

# Data Analysis in Geophysics

## ESCI 7205

Class 17

Bob Smalley

More Matlab.

Final project selection due soon.

# Saving & Exporting Graphics

The default graphics file is a Matlab Figure or `.fig` formatted file.

This format retains the most information about how the figure was created within matlab, but it is not particularly portable

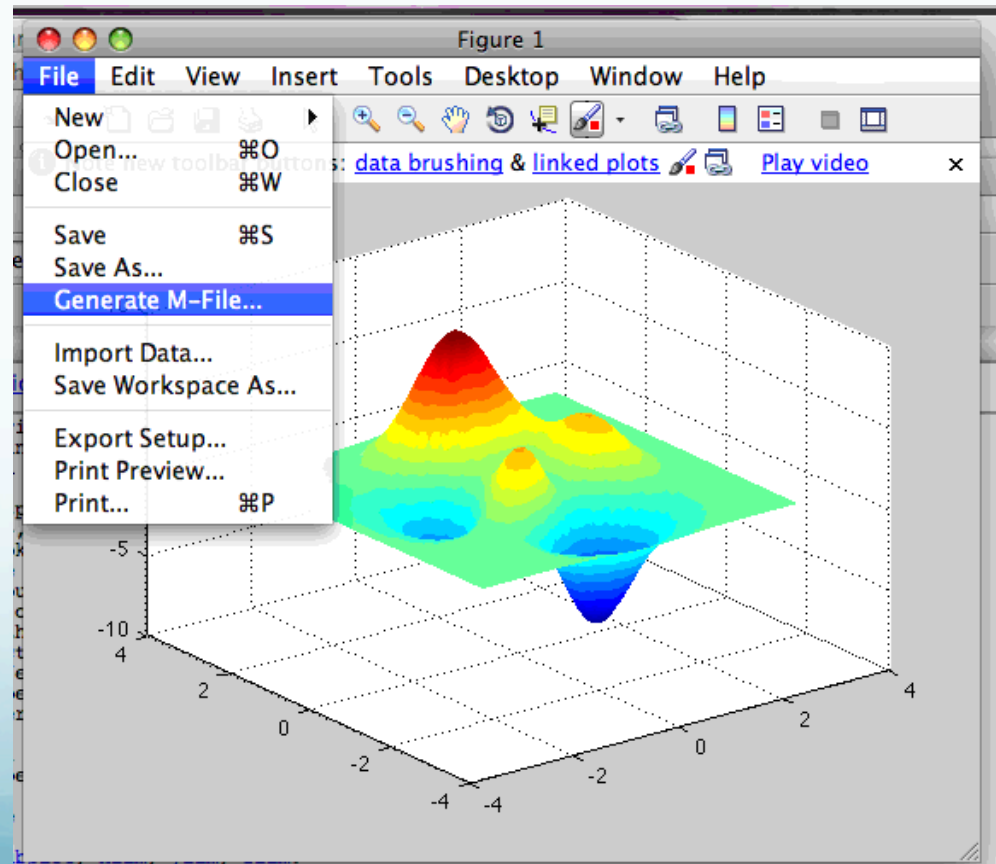
you can also save as `eps` (encapsulated postscript), which can be read by Illustrator

or you can save in one of the picture formats like `tiff` and `jpg` which do not allow additional editing but maintain good resolution

# Cool Feature

After creating the perfect figure, you can generate an m-file so that the figure can be recreated using different data in the future.

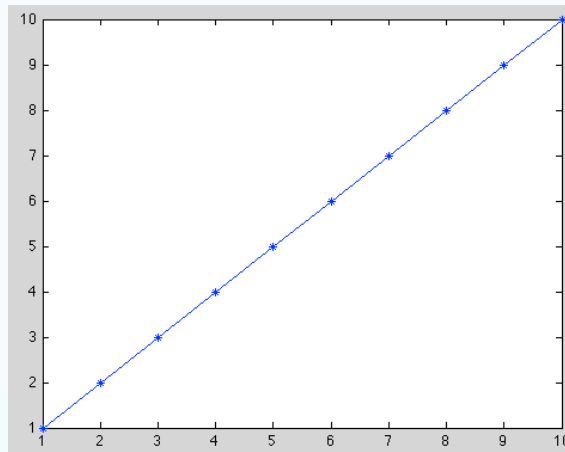
This feature is found under the File drop down menu in the Figure toolbar.



