# CM0340 Tutorial 3: MATLAB Functions, Graphics and GUIs

We conclude our brief overview of MATLAB by looking at some other areas:

- MATLAB Functions: built-in and user defined
- Using MATLAB M-files to store and execute MATLAB statements and functions
- Brief introduction to MATLAB graphics and GUI building.






# MATLAB functions

MATLAB makes extensive use of functions
(We have seen many in action already)

- MATLAB provide an extensive set of functions for almost every kind of task
- Extensible through toolboxes — essentially a collection of functions
- Functions can operate on Scalars, Matrices, Structures, sometime is subtly different ways.
- Soon we will learn how to create our own functions






# MATLAB Scalar Functions

Certain MATLAB functions operate essentially on scalars:

- These will **operate element-by-element** when applied to a **matrix**.

Some common scalar functions are:

```
sin  asin  exp              abs   round
cos  acos  log (natural log) sqrt  floor
tan  atan  rem (remainder)   sign  ceil
```



# MATLAB Vector functions

Some MATLAB functions operate essentially on a **vector** (row or column):

- These will act on an $m$-by-$n$ matrix ($m \geq 2$) in a **column-by-column** fashion to produce a **row vector** containing the results of their application to each column.
- **Row-by-row** operation can be obtained by using the transpose, `'`; for example, `mean(A')'`.

Some common vector functions are

```
max   sum   median  any
min   prod  mean    all
sort  std
```






# MATLAB Vector Function Examples

```
>> A = rand(4,4)
A =
   0.8600   0.8998   0.6602   0.5341
   0.8537   0.8216   0.3420   0.7271
   0.5936   0.6449   0.2897   0.3093
   0.4966   0.8180   0.3412   0.8385

>> max(A)
ans =
   0.8600   0.8998   0.6602   0.8385

>> max(max(A))
ans =
   0.8998
```

```
>> mean(A)
ans =
   0.7009 0.7961 0.4083 0.6022

>> mean(A')
ans =
   0.7385 0.6861 0.4594 0.6236

>> mean(A')'
ans =
   0.7385
   0.6861
   0.4594
   0.6236
```



Back

Close

# Matrix functions

Much of MATLAB's power comes from its matrix functions, many are concerned with specific aspects of linear algebra and the like (which does not really concern us in this module)

Some common ones include:

| | | | |
|---|---|---|---|
| `inv` | inverse | `det` | determinant |
| `size` | size | `rank` | rank |

MATLAB functions may have single or multiple output arguments.

For example, `rank()` always returns a scalar:

```
>> A
A =
    0.8600    0.8998    0.6602    0.5341
    0.8537    0.8216    0.3420    0.7271
    0.5936    0.6449    0.2897    0.3093
    0.4966    0.8180    0.3412    0.8385
>> rank(A)
ans =
     4
```

# Return Multiple Output Arguments

`size`, for example, **always** returns 2 values even for a vector:

```
>> X = [1 2 3 4]; size(X)
ans =
     1     4
```

Therefore it's best to usually return multiple arguments to scalar variables:

```
>> [n] = size(A)
n =
     5     5
>> [n m] = size(A)
n =
     5
m =
     5
```

**Note**: In the first call above `n` is returned as vector.



# Writing Your Own Functions

The basic format for declaring a function in MATLAB is:

```
function a = myfunction(arg1, arg2, ....)
% Function comments used by MATLAB help
%
matlab statements;
a = return value;
```

The function **should** be stored as `myfunction.m` somewhere in your MATLAB file space.

- This is a common use of a MATLAB, **M-file**.
- To call this function simply do something like:

  ```
  b = myfunction(c, d);
  ```

- May need to set MATLAB search path to locate the file (**More soon**).






# MATLAB Function Format Explained

```
function a = myfunction(arg1, arg2, ....)
% Function comments used by MATLAB help
%
matlab statements;
a = return value;
```

- A function may have 1 or more input argument. Arguments maybe matrices or structures.
- The return value, in this case, `a`, may return multiple output (see example soon)
- Contiguous **comments** immediately after (**no blank line**) are used by MATLAB to output help when you type: `help myfunction` — **This is very neat and elegant**






# Simple MATLAB Example: Single Output Argument

mymean.m (**Note**: A better built-in `mean()` function exists):

```
function  mean = mymean(x)
% MyMean  Mean
% For a vector x, mymean(x) returns the mean of x;
%
% For a matrix x, mymean(x) acts columnwise.
 [m  n] = size(x);
 if m == 1
   m = n;     % handle case of a row vector
 end
 mean = sum(x)/m;

>> help mymean
MyMean  Mean
For a vector x, mymean(x) returns the mean of x;

For a matrix x, mymean(x) acts columnwise.

>> A = [1 2 3;4 5 6; 7 8 9]; mymean(A)
ans =  4  5  6
```






Back

Close

# Simple MATLAB Example: Multiple Output Argument

```
function  [mean, stdev] = stat(x)
% STAT  Mean and standard deviation
% For a vector x, stat(x) returns the mean of x;
% [mean, stdev] = stat(x) both the mean and standard deviation.
% For a matrix x, stat(x) acts columnwise.
 [m  n] = size(x);
 if m == 1
   m = n;     % handle case of a row vector
 end
 mean = sum(x)/m;
 stdev = sqrt(sum(x.^ 2)/m - mean.^ 2);

 >> help stat
  STAT  Mean and standard deviation
  For a vector x, stat(x) returns the mean of x;
  [mean, stdev] = stat(x) both the mean and standard deviation.
  For a matrix x, stat(x) acts columnwise.
>> [mean sdev] = stat(A)
mean =   4     5     6

sdev =    2.4495    2.4495    2.4495
```






Back
Close

# Useful Function MATLAB commands

`type` : list the function from the command line. *E.g.* `type stat`

`nargin` : The special variable that holds the current number of function input arguments — useful for checking a function has been called correctly. (`nargout` similar)

`disp` : Text strings can be displayed by the `disp()` function: *E.g.* `disp('Warning: You cant type').`

`error` : More useful in functions is the `error()` function which displays text and also **aborts** the M-file. *E.g.* `error('Error: You're an idiot')`

`global` : Variables are **local** to functions. Arguments may be passed in but also `global` variables may be used — See `help global` for more details.

**Debugging** : Many debugging tools are available — See `help dbtype`






# MATLAB Script M-files

Not all MATLAB M-files need contain function declarations:

- A **script** file simply consists of a sequence of normal MATLAB statements.
- It is called in much the same way:

  If the file has the filename, say, `myscript.m`, then the MATLAB command:

  ```
  >> myscript
  ```

  will cause the statements in the file to be executed.
- Variables in a script file are global and will change the value of variables of the same name in the environment of the current MATLAB session.
- An M-file can reference other M-files, including referencing itself recursively.
- Useful to create batch processing type files, inputting data, or just saving last session.



# Creating function and script M-files

There are a few ways you can create an M-file in MATLAB:

- Use MATLAB's or you computer system's text editor:
  - Type commands directly into text editor.
  - Copy and Paste commands you have entered from MATLAB's **Command** or **History** windows to your text editor.
- Use `diary` command:

  `diary FILENAME` causes a copy of all subsequent command window input and most of the resulting command window output to be appended to the named file. If no file is specified, the file 'diary' is used.

  `diary off` : suspends it.

  `diary on` : turns it back on.

  `diary` : initially creates a file 'diary', afterwards toggles diary state.






# MATLAB Paths

- M-files **must** be in a directory accessible to MATLAB.
- M-files in the **present** in current working directory are **always** accessible.
- The current list of directories in MATLAB's search path is obtained by the command `path`.
- This command can also be used to add or delete directories from the search path — See `help path`.
- If you use all lot of libraries all the time then the `startup.m` in your MATLAB top level directory (`/Users/username/matlab`) can be edited to set such paths.
- You can use the Main Menu: File⟶Set Path to set paths also.






