Appendix 3B
MATLAB Functions for Modeling and Time-domain analysis

MATLAB control system Toolbox contain the following functions for the time-domain response

step	Step response.
impulse	Impulse response.
initial	Response of state-space system with given initial state.
lsim	Response to arbitrary inputs.
gensig	Generate input signal for LSIM.
damp	Natural frequency and damping of system poles.
ltiview	Response analysis GUI (LTI Viewer).

Given a transfer function of a closed-loop control system, the function **step(num, den)** produces the step response plot with the time vector automatically determined. If the closed-loop system is defined in state space, we use **step(A, B, C, D), step(A, B, C, D, t)** or **step(A, B, C, D, iu, t)** uses the supplied time vector **t**. The scalar **iu** specifies which input is to be used for the step response. If the above commands are invoked with the left-hand argument **[x, y, t]**, the output vector **y**, the state response **x**, and the time vector **t** are returned, and we need to use plot function to obtain the plot. Similarly for **impulse**, **initial**, and **lsim**.

Example B.1

Obtain the unit step response for the system with the following closed-loop transfer function

$$\frac{C(s)}{R(s)} = \frac{25(1+0.4s)}{(1+0.16s)(s^2+6s+25)}$$

Use the **damp** function to obtain the roots of the characteristic equation, the corresponding damping function, and natural frequencies

The following commands

```
num=25*[0.4  1];
den = conv([1  0.16], [1   6   25]);   % multiplies the two polynomials
step(num, den), grid                    % obtains the step response plot
T = tf(num, den)
damp(T)                                 % returns roots of C.E., damping ratio, wn
```

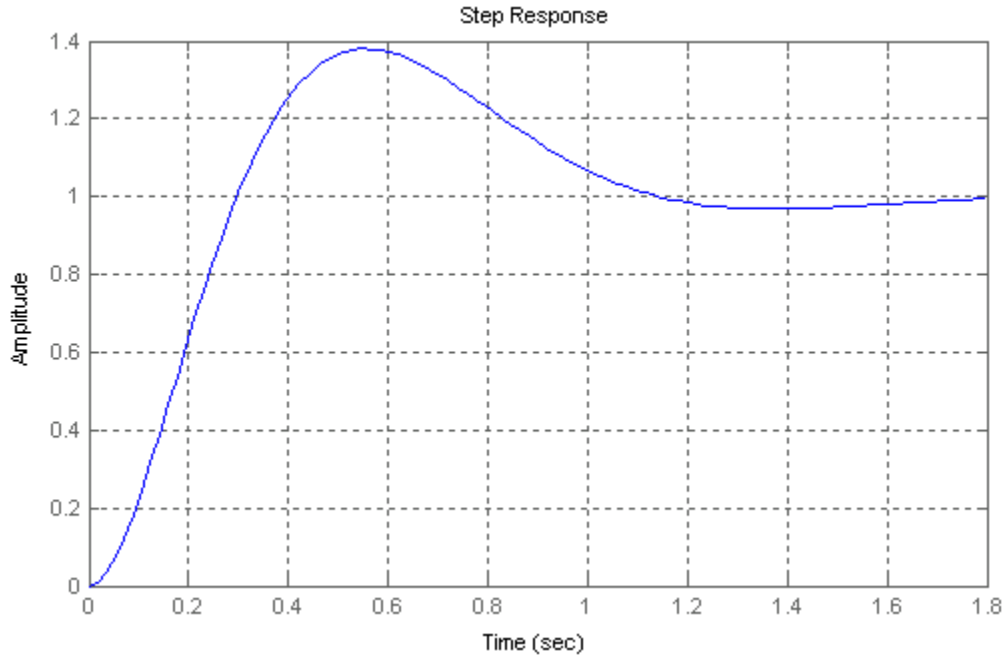

Figure B.1 Step response of Example 1

Eigenvalue	Damping	Freq. (rad/s)
-3.00e+000 + 4.00e+000i	6.00e-001	5.00e+000
-3.00e+000 - 4.00e+000i	6.00e-001	5.00e+000
-6.25e+000	1.00e+000	6.25e+000

