Plotting

Load the housing data set from the last lecture (available here):

```
% either load the source original data: readtable('housing.
    csv')
% or the version we cleaned
load('housing_clean.mat');
data = data_no_missing;
clear data_no_missing;
```

1 Histogram

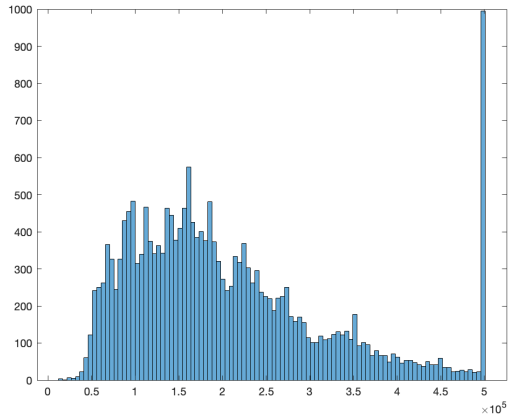

Figure 1: A Simple Histogram.

We can better understand a data set by creating histograms of its variables. In Matlab, the function `histogram` can generate histograms and has several configurable properties:

```
% create a histogram for median_house_value
histogram(data.median_house_value);
% histogram automatically determines number of bins and
    widths
% but we can define it ourselves
histogram(data.median_house_value, 100); % 100 bins
% histogram also works with categorical data
```

We can also create histograms for categorical data. We can create a categorical array from another array with the `categorical` function, which can then be used to create a histogram.

```
% convert string array to categorical data
ocean_proximity = categorical(data.ocean_proximity);
histogram(ocean_proximity);
```

2 Scatter Plot

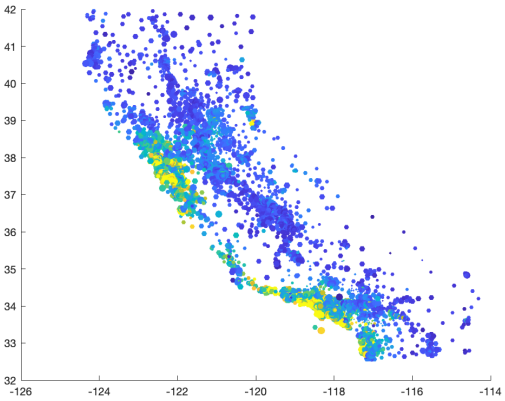

Figure 2: A Simple Scatter Plot.

To create a scatter plot we use the `scatter` function.

```
% scatter takes 2 main arguments: x and y values
scatter(data.longitude, data.latitude)
% we can change the shape, size and color of the points
% syntax: size, color, symbol
scatter(data.longitude, data.latitude, '+')
scatter(data.longitude, data.latitude, '.')
scatter(data.longitude, data.latitude, 2, '.')
scatter(data.longitude, data.latitude, 200, '.')
scatter(data.longitude, data.latitude, 2, 'black', '.')
% we can use another variable to determine the size of the
   points
scatter(data.longitude, data.latitude, data.
   median_house_value./1000, ...
        'black', 'o', 'filled');
% the filled option fills in the circles
% we can use another variable to determine the color of the
   points
scatter(data.longitude, data.latitude, 2, data.
   median_house_value./1000, '.');
colorbar; % add a colorbar to the figure
% we can do both
scatter(data.longitude, data.latitude, 10*data.
   housing_median_age, ...
        data.median_house_value./1000, '.');
```

Matlab accepts color definitions in three ways: RGB values, short names or long names. RGB values are defined as vectors with three numbers, each defining how much red, green and blue should be displayed. The short names are shortcuts for the color long names, and a full list of the color names is available here. A complete list of the properties of a figure is available here.

3 Specifying Labels, Legends and Title

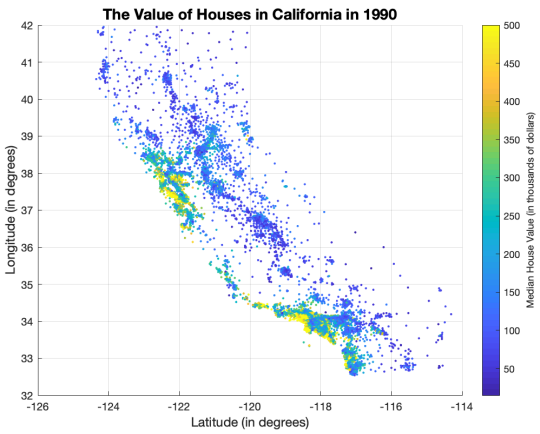

Figure 3: Specifying Labels, Legends and Title.