# Advanced graphics: Handle graphics.

Handle graphics provides a rich set (i.e. powerful, difficult and confusing) of functions and properties for generating imagery using Matlab.

```
>> x=1:10;  
>> y=x;  
>> h=plot(x,y, '*-')  
h =  
    171.0034  
>>
```



Notice the output, saved in “h”, from the plot command.

“h” is a number that identifies the “handle” for the graphics object created by the plot command.

To view the information associated with the handle for the plot, use the `get` function. This displays all of the properties of the line we just plotted.

```
>> get(h)
    DisplayName: ''
    Annotation: [1x1 hg.Annotation]
        Color: [0 0 1]
    LineStyle: '-'
    LineWidth: 0.5000
    Marker: '*'
    MarkerSize: 6
    MarkerEdgeColor: 'auto'
    MarkerFaceColor: 'none'
        XData: [1 2 3 4 5 6 7 8 9 10]
        YData: [1 2 3 4 5 6 7 8 9 10]
        ZData: [1x0 double]
    BeingDeleted: 'off'
    ButtonDownFcn: []
    Children: [0x1 double]
    Clipping: 'on'
    CreateFcn: []
    DeleteFcn: []
    BusyAction: 'queue'
    HandleVisibility: 'on'
    HitTest: 'on'
    Interruptible: 'on'
    Selected: 'off'
    SelectionHighlight: 'on'
    Tag: ''
    Type: 'line'
    UIContextMenu: []
    UserData: []
    Visible: 'on'
    Parent: 170.0012
    XDataMode: 'manual'
    XDataSource: ''
    YDataSource: ''
    ZDataSource: ''
    >>
```

```
>> help get
```

```
GET      Get object properties.
```

`V = GET(H, 'PropertyName')` returns the value of the specified property for the graphics object with handle `H`. If `H` is a vector of handles, then `get` will return an `M-by-1` cell array of values where `M` is equal to `length(H)`. If `'PropertyName'` is replaced by a `1-by-N` or `N-by-1` cell array of strings containing property names, then `GET` will return an `M-by-N` cell array of values.

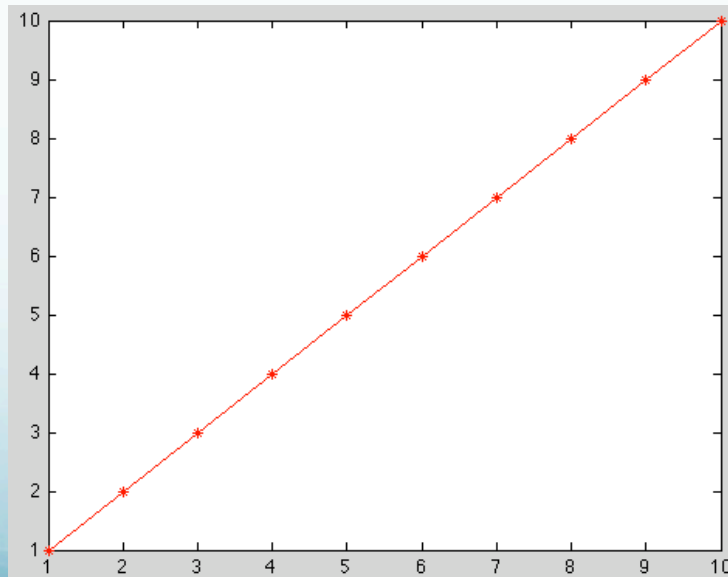
`GET(H)` displays all property names and their current values for the graphics object with handle `H`.

`V = GET(H)` where `H` is a scalar, returns a structure where each field name is the name of a property of `H` and each field contains the value of that property.