# Matlab Graphics

We have already seen some simple examples of how we do simple plots of audio and images.

Lets formalise things and dig a little deeper. MATLAB can produce:

- 2D plots — `plot`
- 3D plots — `plot3`
- 3D mesh surface plots — `mesh`
- 3D faceted surface plots — `surf`.

We are not so concerned with **3D Plots** in this course so we wont deal with these topics here except for one simple example.

See MATLAB `help graphics` and plenty of MATLAB demos (type `demo` or using IDE) for further information.






Back
Close

# 2D Plots

The main function we use here is the `plot` command:

- `plot` creates linear x-y plots;
- If $x$ and $y$ are vectors of the same length, the command:

  ```
  plot(x,y)
  ```

  – opens a MATLAB figure (graphics window)
  – draws an x-y plot of the elements of $x$ versus the elements of $y$.
- Example: To draw the graph of the sin function over the interval 0 to 8 with the following commands:

  ```
  x = 0:.01:8; y = sin(x); plot(x,y)
  ```

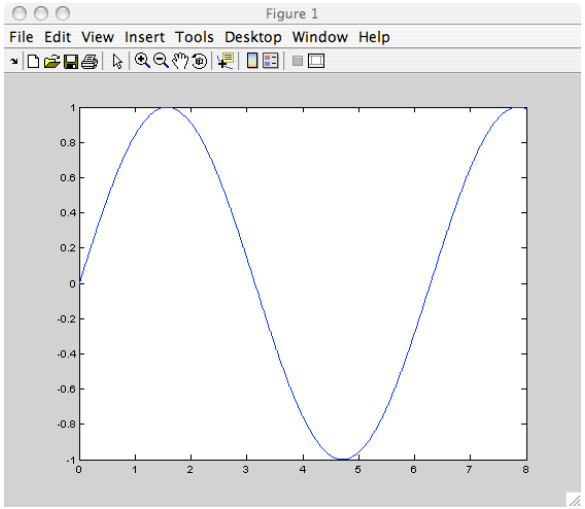
  – The vector $x$ is a partition of the domain with step size 0.01 while $y$ is a vector giving the values of sine at the nodes of this partition



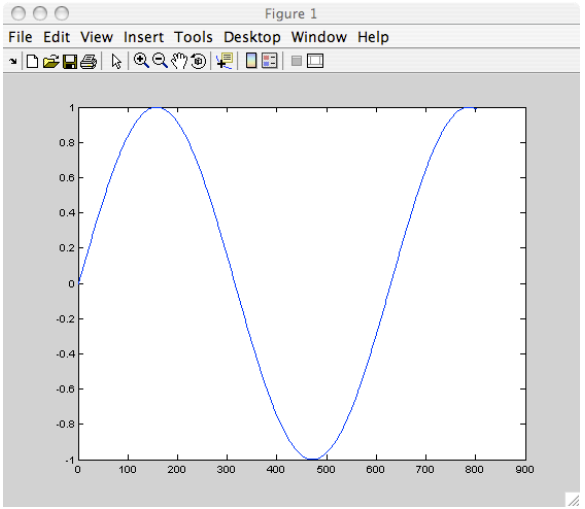



# 2D Plots (cont.)

- It is generally more useful to plot elements of $x$ versus the elements of $y$ using `plot(x,y)`
- `plot(y)` plots the columns of Y versus their index.
  Note the difference in the x axes in the two figures below:

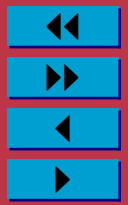



**Result of** `plot(x,y)`




**Result of** `plot(y)`





# Controlling MATLAB Figures

So far we have let `plot` (or `imshow`) automatically create a MATLAB figure.

- In fact it will only create a figure if one **does not exist**.
- If a figure **exists** it will draw to the current figure
  - Possible overwriting currently displayed data
  - This may not be ideal?

MATLAB affords **much greater control** over figures.






# The MATLAB `figure` command

To create a new figure in MATLAB simply enter the MATLAB command:

`figure` or `figure(n)`

where `n` is an index to the figure, we can use later.

- If you just enter `figure` then figure indices are numbered consecutively automatically by MATLAB
- Example:
  - If figure 1 is the current figure, then the command `figure(2)` (or simply `figure`) will open a second figure (if it does not exist) and make it the *current figure*.
  - The command `figure(1)` will then expose figure 1 and make it once more the *current figure*.






## The MATLAB `figure` command (cont.)

- Several figures can exist, **only one** of which will at any time be the designated *current figure* where graphs from subsequent plotting commands will be placed.
- The command `gcf` will return the number of the current figure.






# MATLAB figure control

The figures/graphs can be given titles, axes labeled, and text placed within the graph with the following commands which take a string as an argument.

| | |
|---|---|
| `title` | graph title |
| `xlabel` | x-axis label |
| `ylabel` | y-axis label |
| `gtext` | place text on the graph using the mouse |
| `text` | position text at specified coordinates |

For example, the command:

```
title('Plot of Sin 0-8')
```

gives a graph a title.






Back
Close

# Figure Axis Scaling

- By default, the axes are **auto-scaled**.
- This can be **overridden** by the command `axis`.
- Some features of `axis` are:

| | |
|---|---|
| `axis([`$x_{min}$`,`$x_{max}$`,`$y_{min}$`,`$y_{max}$`])` | set axis scaling to given limits |
| `axis(axis)` | freezes scaling for subsequent graphs |
| `axis auto` | returns to auto-scaling |
| `v = axis` | returns vector $v$ showing current scaling |
| `axis square` | same scale on both axes |
| `axis equal` | same scale and tic marks on both axes |
| `axis off` | turns off axis scaling and tic marks |
| `axis on` | turns on axis scaling and tic marks |

**Note:** The `axis` command should be type *after* the `plot` command.






# Multiple Plots in the Same Figure

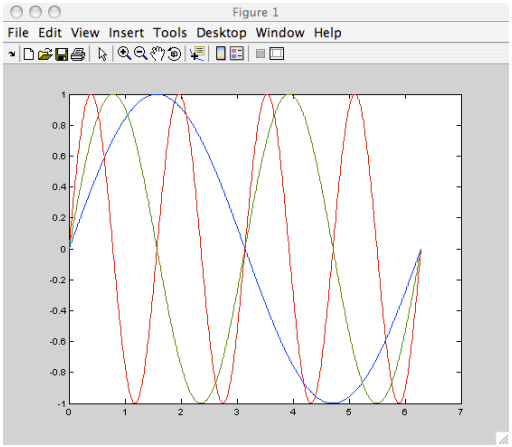
There are a few ways to make multiple plots on a single graph:

- Multiple plot arguments within the `plot` command:

```
x=0:.01:2*pi;
y1=sin(x);y2=sin(2*x); y3=sin(4*x);
plot(x,y1,x,y2,x,y3)
```

- By forming a matrix `Y` containing the functional values as columns and calling the `plot` command:

```
x=0:.01:2*pi;
Y=[sin(x)', sin(2*x)',
sin(4*x)'];
plot(x,Y)
```