There are several functions to specify labels, legends and the title of a figure.

```
% create scatter plot with color varying with house value
scatter(data.longitude, data.latitude, 20, data.
   median_house_value./1000, ...
        '.');
% add labels
xlabel('Latitude (in degrees)');
ylabel('Longitude (in degrees)');
% change font size of labels
xlabel('Latitude (in degrees)', 'FontSize', 12);
ylabel('Longitude (in degrees)', 'FontSize', 12);
% add a title
title('The Value of Houses in California in 1990', 'FontSize'
   , 14);
% add a grid to the plot
grid on;
% add colorbar
cbar = colorbar;
cbar.Label.String = 'Median House Value (in thousands of
   dollars)';
cbar.Label.FontSize = 12;
```

Links to the documentation for the functions used above: `xlabel`, `ylabel`, `title`, `grid` and `colorbar`.

4 Saving a Figure

We can save a figure to a `.fig` file with the function `savefig`, which is similar to a `.mat` file for variables. This allows us to reopen the figure in Matlab with the function `openfig`.

```
% save figure to .fig file
savefig('house_value');
savefig('house_value.fig'); % equivalent
% close the figure by closing the window
% reload it
openfig('house_value.fig');
```

We can also save a figure to a file that can opened by other software, like `.png` files. This can be done with the function `print`:

```
% save figure as .png
print('house_value', '-dpng');
% save figure as .jpg
print('house_value', '-djpeg');
% save figure as .png with a resolution of 300 dots per inch
print('house_value', '-dpng', '-r300');
```

5 2-D Line Plot

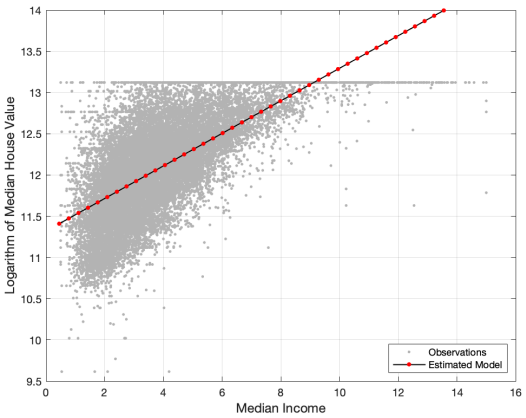

Figure 4: 2-D Line Plots.

Let's create a line plot representing the result of a linear regression using the function `plot`:

```
% regression of logarithm of median house value on median
   income
result = linreg_ols(log(data.median_house_value), data.
   median_income);
% 2D-plot of the estimated regression
% define range of values
```

```
x_min = 0.9*min(data.median_income);
x_max = 1.1*max(data.median_income);
% generate 50 points between x_min and x_max
x = linspace(x_min, x_max, 50);
y = result.b(1) + result.b(2)*x;
% actual plot
plot(x, y);
% change the line style
plot(x, y, 'LineStyle', '--');
% other values '-' (default), ':', ':.'
% mark the data points in the line with an x
plot(x, y, 'Marker', 'x');
% change marker size
plot(x, y, 'Marker', 'x', 'MarkerSize', 2);
% change the color of the line
plot(x, y, 'Marker', 'x', 'Color', 'black');
% change the width of the line
plot(x, y, 'Color', 'black', 'LineWidth', 1);
% change the color of the marker
plot(x, y, 'Marker', '.', 'MarkerEdgeColor', 'red', '
   MarkerSize', 10, ...
      'Color', 'black', 'LineWidth', 1);
```

Let's add in a scatter plot containing the original data points. To add a new plot on an already existing plot, we need to use the command `hold`. We can call `hold on` so that new plot commands are added to the same figure, and then `hold off` when we want new plot commands to generate new figures.

```
hold on;
% add a scatter plot with the original data
% save the scatter object so that we can modify the
   properties
% after the figure is generated
sobj = scatter(data.median_income, log(data.
   median_house_value));
% let's modify the scatter properties with the scatter object
    saved
% in the splot variable
sobj.Marker = '.';
sobj.MarkerEdgeColor = [0.7 0.7 0.7];
```

Notice that the points of the scatter are on top of the line. We can change this by code with the function `uistack`:

```
uistack(sobj, 'bottom');
```

Alternatively, you would need to recreate the figure and change the order of the plots (first create the scatter, then `hold on` and create the line plot).

