STATISTICS USING MATLAB

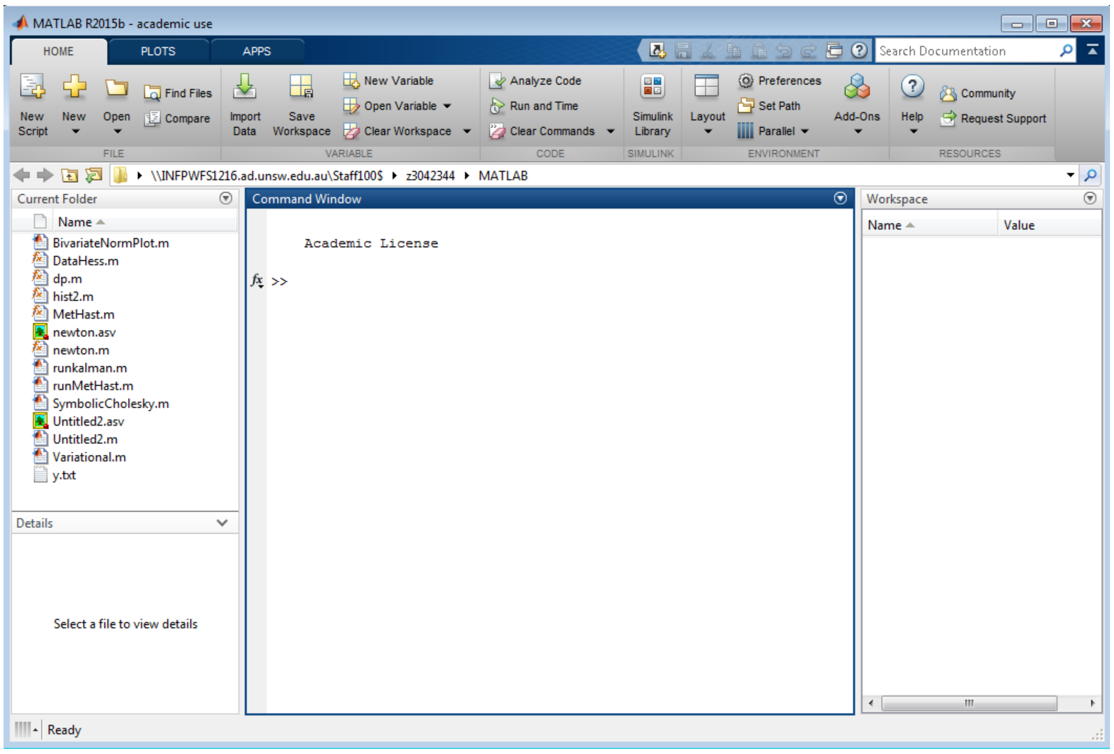
A General Guide

Contents

1. Getting Started

1.1. The MATLAB Statistics Toolbox

The MATLAB statistics toolbox will be used extensively in your computer lab sessions. Note that MATLAB is available on all PC's within the School of Mathematics and Statistics computing labs, and the statistics toolbox is also loaded on to all of these machines. The discussion below should help you get started using MATLAB. It's also a good idea to try some of the demos. When you start MATLAB you will get a window such as the one below:



```
Academic License
>>
```

You type all commands at the "»" prompt. Some important commands to remember are:

- `help` – will return help information on the function you specify. Suppose the function we want help on is the `mean` function. Then the help information is obtained by typing `help mean`:

```
MEAN   Average or mean value.
    For vectors, MEAN(X) is the mean value of the elements in X. For
    matrices, MEAN(X) is a row vector containing the mean value of
    each column.  For N-D arrays, MEAN(X) is the mean value of the
    elements along the first non-singleton dimension of X.

    MEAN(X,DIM) takes the mean along the dimension DIM of X.

    Example: If X = [0 1 2
                     3 4 5]

    then mean(X,1) is [1.5 2.5 3.5] and mean(X,2) is [1
                                                      4]
    Class support for input X:
       float: double, single

    See also median, std, min, max, var, cov.


    Reference page in Help browser
       doc mean
```

- `quit` – lets you quit MATLAB

- `who` – gives you a listing of all objects in your workspace. By objects we mean MATLAB vectors or matrices that store data or the results of calculations. This list can easily grow quite quickly in a MATLAB session, so make sure you delete any variables you don't need. To delete a variable from your workspace you use the `clear` command (type `help clear` for further details).