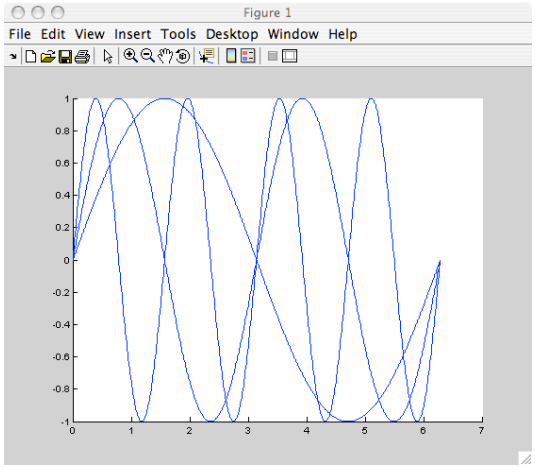

- Using the `hold` command (**next slide**)




# The `hold` Command

- Use the `hold` command:
  - The command `hold on` freezes the current figure
  - Subsequent plots are superimposed on it.
    **Note**:The axes may become rescaled.
  - Entering `hold off` releases the `hold`.
  - Example:

```
figure(1);
hold on;
x=0:.01:2*pi;
y1=sin(x);
plot(x,y1);
y2=sin(2*x);
plot(x,y2);
y3=sin(4*x);
plot(x,y3);
hold off;
```



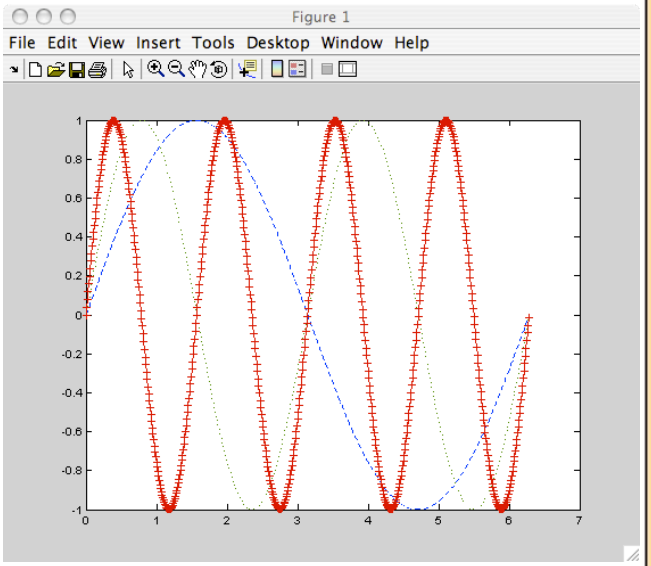
# Line, Point and Colour Plot Styles

- One can override the default linetypes, pointtypes and colors.
- The `plot` function has additional arguments:

Example:

```
x=0:.01:2*pi;
y1=sin(x); y2=sin(2*x);
y3=sin(4*x);
plot(x,y1,'--',x,y2,':',x,...
y3,'+')
```

- renders a dashed line and dotted line for the first two graphs
- the third the symbol + is placed at each node.



# Line and Mark Types

- The line and mark types are

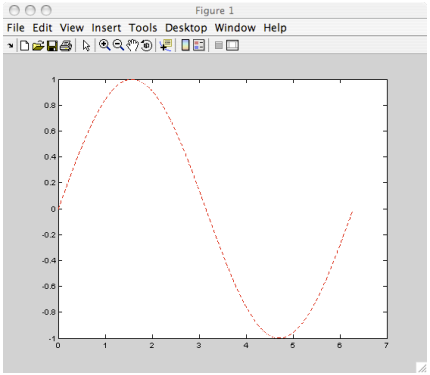
| | |
|---|---|
| Linetypes | solid (`-`), dashed (`--`). dotted (`:`), dashdot (`-.`) |
| Marktypes | point (`.`), plus (`+`), star (`*`), circle (`o`), x-mark (`x`) |
| Colors | yellow (`y`), magenta (`m`), cyan (`c`), red (`r`)<br>green (`g`), blue (`b`), white (`w`), black (`k`) |

- Colors can be specified for the line and mark types.

  - Example:

    `plot(x,y1,'r--')`

    plots a red dashed line:



# Other Related Plot Commands

- The command `subplot` can be used to partition the screen so that several small plots can be placed in one figure — **See** `help subplot`.
- Other specialized 2-D plotting functions you may wish to explore via `help` are:

  `polar, bar, hist, quiver, compass, feather, rose, stairs, fill`






# Saving/Exporting Graphics

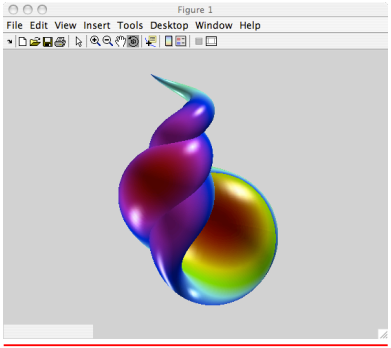
- Use File⟶Save As.. menu bar option from any figure window you width to save
- From the MATLAB command line use the command `print`.
  - Entered by itself, it will send a high-resolution copy of the current graphics figure to the default printer.
  - The command `print` *filename* saves the current graphics figure to the designated filename in the default file format.
    If *filename* has no extension, then an appropriate extension such as `.ps`, `.eps`, or `.jet` is appended.
- See `help print` for more details

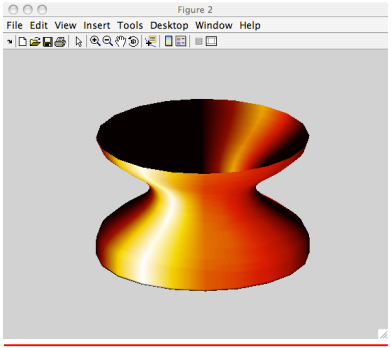


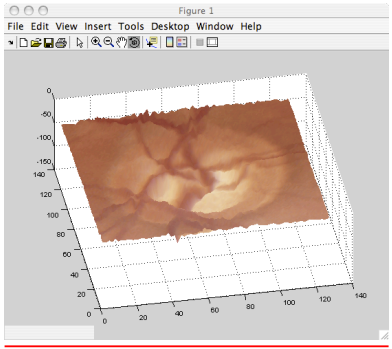



# 3D Plot Example

We have seen some 3D examples in earlier examples and the code is available for study:

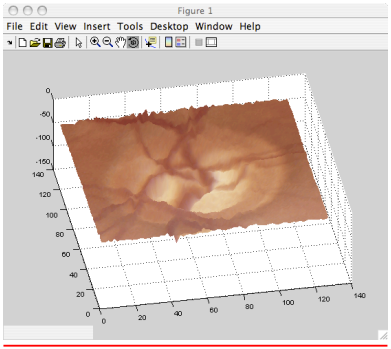





# 3D Plot Example Explained




```
% Read Heightmap
d = imread('ddd.gif');
% Read Image
[r,map] = imread('rrr.gif');
% Set Colourmap
colormap(map);
r = double(r);
d = double(d);
d = -d;
% Set figure and draw surface
figure(1)
surf(d,r)
shading interp;
```

- Two Images store 3D Information: Height map `ddd.gif`, Image: `rrr.gif`
- **Note:** use `imread` to extract image values and colour map
- Set `colourmap` for display
- Use `surf(d,r)` to plot a 3D surface, `d`, with colour values, `r` — see `help surf`
- Set `shading` style.
- `mesh` similar — see `help mesh`
- `plot3(x,y,z)` similar to `plot(x,y)` — see `help plot3`

# MATLAB GUIs

Building a GUI in MATLAB is pretty straight forward and quick.

- You can create a GUI by hand.
- Use MATLAB's GUI Development Environment (GUIDE) to assist you

### Predefined GUI Dialog Boxes

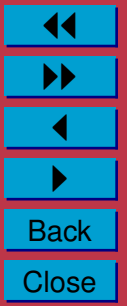
MATLAB Provides a variety of dialog boxes that are ready made for you to use:

**Simple uicontrol objects** : **errordlg** , **helpdlg**, **msgbox**, **warndlg** , **inputdlg** and **questdlg** — pretty self explanatory.

