Interactive Robotics Letters

Tutorial

Presenting Engineering Data Using MATLAB Figures

Matthew Rhudy* and Yu Gu†

West Virginia University, Morgantown, WV 26506, USA

***Abstract*— Research ideas can be effectively communicated using visual representations of data. Computer software, such as MATLAB, provides many useful tools which can be used to aid in the preparation of illustrative figures. Some general recommendations are provided in this article in order to generate quality figures for various technical works. Additionally, more detailed examples are offered to provide examples of these techniques in practice.**

***Index Terms*—MATLAB, Programming**

I. Introduction

When it comes time to prepare figures for a paper, it is important to find a way to show the data in a manner that is both meaningful and easy to understand. Most of the time there are many different ways to display the data. It is important to consider the different options in order to find an effective representation of the results so that readers can easily understand what is being communicated. A recommended approach for creating quality figures is to try a few different ideas, and see what turns out the best. Take time in preparing figures, as they are a very important part of the document, and many readers may only look at the figures before making judgements about the work. Be aware of the different tools that are available. While standard line plots are perhaps the most common, MATLAB has many different built-in functions that provide different visual representations of data, such as bar plots, contour plots, surface plots, histograms, pie charts, 3D plots, etc. The purpose of this article is to provide suggestions for preparing effective figures using MATLAB. Note that this tutorial assumes the reader has some programming experience using MATLAB and a basic understanding of the plotting commands and features. In this article, some general suggestions for plotting are provided, followed by more advanced examples. Each section is written to be standalone; however there is some overlap between sections, as combinations of the different techniques are encouraged.

II. General Tips and Discussion

This section provides a few general tips which can be used in many different situations to create effective figures. This is not a complete guide to every possible feature in MATLAB, but instead highlights a few select cases that were found to be

* Ph.D. Candidate, Department of Mechanical and Aerospace Engineering

† Assistant Professor, Department of Mechanical and Aerospace Engineering and Adjunct Assistant Professor, Lane Department of Computer Science and Electrical Engineering

Citation: Rhudy, M., and Gu, Y., "Presenting Engineering Data Using MATLAB Figures," *Interactive Robotics Letters*, West Virginia University, June 2013. Link: http://www2.statler.wvu.edu/~irl/page13.html

important for generating quality figures for technical documents and presentations. More information and details on these topics can be found in the MATLAB help feature (Mathworks, 2013).

A. Apply Grid Lines

Figures can sometimes be easier to read with grid lines. For many situations, it is recommended that grid lines be added to a figure. This is done by simply executing the command *grid on* after a plotting command. There are also options to show the minor gridlines, or to use grid lines on only the *x*- or *y*-axes. Examples of the different cases of grid lines are shown in Figure 1.

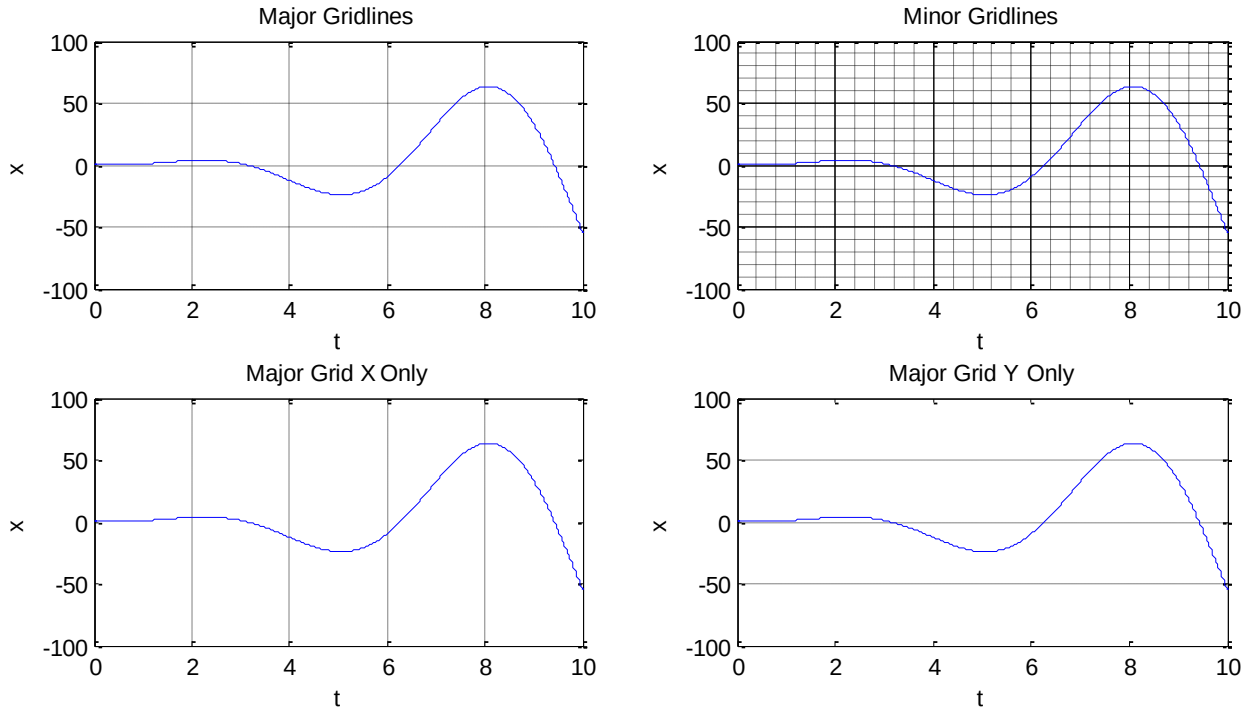

Figure 1. Examples of Grid Lines

```
%% Example Code for Grid Lines
t = 0:0.001:10;
x = t.^2.*sin(t);
subplot(221); plot(t, x); grid on;
xlabel('t'); ylabel('x'); title('Major Gridlines');
subplot(222); plot(t, x); grid minor;
xlabel('t'); ylabel('x'); title('Minor Gridlines');
subplot(223); plot(t, x); set(gca, 'XGrid', 'on');
xlabel('t'); ylabel('x'); title('Major Grid X Only');
subplot(224); plot(t, x); set(gca, 'YGrid', 'on');
xlabel('t'); ylabel('x'); title('Major Grid Y Only');
```

B. Increase Width of Lines

The default line width in MATLAB tends to be rather thin when copying into documents and presentations. This is especially true if the size of the figure is small. Increasing the line width to 2 provides a good thickness for most applications. Figure 2 shows a line with default width of 1 (left) and with increased width of 2 (right).