Most of the properties can be changed using the set function (a few are read-only).

```
>> set(h,'color',[1 0 0])
```

If you need to change a lot of values, this will improve your typing skills even more than GMT.





Assigning a graphic's object to a variable simplifies modifying the object after creation, but there are also ways to access a graphics object even if you forget.

Access to recently plotted/accessed objects is provided through pre-defined variables

`gcf` (handle to current figure (`get_current_figure`)),

`gca` (handle to current axis (`get_current_axis`)) and

`gco` (handle to current object (`get_current_object`), which is almost always graphics object created as result of last graphics command).

# Use get function with predefined variables to get parameters.

```
>> f=get(gcf)
```

```
f =
```

```
    AlphasMap: [1x64 double]
    BeingDeleted: 'off'
    BusyAction: 'queue'
    ButtonDownFcn: ''
    Children: 170.0044
    Clipping: 'on'
    CloseRequestFcn: 'closereq'
    Color: [0.8000 0.8000 0.8000]
    Colormap: [64x3 double]
    CreateFcn: ''
    CurrentAxes: 170.0044
    CurrentCharacter: ''
    CurrentObject: []
    CurrentPoint: [0 0]
    DeleteFcn: ''
    DockControls: 'on'
    FileName: ''
    HandleVisibility: 'on'
    HitTest: 'on'
    IntegerHandle: 'on'
    Interruptible: 'on'
    InvertHardcopy: 'on'
    KeyPressFcn: ''
    KeyReleaseFcn: ''
    MenuBar: 'figure'
    Name: ''
    NextPlot: 'add'
    NumberTitle: 'on'
    PaperOrientation: 'portrait'
    PaperPosition: [0.2500 2.5000 8 6]
    PaperPositionMode: 'manual'
    PaperSize: [8.5000 11]
```

```
    PaperType: 'usletter'
    PaperUnits: 'inches'
    Parent: 0
    Pointer: 'arrow'
    PointerShapeCData: [16x16 double]
    PointerShapeHotSpot: [1 1]
    Position: [1029 583 560 420]
    Renderer: 'painters'
    RendererMode: 'auto'
    Resize: 'on'
    ResizeFcn: ''
    Selected: 'off'
    SelectionHighlight: 'on'
    SelectionType: 'normal'
    Tag: ''
    ToolBar: 'auto'
    Type: 'figure'
    UIContextMenu: []
    Units: 'pixels'
    UserData: []
    Visible: 'on'
    WindowButtonDownFcn: ''
    WindowButtonMotionFcn: ''
    WindowButtonUpFcn: ''
    WindowKeyPressFcn: ''
    WindowKeyReleaseFcn: ''
    WindowScrollWheelFcn: ''
    WindowStyle: 'normal'
    XDisplay: '/tmp/launch-GGBpjE/:0'
    XVisual: '0x24 (TrueColor, depth
24, RGB mask 0xff0000 0xff00 0x00ff)'
    XVisualMode: 'auto'
```

```
>>
```

And set function with predefined variables to set parameters.

```
>> set(gcf, 'Units', 'Inches');
```

As with GMT you have to know all the values (you can get the property names from the get function), no cheating with menus, etc.

Modified from Mathworks Matlab documentation  
web pages

Matlab assigns a handle to every graphics object it creates. All object creation functions optionally return the handle of the created object.

If you want to access the object's properties (e.g., from an M-file), assign its handle to a variable at creation time to avoid searching for it later.

If you forget to assign the handle when you plot the figure, you can always obtain the handle of an existing object with the `findobj` function or by listing its parent's `Children` property.

# Special Object Handles

The root object's handle is always zero.

The handle of a figure is either:

- An integer
- A floating point number requiring full Matlab internal precision

The figure property `IntegerHandle` controls the type of handle the figure receives.

All other graphics object handles are floating-point numbers.



You must maintain the full precision of these numbers when you reference handles.

(The “full precision” condition means means you cannot read handles off the screen [usually an approximation to the actual floating point value in memory] and retype them, you must store the value in a variable and pass that variable whenever Matlab requires a handle).

