

# Matlab

## Introduction:

basic commands,  
loops & conditionals,  
scripts & functions

MRC  
Cognition and  
Brain Sciences Unit

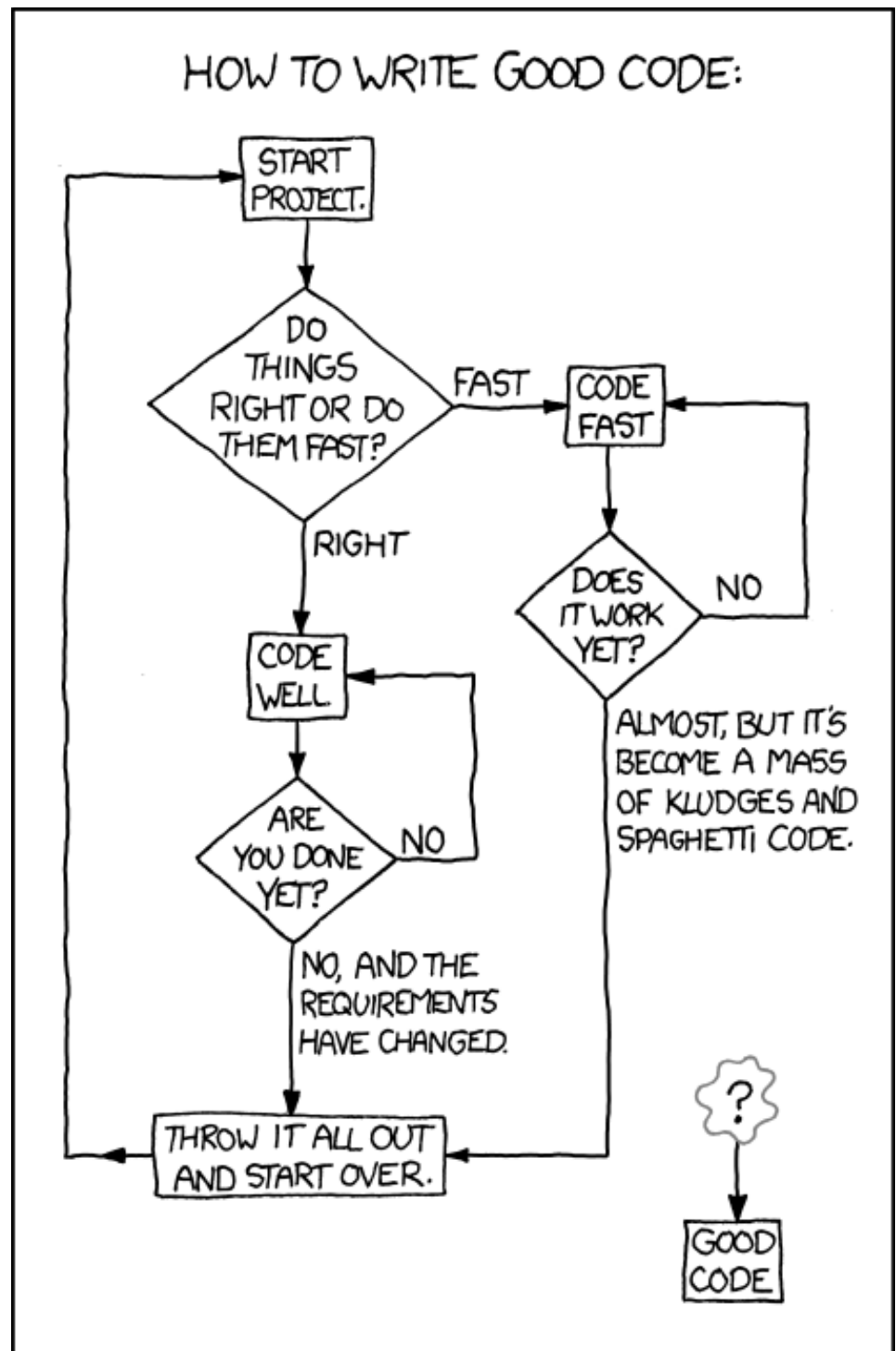
Jason Taylor

MRC CBU

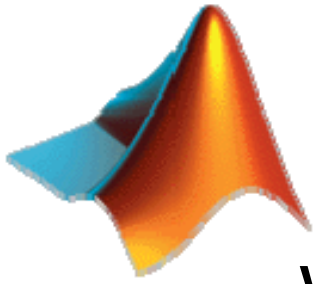
Cambridge

04 December 2013

jason.taylor@mrc-cbu.cam.ac.uk  
jason.taylor@manchester.ac.uk



[from xkcd.com]



# Outline

- Why Matlab?
- A Brief (Interactive) Introduction to Matlab
  - Starting Matlab
  - Path
  - Workspace
  - Numeric Variables
  - Maths/Logic
  - Strings, Cells, Structures
  - Loops
- Scripts and Functions
  - An Example Script (summarise RT data over subjects)
  - An Example Function (standard error of the mean)
- HELP?! (Where to get it)

I will race through  
this as much of it  
has been covered  
by Russell & Yaara!

\* NOTE: Presentation, data, and scripts are available in:  
/imaging/jt03/demo



# Why Matlab?

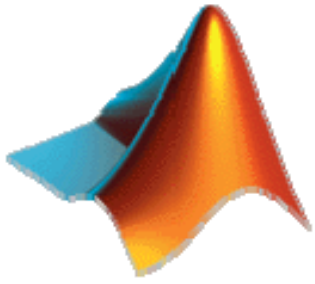
## ...or any other command-line software

**There are distinct advantages to analysing your data using scripts and functions:**

- Leave data in its original format
- Retain a complete record of all processing
- Hard work for the first subject, easy sailing for the rest
- Easily modify analysis pipeline and re-run analyses

### **Alternatives to Matlab:**

- Octave (free!) ... matlab clone
- S-Plus (not free) or R (free!) ... stats
- Yes, you can script Excel (Visual Basic) and SPSS (syntax) too, but these are less flexible/powerful



# Why Matlab?

## ... specifically

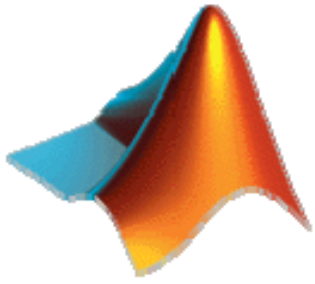
