Conceptualizing Full and Reduced Order Linear Observers Using MATLAB GUI

Prajyot K. Meghrajani* and Ravindra K. Munje†
K. K. Wagh Institute of Engineering Education and Research, Nashik-422003, (M.S.), India.
E-mail: *prajyot.meg@gmail.com, †ravimunje@yahoo.co.in.

***Abstract*—This paper basically depicts the design of full and reduced-order observers using MATLAB software package and elaborates the scheme to implement linear time-invariant systems and observers, represented in state space domain, using the SIMULINK state space blocks for computerized control system modelling. Further, in pursuit of ease of implementation, a Graphical User Interface (GUI) is designed. Techniques for adopting the observer(s) initiatory states and their respective gain(s) are explained. Methodology to develop a GUI for facile execution of the system is elaborated. Exportability feature of feedback gain matrix and observer gain matrix as notepad files enhance the overall practicality. Thus as a whole, this paper is intended to design linear observers by means of GUI which further provides essential parametric data to practically deploy observers and observer dependent controllers in copious engineering and scientific operations.**

***Keywords*—Full order observer, Reduced order observer, Initial conditions, GUI, MATLAB.**

I. Introduction

The controller design often relies upon access to the state variables for state-feedback, through adjustable gains. This access can be provided by hardware. However, sometimes it is impractical to use this hardware for reasons of cost, accuracy or availability which means that some of the state variables may not be available at all, or it is too expensive to measure and send them to the controller. If state variables are not available because of the system configuration or cost, it is possible to estimate the states. The estimated states, rather than the measured states, are then fed to the controller. Such an estimator is thus termed as an observer [1]. In this paper, the technique to implement observers using MATLAB-SIMULINK software package is explained using the mathematical strategy suggested in [2] but with a different, user friendly executional approach. In contrast to the work proposed in [2], this paper intends to simplify the execution scheme by realizing a Graphical User Interface (GUI).

Theoretical analysis of observers at the undergraduate level may not sufficiently induce the logical insight to comprehend as well as distinguish between full and reduced order observers. Simulation techniques are thus adapted to generate graphical responses of the system further enabling visualization and conceptual justification of the modelled system. However, algorithms necessary to develop the simulations are monotonous which may divert the students from the gist of perceiving the concept and make them focus on the elaborate lines of codes. In comparison to the work presented in [3], the goal of this paper is to make the algorithm implementation strategy lucid by means of a GUI and thus create a standalone tool to design linear observers for SISO and MIMO systems. Hence this tool can serve as a teaching aid to obtain essential system parameters as well as graphical responses at the click of a button without compromising on student conceptual understanding [4]. This user friendly tool can, not only generate graphical responses but also can export vital system parameters as notepad files which makes it appealing for industrial applications as well. The amalgamation of the mathematical model with GUI to ensure ease of practical realizability is thus elaborated and explained by means of numerical examples.

The paper is organized as follows. Section II provides a brief overview of observer design. The main result of this paper i.e. development of GUI is discussed in Section III. Section IV presents numerical examples followed by conclusion in Section V.

II. Overview of Observer Design

Consider a linear time invariant (LTI) system depicted in state-space domain as

$$\dot{x}(t) = Ax(t) + Bu(t),\ x(0) = x_0 \tag{1}$$

where $x(t)$ is the state space vector of dimension n, $u(t)$ is the system input of dimension m, and matrices A and B are constants of suitable dimensions. Considering the aspects of stability, it may be favorable to place the closed-loop poles at strategic locations. This initiates the need to establish a feedback loop [5]-[7]. It is frequently assumed that all the state variables are accessible for feedback and thus the feedback control input can be implied as

$$u(x(t)) = -Fx(t) \tag{2}$$

where F is a constant feedback gain matrix of dimension $(m \times n)$. Full-state feedback controllers need all state variables for feedback, which as mentioned earlier may be impractical. As an alternative, the output signal, $y(t)$ can thus be availed and is represented as

$$y(t) = Cx(t). \tag{3}$$

In general, the dimension of the output signal is much smaller than that of the state variables, i.e. $dim\{y(t)\} = c < n$ where

