5. MATLAB I/O 1

Beyond the Mouse

GEOS 436/636

Jeff Freymueller, Sep 26, 2017

"The Uncomfortable Truths Well", http://xkcd.com/568 (April 13, 2009)

Topics

- Loading and Saving the Workspace
- File Access
- *Plotting Data*
- *Annotating Plots*
- *Saving a Plot*
- You have seen some of these in action already, but we will go into more detail this time.

Load and Save

- You can save some or all of your workspace to a file, and load it back later
  - These files can be ascii text or binary
- `save filename`
  - Saves all workspace variables to file `filename.mat`
- `save('filename', 'var1', 'var2');`
  - Saves only the variables `var1` and `var2` to file `filename.mat`
- `load filename`
  - Loads all the variables saved in file `filename.mat`

Reading Data from Files

- First business: Opening and closing files
- How to read and write data from:
  - MS Excel files
  - Text files
    - textread (deprecated)
    - textscan
    - fprintf

Opening and Closing Files

- For most functions that read data from a file or write data to a file, you have to open the file first, and close it when you are done.
- MATLAB can have several files open at once, and uses a file ID so that you access the correct file.
- You get a file ID when you call the function `fopen()`, and you release it when you call `fclose()`.

Opening a File: fopen()

- fid = fopen(filename, mode)
  - Filename is a string or string variable with the name of the file to be opened
  - Mode is a string or string variable telling MATLAB whether to read, write, or append:
    - 'r': read
    - 'w': write **Be careful with 'w', as it will overwrite an existing file!**
    - 'a': append
- Don't throw away the return value, as you will need it to access the file!

Closing a File: fclose()

- `fclose(fid)`
  - `fid` is the file ID of the file you want to close.
- Call `fclose` when you are done with a file so that you can be sure all changes are written to the file (if you opened for writing).
  - When you write to a file, exactly when that is saved to disk depends on the operating system, not MATLAB
- It is good practice to close files when you are done, like brushing your teeth after eating.
  - Also, there is some limit to how many files MATLAB can keep open at one time (it is large, but…)

Reading/Writing MS Excel Files

- There are many data files hanging around in MS Excel format. You might want to read one, or export some data into that format
- Read with ***xlsread***, write with ***xlswrite***
  - On Windows systems with Excel installed, **xlsread** reads any file format recognized by your version of Excel, including XLS, XLSX, XLSB, XLSM, and HTML-based formats.
  - If your system does not have Excel for Windows, **xlsread** operates in 'basic' mode
  - If your system does not have Excel for Windows, or if the COM server (part of the typical installation of Excel) is unavailable, **xlswrite** operates in a limited mode

xlsread

- `[num, txt, raw] = xlsread('myfile.xls', 'sheet23', 'A3:B7');`
  - First argument is the filename
  - Second argument (if given) is the sheet name (default is the first sheet if this argument is omitted)
  - Third argument (if given) is a range specifier for elements of the sheet.
  - Output values
    - num – a matrix that contains all numeric data
    - txt – a cell array that contains all text data
    - raw – cell array with columns xlsread could not interpret

xlsread: Basic Mode

- Basic Mode for **xlsread**
  - Basic mode behavior changes with MATLAB version
  - Only reads XLS files compatible with Excel 97-2003 software (R2012a and later also read XLSX and others).
  - Imports the entire active range of the worksheet.
  - Requires a string to specify the SHEET, and the name is case sensitive.
  - Excel and MATLAB can store dates as strings (such as '10/31/96') or serial date numbers. Serial date numbers in Excel use a different reference date than date numbers in MATLAB. In 'basic' mode, **xlsread** imports all dates as Excel date numbers.
    - Translation: you may have date problems in basic mode
- Just be aware that if you want to use any other feature of **xlsread**, you will be tied to Windows

xlswrite

- `[status, msg] = xlswrite('myfile.xls', M, 'sheet42');`
  - attempts to write matrix M to sheet42 of myfile.xls
  - You can also specify the range of elements to use
  - Outputs:
    - status – 1 on success, 0 on error
    - msg – error message object with fields message and identifier

csvread, csvwrite

- You can read and write csv (comma separated values) files on any machine
- `M = csvread('filename', R, C)`
  - reads data from the comma separated value formatted file starting at row R and column C.
  - R and C are optional
    - R and C are counted from zero so that R=0 and C=0 specifies the first value in the file.
- `csvwrite('filename', M, R, C)`
  - writes matrix M starting at offset row R, and column C in the file
  - R and C are optional
- `dlmread` and `dlmwrite` can use any character as delimiter instead of a comma (e.g., `':'`).