### **Versatility (stock functions + add-on toolboxes):**

- Statistics (basic commands, statistics toolbox, other toolboxes)
- Image Processing
- Signal Processing
- 3D Visualisation
- Custom

see: <http://www.mathworks.com/matlabcentral/fileexchange>

### **Neuroimaging (MRI , fMRI , DTI , M/EEG)**

- SPM
- GIFT / EEGIFT
- EEGLAB / FMRLAB
- FieldTrip
- etc.



# A Brief (Interactive) Introduction to Matlab

Go to demo (`brief_intro_to_matlab.m`)

On the following slides I've simply copied the contents of `brief_intro_to_matlab.m`, one cell per page. It's meant to be explored interactively and run line-by-line, so why not go open it in Matlab Editor and try it out?

(Otherwise, skip ahead to slide 22)

```
% A Brief Introduction to Matlab
%
% This script is intended to introduce novices to
% the Matlab environment. It is meant to be executed
% line-by-line, allowing for interactive exploration
% of data types and whatnot.
%
% Notes:
% - odd spaces between cells for demonstration purposes!
% - navigate between cells using <ctrl>+<down> <ctrl>+<up>
%
% by Jason Taylor (18 Nov 2011 + subsequent small updates)
% MRC Cognition and Brain Sciences Unit
% Cambridge, UK
% email: <first>.<last>@mrc-cbu.cam.ac.uk
% School of Psychological Sciences
% University of Manchester
% email: <first>.<last>@manchester.ac.uk
```

## %% SOME TIPS:

%

### % GENERAL:

% % indicates a comment (ignored)

% ; (@ end of command) means don't print result

%

### % COMMAND WINDOW:

% >> this is the 'prompt': type commands here!