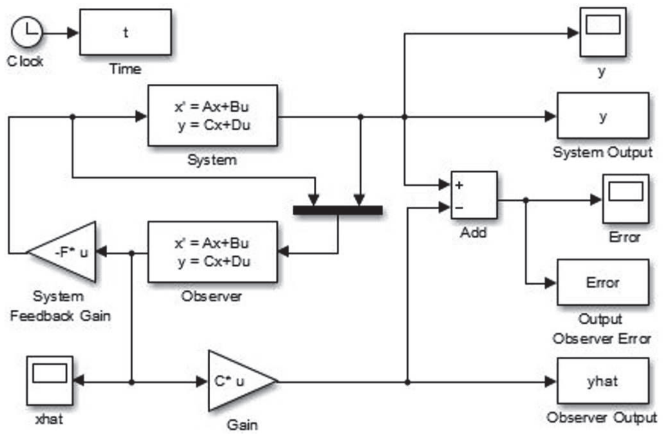

Fig. 1: SIMULINK implementation of full order observer [2].

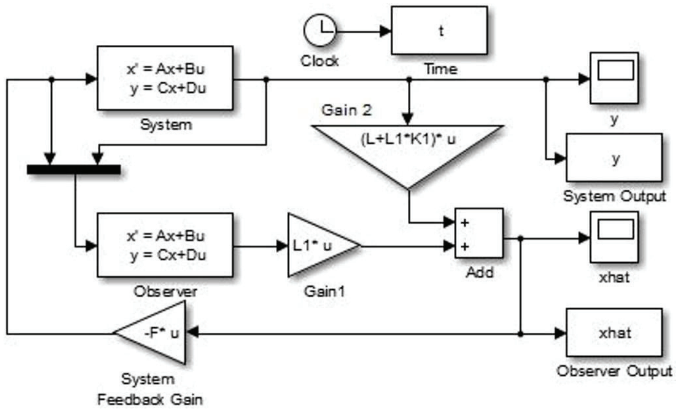

Fig. 2: SIMULINK implementation of reduced order observer [3].

$c = rank\{C\}$. Under such circumstances, an observer, which is basically a dynamic system impelled by the system output and input signals, with the sole objective to perpetually reconstruct system state variables can be employed [3]. In the following subsections, full order and reduced order observer executional aspects are discussed briefly.

A. Full Order Observer Implementation in SIMULINK

Nowadays, control engineers prefer the use of computers in lieu of electro-mechanical systems to develop observers that simulate the system and further solve the corresponding differential equations. An observer is structurally akin to the parent system but additionally comprises the driving feedback term that bears the particulars of the observation error. The observer is expressed as

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + K(y(t) - \hat{y}(t)) \tag{4}$$

where K is observer gain matrix. The observer, for practical implementation, is deployed as a linear dynamic system propelled by the parent system's output and input signals, i.e. $y(t)$ and $u(t)$. Substituting $\hat{y}(t)$ as $\hat{y}(t) = C\hat{x}(t)$ in (4) produces the observer form used in viable applications as

$$\dot{\hat{x}}(t) = (A - KC)\hat{x}(t) + Bu(t) + Ky(t). \tag{5}$$

In order to implement (5) in MATLAB, the SIMULINK state-space block is implied. However, this block permits only a single input vector and single output vector [3]. Hence, the observer in (5) is characterized as a system with a single augmented input as

$$\dot{\hat{x}}(t) = (A - KC)\hat{x}(t) + [B \quad K]\begin{bmatrix} u(t) \\ y(t) \end{bmatrix}. \tag{6}$$

The corresponding SIMULINK diagram for (6) is recognized as the observer based controller structure and is depicted in Fig. 1 with the closed feedback loop using the aforementioned feedback controller (2).

Further, exploring the 'System' state-space block in SIMULINK window, one can set up state space data as

$$\mathtt{A, B, C, zeros(c, m), x0}. \tag{7}$$

And, after exploring 'Observer' state-space block in SIMULINK window, one can set up state space data as

$$\mathtt{(A - K * C), [B \quad K], eye(n), zeros(n, m + c), x0hat} \tag{8}$$

in which $\hat{x}(0)$ can be obtained via the least-squares method [9] as

$$\hat{x}(0) = (C^T C)^{-1} C^T y(0) \tag{9}$$

where $y(0)$ is a known measured signal at $t = 0$.

