual0208G.mat;
load TS0208G_&_ITADMM_01-22-2016.mat;

% Create iddata object (use bestx' to have it as column vector, same as
% Input):
z1 = iddata(bestx', Input0208G, 1);

% Create transfer function estimate with 2 poles and 1 zeros, with
% io-delay=1, Ts=1:
model1 = tfest(z1, 2, 1, 1, 'Ts', 1);
```

```
% Convert model to discrete:
% model1 = c2d(model1, 1);

% Obtain impulse response:
[y_impulse1, t1] = impulse(model1);

% y_impulse is column vector. Transpose to get as row vector:
y_impulse1 = y_impulse1';

%% IterativeADMM results converted to impulse respose for era function:
%  System0208G: ([0.7071z + 4.2426] / [z^2 + 0.750z + 0.125])

% Need to input a discrete time impulse response h for era.m:

% load variables:
load iodataactual0208G.mat;
load TS0208G_&_ITADMM_01-22-2016.mat;

% Create iddata object:
z2 = iddata(ynew, Input0208G, 1);

% Create transfer function estimate with 2 poles and 1 zeros, with
% io-delay=1, Ts=1:
model2 = tfest(z2, 2, 1, 1, 'Ts', 1);

% Convert model to discrete:
% model2 = c2d(model2, 1);

% Obtain impulse response:
[y_impulse2, t2] = impulse(model2);

% y_impulse is column vector. Transpose to get as row vector:
y_impulse2 = y_impulse2';
```

A10 –Eigen System Realization Function

```
% File Name: era2.m
% Date Created: 01/21/2016
% Author: Asif Ahmed
% Description: Harshad Deshmane Thesis, Iterative ADMM Algorithm,
%            Section 3.3
% Reference1: Harshad Deshmane, “System Identification via Nuclear
%           Norm Regularization,” 2014.
% Reference3: Mathworks, "http://www.mathworks.com/matlabcentral/fileexchange/12848-eigensystem-
realization-algorithm/content/era.m", 2006.
% Reference4: Samuel da Silva, UNICAMP, 2006
% Reference5: Juang, J. N. and Phan, M. Q. "Identification and Control of
%           Mechanical Systems", Cambridge University Press, 2001
% Note:      This function is originally the creation of Samuel da Silva
%           from Reference3, Reference4, Reference5, and has been adapted
%           to be used for system identification academic research
% ____________________________________________________________________

function [A,B,C,D]=era2(h1,n1,N1,Ts,def)
```

```
% Call function as follows:
% System0208G: ([0.7071z + 4.2426] / [z^2 + 0.750z + 0.125])
% Actual input-output data: [A,B,C,D]=era2(y_impulse,2,22,0.1,1)
% TS: [A,B,C,D]=era2(y_impulse1,2,22,0.1,1)
% IterativeADMM: [A,B,C,D]=era2(y_impulse2,2,22,0.1,1)
%
% Eigensystem Realization Algorithm (ERA)
%
% [A,B,C,D]=era(h,n,N,Ts,def);
%
% Inputs:
%    h1: discrete-time impulse response
%    n1: order of system
%    N1: number of samples to build Hankel matrix
%    Ts: sample time
%    def: if = 1: the output will be the discrete-time state-space model
%         if = 2: the output will be the continuous-time state-space model
%
% Otputs:
%    [A,B,C,D]: state-space model
%

% Square hankel matrix construction:
H0 = hankel(h1(2:N1+1));          % k = 0
H1 = hankel(h1(3:N1+2));          % k = 1

% Factorization of the Hankel matrix by use of SVD
[R,Sigma,S] = svd(H0);

% Sigman obtained with respect to order of system:
Sigman = Sigma(1:n1,1:n1);

% observability matrix:
Wo = R(:,1:n1)*Sigman^0.5;
% controllability matrix:
Co = Sigman^.5*S(:,1:n1)';

% Compute system matrices:
% dynamic matrix:
A = Sigman^-.5*R(:,1:n1)'*H1*S(:,1:n1)*Sigman^-.5;
% input matrix:
B = Co(:,1);
% output matrix:
C = Wo(1,:);
% direct-transmission matrix:
D = h1(1);

