Analysis of single datasets

One of the most important tools in engineering is statistical analysis. Almost any experiment involves collection of measurements/data/recordings. Statistical analysis provides a good way to understand the properties of the measurements that have been collected. For example, suppose you are collecting temperature measurements in a city, where you collect a measurement every half hour. You now have a set of recordings that give you some information about that city. There are many different statistics you can compute to really understand the weather of that day.

Generic Matlab tools:

Here are some of the basic tools available in matlab (even without the statistics toolbox) which can be used for analysis of such data:

Location Statistics

These statistics can give you an overall perspective of the data. Location essentially simply means a point in the dataset that best describes the overall sample set.

Mean, median and mode:

```matlab
>> A = [ 0 8 1 6 2 1 5 2 3 ];
>> B = [ 1 7 1; 7 8 5; 1 6 7; 8 6 6];
>> mean(A)
>> mean(B)
>> mean(B,2)
>> median(B,2)
>> mode(B,2)
```

Other location stats:

```matlab
>> trimmean(B,2)
>> geomean(B,2)
>> harmmean(B,2)
```

Spread

A location statistic coupled with a spread statistic can be more useful.

1. Standard Deviation
   This shows how much variation there is from the "average" (mean). It may be thought of as the average difference of the scores from the mean of distribution,
how far they are away from the mean. A low standard deviation indicates that the data points tend to be very close to the mean, whereas high standard deviation indicates that the data are spread out over a large range of values.

2. **Variance**
   The variance is simply the square of the standard deviation.

3. **MAD**
   In statistics, the median absolute deviation (or "MAD") is a robust measure of the variability of a univariate sample of quantitative data. It can also refer to the population parameter that is estimated by the MAD calculated from a sample. For a univariate data set $X_1, X_2, \ldots, X_n$, the MAD is defined as the median of the absolute deviations from the data's median:

   ```matlab
   >> load carbig
   >> plot(Weight)
   >> mean(Weight)
   ans =
     2.9794e+03
   >> median(Weight)
   ans =
     2.8225e+03
   >> std(Weight)
   ans =
     847.0043
   >> var(Weight)
   ans =
     7.1742e+05
   >> mad(Weight)
   ans =
     719.6462
   ``

   **The statistical toolbox** has some functions that deal with Nan values.

   ```matlab
   >> plot(Horsepower)
   >> nansum(Horsepower)
   ans =
     42033
   >> nanmean(Horsepower)
   ans =
     105.0825
   >> nanvar(Horsepower)
   ans =
     1.5030e+03
   ``

   Notice that these assume a NORMAL distribution.
What is a normal distribution?

In probability theory and statistics, the normal distribution or Gaussian distribution is a continuous probability distribution that describes data that cluster around a mean or average. The graph of the associated probability density function is bell-shaped, with a peak at the mean, and is known as the Gaussian function or bell curve.

The normal distribution can be used to describe, at least approximately, any variable that tends to cluster around the mean. For example, the heights of adult males in the United States are roughly normally distributed, with a mean of about 70 in (1.8 m). Most men have a height close to the mean, though a small number of outliers have a height significantly above or below the mean. A histogram of male heights will appear similar to a bell curve, with the correspondence becoming closer if more data are used.

Visualization

Let now see what data samples from a normal distribution look like:

```matlab
>> hist(Horsepower)
>> hist(Horsepower,20)
>> plot(histc(Horsepower,[0:50:250]))
>> plot(histc(Horsepower,[0:10:250]))
>> boxplot(Horsepower)
```

Some statistics to measure for normal distributions
1. Skewness: Skewness is a measure of the asymmetry of the data around the sample mean. If skewness is negative, the data are spread out more to the left of the mean than to the right. If skewness is positive, the data are spread out more to the right. The skewness of the normal distribution (or any perfectly symmetric distribution) is zero.