Example B.2

The closed-loop transfer function of a control system is described by the following third-order transfer function

$$\frac{C(s)}{R(s)} = T(s) = \frac{750}{s^3 + 36s^2 + 205s + 750}$$

(a) Find the dominant poles of the system
(b) Find a reduced-order model
(c) Obtain the step response of the third-order system and the reduced-order system on the same figure plot

(a) The command

```
den = [ 1  36  205  750];
r = roots(den)
```

result in

```
r =
    -30
    -3  + 4i
    -3  -  4i
```

Therefore the transfer function is

$$T(s) = \frac{750}{(s+30)(s^2+6s+25)} = \frac{25}{(1+0.0333s)(s^2+6s+25)}$$

The time constant of the real pole at $s = -30$ is $\tau_1 = 1/30$ which is negligible compared to the time constant of $\tau_2 = 1/3$ for the dominant poles $-3 \pm j4$. Therefore the approximate reduced model transfer function is

$$T(s) \simeq \frac{25}{(s^2+6s+25)}$$

We use the following commands

```
num1 = 750;                  % third-order system num
den1 = [1  36   205   750];  % third-order system den
num2=25;                     % reduced-order system num
den2 = [1  6  25];           % reduced-order system den
t = 0: 0.01: 2;
c1 = step(num1, den1, t);    % third-order system step response
c2 = step(num2, den2, t);    % reduced-order system step response
plot(t, c1, 'b', t, c2, 'r') % plots both response on the same figure
grid, xlabel('t, seconds'), ylabel('c(t)')
legend('Third-order', 'Reduced-order')
```

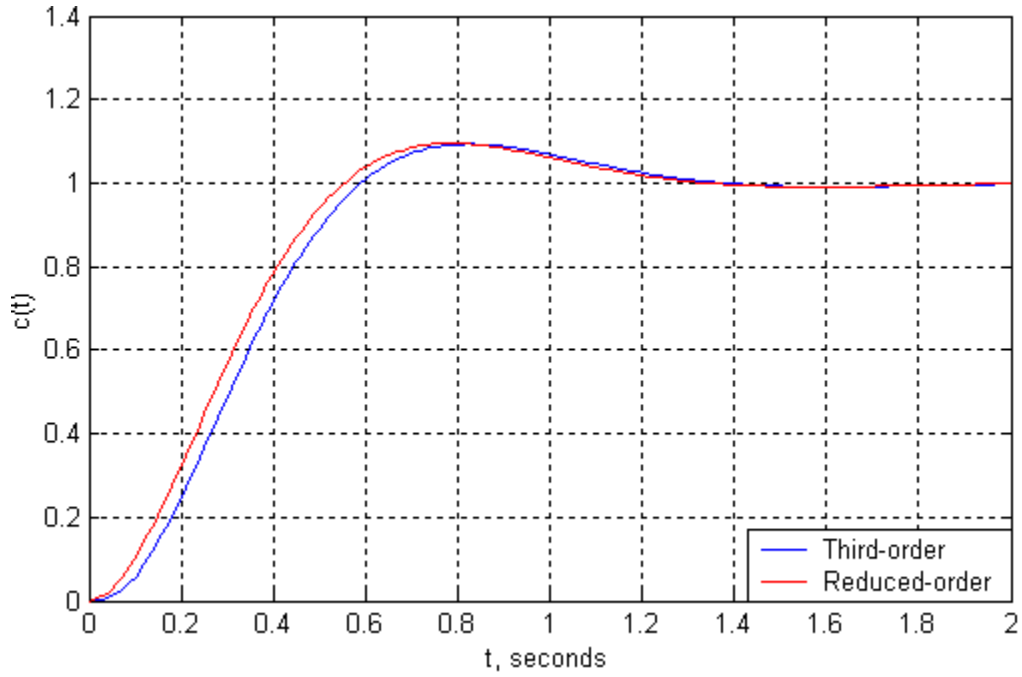

Figure B.2 Third-order and reduced-order Step response step responses of Example 1