B. Reduced Order Observer Implementation in SIMULINK

In the system (1), it is assumed that the output matrix C has full rank (i.e. c) so that there are no superfluous measurements. This part thus depicts the construction of an observer of reduced order for predicting the remaining $r = n - c$ state variables [c state variables can be deduced directly from the system measurements $y(t) = Cx(t)$]. As pointed out in [8], reduced order observers are dependable only when the system measurements have no disturbances. The following mathematical expressions are obtained from [3] and are used to develop the model. A random matrix C_1 of dimension $(r \times n)$ whose rank is equal to $r = n - c$ can be determined such that the augmented matrix

$$rank\begin{bmatrix} C^T & C_1^T \end{bmatrix}^T = n. \tag{10}$$

Further by introducing a vector $p(t)$ of dimension r such that $p(t) = C_1 x(t)$, the derivative of its estimate can be given as

$$\begin{aligned} \dot{\hat{p}}(t) \quad &= \quad C_1 A L_1 \hat{p}(t) + C_1 A L y(t) \\ &\quad + C_1 B u(t) + K_1(\dot{y}(t) - \hat{\dot{y}}(t)) \end{aligned} \tag{11}$$

where K_1 symbolizes the reduced-order observer gain. The final mathematical expression is

$$\dot{\hat{q}}(t) = A_q \hat{q}(t) + B_q u(t) + K_q y(t) \tag{12}$$

where

$$\begin{aligned} A_q &= (C_1 - K_1C)AL_1, B_q = (C_1 - K_1C)B, \\ K_q &= (C_1 - K_1C)A(L + L_1K_1), \hat{q}(t) = \hat{p}(t) - K_1y(t) \end{aligned}$$

While the 'System' state-space block remains same as that of (7), the 'Observer' state-space block gets modified as

$$\mathtt{A\_q}, [\mathtt{B\_q} \quad \mathtt{K\_q}], \mathtt{eye(n-c)}, \mathtt{zeros(n-c, m+c)}, \mathtt{x0hat\_red}.$$

Again, the full-order observer initial conditions [9] expressed in (9) can be extended to the reduced-order observer as

$$\delta\hat{q}(0) = (L_1^T L_1)^{-1} L_1^T [(C^T C)^{-1} C^T - (L + L_1K_1)] \delta y(0).$$

For detailed mathematical description, readers are advised to refer [7].

III. Development of Graphical User Interface

A user interface (UI) is a graphical display in one or more windows containing controls, called components, that enable a user to perform interactive tasks. A graphical user interface (GUI) can thus be termed as a pictorial interface to a program. The user does not have to create a script or type commands at the command line to accomplish the tasks. Unlike coding programs to accomplish tasks, the user does not need to understand the details of how the tasks are being performed. The three principal elements required to create a MATLAB Graphical User Interface are: 'Components', 'Figures' and 'Callbacks' [13]. Typically, UI's wait for the user to manipulate a control, and then respond to each user action in turn. Each control and the UI itself has one or more callbacks (named for the fact that they "call back" to MATLAB to ask it to do things). A particular user action, such as pressing a screen button, triggers the execution of each callback. The UI then responds to these events [11].

The basic motto of this paper is to design observers and thus obtain suitable values of feedback gain and observer gain matrices. In order to achieve this objective, 3 GUI's are developed. Each of them is associated with an exclusive 'callback' function to execute the next stage. Thus in accordance with the selection made by the user, different functions (which may be another GUI or some computational algorithm) are sequentially called so as to obtain the results.
The GUI is developed using MATLAB 'GUIDE'. This approach starts with a figure that can be populated with desired components from a graphic layout editor. GUIDE creates an associated code file containing callbacks for the UI and its components. GUIDE saves both, the figure (as a FIG-file) and the code file. The application can be launched from either file. The GUI is thus deployed as explained in the following stages.

