Creating Graphical User Interfaces with MATLAB

Dr. Howard Silver
silver@fdu.edu
Fairleigh Dickinson University
1000 River Road Teaneck, NJ 07666

Abstract: MATLAB is a widely used matrix based equation solving program, with the features of a general purpose programming language along with a vast collection of built-in functions, which include extensive graphical capability. More recent versions of MATLAB have allowed users to Create Graphical User Interfaces (GUIs), enabling interaction with graphical objects, such as text boxes, push-buttons, pop-up menus and sliders.
MATLAB has a built-in Graphical User Interface Development Environment (GUIDE), with which we can lay out the GUI graphically and have MATLAB automatically generate the code. However, writing our own programs gives us more understanding and flexibility in being able to modify the code to suit our application. Therefore, we will emphasize the programming approach in this presentation.

The GUI examples include the following:

- Text boxes displaying
  - Static text
  - User entry text
- Push-buttons activating
  - Up-counter
  - Up/Down counter
  - Four function calculator
  - Tic-tac-toe game
  - Alphabetic character pattern generation for neural network testing
- Pop-up menu activating
  - Four function calculator
- Sliders controlling
  - Slope of a plotted line
  - Sample interval for numerical solution of a differential equation
  - Frequency of a sinusoidal input to a series RLC circuit
  - Noise level and duration for signal detection using averaging

Key words:
MATLAB
GUI
Text Box
Push Button
Pop up Menu
Slider

Introduction:

MATLAB is a widely used matrix based equation solving program, which has a Command Window for interactive use and a program editor. It has the features of a general purpose programming language along with a vast collection of built-in functions which include extensive graphical capability. MATLAB‟s basic **plot**or **plot3**functionsgenerate two or three dimensional graphs of data vectors. Creating Graphical User Interfaces (GUIs) enable interaction with graphical objects such as text boxes and push-buttons.

GUIs are examples of hierarchal object oriented programming, where the graphical objects are "children" of a "parent", which can be a figure or a panel of objects or group of buttons. For the examples to be presented, the "parent" will always be a figure. There are more than ten graphical objects available ("children"), but the examples will use only four types - text boxes, push-buttons, pop-up menus and sliders.

MATLAB has a built-in Graphical User Interface Development Environment (GUIDE), with which we can lay out the GUI graphically and have MATLAB automatically generate the code. However, writing our own programs gives us more understanding and flexibility in being able to modify the code to suit our application. Therefore, we will emphasize the programming approach in this presentation.

The GUI examples include the following:

Text boxes displaying static text and user entry text

Push-buttons activating up-counter, up/down counter, four function calculator, tic-tac-toe game, and alphabetic character pattern generation for neural network testing

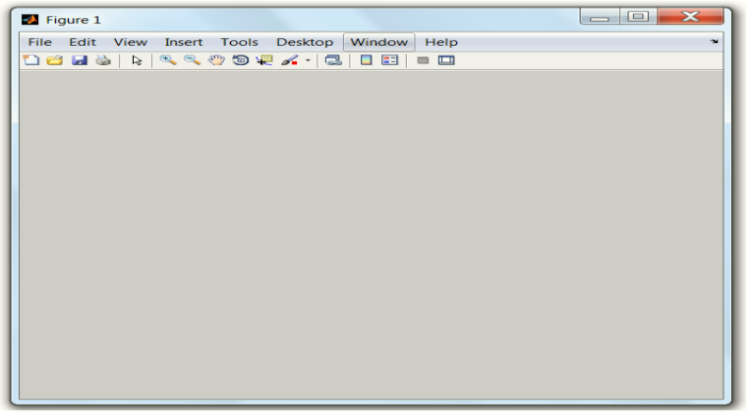
Pop-up menu activating the four function calculator

Engineering applications with sliders controlling the slope of a plotted line, sample interval time for numerical solution of a differential equation, frequency of a sinusoidal input to a series RLC circuit and noise level and duration for signal detection using averaging

How to Start to Create a GUI

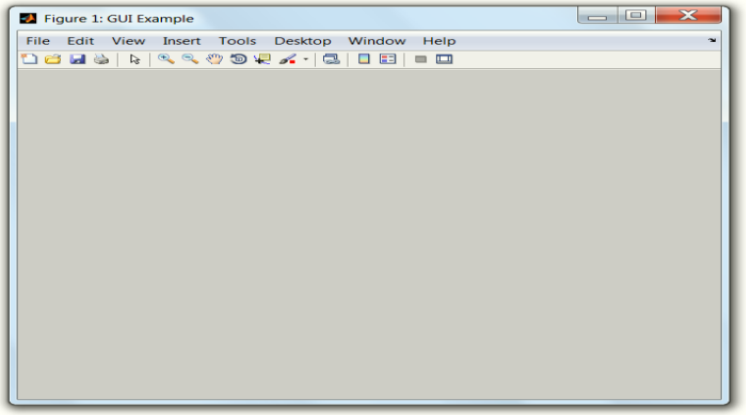
To start, entering **figure**in the command mode opens a figure window as shown. The default title is Figure 1 (or Figure 2, Figure 3, etc. if other figure windows are still open).

```
>>figure
>>
```



The command below also opens a figure window. Since the figure is a graphical object, its properties can be specified given using MATLAB‟s **set** function. In this case, the property **Name** assigns the string that follows as the figure‟s title.

```
>> set (figure, 'Name', 'GUI Example')
>>
```



The function **uicontrol**, to be used in all the examples, enables the creation of user interface control objects. Its general form is

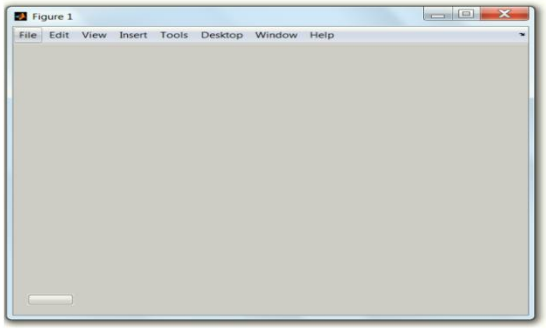
uicontrol(parent, „PropertyName‟,‟PropertyValue‟,…..)

or

uicontrol(„PropertyName1‟, PropertyValue1, „PropertyName2‟, PropertyValue2, …..)

As seen below, a default figure (“parent”) object is created along with a push-button (“child”) object in the lower left corner of the screen.

```
>>uicontrol
>>
```



The same figure would result from entering

uicontrol(„Style‟, ‟pushbutton‟)

The following are examples of property names for the GUI objects:

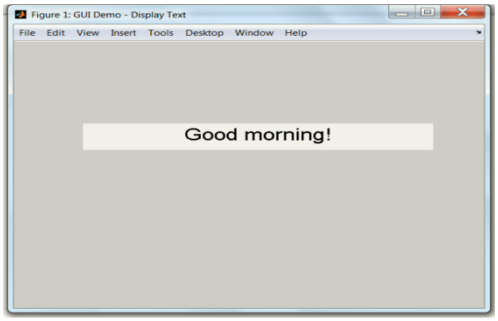
Style	Specifies type of GUI object (e.g. text box, push-button, slider)
Position	A vector with format [left, bottom, width, height] specifying position, width and height in pixels of the GUI object relative to the figure window
String	Text to be displayed
Font Size	Text character size
Callback	Invokes a nested call back function in response to a user action (e.g. clicking on a push button)
Min, Max, Value	Specifies smallest, largest and current values for a slider

The types of GUI objects (i.e. Style property) to be used in our examples are shown below:

Style property	**Object**
Text	Static text box – displays text
Edit	Edit text box – allows user to input text
Pushbutton	Push button - generates action when user presses (i.e. clicks mouse) on it
Popup	Pop-up menu – displays options and generates actions when clicked
Slider	Slide –enables user to adjust position to provide numeric input over a specified range

To generate the GUI below, the code can be entered in the command mode or run as a program (i.e. script M file). A figure box with a title is created and a text box is placed in it. The **Style** property creates a static text box, in which a fixed message is placed. The **String**, **Position** and **FontSize** properties specify the message, the location of the text box and the font size of the text respectively.The resulting display is shown.

```
% GUI - Text Box
set(figure,'Name','GUI Demo - Display Text')
htext=uicontrol('Style', 'text',...
 'String', 'Good morning!', ...
'Position',[100,300,500,50], ...
'FontSize',20);
```



For the remaining examples, function files have been created without any input parameters, allowing them to be run directly and to access a callback function in the same file. The general format of these programs is:

```
functiongui_demo…..
% Set up GUI
        :
        var = uicontrol(…..  „Callback‟, @callbackfn…..);
        :
functioncallbackfn(source, eventdata)
  % User created - tailored to application
        :
end
end
```

Generally the outer function, **gui_demo…..**, sets up the GUI and accesses **callbackfn** in response to a user action, such as clicking the mouse on a push-button. Values (e.g. **var**) assigned by**uicontrol** functions in **gui_demo**are passed to **callbackfn**, where they can be used by the application. The parameter **source** refers to the **uicontrol** object that invoked the function. The variable **eventdata**is reserved for future MATLAB use. The names **callbackfn**, **source** and **eventdata**are arbitrary.For each example, the code is listed and followed by sample displays resulting from executing the program.

Examples with Text Boxes

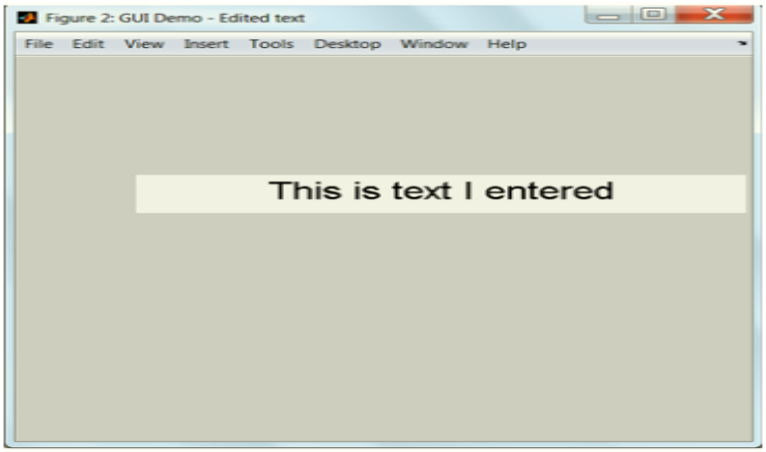
Program 1 illustrates edited rather than static text, a concept similar to MATLAB‟s **input**function, which allowsthe user to enter data rather than assigning a value in the program. The **uicontrol**function in the previous example replaces the **text** style by **edit** and does not specify the string itself. When the program is run, the user can click in the edited text box created, enter a string and press the **<Enter>** key. The function **callbackfn** is then invoked, the string is passed to it via the variable **usertext**, and saved as **printstr**. Once **<Enter>** is pressed, the string cannot be further edited.