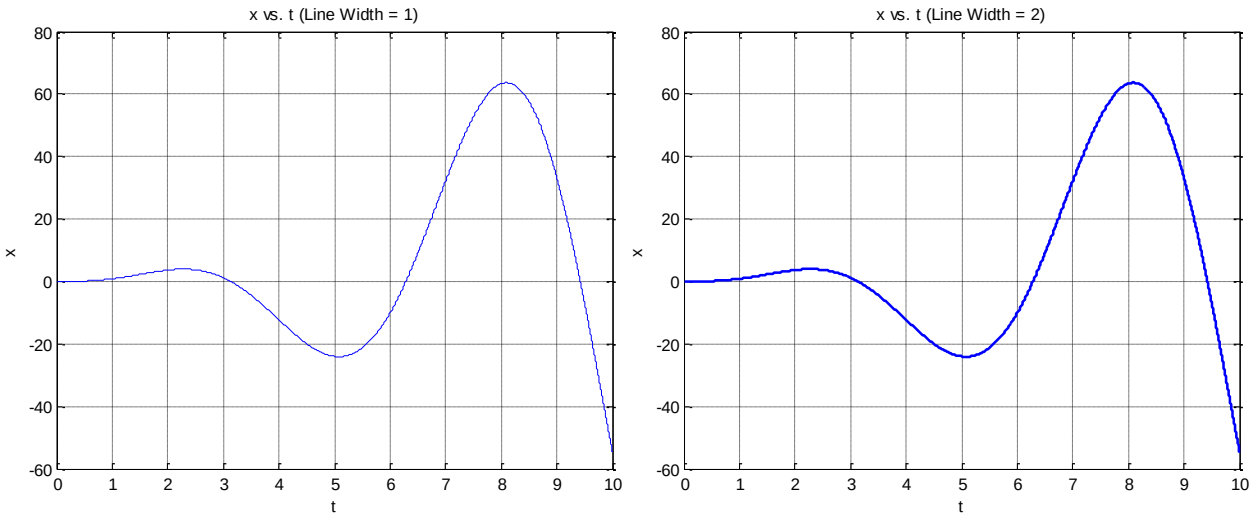

Figure 2. Example for Increasing Line Width

```
%% Example Code for Changing Line Width
t = 0:0.001:10;
x = t.^2.*sin(t);
plot(t, x);
xlabel('t'); ylabel('x'); title('x vs. t (Line Width = 1)'); grid on;
figure(2); plot(t, x, 'LineWidth', 2);
xlabel('t'); ylabel('x'); title('x vs. t (Line Width = 2)'); grid on;
```

C. Make Figure Readable in Black and White

Although many publishers now support color figures for publication, there are still cases where an originally color figure may end up in black and white, e.g. a document might be printed on a black and white printer. Because of this, it is a good idea to whenever possible make figures understandable in both color and black and white. This can sometimes be a challenge, especially with a large number of legend entries. There are two methods which can be used to handle this issue. One method is to vary the line style for each line in a single figure. MATLAB has four possible line types: solid '-', dashed '--', dash-dot '-.', and dotted ':' (Mathworks, 2013). This can give you up to four unique line types that do not rely on color to distinguish from one another. This property can be changed by setting the 'LineStyle' property, or directly on the plotting line, e.g. plot(x, y, '--'). If more than 4 lines are required, markers can be placed on the lines to distinguish the different values. There are 13 types of markers in MATLAB: plus sign '+', circle 'o', asterisk '*', point '.', cross 'x', square 's', diamond 'd', up/down/right/left triangles '^' 'v' '>' '<', pentagon 'p', and hexagon 'h' (Mathworks, 2013). The marker can be set using the 'Marker' property or directly on the plotting line, e.g. plot(x, y, '+'). For data sets with a large number of points, a marker might not be necessary at every point along the curve. Instead, a marker can be placed at set intervals along the line to provide indication of which line is which. Examples of both methods of distinguishing lines are provided in Figure 3.

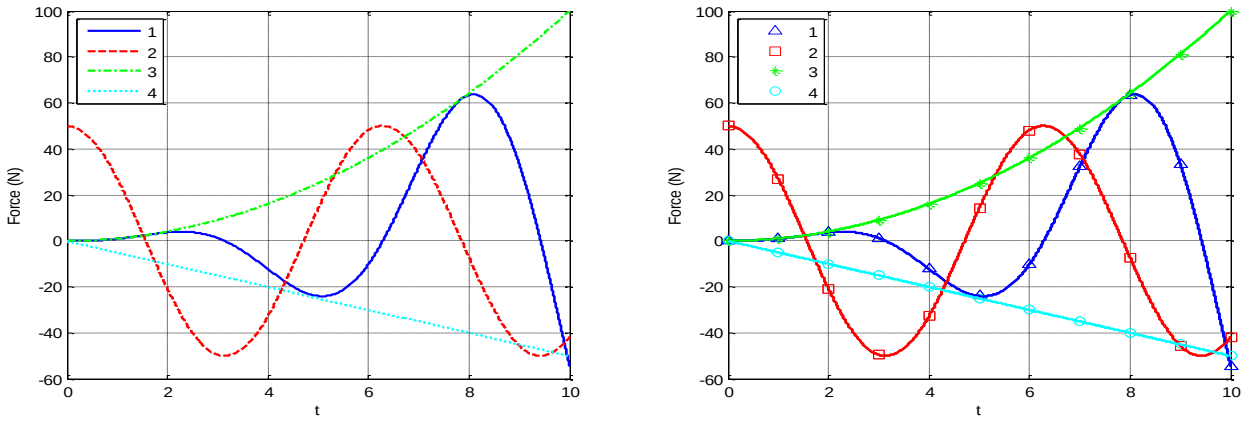

Figure 3. Examples for Display in Black and White

```
%% Example Code for Black and White Line Labeling
t = 0:0.001:10;
x = t.^2.*sin(t);
y = 50*cos(t);
z = t.^2;
r = -5*t;
% Plot Using Different Line Styles
subplot(121);
plot(t, x, 'b-', t, y, 'r--', t, z, 'g-.', t, r, 'c:', 'LineWidth', 2)
xlabel('t'); ylabel('Force (N)'); grid on;
legend('1','2','3','4','Location','NorthWest');
% Plot Using Different Symbols
ind = 1:1000:length(t);     % Index for every 1000th point to apply a marker
subplot(122);
% Plot the Markers at Set Intervals
plot(t(ind), x(ind), 'b^', t(ind), y(ind), 'rs', t(ind), z(ind), 'g*', t(ind), r(ind), 'co')
% Plot the Lines
hold on; plot(t, x, 'b-', t, y, 'r-', t, z, 'g-', t, r, 'c-', 'LineWidth', 2)
xlabel('t'); ylabel('Force (N)'); grid on; hold off;
legend('1','2','3','4','Location','NorthWest');
```

D. Challenge Color Defaults

MATLAB relies on default colors for lines based on the following presets: yellow 'y', magenta 'm', cyan 'c', red 'r', green 'g', blue 'b', white 'w', and black 'k'. However, since these presets are very common, using colors outside of these default values may make your figures stand out. To define a different color, use RGB components to define the color as a vector of three values between 0 and 1. E.g. white is [1 1 1], black is [0 0 0], and blue is [0 0 1]. While essentially any color can be created from these three components, two different examples are provided using two different types of figures in Figure 4.