A. *Stage 1: Selection of observer*

The first dialogue box which is made to appear is an option selection box with a static text and 2 push buttons. The static text basically serves as a label for the user. The upper push button is named as "Full Order Observer" and the lower is named as "Reduced Order Observer" as depicted in Fig. 3. Each push button is associated with a 'callback' function. The GUI in Fig. 3 is developed in *Graphic Layout Editor* of MATALB. After opening the editor, one 'static-text' and two 'push-button' tabs are selected from the *component pallet*. Double clicking on each component opens a window (termed as the *Property Inspector* or simply *Inspector*) with multiple editable features [14].

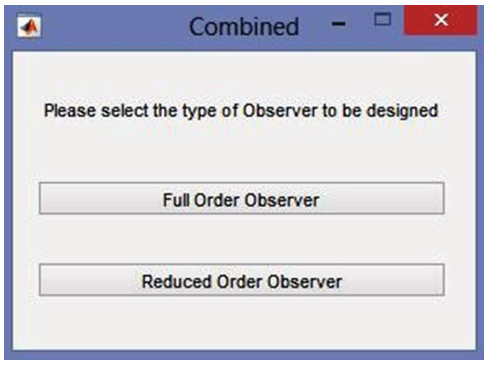

Fig. 3: Stage 1- Option selection GUI.

In the Inspector window of the 'static-text' the 'string' is edited as, "Please select the type of Observer to be designed". In the Inspector window of the 'pushbutton1' the string is edited as "Full Order Observer". Further the 'callback' box, is edited as "Full_Order_Observer_GUI". The Full_Order_Observer_GUI is basically a *.m* code file (or script file) which is called and executed when the pushbutton1 is pressed.
Similarly in the Inspector window of the 'pushbutton2' the string is edited as "Reduced Order Observer". Further the 'callback' box, is edited as "Reduced_Order_Observer_GUI". The Reduced_Order_Observer_GUI is again a *.m* code file (or script file) which is processed when the pushbutton2 is pressed.

B. *Stage 2.1: Full_Order_Observer_GUI*

When the user presses the upper push button, a *.m* file named as Full_Order_Observer_GUI is called. This script file is basically another GUI which is also developed in *Graphic Layout Editor* of MATALB as depicted in Fig. 4. The GUI depicted in Fig. 4 is constructed using 'static-text', 'pushbutton' and 'edit-text'.

C. *Stage 2.2 Reduced_Order_Observer_GUI*

When the user presses the lower push button, *.m* file named as Reduced_Order_Observer_GUI is called. This script file is one more GUI which is also developed in *Graphic Layout Editor* of MATALB as illustrated in Fig. 5. which is constructed using 'static-text', 'pushbutton' and 'edit-text'.

D. *Importing Data to Workspace From GUI*

The previously mentioned aspects pertaining to the GUI basically focussed on its structural development and layout.

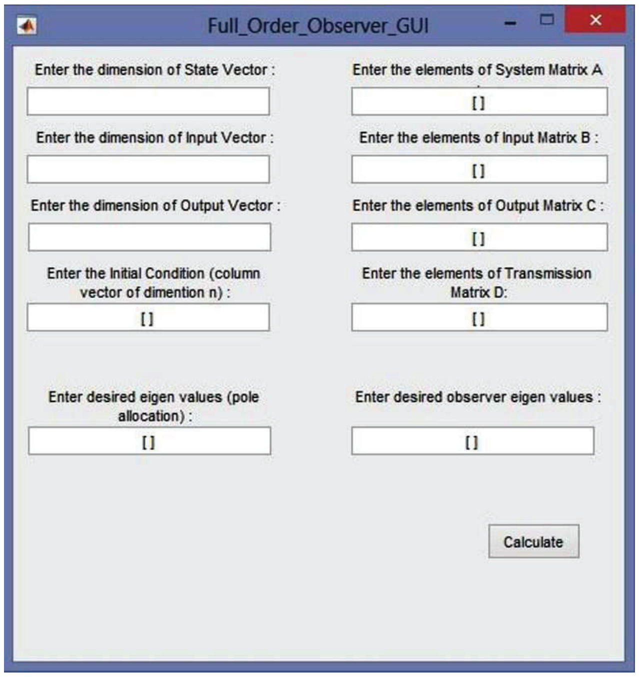
Full_Order_Observer_GUI
Enter the dimension of State Vector :
Enter the elements of System Matrix A
[]
Enter the dimension of Input Vector :
Enter the elements of Input Matrix B :
[]
Enter the dimension of Output Vector :
Enter the elements of Output Matrix C :
[]
Enter the Initial Condition (column vector of dimention n) :
[]
Enter the elements of Transmission Matrix D:
[]
Enter desired eigen values (pole allocation) :
[]
Enter desired observer eigen values :
[]
Calculate

Fig. 4: Stage 2.1- Full order observer GUI.

