Lecture 4

Text Input and Output

Objectives:

- Read input values from the command window into array variables using a script
- Output formatted text and data to the command window
- Output formatted text and data to a text file
- Read data from a text file into arrays

Key Terms:

disp

sprintf

fopen

fclose

fprintf

fscanf

sscanf

input

Outputting Data to the Screen

MATLAB has several ways of writing output to the command window. We have already used one method several times. To display a variable's value, simply type the name of the variable without a semicolon.

Script:

```
Var1=27;
Var1
```

Command Window:

```
Var1 =

   27
```

We may not always want MATLAB to echo the name of the variable to the command window. We can suppress this by using the **disp** function. Note that since the disp function always writes to the screen and doesn't generate a numerical output itself, the semicolon has no effect.

Script:

```
Var1=27;
disp(Var1);
```

Command Window:

```
   27
```

If we use single quotes within the disp function, MATLAB will treat the input as a text string instead of a variable name. This lets you print a text message to the Command Window. MATLAB will automatically move to the next line, so you can't mix variable output and text output on the same line.

Script:

```
Var1=27;
disp('Var1');
disp(Var1);
```

Command Window:

```
Var1
   27
```

The sprintf Function

To combine text and variable contents in the same output line, we need a new function called **sprintf**. This function writes formatted data to a string, and string is a data type for a variable that holds text instead of a number. Once the output text is in a string variable, it can then be sent to the Command window using disp, or the sprintf can be the input to the disp function itself.

The sprintf is a bit more complicated to use than disp. The first parameter for sprintf is always text. Within this text, we can't put a variable name and expect MATLAB to use its value. Let's see what we mean by this.

Script:

```
Var1 = 27;
mytext = sprintf('My number is Var1');
disp(mytext);
```

Command Window:

```
My number is Var1
```

The sprintf command can't tell the difference between text and variable names, so by default, everything in the single quotes is text. We can get MATLAB to insert values into the text though, but we need to use **conversion characters** instead of the variable name. A conversion character will tell the sprintf function which data type to insert and how to formant the value, and the variable name is supplied after the quoted text string.

Script:

```
Var1 = 27;
mytext = sprintf('My number is %d', Var1);
disp(mytext);
```

Command Window:

```
My number is 27
```

Conversion Characters

Here are some things to know about conversion characters.

- All conversion characters start with a % symbol.
- They all contain a single letter that specifies the type of value to display.
- A number between the % and the letter specifies at least how many characters to use to display the value (useful for formatting so that columns line up.
- When displaying floating point, a period followed by a number specifies how many digits will follow the decimal point.
- Look at the MATLAB help for sprintf for more examples and options

Here are some examples. Enter them into the Command Window to see the differences in the output.

```
disp(sprintf('%s %s %s\n%f %f %f', 'pi', 'e', 'zero', 3.1415, 2.7182, -273.15));

disp(sprintf('%7s %7s %7s\n%7f %7f %7f', 'pi', 'e', 'zero', 3.1415, 2.7182, -273.15));

disp(sprintf('%7s %7s %7s\n%7.2f %7.2f %7.2f', 'pi', 'e', 'zero', 3.1415, 2.7182,
-273.15));

disp(sprintf('%-7s %-7s %-7s\n%-7.2f %-7.2f %-7.2f', 'pi', 'e', 'zero', 3.1415, 2.7182,
-273.15));

disp(sprintf('%-7s %-7s %-7s\n%07.2f %07.2f %07.2f', 'pi', 'e', 'zero', 3.1415,
2.7182, -273.15));

disp(sprintf('%s %s %s\n%d %d %d', 'pi', 'e', 'zero', 3.1415, 2.7182, -273.15));
```

Escape Characters

Some the characters that you might want to write to a display are already being used for special purposes. For example, you can’t write a single quote, ‘, because a single quote is used to mark the end of the text. Also, you can’t write a percentage sign, %, since those are used to start conversion characters. To access these characters and other, you use an **escape character**. These act much like conversion characters, but they cause an ASCII character replacement instead of a variable value replacement.

The backslash, \, denotes an escape character, and it is always followed by one character. Here are some of the common ones.

- ‘’ – single quote (two single quotes, not a double quote)
- \\ - backslash
- \n – newline, move the cursor down one line
- \r – carriage return, move the cursor to the beginning of the line
- \t – tab
- %% - percentage sign…which oddly requires itself for the escape character

The industry has some different ideas on what should happen when you press the Enter or Return key. In many systems, a newline character is enough to move the cursor to the beginning of the next line. In MATLAB, as with more strict systems, you will need to add both a carriage return and a newline character in that order when writing to a file.

Writing to a File

Writing to a file is almost the same as writing to the Command Window using sprintf. The first difference is that you must open a file and assign it to a variable. You don't store to this variable as you would a regular one. Instead, it is only used as an input into the **fprintf** function. The fprintf function writes formatted text to a file, and other than taking the file variable as the first input, it requires a text string in quotes followed by variables just like sprintf.