Program 1

```
% GUI - Edited Text
functiongui_demo_edited_text
set(figure, 'Name', 'GUI Demo - Edited text')
usertext=uicontrol('Style', 'edit', ...
'Position', [100,300,500,50], ...
```

```
'FontSize', 20,...
'Callback', @callbackfn);

functioncallbackfn(usertext, eventdata)
printstr = get(usertext,'String');
hstr=uicontrol('Style', 'text', ...                % Saves string when <Enter> is pressed
 'String', printstr, ...
'Position', [100,300,500,50], ...
'FontSize', 20);
end
end
```

Sample Display



Note that an edit box can be created without the need for a callback function. The code shown below, when run, shows a blank text box and the user can edit and modify text. The **<Enter>** key has no effect, however, and the text is not saved.

```
set(figure,'Name','GUI Demo - Edited Text')
htext=uicontrol('Style', 'edit',...
'Position',[100,300,500,50], ...
'FontSize',20);
```

In Program 2, the edited text box is supplemented by a push-button placed near the lower left edge of the figure window. **Enter** is the arbitrary name displayed on the push-button. The callback function, which is the same one used in Program 1, is now a property of the push-button rather than the edit text box. Thus, the clicking of the push button rather than the pressing of the **<Enter>** key saves the string and prevents further editing.

Program2

```
% GUI - Edited Text
functiongui_demo_pushbutton
set(figure, 'Name', 'GUI Demo - Pushbutton')
usertext=uicontrol('Style', 'edit', ...
```

```
'Position', [100,300,500,50], ...
'FontSize', 20);
userbutton=uicontrol('Style', 'pushbutton', ...
'String', 'Enter', ...
'Position', [20, 20, 200,50], ...
'FontSize', 20, ...
'Callback', @callbackfn);

functioncallbackfn(userbutton, eventdata)
printstr = get(usertext, 'String');
userstring=uicontrol('Style', 'text', ...              % Saves string when pushbutton is clicked
'String', printstr, ...
'Position', [100,300,500,50], ...
'FontSize', 20);
end
end
```

Sample Display

Examples with Push-Buttons and Pop-Up Menu

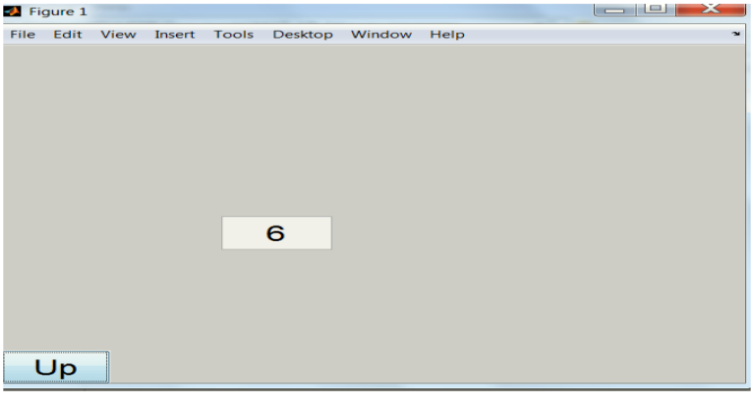
In Program 3 the static text box displays a numerical value (initially zero) rather than user entered text. The push-button, labeled **Up**, increments the numerical value on every click of the button. The MATLAB function **num2str** converts a numerical value to a string and **str2num** does the reverse. Each time the callback function is invoked, it gets the number (**count**) and adds 1 to it and displays it as text in the edit window.

Program3

```
functiongui_demo_up_counter
  n=0;
count=uicontrol('Style','text',...                % Creates text box displaying count
'Position',[200,200,100,50],...
```

```
'FontSize', 20,...
                    'String',num2str(n));
push_button=uicontrol('Style', 'pushbutton',...          % Creates pushbutton in lower left corner
'String','Up',...
                    'Position',[0,0,100,50],...
                    'FontSize', 20,...
'Callback',@callbackfn);

functioncallbackfn(pushbutton,eventdata)
    n=str2num(get(count,'String'));
    n=n+1;
set(count,'String',num2str(n))
end
end
```

Sample Display



Program 4 modifies Program 3 by adding a second push-button (**Down**) which decrements the count. Both buttons need to access the callback function when clicked, with the parameter **source**identifying the appropriate button. The if-else structure either increments or decrements the count by 1. Thus, negative numbers can appear as illustrated by the sample output.

Program 4

```
functiongui_demo_up_down_counter
  n=0;
count=uicontrol('Style','text',...                       % Creates text box displaying count
 'Position',[200,200,100,50],...
           'FontSize', 20,...
'String',num2str(n));
up_button=uicontrol('Style', 'pushbutton',...            % Creates two pushbuttons
'String','Up',...
                    'Position',[0,0,100,50],...
                    'FontSize', 20,...
'Callback',@callbackfn);

down_button=uicontrol('Style', 'pushbutton',...
 'String','Down',...
                     'Position',[150,0,100,50],...
                    'FontSize', 20,...
```

```
 'Callback',@callbackfn);

functioncallbackfn(source,eventdata)
     n=str2num(get(count,'String'));
if source==up_button
         n=n+1;
else
         n=n-1;
end
set(count,'String',num2str(n))
end
end
```

Sample Display

Program 5 creates a display for a four-function calculator (add, subtract, multiply, and divide operations). Static text boxes are provided for a title in the figure window (not to be confused with a figure title), the character showing the operation performed (+, -, * or /), and the equal to symbol (=). Edit text boxes are required for user entry of the two numbers to be operated upon (**n1** and **n2**);the calculated answer (**result**) is placed in a static text box. A separate push-button is provided for each of the four operations.

Similar to the previous program, the callback function parameter **source** identifies which of the four operations is chosen and the corresponding operator (**op**) and the calculated result (**answer**). As shown below, zeros are displayed for **n1** and **n2** initially. The user can change the numbers and display the result by clicking one of the buttons. The resulting displays are shown for each operation performed on seven digit numbers. The text boxes widths should be sufficient to display long digit strings.

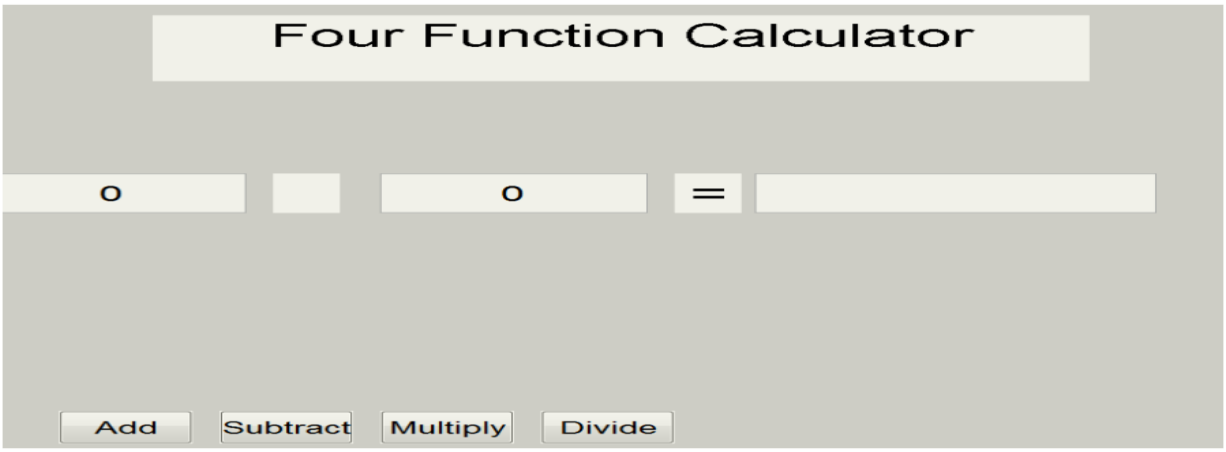
The structure of this program makes it straightforward to add more functions to the calculator.

Program 5

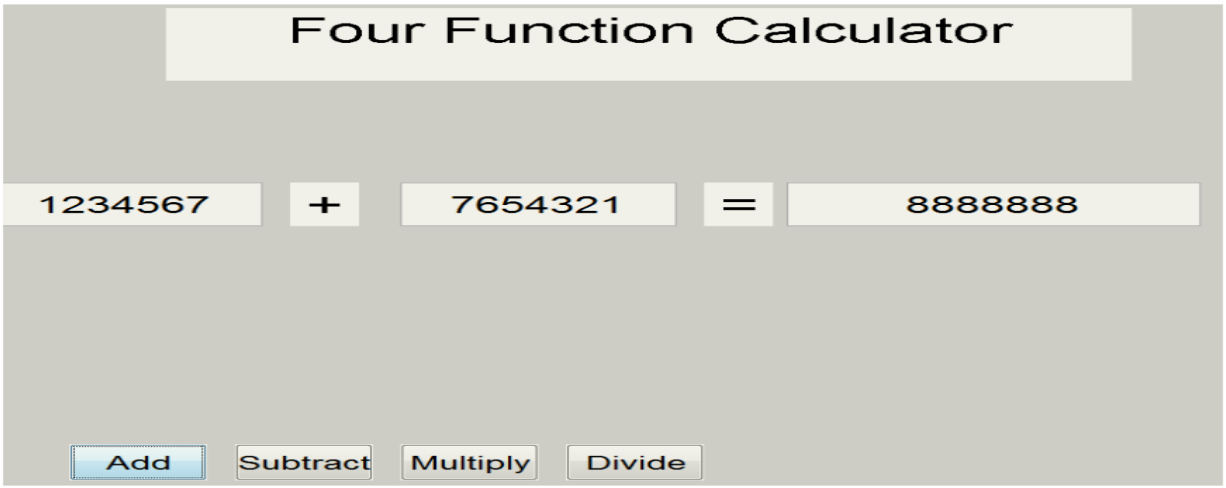
```
function gui_calculator_4function
% gui_calculator has 2 edit boxes for numbers and
%   adds, subtracts, multiplies or divides them
```