2. Kurtosis: In probability theory and statistics, kurtosis (from the Greek word κυρτός, kyrtos or kurtos, meaning bulging) is a measure of the "peakedness" of the probability distribution of a real-valued random variable. Higher kurtosis means more of the variance is due to infrequent extreme deviations, as opposed to frequent modestly sized deviations.

```matlab
>> X = randn(1,100);
>> hist(X)
>> skewness(X);
>> kurtosis(X)

>> Y = X.^3;
>> hist(Y)
>> skewness(Y)
>> kurtosis(Y);
```

**Comparing Two different Datasets**

Consider the case where you have a population of 200 people. Suppose you measure all of their heights and all of their weights. You want to determine if these two measures are related in any way for this particular population. For the rest of this section, we'll call the vector containing heights \( H \) and the vector containing weights \( W \).

**Covariance:** Covariance is a measure of how much two variables change together. For two different datasets \( X \) and \( Y \), the covariance is measured as:

\[
\text{Cov} = E((X-\text{mean}(X))(Y-\text{mean}(Y))
\]

When there are \( n \) variable (or measure), \( \text{cov} \) returns an \( n \times n \) symmetric matrix with covariance values between every pair of variables.

Example:

```matlab
>> load carbig
>> X = [Acceleration,Displacement];
>> X = X./repmat(max(X),size(X,1),1);
>> cov(X)
ans =
```
Repeat with other variables...

**Correlation: corrcoeff**
This is another measure of how variables differ. If C is the covariance matrix, C = COV(X), then CORRCOEF(X) is the matrix whose (i,j)'th element is

\[ C(i,j)/\sqrt{C(i,i)C(j,j)}. \]

\[ >> x = randn(30,4); \quad \% \text{uncorrelated data} \]
\[ >> x(:,4) = x(:,1)*5; \quad \% \text{introduce correlation} \]
\[ >> [r,p] = corrcov(x); \quad \% \text{compute sample correlation and p-values} \]
\[ >> [i,j] = find(p<0.05); \quad \% \text{find significant correlations} \]
\[ >> [i,j] \quad \% \text{display their (row,col) indices} \]

**Hypothesis Testing:**

Tests: [H,P] = TTEST(X) performs a T-test of the hypothesis that the data in the vector X come from a distribution with mean zero, and returns the result of the test in H. H=0 indicates that the null hypothesis ("mean is zero") cannot be rejected at the 5% significance level. H=1 indicates that the null hypothesis can be rejected at the 5% level. The P value is the probability of observing the given result, or one more extreme, by chance if the null hypothesis is true. Small values of P cast doubt on the validity of the null hypothesis.

The data are assumed to come from a normal distribution with unknown variance.

\[ >> X = randn(1,20); \]
\[ >> Y = randn(1,20) + 5; \]
\[ >> Z = randn(1,20)*10; \]
\[ >> plot(X,'rx');hold on; plot(Y,'bx'); hold on ;plot(Z,'gx'); \]
\[ >> ttest(X) \]
\[ >> ttest(Y,0) \]
\[ >> ttest(X,Y) \]
\[ >> ttest(X,Z) \]

**Linear Models:**

**Anova:** In its simplest form ANOVA gives a statistical test of whether the means of several groups are all equal, and therefore generalizes Student's two-sample t-test to more than two groups. ANOVAs are helpful because they possess a certain advantage over a two-sample t-test. Doing multiple two-sample t-tests would result in a largely increased chance of committing a type I error. For this reason, ANOVAs are useful in comparing three or more means.

The following example is from a study of the strength of structural beams in Hogg. The vector strength measures deflections of beams in thousandths of an inch under 3,000
pounds of force. The vector alloy identifies each beam as steel ('st'), alloy 1 ('al1'), or alloy 2 ('al2'). (Although alloy is sorted in this example, grouping variables do not need to be sorted.) The null hypothesis is that steel beams are equal in strength to beams made of the two more expensive alloy.

```matlab
>> strength = [82 86 79 83 84 85 86 87 74 82 ... 
              78 75 77 79 79 78 82 79];
>> alloy = {'st','st','st','st','st','st','st','st','
             'al1','al1','al1','al1','al1','al1','...
             'al2','al2','al2','al2','al2','al2'};
>> p = anova1(strength,alloy)
p =
   1.5264e-04
```

The small p-value suggests rejection of the null hypothesis. The box plot shows that steel beams deflect more than beams made of the more expensive alloys.

