Contents

READING DATA FROM TEXT FILES

We have already learned the `input()` command, which allows us to prompt the user for data entry and store any entered value(s) in a variable. The user can easily enter a single number or character string, but cannot easily enter many values simultaneously. Furthermore, the `input()` command does not allow you to read in data from an external file – it only takes input from the *command window*. There are many commands in MATLAB that can be used to read in data from external files, and the best choice will depend on the type and format of the file. For example, the command `csvread()` is useful if you need to read numeric data from a comma-separated value (CSV) file, and the `xlsread()` command can easily import data from Microsoft Excel spreadsheets. While these commands are useful for specific applications, in MAE10 we will focus on the `fscanf()` command.

The `fscanf()` Command

The `fscanf()` command is more generally applicable and can be used to read in many different types of data from any ASCII text file. The general form of `fscanf()` is as follows:

```
array = fscanf(fileID,‘format specifiers’,[cols,rows])
```

- Where `array` is any valid variable name. This variable will store all data that is read in from the file specified by `fileID`.
- Where `fileID` is the file identifier variable that is created when a file is opened using `fopen()`.
    - Remember that you must open a file with read permission using `fopen()` (and create a `fileID`) before you can use `fscanf()` to read in data from the file.
- Where the `format specifiers` are the same ones you have already learned for `fprintf()`. Note that you must provide format specifiers to tell MATLAB the format of the data in the file. These format specifiers do **not** determine how MATLAB will display the data in the *command window* when it is read in. The format specifiers only describe the data that already exists in the file so that MATLAB reads it in appropriately. When `fscanf()` reads a file, it attempts to match the data in the file to the formats that you specify.

    - If all of the data in the file is in the same format (e.g., all numbers in decimal notation), you typically only need to provide one format specifier – just a `%f` would do in this case. Note that you typically do not need to provide a field width or precision either; again we just need to tell MATLAB what type of data it needs to look for in the file.
- Where `[cols,rows]` specifies the size of the data in the text file. Note that this notation is the reverse of how we typically index arrays in MATLAB. That is, you must specify first how many columns of data are in the text file, followed by how many rows of data are in the text file.
    - You should always specify the number of columns exactly (by looking in the file) to ensure that the data is read in correctly. If the number or columns you give in the `fscanf()` command does not match the number of columns in the data file, the data will be read in incorrectly and you may miss some data.
    - When specifying the number of rows, you have several options.
        - You can specify less than the number of rows of data that exist in the file. This means that MATLAB will not read in all the data in the file – you can read in as many or as few lines (rows) as you like. This is particularly useful if you have a large data file and would like to test your code by reading in only the first few rows.
        - You can specify exactly the number of rows in the data file. This typically means opening the file and counting the number of rows of data. However, if you know you want to read in all rows of data, the next option may be easier.
        - You can specify `inf` – `inf` is a keyword in `fscanf()` that tells MATLAB to read to the end of the file, regardless of how many rows of data are present.
            - Note: Do not use `inf` outside of `fscanf()` – it cannot be used to index arrays (and is different than `end`).
            - You cannot use `inf` for the number of columns of data, only for the number of rows.
    - The size argument is optional although it is almost always included to ensure that the data in the file is read in appropriately.

Now that we know what each of the arguments in the `fscanf()` command does, let's look at an example where we read in some numeric data from a file called `my_data.txt` (which contains a mix of integers and decimals). We need to make sure that `my_data.txt` is in the same directory as `myfile.m` so that MATLAB can find it (the M-File and data file must be in the same folder on your computer). In this first example, we are going to read in only the first row of data from the file. This is achieved by placing a `1` in the `rows` spot of the 3rd argument in the `fscanf()` command.