% Obtain discrete-time system:
sysdisc = ss(A,B,C,D,Ts);

% Obtain continuous-time system:
if def == 2
   % Conversion of discrete LTI models to continuous time:
   syscont = d2c(sysdisc,'zoh');
   % continuous-time system:
   [A,B,C,D]=ssdata(syscont);
```

```
end

%-----------------------------------------------------------------------

% Obtain discrete time-transfer function from A, B, C, D matrices, and
% evaluate eigen-values of A matrix:
[num11, den11] = ss2tf(A, B, C, D);
tf11 = tf(num11, den11, 0.1)
eig_A = eig(A)
```

APENDIX B

B1 - Simple Tabu Search Main Program

```
% File Name: simpleTS.m
% Date Created: 01/29/2016
% Author: Asif Ahmed
% Description: Simple Tabu Search Algorithm
% Versions: Version 1.0, April 11, 2010
% Reference1: Dr. Marco P. Schoen, Idaho State University, 2010.
% Note:      This function is originally the creation of Dr. Marco P.
%           Schoen, and has been adapted to be used for system
%           identification academic research
% ____________________________________________________________________

clear;
% Add promising list and promising balls - tabu
% Add adaptive neighborhood sphere size in aspiration part

% Define input parameters
input('Nominal 0, or specific 1: ');spec=ans;
if spec==0
   nx=2;;maxx=10;minx=0;tll=5;pll=5;r=2.5/100*(maxx-minx);r1=r/2;r3=r1/2;n1=25;x=[rand*(maxx-
minx)+minx,rand*(maxx-minx)+minx];nd=100;ftheor=-18.5547;Sopt=[9.0389 8.6674];
   funchoice=12;
else
   %input('Dimension of solution nx: ');nx=ans;
   %input('Length of Tabu List tll: ');tll=ans;
   %input('Length of Promising List pll: ');pll=ans;
   %input('Tabu Ball Radius r: ');r=ans;
   %input('Radius of Neighborhood r1: ');r1=ans;
   %input('Radius of Promising Balls r3: ');r3=ans;
   %input('Search space, maximum: ');maxx=ans;
   %input('Search space, minimum: ');minx=ans;
   %input('Number of Neighborhoods n1: ');n1=ans;
   %input('Random point in Search space: ');x=ans;
   %input('How many iterations to be carried out nd: ');nd=ans;

   input('Cost function: 1.Spher, 2.Quad, 3.Ackl, 4.Boha I, 5.Colv, 6.Eas, 7.Griew, 8.Hypel, 9.Rast,
10.Rosenb, 11.Schwl, 12.M:, 13.Deshmane ');funchoice=ans;

   if funchoice==1 % Spherical parameters
      nx=2;maxx=100;minx=-100;tll=5;pll=5;r=2.5/100*(maxx-minx);r1=r/2;r3=r1/2;n1=25;
      nd=100;ns=40;nd=100;ftheor=0;Sopt=[0 0];x=[rand*(maxx-minx)+minx,rand*(maxx-minx)+minx];
   elseif funchoice==2 % Quadric parameters
      nx=2;maxx=100;minx=-100;tll=5;pll=5;r=2.5/100*(maxx-minx);r1=r/2;r3=r1/2;n1=25;
      nd=100;ftheor=0;Sopt=[0 0];x=[rand*(maxx-minx)+minx,rand*(maxx-minx)+minx];
   elseif funchoice==3 % Ackley parameters
      nx=2;maxx=30;minx=-30;tll=5;pll=5;r=2.5/100*(maxx-minx);r1=r/2;r3=r1/2;n1=25;
      nd=100;ftheor=0;Sopt=[0 0];x=[rand*(maxx-minx)+minx,rand*(maxx-minx)+minx];
   elseif funchoice==4 % Bohachevsky I parameters
      nx=2;maxx=50;minx=-50;tll=5;pll=5;r=2.5/100*(maxx-minx);r1=r/2;r3=r1/2;n1=25;
      nd=100;ftheor=0;Sopt=[0 0];x=[rand*(maxx-minx)+minx,rand*(maxx-minx)+minx];
   elseif funchoice==5 % Colville parameters
```