# The Current Figure, Axes, and Object

An important concept in the Handle Graphics technology is that of being current.

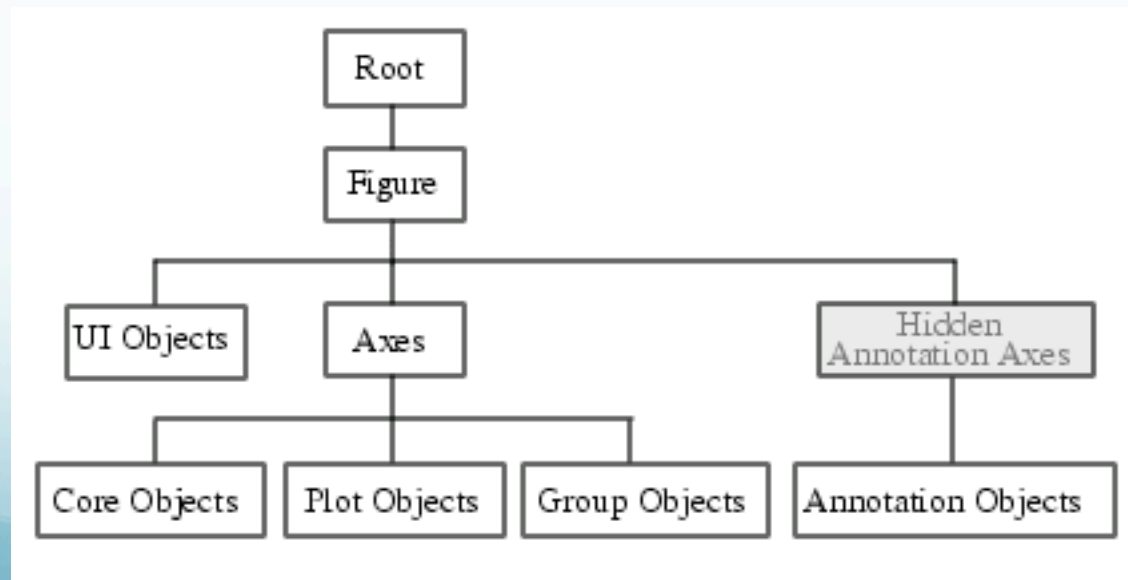
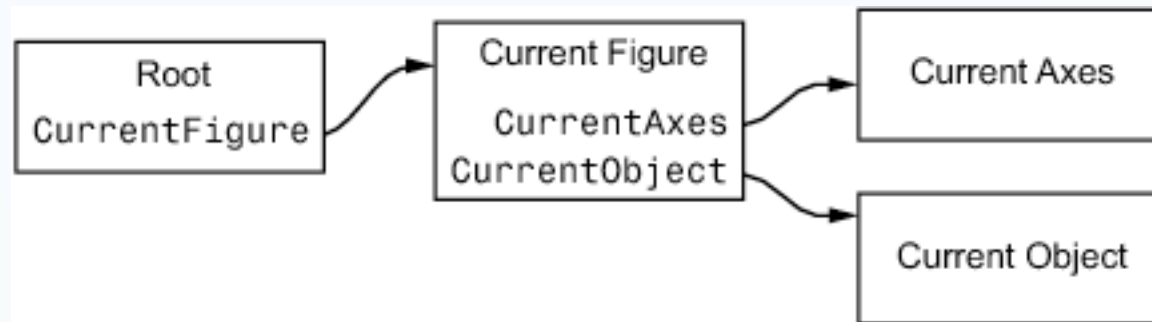
The current figure is the window designated to receive graphics output.

Likewise, the current axes is the target for commands that create axes children.

The current object is the last graphics object created or clicked on by the mouse.

Matlab stores the three handles corresponding to these objects in the ancestor's property list.

Relationship between various objects is hierarchical.



As you work with higher-dimensional plots, or figures with lots of plots on top of each other, the set of handles can seem a little unwieldy.

Fortunately, the set of handles for each figure are nicely organized in a parent-child hierarchy.

The figure handle is at the top.