The LTI Viewer

The Control System Toolbox LTI Viewer is a GUI that simplifies the analysis of linear, time-invariant systems. You use the LTI Viewer to view and compare the response plots of several linear models at the same time. You can generate time and frequency response plots to inspect key response parameters, such as rise time, maximum overshoot, and stability margins. Using mouse-driven interactions, you can select input and output channels from MIMO systems. The LTI Viewer can display up to six different plot types simultaneously, including step, impulse, Bode (magnitude and phase or magnitude only), Nyquist, Nichols, sigma, and pole/zero.

The command syntax is

ltiview('plot type', sys, extra)

where **sys** is the transfer function name and **'plot type'** is one of the following responses:

step	**bode**
impulse	**nyquist**
initial	**nichols**
lsim	**sigma**

Extra is an optional argument specifying the final time. Once an LTI Viewer is opened, the right-click on the mouse allows you to change the response type and obtain the system time-domain and frequency domain specifications, including:

Plot Type	changes the plot type
Systems	selects any of the models loaded in the LTI Viewer
Characteristics	displays key response characteristics and parameters
Zoom	zooms in and out of plot regions
Grid	adds grids to your plots
Properties	Property Editor, where you can customize plot attributes

Example B.3

Use the **ltiview** to obtain the step response and the time-domain specifications for the control system shown in Figure 3.

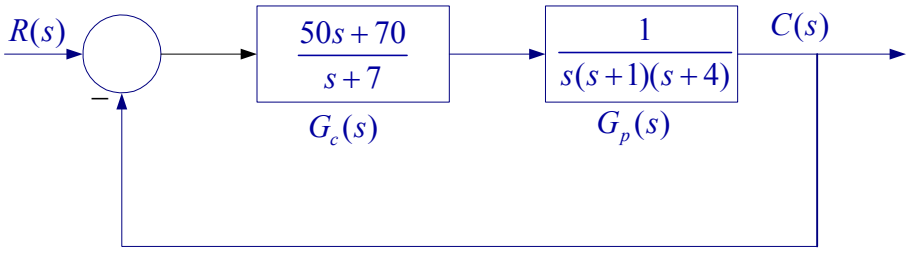

Figure B.3 Block diagram for Example 3

The following commands

```
Gc = tf([50  70], [1  7])        % transfer function Gc
Gp = tf([1], [1 5 4 0]);         % transfer function Gp
H = 1;
G = series(Gc, Gp)               % connects Gc & Gp in cascade
T = feedback(G, 1)               % obtains the closed loop transfer function
ltiview('step', T)
```

result in

Transfer function:

$$\frac{50\,s + 70}{s^4 + 12\,s^3 + 39\,s^2 + 78\,s + 70}$$

The system step response is obtained as shown in Figure 4. The mouse right-click is used to obtain the time-domain specifications. From **File** menu you can select **Print to Figure** option to obtain a Figure Window for the LTI Viewer for editing the graph.