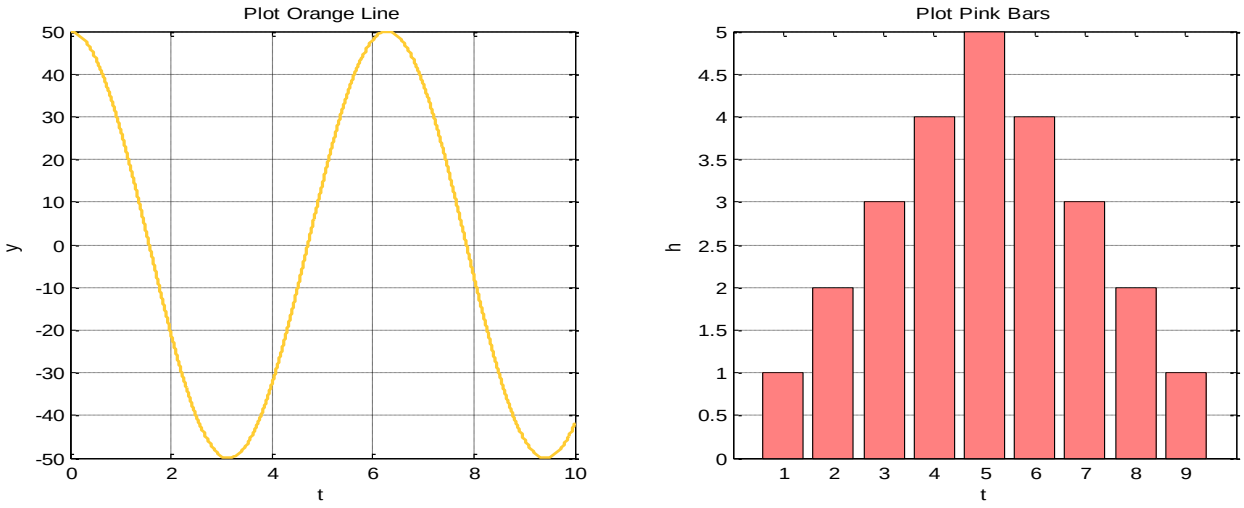

Figure 4. Examples for Changing Colors

```
%% Example Code for Changing Colors
t = 0:0.001:10;
y = 50*cos(t);
h = [1 2 3 4 5 4 3 2 1];
subplot(121); plot(t, y, 'Color', [1 0.8 0.2], 'LineWidth', 2);
xlabel('t'); ylabel('y'); grid on; title('Plot Orange Line');
subplot(122); bar(h, 'FaceColor', [1 0.5 0.5])
xlabel('t'); ylabel('h'); set(gca, 'YGrid', 'on'); title('Plot Pink Bars');
```

E. Assign Proper Labels

It is important for every figure to contain proper labels including the appropriate units. This is typically done in MATLAB through the use of the *xlabel*, *ylabel*, *title*, and *legend* commands. Within any of these commands, the text inside of single quotes will be displayed. In this text, various symbols can be entered using \ followed by the shortcut for the symbol. For example, Greek letters can be input using '\sigma' for the lower case and '\Sigma' for the upper case version of the symbol. Other symbols such as infinity 'infty' can be used as well. The *legend* command by default is placed in the top right corner of the figure window. This location can be set automatically within the legend command using the 'Location' property to set the location as 'NorthEast' (default), 'NorthWest', 'SouthEast', 'SouthWest', 'North', 'South', 'East', or 'West'. Alternatively, the legend can be moved manually using the mouse by clicking and dragging to the desired location. Titles and labels can be set to multiple lines by using a cell array in the argument, i.e. title({'First Line', 'Second Line', 'Third Line'}). Bold and italics can be used in labels by using /bf and /it respectively prior to the text. If only part of the text needs to be bold or italics, /rm can be used to turn of the formatting, e.g. xlabel('\bfThis is in bold\rm but this is not bold'). Superscripts and subscripts can be used by using the ^ and _ characters respectively before the character of interest. If there are multiple characters that need to be in the subscript, the _ must be used again, e.g. xlabel('x_s_u_b and x^s^u^p^e^r'). An example including each of these items is provided in Figure 5.

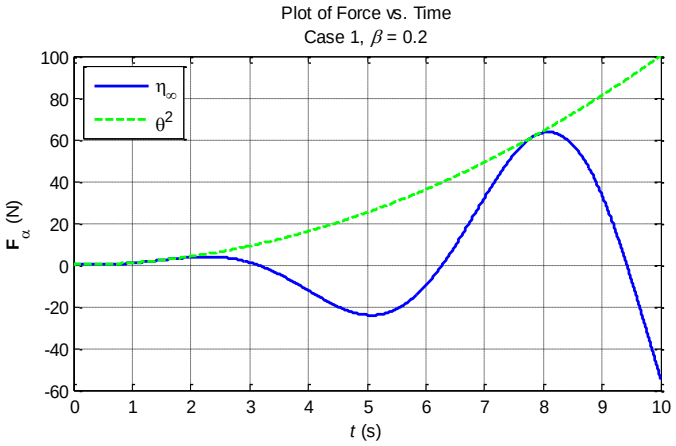

Figure 5. Example of Labeling

```
%% Example Code for Labeling
t = 0:0.001:10;
x = t.^2.*sin(t);
z = t.^2;
plot(t, x, 'b-', t, z, 'g--', 'LineWidth', 2); grid on;
xlabel('\itt\rm (s)'); ylabel('\bfF\rm_\alpha (N)');
title({'Plot of Force vs. Time','Case 1, \it\beta\rm = 0.2'});
legend('\eta \infty','\theta^2','Location','NorthWest');
```

F. Use Subplot Command Effectively

The *subplot* command provides a useful means to organize multiple plots within a single figure window. This is particularly useful for displaying related data. Typical use of the *subplot* command provides equally sized figures across a grid defined by the first two arguments of the *subplot* command, while the third entry of the command dictates which item on the grid is currently being plotted. The numbering of the third input to the *subplot* command is left-to-right then top-to-bottom. This is convenient because plotting can be done in a loop. An example is provided in Figure 6.

```
%% Example Code for Same Size Subplots
t = 0:0.001:10;
x = t.^2.*sin(t);
for i = 1:6
  subplot(3, 2, i); plot(t, x);
  xlabel('t'); ylabel('x');
  title(['subplot(3,2,',num2str(i),')']);
  grid on;
end
```

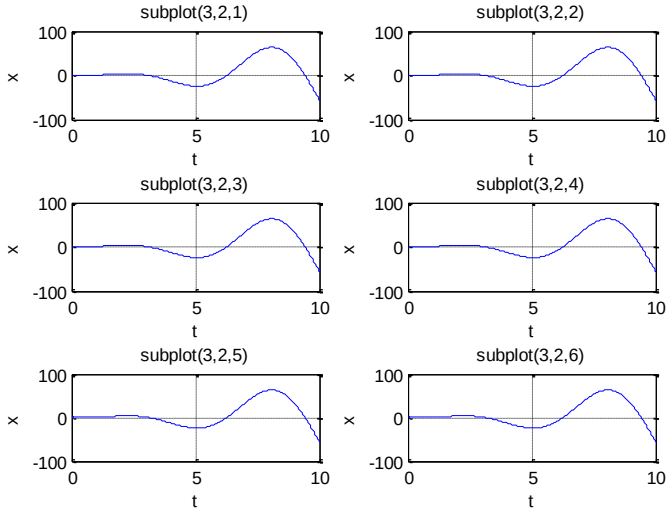

Figure 6. Example of using Same Size Subplots

The *subplot* command can also be used for different sized plots within a single figure window. To do this, the third argument of the subplot command is given as a vector of each block in the grid that the figure should occupy, e.g. see Figure 7.