It's child is the axes.

To see this, type the command `get(gcf, 'Children')` and compare the result to just typing `gca`.

We could also discover that the figure is the parent by typing `get(gca, 'Parent')` and comparing the result to just typing `gcf`.

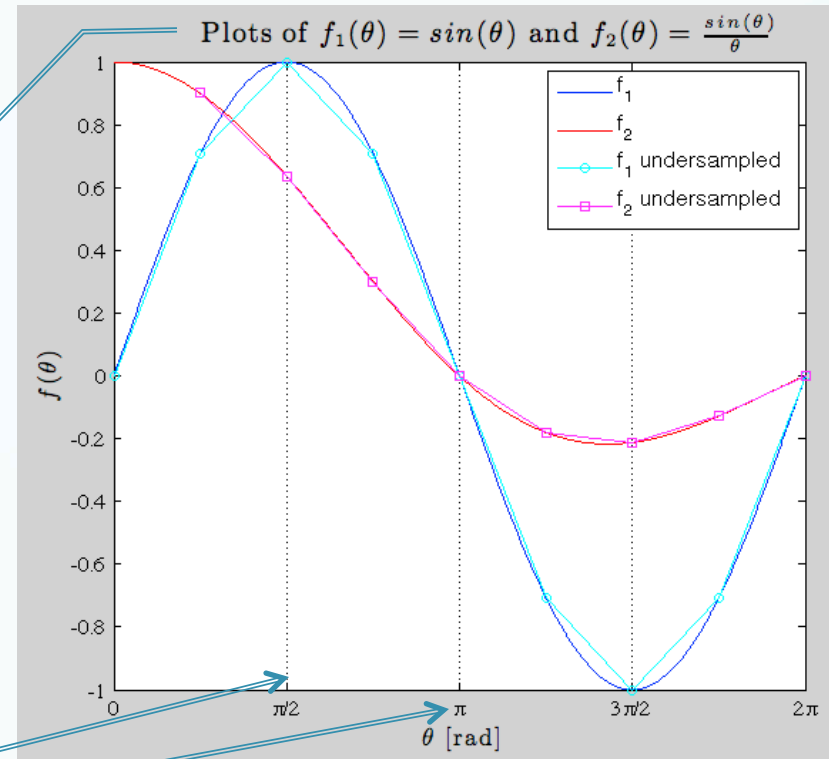
The axes are in turn the parent of each plot, and also of the `xlabel`, `ylabel`, and `title`.