**Other Types of Distributions**

**Probability Density Function**
In probability theory, a probability density function (abbreviated as pdf, or just density) of a continuous random variable is a function that describes the relative likelihood for this random variable to occur at a given point in the observation space. The probability of a random variable falling within a given set is given by the integral of its density over the set.

**Cumulative Distribution Function**
In probability theory and statistics, the cumulative distribution function (CDF), or just distribution function, completely describes the probability distribution of a real-valued random variable X. Cumulative distribution functions are also used to specify the distribution of multivariate random variables.

**Demo** : disttool

**Clustering**

*Cluster analysis*, also called *segmentation analysis* or *taxonomy analysis*, creates groups, or *clusters*, of data. Clusters are formed in such a way that objects in the same cluster are very similar and objects in different clusters are very distinct. Measures of similarity depend on the application.

**K-Means Clustering** is a partitioning method. The function kmeans partitions data into k mutually exclusive clusters, and returns the index of the cluster to which it has assigned each observation. kmeans treats each observation in your data as an object having a location in space. It finds a partition in which objects within each cluster are as close to each other as possible. Each cluster
in the partition is defined by its member objects and by its centroid, or center. The centroid for each cluster is the point to which the sum of distances from all objects in that cluster is minimized. kmeans computes cluster centroids differently for each distance measure, to minimize the sum with respect to the measure that you specify.

Example:

```matlab
>> numsamples = 100;
>> X = [randn(numsamples,2)+1; randn(numsamples,2)-1];
>> y = kMeansCluster(X, 2);
>> cidx = y(:,end);
>> plot(X(cidx==1,1),X(cidx==1,2),'r.',X(cidx==2,1),X(cidx==2,2),'b.');
```

kmeans uses an iterative algorithm that minimizes the sum of distances from each object to its cluster centroid, over all clusters. This algorithm moves objects between clusters until the sum cannot be decreased further. The result is a set of clusters that are as compact and well-separated as possible. You can control the details of the minimization using several optional input parameters to kmeans, including ones for the initial values of the cluster centroids, and for the maximum number of iterations.

**Exercise: Clustering for Classification**

Consider the task of finding groups of pixels based on color within the image rubiks-cube.jpg (downloaded from the course website).

You may want to write a script file for this so that you can repeatedly run code by changing some parameter values.

1. Download rubiks-cube and kmeans code from the course website.
2. Read in the image rubiks-cube.jpg using matlab function imread into a variable called Im.
   
   ```matlab
   Im = imread('rubiks-cube.jpg');
   ```
3. Convert it to double format:
   
   ```matlab
   Im = double(Im);
   ```
4. View the image in matlab using the command imagesc. This should show you the image of a rubik's cube.

   Matlab reads in the image as an NXMX3 array. Each NXM array is a color band: R, G and B. So you can think of each pixel being represented by 3 values. You can also think of it as each pixel represented by a vector of length 3. The objective is to group each pixel using kmeans.
5. Reshape Im into an NX3 matrix so that we can use kmeans on it:
   
   ```matlab
   X = reshape(Im,[size(Im,1)*size(Im,2),3]);
   ```
6. Plot all points using plot3.
7. Apply kmeans with 2 classes on X and save it into a variable called classes.
8. Reshape the variable classes into an NXM image:
   \[ y = \text{reshape}(\text{classes}, [\text{size}(\text{Im}, 1), \text{size}(\text{Im}, 2)])]; \\
9. Display this result (using the function called image or imagesc).
   (You can view the original image, plotted points and final result using subplots.)
   \[ \text{imagesc}(y); \]
10. Repeat by increasing the number of classes to 3, 4, 5, and 6.
    Plot the original image (im), 3d plot and classification (y) for all these values.
    Why does this run into an error sometimes?
11. Now generate a grayscale version of the same image. (Remember how to do it
    from the last assignment). Save this in a variable called grayIm.
12. View this Image.
13. Reshape this into a NX1 vector Xgray.
14. Plot these points. (Since it is 1D, you can use the function plot).
15. Apply kmeans with 2 classes on this vector Xgray and store your result in a
    variable called classesgray.
16. Reshape this back to the size of the gray image.
17. Display the gray image (grayIm), plotted points and classification.
18. Repeat for 3, 4, 5 and 6 classes.
19. How are the results different from the results on the color image? Why do you
    think they are different?