Let's now add a grid to the figure, labels, title and a `legend`:

```
% add grid
```

```
grid on;
% add labels
xlabel('Median Income', 'FontSize', 12);
ylabel('Logarithm of Median House Value', 'FontSize', 12);
% add a legend for the line plot and for the scatter
% also move the legend location so that the line is entirely
   visible
legend(["Observations", "Estimated Model"], ...
       'Location', 'southeast');
% we can also change the limits of both axes
xlim([0, 16]);
ylim([9.5, 14]);
```

Documentation for: `xlim` and `ylim`.

6 Latex in Plots

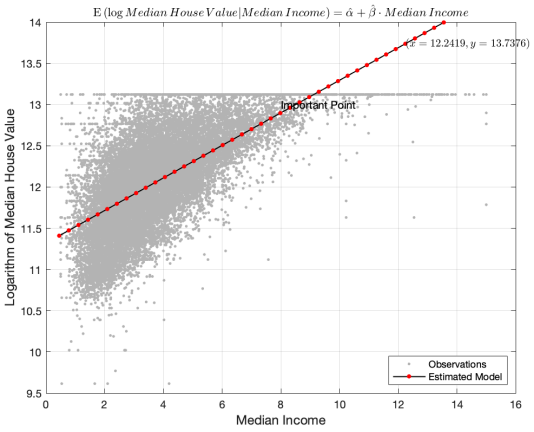

Figure 5: Latex in Plots.

Figures in Matlab also support equations created by Latex (to some extent):

```
title(['$\mathrm{E}\left(\log{Median\, House\, Value\vert
   Median\, ' ...
       'Income}\right)=\hat{\alpha}+\hat{\beta}\cdot{Median\,
           ' ...
       'Income}$'], 'Interpreter', 'latex');
```

The Latex support is convenient, but is limited to a few basic packages, and extending the number of packages is not straightforward.

We can also add text and Latex equations to the plot itself with the function `text`:

```
% specify a point in the plot to add the text to
text(8, 13, 'Important Point', 'FontSize', 10);
% latex can also be used
text(12.2419, 13.7376, '$(x=12.2419, y=13.7376)$', '
   Interpreter', 'latex');
```

7 Surface Plot

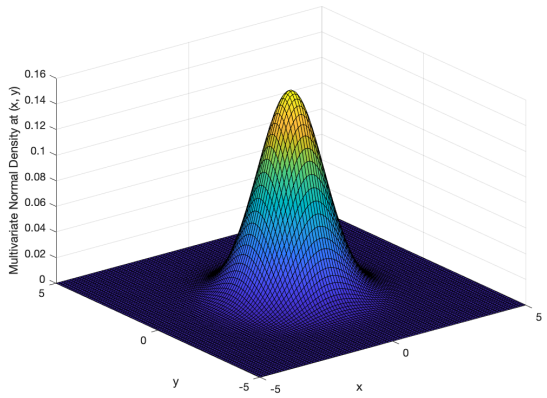

Figure 6: Surface Plot of a 2-Dimensional Normal Distribution Density.

We can create a surface plot with the function `surf`. Let's plot the density of a multivariate normal distribution using `mvnpdf`:

```
% generate points for the x-axis and for the y-axis
x = linspace(-5, 5);
y = linspace(-5, 5);
% create a "cartesian product" or a grid of the x and y points
[X, Y] = meshgrid(x, y);
% mean and variance of a 2-dimensional normal distribution
mu = [0 0];
sigma = [1, 0; 0, 1];
% obtain the density at the different points
% we need to pass to mvnpdf a 2-dimensional matrix containing the x
% and y points (x points on 1st column, y points on 2nd column)
points = [X(:), Y(:)]; % (:) stacks columns of a matrix
Z = mvnpdf(points, mu, sigma);
% reshape the z values to be a matrix
% reshape works by fixing a column and then filling each row, and
% then moving to the next column ...
z = reshape(Z, length(y), length(x));
% create the surface plot
surfplot = surf(x, y, z);
xlabel('x');
ylabel('y');
zlabel('Multivariate Normal Density at (x, y)');
```

There are a few important things to notice in the code above. First, we needed to create a 2-dimensional grid with the x and y points. We did it with the function `meshgrid`.

