Data Visualization Using MATLAB

MATLAB is not only useful for analyzing systems, but it is also helpful for displaying and interpreting the analysis results. This is especially important when we want to examine particular aspects of a solution (such as its behaviour over a period of time or its frequency response) or if we have a number of solutions that we want to compare.

Plotting

As we learned in the previous session, it is easy to create plots in MATLAB. The `plot` function displays a graph of pairs of vectors, the elements of which together comprise a sequence of ordered pairs. For example, `plot(t, y)` displays the vectors `t` on the x axis and `y` on the y axis:

```matlab
t = 0 : 0.25 : 7;
y = sin(t);
plot(t, y)
```

Note that the plot command does not have a semicolon after it; otherwise, output is suppressed.

To specify the x-axis range (a,b) and y-axis range (c,d), type:

```matlab
plot(t, y), axis([abcd])
```

The axis command is useful for maintaining the same scale when replotting.

To annotate the plot, the following commands are useful. They should be executed after the plot command. Note that the argument for each command is a character string enclosed with single quote marks.

- `title('title')` - Graph title.
- `xlabel('label for the x axis')` - X-axis label.
- `ylabel('label for the y axis')` - Y-axis label.

The `hold` command keeps the existing graph and adds the new data to the current graph; but the graph is rescaled if necessary to include new data that are outside of the previous range of axes.

Multiple plots can be shown on a single graph using the subplot command, which has three arguments: `m`, `n`, and `p`. The first two arguments specify that the graph is to split into an set of subplots with `m` rows and `n` columns. The third argument specifies which subplot is to be drawn the next time the plot command is called (starting at 1 for the top left and then proceeding across the first row left to right, then on to the subsequent rows). For example,

```matlab
subplot(3, 2, 1), plot(x, y)
```

plots the upper left plot of a figure that has three rows and two columns, and
Here is a listing of commands that are particularly useful for two-dimensional plots:

**Elementary X-Y graphs.**

- `plot` - Linear plot.
- `loglog` - Log-log scale plot.
- `semilogx` - Semi-log scale plot.
- `semilogy` - Semi-log scale plot.
- `fill` - Draw filled 2-D polygons.

**Specialized X-Y graphs.**

- `polar` - Polar coordinate plot.
- `bar` - Bar graph.
- `stem` - Discrete sequence or ”stem” plot.
- `stairs` - Stairstep plot.
- `errorbar` - Error bar plot.
- `hist` - Histogram plot.
- `rose` - Angle histogram plot.
- `compass` - Compass plot.
- `feather` - Feather plot.
- `fplot` - Plot function.
- `comet` - Comet-like trajectory.

Clear presentation of data in context is extremely important for understanding and communicating to others. Avoid the temptation to make the graph pretty; instead, try to make the relationships amongst the data simple to see.

For time series data, animation can be a fantastic way to see what is happening as a sequence progresses through time. Animated sequences can be created in MATLAB in two ways:

- save a set of graphs as pictures and play them back as a movie; or
- make incremental changes by erasing and redrawing some of the objects.

Movies are a better idea when each ”frame” is complicated and would take a lot of calculation time to produce each picture. So, each movie frame is done in advance, and then the movie just plays back the set of pictures that has been created previously. Once you have a figure that you want to capture, you use the command `getframe` to generate a movie frame (making sure that the computer is not in screen saver mode when `getframe` is called, and the desktop in which MATLAB is running is visible). Usually `getframe` is used in a for loop to generate an array of movie frames. Once the set of frames has been saved, then the command `movie` will run the movie a specified number of times at the specified rate.

A data point can be made to move on a figure by selectively erasing old data and then drawing the new point on the plot. This is typically done by drawing a graphics object, and then changing its position by respecifying

```matlab
csubplot(3, 2, 4), cplot(x, y)
cplots the right-hand cplot of the middle row.
```
the x and y coordinate data by a small amount at the next step through a loop. (This can be done for 3-D plots as well.) Different effects are possible by selecting one of different erase modes as an optional argument in the plot command:

none - none of the objects are erased, which is like leaving a trail as the trajectory is generated;
background - the object is erased by redrawing it in the background color. This mode erases not only the object but also anything behind it (such as grid lines);
xor - only the object is erased, and so this is usually what is used for animation.

