Matlab tips [x.chen1@ncl.ac.uk](mailto:x.chen1@ncl.ac.uk) Xing

-Write function name, author name, date, and helpful description of your script at the top of your code. If you type in ‘help ‘ followed by the name of the function at the prompt, Matlab will print the description in the first block of text (everything that is commented up to the first empty line).

>> help example\_header\_code

-Keep a list of your important functions somewhere, with descriptions. You could use Microsoft Word’s ‘styles’ to easily differentiate code from the rest of the text, e.g. in this document, lines of code are set to the style ‘code’ while descriptive text is set to ‘description.’

-Comment code using the ‘%’ sign. Keyboard shortcut: Ctrl+R. To uncomment: Ctrl+T.

-Heed conventions to facilitate code sharing & enhance readability (refer to johnson\_Matlab\_style\_guide.pdf). E.g. Capitalise first letter of all words after the first; use all caps only for GLOBAL variables.

-Initialise variables when you first use them, to avoid mistakes later.

initialise\_variables

-Check whether a variable exists, type of variable.

exist(‘var1’)

-As far as possible, minimise ‘hard coding’ and use variables instead.

minimise\_hard\_coding

-Keep indentation neat with Ctrl+I.

-Open highlighted function with Ctrl+D.

-Press F9 to run highlighted code, or to print a highlighted variable to the screen.

-At the prompt, use the ‘Up’ arrow key to cycle through lines of code that were previously run at the prompt, for easy access. Or drag a command from the Command History window to the Command Window.

-Type in first letters of commands in the Command History window to locate commands that match these letters.

-When debugging, press F5 to continue.

-Use meaningful folder and file names. Be descriptive. Anticipate the explosion of collected data, iterations of analyses, and proliferation of folders. Create subfolders where needed. Name folders using variables names and numbers where needed. Incorporate date/time stamps if that will help.

-In Matlab Window, use Edit>Find or Ctrl+Shift+F to find files based on their names or to search through their text for key words.

-Use the split screen tool in the Editor Window to browse, copy and paste easily between scripts.

-Use Tools>Compare Against to compare two pieces of code against each other (good for the initial parts of the code, but gets lost after a lot of differences accumulate).

-Saving data in ‘.mat’ format to a particular folder. How to set the full path to the folder ‘automatically’ rather than manually; how to make directories automatically if they don’t already exist; how to check whether a file of the same name already exists; how to save and/or overwrite a mat file. An example here:

save\_mat\_file.m

-You can use Matlab commands to create Excel spreadsheets, write and read data, and retrieve info about particular spreadsheets.

excel\_read\_write

Other handy tips:

-MS Word: Press Shift+F5 multiple times, to cycle cursor across its last three to four locations in a document. Very handy for editing long documents.

-Doubleclick a word to highlight it. Triple click to highlight a line (in Matlab) or a paragraph (in MS Word).

-Use the Ctrl key to move the cursor word by word, rather than letter by letter. Home and End move the cursor to the start and end of it current line. Ctrl+Home moves the cursor to the top of the text, while Ctrl+End moves it to the bottom.

-Alt+Tab cycles between windows (I think it’s Command+Tab on a Mac). Hold down the Alt key and press the Tab key repeatedly. To cycle through in the opposite direction, hold down ALT and Shift at the same time, while pressing Tab.

-Obtain a clear view of your desktop in Windows with ‘Windows key’+D.

5/6/14

-Use cell arrays for data of variable length

The number of items within your variables may vary, e.g. the number of trials per condition (e.g. condition 1: 5 trials; condition 2: 10 trials), or the length of the names of animals that you record from (e.g. ‘andrew’ with 6 letters, versus ‘bess’ with 4 letters).

There are several ways to deal with this:

1) Create pre-allocated, arbitrarily large arrays and populate them with unrealistic values such as negative numbers, to distinguish them from the actual recorded data (this method is clunky and prone to errors):

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 2 | 3 | 4 | 1 | -1 | -1 | -1 | -1 | -1 |
| 4 | 1 | 3 | 5 | 1 | 4 | 2 | 3 | 5 | 2 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a | n | d | r | e | w |
| b | e | s | s | i | e |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a | n | d | r | e | w |
| b | e | s | s | 1 | 1 |

2) Use cells to accommodate data arrays of varying length and avoid the use of unwanted ‘dummy’ data.

use\_cell\_arrays

-Use the Help function, or type ‘help‘ followed by a space and the name of the function you’re querying.

-Create an account on Matlab Central (Mathworks) to post and answer questions; use the file exchange to download code from reliable sources for analyses that Matlab doesn’t cover adequately.

-Instruct Matlab to stop at point in code where error occurred, allowing you to debug it. Otherwise, code just exits and you don’t pause.

dbstop if error

-For long (e.g. overnight) runs, use error throwing and catching to process data wherever possible and skip errors that arise unexpectedly with parts of dataset. Scenario: you’ve tested your code on part of your data, and it works fine. You decide to leave it running overnight to process the rest of your data. The next morning, however, you find that the run has halted prematurely because one of your new data files contains missing values, or some such aberration. Error catching and throwing allows you to skip the portion of data on which the code does not run, and continue attempting to process the rest of the data. You can instruct Matlab to keep a record of any errors, and review them and carry out debugging after the initial run.

handling\_errors

-Use a version control system like Git or Subversion, to keep code organised; facilitate retrieval; find code that was created at a similar period; recall code’s purpose; and, potentially, share code with collaborators.

How code evolves.  
Accessing code from multiple computers/locations (e.g. various lab and office computers; your laptop).  
Sharing code with colleagues and collaborators (e.g. developing multiple versions).  
What is version control? <http://git-scm.com/book/en/Getting-Started-About-Version-Control>   
Types of version control systems available.  
Useful features of VCS: repositories; synchronization; checking code in and out; descriptions of alterations to code; reverting to previous states or rolling back changes; comparing scripts; branching; tracking and merging changes.  
Overview of two of the most popular and free VCS (Git and Subversion).  
Simple to install, GUIs available for various OS platforms.

Git: “Installing Git on Windows is very easy. The msysGit project has one of the easier installation procedures. Simply download the installer exe file from the GitHub page, and run it:

<http://msysgit.github.com/>

After it’s installed, you have both a command-line version (including an SSH client that will come in handy later) and the standard GUI.”

Demonstration of features.

12/6/14

Use Autohotkey to automate repetitive tasks (file management, routine analyses)

How to implement commonly used statistics in Matlab: *t*-tests, correlations, ANOVAs, signed rank/ rank sum, chi-squared test. More advanced: circular stats, regression, goodness of fit, skewness, Levene’s test, Akaike information criterion, partial correlation.

19/6/14

How to export figures, choose image formats, use transparency, create publication-quality figures, use a standard template of image settings.

Subject to demand:

Use parallel toolbox for large datasets and high throughput processing