**File/Directory Chooser** : **uigetfile**

**Font and Colour Choosers** : **uisetfont** and **uisetcolor**



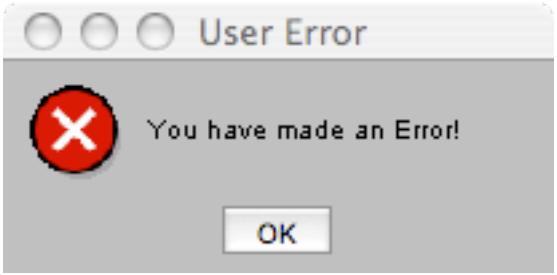


# The Error Dialog box: `errordlg`

To create an error dialog you do something like this:

```
errfig = errordlg('You have made an Error!','User Error','on');
```

This creates:




Note:

- The first string specifies the main error dialog text.
- The second string specifies the dialog window title text
- The third string specifies as **CREATEMODE** which when `'on'` forces MATLAB to use only one error window — do not create another one it exists



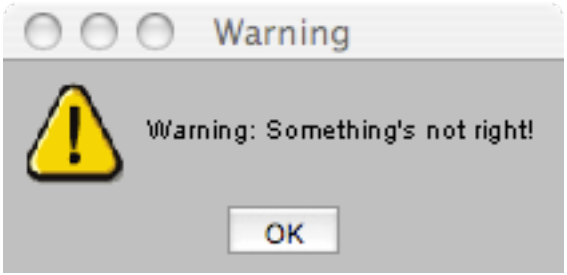



# The Warning Dialog box: `warndlg`

To create a warning dialog you do something like this:

```
warnfig = warndlg('Warning: Something''s not right!', 'Warning');
```

This creates:




Note:

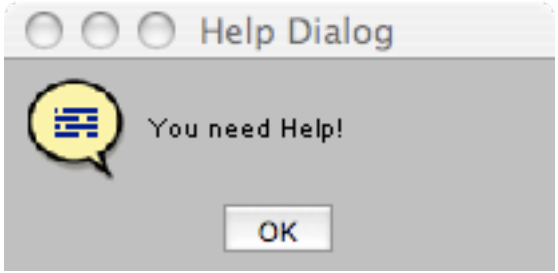
- The first string specifies the main error dialog text.
- The second string specifies the dialog window title text
- **Use** `''` to get a `'` character in a string

# The Help Dialog box: `helpdlg`

To create a help dialog you do something like this:

```
helpfig = helpdlg('You need Help!');
```

This creates:




Note:

- The string specifies the main error dialog text.
- An optional second string could specify the dialog window title text — often unnecessary.






# The Message Dialog box: `msgbox`

**Error, Warning and Help** dialogs are all special cases of a **msgbox**, *E.g.*:

```
errfig = msgbox('You have made an Error!','User Error','error');
warnfig = msgbox('Warning: Something''s not right!', 'Warning', ...
'warn');
helpfig = msgbox('You need Help!','Help Dialog','help')
```

All achieve same as above.

It is more general and can just create a general message:

```
msgfig = msgbox('This is a Message','Msg');
```

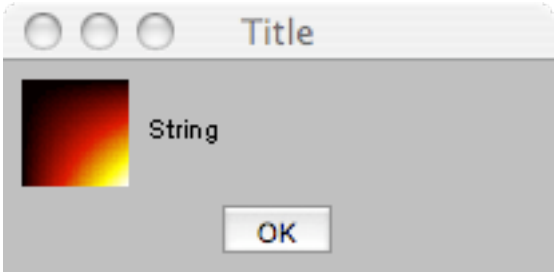
# Customised Message Dialog Icons

You can even be used to create a message with a customised icon with the format:

```
msgbox(Message,Title,'custom',IconData,IconCMap)
```

*E.g.*:

```
Data=1:64;Data=(Data'*Data)/64;
msgfig =msgbox('String','Title','custom',Data,hot(64));
```

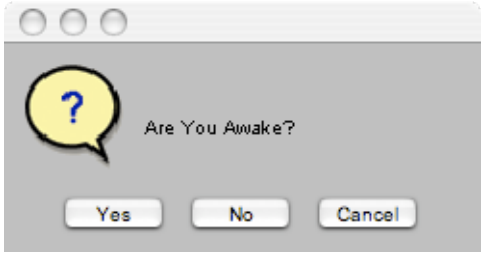


# The Question Dialog Box: `questdlg`

To create a question dialog you do something like this:

```
ret_string = questdlg('Are You Awake?');
```

This creates:




Note:

- The string specifies the main question dialog text.
- The **questdlg** is **modal** — MATLAB always waits for a response.
  - *Note*: **msgbox** dialogs can also be set to be `modal`/`non-modal` as well as `replace`
- `ret_string` stores the text for the reply: `'yes'`,'no' or 'cancel' in this case.





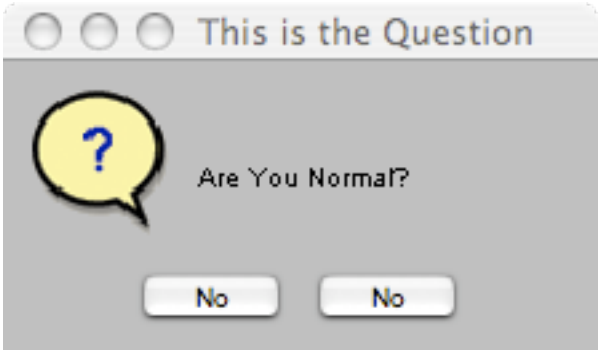

# Customising The Question Dialog Box

The general form of the **questdlg** is:

```
ret_string = questdlg(QuestionString, ....
                      WindowTitleString, ...
                      Button_1_String,...
                      Button_2_String,...
                      Button_3_String,...
                      DefaultString);
```

For example:

```
ret_string = questdlg('Are You Normal?','This is the Question',...
                      'No','No','No');
```







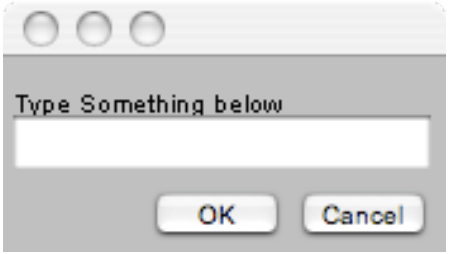


# The Input Dialog Box: `inputdlg`

To create a input dialog you do something like this:

```
Answers = inputdlg('Type Something below');
```

This creates:




Note:

- The string specifies the main input dialog text.
- Answer stores the returned string
- If more than one input and array of strings returned
- This dialog is also **modal**
- Default answers maybe supplied — see `help inputdlg`






# Multiple Input Dialogs

To create multiple inputs you do something like this:

```
Answers = inputdlg({'Q1: What Your Name?',...
                    'Q2: What is your Address?',
                    'Q3: What is your age?'},
                    Questionnaire', [1 3 1]);
```

Note:

- A **cell array** (denoted by `{...}` string specifies the set of questions
- Respective window sizes can be set with an array:
- Answer stores the returned array of strings: `[1 3 1]`, here.

```
Answers =
    'David'
    'COMSC'
    '??'
```

