MATLAB comes with extensive plotting tools, and comes with extremely detailed documentation online. We will only touch on the basics here and provide relevant references for further reading.

1 Line plots

The basic syntax for creating line plots is \texttt{plot(x,y)}, where \texttt{x} and \texttt{y} are arrays of the same length that specify the \((x,y)\) pairs that form the line. For example, let’s plot the cosine function from \(-2\) to \(1\). To do so, we need to provide a discretization (grid) of the values along the \(x\)-axis, and evaluate the function on each \(x\) value. This can typically be done with : (colon operator) or \texttt{linspace}.

\begin{verbatim}
xvals = -2:0.01:1; % Grid of 0.01 spacing from -2 to 10
yvals = cos(xvals); % Evaluate function on xvals
plot(xvals, yvals); % Create line plot with yvals against xvals
\end{verbatim}

Suppose we want to add another plot, the quadratic approximation to the cosine function. By default additional calls to \texttt{plot} replaces the plot in the current figure window. We can prevent that by calling:

\begin{verbatim}
hold on; % Prevent next plot from replacing existing one
\end{verbatim}

This allows you to add plots on top of previous ones. We do so below using a different color and line type. We also add a title and axis labels, which is \textit{highly recommended} in your own work.

\begin{verbatim}
newyvals = 1 - 0.5 * (xvals.^2); % Evaluate quadratic approximation on xvals
plot(xvals, newyvals, 'r--'); % Create line plot with red dashed line

title('Example plots');
xlabel('Input');
ylabel('Function values');
\end{verbatim}

The third parameter supplied to \texttt{plot} above is an optional format string. The particular one specified above gives a red dashed line. See the extensive MATLAB documentation online for other formatting commands, as well as many other plotting properties that were not covered here:

\url{http://www.mathworks.com/help/techdoc/ref/plot.html}
2 Contour plots

The basic syntax for creating contour plots is `contour(X,Y,Z,levels)`. To trace a contour, `contour` requires a 2-D array `Z` that specifies function values on a grid. The underlying grid is given by `X` and `Y`, either both as 2-D arrays with the same shape as `Z`, or both as 1-D arrays where `length(X)` is the number of columns in `Z` and `length(Y)` is the number of rows in `Z`.

In most situations it is more convenient to work with the underlying grid (i.e., the former representation). The `meshgrid` function is useful for constructing 2-D grids from two 1-D arrays. It returns two 2-D arrays `X,Y` of the same shape, where each element-wise pair specifies an underlying `(x,y)` point on the grid. Function values on the grid `Z` can then be calculated using these `X,Y` element-wise pairs.

```matlab
figure; % Create a new figure window
xlist = linspace(-2.0, 1.0, 100); % Create 1-D arrays for x,y dimensions
ylist = linspace(-1.0, 2.0, 100);
[X,Y] = meshgrid(xlist, ylist); % Create 2-D grid xlist,ylist values
Z = sqrt(X.^2 + Y.^2); % Compute function values on the grid
```

We also need to specify the contour levels (of `Z`) to plot. You can either specify a positive integer for the number of automatically-decided contours to plot, or you can give a list of contour (function) values in the `levels` argument. For example, we plot several contours below:

```matlab
% Create contour plot with 2-D grids, 4 contour levels, black solid contours
contour(X, Y, Z, [0.5 1.0 1.2 1.5], 'k');
```

Note that we also specified the contour colors to be black. Again, many properties are described in the MATLAB specification:

```
http://www.mathworks.com/help/techdoc/ref/contour.html
```

3 More plotting properties

The function considered above should actually have circular contours. Unfortunately, due to the different scales of the axes, the figure likely turned out to be flattened and the contours appear like ellipses. This is undesirable, for example, if we wanted to visualize 2-D Gaussian covariance contours. We can force the aspect ratio to be equal with the following command:

```matlab
axis equal; % Scale the plot size to get same aspect ratio
```

Finally, suppose we want to zoom in on a particular region of the plot. We can do this by changing the axis limits. The input list to `axis` has form `[xmin xmax ymin ymax]`

```matlab
axis([-1.0 1.0 -0.5 0.5]); % Set axis limits
```

Notice that the aspect ratio is still equal after changing the axis limits. Also, the commands above only change the properties of the current axis. If you have multiple figures you will generally have to set them for each figure before calling `figure` to create the next figure window.

You can find out how to set many other axis properties at:

```
```
4 Figures

Figure 1: Example from section on line plots.

Figure 2: Example from section on contour plots.
Figure 3: Setting the aspect ratio to be equal and zooming in on the contour plot.

5 Code

```matlab
xvals = -2:0.01:1; % Grid of 0.01 spacing from -2 to 10
yvals = cos(xvals); % Evaluate function on xvals
plot(xvals, yvals); % Create line plot with yvals against xvals

hold on; % Prevent next plot from replacing existing one
newyvals = 1 - 0.5 * (xvals.^2); % Evaluate quadratic approximation on xvals
plot(xvals, newyvals, 'r--'); % Create line plot with red dashed line

figure; % Create a new figure window
xlist = linspace(-2.0, 1.0, 100); % Create 1-D arrays for x,y dimensions
ylist = linspace(-1.0, 2.0, 100);
[X,Y] = meshgrid(xlist, ylist); % Create 2-D grid xlist,ylist values
Z = sqrt(X.^2 + Y.^2); % Compute function values on the grid

contour(X, Y, Z, [0.5 1.0 1.2 1.5], 'k'); % Create contour plot with 2-D grids, 4 contour levels, black solid contours
axis equal; % Scale the plot size to get same aspect ratio
axis([-1.0 1.0 -0.5 0.5]); % Set axis limits
```