At this point one is tempted to wonder why all this matters... the important result here is that we can find handles that we did not store in variables when the object was created.

```

clear all % clears the variable space
close all % closes all figures
x1 = linspace(0,2*pi,1000);
y1 = sin(x1);
y2 = sin(x1)./x1;
x2 = linspace(0,2*pi,9);
y3 = sin(x2);
y4 = sin(x2)./x2;
figure(2) % opens a new figure with
%ID 2, or goes to figure 2 if open
p1 = plot(x1,y1,'color','b');
hold on;
p2 = plot(x1,y2,'color','r');
p3 = plot(x2,y3,'-o','color','c');
p4 = plot(x2,y4,'-s','color','m');
hold off;
set(gca,'XLim',[0 2*pi],'XTick',[0:pi/2:2*pi],'XGrid','on');
set(gca,'XTickLabel',{'0','p/2','p','3p/
2','2p'},'FontName','Symbol','FontSize',12)
xlabel('$\theta$ [rad]','FontSize',14,'Interpreter','latex')
ylabel('$f(\theta)$','FontSize',14,'Interpreter','latex')
title('Plots of $f_1(\theta) = \sin(\theta)$ and $f_2(\theta) = \frac{\sin(\theta)}{\theta}$
','FontSize',16,'FontWeight','b','Interpreter','latex');
l1 = legend([p1,p2,p3,p4],'f_1','f_2','f_1 undersampled','f_2
undersampled',1);
set(l1,'FontName','Helvetica')

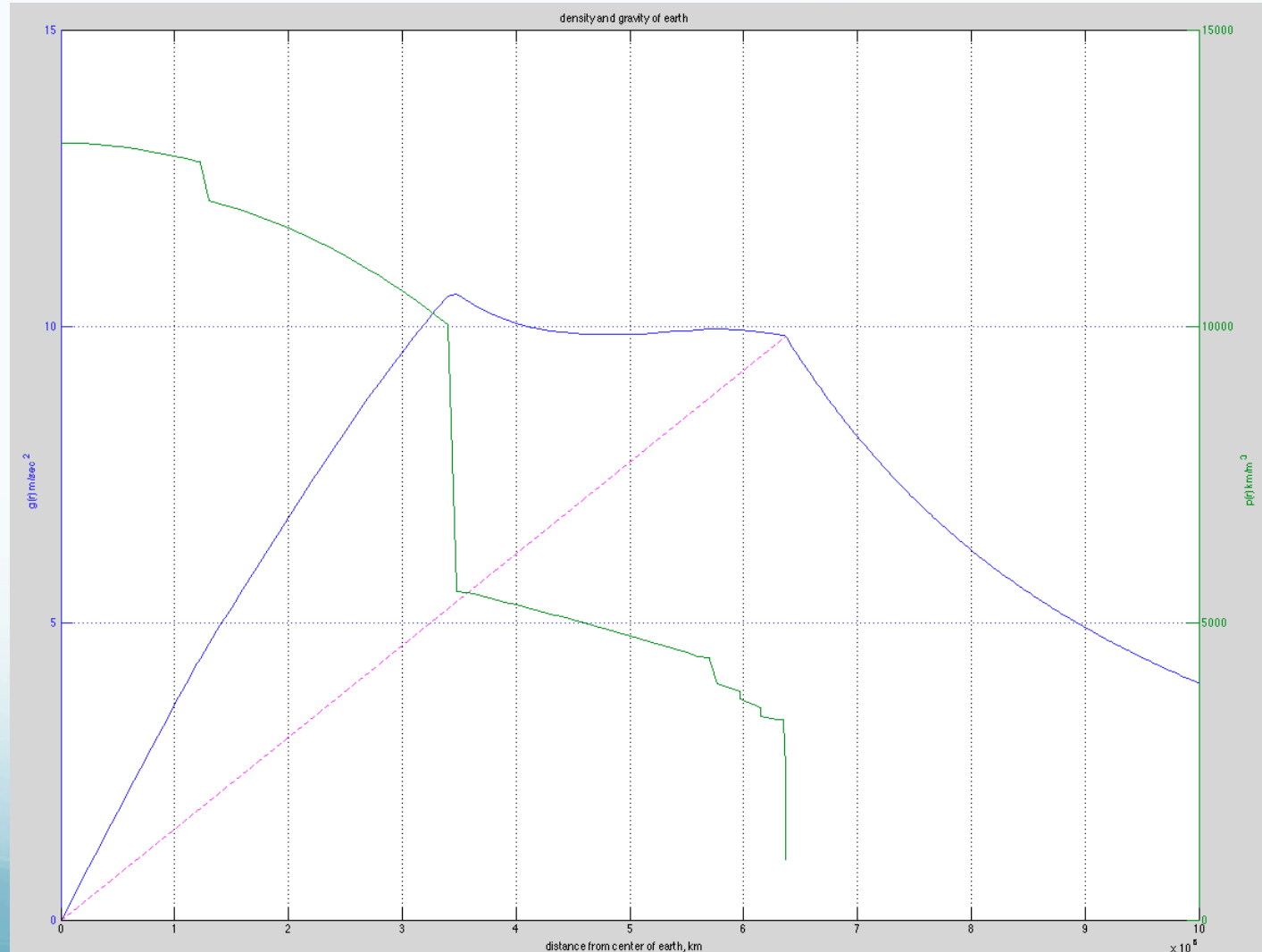
```





```
[AX,H1,H2]=plotyy(rt,gt,r,pofr);
grid
set(get(AX(1),'Ylabel'),'String','g(r) m/sec^2')
set(get(AX(2),'Ylabel'),'String','p(r) km/m^3')
xlabel('distance from center of earth, km')
title('density and gravity of earth')
```

Plotting  
and labeling  
multiple  
axes. Two  
sets  $x$  and  
 $y$  vectors.  
Plot shows  
 $g$  and  $\rho$  as  
functions  
of  $r$ .



