Session 2520

Interactive Continuous and Discrete Step Response Using MATLAB GUI's

Robert DeMoyer
United States Naval Academy

Abstract

It is important in the study of classical feedback control to understand the correspondence between pole locations and time response. Quadratic poles are particularly important because many systems can be approximately characterized by a quadratic. This paper describes a program, written in MATLAB and using the Graphical User Interface (GUI), which permits the user to drag a quadratic pole in the s-plane and observe, in real time, the changing continuous step response. The program provides the student with the means to actually observe concepts presented in class, including s-plane lines of constant overshoot, time to peak, and settling time.

A second program is described which allows the student to drag a quadratic pole in the z-plane and to observe the corresponding discrete step response. In addition to overshoot, time to peak, and settling time, the number of samples per cycle are easily seen as the z-pole is moved. The sample time can be changed while the program is in execution. Before describing these programs, the paper discusses aspects of GUI program development.

I. Introduction

Many engineering faculty members have taught long enough to see computer programming evolve from batch processing, to timesharing, to PCs with character based operating systems, to PCs with windows based operating systems. In the earlier evolutions it was easy to write code, frequently in FORTRAN or Basic, to serve as an aid to engineering education. In the latest evolution there are many useful commercial programs written for engineering education, but most engineering faculty still would like to write programs of their own.

Programs written for window based operating systems usually contain Graphical User Interfaces, or GUIs. Someone contemplating writing code to produce GUIs might suspect that it is a complicated process, and it is. There are some languages which help to reduce the complication, such as Visual Basic and Visual C. However, these languages require programming at a rather low level. Frequently specialized libraries must be acquired as well. The capability of a high level language, such as MATLAB along with its toolboxes, is hard to ignore.

Along with MATLAB 4 came a number of demonstration programs with GUIs. The potential of combining GUIs with the enormous capability of MATLAB is enticing. However, an examination of the code for these programs is daunting. The code is largely devoted to the

handle graphics required for the GUIs. An examination of the parts of the manuals pertaining to handle graphics shows that such programming is indeed very complicated.

New with MATLAB 5 is the Graphical User Interface Development Environment, or GUIDE, which facilitates GUI programming. Simply put, when using GUIDE the user drags GUIs into position within a figure and then facilitates the writing of typical MATLAB code to produce the desired results. What GUIDE ultimately produces is a file of handle graphics code described above.

The following sections contain a brief description of handle graphics, a description of GUIDE, the application of GUIDE to a simple linear system example, and then the description of a modified approach to the use of GUIDE to carry out more complicated linear system applications. Because the relevant manuals and references [3,4] cover hundreds of pages, this is not meant to be a complete description. The hope is that this introduction to GUIDE will encourage the reader to make further investigation into this relatively easy means to write programs with GUIs along with the high level capabilities of MATLAB.

II. An Introduction to Handle Graphics

MATLAB employs handle graphics. Each low level graphic component is referred to as an object and is assigned a numerical value called a handle. The nature of an object is specified by a set of properties. A quick impression of graphic objects, and how to determine their properties and change their properties, can be obtained by the following MATLAB instructions:

```
>>figure(1);
>>t=0:0.1:10;
>>plot(t,sin(t));
>>title('sin(t)');
>>xlabel('Time, seconds')
```

The get function displays the properties of an object. The handle of a figure is the figure number, so get(1) lists the properties of figure 1. The result of this instruction, which is too lengthy to reproduce here, consists of forty properties unique to the figure object type followed by a blank line and then fifteen more properties common to all objects.

Objects are organized in a tree structure. The root object, whose handle is 0, is the computer screen. A figure is a 'child' of the root, and the root in turn is the 'parent' of the figure. Upon examining the results of get(1) it can be seen that the 'Parent' property of figure 1 is [0] and the 'Children' property is something like [9.00049]. The figure's tag, or descriptive character string name, is 'Fig1'. Moving down, get(9.00049) reveals an object whose tag is 'Axes1'. This object leads to several objects, other than its 'Children', whose tags are 'Title', 'Xlabel', 'Ylabel', and 'Zlabel'. The 'Title' handle is 10.0004, so get(10.0004) shows a 'String' property whose value is 'sin(t)', the title assigned by the MATLAB code above. The 'Children' property of 'Fig1' is [7.00049], which is the handle of the plotted line, whose tag is 'Line1'.