% <up> scrolls through command history, last-to-first

% - all commands if you've typed nothing

% - matching commands if you've typed something

% <tab> completes (e.g., mea<tab> gives 'mean'...)

% <ctrl+c> stop command

% clc clears command window

%

### % EDITOR:

% %% begins a new 'cell' (section of code)

% <F9> executes a highlighted line (or set of lines)

% <ctrl+dn> go to next cell

% <ctrl+up> go to prev cell

## %% STARTING MATLAB

```
% Windows: Start->Matlab or double-click icon
% * Note: can set startup preferences in shortcut
%           e.g., starting dir, -nojvm, etc.

% Linux Machines:
%   Type 'matlab'
%   Type 'matlab <-options>'
%   Type 'spm <options>'           *wrapper script @ CBU
%   ... see demo ...
```



```
%% Matlab & Linux
```

```
% Within matlab, linux commands can be run:
```

```
% ! <command>
```

```
% or:
```

```
% [status,result] = unix(<command>);
```

```
% e.g.:
```

```
! hostname
```

```
[status,hname] = unix('hostname');
```

```
if ~status, fprintf(1,'You are connected to %s\n',hname); end
```

```
%% PATH
%   = search path Matlab uses to identify and execute
%   commands, functions, scripts...

% Report the contents of path:
path

% Add a directory to your path (prepend):
% >> addpath <path/to/directory>

% e.g.,
addpath /imaging/jt03/demo/scripts/
% or, append:
addpath /imaging/jt03/demo/scripts/ -END

% edit /home/<user>/matlab/startup.m

% Find the path of a particular function/script/command:
% which <command>
which mean
```

```
%% THE WORKSPACE
```

```
%   = variables that are currently available to be used  
%       by you (or by functions as input)
```

```
% Two ways to get the mean of a vector:
```

```
mean([4.1 3.3 4.8]) % <-this will give you the answer
```

```
% or,
```

```
x = [4.1 3.3 4.8]    % <- this will store the values  
mx = mean(x)         %       and the answer in variables
```

%% ... and now you can:

% - get other summary statistics,

`sx = std(x)`

`[min(x) max(x)]`

% - plot

`bar(x);`

`hold on;`

`plot(2,mx,'ko','MarkerFaceColor','r','MarkerSize',12);`

% - write it to a text file:

`dlmwrite('x.txt',x,'\t');`

% - save as a .mat file

`save('x.mat','x');`

`clear x`

`load('x.mat')`

% - etc.

`figure; imagesc(rand(64,64)*std(x));`

%% Some WORKSPACE Commands:

% List names of all variables in the workspace:

who

% List names, size, class of all variables in the workspace:

whos

% List ... of a subset of variables in the workspace:

% whos [<variablename>]

% eg.,

whos x

whos \*x\*            % <- '\*' = wildcard

% Clear (all or subset of) variables out of workspace:

clear x

```
%% NUMERIC VARIABLES:
```

```
% Scalar values:
```

```
x = 42
```

```
% Vectors:
```

```
xvec = [1 2 3 4 5 6]
```

```
xvec2 = 1:6 % equivalent
```

```
% Matrices:
```

```
xmat = [1 2 3; 4 5 6; 7 8 9]
```

```
xmat'
```

```
% N-dimensional arrays:
```

```
x3d = cat(3,xmat,xmat+10)
```

```
% Get size of each dimension:
```

```
size(xmat)
```

```
% Indexing:
```

```
xmat(:, [2 3]) % <- all rows, columns 2 and 3
```

```
%% MATHS ( + - * / ^ )
```

```
% Add/subtract
```

```
42 + 10
```

```
x + 10
```

```
y = x - 10
```

```
% Multiply/divide (scalar):
```

```
y = x * 5
```

```
y = x/2
```

```
% Multiply/divide (vector):
```

```
y = xvec .* [10 100 1000 10 100 1000]
```

```
y = xvec/(xvec(1))
```

```
% etc.:
```

```
y = sqrt(x^3)
```

```
y = (x^3)^(1/2) % equivalent
```

```
%% LOGIC & LOGICAL INDEXING ( == ~= > < & && | || )
```

```
% Logic:
```

```
v = xvec*10
```

```
v > 30
```

```
v > 30 & v ~= 60
```

```
% Find index of 'true' (or nonzero, generally):
```

```
find(v>30 & v~=60)
```

```
% Use logical index:
```

```
v(v>30 & v~=60)
```

```
v(find(v>30 & v~=60)) % equivalent
```

```
% Use logical index on a different variable:
```

```
xvec(v>30 & v~=60)
```

```
% Valid numbers:
```

```
v(end) = NaN
```

```
v(~isnan(v))
```



```
%% STRINGS AND CELLS (  ''{} isstr iscell findstr strmatch )

% Strings:
mystring = 'hello world'
xstr = '42'      % not the same as x = 42 (see 'isnumeric')

% String matching:
findstr('o',mystring)
findstr('world',mystring)

% Cell arrays (may mix types, sizes):
mycell = {'hello' 'world'}
xcell = {x xstr}
xcell(2)
xcell{2}

% Cell-string matching:
strmatch('world',mycell)
```

```
%% STRUCTURES ( struct fieldnames isfield )
%    ** SPM users take note (SPM.mat) **

% Struct:
S = struct()
S.subj = 's01'
S.sex  = 'male'
S.age  = 27
S.data = [1 2 3 4 5 6]
isfield(S, 'age')

% Adding layers:
S(2).subj = 's02';
S(2).sex  = 'female';
S(2).age  = 19;
S(2).data = [11 12 13 14 15 16]

% Extracting data:
[S.age]
{S.sex}
```

```
%% LOOPS ( for..end if..end while..end switch..case..end )
```

```
% For loop (with embedded if):
```

```
for i = 1:10
```

```
    if i>3, fprintf(1,'subject %02d\n',i); end
```

```
end
```

```
% While loop:
```

```
i = 0;
```

```
while i<3
```

```
    i = i+1;
```

```
    fprintf(1,'subject %02d\n',i);
```

```
end
```

```
%% Loops continued
```

```
% Switch ... case ... otherwise ... end
```

```
switch S(1).sex
```

```
    case 'male'
```

```
        fprintf(1,'He is subject 1.\n');
```

```
    case 'female'
```

```
        fprintf(1,'She is subject 1.\n');
```

```
otherwise
```

```
    fprintf(1,'Subject 1''s sex was not recorded.\n');
```

```
end
```

```
%% That's enough for now!
```

```
% On to scripts and functions...
```

```
% If you got here via the presentation, type 'return' +  
[ENTER] at the
```

```
% command line, or highlight and <F9>:
```

```
return
```

# Functions vs. (Batch) Scripts

## Function:

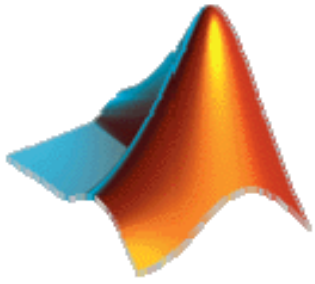
- **General**  
(usually applies to any data, project)
- **Run as command**  
(specify input, output arguments)
- **Variables do not stay in workspace**  
(except input/output arguments, debugging environment)
- **Can get help by typing:**  
`help <function name>`
- **First line of code MUST BE:**  
`function [<out>] = <function_name>(<in>)`  
e.g. `function y = mean(x,dim)`

## Script:

- **'Hack and Run'**  
(customise to your data, project)
- **Copy&Paste (<F9>) or command**  
(no arguments allowed)
- **Variables stay in workspace**

Both are text files, which you can edit in Matlab's editor (see `edit` command) or your favourite text editor (emacs, nedit, gedit, wordpad, notepad, etc.)  
**NOTE:** These vary in terms of debugging friendliness!