# The code to do the calculation and make the last plot is amazingly short.

```
G=6.67300e-11
%load prem density model
prem_r_ro=load('prem_r_ro_fixed.dat');
%make running ave for mass integration
ra=[1 1];
avro=conv(prem_r_ro(:,2),ra)/2;
%calc mass in shells
massinshell=4/3*pi*(prem_r_ro(2:end,1).^3-prem_r_ro(1:end-1,1).^3).*avro(2:end);
%calc total mass inside as fn radius
massinside=cumsum(massinshell);
%calc g inside as fn radius
ginside=G*massinside(:)./(prem_r_ro(2:end,1).^2;
%calc g outside to 10000km as fn radius in steps 25000
rout=[prem_r_ro(end,1):25000:10e6]';
gout=[G*massinside(end)./(rout.^2];

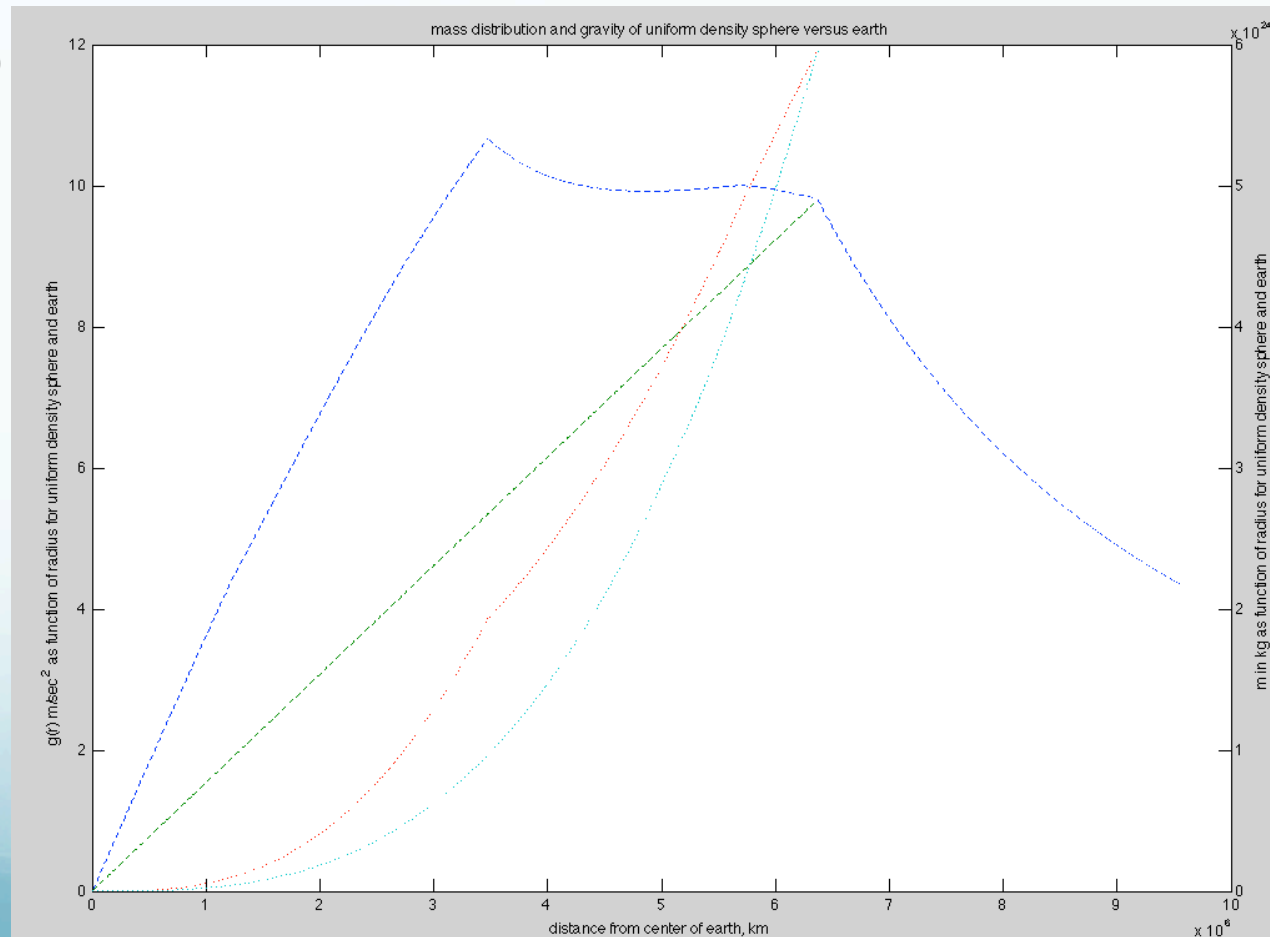
%plot
[AX,H1,H2]=plotyy([0;prem_r_ro(2:end,1);rout(2:end)],[0;ginside;gout
(2:end)],prem_r_ro(:,1),prem_r_ro(:,2));
grid
set(get(AX(1),'Ylabel'),'String','g(r) m/sec^2')
set(get(AX(2),'Ylabel'),'String','p(r) km/m^3')
xlabel('distance from center of earth, km')
title('density and gravity of earth')
hold
%add g inside for uniform density to compare
plot([0 prem_r_ro(end,1)],[0 ginside(end)],'m--')
```