The set instruction may be used to change properties. For example, in the first of the following instructions it is determined that the 'Color' property of the line is [0 0 1] which means [0% red 0% green 100% blue]. The following instruction changes the color to [1 0 0], or all red.

```
>>get(7.00049,'Color')
        Color = [0 0 1]
>>set(7.00049,'Color',[1 0 0]);
```

User interface control, or uicontrol, objects are used for programs with graphical user interfaces such as push buttons, sliders, or editable text. An important property of a uicontrol is the 'Callback' which consists of MATLAB code which is executed when the object is selected by a mouse click. In the following section we will see how GUIDE is used to place uicontrols on a figure and to change the callback property of uicontrols.

III. An Introduction to GUIDE

The instruction >>guide brings up the GUIDE Control Panel. GUIDE consists of a set of tools designed for rapid coding of GUIs [1]. The Property Editor tool displays object properties and facilitates changes to properties. The Callback Editor tool facilitates writing multi-line callback code. The Alignment Tool is used to align uicontrols within the figure, and the Menu Editor tool is used to develop pull-down menus. GUIDE also contains an Object Palette consisting of uicontrols which can be graphically placed into a controlled figure.

A simple application of GUIDE is shown in figure(1). It is a unit step response of a quadratic transfer function specified by natural frequency and damping ratio. Numerical values are placed in editable text objects, the computation is begun by pressing the Plot button, and the window is closed by pressing the Done button. The following steps trace the GUIDE aided development of this application:

1. Start GUIDE (>>guide). Figure 1 is created in the controlled state.
2. From the New Object Pallet place the text boxes, edit boxes, push buttons, and the axis on the figure.
3. Use the Alignment Tool to horizontally align the objects.
4. Select the top text box, by clicking on it. Use the Property Editor to set 'String' property to 'Wn'. Do the same for 'zeta'.
5. Select the upper edit box. Use the Callback Editor to make the callback instruction:

```
Wn=eval(get(gcbo,'String'))
```

Here, the string value of the callback object is gotten and converted to a numerical value.

6. Likewise, make the callback instruction:
   zeta=eval(get(gcbo,'String'))
   for the lower edit box.
7. Select the upper button. Use the Property Editor to set 'String' to 'Plot' and the Callback Editor to create the following callback:
   num=Wn^2;
   den=[1 2*zeta*Wn Wn^2];
   step(num,den);grid
8. Select the lower push button. Set the 'String' property to 'Done' and the callback to:
   close(gcbf).
9. Within the Control Panel, change the figure from controlled to active. Save as example1, producing example1.m and example1.mat.
10. Enter numerical values for Wn and zeta, and watch the results.

The file example1.m written by GUIDE contains twelve objects containing an average of six properties which have been changed from their default values. GUIDE has clearly saved a lot of work, even for an experienced GUI programmer.

IV. Continuous Step Response

A more interesting example, which utilizes animation, is shown in figure(2). On the right is a unit step response of a transfer function containing a zero, quadratic poles, and a gain normalized for unity final value. On the left is the s-plane showing the zero and the second quadrant quadratic pole. When the program is in execution, as the zero or pole is dragged into different positions, the step response is continuously updated. This program nicely illustrates what can be a difficult concept, the relation between s-plane poles and zeros and the corresponding time response. This program is made available to the students, through the network, as a study aid. The logic required is summarized as follows:

```
if (mouse button down)
  if (mouse in motion)
    if (pole selected)
      recompute transfer function
      replot pole 'x' in s-plane
    end
    if (zero selected)
      recompute transfer function
      replot zero 'o' in s-plane
    end
    recompute step response
    replot step response
```

```
  end
end
if (mouse button released)
  clear button down code
end
```

Several callbacks are required. It is desirable to put them into functions so that the compiled function code will execute faster than interpreted script file code and so that variables used will not remain in the MATLAB workspace. Figure(3) is the function file containing the callbacks required to implement the logic. By use of the switch statement multiple callbacks may be placed in a single function using a process called switchyard callbacks [2].