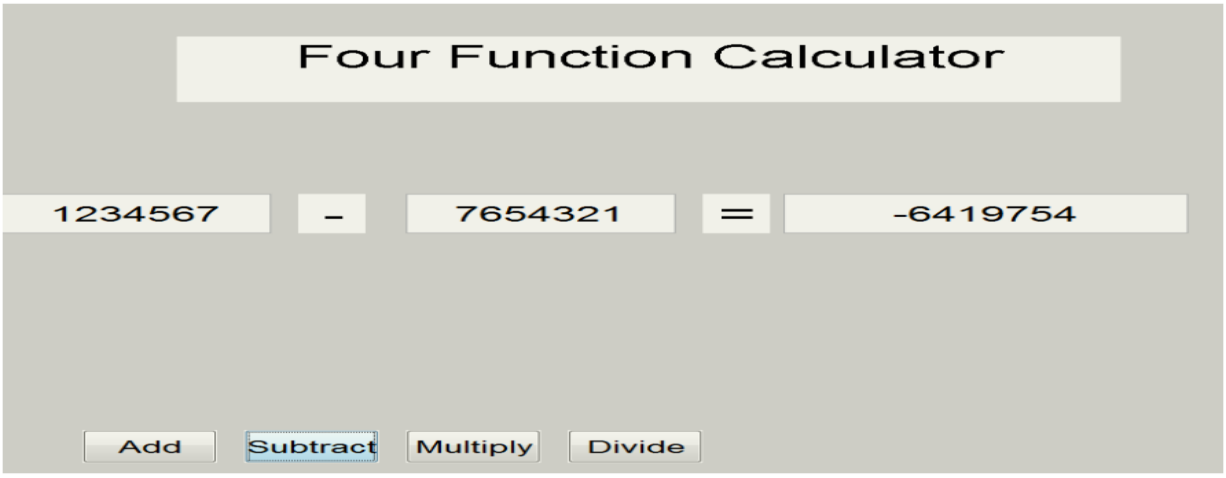
```
set(figure,'Name','GUI Calculator')
n1=0; n2=0;

title = uicontrol('Style','text','Position',[150,600,700,100],...
            'FontSize',30,'String','Four Function Calculator');
operator = uicontrol('Style','text',...
                        'Position',[240,400,50,60],'FontSize', 20);
equal_sign = uicontrol('Style','text',...
                        'Position',[540,400,50,60],'FontSize', 30,'String','=');
result = uicontrol('Style','text',...
                        'Position',[600,400,300,60]);
firstnum = uicontrol('Style','edit','Position',[20,400,200,60],...
                        'FontSize', 20, 'String',num2str(n1));
secondnum = uicontrol('Style','edit','Position',[320,400,200,60],...
                        'FontSize', 20,'String',num2str(n2));
add_button = uicontrol('Style','pushbutton', 'String','Add',...
                        'Position',[80,50,100,50], 'Callback',@callbackfn);
subtract_button= uicontrol('Style','pushbutton', 'String','Subtract',...
                                    'Position',[200,50,100,50], 'Callback',@callbackfn);
multiply_button = uicontrol('Style','pushbutton', 'String','Multiply',...
                                    'Position',[320,50,100,50], 'Callback',@callbackfn);
divide_button = uicontrol('Style','pushbutton', 'String','Divide',...
                                    'Position',[440,50,100,50], 'Callback',@callbackfn);

functioncallbackfn(source,eventdata)
      n1=str2num(get(firstnum,'String'));
      n2=str2num(get(secondnum,'String'));
if source == add_button
op='+';
answer=n1+n2;
elseif source == subtract_button
op='-';
answer=n1-n2;
elseif source == multiply_button
op='*';
answer=n1*n2;
elseif source == divide_button
op='/';
answer=n1/n2;
end
set(operator,'String',op,'FontSize',30)
set(result,'String',num2str(answer),'FontSize',20)

end
end
```

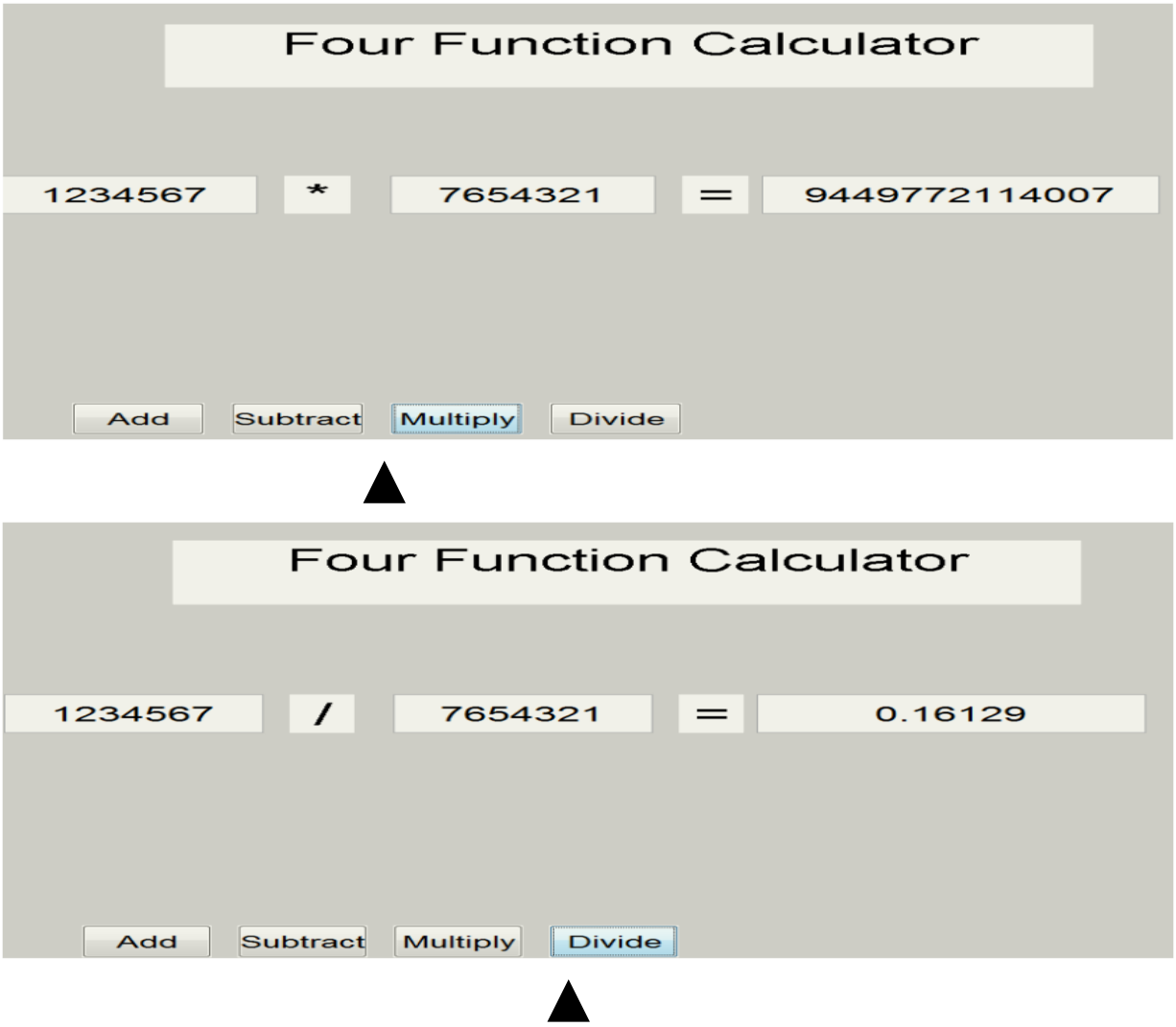
Initial Display



Other Sample Displays







In Program 6, the push-buttons used in Program 5 for the four function calculator are replaced by a pop-up menu. The statement

op_menu = uicontrol('Style', 'popup', 'String', 'Add|Subtract|Multiply|Divide',…..

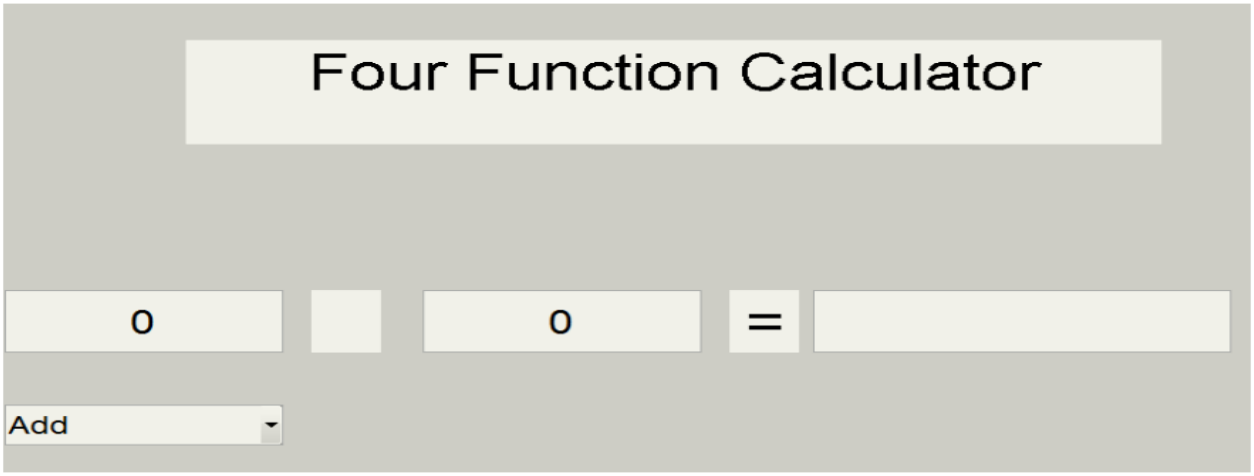
creates the menu and assigns the text shown to each menu item, with |as the separator. When the menu is first displayed, only the first operation (**Add**) is visible, but clicking on the menu's down arrow enables selection of the desired operation. The calculation is then performed on the two numbers entered and the result is displayed.

Since the **'Value'** property of the pop-up menu assigns integers 1, 2, 3 and 4 to the operations listed in the order shown, it is convenient to use a **switch** structure in the callback function, instead of the more awkward **if/elseif**code.