Please turn in the scripts. Responses to questions 10 and 19 can be in the text of the email
or in a separate text file.
Lecture 5  
Part 2: Graphics and Visualization in Matlab

We’ve already seen some functions used to visualize data in Matlab. In this lecture we will rapidly survey many of the tools Matlab provides for visualizing data.

2D Plotting

We’ve already used the `plot` function in several previous examples and seen the various parameters it takes. This function plots x and y data on linear scales along the horizontal and vertical axes, respectively. However there are other plotting styles that are often of use in practice. The following functions mostly use the same syntax styles as the basic `plot` function.

For instance, we may wish to plot data using logarithmic scales.

```matlab
>>x = logspace(-1,2);
>>loglog(x,exp(x),'-s')
>>xlabel('x')
>>ylabel('exp(x)')
>>grid on
```

Perhaps we only wish to plot data using one linear scale and one logarithmic scale.

```matlab
>>x = 0:.1:10;
>>semilogy(x,10.^x)
>>figure(2)
>>semilogx(10.^x,x)
>>grid minor
```

Or perhaps we wish to display two different data sets on the same plot but using different axes for each data set.

```matlab
>>x=0:.1:10;
>>y=0:.1:10;
>>figure(3)
>>plotyy(x,1+y,x,10.^y,'plot','semilogy')
>>plotyy(x,1+y,x,10.^y,@plot)
```

Another useful trick is displaying multiple plots, each within their own subwindow, within a figure window. To do this we use the command `subplot`. `Subplot(m,n,p)` breaks the figure window into an m by n matrix of windows and sets the current axes to the pth window. Windows are counted along rows.

```matlab
>> income = [3.2 4.1 5.0 5.6];
>>outgo = [2.5 4.0 3.35 4.9];
>>subplot(2,1,1); plot(income)
>>title('Income')
```
You can also create asymmetrical subplots by specifying a vector of windows to combine.

```matlab
>>subplot(2,2,[1 3])
>>subplot(2,2,2)
>>subplot(2,2,4)
```

### 3D Plotting

In addition to 2D plotting tools, Matlab also offers a variety of tools for visualizing 3D data. We’ve briefly seen plot3 and mesh in previous examples, but we will review them again.

**Plot3** uses the same general syntax as `plot` but requires that all 3 X,Y and Z values be specified. This function is useful for plotting 3D curves or scattered data, but is of limited use for plotting more general 2D or 3D data sets.

```matlab
>> t = 0:pi/50:10*pi;
>>plot3(sin(t),cos(t),t)
>>grid on
>>axis square
```

**Surf** and **Mesh** are functions for rendering surfaces in Matlab. We’ve seen `mesh` before while discussing interpolation. **Surf** is similar but renders a filled surface rather than a wire frame model. In both cases, if no color is specified, the height of the surface is used as a color. Both functions have the same calling parameters.

```matlab
>> [X,Y,Z] = peaks(30);
>> surf(Z);
>>colormap hsv
>>axis([-3 3 -3 3 -10 5])
>>mesh(X,Y,Z)
>>colormap jet
>>axis([-3 3 -3 3 -10 5])

>>k = 5;
>>n = 2^k-1;
>>[x,y,z] = sphere(n);
>>c = hadamard(2^k);
>>surf(x,y,z,c);
>>colormap([1 1 0; 0 1 1])
>>axis equal
```

There are additional variants of the **surf** and **mesh** command that are of use.

```matlab
>>surf(X,Y,Z)
>>meshc(X,Y,Z)
```
A “Brief” Aside on Lighting, Shading and the Matlab Camera
Matlab provides several commands to control shading, lighting and camera viewpoints when rendering 3D plots and volumetric data. These controls tend to mimic OpenGL style structures and behaviors. Since a full overview of the algorithms and reasons for these commands is beyond the scope of this course, we will briefly discuss the how and what of using these functions while skipping most of the why.