**You may start writing a batch script, then later find it useful to convert sections of it into functions.**



# An Example Batch Script

Go to demo ([demo\\_script\\_cell\\_by\\_cell.m](#))

On the following slides I've simply copied the contents of [demo\\_script\\_cell\\_by\\_cell.m](#), which shows the evolution of a simple script to analyse response time data from 15 subjects and produce a figure with a bar plot of mean RT + standard error bars. You can also view and run the resulting script – [demo\\_script\\_simple.m](#) –and the more elaborate version – [demo\\_script\\_final.m](#) –in the CBU imaging workspace.

More information is given in [demo\\_readme.m](#)

```
%  
% This is what I showed in the demonstration. It is meant to show  
% the evolution of demo_script_simple.m:  
%  
% - First, write description of what the script will do  
% - Second, write comments describing each step  
% - Third, flesh out each step with code  
%  
% The 'strings' at the top of each cell are annotations.  
%  
% Use <ctrl>+<down> and <ctrl>+<up> to navigate between cells.  
%  
% by Jason Taylor (21 Nov 2011 + subsequent revisions)  
%
```



%%

'At top: What the script does, when created (updated)?'

% This is a batch script to get the median of each subject's RT data,  
% plot the grand mean and standard error for the two conditions.

%

% by Jason Taylor (17/11/2008)

% + updated (jt 17/11/2008): added error bars

%

```
%%
```

```
'In body: Write an outline using comments'
```

```
% (1) Define directory, filename, subject parameters
```

```
% (2) Get each subject's median RT
```

```
% (3) Compute grand mean, standard error of median RTs
```

```
% (4) Plot bar graph with error bars
```

```
%%
```

```
'Then fill in with increasingly specific comments (as necessary) &  
commands'
```

```
% (1) Define directory, filename, subject parameters:
```

```
% Project directory:
```

```
if ispc
```

```
    projdir = '\\cbsu\data\imaging\jt03\demo\rtdata\subjects';
```

```
else
```

```
    projdir = '/imaging/jt03/demo/rtdata/subjects';
```

```
end
```

```
% Working directory (where summary data will be saved):
```

```
if ispc
```

```
    wkdir = '\\cbsu\data\imaging\jt03\demo\rtdata\ga15';
```

```
else
```

```
    wkdir = '/imaging/jt03/demo/rtdata/ga15';
```

```
end
```

```
% Subjects:
```

```
subjects = [1:15];
```

```
%%
```

```
'...continue to fill in ...'
```

```
% (2) Get each subject's median RT:
```

```
% Initialise variable (subjects x conditions) to collect median RTs:
```

```
mdrt = zeros(length(subjects),2);
```

```
% Loop over subjects:
```

```
for i = 1:length(subjects)
```

```
    % Get current subject label:
```

```
    subj = sprintf('s%02d',subjects(i));
```

```
    % Go to subject's directory, load data:
```

```
    cd(fullfile(projdir,subj));
```

```
    load('word_nonword.mat');
```

```
    % Put median RT for each condition into summary matrix:
```

```
    mdrt(i,1) = median(D.rt(D.event==1));
```

```
    mdrt(i,2) = median(D.rt(D.event==2));
```

```
end % i subjects
```

```
%%
```

```
'...continue to fill in ...'
```

```
% (3) Compute grand mean, standard error:
```

```
% Compute mean (collapsing over rows):
```

```
gm = mean(mdrt,1);
```

```
% Get standard error:
```

```
se = std(mdrt)/sqrt(size(mdrt,1));
```

```
% Save it as a .mat file in working directory:
```

```
cd(wkdir)
```

```
save rtdata.mat gm se
```

```
%%
```

```
'...continue to fill in ...'
```

```
% (4) Plot:
```

```
% Open a figure, make background white:
```

```
fig = figure;
```

```
set(fig, 'color', [1 1 1])
```

```
% Plot means:
```

```
bar(gm);
```

```
% Rescale axes:
```

```
ymax = ceil(max(gm+se));
```

```
set(gca, 'ylim', [0 ymax]);
```

```
% Plot and format error bars:
```

```
ebar1 = line([1 1],[gm(1) gm(1)+se(1)]);
```

```
ebar2 = line([2 2],[gm(2) gm(2)+se(2)]);
```

```
set([ebar1 ebar2], 'linewidth', 6);
```

```
%%
```

```
'...continue to fill in ...'
```

```
% Apply title, labels, etc.:
```

```
title('Grand Mean of Median RTs');
```

```
xlabel('Stimulus Type');
```

```
ylabel('RT + SEM (ms)');
```

```
set(gca, 'xticklab', {'word', 'nonword'});
```

```
% End gracefully:
```

```
fprintf(1, '\n++ done! ++\n\n');
```