This function takes two vectors, say x and y, and outputs two matrices, say X and Y. The rows of X are repetitions of the vector x. The columns of Y are repetitions of the vector y. Taking elements of each matrix at the same position gives an element in the grid created by the vectors x and y:

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

$$X = \begin{bmatrix} x_1 & \cdots & x_n \\ x_1 & \cdots & x_n \\ & \vdots & \\ x_1 & \cdots & x_n \end{bmatrix}_{m \times n} \quad Y = \begin{bmatrix} y_1 & \cdots & y_1 \\ y_2 & \cdots & y_2 \\ & \vdots & \\ y_m & \cdots & y_m \end{bmatrix}_{m \times n}$$

We use the grid represented by the matrices X and Y to obtain the density of a 2-dimensional normal distribution. We pass various 2-dimensional points to the `mvnpdf` function as a matrix, which we gave the name of `points`. We create this matrix my flattening X and Y using the syntax `X(:)` and `Y(:)`. The command `X(:)` stacks all columns of X. Thus, the `points` matrix becomes:

$$\text{points} = \begin{bmatrix} x_1 & y_1 \\ x_1 & y_2 \\ \vdots & \vdots \\ x_1 & y_m \\ x_2 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_2 & y_m \\ \vdots & \vdots \end{bmatrix}$$

The output of `mvnpdf` stored in `Z` needs to be reshaped into a matrix for plotting in 3-dimensions:

$$Z = \begin{bmatrix} z_{(x_1,y_1)} \\ z_{(x_1,y_2)} \\ \vdots \\ z_{(x_1,y_m)} \\ z_{(x_2,y_1)} \\ z_{(x_2,y_2)} \\ \vdots \\ z_{(x_2,y_m)} \\ \vdots \end{bmatrix}, z = \begin{bmatrix} z_{(x_1,y_1)} & z_{(x_2,y_1)} & \cdots & z_{(x_n,y_1)} \\ z_{(x_1,y_2)} & z_{(x_2,y_2)} & \cdots & z_{(x_n,y_2)} \\ & & \vdots & \\ z_{(x_1,y_m)} & z_{(x_2,y_m)} & \cdots & z_{(x_n,y_m)} \end{bmatrix}$$

To reshape correctly, the resulting matrix `z` needs to be of dimension $m \times n$. With the matrix z and the vector x and y we can create the surface plot.

8 Plotting Functions

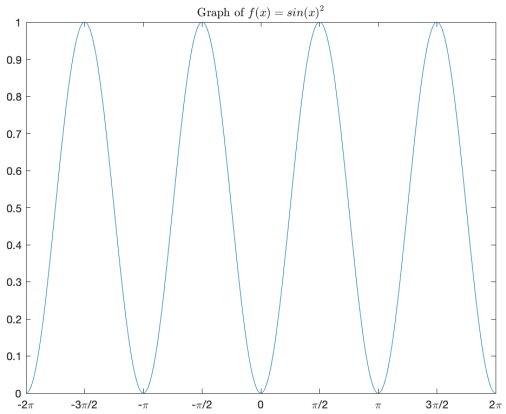

Figure 7: Plot of a Function.

We can quickly plot the graph of a function $f : \mathbb{R} \mapsto \mathbb{R}$ over some interval using the function `fplot`:

```
% create a function to plot
f = @(x) sin(x).^2;
% plot the function over some interval
fplot(f, [-2*pi, 2*pi]);
```

We can use the functions `xticks` to modify the tick values of the x-axis. The tick values represent the locations on the x-axis that have the tick marks. We can then use the function `xticklabels` to change the tick labels for the x-axis.

```
% change the tick values
xticks(-2*pi:pi/2:2*pi);
% change the labels
xticklabels(["-2\pi","-3\pi/2","-\pi","-\pi/2","0","\pi/2","\
   pi","3\pi/2","2\pi"]);
% add a title
title('Graph of $f(x) = {\sin(x)}^2$', 'Interpreter', 'latex'
   );
```

9 Subplots

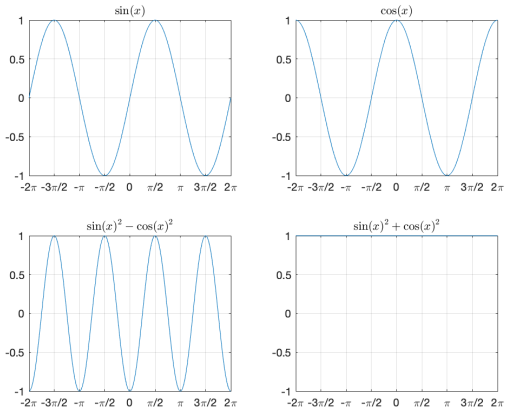

Figure 8: Figure with Multiple Subplots.