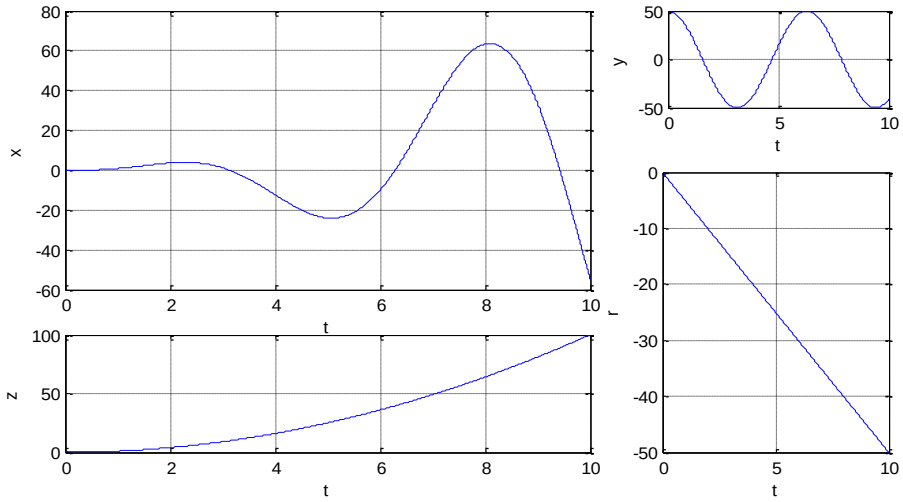

Figure 7. Example of using Different Size Subplots

```
%% Example Code for Different Size Subplots
t = 0:0.001:10;
x = t.^2.*sin(t);
y = 50*cos(t);
z = t.^2;
r = -5*t;
subplot(3,3,[1 2 4 5]); plot(t, x); xlabel('t'); ylabel('x'); grid on;
subplot(3,3,3); plot(t, y); xlabel('t'); ylabel('y'); grid on;
subplot(3,3,[7 8]); plot(t, z); xlabel('t'); ylabel('z'); grid on;
subplot(3,3,[6 9]); plot(t, r); xlabel('t'); ylabel('r'); grid on;
```

Another possible use of this method is to provide a small inset to the main figure with a zoomed in version of a particular part of the plot of interest, e.g. see Section 5.3 in (Rhudy, 2013).

G. Use Logarithmic Scales to Appropriately Display Data

Using logarithmic scales on the *x* and/or *y* axes of a figure can sometimes help to show data in a more meaningful way. This is useful for quantities which are typically shown in logarithmic scale such as frequency, or in general for particular data sets which have significant changes in order of magnitude. To apply a log scale on the *x*-axis, use the command *semilogx* in place of *plot*. For a log scale on the *y*-axis only, using *semilogy*, and for both *x* and *y* axes use *loglog*. Alternatively, the property 'XScale' and/or 'YScale' can be changed from 'Linear' to 'Log'. Examples of the different cases of linear and logarithmic axes are given in Figure 8.

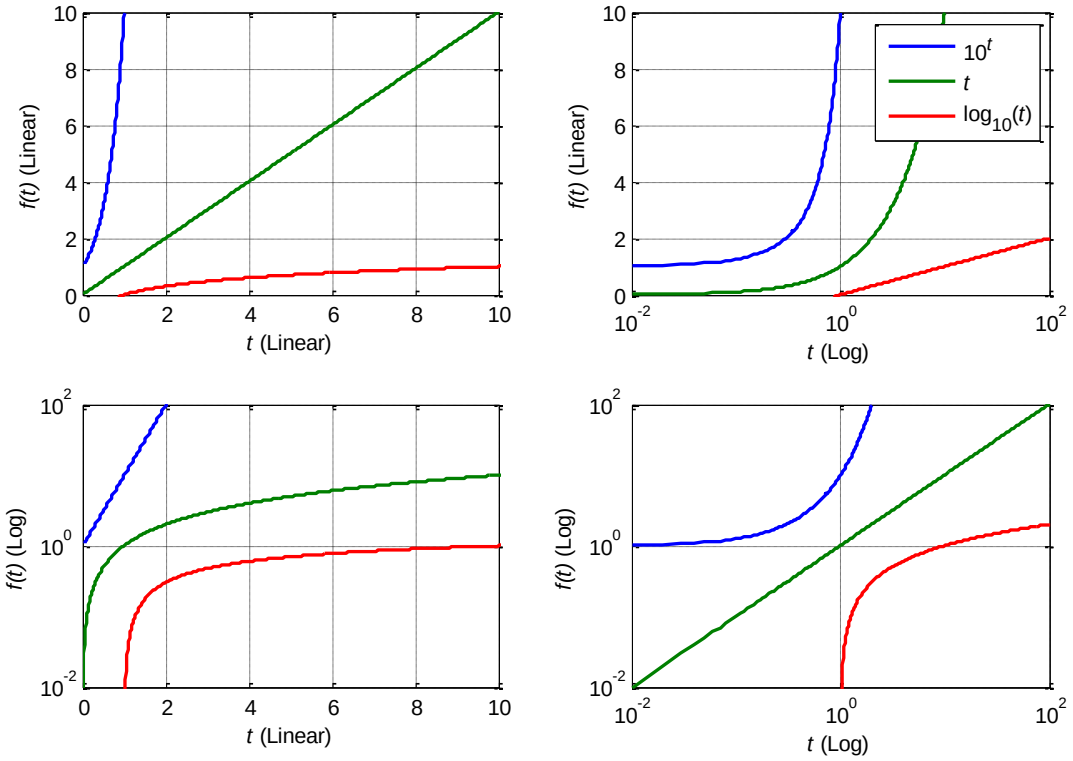

Figure 8. Examples of Linear and Logarithmic Scales

```
%% Example Code for Logarithmic Scale Axes
t = 0.01:0.01:100;
x = 10.^t;
y = t;
z = log10(t);
subplot(221); plot(t, x, t, y, t, z, 'LineWidth', 2); grid on;
xlabel('\itt\rm (Linear)'); ylabel('\itf(t)\rm (Linear)'); axis([0 10 0 10]);
subplot(222); semilogx(t, x, t, y, t, z, 'LineWidth', 2); grid on;
xlabel('\itt\rm (Log)'); ylabel('\itf(t)\rm (Linear)'); axis([0.01 100 0 10]);
legend('10\it^t\rm','\itt','log_1_0(\itt\rm)');
subplot(223); semilogy(t, x, t, y, t, z, 'LineWidth', 2); grid on;
xlabel('\itt\rm (Linear)'); ylabel('\itf(t)\rm (Log)'); axis([0 10 0.01 100]);
subplot(224); loglog(t, x, t, y, t, z, 'LineWidth', 2); grid on;
xlabel('\itt\rm (Log)'); ylabel('\itf(t)\rm (Log)'); axis([0.01 100 0.01 100]);
```

III. Special Types of Figures

This section provides a few examples of some more advanced figures that the authors have used for various other works. In the example code for these figures, the plotting data is not defined for brevity.