In `my_data.txt`:

```
    1     1    1.9  1.67     1   1.4
    1     2    1.3  2.71     2   1.8
    1     2    2.4  1.36     2   1.7
    2     3    2.0  2.16     3   2.1
    5     5    4.7  2.72     3   1.5
    6     6    5.9  2.56     4   1.1
    3     3    2.1  2.25     3   1.7
    5     4    6.6  6.60     6   1.3
    1     3    5.8  6.89     4   8.4
    8    10    6.3  4.25     2   7.1
```

In `myfile.m`:

```
myfile = fopen('my_data.txt','r');
mydata = fscanf(myfile,'%f',[6,1])
fclose(myfile);
```

Sample Output:

```
mydata =

    1.0000
    1.0000
    1.9000
    1.6700
    1.0000
    1.4000
```

Right away you notice that we read in one row of data from the file, but it was saved as one column in the array `mydata` in MATLAB. This is because **`fscanf()` fills the array in column order**. Similar to how `fprintf()` prints in column order, `fscanf()` reads in data (across the rows in the file) and fills the array in column order. Therefore, what appears as a row in the data file will be saved as a column in the array inside MATLAB.

Let's expand the amount of data we read to be 4 rows and see how the `mydata` array looks in MATLAB.

In `myfile.m`:

```
myfile = fopen('my_data.txt','r');
mydata = fscanf(myfile,'%f',[6,4])
fclose(myfile);
```

Sample Output:

```
mydata =

    1.0000    1.0000    1.0000    2.0000
    1.0000    2.0000    2.0000    3.0000
    1.9000    1.3000    2.4000    2.0000
    1.6700    2.7100    1.3600    2.1600
    1.0000    2.0000    2.0000    3.0000
    1.4000    1.8000    1.7000    2.1000
```

Again we can see that `fscanf()` read in a row of data and saved it as a column in the `mydata` array. The first row of data from the file becomes the first column in the `mydata` array, and so on. If you look carefully at the size we provided in `fscanf()`, you should see that this is in fact the size of the `mydata` array in the traditional row and column notation. However, do not forget the rules for using `fscanf()` – the size you specify in the `fscanf()` command should correspond to the size of the data in the file in `[col,row]` notation (which just happens to be the size of the array in MATLAB in row and column notation since `fscanf()` fills in array in column order). If you recall the `transpose()` command, you should remember that it operates by switching the rows of columns of an array. Therefore, if we want to manipulate our array in MATLAB to be back in the original shape of the data in the file, we simply need to transpose the array after using `fscanf()`.

Again working with the same data file, we will now use `fscanf()` to read in all the data in the file with the `inf` keyword, and we will transpose the array so that the data in the array in MATLAB is in the same size/shape as it was in the file.

In `myfile.m`:

```
myfile = fopen('my_data.txt','r');
mydata = fscanf(myfile,'%f',[6,inf]);
mydata = transpose(mydata)
fclose(myfile);
```

Sample Output:

```
mydata =

    1.0000    1.0000    1.9000    1.6700    1.0000    1.4000
    1.0000    2.0000    1.3000    2.7100    2.0000    1.8000
    1.0000    2.0000    2.4000    1.3600    2.0000    1.7000
    2.0000    3.0000    2.0000    2.1600    3.0000    2.1000
    5.0000    5.0000    4.7000    2.7200    3.0000    1.5000
    6.0000    6.0000    5.9000    2.5600    4.0000    1.1000
    3.0000    3.0000    2.1000    2.2500    3.0000    1.7000
    5.0000    4.0000    6.6000    6.6000    6.0000    1.3000
    1.0000    3.0000    5.8000    6.8900    4.0000    8.4000
    8.0000   10.0000    6.3000    4.2500    2.0000    7.1000
```

Success! We read in all the data from `my_data.txt` and used the `transpose()` command to 'flip' the `mydata` array back into the same dimensions as the data in the file. While it is never required that you transpose your arrays after using `fscanf()`, it often helpful when you need to index the array in MATLAB and manipulate the data. If you use `fscanf()` but do not transpose the array afterwards, you will need to be very careful indexing the array because it will not be in the same shape as the data in the file (the rows and columns will be switched compared to what you see in the file!). HINT: During an exam, is it particularly useful to transpose your array after using `fscanf()` so that you can index it based on the size/shape of the data printed on the page.

Finally, let's look at a detailed example where we create a text file using `fprintf()`, then read the data back into MATLAB using `fscanf()`.