For a simple case like the previous example GUIDE produces two files. One is the .m file containing object definitions and property modifications. The other is a .mat file containing arrays of numeric data. More complex callbacks in an application such as the current example require a third file containing the switchyard callbacks. I recommend the addition of a fourth file, the one which is invoked to run the program. It also contains global variable initialization and key object properties. The initial values of variables can be stored in various object properties, but it is more convenient to see them all in one place in the file. The reason for including key object properties in the first file is that this file can serve as a starting point if major revisions are required. If object properties are modified entirely interactively through GUIDE, it can be a lengthy process to recreate the initial steps for a major revision. The source of these object properties can be the manuals, previous experimentation with GUIDE, and examination of the code which produces the many MATLAB demonstration programs. The last statement of this file, which is put in place after all development is complete, invokes the .m file written by GUIDE.

File polezero.m in figure(4) is the initial file for this example. It initializes global step response parameters and sets up the s-plane and time response axes. Initial plots are made with the critical 'Erasemode' property of 'xor' required for animation. The bulk of the GUI work is still done by GUIDE when the figure is saved, as pz.m and pz.mat in this case.

V. Discrete Step Response

Another interesting example is the correspondence between quadratic z-plane poles and a unit step response, as shown in figure(5). When the program is in execution the quadratic pole is dragged within the unit circle and the step response changes almost instantly. An additional feature of this program is an editable text box in which the sample time is specified. This value can be changed at any time the program is running.

The z-transfer function of a pure quadratic is more complicated than its s-plane counterpart because it contains a zero. An easy way to compute this transfer function is to have MATLAB compute it, using the following sequence:

1. As the mouse is dragged in the z-plane, the current z quadratic pole location is determined.
2. Using $z=e^{sT}$, the pole location is mapped into the s-plane.
3. The s-plane transfer function, Gs, of a pure quadratic is determined for the current s-plane quadratic poles. This is easy because there is no zero.
4. The discrete transfer function, Gz, which has the same step response as Gs at the sample times, is computed by Gz=c2d(Gs,T,'zoh'). The correct zero for Gz is computed as well as its poles.

It is interesting for the students to run this program. They can see that the overshoot remains constant as the pole is dragged along the constant zeta lines provided by the zgrid function. As the pole is dragged along a semicircle it can be seen that settling time remains constant. Finally, if the pole is dragged along a radial line from the origin, it can be seen that the number of samples per cycle remains constant. The number of samples per cycle is 360 degrees divided by the angle to the pole. In figure(5) this angle is about 45 degrees so the number of samples per cycle is 8, which is easily seen in the step response. Some texts which show quadratic z-poles and the corresponding step responses erroneously show the same number of samples per cycle for poles at all locations in the z-plane.

The discrete counterparts of the files shown in figure(3) and figure(4) are somewhat lengthy. Rather than include them here they are made available at the web address shown below.

VI. Conclusions

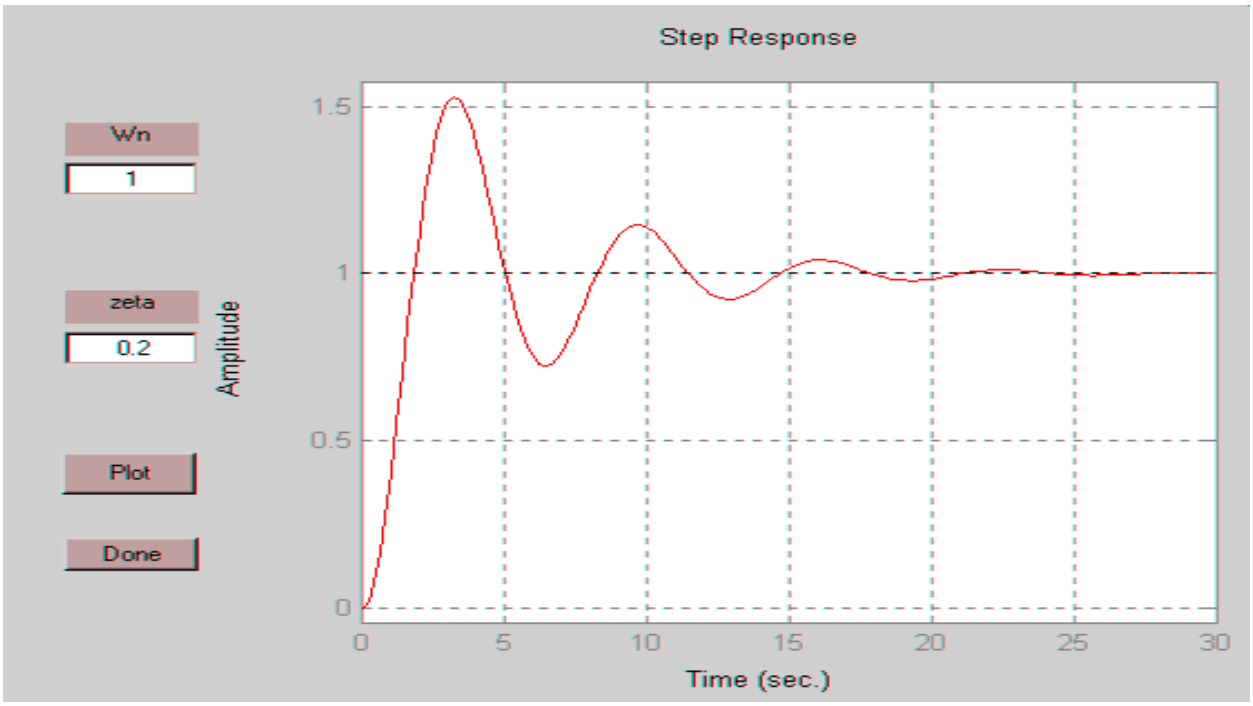
GUIDE is an excellent resource to help a user to become acquainted with the GUI capabilities of MATLAB. While it may not be sufficient, by itself, for more complex applications, it still can do the bulk of the GUI work. It is an extremely useful tool to help engineering educators to create new GUI applications or to attach GUI front ends to existing MATLAB applications.