A. *Bar Graphs*

Bar graphs can be a useful display tool, and can provide a nice visual alternative to tables for displaying and comparing data. Numeric bar graphs are relatively straight forward; however it is sometimes necessary to add text labels to clarify which values are being shown. One technique which can be used to substitute numeric labels for text is to change the ticks and tick labels by setting the properties 'XTick', 'YTick', 'XTickLabel', and 'YTickLabel'. Horizontal bar graphs (using *barh*) tend to be better for this because the *y* axis tick labels can be as long as necessary, while x tick labels can have issues with running together. The basic idea here for modifying the *y* axis labels is to set a tick at each location where you want a line of text. Then change the tick labels to instead of the number along the axis be a string of text. This technique is illustrated in the example provided in Figure 9. For the entries which require two lines of text, ticks are set just above and below (0.15 in this example) and then each line is input individually. Note that the labels are given bottom up on the plot, so they look backwards in the command. It can also be useful to add additional text labels on the figure using the *text* command. Adding text labels takes a bit of time and tuning to get the text in just the right location, but can add clarity to a figure.

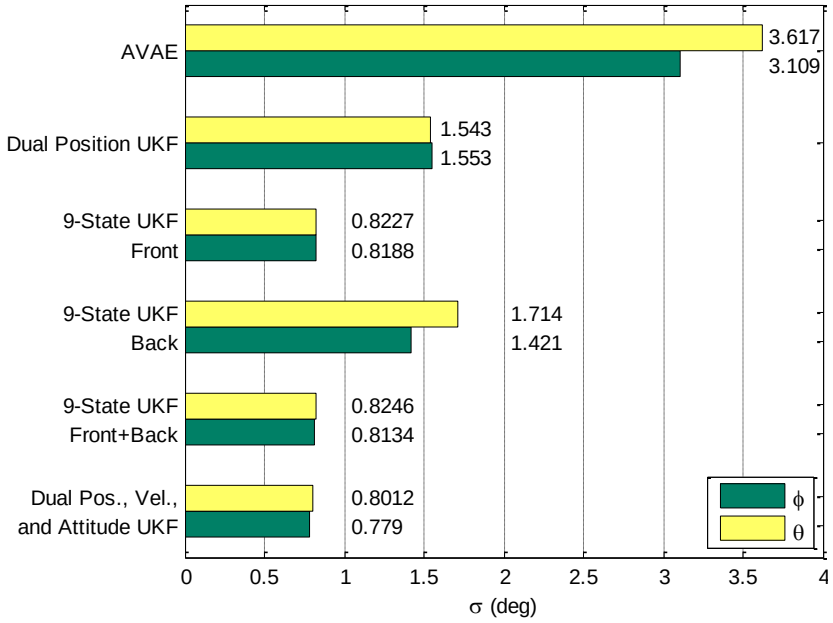

Figure 9. Example of Labeled Bar Graph (Rhudy et al., 2013)

```
%% Example Code for Advanced Bar Plot (Example 1)
colormap('summer');     % Changes color scheme for bars
barh(m,1); ylim([0.5 6.5]);
set(gca,'XGrid','on');
set(gca, 'YTick', [0.85 1.15 1.85 2.15 2.85 3.15 3.85 4.15 5 6]);
set(gca, 'YTickLabel', {'and Attitude UKF','Dual Pos., Vel.,',...
    'Front+Back','9-State UKF', 'Back','9-State UKF',...
    'Front','9-State UKF','Dual Position UKF','AVAE'});
xlabel('\sigma (deg)'); legend('\phi','\theta','Location','SouthEast');
text(m(3,2)+0.05, 5.85, num2str(m(3,1),4)); text(m(3,2)+0.05, 6.15, num2str(m(3,2),4));
text(1.6, 4.85, num2str(m(7,1),4)); text(1.6, 5.15, num2str(m(7,2),4));
text(1.05, 3.85, num2str(m(8,1),4)); text(1.05, 4.15, num2str(m(8,2),4));
text(2.05, 2.85, num2str(m(9,1),4)); text(2.05, 3.15, num2str(m(9,2),4));
text(1.05, 1.85, num2str(m(10,1),4)); text(1.05, 2.15, num2str(m(10,2),4));
text(1.05, 0.85, num2str(m(11,1),4)); text(1.05, 1.15, num2str(m(11,2),4));
```

Another example is provided in Figure 10 using a similar concept, but uses a logarithmic scale for the *x*-axis since the orders of magnitude vary significantly across the different entries. Note also for this example (as well as the previous example) that only vertical grid lines are used, since the horizontal grid lines are meaningless for this case, i.e. the *y*-axis is used to label different data sets, not to numerically quantify data.

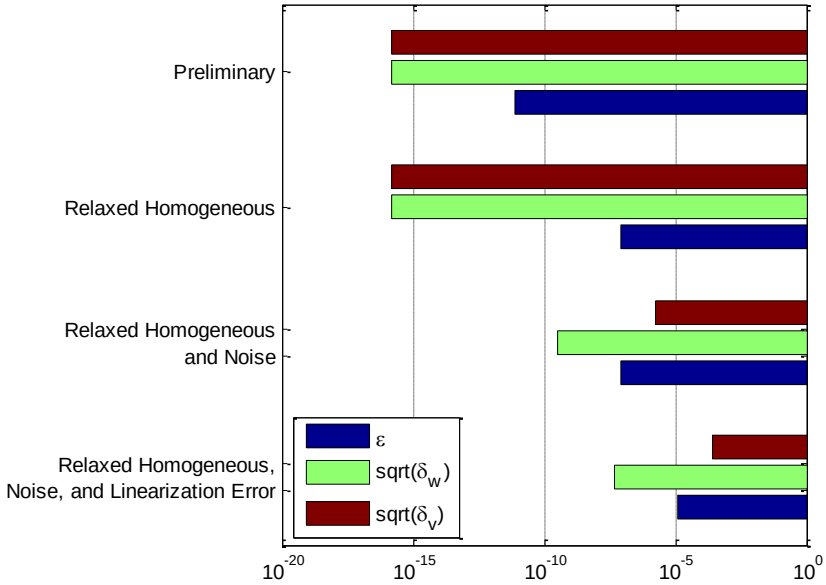

Figure 10. Example of Logarithmic Bar Graph (Rhudy, 2013)

```
%% Example Code for Advanced Bar Plot (Example 2)
barh([eps_v; sqrt(dw_v); sqrt(dv_v)]');
set(gca,'XScale','log','XGrid','on');
legend('\epsilon','sqrt(\delta_w)','sqrt(\delta_v)','Location','SouthWest');
set(gca,'YTick',[0.9,1.1,1.9,2.1,3,4]);
set(gca,'YTickLabel',fliplr({'Preliminary','Relaxed Homogeneous','Relaxed Homogeneous',...
    'and Noise','Relaxed Homogeneous,','Noise, and Linearization Error'}))
```

B. Using Color to Represent a 3rd Dimension

Visual representation in three dimensions can be very challenging for papers, because paper and computer screens are flat. Isometric views can be useful, e.g. using the *plot3*, *mesh*, or *surf* commands. These views, however, can sometimes be vague, and are difficult for certain data sets. Because of this, color can provide an alternative method for displaying a 3rd dimension. The color in the figure will represent the 3rd dimension, e.g. darker is larger, lighter is smaller. MATLAB has different color presets which can be used which vary the color differently, e.g. black to white, red to blue, etc. There are different options for creating color figures, and two of these are discussed.

The first command that is considered is *contourf*. This command generates contour curves, and fills in each region with color. Two different examples are provided using this method in Figure 11.