Here is an example of how it is used (modified from the on-line MATLAB documentation). Before the set of data is generated, the plot command is set up:

\[ p = \text{plot}(y(1), y(2), ' EraseMode', ' none', ' MarkerSize', 5); \% Set EraseMode to none \]

\[ \text{axis}([0 50 -25 25]) \]

\[ \text{holdon} \]

The hold command is used so that the characteristics of the plot will persist as the animation is generated. Then the following command is executed inside the for loop that produces the data to be animated:

\[ \text{set}(p, ' XData', y(1), ' YData', y(2)) \% assign new datum coordinates \]

\[ \text{drawnow} \% update plot with the new datum \]

There are a couple of cool animation demos in MATLAB help that you can try out.

Symbolic Expressions

In MATLAB, symbolic algebraic expressions can in many cases be manipulated and evaluated directly. The expressions are stored as character string, for example

\[ '\tan(y/x)' \]
\[ '\ t^2-2*t^2+3' \]
\[ '\t^3-a*b-6' \]

The precedence of operations convention must be followed. Variables have a single character; \( i \) and \( j \) are reserved for use as complex numbers. For expressions with more than one variable, the independent variable is specified (the default is \( x \)).

An expression can be assigned to a variable name, e.g.,

\[ S1 = 'x^3 - 1'; \]

Standard arithmetic operations can be done using symbolic functions. For example, the expressions

\[ p1 = '1/(y - 3)'; \]
\[ p2 = '3*y/(y + 2)'; \]
\[ pb = '(y + 4) * (y - 3) * y'; \]

when operated on have the following result: for symbolic multiplication \( \text{symmul}(p1,p3) \) the answer is \( (y + 4) * y. \)
A simple way to plot a symbolic expression of one variable is to use the function `ezplot`:

\[ ezplot(S1) \]

which can also be specified for a range of the independent variable from \( x_{\text{min}} \) to \( x_{\text{max}} \):

\[ ezplot(S1, [x_{\text{min}}, x_{\text{max}}]) \]

One nice aspect of using this plotting method is that you do not have to worry about element-by-element operations. You type the expression as you would evaluate it for a scalar operation.

**Symbolic Expressions**

Symbolic equations are used to solve single equations and also systems of equations. The function `solve(f)` solves the symbolic equation \( f \) for the symbolic variable (if \( f \) is an expression, then the solution is given for \( f = 0 \)). The function `solve(f1, , fn)` solves the system of equations expressed symbolically by the \( N \) equations \( f1 \) to \( fn \).

**Direct input and output at the MATLAB command line**

It is sometimes convenient to enter information at the command line when running a command (such as a function). The input can take a scalar or a vector input, for example:

\[ x = \text{input}(‘Enter your height in metres: ’); \]

To display a variable or write a text string, the command `disp` is handy for displaying text enclosed in single quote marks or to print the contents of a vector or matrix. To control the output more than a single output per line, the command `fprintf` is much more versatile. For example, information about the variable `temp` with value of 23 can be printed out in this way:

\[ \text{fprintf(‘The temperature is } \%f \text{ degrees Celsius } \backslash n’, \text{temp})} \]

And the output would be

*The temperature is 23 degrees Celsius*

including a carriage return at the end to start a new line.

**Exercises**

An advanced turboprop aircraft engine called the unducted fan is currently under development, which promises to be more powerful and quieter than conventional turboprops presently used on small commercial aircraft. This exercise involves generating the velocity and acceleration plots for data from a test flight, with conditions as follows: initially engine power is 40,000 N, and the 20,000 kg plane has a cruising speed of 180 m/s. Throttling up to 60,000 N causes the plane to have an acceleration \( a \), with drag increasing proportionally to the square of the airspeed \( v \) until the plane reaches a new steady-state airspeed, according to

\[ v = 0.00001t^3 - 0.00488t^2 + 0.75795t + 181.3566 \quad (1) \]
where \( t \) is time. Assume that throttle-up occurs at \( t = 0 \).

1) Write an m-file program in MATLAB that asks the user to enter start and end times (in seconds) over which to plot the velocity and acceleration of the aircraft, and then generates the plots. The end time should be no more than 12 seconds. Use on the order of 100 points over the time interval for plotting, and then generate the velocity and acceleration data. Plot both velocity and acceleration plots on the same figure using subplots with identifying titles.

2) Plot the velocity and acceleration using symbolic expressions.

3) Plot the velocity as an animation.

Submit your m-file codes for the 3 exercises.