%%

'To run the script, type at the Command Line (or highlight and <F9>):'

demo\_script\_simple

'To run a version with nicer formatting, type:'

demo\_script\_final

'If you launched this from the presenation...'

'To return to the presentation, type:'

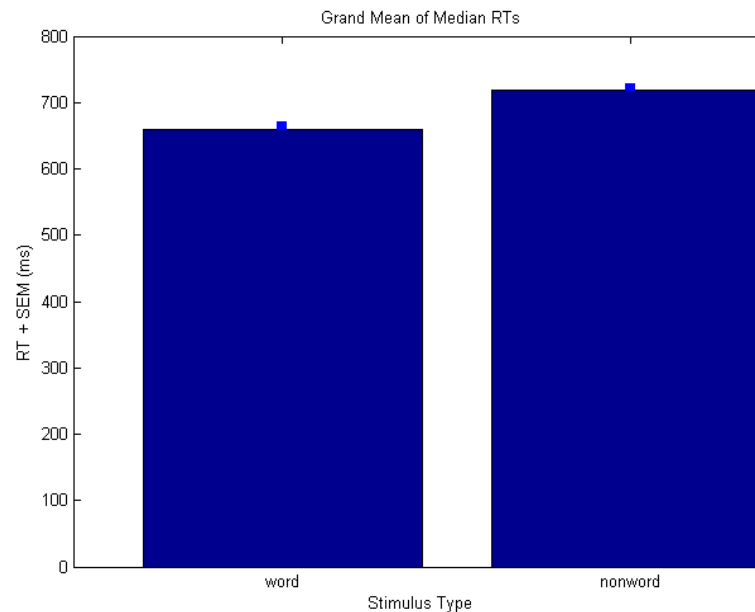
return



# An Example Batch Script

Running the batch script `demo_script_simple.m` should:

- Add several variables to the workspace, including
  - `gm` (grand mean of median RTs for 2 conditions)
  - `se` (standard error of the mean for 2 conditions)
  - `mdrt` (median RTs for each subject and condition, 15x2)
- Open a figure window and plot  $M + SE$  for each condition



# An Example Batch Script

The script `demo_script_final.m` shows how you might improve upon the simple batch script. Some improvements include:

'Adding (at top) some options to make the figure a bit more attractive'

```
%% (0) Define options:
```

```
% Plot format:
```

```
barcolor = [.5 .5 .5];
```

```
ebarcolor = [0 0 0];
```

```
ebarsize = 3;
```

```
plotfont = 12;
```

'These get called later in the call to `bar` (which plots the data):'

```
% Plot means:
```

```
bar(gm, 'facecolor', barcolor);
```