The step response of a continuous quadratic transfer function with a zero illustrates essentially a pure quadratic if the zero is dragged far to the left of the pole, and then illustrates the effect of the zero as it is dragged into the vicinity of the pole. The step response of the discrete quadratic shows the not only the effect of changing the pole location but the effect of changing sample time. Both programs are useful study tools for the students.

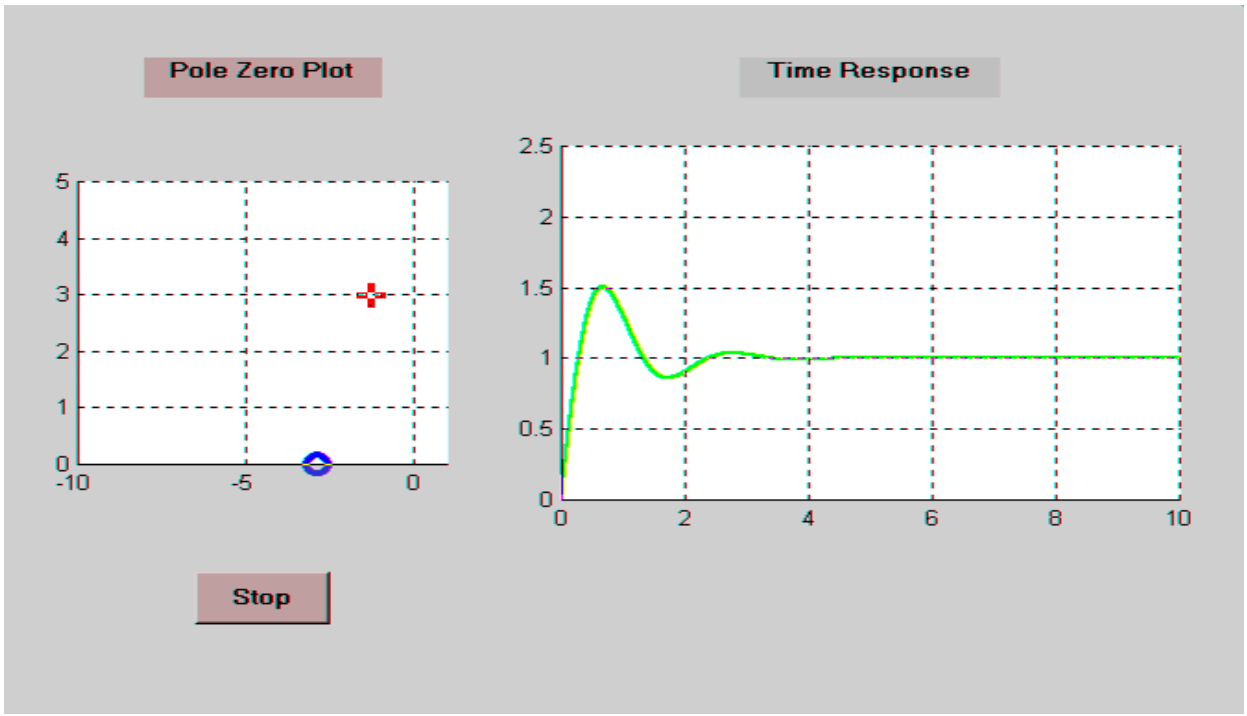
The downloadable files for these examples are available on
http://wseweb.ew.usna.edu/wse/download/