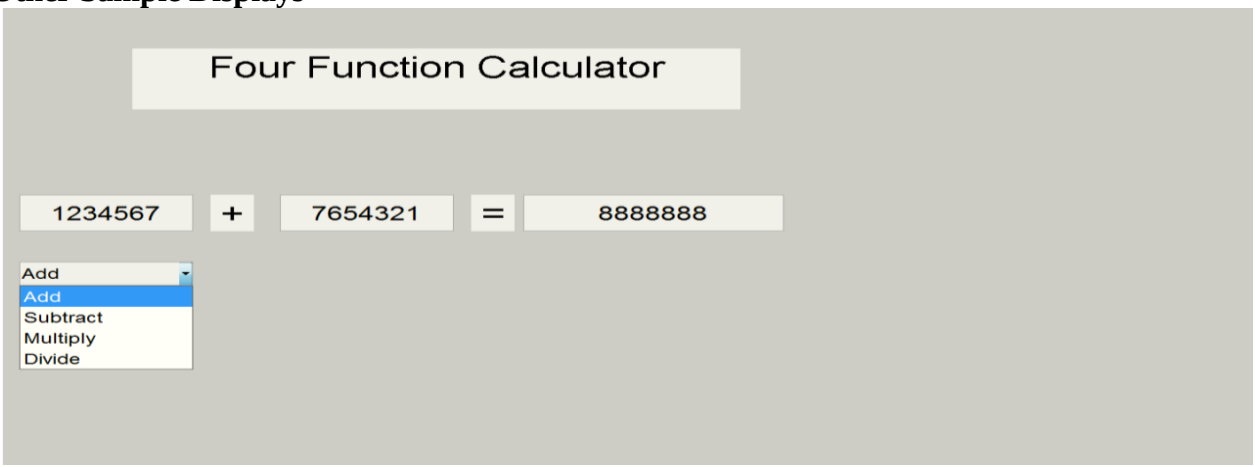
Program 6

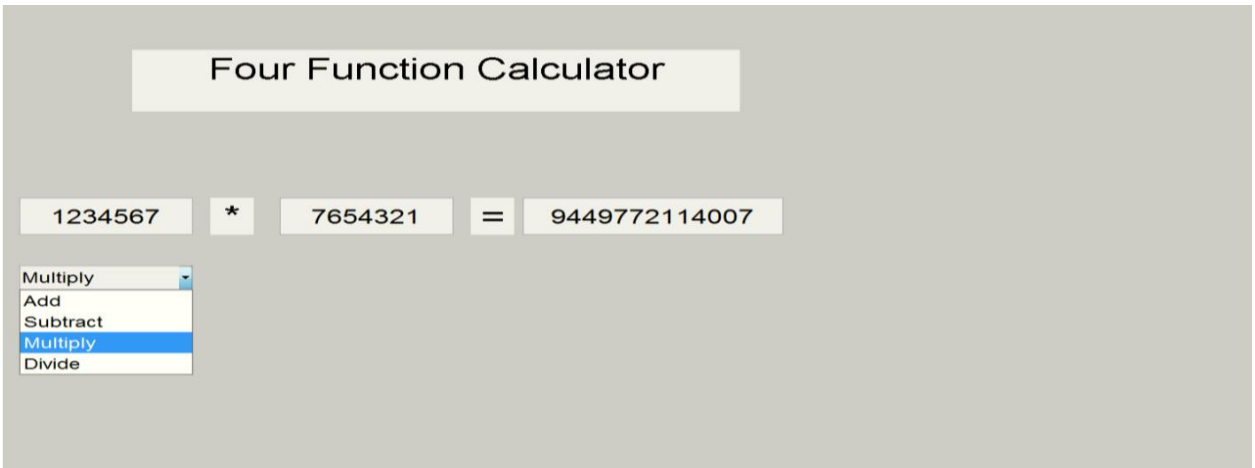
```
function gui_calculator_4function_popup_menu
% gui_calculator has 2 edit boxes for numbers and
%   adds, subtracts, multiplies or divides them

set(figure,'Name','GUI Calculator');

n1=0;n2=0;

title = uicontrol('Style','text','Position',[150,600,700,100],'FontSize',30, ...
          'String','Four Function Calculator');
operator = uicontrol('Style','text','Position',[240,400,50,60], ...
                    'FontSize', 20);
equal_sign=uicontrol('Style','text','Position',[540,400,50,60], ...
                    'FontSize', 30,'String','=');
result=uicontrol('Style','text', ...
          'Position',[600,400,300,60]);
firstnum = uicontrol('Style','edit','Position',[20,400,200,60],'FontSize', 20, ...
                    'String',num2str(n1));
secondnum=uicontrol('Style','edit','Position',[320,400,200,60],'FontSize',20, ...
                    'String',num2str(n2));
op_menu = uicontrol('Style', 'popup', 'String', 'Add|Subtract|Multiply|Divide',...
                    'Position', [20, 100, 100, 50], 'Callback',@callbackfn);

functioncallbackfn(source,eventdata)
     n1=str2num(get(firstnum,'String'));
     n2=str2num(get(secondnum,'String'));
val=get(op_menu,'Value');
switchval
case 1
          op='+';
          answer=n1+n2;
case 2
          op='-';
          answer=n1-n2;
case 3
          op='*';
          answer=n1*n2;
case 4
          op='/';
          answer=n1/n2;
end
set(operator,'String',op,'FontSize',30)
set(result,'String',num2str(answer),'FontSize',20)

end
end
```

Initial Display



Other Sample Displays





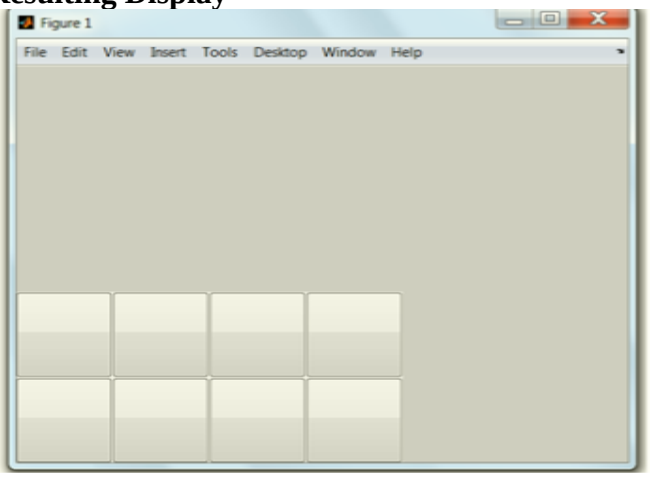
Program 7 uses nested **for** loops to simply create an array of square sized pushbuttons,bordering the lower left edge of the figure window. The distances from the left edge and bottom are set so that the button edges are touching one another. In this example, a 2 row by 4 column array with square size of 100 pixels is displayed.

As will be shown in the next example, an action associated with a particular pushbutton can be carried out by including the callback function and referencing **button(i,j)** in that function.

Program 7

```
functiongui_button_array
rows=2; cols=4; size=100;
fori=1:rows
for j=1:cols
button(i,j)=uicontrol('Style', 'pushbutton',...
        'Position',[(j-1)*size,(rows-i)*size,size,size]);
end
end
end
```

Resulting Display



In Program 8, a pushbutton array is created for a game of tic-tac-toe, with an equal number of rows and columns (**N**). The traditional game uses a 3x3 array, but the user can specify a 4x4 or 5x5 array for a greater challenge. A larger array size can be specified, but may require some re-sizing of the squares. When a player clicks on any button, the callback function is accessed. If the square has not been selected previously, depending on whose turn it is, an**X**or **O** is entered into the square. A static text box (**message**) is also created to display the status of the game.

The variable **first_player**, initially set to 1, indicates who has the next move; the square matrices **Xseq** and **Oseq** (initially all zeros) keep track of the player selections by setting the appropriate entry to 1. After each entry, all rows and columnsalong with the two diagonals are checked to determine if any one contains all **X**s or all **O**s. If so, a message identifying the winner is displayed in the text box. A record is kept of the number of entries (**moves**), and when all squares have entries and there is no winner, **Game Ends in a Tie** is displayed.

Program 8

```
functiongui_tictactoe
   N=input('Enter game size (3, 4 or 5)    ');
Xseq=zeros(N);
Oseq=zeros(N);
first_player=1;
   X_WIN=0;
   O_WIN=0;
moves=0;

message=uicontrol('Style','text','Position',[500,0,400,100],...
           'FontSize',20,...
         'String', 'First Player Enter "X" ');

    % Create pushbutton array
fori=1:N
for j=1:N
button(i,j)=uicontrol('Style', 'pushbutton',...
           'Position',[(j-1)*100,(N-i)*100,100,100],...
          'FontSize', 40,...
        'Callback',@callbackfn);
end
end

functioncallbackfn(source,eventdata)
fori=1:N
for j=1:N
if source==button(i,j)&Xseq(i,j)==0 &Oseq(i,j)==0       % Select empty box only
moves=moves+1;
iffirst_player
set(button(i,j),'string','X');
Xseq(i,j)=1;
first_player=0;
else
set(button(i,j),'string','O');
Oseq(i,j)=1;
first_player=1;
end
end

             % Check all rows, columns and diagonals for all X's or all O's
            Xrow=sum(Xseq(i,:));
Xcol=sum(Xseq(:,j));
Orow=sum(Oseq(i,:));
Ocol=sum(Oseq(:,j));

              Xdiag1=0; Xdiag2=0; Odiag1=0; Odiag2=0;
for k=1:N
                 Xdiag1=Xdiag1+Xseq(k,k);
                 Xdiag2=Xdiag2+Xseq(k,N-k+1);
                 Odiag1=Odiag1+Oseq(k,k);
                 Odiag2=Odiag2+Oseq(k,N-k+1);
end
```

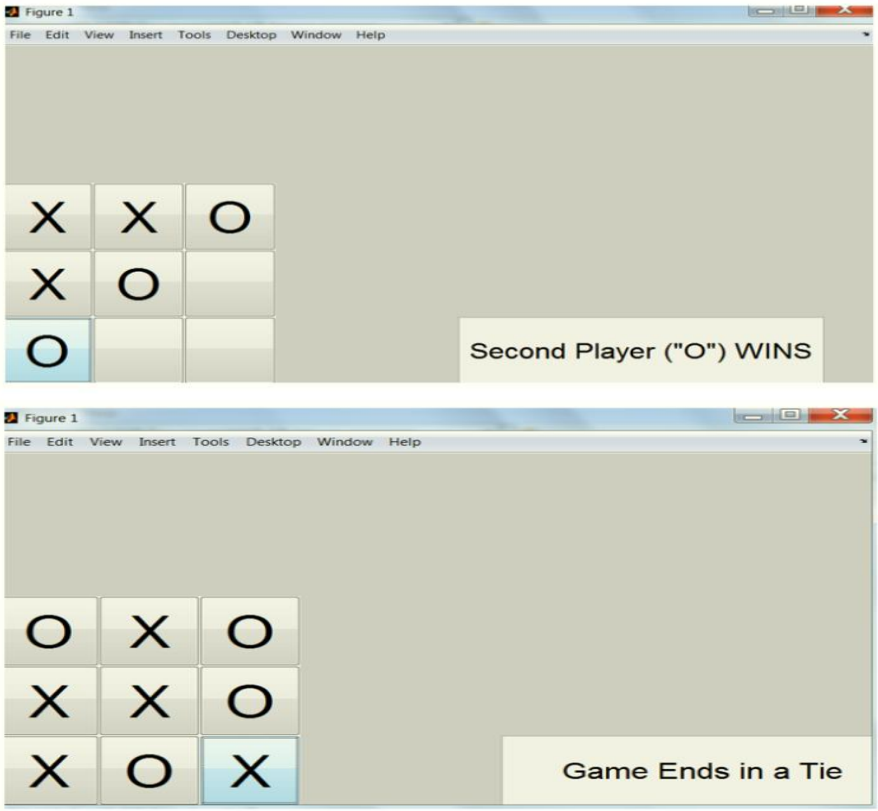
```
            ifXrow== N | Xcol==N | Xdiag1==N | Xdiag2 ==N
                X_WIN=1;
elseifOrow== N | Ocol==N | Odiag1==N | Odiag2 ==N
                O_WIN=1;
end

ifX_WIN
set(message, 'String', 'First Player ("X") WINS!'), return
elseif O_WIN
set(message,'String', 'Second Player ("O") WINS'), return
elseif moves==N*N
set(message,'String', 'Game Ends in a Tie'), return
elseiffirst_player
set(message,'String', 'First Player Enter "X" ')
else
set(message,'String', 'Second Player Enter "O" ')
end
end
end
end
end
```

Sample display for a traditional 3x3 game:

>>Enter game size (3, 4 or 5) 3

Initial Display

Other Sample Displays (for a 3x3 game)



Examples with Sliders – Engineering Applications

Programs 9 to 12 illustrate the use of sliders to control certain variables in a user program. In these examples plotted data is placed into the figure window and moving the slider varies a parameter of the graph. In this manner we can see the effect of a change on the plot without the need to restart the program each time.