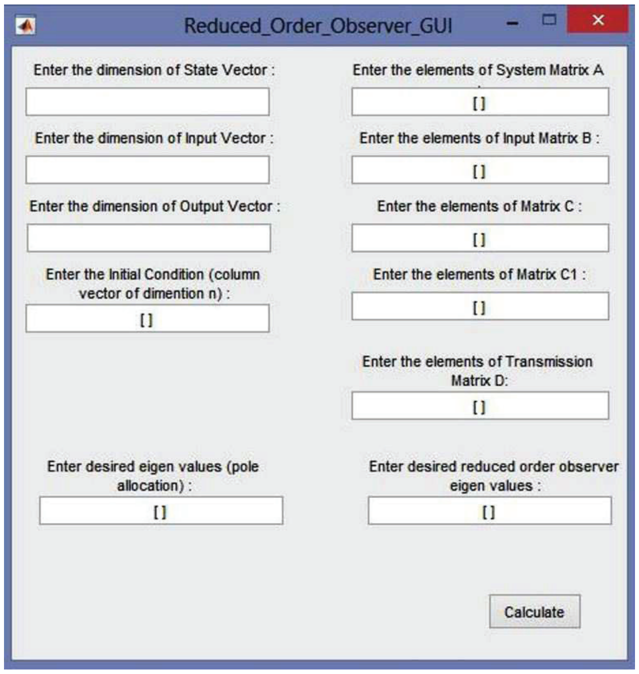
Reduced_Order_Observer_GUI
Enter the dimension of State Vector :
Enter the elements of System Matrix A
[]
Enter the dimension of Input Vector :
Enter the elements of Input Matrix B :
[]
Enter the dimension of Output Vector :
Enter the elements of Matrix C :
[]
Enter the Initial Condition (column vector of dimention n) :
[]
Enter the elements of Matrix C1 :
[]
Enter the elements of Transmission Matrix D:
[]
Enter desired eigen values (pole allocation) :
[]
Enter desired reduced order observer eigen values :
[]
Calculate

Fig. 5: Stage 2.2- Reduced order observer GUI.

However, after developing the layout, it is necessary to deploy the GUI's depicted in Fig. 4 and 5 so as to procure data from the user, further manipulate and process it to obtain the desired results. The most important aspect of doing so is to import the numerical data from GUI to the workspace.
For the user to enter the data, 'edit text' boxes (which can be chosen from the *component pallet*) are provided for respective parameters. Each text box is identified by a suitable 'tag' name. The tag name can be set in the Inspector window of the respective 'edit text' box. The data entered by the user in these edit-text boxes is interpreted by the GUI in ASCII format and is thus needed to be converted into numeric data. Thus 'str2num' command is used which converts the string 'str', which is an ASCII character representation of a numeric value, to numeric representation. 'str2num' also converts string matrices to numeric matrices [12]. Further in order to import data to workspace 'assignin(ws, 'var', val)' command is used which assigns the value 'val' to the variable 'var' in the workspace 'ws'. The 'var' input must be the array name only; it cannot contain array indices. If 'var' does not exist in the specified workspace, 'assignin' creates it. 'ws' can have a value of 'base' or 'caller' to denote the MATLAB base workspace or the workspace of the caller function. The algorithm below illustrates importing of parameter n which is the order of the system (or the dimension of the State Vector). In the Inspector window of this edit box, the box is tagged with name 'n'. Further, the following MATLAB code is implemented.

```
function n_Callback(hObject, eventdata, handles)
n=str2num(get(handles.n,'String'));
assignin('base','n',n)
```

Similarly other parameters viz. A, B, C etc. can be imported.