Bibliography
1. *Building GUIs with MATLAB Version 5*, The Mathworks, 1998.
2. Dean, Loren, *The Design of Callbacks*, The Mathworks, 1997.
3. Hanselman, D., Littlefield, B., *Mastering MATLAB 5*, Prentice Hall, 1998, Chapters 31 & 32.
4. *Using MATLAB Graphics Version 5.2*, The Mathworks, 1998, Chapter 8.

ROBERT DeMOYER
Bob DeMoyer is a faculty member in the Weapons and Systems Engineering Department of the United States Naval Academy. He has been active for years in the CoED Division of ASEE, and currently serves as Secretary/Treasurer. Dr DeMoyer received a BS degree in Electrical Engineering from Lehigh University and a MS and PhD in System Engineering from the Polytechnic Institute of Brooklyn.



Figure(1) Unit Step Response of Quadratic Transfer Function



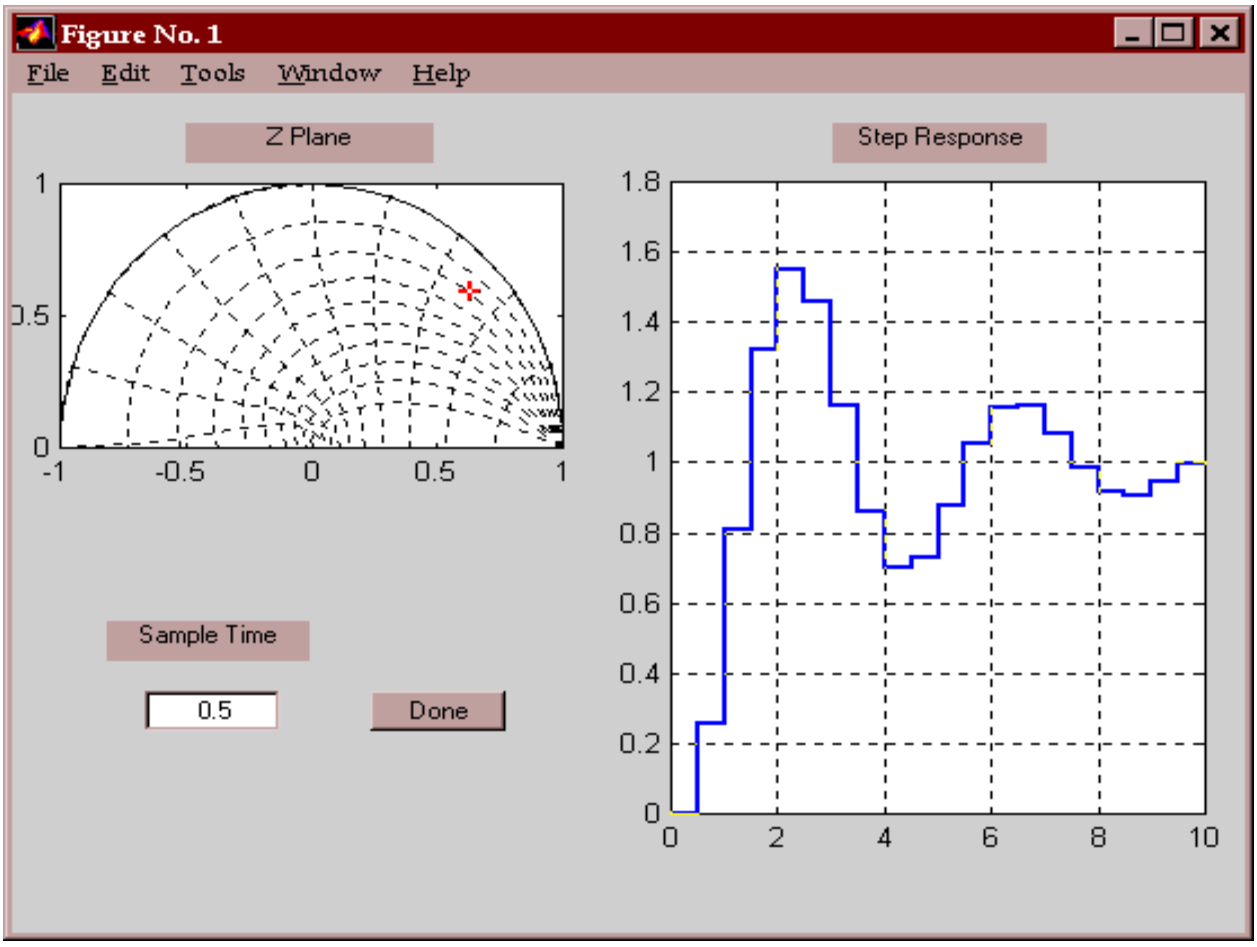
Figure(2) Continuous Step Response

```
% anim.m         Animation callback function for Pole Zero Program
%                R. DeMoyer & E. E. Mitchell, 1999
function anim(action)
global gain num dem pole zero Tim;                    % global variables
switch(action)                                        % switchyard callbacks
case 'start'
   set(gcbf,'windowbuttonmotionfcn','anim move')      % 'move' when in motion
   set(gcbf,'windowbuttonupfcn','anim stop')          % 'stop' when button up
case 'move'
   point = get(gca,'CurrentPoint');                   % get mouse coordinates
   tag = get(gco,'tag');                              % get the tag of the object
   if tag == 'Pole'                                   % execute this if pole current
      pole = point(1,1) + j*point(1,2);               % new value for pole
      dem = [1 -(pole+conj(pole)) pole*conj(pole)];   % re-compute denominator
      gain = abs(pole*conj(pole)/zero);               % re-compute gain
      set(gco,'xdata',real(pole))                     % re-plot the pole
      set(gco,'ydata',imag(pole))
   end
   if tag == 'Zero'                                   % execute if zero is current
      zero = point(1,1);                              % new value for zero
      num = [1 -zero];                                % re-compute num
      gain = abs(pole*conj(pole)/zero);               % re-compute gain
      set(gco,'xdata',zero)                           % re-plot the zero
      set(gco,'ydata',0)                              % on the horizontal axis
   end
   h_plot = findobj('tag','Time');                    % handle of object  tag 'Time'
   y=step(gain*num,dem,Tim);                          % re-compute the step response
   set(h_plot,'xdata',Tim)                            % horizontal component of plot
   set(h_plot,'ydata',y)                              % vertical component of plot
case 'stop'
   set(gcbf,'windowbuttonmotionfcn','')               % button up: nothing to happen
   set(gcbf,'windowbuttonupfcn','')                   % so no callbacks.
end
```