First things first. The fopen command opens the file in single quotes. You use the 'w' as the second input so that the file is open in write mode. Now you can write anything to example.txt by using the myfile variable in an fprintf. You should also close the file using **fclose** so that the changes are committed to the file. Enter the script below, run it, and check the output by opening example.txt.

```
myfile = fopen('example.txt', 'w');

favnum = 27;

fprintf(myfile, 'Hello, World!');

fprintf(myfile, 'My favorite number is %d!',  favnum);

fclose(myfile);
```

You'll notice, that even though there were two fprintf statements, they also do not automatically start a new line of text. You need to insert a "\r\n" carriage return\newline characters if you want to start a new line. Alter the third line as shown below and check what happens to the text file.

```
fprintf(myfile, 'Hello, World!\r\n');
```

Reading from a File or String

The read commands are fairly similar to the write commands. Reads are called scans, so there is a sscanf for reading from a string variable and fscanf for reading from a file. Both take a variable as the input to read from, and both require a format string that contains conversions characters. The biggest difference is that when writing, the variables are supplied as inputs to fill in the conversion characters. When reading, all of the conversion characters are stored in a single output array.

- The format string should match with the read text, i.e. there is usually an agreement on the format for the file.

- The entire format string doesn’t need to match. MATLAB will start matching from the beginning of the input and the format string. As long as the two match, MATLAB will add an element to the output array for every conversion character.

- As soon as the input and the format string are different, the process stops.

- If the input is longer than the format string and the entire format string has been match, MATLAB will repeat the format string and keep matching.

- Asterisks can be used in the conversion character to make a match but ignore the value.

Read Examples

Enter the following sequences into the command window to see how the format string affects what values are read. Make sure you can explain why each line generates its output.

```
myinput = ‘1 2 3 4 5’;
sscanf(myinput, ‘%d’)
```

```
myinput = ‘1 -27 32.5 3';
sscanf(myinput, '%d %d %f %d')
```

```
sscanf('1 -27 32.5 3', '%d %d %d %d')
```

```
sscanf('1 -27 32.5 3', '%d %d %d.%d')
```

```
sscanf('1 -27 some text 32.5 3', '%d %d some text %f %d')
```

```
sscanf('1 -27 32.5 3', '%*d %d %*f %d')
```

MATLAB Input

MATLAB's sscanf and fscanf are borrowed from the C /C++ to make working with MATLAB more familiar to those programmers. MATLAB has its own built in function, input, which is tailored to reading a list of numbers into an array.

- The input function requires one parameter, which is text to display on the command window as a prompt to the user.
- It returns the value entered in the command window. Like other MATLAB functions, if you want to save the value, you must store it to a variable.
- Unlike the sscanf and fscanf functions, input can only store a value into a single variable.
- Unlike sscanf and fscanf, the input function returns an array, so the user can enter any expression in the command window that results in an array, and MATLAB will process it correctly.

Write the two commands below into a script file and run it once for each of the entries below to see how the input function works.

Script:
```
x = input('Enter a number:');
disp(x);
```

Entries:
1. 27
2. 3.14159
3. [1 2 3 4 5]
4. [1 2 3; 4 5 6]
5. [1:2:10]
6. [1 2 3 4 5].^3+1

Vectorized Output

You learned about MATLAB's vectorization in Lecture 2. When you give MATLAB an array in a command that would normally expect a scalar, MATLAB will repeat the command once using each element in the array. You saw this demonstrated in Lecture 2 using the sin() function, but it also works with sprintf and fprintf. Unfortunately, this is usually something to be avoided instead of take advantage of. Consider the two commands below.

```
x = input('Enter a number:');
disp(sprintf('You entered: %d\n',x));
```

Enter them into a script file, and run the script once entering "27" when prompted, and a second time entering "[21 22 23 24]".

As you see, the text is repeated as MATLAB inserts an element into the disp function. This is usually not want you want. Normally, you will need a second disp or a loop. Let's try just an additional disp function.

```
x = input('Enter a number:');
disp('You entered: ');
disp(x);
```

Better, but a disp function always automatically inserts a new line at the end. If we use a for loop, we can build a single string. Note the strcat() function, which is used to combine two or more strings of characters into a single string.

```
x = input('Enter a number:');
temptext = 'You entered: ';
for i=[1:1:size(x,2)]
   temptext = strcat(temptext,sprintf(' %d', x(i)));
end
disp(temptext);
```

In-Class Exercise 2

Write a script that prompts the user to enter a number between 2 and 10000, and then prints all the Fibonacci numbers less than or equal to that number to the file fibnums.txt. The format for each output line must be

Fibonacci Number *count*: Value *value*

where *count* and *value* are the index of the number and its Fibonacci value respectively. The output must be formatted so that all numbers and text are aligned into columns. A sample output for the limit 8 is shown below.

```
Fibonacci Number    1:  Value      1
Fibonacci Number    2:  Value      1
Fibonacci Number    3:  Value      2
Fibonacci Number    4:  Value      3
Fibonacci Number    5:  Value      5
Fibonacci Number    6:  Value      8
```

Demonstrate the script to the instructor, and turn in a copy of the m-file. It must have appropriate comments.