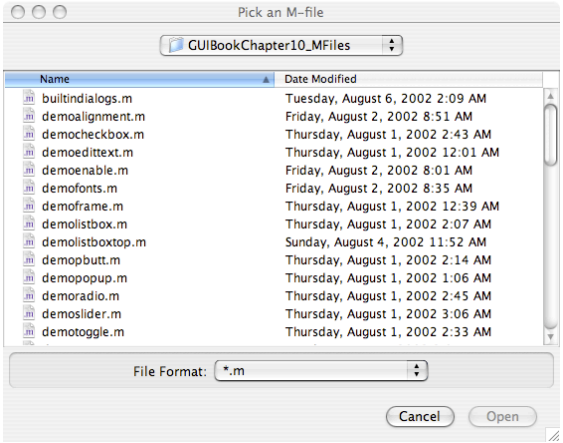
# The File/Directory Selection Dialog Boxes

To create a input dialog you do something like this:

```
[filename, pathname] = uigetfile('*.m', 'Pick an M-file');
```

Note:

- The first string specifies a **file filter**
- The second string is the window title.
- `filename` and `pathname` store the returned respective values of the selected file
- More options — see `help uigetfile`
- **uiputfile** similar — see `help uiputfile`

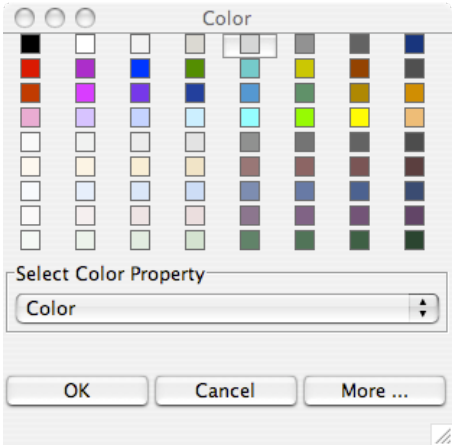


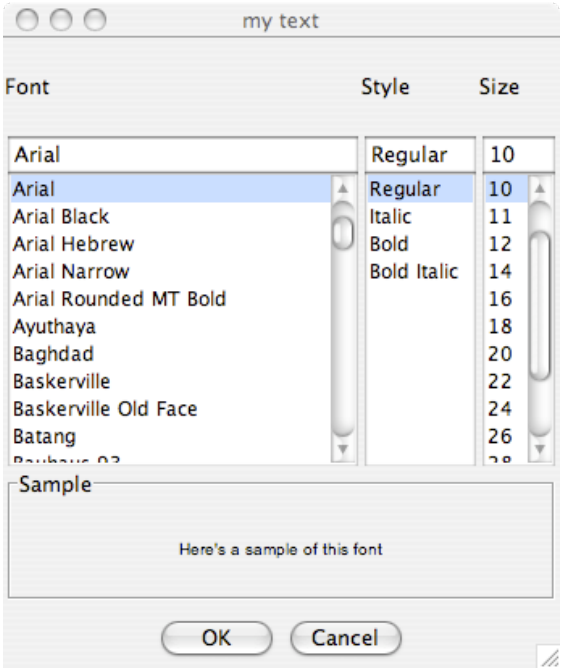
# Setting Fonts and Colours

`uisetfont` and `uisetcolor` can be used to set properties of respective text and graphics objects. *E.g*:

```
text= 'my text';
uisetfont(text);
```

```
myfig = figure(1);
uisetcolor(myfig)
```

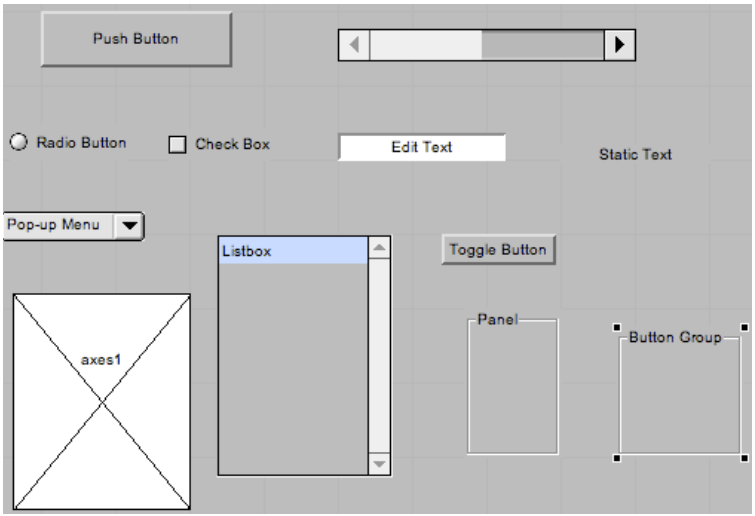


# Uicontrol Elements

MATLAB provides a number basic GUI elements:

- Check boxes
- Editable text fields
- Frames
- List boxes
- Pop-up menus
- Push buttons
- Radio buttons
- Toggle buttons
- Sliders
- Static text labels
- Toggle buttons



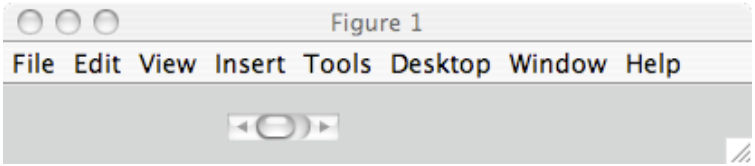
# Manually Creating Uicontrol Elements

To create a `uicontrol` element, use the MATLAB command:

```
handle = uicontrol('Property1Name', Property1Value, ...
                   Property2Name', Property2Value, ...
                   .
                   .
                   );
```

- The first property name usually sets the style: Check box, slider, *etc.*
- Others specify attributes of that object.
- Simple Example:

```
h_slider = uicontrol('Style','slider',...
'units','normalized',...
'position',[.3 .6 .15 .05],...
'String','Radio Button')
```



# Uicontrol Callbacks

Having created and UI element such as a slider, we need to attach a callback to the element:

- Simply set the `'callback'` property value with an appropriate MATLAB function, E.g.

```
h_slider = uicontrol(h_fig,...
'callback','slidergui(''Slider Moved'');',...
```

- Callback can be a *self-referenced* function (as in example below) or an entirely new function (see GUIDE example later).
- Within the callback, you need to access the value of the Uicontrol element:
  - Store data in graphics handle `'userdata'`:

```
set(h_fig,'userdata', h_slider);
```

  - Retrieve values via a few `get`s:

```
h_slider = get(gcf,'userdata');
value = get(h_slider,'value');
```



# Full Slide Callback Code Example

```
function slidergui(command_str)
% Slider
%
% Simple Example of creating slider GUIs.

if nargin < 1
command_str = 'initialize';
end

if strcmp(command_str,'initialize')
        h_fig = figure(1);  clf;

        h_slider = uicontrol(h_fig,...
             'callback','slidergui(''Slider Moved'');',...
             'style','slider',...
             'min',-100,'max',100,...
             'position',[25 20 150 20]);

        set(h_fig,'userdata',h_slider);
else
        h_slider = get(gcf,'userdata');
        value = get(h_slider,'value');
        disp(value);
end;
```

# MATLAB's Graphical User Interface Development Environment — GUIDE

GUIDE provides a WYSIWYG way to assemble your GUI:

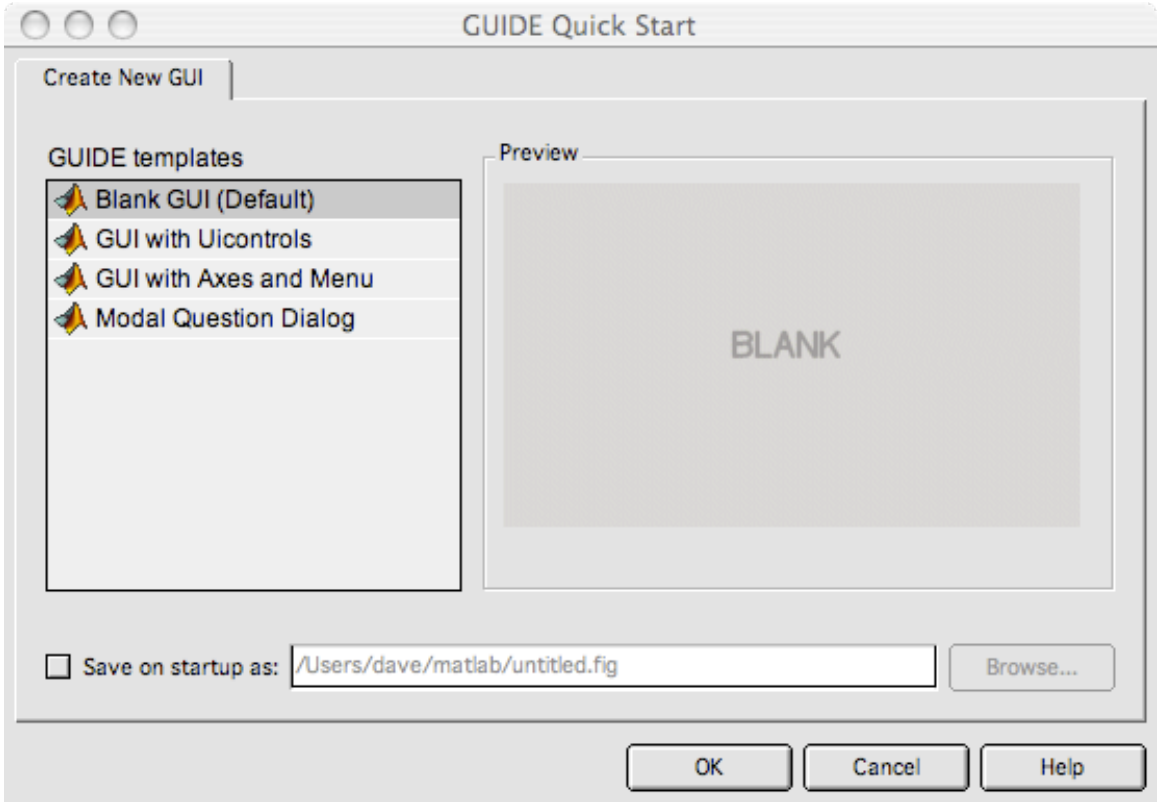
- Designing the overall layout and placement of UI elements is easy
- Editing UI element properties is easy
- Guide provides 4 templates with which to assemble your GUI:
  - A blank GUI (default)
  - GUI with Uicontrols
  - GUI with Axes and Menu
  - Modal Question Dialog
- Can also open existing GUIDE GUIs you have made