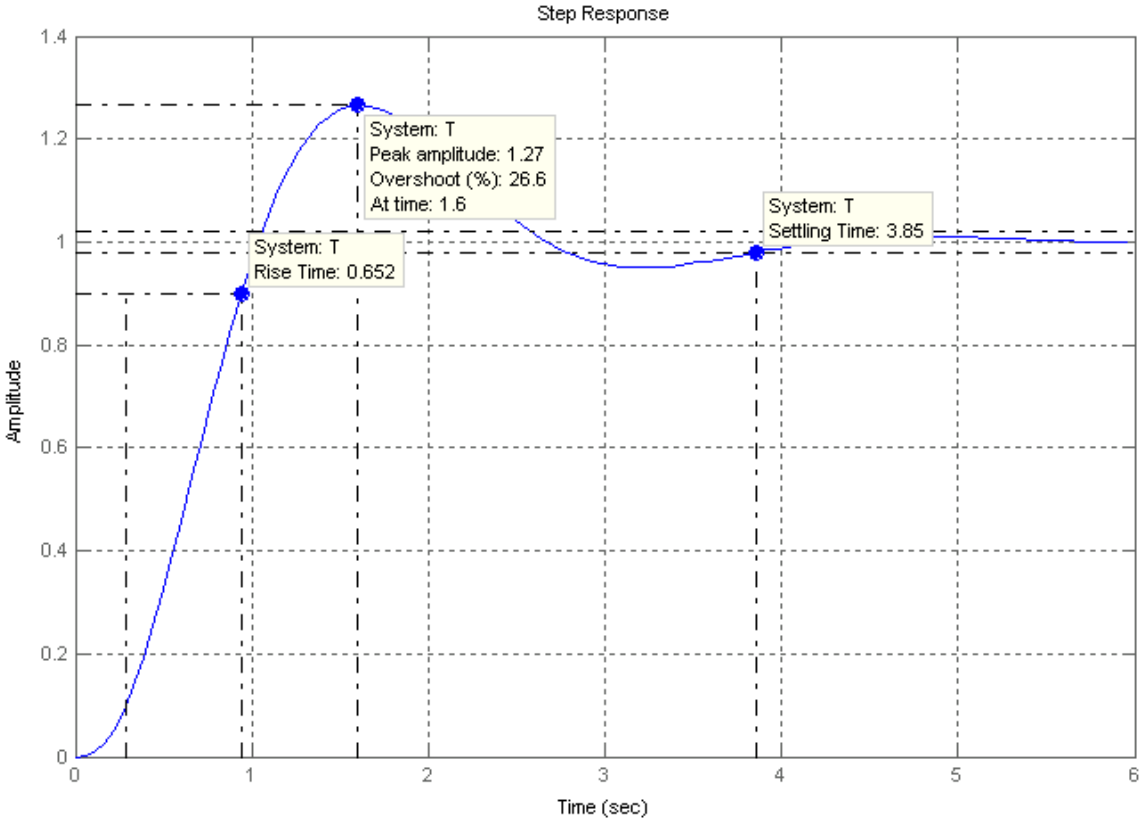

Figure B.4 Step response of Example B.3

For the time-domain analysis it is recommended to use the LTI Viewer, this will make it possible to obtain the time-domain specification with a simple right-click on the mouse.

In addition it allows you to select the **Plot type**, which would enable you to find other type of time-domain responses as well as the frequency domain responses and specifications.

MATLAB Functions for Numerical Solution of Differential Equations

There are many other more powerful techniques for the numerical solution of nonlinear equations. A popular technique is the Runge-Kutta method, which is based on formulas derived by using an approximation to replace the truncated Taylor series expansion. The interested reader should refer to textbooks on numerical techniques. MATLAB provides several powerful functions for the numerical solution of differential equations. Two of the functions employing the Runge-Kutta-Fehlberg methods are **ode23** and **ode45**, based on the Fehlberg second- and third-order pair of formulas for medium accuracy and forth- and fifth-order pair for higher accuracy. These functions are as follows:

ode23	Solve non-stiff differential equations, low order method.
ode45	Solve non-stiff differential equations, medium order method
ode113	Solve non-stiff differential equations, variable order method.
ode15S	Solve stiff differential equations and DAEs, variable order method.
ode23S	Solve stiff differential equations, low order method.
ode23T	Solve moderately stiff ODEs and DAEs, trapezoidal rule.
ode23TB	Solve stiff differential equations, low order method.