It is possible to plot multiple figures in a single figure window. We do so by specifying a grid of subplots, and then we fill each subplot by using the usual plotting commands. To specify the grid we use the function `subplot`:

```
% close the previous figure
% create a 2 by 2 grid of subplots and choose the 1st one as the
% current figure to plot on
subplot(2, 2, 1);
% we can now issue plot commands and they will add a figure to the
% first subplot on the 2x2 grid
fplot(@sin, [-2*pi, 2*pi]);
% add labels to the 1st plot
xticks(-2*pi:pi/2:2*pi);
xticklabels(["-2\pi","-3\pi/2","-\pi","-\pi/2","0","\pi/2","\pi","3\pi/2","2\pi"]);
grid on;
title('$\sin(x)$', 'Interpreter', 'latex');

% move the current figure to the 2nd subplot
subplot(2, 2, 2);
% second subplot on the 2x2 grid
fplot(@cos, [-2*pi, 2*pi]);
% add labels to the 2nd plot
xticks(-2*pi:pi/2:2*pi);
xticklabels(["-2\pi","-3\pi/2","-\pi","-\pi/2","0","\pi/2","\pi","3\pi/2","2\pi"]);
grid on;
title('$\cos(x)$', 'Interpreter', 'latex');

% move the current figure to the 3rd subplot
```

```
subplot(2, 2, 3);
% third subplot on the 2x2 grid
fplot(@(x) (sin(x).^2 - cos(x).^2), [-2*pi, 2*pi]);
% add labels to the 2nd plot
xticks(-2*pi:pi/2:2*pi);
xticklabels(["-2\pi","-3\pi/2","-\pi","-\pi/2","0","\pi/2","\
   pi","3\pi/2","2\pi"]);
grid on;
title('${\sin(x)}^2 - {\cos(x)}^2$', 'Interpreter', 'latex');

% move the current figure to the 3rd subplot
subplot(2, 2, 4);
% third subplot on the 2x2 grid
fplot(@(x) (sin(x).^2 + cos(x).^2), [-2*pi, 2*pi]);
% add labels to the 2nd plot
xticks(-2*pi:pi/2:2*pi);
xticklabels(["-2\pi","-3\pi/2","-\pi","-\pi/2","0","\pi/2","\
   pi","3\pi/2","2\pi"]);
grid on;
title('${\sin(x)}^2 + {\cos(x)}^2$', 'Interpreter', 'latex');
ylim([-1, 1]);
```

10 Default Properties for Plot

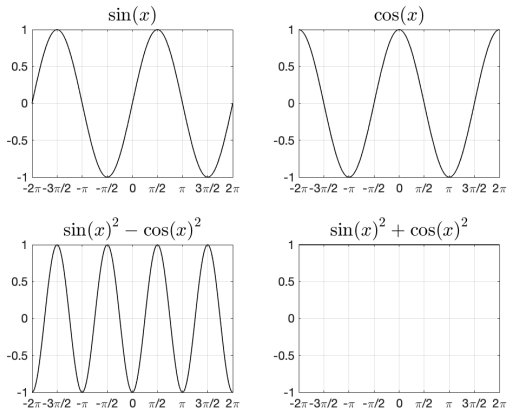

Figure 9: Figure Created After Setting Plotting Defaults.