Figure(3) Animation Callback Function

```
% polezero.m     Main program
%                R. DeMoyer & E. E. Mitchell, 1999
global gain num dem pole zero Tim;                    % global variables
zero = -3; pole = -2+j*2;                             % initial zero and pole
gain = abs(pole*conj(pole)/zero);                     % gain for unity final
num=[1 -zero];                                        % transfer function numerator
dem=[1 -(pole+conj(pole)) pole*conj(pole)];           % denominator
Tim=0:.01:10;                                         % definition of time vector
h_splane_axis = axes('position',[.06 .35 .3 .4]);
axis([-10 1,0 5]); hold;                              % s-plane axis
h_pole = plot(real(pole),imag(pole),'r+');grid        % plot the pole
set(h_pole,'ButtonDownFcn','anim start',...           % set some properties
   'EraseMode','xor','Tag','Pole','markersize',10,...
   'linewidth',3);
h_zero=plot(zero,0,'go');                             % plot the zero
set(h_zero,'ButtonDownFcn','anim start',...           % set some properties
   'EraseMode','xor','Tag','Zero','markersize',10,...
   'linewidth',3)
h_tim_axis = axes('position',[.45 .3 .5 .5]);         % time response axis
axis([0 10 0 2.5]);hold;
y=step(gain*num,dem,Tim);                             % unit step response
   h_plot=plot(Tim,y);grid;                           % plot step response
set(h_plot,'ButtonDownFcn','anim start',...           % set some properties
   'EraseMode','xor','Tag','Time','markersize',10,...
   'linewidth',1.7);
close all                                             % will be re-opened
                                                      % by Guide generated .m file
pz                                                    % execute Guide generated file
```

Figure(4) Main Program



Figure(5) Discrete Step Response