L18: File IO

And Lab 07 and Midterm Feedback

Lucas A. J. Bastien

E7 Spring 2017, University of California at Berkeley

March 03, 2017

Version: release

Announcements

Lab 07 is due on March 10 at 12 pm (noon)

Today:

- File input and output (I/O)
- Introduction to Lab 07
- Midterm feedback

Next week:

- Monday: Root finding *i.e.* solving $f(x) = 0$ for x (Chapter 16)
- Wednesday: Systems of linear algebraic equations (Chapter 12)
- Friday: discussion

Need to reduce chatting and noise during lecture!

- Noise is distracting for others

Motivation for reading data from and writing data to disc

Motivations

- Save data for later use
  (When Matlab closes, variables in the workspace are lost)
- Communicate data to other people
  (*e.g.*, by emailing a file)
- Communicate data between programs
  (*e.g.*, Plot results of a non-Matlab computer model using Matlab)

Examples of file formats to store data:

- Text files (generic, taught in E7)
- Binary files (generic, not taught in E7)
- `mat` files (Matlab-specific, taught in E7)

What are text files?

A text file is a file where data are described by strings of characters and organized by lines

- Data are still stored as zeros and ones; but
- Text editors transform these 0s and 1s into human-readable form
  - The conversion depends on character encoding (ASCII, unicode, ...)

Examples:

- m-files are text files

Advantages:

- Portable: many, many programs can read and write text files
- Human-readable when opened in a text editor

Downsides:

- Not the most efficient (disc space) way to store data

Text files are still stored as 0s and 1s

How are text files stored as 0s and 1s? Each character (including digits) is individually converted to a binary representation

Example with 104 and ASCII:

Character	ASCII code	7-bit unsigned representation of the ASCII code
0	48	0110000
1	49	0110001
4	52	0110100

Representation of 104 in an ASCII text file:

$$01100010110000110100$$

In contrast: unsigned 8-bit representation of $104 = 2^6 + 2^5 + 2^3$

$$01101000$$

Reading and writing text files in Matlab

Steps: (the process is similar in many programming languages)

- Open the file (obtain a file identifier)
- Read and/or write data to the file
- Close the file

Open a file

```
>> file_id = fopen(filename, mode)
```

- `filename`: name of the file (character string)
- `file_id`: "nickname" for the file while working with it in Matlab (integer of class `double`, -1 if the file could not be opened)
- `mode`: read/write permissions
  - `'r'`: read-only (can't write to the file)
  - `'w'`: write mode (overwrites existing file if any)
  - `'a'`: append mode (adds data to existing files or creates new file)
  - Type `help fopen` for full list

Reading text from text files

Read text from text files:

```
>> line_of_text = fgetl(file_id)
```

- `file_id`: file ID of a file that is already opened
- `fgetl` reads one line of text
- The first call to `fgetl` reads the first line of text in the file
- The second call to `fgetl` reads the second line of text in the file
- And so on...
- When a line of text is read successfully, `fgetl` returns the line of text read from the file as a variable of class `char`
- When the end of the file is reached, `fgetl` returns -1 (class `double`)

See function `my_multiplication_table` for an example

Writing text to text files and closing files

Write text to text files:

```
>> fprintf(file_id, 'Some text to write')
>> fprintf(file_id, ' for E%d', 7)
```

- `file_id`: file ID of a file that is already opened
- `fprintf` does not automatically add newline characters (`'\n'`), so the above two commands would yield a single line of text in the corresponding text file:

  ```
  Some text to write for E7
  ```

Close the file when you are done with working on it:

```
>> fclose(file_id)
>> % Or, for more information:
>> status = fclose(file_id)
>> % Status will be 0 (class double) if file closed successfully
>> % Status will be -1 (class double) otherwise
```

Special text file format

Comma-separated values (csv). Example:

```
5, 6, 7, 8
5, 402, 7, 10000
5, hello, world, "Hello, World"
0, 5, 1, 4
```