In these programs we initially specify the slider range from the smallest (**minval**) to largest (**maxval**) value. The slider is created with the statements

slider_handle = uicontrol(„Style“, „slider“, „Position“, ….., „Callback“, …..);
min_slide = uicontrol(„Style“,“text“, „Position“, ….., „String“, num2str(minval));
max_slide= uicontrol(„Style“,“text“, „Position“, ….., „String“, num2str(maxval));
value_slide= uicontrol(„Style“, „text“, „Position“, …..);

The first line displays the slider and invokes the callback function when the slider position is moved (i.e. by clicking or dragging with the mouse). The remaining code creates text boxes to display the slide range and current value proportional to the slide“s relative position. In the examples, the range values are positioned below the slide and the current value above it. The slide range and initial position are specified by assigning values to the properties **‘Min’**, **‘Max’**and **‘Value’**in **slider_handle**. If **‘Value’** is not assigned, it defaults to zero.

Since the slider range values are constant, they are displayed immediately as strings. The current value (**'Value'**) is initialized in each example to the minimum value;it is displayed and then updated by the callback function, which is activated by moving the slide. Additional text boxes are placed below each slide in Programs 10 to 12, to label the quantity represented by the slide‟s value.

As another option, MATLAB‟s **axes** function (not to be confused with **axis**, which sets the x-y scales for plotting)creates a graphics object in the figure window that provides a coordinate system for data to be plotted. If **axes**is not used, the subsequent plot will fill the entire window. The use of **axes** is illustrated in Program 9 only. In the other programs, where multiple plots are shown (i.e. subplots), the entire figure window was used for clarity and the slider was kept small in size and squeezed in between the plots.

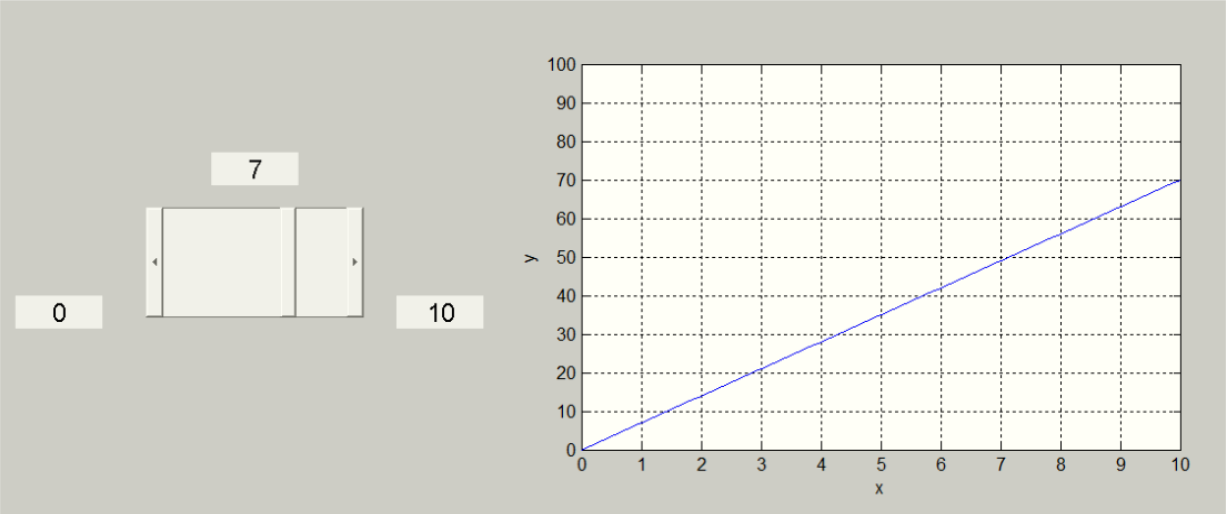
In Program 9, a simple straight line passing through the coordinate origin is plotted, with a slope (**m**) proportional to the slider position. The sample displays shows the plot of

$$y = 6\,x \qquad (0 \le x \le 10)$$

Program 9

```
function gui_demo1_slider
% Slider controls slope of a plotted line

% Minimum and maximum values for slider
minval = 0;
maxval = 10;
 % Create the slider object
slider_handle = uicontrol('Style','slider', ...
                'Position',[200,320,200,100], ...
                'Min', minval, 'Max', maxval,'Value', minval, ...
                'Callback', @callbackfn);
% Text boxes to show the min and max values and slider value
min_slide= uicontrol('Style','text', ...
              'Position', [80, 310, 80,30], ...
             'String', num2str(minval), 'FontSize', 15);
max_slide= uicontrol('Style','text', ...
 'Position', [430, 310, 80,30], ...
'String', num2str(maxval),  'FontSize', 15);
value_slide = uicontrol('Style','text','Position', [260,440,80,30],...
'String', num2str(minval), 'FontSize', 15);
% Create axes handle for plot
axes_handle = axes('Units','Pixels',...
'Position', [600,200,550,350]);

% Call back function displays the current slider value & plots n points
functioncallbackfn(slider_handle,eventdata)
      m=get(slider_handle, 'Value');
set(value_slide,'String',num2str(m))
      x = 0:0.1:10;
      y = m*x;
plot(x,y), grid, xlabel('x'), ylabel('y'), axis([0, 10, 0, 100])
end
end
```

Sample Display



In Program 10, a first order linear differential equation is solved numerically by converting it to a difference equation. The accuracy of the solution depends on the sampling interval used in converting the continuous time function to a discrete set of values. A slider is provided to control the sampling interval.

In this example, the differential equation for y(t) is given by

$$y'' + y = t \qquad \text{for} \qquad t \geq 0 \,,$$

with the initial condition y(0) = 0. The exact solution of this equation is

$$y(t) = t - 1 + e^{-t}$$

The difference equation can be shown to be

$$y(n) = [\, y(n-1) + n\, T_s^{\,2}] \,/\, (1 + T_s) \qquad \text{with } y(1) = 0$$

for the n-th discrete (sampled) value, where T_s is the sampling interval.

The function y(t) is sampled over the interval $0.01 \leq t \leq$ **Tmax**, with **Tmax** set to 5. The values of **y(n)** are calculated and plotted. With the aid of MATLAB's functions **dsolve**(from the Symbolic Math Toolbox) and **ezplot**, the exact solution is shown in the lower subplot.

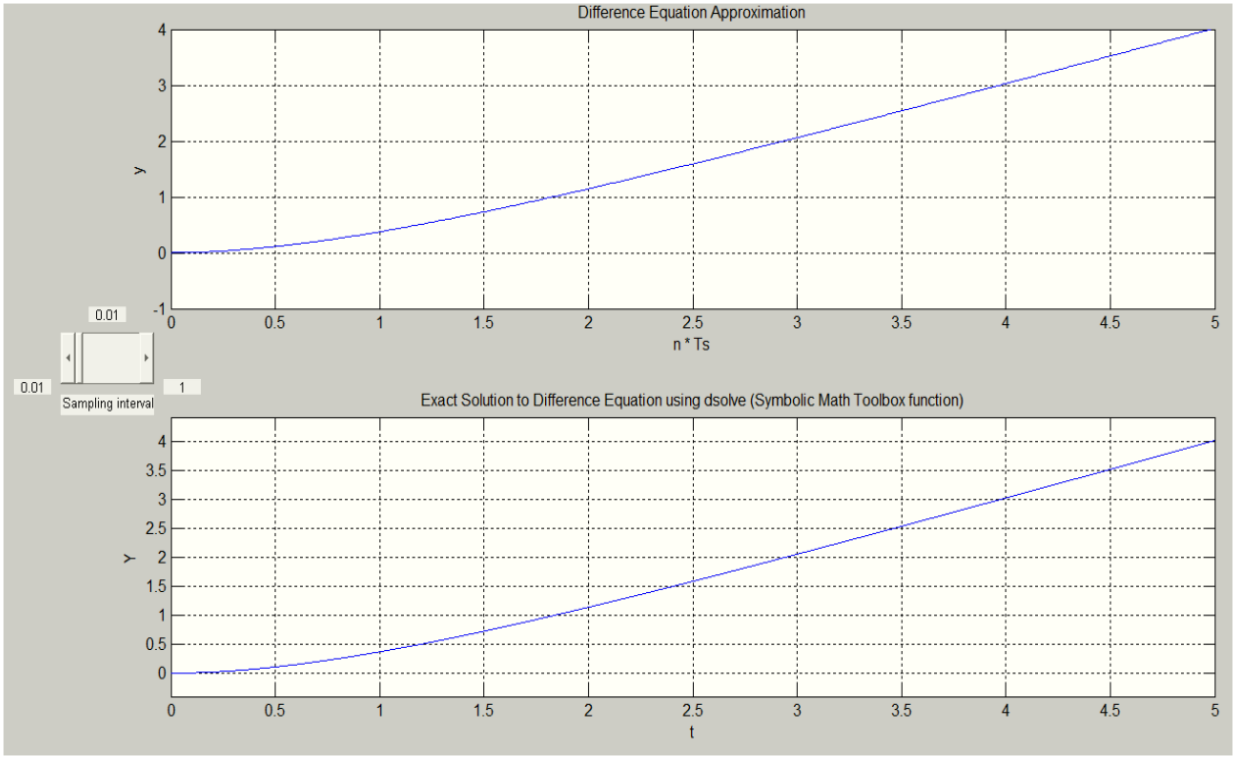
Sets of plots are shown for sampling intervals $T_s = 0.01$ and $T_s = 1$, which show that the approximate solution is more accurate for the smaller interval.