Several of the properties we modified when plotting can be set up to have different default values, like the size of fonts used, the color and width of plot lines, and so on. It is useful to have a `.m` script with all your plotting defaults, and then have this script be run at the beginning of your main program, so that any plots you make have the desired default properties.

```
% plot_defaults.m
```

```
% Changes default values for plots.

% Use Interpreter Latex by Default
% for text
set(groot,'DefaultTextInterpreter','Latex');
% for legends
set(groot,'DefaultLegendInterpreter','Latex');

% Default color order for plots
% Each line is a color defined by an RGB triple
set(groot, 'DefaultAxesColorOrder', [0 0 0; 0.5 0 0; 0 0.5 0;
    0 0 0.5]);

% Default line style order
% If we only use one color for plots, then we can cycle
   through
% line styles
% use only black for plots
set(groot, 'DefaultAxesColorOrder', [0 0 0]);
% but cycle through many styles
set(groot, 'DefaultAxesLineStyleOrder','-|--|:|-.');

% Default line width
def_width = 0.8;
% for line plots
set(groot, 'DefaultLineLineWidth', def_width);
% for scatter plots
set(groot, 'DefaultScatterLineWidth', def_width);
% for function plots
set(groot, 'DefaultFunctionLineLineWidth', def_width);

% Default axes font size (for x-ticks and y-ticks labels)
set(groot,'DefaultAxesFontSize', 12);

% Default title font size
% The font size for the title is defined as a multiplier of
   the
% font size for the axes
set(groot,'DefaultAxesTitleFontSize', 1.6);

% Default legend font size
set(groot,'DefaultLegendFontSize', 12, ...
          'DefaultLegendFontSizeMode','manual');
```

The function `set` is used to change the properties of the `groot` object, which is keeps track of various parameters used by Matlab.

11 Building Blocks for Figures

Matlab offers very good plotting functions that can do the majority of plots we might ever need with few lines of code. However, to create more complex figures we need to understand some of the building blocks Matlab functions use to create plots.

11.1 The Figure Window

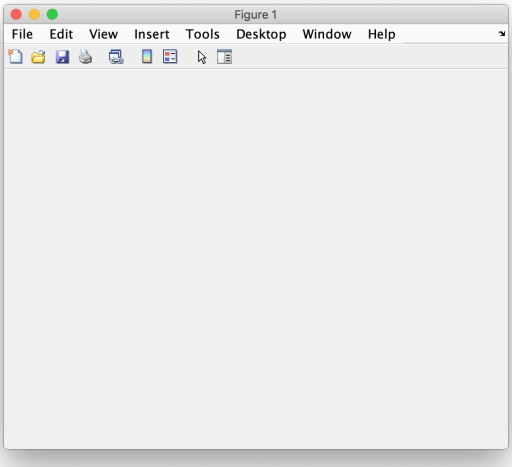

Figure 10: An Empty Figure Window.

The first thing any plotting function does is to tell Matlab to create an empty window to hold the figure. This window is called the figure window. When a figure window is created, it becomes the `current figure` and is the default place where the plotting commands generate the images on this figure. We can create this figure window with the `figure` function. And the handle to the `current figure` can be accessed via the command `gcf`.

```
figure
```

When you run the command above you should see a GUI window pop up. The window is titled "Figure 1", and it is used to hold plots. When you close this window, it will delete the figure.

When we call `figure`, it creates returns `figure object`, which we can assign to a variable. This is useful to change properties of the figure after its creation.

```
% assign figure object to variable fig
fig = figure;
% change name of the figure
fig.Name = 'Histogram of Median House Value';
% remove the figure number from the title
fig.NumberTitle = 'off';
% change color
fig.Color = 'Black';
% change color with rgb values
```

```
fig.Color = [0.9 0.9 0.9]; % rgb values
% do not display the toolbar
fig.ToolBar = 'none';
% do not display the menu bar
fig.MenuBar = 'none';
% maximize the window
fig.WindowState = 'maximized';
% these properties can be set at the time of creation
close(fig);
fig = figure('Name', 'Histogram of Median House Value', ...
             'NumberTitle', 'off', ...
             'WindowState', 'maximized');
```

The `figure object` is useful to manage several figures windows. If we have a `figure object`, we can display its window by calling the function `figure` passing a figure handle as its input.

The handle to the `figure object` can also be used to save the figure with `print`.

```
fig = figure;
fig.Color = [0.9 0.9 0.9];
print(fig, 'empty_fig', '-dpng', '-r300');
```

11.2 Creating Axes

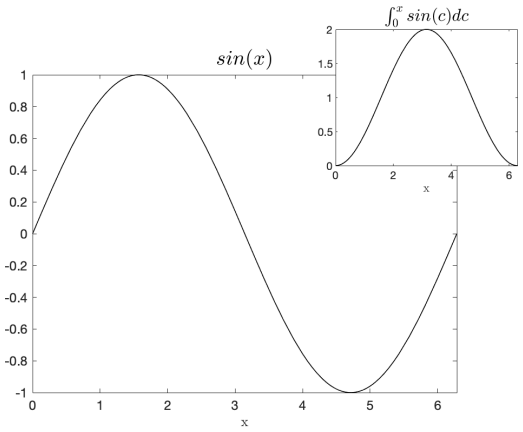

Figure 11: One Figure with Two Axes.