- If a fied contains a comma, use double quotes
- csv is not a standard → many different styles of csv files
- Can use a character other than a comma as the delimiter
- Matlab has a built-in function `csvread`, but it only works when all fields are numerical (so it would not work on the example above)

Other examples of special formats: (not taught in E7)

- JSON (JavaScript Object Notation), standardized
- XML (Extensible Markup Language), standardized

JSON and XML are used extensively for communication between web browsers and the servers hosting the websites accessed by the browsers

Useful built-in functions for file I/O: str2num

```
>> x = str2num('50')
x =
    50

>> class(x)
ans =
double

>> vector = str2num('5 10 7')
vector =
     5    10     7

>> array = str2num('5, 10, 7; 3 0 -5')
array =
     5    10     7
     3     0    -5

>> array = str2num(sprintf('5, 10, 7\n3 0 -5'))
array =
     5    10     7
     3     0    -5
```

Useful built-in functions for file I/O: strsplit

```
>> % The default delimiter is white space (space, tab, newline)
>> c = strsplit('How are you?')
c =
  1x3 cell array
    'How'    'are'    'you?'

>> d = strsplit('7 4 4 1 01')
d =
  1x5 cell array
    '7'    '4'    '4'    '1'    '01'

>> class(d{1})
ans =
char

>> % We can manually specify which delimiter to use
>> d = strsplit('Sp-lit a-round dash-es', '-')
d =
  1x4 cell array
    'Sp'    'lit a'    'round dash'    'es'
```

Useful built-in functions for file I/O: strrep

```
>> strrep('test', 'e', 'oa')
ans =
toast


>> strrep('the test', 'e', 'E')
ans =
thE tEst


>> strrep('test', 'e', {'oa', 'he bread cru'})
ans =
  1x2 cell array
    'toast'    'the bread crust'


>> strrep('test', {'e', 'te'}, {'oa', 'co'})
ans =
  1x2 cell array
    'toast'    'cost'
```

Matlab's mat files

In Matlab, one can save variables currently defined in the workspace in a `mat` file. This file format is specific to Matlab, so:

- not very convenient to communicate data to other programs, **but**
- very convenient to save Matlab variables for future use, and for communicating data between Matlab users

Functions:

- `save`: save variables from the workspace to a `mat` file
- `load`: load variables from a `mat` file to the workspace

Matlab's mat files: example

```
>> s = 'this is a character string';
>> x = 10.1;
>> my_function_handle = @cos;
>> my_cell_array = {10, {2, 'hello'}, 4};

>> % The following command will create a file named
>> % 'my_mat_file.mat' which contains all the variables
>> % that are currently defined in the workspace.
>> %
>> % Example of saving only certain variables:
>> % save('my_mat_file.mat', 's', 'my_function_handle')
>> save('my_mat_file.mat')

>> clear all
>> load('my_mat_file.mat');
>> my_cell_array{2}
ans =
  1x2 cell array
    [2]    'hello'
```

Introduction to lab 07

Q1 is an application of today's material. Q2 and Q3 are "review" questions i.e. they rely only on concepts from past lectures

Estimated complexity:

Q1	⋆	`my_read_array.m`	
Q2.1	⋆	`my_destination.m`	
Q2.2	⋆⋆	`my_fill_open_space.m`	
Q2.3	⋆	`my_gridcells_connected.m`	(provided that you did Q2.2)
Q2.4	⋆⋆	`my_open_spaces.m`	(provided that you did Q2.2)
Q3	⋆⋆⋆	`my_buses.m`	

Use this "review" lab to:

- Practice creating your own test cases to test your functions
- Keep the "keep it simple" approach in mind

Introduction to lab 07: create your own test cases

For example, in Q2.3, you have to write a function that determines whether two open spaces in a maze are connected

We provide you with the following support functions:

- `my_create_maze` creates a randomly-generated $m \times n$ maze
- `my_show_maze` creates a figure that shows an $m \times n$ maze

You can create as many test cases as you wish!
(see next slide for example)