The calculation is above  
this line (and almost half  
of this is comments)

This just plotting.

```
[AX,H1,H2]=plotyy( rt, [gt; rt.*([g(end)/...
r(end)*ones(1,length(r)) repmat(NaN,1,length(ro))]]),r,...
[m; mpave]);
set(get(AX(1),'Ylabel'),'String','g(r) m/sec^2 as function of radius for
uniform density sphere and earth')
set(get(AX(2),'Ylabel'),'String','m in kg as function of radius for
uniform density sphere and earth')
xlabel('distance from center of earth, km')
title('mass distribution and gravity of uniform density sphere versus
earth')
set(H1,'LineStyle','--')
set(H2,'LineStyle',':')
```

Plotting  
multiple  
functions on  
same axes.  
Multiple y  
vectors for  
each x.



# Matlab

Misc stuff

# Saving and reading your workspace

```
>> eig_mov
>> whos
```

Name	Size	Bytes	Class	Attributes
D	2x2	32	double	
V	2x2	32	double	
. . .				
cnt	1x1	8	double	
x	2x361	5776	double	

```
>> save eig_mov_ex.mat
```

Saves workspace in file stuff.mat

```
>> clear
>> whos
>> load eig_mov_ex.mat
>> whos
```

Name	Size	Bytes	Class	Attributes
D	2x2	32	double	
V	2x2	32	double	
. . .				
cnt	1x1	8	double	
x	2x361	5776	double	

```
>>
```

Saving what you type

```
>> diary everythingItype.txt
```

Saves everything you type

```
>> diary
```

To turn it off

## Garbage collection

Any system such as Matlab that maintains an environment with variables continually being created and destroyed must have a form of “garbage collection” to remove dead (or no longer needed/used) space.

Unfortunately, Matlab has no automatic garbage collection mechanism.



The function `clear` allows the user to manage his workspace and do his own house cleaning.

Even that is not enough, since other temporary arrays might be created and destroyed whenever M-files are run.

In place of garbage collection, there is a Matlab function called

pack

which saves all the active variables in the entire workspace, clears the workspace, and then loads the saved variables.

This is time-consuming, but it is the best way to get some room to work if memory limits start to hinder your progress.

Supressing “Current plot held”, etc. messages  
be more specific - hold (‘on’), etc.

# Importance of thinking through how to program something

<http://www.joelonsoftware.com/articles/fog0000000319.html>

(so the world is not stuck with your mistake forever).