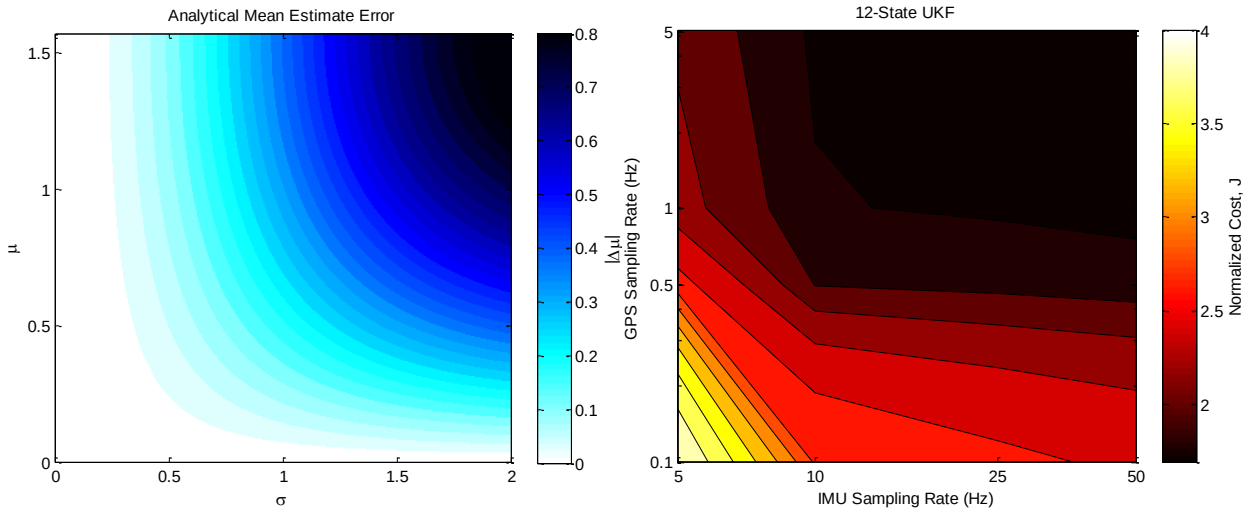

Figure 11. Examples of Color Plotting using *contourf* (Rhudy, 2013)

```
%% Example Code for Color Plotting using contourf (Example 1)
coloraxislims_mu = [0 0.8];      % Set limits for color scale
ncontours = 30;                  % Set number of contours
colormap(flipud(fliplr(hot)));  % Customize color scale by modifying MATLAB 'hot'
[hc hc] = contourf(s, muv, abs(Mme), ncontours);
set(hc, 'LineStyle', 'none');    % Remove lines the contour lines
ylabel('\mu'); xlabel('\sigma'); title('Analytical Mean Estimate Error');
caxis(coloraxislims_mu);         % Assign limits to color scale
h = colorbar; ylabel(h, '|\Delta\mu|'); % Label the colorbar
```

```
%% Example Code for Color Plotting using contourf (Example 2)
contourf(imuv,gpsv,mu12); colormap('hot');
set(gca,'xscale','log','yscale','log','XTick',imuv,'YTick',gpsv);
h = colorbar; ylabel(h,'Normalized Cost, J')
xlabel('IMU Sampling Rate (Hz)'); ylabel('GPS Sampling Rate (Hz)'); title('12-State UKF');
```

The second command that is considered is *imagesc*. This command will display a matrix of values as a grid of colors. Two different examples are provided using this method in Figure 12 and Figure 13.

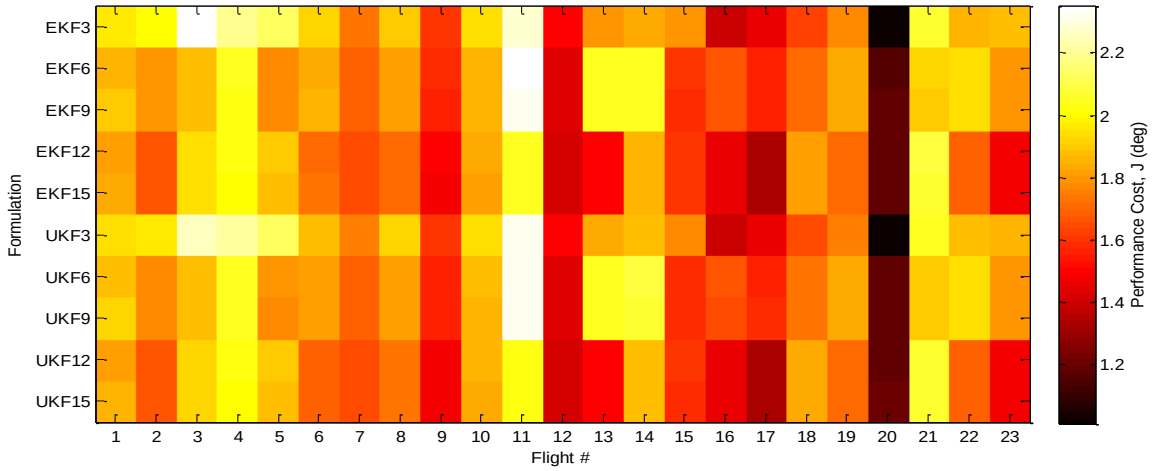

Figure 12. Example 1 of Color Plotting using *imagesc* (Rhudy, 2013)

```
%% Example Code for Color Plotting using imagesc (Example 1)
colormap('hot');
imagesc(data');
set(gca, 'XTick', 1:23);
set(gca, 'YTick', 1:6, 'YTickLabel', {'EKF3','UKF3','EKF6','UKF6','EKF12','UKF12'});
xlabel('Flight #'); ylabel('Formulation');
h = colorbar; ylabel(h, 'J (deg)');
```

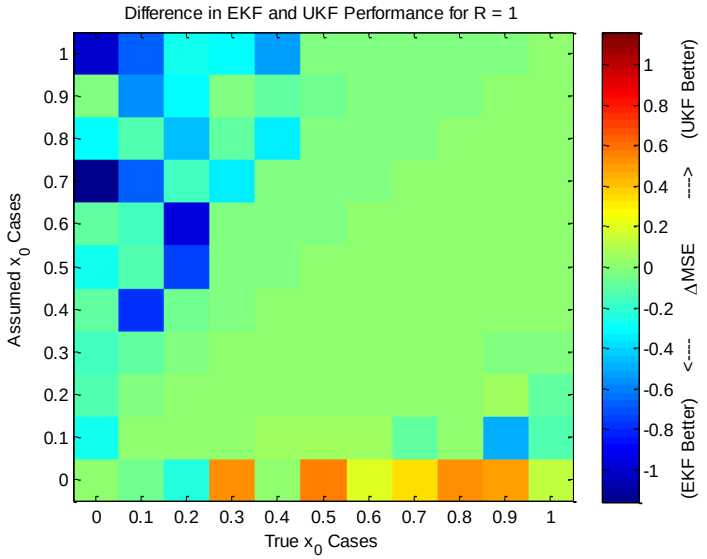

Figure 13. Example 2 of Color Plotting using *imagesc* (Rhudy, 2013)

```
%% Example Code for Color Plotting using imagesc (Example 2)
imagesc(data');
ylabel('Assumed x_0 Cases');
set(gca,'YTick',1:na,'YTickLabel',x0alab);
xlabel('True x_0 Cases');
set(gca,'XTick',1:nx,'XTickLabel',x0lab);
cabslim = max(abs(min(min(data))), max(max(data))); % Force color scale to be symmetric
set(gca,'CLim',[-cabslim cabslim]);
h = colorbar;
ylabel(h,'(EKF Better)     <----        \DeltaMSE        ---->     (UKF Better)');
title(['Difference in EKF and UKF Performance for R = ',num2str(R)]);
```