Introduction to lab 07: create your own test cases

1. Create a randomly-generated maze:

```
>> maze = my_create_maze(15,25);
```

2. Look at what this maze looks like (see next slide):

```
>> maze = my_show_maze(maze);
```

3. Pick two open spaces that are connected *e.g.*, $(13, 8)$ and $(9, 4)$
   Test your function (it should return `true`)

```
>> my_gridcells_connected(maze, 13, 8, 9, 4)
ans =
  logical
   1
```

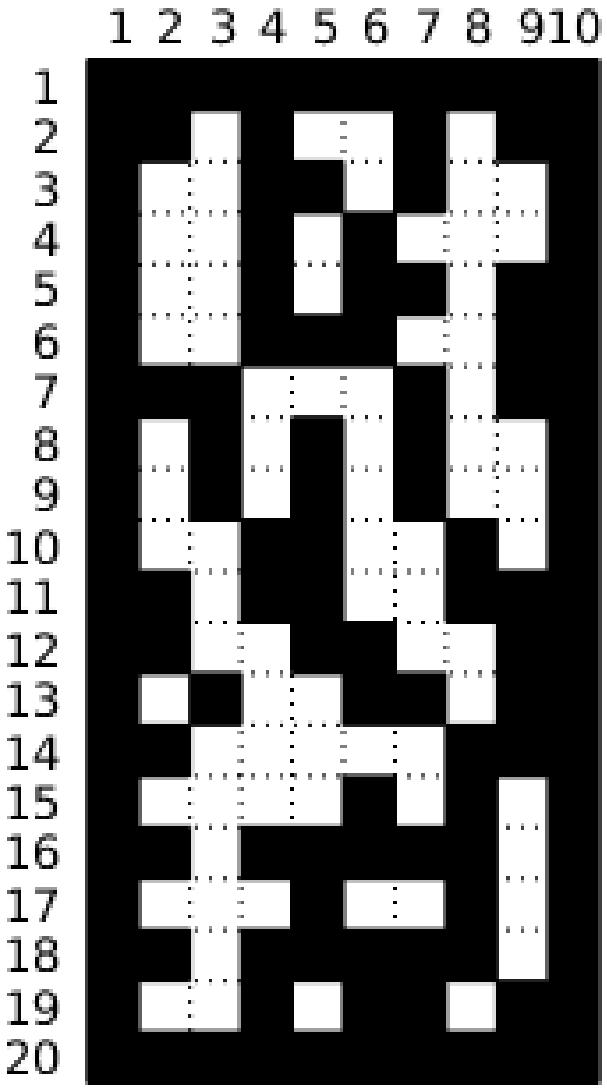
4. Pick two open spaces that are **not** connected *e.g.*, $(6, 2)$ and $(18, 9)$
   Test your function (it should return `false`)

```
>> my_gridcells_connected(maze, 6, 2, 18, 9)
ans =
  logical
   0
```

5. And so on...

Introduction to lab 07: create your own test cases

Maze created in the previous slide:



Introduction to lab 07: keeping it simple

Keep it simple: don't add complications that do not exist

Simple $\neq$ easy – This distinction is very important!

- Keeping it simple can be difficult!
- It involves **solving the problem on paper first!**
  - Understand what all the input parameters are
  - Understand what the function should do
  - Work through (parts of) the problem on paper
  - Before doing any coding, select:
    - Data structures that you will use
    - Algorithms that you will use

    Write the algorithms in words (*e.g.*, as a series of bullet points) and draw sketches of the data structures
- If you start coding without planning ahead, your code may be very...
  - difficult to write
  - difficult to read
  - difficult to debug

Introduction to lab 07: keeping it simple

Example with Q3: bus lines. **At first glance, this problem may seem very complex. It will most likely be very much so if you start coding it without doing any preparation on paper first**

Spend some time working on paper making sure that you understand:

- What the question is asking
- What the input parameters to the function are (one of them, `passengers`, is quite complex)

We provide you with a template of a solution, where part of the code is already written. We describe in words (comments) what tasks remain to be coded in this template to make the function work

- We give you the structure of a solution, to help you keep it simple
- You complete the remaining tasks within this structured code
- You can start a new function from scratch (at your own risk)