Textread

- Textread is a *deprecated* function, which means that it will eventually disappear.
  - In new code, use textscan instead.
- `[A, B, C, ...] = textread('filename', 'format', N);`
  - reads data from file `'filename'` to multiple outputs A,B,C,... using specified format until entire file is read, or N times. Each column of file in one array
- Textscan is similar but requires a file ID instead of a filename
  - Meaning you have to open the file yourself

Textscan

- Textscan is really great! It is designed to read files where the values are in columns, defined by white space (spaces or tabs)
  - If you want to read strings that contain spaces, you have to use another function
- `C = textscan(fid, 'format', N);`
  - reads data from file fid to cell array C using specified format until entire file is read, or N times
    - Each column of data is stored in one cell of C
    - resume from where you left off by calling textscan again later).

Textscan

- We have used it before, to read files that had a few columns of numbers.
- Read four floating point numbers:
  - `C = textscan(fid, '%f%f%f%f');`
- Read a string, and integer and then floating point numbers:
  - `C = textscan(fid, '%s%d%f%f%f%f');`
- Read from a string
  - `C = textscan('2 4 6 8', '%d%d%d%d');`
- More complex format statements can be used

More general: fscanf

- You can read any general format using **fscanf**, but it does not handle strings as nicely as textscan.
- `[A,COUNT] = fscanf(FID,FORMAT,SIZE)`
  - reads data from the file specified by file identifier FID, converts it according to the specified FORMAT string, and returns it in matrix A.
  - Strings come back as character arrays (numeric values in the matrix A)
- You can used ***sscanf*** to do the same from a string.

What are These Format Codes?

- The format strings use the style of the C programming language
- Read/write a string: **%s**
  - A 7 character string: **%7s**
- Read/write a floating point number: **%f**
  - Use 9 digits, 4 after the decimal: **%9.4f**
- Read/write an integer: **%d**
- There are other options as well: `help fscanf`

Lower-level File Access

- `TLINE = fgets(FID)`
  - Returns one line of file as a text string, *including* the end of line character(s)
- `TLINE = fgetl(FID)`
  - Returns one line of file as a text string, *without* the end of line character(s)
- You can then use `sscanf()` to read any part of the line that you might want
  - Syntax for `sscanf` follows that for `fscanf`.

Read in an Image with Importdata

- ***Importdata*** allows you to read the data from a text file into a structure:
- `filename = 'myfile01.txt';`
- `delimiter = ' ';`
- `headerlines = 1;`
- `A = importdata(filename,delimiter,headerlines);`
- The input file might look like this:

```
Day1  Day2  Day3  Day4  Day5  Day6  Day7
95.01 76.21 61.54 40.57  5.79 20.28  1.53
23.11 45.65 79.19 93.55 35.29 19.87 74.68
60.68  1.85 92.18 91.69 81.32 60.38 44.51
48.60 82.14 73.82 41.03  0.99 27.22 93.18
89.13 44.47 17.63 89.36 13.89 19.88 46.60
```

Read in an Image with Importdata

- ***Importdata*** allows you to read the color values from a JPEG file into an array so that you can display it:
- `A = importdata('ngc6543a.jpg');`
- `image(A)`

Output: fprintf

- fprintf is the workhorse of writing out formatted print output
- `count = fprintf(fid, 'format', A, ...);`
  - formats data in matrix A (and any additional arguments) according to format string and writes to the file associated with fid
  - count – number of bytes written

An fprintf example

fprintf example

```
clear all, clc, close all;

% create data here (row vectors!)
x = 1:10
y = rand(1,10)
z = rand(1,10)

% open a file in write mode
fout = fopen('random_numbers.txt', 'w');

% write our data:
%   x is first column,
%   y is second column
fprintf(fout, '%d\t%f\t%f\n', [x; y; z])

% don't forget to close the file!
fclose(fout)
```

output

```
1   0.438744  0.276025
2   0.381558  0.679703
3   0.765517  0.655098
4   0.795200  0.162612
5   0.186873  0.118998
6   0.489764  0.498364
7   0.445586  0.959744
8   0.646313  0.340386
9   0.709365  0.585268
10  0.754687  0.223812
```

"Feature" alert: Each ***row*** of the matrix is written as a ***column*** in the output file.
Or: Each ***column*** of the matrix is written as a ***row*** in the output file

Plotting Data

- Will be covered in detail in the next lecture and lab.

Plotting Data

- Will be covered in detail in the next lecture and lab.