After creating the figure window, we can add axes to it with the function `axes`. We can then plot on these axes and modify their properties to add labels, titles and so on.

```
% create a figure window
fig = figure;
% add axes to figure
ax1 = axes(fig);
% plot something on the axes
fplot(ax1, @sin, [-2*pi, 2*pi]);
```

A single figure can have multiple axes:

```
% close the previous figure
% create a new figure window
fig = figure;

% add axes to figure specifying its position
ax1 = axes(fig, 'Position', [0.1 0.1 0.7 0.7]);
% the first 2 numbers specify the left and bottom positions of the
% axes in the figure window
% the last 2 numbers specify the width and height of the axes
% we do not use [0 0 0.7 0.7] because we need space for the tick
% labels

% add a 2nd axes that is smaller
ax2 = axes(fig, 'Position', [0.6 0.6 0.3 0.3]);

% plot on ax1
fplot(ax1, @sin, [0, 2*pi]);

% plot on ax2
fplot(ax2, @(x) (integral(@sin, 0, x)), [0, 2*pi]);
```

We can add labels and a title to each of the axes:

```
% add title to axes
ax1.Title.String = '$sin(x)$';
ax2.Title.String = '$\int_0^x sin(c)dc$';

% add labels to axes
ax1.XLabel.String = 'x';
ax2.XLabel.String = 'x';
```

11.3 Manipulating Axes

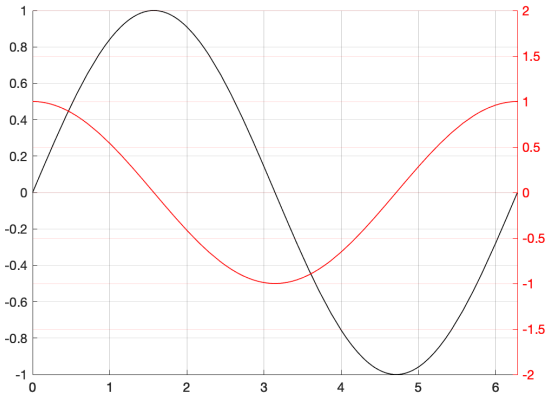

Figure 12: Manipulating Axes to Generate a Figure with Two y-Axis.

We can manipulate the axes to create complex figures. An example, is creating a figure with multiple y-axes.

```
% create figure window
fig = figure;
% add two overlapping axes
ax1 = axes(fig, 'Position', [0.1 0.1 0.8 0.8], ...
           'XLim', [0, 2*pi], 'YLim', [-1, 1]);
ax2 = axes(fig, 'Position', [0.1 0.1 0.8 0.8], ...
           'XLim', [0, 2*pi], 'YLim', [-1, 1]);
% create one plot per axes
p1 = fplot(ax1, @sin, [0, 2*pi]);
p2 = fplot(ax2, @cos, [0, 2*pi]);

% remove the color of both axes, so that both plots can be
% displayed no matter the order we add the plots
ax2.Color = 'none';
ax1.Color = 'none';
% add color to the background of the figure
fig.Color = 'white';

% both plots use a y-axis and both are on the left of the
   figure
% and are overlapping
% let's move the y-axis of the 2nd plot to the right of the
   figure
ax2.YAxisLocation = 'right';

% change the y-limits of the second axes
ax2.YLim = [-2, 2];

% notice that we have two sets of tick marks on both sides
```

```
% this happens because the ax1 marks extend to the side of ax2
% and the same happen with the ax2 marks extending to the side ax1
% we can remove these by setting the Box property to off
ax1.Box = 'off';                          % removes marks on right side
ax2.Box = 'off';                          % removes marks on left side

% we can add grids to the figure
% however, the grids use the tick marks from one of the y-axis
% we can choose which axes to use with the axes command
% choose the first axes
axes(ax1);
grid on;
% it is also possible to add another grid using the tick marks
% using the 2nd y-axis
% choose the second axes
axes(ax2);
grid on;

% change the color of the second y-axis
ax2.YColor = [1 0 0];
% change the color of  the line to match the axis
p2.Color = ax2.YColor;
```

We can use the above to create all sorts of figures. All we have to keep in mind is that a `figure window` is a blank canvas, to which we add axes. The axes can be created in various positions and moved around. Lastly, we add the actual plots to these axes.

