Example 4 – Printing and Plotting
Matlab provides numerous print and plot options. This example illustrates the basics and provides enough detail that you can use it for typical classroom work and assignments.

Printing Figures
Using the command `print` by itself either from the command line or in an m-file causes the current figure to be sent to the printer. If there are multiple figures, you can designate a figure number in an argument. For example `print(2)`; will print figure 2 even if figure 1 is the current figure. There are numerous formatting options including options that allow you to print a figure to a file as a jpg image. See help print for details. For classroom purposes, what is generally needed is a way to put a figure into a document. You can do this easily by choosing Edit→Copy Figure from the menu on the current figure. This option copies the current figure to the clipboard. You can then open Word and use Edit→Paste to past the figure into the Word document. Word has easy to use commands to let you resize and center the figure on your page. All of the figures in the help files were pasted in from Matlab using this technique.

One other alternative is to push the Print Screen button on the keyboard when you have a Matlab figure on the screen. This command copies the entire screen to the clipboard. You can then use Paint (from the accessories menu in Windows) and past the screen image into Paint. Paint has many options to allow you to clip out portions of an image or to add in other images or text. Once the figure is in paint it is easy to select the figure or a part of it and copy it to Word for further text documentation.

Printing Results
One of the frustrating things about Matlab is getting it to print your results to the screen in some specified format. Matlab uses the old c-style print commands and the formatting options are hard to remember if you don’t use them often.

Any expression in Matlab which does not end in a semicolon will print results to the screen. Thus if I enter `y = 5 + 7` in an m-file or the command line I get the following:

```plaintext
>> y = 5 + 7
```

```
y =

12
```

whereas if I terminate the expression with a semicolon, the printed output is suppressed.

Matlab also has a display function called `disp`. The `disp` function is similar to omitting the semicolon but printing is a bit more explicit and Matlab produces fewer blank lines in the output. For example, here’s how to print the value of `y` in the problem above using `disp`.

```plaintext
>> y = 5 + 7;
>> disp(y);
```

```
12
```
While the `disp` function does a little better job than just omitting a semicolon, the output is not formatted and you cannot display multiple items on one line. To do that you have to use the `fprintf` function.

The `fprintf` function has the following format.

```matlab
fprintf(fid, format string, variable, variable, ... );
```

In this format, `fid` is file ID field. For the screen, set `fid` to 1 or omit it completely. The format string is a Matlab string enclosed in single quotes that specifies the format of the output. The variables may be in a matrix or they may be single variables separated by commas. If the variables are in a matrix they are printed by rows.

The format string contains conversion characters that determine how the output is printed. Each conversion character must be preceded by a percent sign (%). Adding a `\n` to the format string produces a new line. The common conversion characters are:

- `c` single character
- `d` signed integer
- `e` floating point with an exponent
- `f` floating point without an exponent
- `g` general format. Uses e or f as needed.
- `i` signed integer
- `o` octal integer
- `s` string
- `u` unsigned integer
- `x` hexadecimal integer

**Figure E4-1**
Format characters for the `fprintf` command in Matlab.

This is similar to the command for formatted output in C (printf) and you can look up more details about the print format in a C manual. Here are few examples.

```matlab
i = 13; j = -45; r = 39.567; s = 5.4136789;
```

<table>
<thead>
<tr>
<th>Print Command</th>
<th><code>fprintf(...)</code></th>
<th>Printed output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fprintf(1, '%6.3f\n', r, s);</code></td>
<td>39.567 5.414</td>
<td></td>
</tr>
<tr>
<td><code>fprintf(1, 'r =%2.8f s  = %3.4f\n', r, s);</code></td>
<td>r =39.56700000 s = 5.4137</td>
<td></td>
</tr>
<tr>
<td><code>fprintf(1, 'r = %g s = %g\n', r, s);</code></td>
<td>r = 39.567 s = 5.41368</td>
<td></td>
</tr>
<tr>
<td><code>fprintf(1, 'i  = %i j = %i\n', i, j);</code></td>
<td>i = 13 j = -45</td>
<td></td>
</tr>
<tr>
<td><code>fprintf(1, 'i  = %5i j = %5i\n', i, j);</code></td>
<td>i = 13 j = -45</td>
<td></td>
</tr>
<tr>
<td><code>fprintf(1, 'i = %x\n', i);</code></td>
<td>i = d</td>
<td></td>
</tr>
<tr>
<td><code>fprintf(1, 'Hello Mom!\n');</code></td>
<td>Hello Mom!</td>
<td></td>
</tr>
</tbody>
</table>

**Figure E4-2**
Examples of format strings and the output that they produce using `fprintf`.

The most commonly used format strings are `f`, `i`, and `e`. For the `f` (floating point) and `e` (exponential) format strings the form of the string is '%w.d' where `w` is the minimum field width and `d` is the number of digits after the decimal point. Note the `w` is the `minimum` field width and the width may be longer if necessary to take care of the precision. For the `i` (signed integer) format the general form of the format string is '%wi' where `w` is the field width. The `f`, `i`, and `e` format specifiers are right justified.
If you are completely uncertain about the size of a number you want to output you can use the g (general) format string which uses either f or e as needed.

You can also use fprintf to print to a file. In this case, you need an fid number which you can get by using the command

```
fid = fopen('filename');
```

**Plotting Data**

Matlab supports a large number of plot options including plots for 2-D and 3-D graphs. For class work we will look at the `plot` and `stem` functions and their options.