Program 10

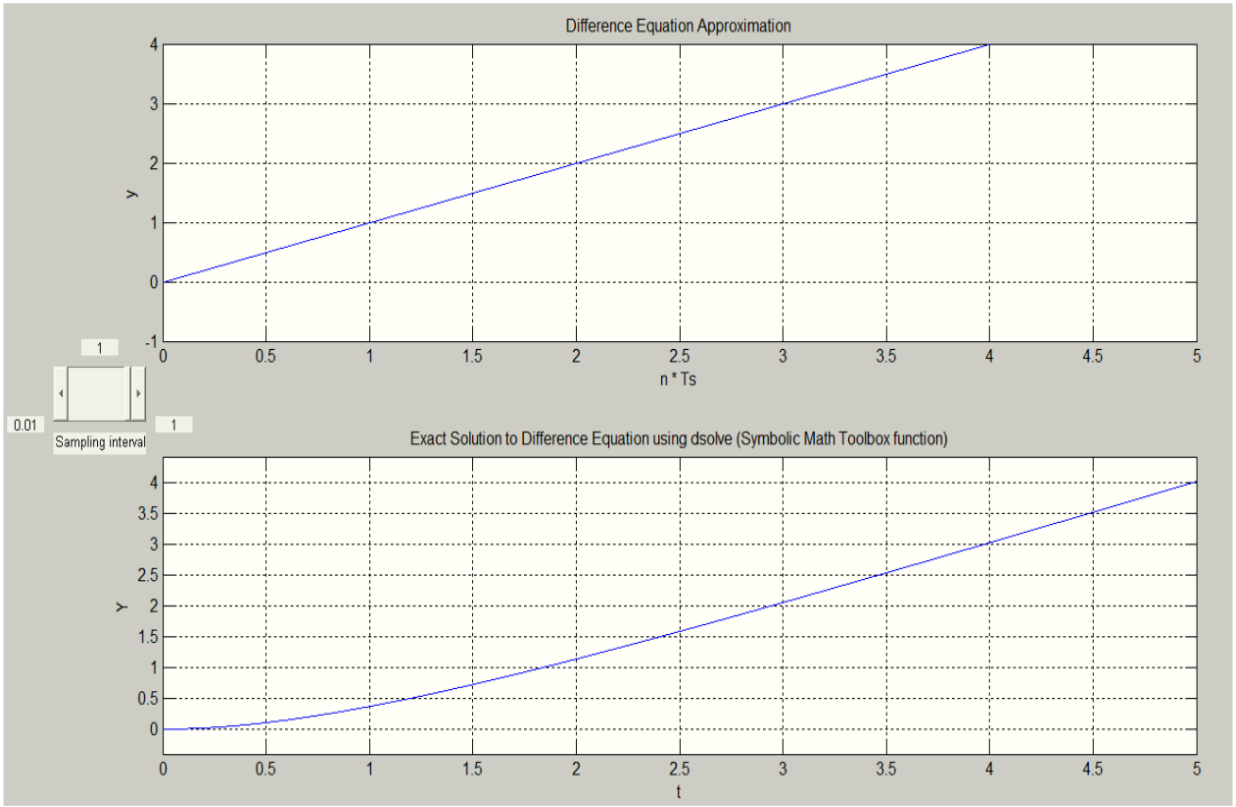
```
function gui_demo2_slider
% Plot the approximate solution to the differential equation      y' + y = t u(t), y(0)=0
% where the sampling interval is the value of the slider

f = figure;
set(f, 'Name','Slider Example with Plot of Diff. Equation Solution')
% Minimum and maximum values for slider
minval = 0.01;
maxval = 1;

% Create the slider object
slider_handle = uicontrol('Style','slider', ...
   'Position',[70,390,100,50], ...
  'Min', minval, 'Max', maxval, 'Value', minval, ...
 'Callback', @callbackfn);
% Text boxes to show the min and max values and slider value
min_slide = uicontrol('Style','text', ...
'Position', [20, 380, 40,15], ...
'String', num2str(minval));
max_slide = uicontrol('Style','text', ...
'Position', [180, 380, 40,15], ...
'String', num2str(maxval));
title_slide = uicontrol('Style','text','Position', [70,360,100,20], ...
                        'String','Sampling interval');
value_slide = uicontrol('Style','text','Position', [100,450,40,15], ...
                        'String', num2str(minval));

 % Call back function displays the current slider value & plots the D.E. solution
functioncallbackfn(slider_handle,eventdata)
Ts = get(slider_handle, 'Value');
set(value_slide,'String',num2str(Ts))
      y = 0;
y(1) = 0;                                          % Initial condition
Tmax = 5;
      N = Tmax / Ts;

for n = 2 : N                                       % Solution to difference equation
y(n) = ( y(n-1) + n * Ts ^ 2) / (1 + Ts);
end
      n = 1:N;
subplot(2,1,1)
plot( (n-1) * Ts, y ), grid
title('Difference Equation Approximation')
xlabel('n * Ts')
ylabel('y')
axis([0 5 -1 4])

% Repeat using dsolve and ezplot functions (Symbolic Math toolbox)
      Y = dsolve('DY + Y = t,  Y(0) = 0');
subplot(2,1,2)
ezplot(Y, [0 5]), grid
ylabel('Y')
title('Exact Solution to Difference Equation using dsolve (Symbolic Math Toolbox function)')
end
end
```

Sample Displays

Sampling time = 0.01



Sampling time = 1



In Program 11 a slider is used to control the frequency (**f**) of a sinusoidal voltage source applied to a series RLC circuit. The peak value of the source ,$\mathbf{V_{in\_peak}}$, is set to 10 (assume in volts – v.). The component values are chosen so that the current (or proportional resistor voltage) will have a peak amplitude at a center or resonant frequency f_0 = 800kHz., with a bandwidth (i.e. difference between the upper and lower 3dB frequencies) of 10kHz. The circuit could be used to "tune" to an AM radio station with an 800 kHz. carrier frequency and provide the 10kHz. band typically used for audio reception. The quality factor for the circuit (**Q**) is the ratio of f_0 to the bandwidth, giving us **Q** = 80 for this circuit.A slide range of 750,000 to 850,000 is selected.

The callback function applies standard AC circuit analysis, using phasors and complex impedances, to find the steady state sinusoidal voltages across **R**, **L** and **C**. These three voltages, $\mathbf{V_R}$, $\mathbf{V_L}$ and $\mathbf{V_C}$respectively, along with the source voltage $\mathbf{V_{in}}$ are plotted over one period $\mathbf{T} = 1/\mathbf{f}$.Moving the slider position clearly shows how amplitude and phase vary with frequency.

Sample plots are shown for the lowest frequency (750 kHz.) and the center frequency (800 kHz.). The peak values of the component voltages increase and the phase changes as the center frequency is approached. At resonance the peak resistor voltage is $\mathbf{V_{in\_peak}}$=10 v., while the peak inductor and capacitor voltages are approximately equal to $\mathbf{Q\ V_{in\_peak}}$= 800v. Although phase is changing with frequency, $\mathbf{V_C}$ and $\mathbf{V_L}$ are always 180 degrees out of phase with one another and 90 degrees out of phase with $\mathbf{V_R}$. At resonance, $\mathbf{V_R}$ is in phase with the source voltage, since the inductive and capacitive impedances cancel one another.

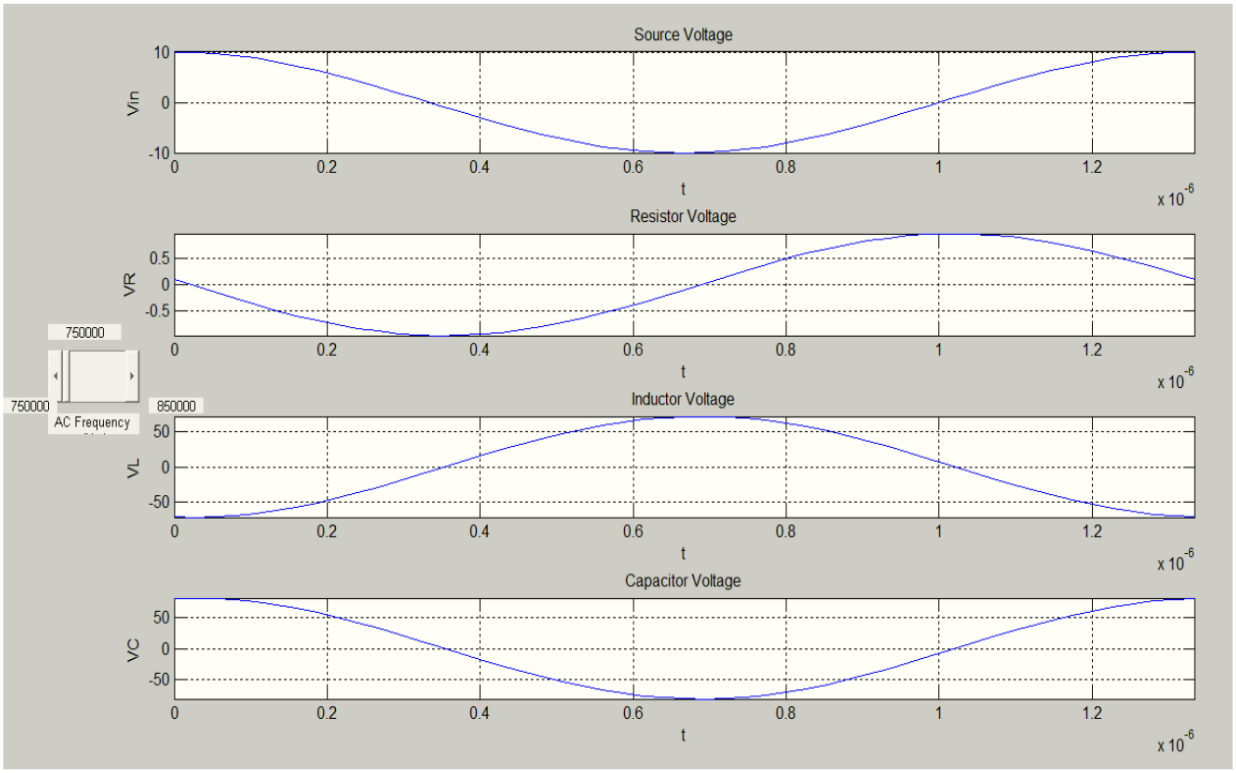
Program 11

```
function gui_demo3_slider
% Computes and plots the sinusoidal steady state response to a
% series RLC circuit. The slider controls the source frequency.

set(figure, 'Name','Slider Example with Plot of RLC Circuit Response')
% Minimum and maximum values for slider
minval = 7.5E5;
maxval = 8.5E5;
% Create the slider object
slider_handle = uicontrol('Style','slider', ...
   'Position',[60,390,100,50], ...
  'Min', minval, 'Max', maxval, 'Value', minval, ...
 'Callback', @callbackfn);
% Text boxes to show the min and max values and slider value
min_slide = uicontrol('Style','text', ...
'Position', [0, 380, 60,15], ...
'String', num2str(minval));
max_slide = uicontrol('Style','text', ...
'Position', [160, 380, 60,15], ...
'String', num2str(maxval));
title_slide = uicontrol('Style','text','Position', [60,360,100,20], ...
                        'String','AC Frequency (Hz.)');
value_slide = uicontrol('Style','text','Position', [70,450,80,15], ...
                        'String', num2str(minval));
```