```
    nx=4;tll=5;pll=5;r=0.25;r1=0.125;r3=0.06;n1=25;x=[5,5,5,5];nd=100;maxx=10;minx=-
10;ftheor=0;Sopt=[0 0 0 0];
  elseif funchoice==6 % Easom parameters
    nx=2;maxx=100;minx=-100;tll=5;pll=5;r=2.5/100*(maxx-minx);r1=r/2;r3=r1/2;n1=25;
    nd=100;ftheor=-1;Sopt=[3.1416 3.1416];x=[rand*(maxx-minx)+minx,rand*(maxx-minx)+minx];
  elseif funchoice==7 % Griewank parameters
    nx=2;maxx=600;minx=-600;tll=5;pll=5;r=2.5/100*(maxx-minx);r1=r/2;r3=r1/2;n1=25;
    nd=100;ftheor=0;Sopt=[0 0];x=[rand*(maxx-minx)+minx,rand*(maxx-minx)+minx];
  elseif funchoice==8 % Hyperellipsoid parameters
    nx=2;maxx=1;minx=-1;tll=5;pll=5;r=2.5/100*(maxx-minx);r1=r/2;r3=r1/2;n1=25;
    nd=100;ftheor=0;Sopt=[0 0];x=[rand*(maxx-minx)+minx,rand*(maxx-minx)+minx];
  elseif funchoice==9 % Rastigin parameters
    nx=2;maxx=5.120;minx=-5.120;tll=5;pll=5;r=2.5/100*(maxx-minx);r1=r/2;r3=r1/2;n1=25;
    nd=100;ftheor=0;Sopt=[0 0];x=[rand*(maxx-minx)+minx,rand*(maxx-minx)+minx];
  elseif funchoice==10 % Rosenbrock parameters
    nx=2;maxx=2.048;minx=-2.048;tll=5;pll=5;r=2.5/100*(maxx-minx);r1=r/2;r3=r1/2;n1=25;
    nd=100;ftheor=0;Sopt=[1 1];x=[rand*(maxx-minx)+minx,rand*(maxx-minx)+minx];
  elseif funchoice==11 % Schwefel parameters
    nx=2;maxx=500;minx=-500;tll=5;pll=5;r=2.5/100*(maxx-minx);r1=r/2;r3=r1/2;n1=25;
    nd=100;ftheor=0;Sopt=[0 0];x=[rand*(maxx-minx)+minx,rand*(maxx-minx)+minx];
  elseif funchoice==12% Marco
    nx=2;maxx=10;minx=0;tll=5;pll=5;r=2.5/100*(maxx-minx);r1=r/2;r3=r1/2;n1=25;
    nd=100;ftheor=-18.5547;Sopt=[9.0389 8.6674];x=[rand*(maxx-minx)+minx,rand*(maxx-
minx)+minx];
  elseif funchoice==13 % Deshmane
    % Start stop-watch timer:
    tic

    nx=1001; % Initialize nx as 1001 - data points from Simulink
    maxx=100;
    minx=-100;
    tll=5;
    pll=5;
    r=2.5/100*(maxx-minx);
    r1=r/2;
    r3=r1/2;
    n1=25;
    nd=100;
    ftheor=0;
    Sopt=[0 0];
    % load Simulink data, and assign Actual Output to x. Transpose it
    % since it is necessary to have x as row vector for simpleTs.m and
    % cost.m:
    load iodataactual0208G.mat
    x = Output0208G';
  else
  end;
end;
TL=zeros(nx,tll); %TL(parameter 1, parameter 2, ..., tabu liste element 1, ...)