**Plot Function**

The plot function creates a 2-D graph. The normal syntax for this command is

```
plot(x,y);
```

where x and y may be matrices or functions. If x and y are both matrices then it takes corresponding elements as the ordered pairs to make the plot. For example:

```
x = (0: pi/100: 4*pi);
y = sin(x);
plot (x,y);
```

produces a plot of sin(x) for x going from 0 to 4π in increments of π/100 as shown in Figure E4-3.

![Figure E4-3](image)

A plot of sin(x) from 0 to 4π.

If you use `plot(x)` alone you get a plot of the values of x versus the array element number. If x is complex it plots the real part against the imaginary part. If you use the form `plot(x, y)` and x and y are complex, the real part is plotted and the imaginary part is ignored.

Plot has a number of options to modify the color and line style.

**Plot Options**

**Color and Form**

You can get various line types and colors by adding a string option to the plot arguments. The syntax is
plot (x, y, S);

where S is a string (1, 2, or 3 characters long) created with single quotes as in Basic. Here is a table of the options.

<table>
<thead>
<tr>
<th>Color</th>
<th>Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>b – blue</td>
<td>point</td>
</tr>
<tr>
<td>c – cyan</td>
<td>o circle</td>
</tr>
<tr>
<td>g – green</td>
<td>x</td>
</tr>
<tr>
<td>k – black</td>
<td>+</td>
</tr>
<tr>
<td>m – magenta</td>
<td>- solid</td>
</tr>
<tr>
<td>r – red</td>
<td>*</td>
</tr>
<tr>
<td>w- white</td>
<td>: dotted</td>
</tr>
<tr>
<td>y – yellow</td>
<td>-. dash dot</td>
</tr>
</tbody>
</table>

Figure E4-4
Color and form characters for use in plot options

For example
plot(x, sin(x), 'k.');
produces a plot of x vs sin(x) in black made out of dots. You may also spell out the color name for clarity, e.g. 'black' or 'blue'.

**Background Color**
The whitebg function changes the figure background color (not necessarily to white. The syntax is:

```matlab
whitebg;
```

The name alone toggles the background color between white and black.

```matlab
whitebg(c) - sets the background to a specific color where c is a letter form the table above as a string. For example to get a blue background you would type:

```matlab
whitebg('b');
```

**Grid Lines**
The grid function adds grid lines to an existing plot. If they are already on the plot then grid toggles them off.

**Axis**
The axis function lets you scale the x and y axis after the plot has been completed. The syntax is:

```matlab
axis([Xmin  Xmax  Ymin  Ymax]);
```

Notice the confusing requirement for both parentheses and brackets in this command.

**Title**
The title function adds a title to the existing graph. The format is:

```matlab
title('THIS IS A TITLE');
```

The title gets printed centered at the top of your graph.

**Axis Labels**
To add a label to the axis of a graph you use the xlabel and ylabel commands. The format is similar to title.

**Adding Text to a Graph**
The text function allows the user to add text information to a figure at a specified coordinate. The syntax is:

\[
\text{text}(x, y, 'text');
\]

where \((x, y)\) is the lower left corner of the text block and ‘text’ is the text added. You can experiment to get the correct coordinates.

**Subplots**
The subplot \((m, n, p)\) function divides the figure into \(m\) rows and \(n\) columns for subplots. \(p\) is the number of the subplot that is current. For example,

\[
\text{subplot}(2,2,1)
\]

creates two rows and two columns for four subplots. It makes the top left current. The following creates two graphs with \(y = \sin(x)\) on the top half and \(z = \cos(x)\) on the bottom half.

```matlab
clear
subplot(2,1,1); % 2 rows - 1 column
x=(0:pi/100:4*pi); % x goes from 0 to 4\pi in steps of \pi/100
y=sin(x);
plot(x,y); % plots \sin(x) in top subplot
subplot(2,1,2); % makes the bottom subplot current
z=cos(2*x);
plot(x,z); % plots \cos(2x) in the bottom subplot
```

**Figure E4-5**
This set of commands creates two subplots on one figure as seen in Figure E4-6.

![Figure E4-5](image)

The top subplot is number 1 and has the sin function. The bottom subplot is number 2 and has the cosine function.

**Stem Function**
The stem function creates a stem plot. This is supposed to be a plot of discrete data with the options of the plot function (above). The syntax is:

\[
\text{stem}(x, y);
\]
where x and y are matrix functions. A stem plot is a sequence of lines from the y value to the x axis. The color and line style are the same for stem as they are for plot. For example, the following commands produce the two stem plots in Figure E4-7.

```matlab
x = -2*pi:pi/10:2*pi;
y = sin(x);
figure(1);clf;
stem(x, y);
figure(2);clf;
stem(x, y, '.');
```

![Figure E4-7](image)

**Figure E4-7**
Two stem plots.

**Other Useful Plot Options**

- **semilogx, semilogy, and loglog** - These are the same as the plot command with the same syntax. They produce log (base 10) plots on either the x, y, or both axes.

- **Multiple plots** - To get more than one plot on an axis you can string vectors together inside the plot argument. For example
  ```matlab
  x = (0 : pi/100 : 4*pi);
  plot(x, sin(x), x, cos(x));
  ```
  produces a plot of x vs sin(x) and x vs cos(x) in two different colors for x from 0 to $4\pi$ in steps of $\pi/100$.

- **hold** - holds the current plot so that the next plot function is plotted without erasing the first plot. You can use hold alone to toggle the hold function or you can use hold on or hold off for clarity.