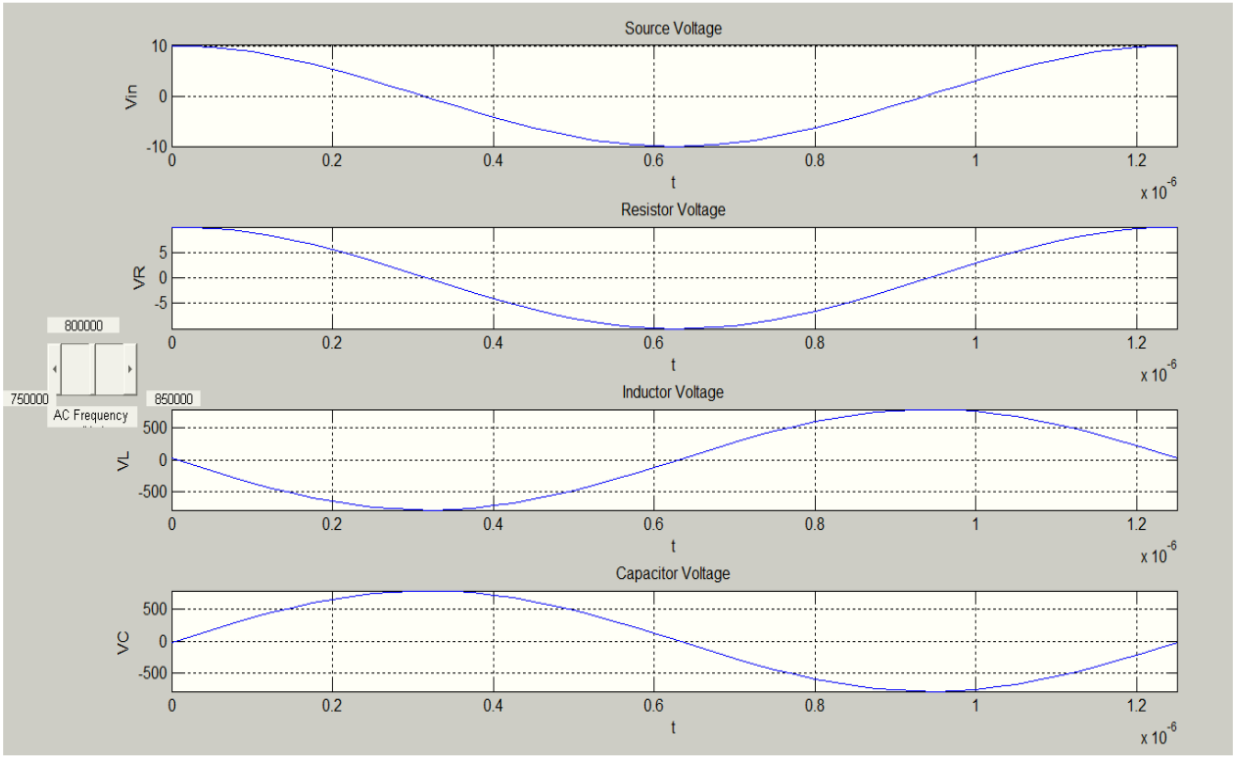
```
% Call back function displays the current slider value & plots the input
% and output voltages
functioncallbackfn(slider_handle,eventdata)
      f = get(slider_handle, 'Value');
set(value_slide,'String',num2str(f))

Vin_peak=10;
      R=1E3; L=15.9E-3; C=2.49E-12;
w =2*pi*f;
ZL=j*w*L;
ZC= -j/(w*C);
w0=1/(sqrt(L*C));
      Q=w0*L/R;
Vmax=Q*Vin_peak;

VR_freq=Vin_peak*R/(R+ZL+ZC);
VL_freq=Vin_peak*ZL/(R+ZL+ZC);
VC_freq=Vin_peak*ZC/(R+ZL+ZC);

      T=1/f; t=0:T/50:T;
      Vin=real(Vin_peak*exp(j*w*t));
      VR=real(VR_freq*exp(j*w*t));
      VL=real(VL_freq*exp(j*w*t));
      VC=real(VC_freq*exp(j*w*t));

subplot(4,1,1)
plot(t,Vin ), grid
xlabel('t')
ylabel('Vin')
title('Source Voltage')
axis([0, T, -max(Vin), max(Vin)])

subplot(4,1,2)
plot(t,VR), grid
xlabel('t')
ylabel('VR')
title('Resistor Voltage')
axis([0, T, -max(VR), max(VR)])

subplot(4,1,3)
plot(t,VL), grid
xlabel('t')
ylabel('VL')
title('Inductor Voltage')
axis([0, T, -max(VL), max(VL)])

subplot(4,1,4)
plot(t,VC), grid
xlabel('t')
ylabel('VC')
title('Capacitor Voltage')
axis([0, T, -max(VC), max(VC)])

end
end
```

Sample Displays

Frequency = 750 kHz.



Frequency = 800 kHz.



Program 12 uses MATLAB‟s random number generator to simulate the addition of noise to a signal. In this example, **signal** is a vector of the sampled values of a single frequency sinusoid at arbitrary unit amplitude and frequency (**f**) of 1000 Hz. The vector **noise** consists of random values between -**A** and +**A**. For **A** = 1, the level of noise being comparable to that of the signal. In this program, a commonly used technique of accumulation (or averaging) is used to reduce the noise. With this method, a vector of sampled values of the signal plus noise is created over an integral number, **M**, of full periods of the sinusoid. By summing the corresponding n-th sampled value in each period and dividing by **M**, we obtain the average value at each sample time **t(n)**. Since the signal is periodic, the average of its sampled values is independent of **M**. The key to reducing the noise by this method is that the noise values are equally likely to be positive or negative, and the summation tends to cancel out values of opposite sign.

Two sliders are provided in the GUI, slider1 for controlling the relative noise amplitude (**A**) and slider2 to control the number of cycles (**M**) over which the averaging is performed.

The composite of signal plus noise values, **x**, over one period of the signal is plotted. With the aid of MATLAB‟s Fast Fourier Transform function (**fft**), discrete (i.e. digital) frequency coefficients are computed and their magnitudes are plotted as a function of discrete frequencies (**k**). The sinusoid as expected results in a single spectral line corresponding to **k**=1. The noise spectrum is observed to be spread out over a wide range of higher frequencies, which is the case when “real” noise is analyzed.

For a noise amplitude (**A**) selected by slider1, four subplots are generated. Time and discrete frequency plots are shown for a single cycle of sampling (**M**=1), and for an arbitrary number of cycles (**M**) set by slider2. The sample plots illustrate the cases **A**=1 with **M**=20 and**A**=5 with **M**=100. For the lower noise level (**A**=1), the signal frequency component is much larger than any noise frequency magnitude and the signal is somewhat recognizable in the time plot. As expected, averaging over 20 cycles reduces the noise frequency components considerably and the signal is much “cleaner”. Increasing the noise level by a factor of 5 results in the signal be unrecognizable. By increasing the number of cycles from 20 to 100 (i.e. by the same factor of 5), the recovered signal looks very much like that for the first case.

Program 12

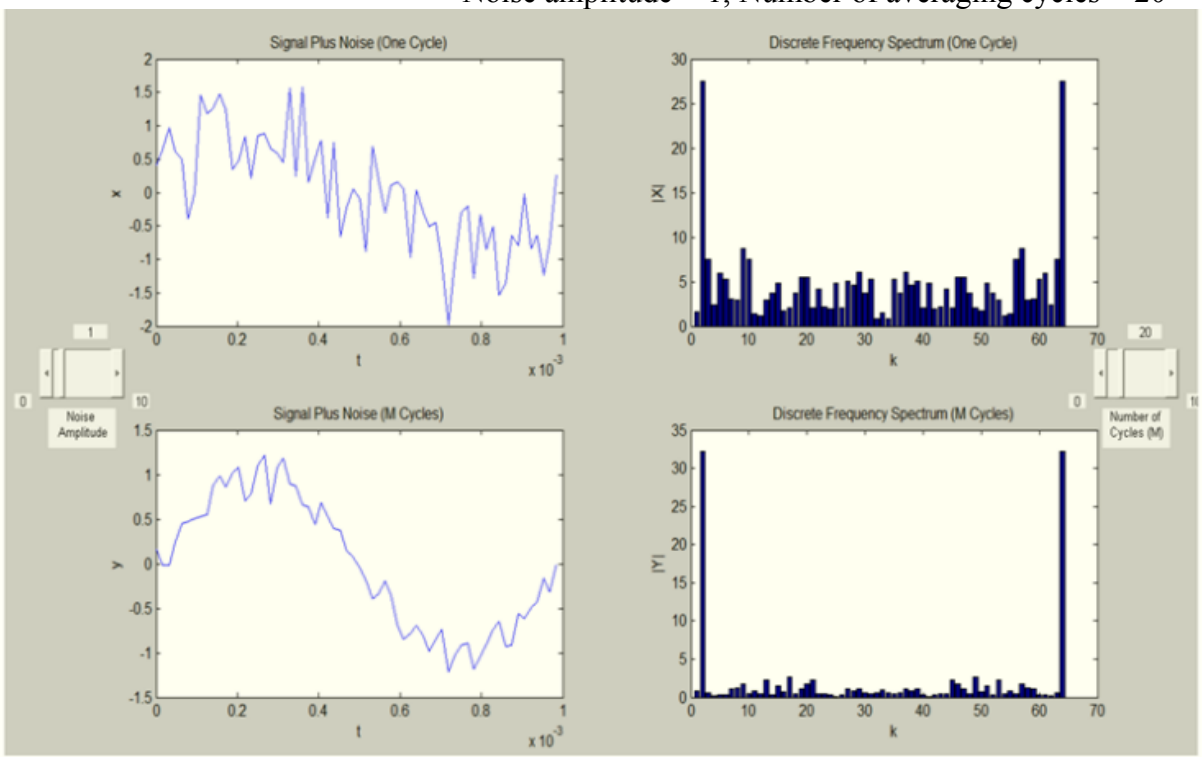
```
function gui_demo4_slider
% Signal Detection in Noise (Accumulation/Averaging Method).
set(figure, 'Name','Slider Example - Signal Detection in Noise (Accumulation/Averaging)')

% Thefirst slider controls the noise amplitude.
% Minimum and maximum value for slider1
minval1 = 0;
maxval1 = 10;
 % Create the slider objects
slider1_handle = uicontrol('Style','slider', ...
   'Position',[50,390,100,50], ...
 'Min', minval1, 'Max', maxval1, ...
'Callback', @callbackfn);
```

```
% Text boxes to show the min and max values and slider value
min_slide1 = uicontrol('Style','text', ...
'Position', [20, 380, 20,15], ...
'String', num2str(minval1));
max_slide1 = uicontrol('Style','text', ...
'Position', [160, 380, 20,15], ...
'String', num2str(maxval1));
title_slide1 = uicontrol('Style','text','Position', [60,340,80,40], ...
                    'String','Noise Amplitude');
value_slide1 = uicontrol('Style','text','Position', [90,450,40,15], ...
                    'String', num2str(minval1));

% The second slider controls the number of cycles to average
% Minimum and maximum value for slider2
minval2 = 1;
maxval2 = 100;
 % Create the slider objects
slider2_handle = uicontrol('Style','slider', ...
  'Position',[1300,390,100,50], ...
'Min', minval2, 'Max', maxval2,'Value', minval2, ...
'Callback', @callbackfn);
% Text boxes to show the min and max values and slider value
min_slide2 = uicontrol('Style','text', ...
'Position', [1270, 380, 20,15], ...
'String', num2str(minval2));
max_slide2 = uicontrol('Style','text', ...
'Position', [1410, 380, 30,15], ...
'String', num2str(maxval2));
title_slide2 = uicontrol('Style','text','Position', [1310,340,80,40], ...
                    'String','Number of Cycles (M)');
value_slide2 = uicontrol('Style','text','Position', [1340,450,40,15], ...
                    'String',num2str(minval2));

% Call back function displays the current slider value & plots the inputand output voltages
functioncallbackfn(slider_handle,eventdata)
      A = get(slider1_handle, 'Value');                    % Noise amplitude
set(value_slide1,'String',num2str(A))

      M = get(slider2_handle, 'Value');                    % Number of Cycles
      M = round(M);
set(value_slide2,'String',num2str(M))

rand('seed', 1234567)                                      % Use same random sequence each time
      f = 1000;                                            % Signal frequency (arbitrary)
     m = 1 : M;                                            % Average over M cycles
     N = 64; n = 0 : N-1;
t = n / (N * f);                                   % Time intervals (N per cycle)
signal = sin(2 * pi * f * t);
noise = A * (2 * rand(1,N) - 1);

     x = signal + noise;                                   % Generate and plot noisy signal
subplot(2,2,1)
plot(t,x)
title ('Signal Plus Noise (One Cycle)')
xlabel ('t')
ylabel ('x')
```