E. Computational Algorithms

Once all the parameters are entered and further imported to the workspace, suitable mathematical computations can be conducted to obtain the 'Feedback Gain Matrix' and 'Observer Gain Matrix' for Full and Reduced order observers. When the user presses the 'Calculate' push button, a *.m* MATLAB code file (or script file) is called which comprises of desired mathematical formulas. For full order observer, the script file is as follows:

```
 F=place(A,B,DEV)
 K_T=place(A',C',OEV);
 K=K_T'
 y0=C*x0;
 x0hat=pinv(C'*C)*C'*y0;
```

and for the Reduced Order Observer, the script file is as follows:

```
C_augm=[C;C1];
L_augm=inv(C_augm);
L=L_augm(1:n,1:c);
L_1=L_augm(1:n,c+1:n);
K_1T=place((C1*A*L_1)',(C*A*L_1)',ROEV);
K_1=K_1T'
A_q=C1*A*L_1-K_1*C*A*L_1;
B_q=C1*B-K_1*C*B;
K_q=C1*A*L_1*K_1+C1*A*L-K_1*C*A*L-K_1*C*A*L_1*K_1;
y0=C*x0;
x0hat_red=inv(L_1'*L_1)*L_1'*(pinv(C'*C)*C'-(L+L_1*
    K_1))*y0;
F=place(A,B,DEV);
```

F. Exporting Data From Workspace to Notepad Files

Here, it is ensured that the system is designed for practical realization. Thus it is necessary to export the 'Feedback Gain Matrix' and 'Observer Gain Matrix' out of MATLAB so that this parametric data can be loaded directly in industrial controllers. Thus these matrices can be exported to excel sheets or to notepad files. When the 'Calculate' pushbutton is pressed, along with computational *.m* file another script file is called which exports the parameters from workspace to notepad files as shown

```
save('Feedback Gain Matrix.txt','F','-ascii','-tabs'
    )
save('Observer Gain Matrix.txt','K','-ascii','-tabs'
    )
save('Reduced Order Observer Gain Matrix.txt','K_1',
    '-ascii','-tabs')
```

IV. Simulation Examples

The GUI and the SIMULINK models developed are practicable for SISO as well as MIMO control systems. The following numerical examples illustrate the graphical responses obtained upon deployment of the GUI.

A. Example 1: An Antenna Full and Reduced-Order Observer Design (azimuth position control)

This example is a SISO system [1]. Numeric data such as order of system, initial conditions, matrices A, B, C, D etc. are entered in the 'edit-text' boxes of respective GUI's. The data entered is:

```
n=3
Input_vector=1
Output_vector=1
Initial_condition=[0.006;0.006;0.006]
A=[0 1 0;0 0 1;0 -171 -101.71]
B=[0;0;1]
C=[1325 0 0]
D=[0]
Desired_eigen_values=[ -40,-4-5.458*j,-4+5.458*j]
Full_Order_Observer_eigen_values=[-400 -40-54.65*j
    -40+54.65*j]
C1=[0 1325 0;0 0 1325]
Reduced_order_observer_eigen_values=[-40-54.65*j
    -40+54.65*j]
```

After entering the above-mentioned parameters when the 'Calculate' pushbutton is pressed, the data is automatically imported into the workspace. It is further processed after which Feedback and Observer Gain Matrices are exported as notepad files in the same folder in which the GUI files are stored. As data is imported to the workspace, the SIMULINK models can be directly executed. Fig. 6 depicts the output response obtained from full and reduced order observer, while Fig. 7 compares the response of respective states.

It can be seen from the response of the states that the state $x1$ being already available for measurement is identical for full and reduced order observer as well. Minor deviations are witnessed in the estimated states of the reduced order as compared to full order.

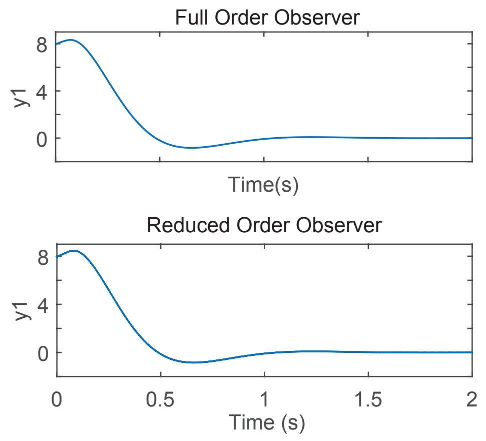

Fig. 6: Output 'y' of Full and Reduced order observer (Example 1).