The *n*th-order differential equation must be transformed into *n* first order differential equations and must be placed in an M-file that returns the state derivative of the equations. The formats for these functions are

[t, x] = ode23('xprime', tspan, x0, option)

where **tspan =[t0, tfinal]** is the time interval for the integration and **x0** is a column vector of initial conditions at time **t0**. **xprime** is the state derivative of the equations, defined in a file named **xprime.m.** Commonly used options are scalar relative error tolerance 'RelTol' (1e-3 by default) and vector of absolute error tolerances 'AbsTol' (all components 1e-6 by default)

Example B.4

Using MATLAB function **ode23** obtain the numerical solution for the differential equation given by

$$\frac{d^2\theta}{dt^2} + \frac{B}{m}\frac{d\theta}{dt} + \frac{g}{l}\sin\theta = 0$$

The above equation describes the motion of the simple pendulum derived in Lab Session 4 Case Study2. Where $m = 0.5\,\text{Kg}$, $l = 0.613\,\text{m}$, $B = 0.05$ Kg-s/m, and $g = 9.81$ m/s^2. The initial angle at time $t = 0$ is $\theta(0) = 0.5$ and $\dot{\theta}(0) = 0$.

First we write the above equation in state variable form. Let $x_1 = \theta$, and $x_2 = \dot{\theta}$ (angular velocity), then

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = -\frac{B}{m}x_2 - \frac{g}{l}\sin x_1$$

The above equations are defined in a function file named pendulumeq.m as follows

```
function xdot = pendulumeq(t, x);  % Returns the state derivative
m = 0.5; l = 0.613; B = 0.05; g = 9.81;
xdot = [x(2); -B/m*x(2)-g/l*sin(x(1))];
```

In a separate file named Lab3ExB4.m, the MATLAB function ode23 is used to obtain the solution of the given differential equations (defined in the file pendulumeq.m from 0 to 20 seconds.

```
tspan = [0, 20];                          % time interval
x0 = [0.5; 0];                            % initial condition
[t, x] = ode23('pendulumeq', tspan, x0);
theta = x(:, 1);omega = x(:, 2);
figure(1), plot(t, theta, 'b', t, omega, 'r'), grid
xlabel('t, sec'), legend('\theta(t)', '\omega(t)')
figure(2), plot(theta, omega);
xlabel('\theta, rad'), ylabel('\omega rad/s')
title('State trajectory')
```

Run Lab3ExB4 at the MATLAB prompt, the result is

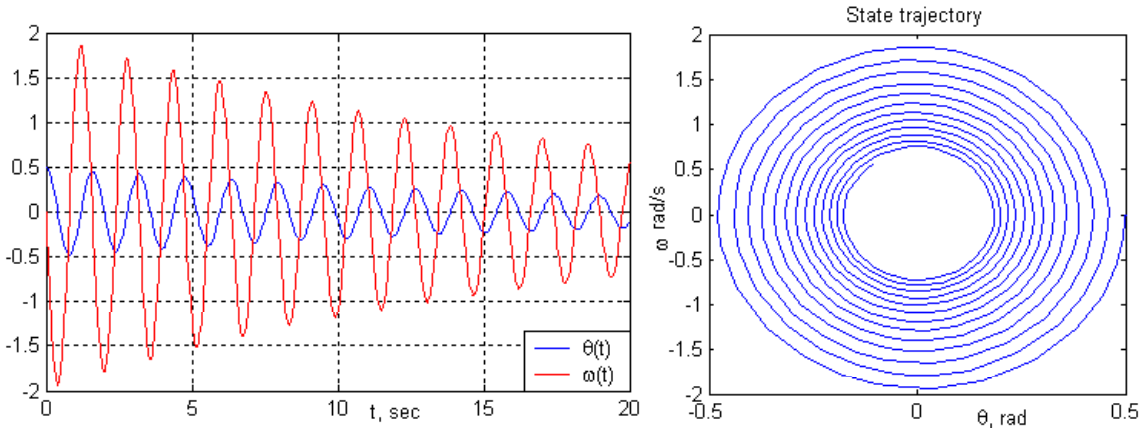

Figure B.5 Response of the pendulum described in Example B.4

Example B.5

Using MATLAB function **ode23** obtain the numerical solution for the nonlinear simultaneous differential equation given by

$$-x_1 + 2x_2 + [20 - 40*\cos(0.02*t)]\frac{dx_1}{dt} + 30*\sin(0.02*t)\frac{dx_2}{dt} = 13*\exp(-t)*\sin(t)$$

$$3x_1 - 4x_2 + [20*\sin(0.02*t)]\frac{dx_1}{dt} + [10 - 40*\cos(0.02*t)\frac{dx_2}{dt} = 40*\exp(-0.2t)*\cos(t)$$