C. 2-Dimensional Distributions

For large data sets, an effective way to represent data is by providing its distribution, e.g. using histograms. This provides statistical information about the data in a visual form. If two different values are of interest with respect to one another, the coupled distribution of data (2-dimensional histogram) can be useful. As in the previous section, color can be a useful way to display a third dimension, e.g. in this case the probability density or number of data points. Two different examples are provided using this method in Figure 14. Both of these examples use a logarithm for the color scale due to the large number of points near the mean of the distribution. This step can be omitted for data sets that do not have this property. The first example sets the bin edges for the data, while the second just defines a set number of bins for each dimension. These values can be tuned to set the resolution of the figure. Be careful, if you use too many bins, unnecessary local maxima and minima might be created in the data. Note that the lines can be removed from the contours for a smoother appearance, as in Figure 14 (left).

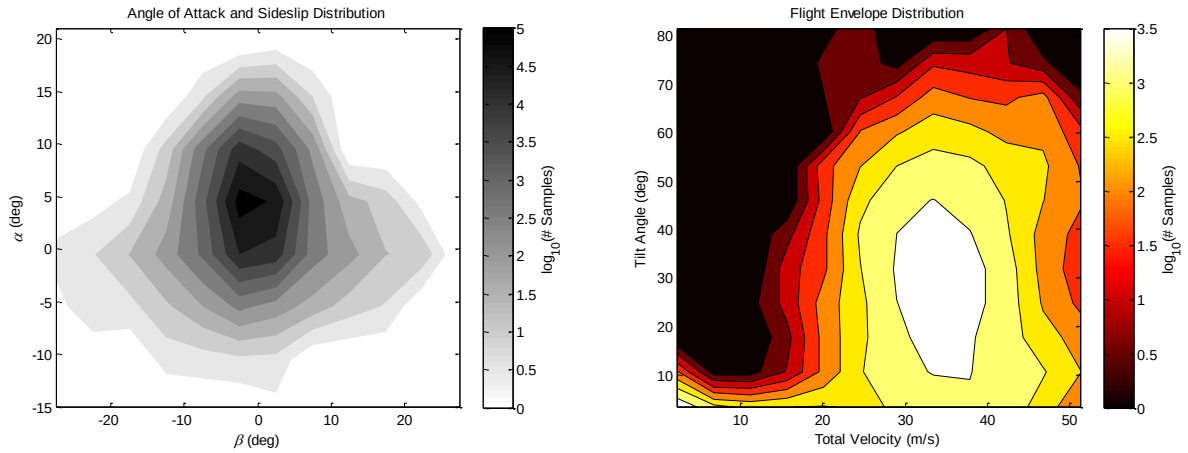

Figure 14. Examples of 2D Distribution (Rhudy, 2013)

```
%% Example Code for 2D Distribution (Example 1)
colormap(flipud('gray'));
[N, c_bins] = hist3([betav', alphav'],'Edges',{[-30:5:25],[-18:5:32]});
N = log10(N+1);
contourf(c_bins{1}, c_bins{2}, N','LineStyle','none')
xlabel('\it\beta\rm (deg)'); ylabel('\it\alpha\rm (deg)'); ylim([-15 21]);
h = colorbar; ylabel(h,'log_1_0(# Samples)')
title('Angle of Attack and Sideslip Distribution');
```

```
%% Example Code for 2D Distribution (Example 2)
colormap('hot');
[N c_bins] = hist3([v, gamma],12*[1, 1]);
N = log10(N+1);
contourf(c_bins{1}, c_bins{2}, N')
xlabel('Total Velocity (m/s)'); ylabel('Tilt Angle (deg)')
h = colorbar; ylabel(h,'log_1_0(# Samples)')
title('Flight Envelope Distribution');
```

D. Fading Color Scale

Sometimes it is useful to display particular variables with respect to one another instead of with respect to time. For example, *x* and *y* positions can be plotted together to show a trajectory. In this case, however, the time information is not clear. In order to convey the change in time of the trajectory, a fading color scale can be used. One method to handle this is by individually plotting a number of lines, with each line containing a slightly different color. This method uses the 'Color' property and varies one or more components e.g. from 0 to 1 or vice versa in order to go through a full range of color. Two different examples are provided using this method in Figure 15.

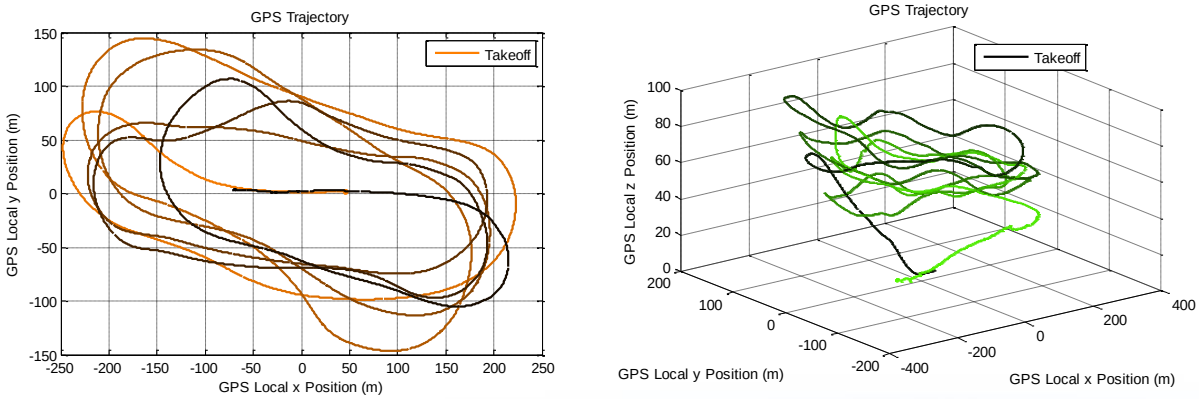

Figure 15. Examples of Fading Color Scale (Rhudy et al., 2013)

```
%% Fading Color Scale (Example 1, Orange to Black)
N = length(x);
ncolor = floor(N/100);
for i = 1:ncolor:(N-ncolor)
    ind = i:(i+ncolor-1);
    plot(x(ind),y(ind),'Color',[1-i/N (1-i/N)/2 0],'LineWidth',1.5); hold on;
end
xlabel('GPS Local x Position (m)'); ylabel('GPS Local y Position (m)');
legend('Takeoff'); title('GPS Trajectory'); grid on;
```

```
%% Fading Color Scale (Example 2, Black to Green)
N = length(x);
ncolor = floor(N/100);
for i = 1:ncolor:(N-ncolor)
    ind = i:(i+ncolor-1);
    plot3(x(ind),y(ind),z(ind),'Color',[(i/N)/3 (i/N) 0],'LineWidth',1.5); hold on;
end
xlabel('GPS Local x Position (m)'); ylabel('GPS Local y Position (m)');
zlabel('GPS Local z Position (m)'); legend('Takeoff');
title('GPS Trajectory'); grid on;
```

Another method that can be used to show variation in color is by using the MATLAB preset "colormaps", e.g. 'jet', 'hot', or 'cool' (Mathworks, 2013). This method is similar to the previous approach, in that multiple lines are drawn, each with a different color. An example is provided using this method with multiple different sized subplots. This shows how the trajectory on the left varies with respect to the other values on the right as well as time.