12 Assignment

There are many other types of plots we have not covered, like the ones generated with the functions `bar` and `contour`. We will cover some other important tools for plotting in the assignment problems. However, a full documentation of the plotting facilities in Matlab is available on the 2-D and 3-D plots reference page.

Problem 1 *Use the housing data set to estimate a linear regression of the logarithm of the median house value on the logarithm of the total number of bedrooms. Given the beta estimate, compute* $\mathbb{E}[ln(\textit{Median House Value})|ln(\textit{Total Bedrooms})]$ *over a range of values for the total number of bedrooms (say from* 1 *to* 30*) and make a plot with the values.*

Problem 2 *Using the standard error estimates for* $\hat{\beta}$*, create a* 95% *confidence interval* $\mathbb{E}[ln(\textit{Median House Value})|ln(\textit{Total Bedrooms})]$*. Extend the plot from the previous problem with the confidence interval.*

Problem 3 *(Shading Confidence Intervals) Use the functions* `fill` *and* `flip` *to shade the area between the confidence interval. The function* `fill` *colors an area defined by connecting points, from start to finish. To create these points you can use the function* `flip`*.*

Problem 4 *When we call the function* `fill`*, it returns a* `Patch` *object. There are two properties of the* `Patch` *object that we can use to improve the shading of the confidence interval. The properties are the* `LineStyle` *and the* `FaceAlpha`*. Set the* `LineStyle` *property value to* `'none'`*. What does this property do? Set the* `FaceAlpha` *property value to* `0.2`*. What does this property do?*

Problem 5 *Download the following file:* `AAPL.csv`*. The file follows the .csv format and contains the stock price of the publicly traded company Apple Inc. The file has 3 columns (no headers): date, time and price. The first column contains the date of a given price in the* `YYYYMMDD` *format. For example, a date of* `20070103` *means January 3rd of 2007. The second column contains the time of a given price in the* `HHMM` *format. For example, a time of* `935` *means that the price in the 3rd column was recorded at 9:35 am. The last column contains the stock price in dollars at the given date and time.*

Load the data into Matlab. There are various functions for importing data into Matlab, choose the appropriate function.

Problem 6 *The geometric return of a stock over a time interval that starts at time* $t-1$ *and ends at time* t *is given by:*

$$r_t \equiv ln\left(\frac{P_t}{P_{t-1}}\right)$$

where P_t *is the price of the stock at time* t*.*

The stock market is open only during some hours of the day, closing at around 4 pm. The data you downloaded has prices sampled every 5-minutes of the day. We can use this data to compute intraday returns and overnight returns. Intraday returns are returns of a stock within a day. In this case, we can compute intraday returns for every 5 minutes interval. Overnight returns are the returns from the time the market closes at a given day, to the time the market opens on the next day.

Compute the intraday geometric returns from the stock prices. You should use the function `reshape` *to facilitate the computation of the intraday returns.*

Problem 7 *Create a histogram of the intraday geometric returns.*

Problem 8 *(Optional) Read about Kernel density estimation. Use the function* `ksdensity` *to estimate the density of the distribution of the intraday returns.*

Problem 9 *Create a 2D-line plot of the intraday returns. For the x-axis use a range of numbers, say* `1:length(returns)`*.*

Problem 10 *To modify the x-axis so that it correctly displays timestamps, we need to use the first two columns of the data set to obtain serial date numbers. This format is used by Matlab to display timestamps in time series plots. Use the function* `datenum` *to convert the first two columns of data set into the serial date format. Specifically, use the syntax* `datenum(Y, M, D, H, MN)` *to obtain the serial dates.*

Problem 11 *Plot the time series of intraday returns. Now, use the serial dates for the x-axis. After the* `plot` *command, you can use the function* `datetick` *to automatically change the tick labels to the appropriate value. For example, if you want to show tick labels for years, you would execute* `datetick('x', 'yyyy')`*.*

Problem 12 *We can use Matlab to generate animations from plots. Adapt this example to display an animation of the evolution of prices over time. You can do this by creating an* `animatedline` *object. Then, specify the* `axis` *limits, so that it is not updated every time we draw new points to the figure. Then,* `addpoints` *to the plot and draw them with the command* `drawnow`*. You should add the points and draw them in a for-loop. The speed of the animation is controlled by how many points you loop through.*

If you want to save this animation into a file (you do not need to do it for this exercise), you should read the Write Animated GIF example.