When rendering 3D plots, we can control the position of the camera and lights, shading models, material models, and lighting models. Controlling these properties can help better illustrate certain aspects of the data or surface you wish to display.

First, the camera! We can control the camera rotation, position, viewing direction and orientation as well as internal camera properties such as field of view or the projection model used.

camproj -- We can control whether the 3D camera uses a perspective (default) or orthographic projection model. A perspective projection model accounts for effects such as foreshortening and is generally a more realistic model. This model will account for behaviors such as parallel railroad tracks meeting at the horizon. Also the size of an object on the screen will depend on the distance of the object from the camera. Orthographic projections project points straight onto the viewing screen without foreshortening effects. The distance between 3D points will be preserved as will parallel lines. The best example of orthographic projection is to think of CAD tools with side, top and front views.

campos – This command allows you to set or query(get) the current camera position. Campos without any arguments will return the current camera position. The following example renders the peak example with the camera panning along the x axis.

```matlab
>>surf(peaks)
>>axis vis3d off
>>for x = -200:5:200
   >> campos([x,5,10])
   >> drawnow
>>end
```

camlookat – This command moves the camera to look at different objects which have been rendered in the current axes. It takes either a handle to an object in the current axes or an axes handle. The following example demonstrates moving the camera to look at the whole screen and then each of 3 separate spheres.  

**Note:** This is the first time we’ve mentioned object handles, but it is not the first time we have discussed handles. Object handles are similar to function handles in nature, but rather than providing access to (or pointing to) a function, they point to some ‘object’ such as a line or a surface. The concept of handles plays a key role in designing GUI’s
and will be discussed in more depth in the next lecture.

```matlab
>> [x y z] = sphere;
>> s1 = surf(x,y,z);
>> hold on
>> s2 = surf(x+3,y,z+3);
>> s3 = surf(x,y,z+6);
>> daspect([1 1 1]) % Sets the unit scale of each axis
>> view(30,10) % Sets the viewing angle of the camera
>> camproj perspective
>> camlookat(gca) % Compose the scene around the current axes
>> pause(2)
>> camlookat(s1) % Compose the scene around sphere s1
>> pause(2)
>> camlookat(s2) % Compose the scene around sphere s2
>> pause(2)
>> camlookat(s3) % Compose the scene around sphere s3
>> pause(2)
>> camlookat(gca)
```

camtarg**et** – This command sets or gets the camera target within the current axes. The target is the position in the axes at which camera is pointed.

```matlab
>> surf(peaks);
>> axis vis3d
>> xp = linspace(-150,40,50);
>> xt = linspace(25,50,50);
>> for i=1:50
>> campos([xp(i),25,5]);
>> camtarget([xt(i),30,0])
>> drawnow, pause(.1)
>> end
```

camva – This command either returns the current viewing angle or sets the viewing angle to some specified value in degrees.

camup – This command either returns the current up direction of the camera or sets it to point towards some point in the current axis, e.g., camup([0,0,1]) will set the top of the camera to point towards the point [0,0,1].

camroll – This command rotates the camera by a specified number of degrees around the current viewing axis.
camdolly—This command moves both the camera target and the camera position by the specified amounts along each axis.

```matlab
>>surf(peaks)
>>axis vis3d
>>t = 0:pi/20:2*pi;
>>dx = sin(t)./40;
>>dy = cos(t)./40;
>>for i = 1:length(t);
    >> camdolly(dx(i),dy(i),0)
    >> drawnow
>>end
```

Other commands in a similar vein to those here are **camzoom, campan, camorbit**. Read the help files on these commands.

Now the Lights and the Materials!

We can specify different shading and lighting models as well as how light is reflected from an object (the material properties)

**light** – This command will create a light with specified properties such as the color, the position (in axes coordinates), the strength of the light and the type of light. Another useful command for specifying a light is **lightangle** which specifies lights in spherical coordinates.

```matlab
>>h = surf(peaks);
>>set(h,'FaceLighting','phong','FaceColor','interp',
    'AmbientStrength',0.5)
>>light('Position',[1 0 0],'Style','infinite');
```

**lighting**—This command selects the lighting algorithm used to determine the effects of a light on a surface. The options are flat, gouraud, phong and none.