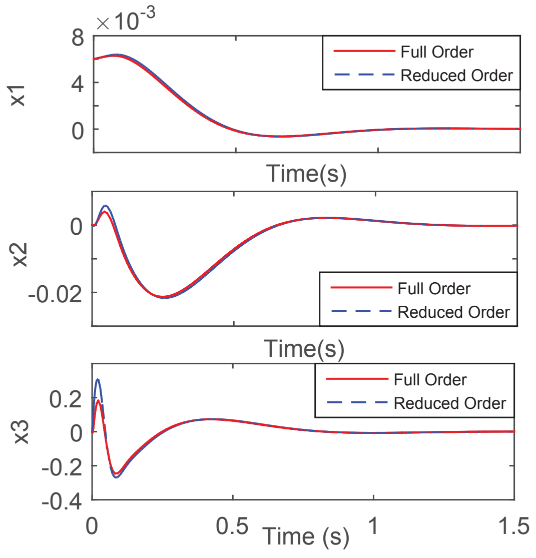

Fig. 7: Comparison of response of states (Example 1).

B. Example 2: An Aircraft Full and Reduced-Order Observer Design

This is the example of MIMO system [2]. System parameters are entered in the 'edit-text' boxes of respective GUI's as follows:

```
n=4
Input_vector=1
Output_vector=2
Initial_condition=[2;2;2;2]
A=[-0.01357 -32.2 -46.3 0;0.00012 0 1.214 0;
    -0.0001212 0 -1.214 1;0.00057 0 -9.1 -0.6696]
B=[-0.433;0.1394;-0.1394;-0.1577]
C=[0 0 0 1;1 0 0 0]
D=[0;0]
Desired_eigen_values=[-0.5,-1+1*j,-1-1*j,-2]
Full_Order_Observer_eigen_values=[-10 -11 -12 -13]
C1=[0 1 0 0;0 0 1 0]
Reduced_order_observer_eigen_values=[-10 -11]
```

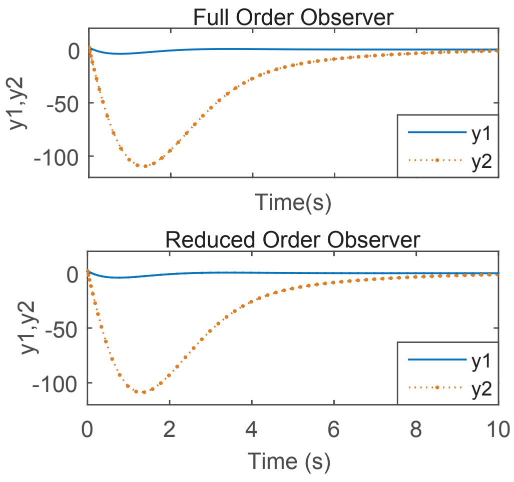

Fig. 8: Output ‘y’ of Full and Reduced order observer (Example 2).

After entering the above-mentioned parameters when the ‘Calculate’ pushbutton is pressed, the data being eventually imported into the workspace is processed and notepad files comprising of feedback gain and observer gain matrices are exported in the same folder in which the GUI files are stored. Further the SIMULINK models can be directly executed. Fig. 8 depicts the output response obtained from full and reduced order observer, while Fig. 9 compares the response of respective states. It can be seen from the response of the states that the states $x1$ and $x4$ being already available for measurement are identical for full and reduced order observers as well. Very little deviations can be observed in the estimated states ($x2$ and $x3$) of the reduced order as compared to full order.