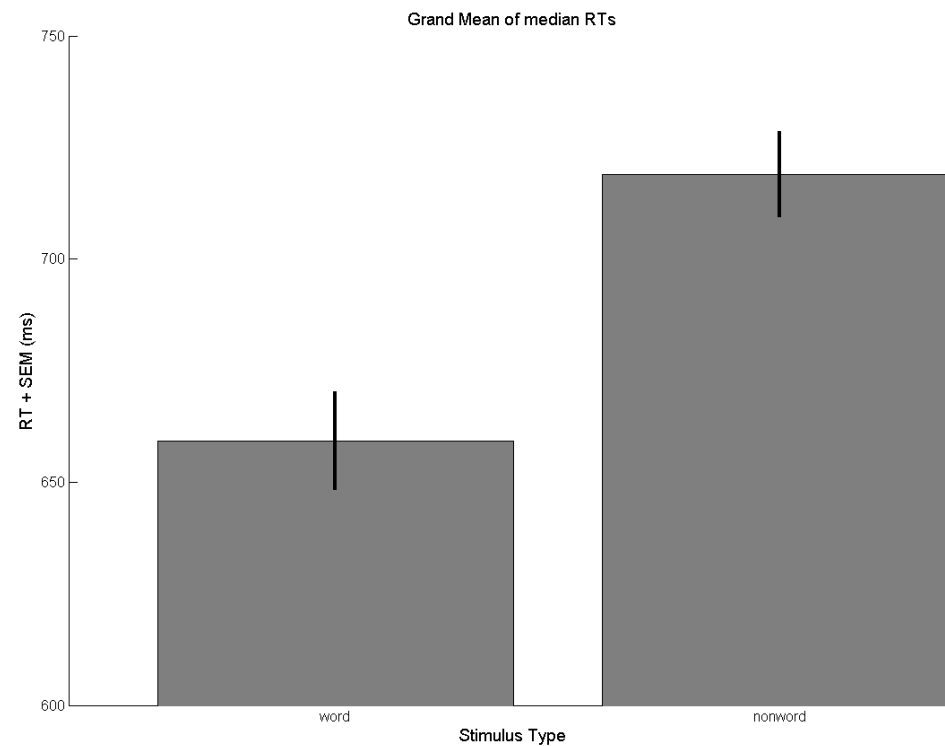
```
set([ebar1 ebar2], 'linewidth', ebarsize, 'color', ebarcolor);
```

```
set(gca, 'fontsize', plotfont);
```

# An Example Batch Script

The script `demo_script_final.m` shows how you might improve upon the simple batch script. Some improvements include:

'Which results in this slightly prettier figure:'



# An Example Batch Script

The script `demo_script_final.m` shows how you might improve upon the simple batch script. Some improvements include:

'Adding some processing options (Which summary statistic? Save? Plot?):'

% Processing options:

```
plotvar = 'median'; % 'median', 'mean', 'trim<N>' (N%-trimmed mean)
dosave  = 0;        % save grandmean data?
doplot  = 1;        % plot grandmean data?
```

'And some data options...'

% Data options:

```
conds      = [1 2];
condlabs    = {'word', 'nonword'};
Nevents     = [240 240];
```

'... which get looped over later'

% Loop over conditions

```
for j = 1:length(conds)
    ' ... '
    rt = D.rt(D.event==conds(j));
    ' ... '
end % j in conds
```

**This loop is more flexible and more powerful than typing out the same command for each condition**

In `demo_script_simple.m`, we had:

```
mdrt(i,1) = median(D.rt(D.event==1));
mdrt(i,2) = median(D.rt(D.event==2));
```

But what if we want to add more conditions?

**\* Better yet, vector/matrix operations are more efficient than loops!**

# An Example Function

At some point, you may find you're often typing out the same formula or set of commands. This is annoying... and inefficient!