% main loop
xt(1,:)=x;clear x;x=xt;globalbestcost=cost(x(1,:),funchoice);%globalbestcost evaluation equals starting
point cost
counter=1;globalbestx(counter,:)=xt(1,:);
for k=1:nd
```

```
%create neighbhors
xneighbor=creatneighbors(r1,n1,nx,x(k,:));


% Select element from neighborhood
for d=1:n1
   xtest(d,:)=xneighbor(d,:);%pick one candidate element from each neighborhood
   costtest(d)=cost(xtest(d,:),funchoice); % and evaluate its cost
end;
[minimumcost,bestelement]=min(costtest);%find minimum cost and element
candidatex=xtest(bestelement,:);

% test if it is in tabu balls
flag=1;
flag=CheckifinBall(candidatex,TL,r,tll,nx);
if flag==0 % New position is in tabu ball
   if minimumcost>globalbestcost
      %'tabu'
      candidatex=xtest(n1,:);%use largest move
      minimumcost=cost(candidatex,funchoice);
   else
   end;
else
end;

 % use aspiration criteria
if minimumcost<globalbestcost
   candidatex=xtest(bestelement,:);
   minimumcost=cost(candidatex,funchoice);
else
   candidatex=x(k,:);
   minimumcost=cost(candidatex,funchoice);
   %change neigborhood
end;

% test if search limit reached
for ik=1:nx
   if candidatex(1,ik)>maxx
      candidatex(1,ik)=maxx;%no wrap around
      minimumcost=cost(candidatex,funchoice);
   elseif candidatex(1,ik)<minx
      candidatex(1,ik)=minx; %no wrap around
      minimumcost=cost(candidatex,funchoice);
   end;
end;
x(k+1,:)=candidatex; mincost(k)=cost(x(k+1,:),funchoice);
if mincost(k)<globalbestcost
   globalbestcost=mincost(k);
   counter=counter+1;
   globalbestx(counter,:)=x(k+1,:);
else
end;

% update Tabu list
% move each element one down
for jj=tll:-1:2
```

```
        TL(:,jj)=TL(:,jj-1);
    end;
    TL(:,1)=x(k+1,:);

    % what about promising list - best performances?
end;
if nx<3
    figure(1)
    plot(x(:,1),x(:,2));axis([minx maxx minx maxx]);grid;title('Position of Agent during course of
Simulation');
    xlabel('nx(1)');ylabel('nx(2)');
    figure(2)
    plot(mincost);grid;title('Minimum Cost');xlabel('Iteration Index');ylabel('Cost');
else
end
ftheor,Sopt
globalbestcost
bestx=globalbestx(counter,:)
% Stop stop-watch timer:
computation_time = toc
```

B2 - Cost Function

```
% File Name: cost.m
% Date Created: 01/29/2016
% Author: Asif Ahmed
% Description: Computation of the cost function
% Versions: July 9, 2009, Version 1.0
%          January 18, 2010, Version 1.1
% Reference1: Dr. Marco P. Schoen, Idaho State University, 2009.
% Note:       This function is originally the creation of Dr. Marco P.
%            Schoen, and has been adapted to be used for system
%            identification academic research
% ____________________________________________________________________

function [y] = cost(x,funchoice)

[n,nx]=size(x);y=0;

if funchoice==12 %Marco
    y=0;
    y = x(:,1)*sin(4*x(:,1))+1.1*x(:,2)*sin(2*x(:,2));    % fitness evaluation (objective function)
elseif funchoice==1 %Spherical
    y=0;
    for i=1:nx
        y=y+((x(:,i))^2);
    end;
elseif funchoice==2 %Quadric
    y=0;
    for i=1:nx
        yy=0;
        for kj=1:i
            yy=yy+x(:,i);
        end;
        y=y+yy^2;
```