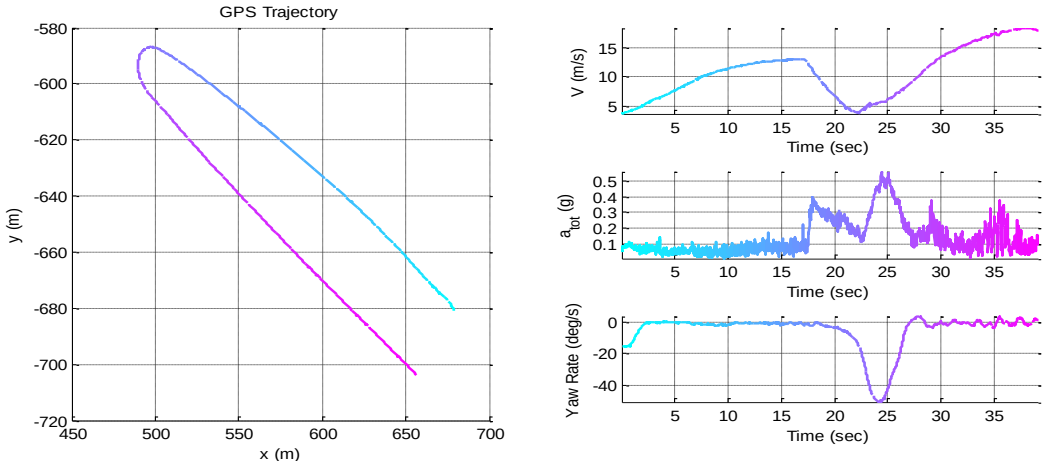

Figure 16. Example of Changing Color Scale using Preset *colormap*

```
%% Fading Color Scale using Colormap
% Define a fading color scale using MATLAB color map
colorscale = colormap('cool');
L = length(m.x);
inc = floor(L/length(colorscale))-1;
for i = 1:length(colorscale)-1
    r = ((i-1)*inc + i):(i*inc + i + 1);
    subplot(3,2,[1,3,5]); hold on;
    plot(m.x(r),m.y(r),'Color',colorscale(i,:),'Linewidth',2); grid on;
    subplot(3,2,2); hold on;
    plot(t(r), m.v_tot(r),'Color',colorscale(i,:),'Linewidth',2); grid on;
    subplot(3,2,4); hold on;
    plot(t(r), m.a_tot(r),'Color',colorscale(i,:),'Linewidth',2); grid on;
    subplot(3,2,6); hold on;
    plot(t(r), m.r(r),'Color',colorscale(i,:),'Linewidth',2); grid on;
end
% Add labels
subplot(3,2,[1,3,5]); xlabel('x (m)'); ylabel('y (m)'); title('GPS Trajectory'); hold off;
subplot(3,2,2); xlabel('Time (sec)'); ylabel('V (m/s)'); axis tight; hold off;
subplot(3,2,4); xlabel('Time (sec)'); ylabel('a_t_o_t (g)'); axis tight; hold off;
subplot(3,2,6); xlabel('Time (sec)'); ylabel('Yaw Rate (deg/s)'); axis tight; hold off;
```

E. Dual Axes

It is sometimes useful to plot values with two different scales on the *y*-axis with a single shared *x*-axis. This can be done using the *plotyy* command. An example of this is given in Figure 17. Similarly, there is a command *plotxx* that applies two scales on the *x*-axis with a single shared *y*-axis.

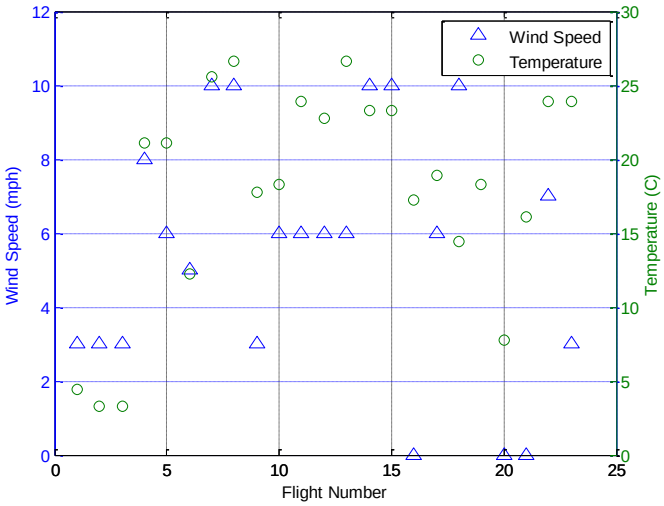

Figure 17. Example of Dual *y*-axes (Rhudy et al., 2012)

```
%% Dual y-axes Example Code
[ax, h1, h2] = plotyy(flt,wind,flt,temp);
set(get(ax(1),'Ylabel'),'String','Wind Speed (mph)');
set(get(ax(2),'Ylabel'),'String','Temperature (C)');
set(h1,'Marker','^','LineStyle','none','MarkerSize',8);
set(h2,'Marker','o','LineStyle','none');
legend('Wind Speed','Temperature'); grid on
xlabel('Flight Number');
```

CONCLUSION

This document provided a few general tips for presenting data using MATLAB figures including various intricate examples. These examples are intended to show how to create similar figures. For further information on any of the functions used, the authors recommend the use of the MATLAB help feature as well as various online forums such as MATLAB Central Answers (Mathworks, 2013).

ACKNOWLEDGEMENT

The authors would like to thank Francis J. Barchesky and Jason N. Gross for their help in deriving and inspiring this work.

REFERENCES

Mathworks, MATLAB Documentation Center, 2013, Available online: http://www.mathworks.com/help/matlab/

Mathworks, MATLAB Central Answers, 2013, Available online: http://www.mathworks.com/matlabcentral/answers/

Rhudy, M., Gu, Y., and Napolitano, M. R., "Relaxation of Initial Error and Noise Bounds for Stability of GPS/INS Attitude Estimation," *AIAA Guidance Navigation and Control Conference*, AIAA-2012-5031, Minneapolis, MN, August, 2012.

Rhudy, M., "Sensitivity and Stability Analysis of Nonlinear Kalman Filters with Application to Aircraft Attitude Estimation," Ph.D. Dissertation, West Virginia University, May 2013.

Rhudy, M., Gu, Y., and Napolitano, M. R., "Low-Cost Loosely-Coupled Dual GPS/INS for Attitude Estimation with Application to a Small UAV," submitted to *AIAA Guidance Navigation and Control Conference*, Boston, MA, 2013.

DOCUMENT HISTORY

Initial Manuscript Submission: June 07, 2013
Initial Review Completed (V1.0 Upload): June 28, 2013