Given $x_1(0) = 1$, and $x_2(0) = -1$

Writing the above equations in matrix form, we have

$$\begin{bmatrix} -1 & 2 \\ 3 & -4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 20 - 40\cos 0.02t & 30\sin 0.02t \\ 20\sin 0.02t & 10 - 40\cos 0.02t \end{bmatrix}\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 13e^{-t}\sin t \\ 40e^{-0.2t}\cos t \end{bmatrix}$$

Writing in compact form

$$RX + L\dot{X} = V$$

Solving for $\dot{X}$, we get

$$\dot{X} = L^{-1}[V - RX] \quad \text{(Note the correction)}$$

The above equations are defined in a function file named nlseq.m as follows

```
function xdot = nlseq(t, x);       % Returns the state derivative
R = [-1  2; 3  -4]; L = [20-40*cos(0.02*t)       30*sin(0.02*t)
                         20*sin(0.02*t)       10-40*cos(0.02*t)];
V = [13*exp(-t)*sin(t);  40* exp(-0.2*t) *cos(t)];
xdot = inv(L)*(V - R*x);
```

In a separate file named Lab3ExB5.m, the MATLAB function ode23 is used to obtain the solution of the given differential equations (defined in the nlseq.m from 0 to 30 seconds

```
tspan = [0, 30];   % time interval
x0 = [1; -1];      % initial condition
[t, x] = ode23('nlseq', tspan, x0);
plot(t, x), grid
xlabel('t, sec'), legend('x_1(t)', 'x_2(t)')
```

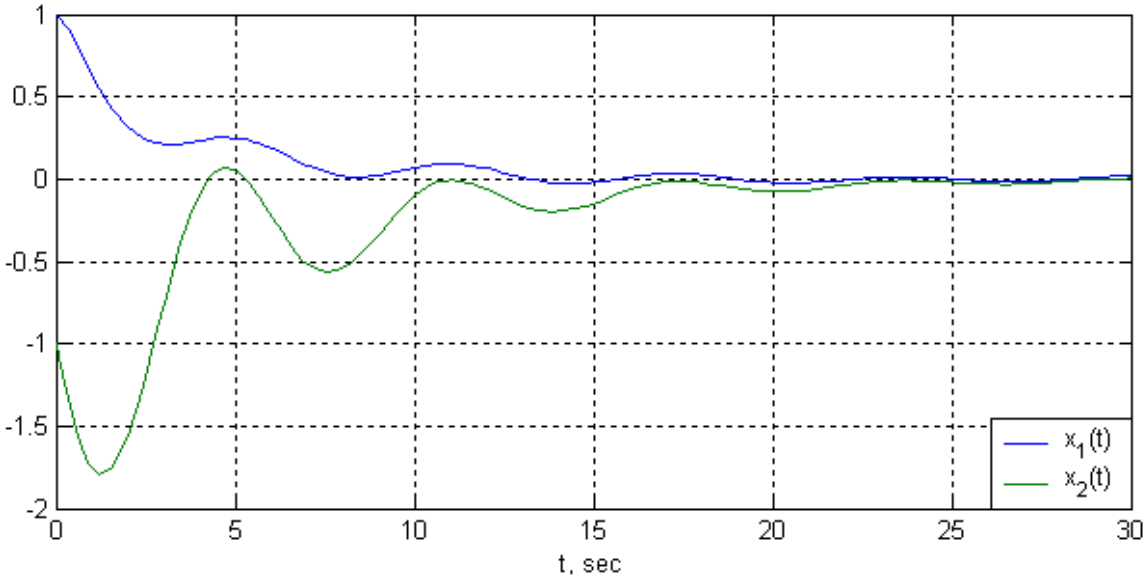

Figure B.6 Response of the system described in Example B.5