Note: When given detailed example problems such as the one below, it is suggested that you copy the code into MATLAB and run it for yourself. In addition to understanding exactly what each line of code is doing as is, you should modify the code in different ways to see how the output changes.

In `myfile.m`:

```
%% First, create the text file using fprintf()
time = [0, 3, 6, 9, 12, 15];
temp = [55.3, 54.1, 54.0, 56.7, 62.9, 63.1];
RH = [67, 76, 77, 80, 90, 93];
array = [time ; temp ; RH]; % combine for printing with fprintf()
file1 = fopen('temp_RH.txt' , 'w'); % open with write permission
fprintf(file1,'%3i \t %5.1f %5.1f \n', array); % print data to file
fclose(file1); % close the file

%% Next, read in the data from the file using fscanf()
file2 = fopen('temp_RH.txt'); % omitted permission - 'r' is default
% The following will read all the data until the end of the file
% [3,6] would do the same thing as [3,inf] in this case.
A = fscanf(file2,'%f',[3,inf]); % '%f' works for decimal and integers
fclose(file2); % don't forget to close the file
disp(A) % Notice that the data is transposed when stored in A.
```

```
% This format is perfect for fprintf since it prints column by column
fprintf('%3i %7.2f %7.2f \n', A)

%% Finally, use the data to perform calculations/analysis
% Once read in, we can easily manipulate the data in A
fprintf('The mean temperature is %5.1f \n', mean(A(2,:)) )
fprintf('The mean RH is %5.1f \n', mean(A(3,:)) )
```

In `temp_RH.txt`:

```
  0       55.3  67.0
  3       54.1  76.0
  6       54.0  77.0
  9       56.7  80.0
 12       62.9  90.0
 15       63.1  93.0
```

Sample Output:

```
         0    3.0000    6.0000    9.0000   12.0000   15.0000
   55.3000   54.1000   54.0000   56.7000   62.9000   63.1000
   67.0000   76.0000   77.0000   80.0000   90.0000   93.0000

  0   55.30   67.00
  3   54.10   76.00
  6   54.00   77.00
  9   56.70   80.00
 12   62.90   90.00
 15   63.10   93.00
The mean temperature is  57.7
The mean RH is  80.5
```

As usual, we will conclude with a summary of key points to remember when using `fscanf()`:

- You must open the file with read permission and create a file identifier using `fopen()` before attempting to read in data from a file using `fscanf()`.
  - Remember to use read permission! If you accidently put a `'w'` for write permission, the file will be erased irreversibly and all data will be lost.
- You must close the file using `fclose()` after you have finished reading in the data using `fscanf()`.
- `fscanf()` is designed to read in data only from text files. However, it does not need to have a `.txt` extension, it just needs to be a plain ASCII text file (e.g., you can view it in MATLAB's

built-in text editor, Notepad++, or any other text editor program). Text files without an extension or with other extensions like `.dat` can also be read using `fscanf()`.

- When using `fscanf()`, remember to have a variable to the left hand side of the equals sign. This variable will be the array that stores all of the data that is read in from the file inside of MATLAB.
- When specifying the size of the data in the `fscanf()` command, you must specify the size of the data that exists in the file in `[col,row]` notation. The number of columns should be specified exactly based on how many columns are in the data file. The number of rows does not need to match what is in the data file – specifying less than the number of rows in the data file will read in only that many rows of data; specifying the number of rows exactly or using the `inf` keyword for the number of rows will read in all the data in the file.
    - Note that `fscanf()` fills the array in column order, such that the array that stores the data in MATLAB will be transposed compared to the size/shape of the data in the file.
        - Although it is not required, transposing the array in MATLAB after reading in the data using `fscanf()` can be useful since it puts the data back in the same size/shape that it appears in the file.
- The format specifier(s) used in `fscanf()` describe the format of the data in the file.
    - MATLAB may display the data differently in the *command window* once it is stored in the array.
- The data file that you want to open and read from must be in the same working directory (*current folder*) as your M-File in order for MATLAB to find it and access it.