Matlab functions

A more efficient method of to calculating the gravitational anomaly due to a sphere is to create a matlab function. As with the matlab script, a matlab function file must have a ".m" extension, but in addition the file MUST have a first line, a function signature, that tells matlab it is a function and declares the arguments and the results of the function.

This is an example function with 2 input and 2 output arguments. It is stored in the file mymultdiv.m. All function files end with .m and the first line begins with the word function and defines the input and output variables for the function

```
function [e,f] =  mymultdiv(a,b)
% MYMULTDIV - my multiply and divide function
%   [e,f] = mymultdiv(a,b)
%  a,b - two input numbers
%  e,f - two output values
% These comments will appear when you type 'help mymultdiv'

% this is a private comment as the is a gap after the previous comments.
% Also all variables in the function are private and disapp

% keyboard
e = a * b;
f = a / b;
% --- End of Function
```

Aside: Debugging scripts & functions

If you are having problems getting your scripts to work and are getting errors, a simple debugging trick is to add the command keyboard to your script just before the problem. This command will stop your script at the position of the keyboard command and allow you to type commands interactively into the command window. The only difference is the prompt will change to K >> If you are satisfied with the results and want to resume to your script simply type the command return

```
K>>  return
```

If you found an error, you need to get the normal prompt back by typing

```
K >> dbquit
>>
```

Examples of calling the mymultdiv function

```
a = 1;
b = 2;
e = 3;
f = 4;

[g,h] = mymultdiv(7,3)
g =
   21
h =
   2.3333
>> a       % The original variables with the same name are not altered by the function
a =
    1
>> b
b =
    2
>>e
e =
    3
>>f
f =
    4
```

The order and number of values in the function call determines the values assigned to the input arguments within the function. Similarly the order of the output variables determines the names of the variables that are assigned the returned values.

```
>> [g,h] = mymultdiv(e,f)   % in function a has value 3 and b has value 4
g =
   12
h =
   0.7500

[g,h] = mymultdiv(7,3,4)     % Calling with Too many input arguments causes an error
??? Error using ==> mymultdiv
Too many input arguments.

>> [g,h] = mymultdiv(7)       % As can too few
??? Input argument 'b' is undefined.
Error in ==>mymultdiv.m
On line 14  ==> e = a * b;

g = mymultdiv(7,3)    % But calling with less outputs is OK
g =
   21
```

Advanced: Calling a function with a reduced number of arguments

We can allow a function to be called with fewer arguments by using the special variable nargin set by Matlab when a function is called. Similarly Matlab sets the variable nargout to tell us how many output arguments the caller is expecting. Adding the following lines to beginning of the function mymultdiv.m allows a default value for b to set in the function if the

```
if (nargin < 2)   % Function called with less than two arguments
 b = 1;           % Supply a default value for b
end

>> [g,h] = mymultdiv(7)
g =
    7
h =
    7

>> [g,h] = mymultdiv(7)
```

Finding information about functions

```
help mymultdiv   % see help comments

  MYMULTDIV - my multiply and divide function
    [e,f] = mymultdiv(a,b)
  a,b - two input numbers
  e,f - two output values
 These comments will appear when you type 'help mymultdiv'

>> type mymultdiv   % show the entire function code
>> type linspace    % works for matlab supplied functions
```

Location of Functions

Function files can be found anywhere in your path. The default places include the current directory and the directory containing the file startup.m. New directories can be added to your path

using the command addpath. A suitable place to put these commands is in the file startup.m, which is executed every time matlab starts up.

Function for gravity anomaly of a sphere

For example if we create a function version of the spherical anomaly calculation, say fnsphereg.m, it will have the following lines

```
function gz = fnsphereg(h,m,x)
% FNSPHEREG - calculates the vertical gravity anomaly gz of a buried sphere
%
%    gz  = fnsphereg(h,m,x)
%
%   h - depth to center of spherical anomaly (m)
%  m - anomalous mass of sphere (kg)
%  x  - vector of surface observation points
% gz - vertical gravity anomaly in mgal.

G = 6.67e-11;
r = sqrt(h*h + x.*x);                  % radial distance to measurement point
cosz = h * ones(1,length(x))./r;  % need to  vertical component of the anomaly.
gz = G*m*cosz./r.^2 * 1e5;       % calculate the anomaly, converting result to mgal.
```

Once you have written a function, using it is the same as using one of matlabs own functions such as plot, or sin. You must call a function with the right number of arguments, fnsphereg expects 3 arguments, the depth of burial, the anomalous mass, and the observation positions and these must be supplied in the correct order. In the first call to to fnsphereg, the depth is 5 m, the excess mass 1000 kg, and the positions are the vector x. Effectively the function makes the assignments h = 5; m = excess_m; & x = x; before it performs the other commands in the M-file. The result of the anomaly calculation is the vector gz, but since the function is called using

```
 g5 = fnsphereg(5,excess_m,x);
```

Matlab takes care of the assignment g5 = gz; that was performed explicitly in the script version above.

The script to calculate the two anomalies becomes

```
>> x = [-5:0.1:5];
>> excess_m = 1000;  % excess mass in kg
>>  g5   = fnsphereg(5, excess_m, x);
>> g10 = fnsphereg(10, excess_m, x);
>> plot(x,g5)
>> hold on
>> plot(x,g10,'r')
```