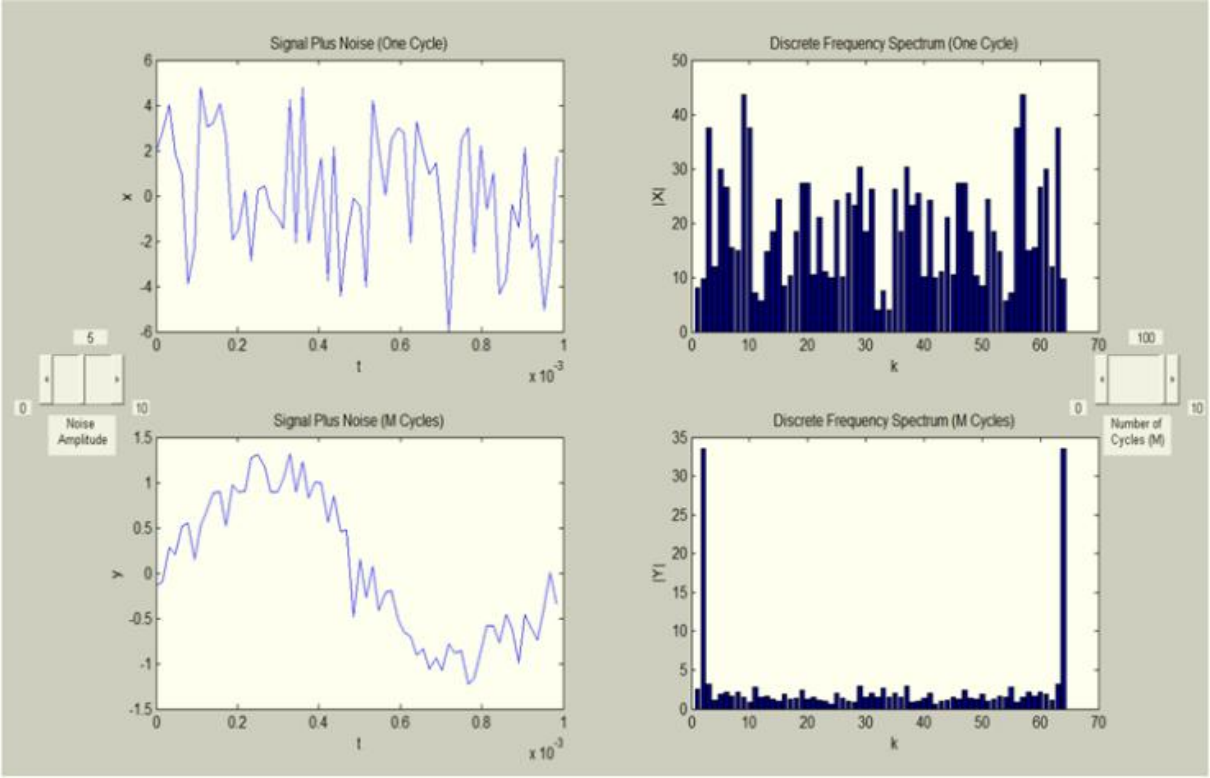
```
X = fft(x);                                          % Generate and plot discrete
subplot(2,2,2)                                       % frequency spectrum (one cycle)
bar(abs(X))
title ('Discrete Frequency Spectrum (One Cycle)')
xlabel ('k')
ylabel ('|X|')

if (M == 1)                                          % Average noise over M cycles
        y = x;                                           % No averaging for 1 cycle
else
        y = signal + sum(A * (2 * rand(M,N) - 1)) / M;   % Averaging for >1 cycle
end
subplot(2,2,3)
plot(t,y)
title ('Signal Plus Noise (M Cycles)')
xlabel ('t')
ylabel ('y')

    Y = fft(y);                                      % Generate and plot
subplot(2,2,4)                                       % frequency spectrum (M cycles)
bar(abs(Y))
title ('Discrete Frequency Spectrum (M Cycles)')
xlabel ('k')
ylabel ('|Y|')

end
end
```

Sample Displays

Noise amplitude = 1, Number of averaging cycles = 20



Noise amplitude = 5, Number of averaging cycles = 100



Neural Network Testing Example

For the final example, a GUI was created for a neural network recognition application. The program is quite complex and will not be shown, but the concepts involved are similar to those used for the Four-function calculator with pop-up menu (Program 6) and the Tic-Tac-Toe game (Program 8).

Programs were originally created to simulate the learning and testing of a character pattern by a neural network, using an algorithm called Perceptron. As an example of pattern recognition, the upper case letters of the alphabet (**A** to **Z**) can be identified by a 5x5 array of pixels as shown below. The network has 25 inputs with 26 different patterns to be learned.

```
..#.. ##### ##### ###.. ##### ##### ##### #...# ##### ....# #...# #.... #...#
.#.#. #...# #.... #..#. #.... #.... #.... #...# ..#.. ....# #..#. #.... ##.##
#...# ####. #.... #...# ##### ####. #..## ##### ..#.. ....# ###.. #.... #.#.#
##### #...# #.... #..#. #.... #.... #...# #...# ..#.. #...# #..#. #.... #...#
#...# ##### ##### ###.. ##### #.... ##### #...# ##### ##### #...# ##### #...#

#...# ##### ##### ##### ##### ##### ##### #...# #...# #...# #...# #...# #####
##..# #...# #...# #...# #...# #.... ..#.. #...# #...# #...# .#.#. .#.#. ...#.
#.#.# #...# ##### #.#.# ##### ##### ..#.. #...# #...# #...# ..#.. ..#.. ..#..
#..## #...# #.... #..## #..#. ....# ..#.. #...# .#.#. #.#.# .#.#. ..#.. .#...
#...# ##### #.... ##### #...# ##### ..#.. ##### ..#.. .#.#. #...# ..#.. #####
```

The network has 26 outputs, each one identifying a unique letter. Thus, using binary notation, the output matrix shown is simply a size 26 identity matrix.

Final outputs after training:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
10000000000000000000000000 A
01000000000000000000000000 B
00100000000000000000000000 C
00010000000000000000000000 D
             :
             :
00000000000000000000000001 Z
```

Learning the patterns is relatively easy with Perceptron, but the interesting problem is then to observe the response to testing the network for input patterns with errors in one or more pixel positions. For example, the patterns below can be viewed as the learned **A** and **A** with errors in four positions (all in the second row):

```
  ..#..          ..#..
 .#.#.            #...#
  #...#          #...#
  #####              #####
  #...#              #...#
```

Learned **A** **A** with errors

The original testing program required the user to enter a string of 25 bits (0‟s and 1‟s) corresponding to the tested pattern. For example, the first five bits for the learned **A** are 00100. The GUI developed has an array of 25 pushbuttons, each of which can be toggled to alternate between • and #. For additional convenience, the pop-up menu right above the buttons enables automatic entry of any of the 26 learned character patterns.

When the testing program is started, the following instructions are first displayed:

Press Enter to open character entry GUI.
To enter a learned pattern, select from pull down menu
Left click to insert or remove "#" in any box.
Click OK to test pattern and press Enter for next pattern.
Close window to end program.

As indicated by the instructions, pressing the **<Enter>** key displays the GUI below:

The learned pattern **A**, when selected from the pop-up menu, is shown below.

Clicking on OK, we see the results below verifying that **A** has indeed been learned. A feature was put into the testing program which also ranks the 26 letters by what are called their "pre-activation" outputs. **A** has the highest value as expected.

Character entered:

```
..#..
.#.#.
#...#
#####
#...#
```

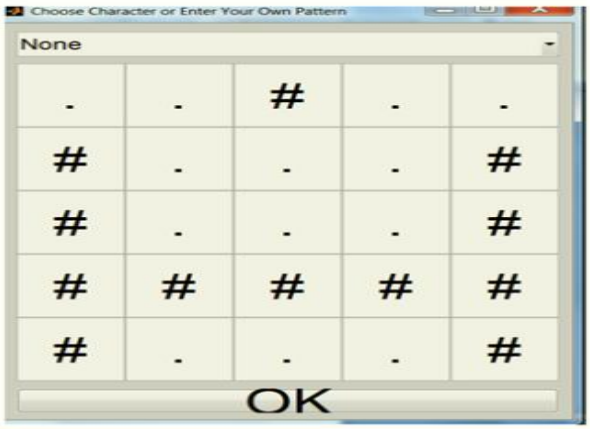
Resulting outputs:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
10000000000000000000000000
```

Sorted outputs before activation (strongest first):

```
ARODYWVCKLSJIHFZTEPNMXGQBU
```

To test the pattern with errors, **<Enter>** is again pressed, which brings up the previous pattern. Clicking on the appropriate four boxes in the second row then puts in the errors, resulting in the new display below.



Clicking on OK, we now see the results below.

Character entered:
```
..#..
#...#
#...#
#####
#...#
```
Resulting outputs:
```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
10000000000000000000?0000
```

Sorted outputs before activation (strongest first):
```
AVWROHTKDNLCQJISBPEZXUYMFG
```

Note that **A** is still identified by the network as the strongest character. A second letter, **V**, shows **?**, indicating that it has almost reached the pre-activation corresponding to a "1" output.

References

[1] **StormyAttaway, "*MATLAB – A Practical Introduction to Programming and Problem Solving*"**, Elsevier, 2012.

[2] **Brian Hahn and Daniel Valentine, Essential *MATLAB for Engineers and Scientists*"**, Elsevier, 2010.

[3] **Stephen Chapman, "*MATLAB Programming for Engineers*"**, Thomson, 2008.

[4] MATLAB On-line Help.

Acknowledgement

Thanks goes to Merrill Preska Steinberg, who was a senior, at the Bergen Academy of Computer Science and Technology High School, last year. Merrill created the GUI function incorporatedin the author"s neural network program for testing alphabetic characters.

Biography

Dr. Howard Silver is currently Professor of Electrical Engineering and a Deputy Director in the GildartHaase School of Computer Sciences and Engineering of Fairleigh Dickinson University. He has co-authored two textbooks: "*32-bit Microprocessors A Primer Plus"*, and "*Modern Instrumentation – A Computer Approach"*. Dr. Silver"s current interests are neural networks, fuzzy logic and computer-aided analysis and design. In 2007 he received the Fairleigh Dickinson University Distinguished Faculty Award for Teaching.
silver@fdu.edu (201) 692-2830