For example: In our script, we had to compute standard error by hand:

```
% Get standard error:  
se = std(mdrt)/sqrt(size(mdrt,1));
```

By contrast, we don't compute the mean by hand (sum elements/number of elements), we just call the function `mean`.

So let's create a standard error function.

# An Example Function

First ... what does a function look like?

To look at a function's contents, you can:

```
edit mean      % open the function's m-file in Matlab Editor  
  
type mean      % dump the function's contents to screen  
  
which mean     % find the function's m-file  
  
unix(sprintf('nedit %s',which('mean'))); % edit in another editor
```

The main elements of a function are ... (next slide)

```
function y = mean(x,dim)
```

function call: function [out] = fname(in)  
e.g., function y = mean(x,dim)

```
%MEAN    Average or mean value.
```

```
%    For vectors, MEAN(X) is the mean value of the elements in X. For  
%    matrices, MEAN(X) is a row vector containing the mean value of  
%    each column. For N-D arrays, MEAN(X) is the mean value of the  
%    elements along the first non-singleton dimension of X.
```

```
%
```

```
%    MEAN(X,DIM) takes the mean along the dimension DIM of X.
```

```
%
```

```
%    Example: If X = [0 1 2  
%                   3 4 5]
```

```
%
```

```
%    then mean(X,1) is [1.5 2.5 3.5] and mean(X,2) is [1  
%                                                     4]
```

```
%
```

```
%    Class support for input X:
```

```
%        float: double, single
```

```
%
```

```
%    See also MEDIAN, STD, MIN, MAX, VAR, COV, MODE.
```

```
%    Copyright 1984-2005 The MathWorks, Inc.
```

```
%    $Revision: 5.17.4.3 $    $Date: 2005/05/31 16:30:46 $
```

```
if nargin==1,
```

```
    % Determine which dimension SUM will use
```

```
    dim = min(find(size(x)~=1));
```

```
    if isempty(dim), dim = 1; end
```

```
    y = sum(x)/size(x,dim);
```

```
else
```

```
    y = sum(x,dim)/size(x,dim);
```

```
end
```

## Description

-will display when  
'help' is called.

-useful to include  
examples

## Author info

## Contents of function

# An Example Function

So, have a crack at a standard error function, `sem`:

```
function y = sem(x)

% Computes standard error (standard deviation divided by
% square root of N) of a vector.
%
% by Jason Taylor (18/11/2008)
% note: should be modified to handle matrices
%

% Check that input is a vector:
if nargin~=1
    help sem
    error('No input!')
elseif sum(size(x)>1)>1
    help sem
    error('Input must be a vector!')
end

% Compute SEM:
y = std(x)/sqrt(length(x));

return
```

Give it a unique name; first try: which `sem`

Describe it

Take credit/blame

Note modifications, limitations, bugs

Check for proper input (here must be a vector)

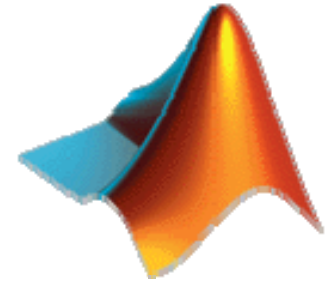
Do it!

Save in your path (e.g., `/home/<user>/matlab/sem.m`)  
(see `sem.m` in `/imaging/jt03/demo/scripts` )



# HELP?!

(where to find it)



Obviously:

```
help <funcname>
```

For pretty, hypertext, browser-based help:

```
doc <funcname>
```

Look at the function!

```
edit <funcname>
```

```
type <funcname>
```

Online: Matlab Central:

```
http://www.mathworks.com/matlabcentral/
```

And the user file exchange:

```
http://www.mathworks.com/matlabcentral/fileexchange/
```

On the imaging wiki:

```
http://imaging.mrc-cbu.cam.ac.uk/imaging/LearningMatlab
```

Email lists (e.g., imagers+, imagerstech)