```
    end;
elseif funchoice==3 %Ackley
    y=0; %*
elseif funchoice==4 %Bohachevsky
    y=x(1)^2+2*x(:,2)^2-0.3*cos(3*pi*x(:,1))-0.4*cos(4*pi*x(:,2))+0.7;
elseif funchoice==5 %Colville
    y=100*(x(2)-x(:,1)^2)^2+(1-x(:,1))^2+90*(x(:,4)-x(:,3)^2)^2+(1-x(:,3))^2+10.1*((x(:,2)-1)^2+(x(:,4)-
1)^2)+19.8*(x(:,2)-1)*(x(:,4)-1);
elseif funchoice==6 %Easom
    y=-cos(x(:,1))*cos(x(:,2))*exp(-1*(x(:,1)-pi)^2-(x(:,2)-pi)^2);
elseif funchoice==7 %Griewank
    y=0; %*
elseif funchoice==8 %Hyperellipsoid
    y=0;
    for i=1:nx
        y=y+(i^2)*(x(:,i))^2;
    end;
elseif funchoice==9 %Rastrigin
    y=0;
    for i=1:nx
        y=y+((x(:,i))^2)-10*cos(2*pi*x(:,i))+10;
    end;
elseif funchoice==10 %Rosenbrock
    y=0;
    %y=y+100*(x(2)-x(1)^2+(1-x(1)^2));
    for i=1:nx/2
        y=y+100*((x(:,2)-(x(:,1)^2))^2+(1-x(:,1))^2);
        %y=y+100*(x(i*2)-(x(i)^2))^2+(1-x(i*2-1)^2);%
    end
elseif funchoice==11 %Schwefel
    y=0;
    for i=1:nx
        y=y+x(:,i)*sin((abs(x(:,i)))^(0.5))+418.9829*nx;
    end;
elseif funchoice==13 %Deshmane

    % Cost Function: [have replaced y with x(:,i)] that is generated in
    % simpleTS.m.
    % Uperp is taken from input data from Simulink.
    % Since H(y) is function of y, it is possible to use hank(x).
    % b is measured output from Simulink.

    load iodataactual0208G.mat;
    load OutputMeasured0208G.mat;
    b = OutputMeasured0208G; % Simulink Measured Output (1001x1)

    % Create y1 as column vector from transpose of x:
    y1 = x'; % 1001x1

    u = Input0208G; % 1001x1
    U = hank(u); % 2x1000
    Uperp = null(U); % 1000x998

    % Choose Gamma as 25, determined to give best results in
    % IterativeADMM from OuterLineSearch2.m:
    Gamma = 25;
```

```
   % Determine y by cost function computation:
   y = (Gamma)*(nucnorm(hank(y1)*Uperp)) + 0.5*(norm((y1-b),2));

else
end;
```

B3 - Tabu Ball Verification Function

```
% File Name: CheckifinBall.m
% Date Created: 01/29/2016
% Author: Dr. Marco P. Schoen
% Description: Tabu Ball Verification Function
% Reference1: Dr. Marco P. Schoen, Idaho State University, 2009.
% Note:      This function is originally the creation of Dr. Marco P.
%           Schoen, and has been used for system identification academic
%           research
% ____________________________________________________________________

function flag = CheckifinBall(Sx,TL,r,tll,nx);
flag=0;NTS=zeros(tll);
for i=1:tll
   for t=1:nx
      NTS(i)=NTS(i)+((TL(t,i)-Sx(1,nx))^2);
   end;
   NTS(i)=(NTS(i))^0.5;
   if NTS(i)>=r %tabu if NTS < r
      flag=1; %not tabu
   else
   end;
end;
```

B4 - Creating Neighbors Function

```
% File Name: createneighbors.m
% Date Created: 01/29/2016
% Author: Dr. Marco P. Schoen
% Description: Computation of neighboors by creating nx-dimensional spheres
%            around current position x
% Versions: April 17, 2010, Version 1.0
% Reference1: Dr. Marco P. Schoen, Idaho State University, 2010.
% Note:      This function is originally the creation of Dr. Marco P.
%           Schoen, and has been used for system identification academic
%           research
% ____________________________________________________________________

function xneighbor=creatneighbors(r1,n1,nx,x)

   for d=1:n1 %for each neighborhood
      for j=1:nx %for each dimension
         ri(d,j)=(d-1)*r1; %inner diameter of neighborhood ball
         ro(d,j)=d*r1; % outer diameter of neighbor
         dif=ro(d,j)-ri(d,j);difr=dif*rand;
         rneighbor(d,j)=(ri(d,j)+difr)*sign(randn);
```

```
        theta(d,j)=rand(1)*2*pi;
        xcomp(d,j)=rneighbor(d,j)*cos(theta(d,j))+x(:,j);
    end;
    xneighbor(d,:)=xcomp(d,:);
end;
```