V. Conclusion

This paper shows the technique to develop full and reduced order linear observers in MATLAB-SIMULINK. Further the procedure to develop a GUI which is amalgamated with the model is elaborated. Methodology to import numeric data to the workspace from GUI is explicitly stated and the strategy to use callback feature is emphasized. This technique is justified by considering examples of SISO and MIMO systems. From the simulation results, it can be seen that the measured states for full, as well as reduced order observers, are coincident, whereas there occurs slight deviation in the estimated states. However, this is not going to pose any difficulty in practical implementation and is not going to affect the performance of the system. Moreover, the reduced order observers can be termed to be more advantageous as they require lesser number of sensors as well as lesser loops of feedback which thus makes them economically viable. The procedure adopted in this paper serves as a vital teaching aid for students and can also be extended to non-linear dynamic systems. The feature to export Feedback Gain Matrix and Observer Gain Matrix as notepad files serve as a leverage for practical deployment of observers in industries. Thus as a whole, the blending of SIMULINK model with GUI and the capability to export essential parametric data makes the overall system alluring for industrial and process control applications.

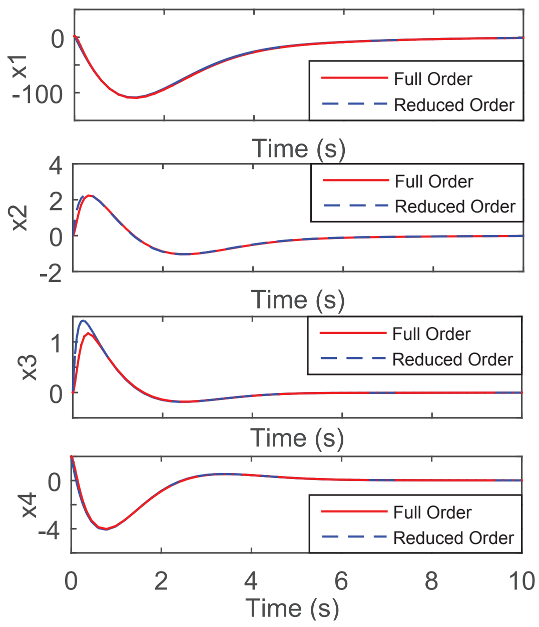

Fig. 9: Comparison of response of states (Example 2).

References

[1] N. Nise, *Control Systems Engineering* John Wiley and Sons, Hoboken New Jersey 2008, pp. 704-708.

[2] Verica Radisavljevic-Gajic, “Linear Observers Design and Implementation”, in *Proc. American Society For Engineering Education (ASEE Zone 1)*, IEEE, 2014, pp. 1-6.

[3] Verica Radisavljevic-Gajic, “Full-and Reduced-Order Linear Observer Implementations in MATLAB/SIMULINK”, IEEE Control Systems Magazine, vol. 35, pp. 91-101, 1066-033X/15 © 2015 IEEE.

[4] Vitor Fernão Pires and José Fernando A. Silva, “Teaching Nonlinear Modeling, Simulation, and Control of Electronic Power Converters Using MATLAB/SIMULINK” IEEE Transactions on Education, Vol. 45, No. 3, August 2002.

[5] D. Luenberger “Observing the state of a linear system” IEEE Transactions on Military Electronics Vol. 8 74-80 1964.

[6] D. Luenberger “Observers for multivariable systems” IEEE Transactions on Automatic Control Vol. AC-11 190-197 1966.

[7] D. Luenberger “An introduction to observers” IEEE Transactions on Automatic Control Vol. AC-16 596-602 1971.

[8] G. Franklin J. Powel and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, Prentice Hall, Upper Saddle River, 2002.

[9] C. D. Johnson “Optimal initial condition for full-order observers” International Journal of Control Vol. 48 857-864 1988.

[10] K. Ogata, *Modern Control Engineering*, Prentice Hall, Upper Saddle River, 2002.

[11] Creating Graphical User Interfaces with MALTAB, The MathWorks Inc, R2015b.

[12] MATLAB R2015b User Guide “Create a Simple UI Using GUIDE”.

[13] Balogh Tibor, Viliam Fedák, František Durovský “Modeling and Simulation of the BLDC Motor in MATLAB GUI” 978-1-4244-9312-8/11/26.00 © 2011 IEEE.

[14] Abdulla Ismail Abdulla, “Introduction to Graphical User Interface (GUI) MATLAB 6.5”, IEEE UAE Section.