**To invoke GUIDE**: Type `guide` at command line.

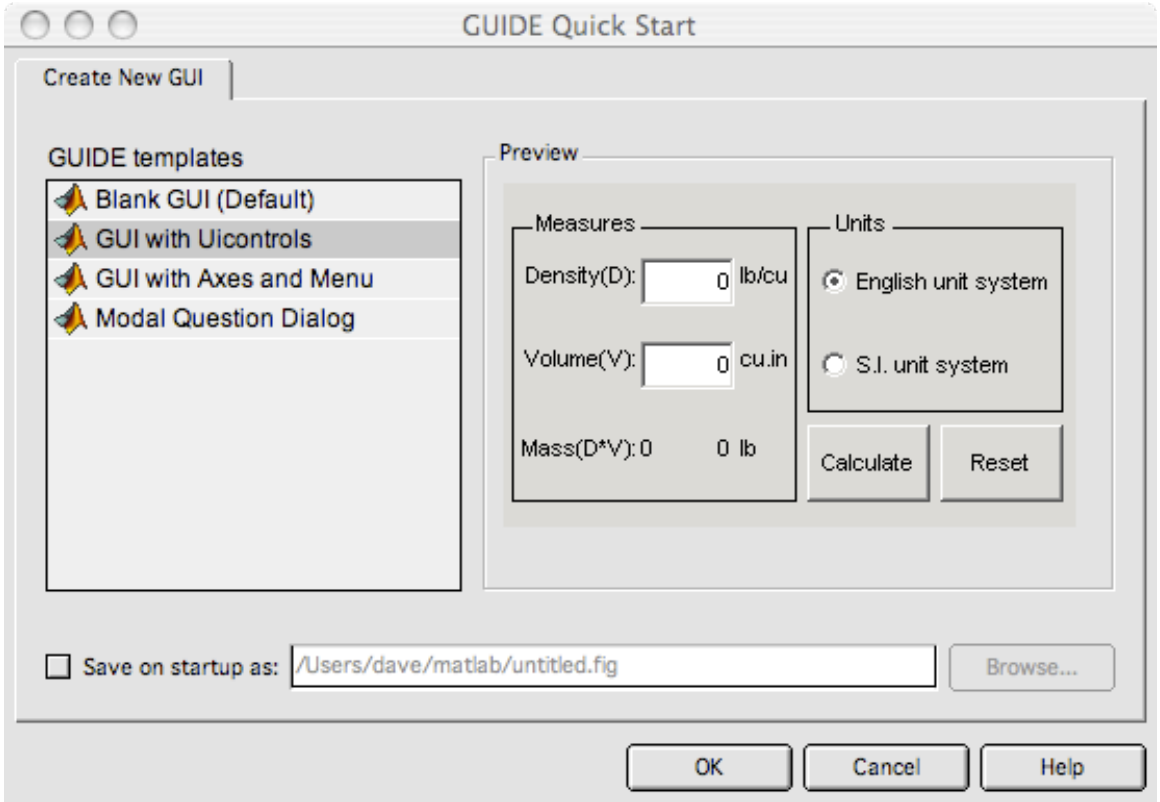




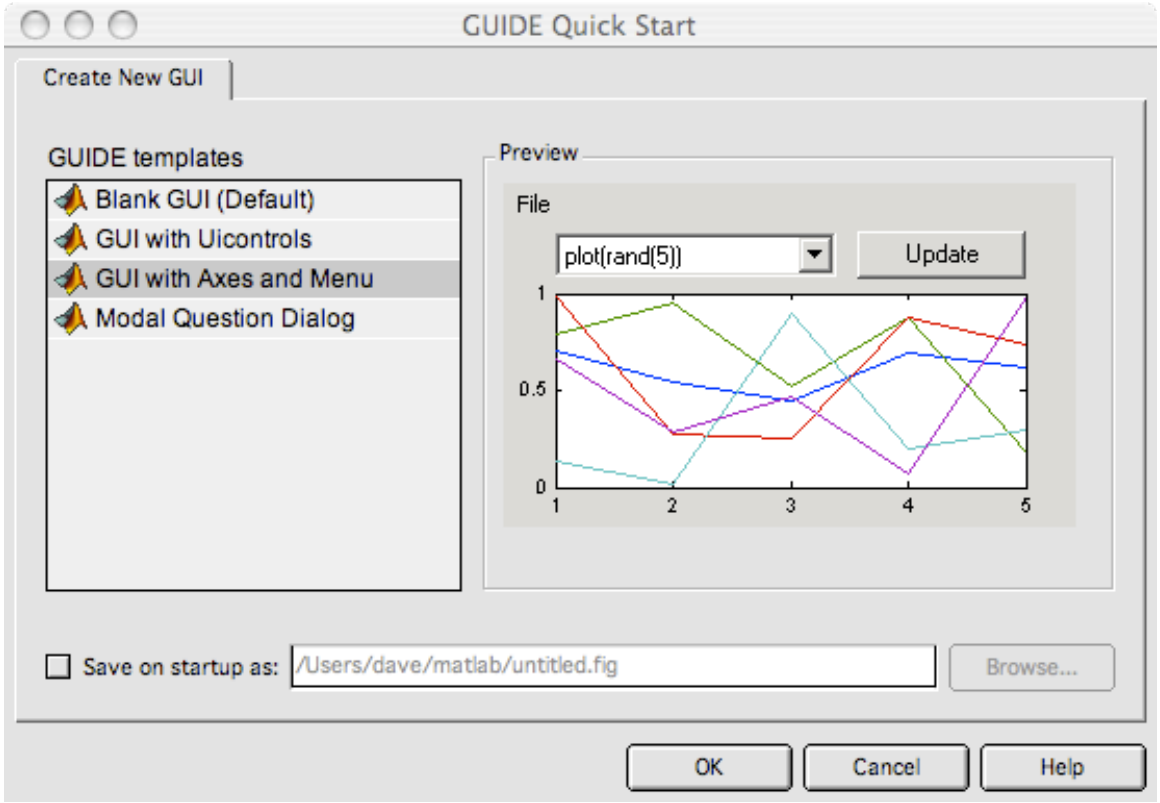

# GUIDE: A blank GUI (default)



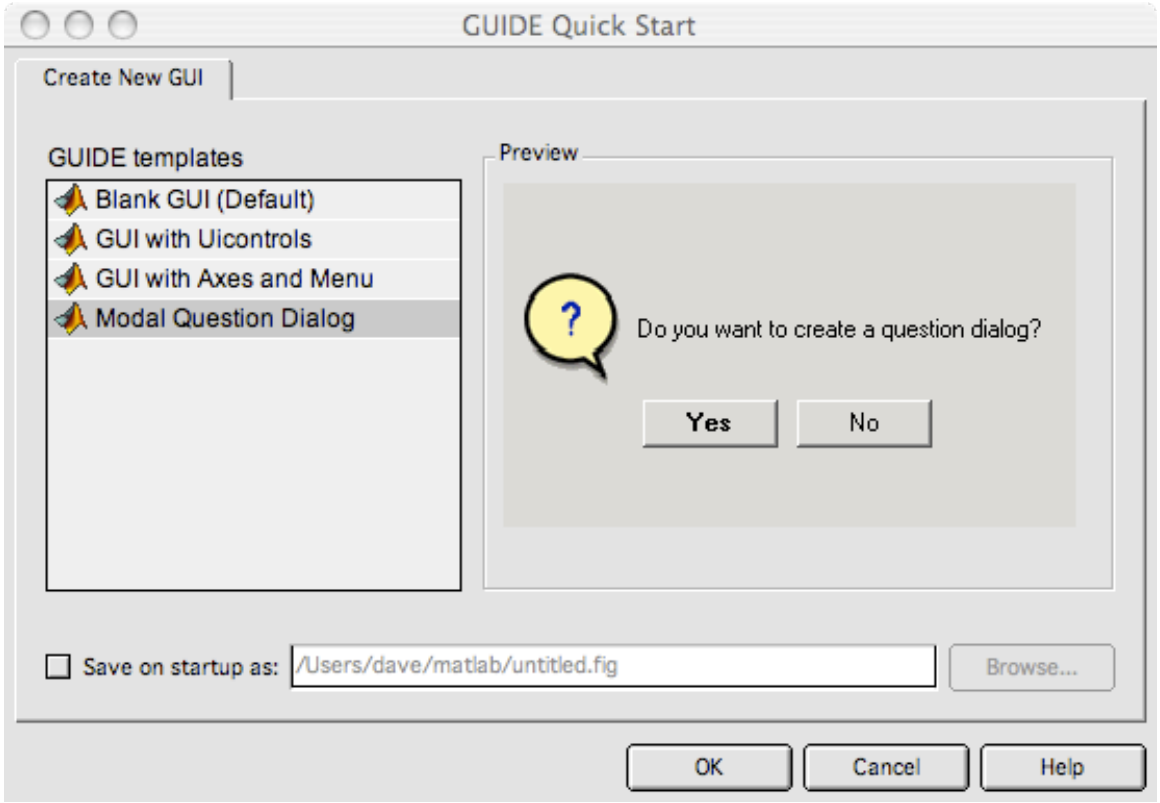
# GUIDE: GUI with Uicontrols



# GUIDE: GUI with Axes and Menu



# GUIDE: Modal Question Dialog



# GUIDE Layout Editor

Whichever GUIDE template you select:

- Click on **Ok** button in chosen template

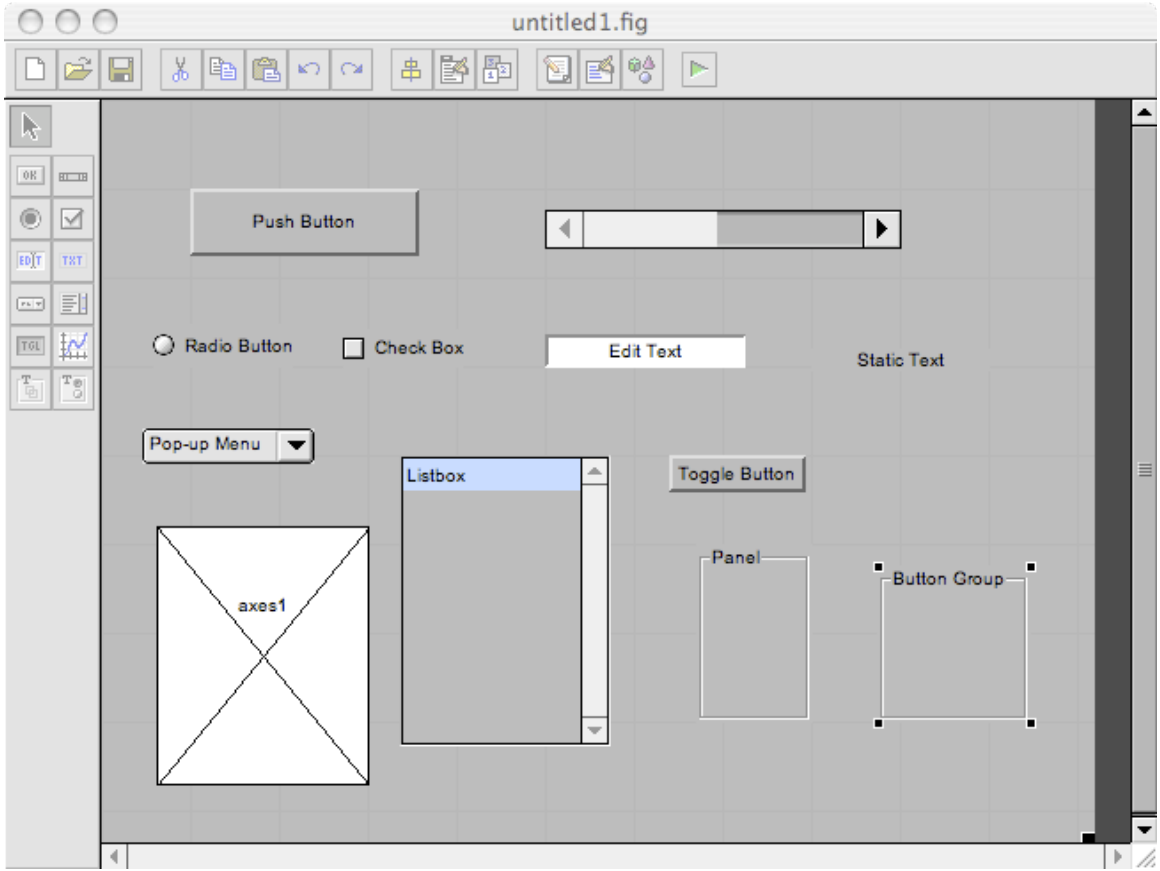
You get the Layout Editor:

- Choose Uicontrol elements on the left panel
- Use **select arrow** to move/resize *etc.*
- Double click on any Uicontrol element to see **Property Inspector** to edit the element — **Example soon**






# Layout Editor with sample Uicontrol Elements



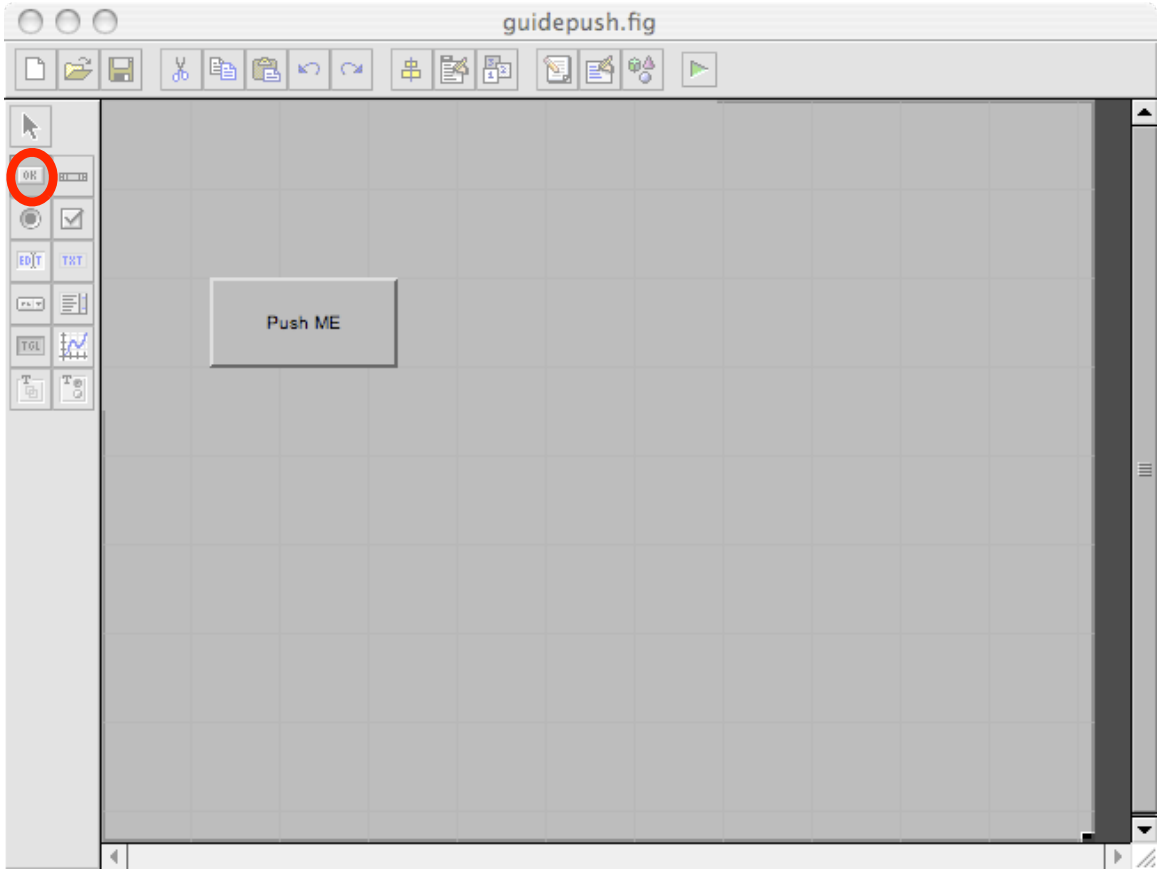
# Creating a Simple GUI

Let's illustrate how we use GUIDE to create a simple push button GUI element:

- Start GUIDE: Type `guide` at command line.
- Select a blank GUI template
- Click on OK Button
- Select a Push Button
- Draw a Push Button
- Double click on the button to invoke **Property Inspector**
- Change the buttons text from `Push Button` to `Push ME`.
- Save session as `guidepush`, for example. **Two files created**
  - `guidepush.m` — run this from the command lin
  - `guidepush.fig` — (binary format) GUI data, read by `guidepush.m`.

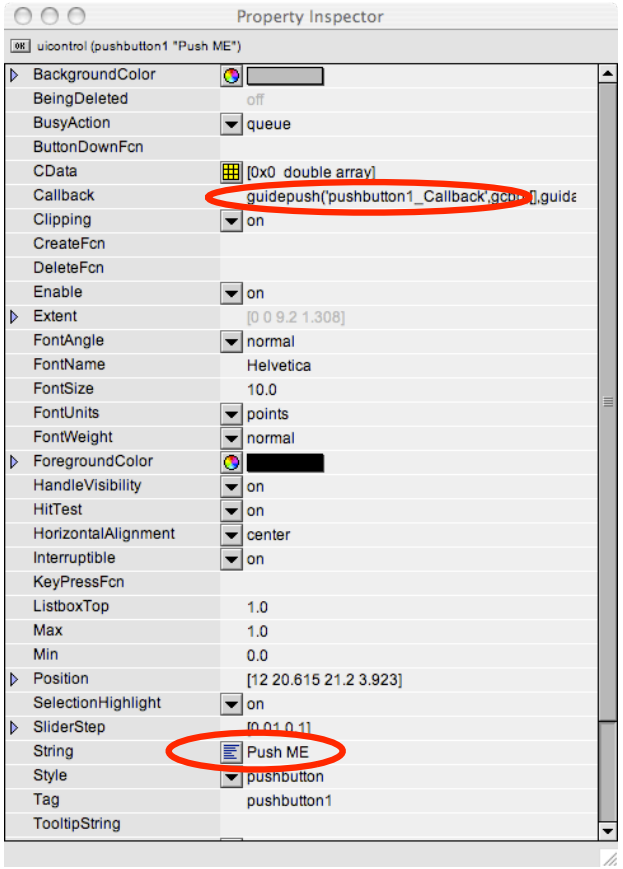


# Example Push Button in Layout Editor



# Push Button Property Inspector

- Note list of properties — useful for command programming reference
- **String** changed to `Push ME`
- Note callback function:
  - Can be changed
  - We edit this callback
- Other useful stuff to edit.



# Adding Functionality to a GUIDE Callback

If you look at the `guidepush.m` file:

- Quite a lot MATLAB code
- **ONLY edit callback** — unless you know what you are doing
- Callback is `pushbutton1_Callback()`
- Let's add some simple functionality to this

```
function varargout = guidepush(varargin)
% GUIDEPUSH M-file for guidepush.fig
%      GUIDEPUSH, by itself, creates a new GUIDEPUSH or raises the existing
%      singleton*.
%
%      H = GUIDEPUSH returns the handle to a new GUIDEPUSH or the handle to
%      the existing singleton*.
%
%      GUIDEPUSH('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUIDEPUSH.M with the given input arguments.
%
%      GUIDEPUSH('Property','Value',...) creates a new GUIDEPUSH or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before guidepush_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to guidepush_OpeningFcn via varargin.

........

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @guidepush_OpeningFcn, ...
                   'gui_OutputFcn',  @guidepush_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before guidepush is made visible.
function guidepush_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guidepush (see VARARGIN)

% Choose default command line output for guidepush
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guidepush wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guidepush_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.


function pushbutton1_Callback(hObject, eventdata, handles)


% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

# Editing the Push Button Callback

Initially the callback has no functioning code:

- Let's add a simple print statemen in traditional Hitchhikers Guide to the Galaxy mode:

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version
% handles    structure with handles and user data (see GUIDATA)

disp('Dont Push Me!');
```

**Clearly a lot more to GUIDE — check MATLAB built in docs and help and textbooks**