**camlight**—This command allows for the position of light objects within the camera coordinate system. It is essentially the same as the light command except for the coordinate system used.

```matlab
>>surf(peaks)
>>axis vis3d
>>h = camlight('left');
>>for i = 1:20;
    >>camorbit(10,0)
    >>camlight(h,'left')
```
>>drawnow;
>>end

shading—This command controls the shading properties for how colors are computed on 3D objects.

material—This command controls the reflectance properties of a surface or 3D object. It controls specular, ambient and diffuse reflectance strengths of the object. It can also control the specular color and the size of the specular highlight.

surfl—This command renders surfaces in the same way as the surf command, except it uses lighting and material properties we’ve just discussed.

Volumetric Visualization
In addition to plotting 3D points, curves and surfaces, Matlab provides functionality for rendering 3D volumetric data such as fluid or wind flows and MRI or CT data. We will demonstrate many of these functions using examples from the Matlab Help files.

One method for visualizing 3D volumes is to plot an isosurface from the data. An isosurface is the 3D analogue of a 2D isocontour which is a curve generated by all the points for which a function takes on a constant value. e.g., all the points (x,y) with f(x,y)=c. As a real world example, one can think of a relief map showing terrain heights.

The following examples show different methods of rendering scalar volumetric data.

Isosurface Example
[x,y,z,v] = flow;
P = patch(isosurface(x,y,z,v,-3));
isonormals(x,y,z,v,P)
set(P,'FaceColor','red','EdgeColor','none');
daspect([1 1 1])
view(3); axis tight
camlight
lighting gouraud

Isocaps Example
load mri
D = squeeze(D);
D(:,1:60,:) = [];
P1 = patch(isosurface(D, 5),'FaceColor','red',
'EdgeColor','none');
P2 = patch(isocaps(D, 5),'FaceColor','interp',
'EdgeColor','none');
view(3); axis tight; daspect([1 1,.4])
colormap(gray(100))
camlight left; camlight; lighting gouraud
isocolors(D,p1)

Note: You can have more control over the colors used to render your isosurfaces using the command isocolors.

Contourslice Example

\[
[x\ y\ z\ v] = \text{flow};
\]

\[
h = \text{contourslice}(x,y,z,v,[1:9],[],[],\text{linspace(-8,2,10))};
\]

\[
axis([0,10,-3,3,-3,3]); \text{daspect([1,1,1])}
\]

\[
camva(24); \text{camproj perspective;}
\]

\[
campos([-3,-15,5])
\]

\[
set(gcf,'Color',[.5,.5,.5],'Renderer','zbuffer')
\]

\[
set(gca,'Color','black','XColor','white', ...
'YColor','white','ZColor','white')
\]

\[
box on
\]

If you only wish to render some portion of the volume data you can extract this using the subvolume command.

It is also possible to render planar slices of a volume.

Example

\[
[x,y,z] = \text{meshgrid}(-2:.2:2,-2:.25:2,-2:.16:2);
\]

\[
v = x.*\exp(-x.^2-y.^2-z.^2);
\]

\[
xslice = [-1.2,.8,2]; yslice = 2; zslice = [-2,0];
\]

\[
slice(x,y,z,v,xslice,yslice,zslice)
\]

\[
colormap hsv
\]

Specialized Plots (Bar, Pie, and more)

Matlab also offers a variety of more specialized plotting tools to display data. We will rapidly review these various commands for creating such plots. Use the helpwin command for more details regarding these functions.

- bar, barh
- bar3, bar3h
- area
- pie
- pie3
- hist
- rose
- silhouette
- stem
- stem3
- stairs
- compass
- feather
- quiver
- quiver3
- coneplot
- contour
- contour3
- contourf
- fill
- fill3
- stream2
- stream3
- streamline
- streamtube
- streamribbon
- streamparticles