File Processing

Basics of File Processing

- Three different types of data files:
  - MAT files for MATLAB variables
  - Text files
  - Binary files

MAT Files

- MATLAB's own file format for MATLAB variables (actually a hdf5-compatible format since v.6)
- Writing: `save(file_name, 'variable_name', …);`
  - Can save any number of variables; variable headers are saved as well as the data.
- Reading: `load(file_name, 'variable_name', …);`
  - Loaded variables are put in the scope of where it is called (current work space if called from command window).
  - Variable names are optional; if not specified, all the variables in the file are loaded.
  - `A = load(…)`: the loaded variables become fields of the structure variable `A` (to be discussed later).

Text Files

- Regular text files that can be used by other applications.
- Open a file: `fid = fopen(file_name, mode)`
  - mode: `'rt'` for reading, `'wt'` for writing; see documentation for other modes.
  - Usage very similar to C.
- Close a file: `fclose(fid)`
- Writing: `fprintf(fid, format_spec, …);`
  - When `fid` is omitted, it writes to the standard output (the command window).
  - `format_spec` is reused as needed.

Text Files

- Reading:
  - `s = fgetl(fid)`: Read a line.
  - `A = fscanf(fid, format_spec)`: Read multiple values into a double array `A`; reuse `format_spec` if necessary.
  - `C = textscan(fid, format_spec)`: `C` is an array of cell arrays; each element (a cell array) in `C` represents the values corresponding to one place holder in `format_spec`.
    - Suitable for reading in table-like text files.

Binary Files

- Open a file: `fid = fopen(file_name, mode)`
  - mode: `'rb'` for reading, `'wb'` for writing; see documentation for other modes.
  - Usage very similar to C.
- Close a file: `fclose(fid)`
- Writing: `fwrite`
- Reading: `fread`
  - Data formats (classes) can be specified (and converted) during reading/writing. See documentation.

String Processing

Representation of Strings

- A string as a vector of characters (class = **`char`**):
  - Each character is an array element.
  - Used when a string is specified with single quotation marks, such as **`s = 'hello'`**.
  - For 2-D character arrays, each row represents a string. Shorter strings are padded with spaces.
  - Pre-allocation: Function **`blanks`**.
- A more common method to represent multiple strings is to use a **cell array of character vectors**:
  - Example: **`{'apple','orange','lemon'}`**

Representation of Strings

- A string as an object of class **`string`**:
  - Newer in MATLAB (since 2016)
  - A string (not individual characters) is a single element.
  - Used when a string is specified with double quotation marks, such as **`s = "hello"`**.
- Multiple strings are stored in arrays just like other data types.
  - Example: **`["apple","orange","lemon"]`**

Representation of Strings

- The representation using class **`string`** is provided to make operations on multiple strings more efficient than using cell arrays of character vectors.
- Conversion between the two representations: Casting with **`char`** and **`string`**.
- MATLAB has provided good compatibility for the two representations. Most string related functions can take character vectors or string objects (as well as cell array of character vectors or string arrays) as inputs.

Strings to/from Other Data Types

- Strings from simple numerical data: Function **`num2str`**; optional format string can be provided.
- Generating formatted strings: Function **`sprintf`** (C-style formatting with repeated use of the format specifier as in **`fprintf`**).
- Strings to numerical data (default format specifier):
  - **`str2double`** and **`str2num`**
- Data from formatted strings: Function **`sscanf`** (C-style formatting with repeated use of the format specifier as in **`fscanf`**)
  - Output is a single array; the data type depends on the format specifier.
- Simple type casting (such as using **`char`** or **`double`**) leads to characters being treated as codes (e.g., **ASCII**).

Useful String Functions

- Element-wise property: **`isletter`** and **`isspace`**
- Find and replace: **`strfind`** and **`strrep`**
- Splitting by delimiters:
  - **`strsplit`** (output is a cell array of character vectors)
  - **`strtoken`** (output only one substring)
- Text formatting:
  - Removing white spaces: **`deblank`** and **`strtrim`**
  - Change case: **`lower`** and **`upper`**
  - Justification: **`strjust`**

Useful String Functions

- Comparing strings:
  - **`strcmp`**, **`strcmpi`**, **`strncmp`**, **`strncmpi`**
    - Output is logical
    - Can compare a string with a cell array of character vectors (or a string array)
  - **`validatestring`** (mainly used for checking function input arguments that are strings)