- `save` – lets you save MATLAB variables in a file – you can restore these variables in your MATLAB session using the `load` command. Use the on-line help to learn more about the syntax of these commands. The `load` command can also be used to read in files in various formats – its use is not restricted to retrieving variables you have previously saved in MATLAB with the `save` command.

MATLAB has hundreds of functions, and when you are beginning to learn MATLAB you will need to use the on-line help extensively. MATLAB may be a bit frustrating in the beginning, but you'll soon realise the advantages of using MATLAB for doing statistical analysis.

1.2. Data in MATLAB

Data are stored in the form of matrices in MATLAB. Suppose you have a matrix named `exampledata`. You can find its size or dimensions:

```
» size(exampledata)

ans =

     2     9
```

The output tells us that `exampledata` is a `2x9` matrix. Suppose that the first row of `exampledata` contains the numbers 1,2,3,…8,9 and that the second row contains 10,11,12,…18. To look at the third column of `exampledata` we type the following:

```
» exampledata(:,3)

ans =

     3
    12
```

Suppose we wish to look at the 2nd to 6th numbers in the first row in `exampledata`. We can type the following:

```
» exampledata(1,2:6)

ans =

      2     3     4     5     6
```

If you type `exampledata` at the prompt, it will display the entire matrix in one go. To create the matrix `exampledata` (with the numbers 1,2,3,…8,9 in the first row, and 10,11,12,…18 in the second) we could type either of the two commands below:

```
» exampledata = [1 2 3 4 5 6 7 8 9;10 11 12 13 14 15 16 17 18]
» exampledata = [1:9;10:18]
```

If we write `a:b` in MATLAB where `a` and `b` are integers, then this denotes the vector containing the consecutive integers from `a` to `b` (for instance `1:4` is the vector `[1 2 3 4]`). The semicolon ';' in the above statements indicates the end of a row.

A useful demo in MATLAB is the one on basic matrix operations – you can get it from `Help/Demos/MATLAB/Basic Matrix Operations`. Experiment with methods for manipulating matrices – you'll need to be comfortable with these methods for using MATLAB in this and some of your other courses.

1.3. Simple descriptive methods

We now discuss methods for graphically and numerically summarizing different types of data.

The example data set used to illustrate the statistical concepts that follow comes from a rainfall measuring station in Italy. The values shown are the yearly maximum hourly storm depths in mm at Genoa University, Italy, from 1931 to 1988. Data sets like this one are of great importance in any hydrologic design, since they can help us to determine the maximum rainfall that a structure or utility should be designed to sustain. The consequences of getting this maximum wrong could be disastrous. The data are given in Table 1.1.

Table 1.1. Yearly maximum hourly storm depths in mm at Genoa University, Italy

Year	**Rain**	**Year**	**Rain**	**Year**	**Rain**	**Year**	**Rain**	**Year**	**Rain**	**Year**	**Rain**
1931	38.6	*1941*	40.2	*1951*	76.2	*1961*	66.5	*1971*	50.4	*1981*	89.4
1932	33.7	*1942*	53.8	*1952*	27.4	*1962*	24.5	*1972*	43.2	*1982*	27.2
1933	33.8	*1943*	26.9	*1953*	69.4	*1963*	64.1	*1973*	39.6	*1983*	32.7
1934	79.2	*1944*	34.7	*1954*	22.8	*1964*	53.9	*1974*	38.7	*1984*	105.7
1935	58.6	*1945*	72.6	*1955*	34.8	*1965*	66.5	*1975*	40.2	*1985*	25.3
1936	39.3	*1946*	30.2	*1956*	38.8	*1966*	32.9	*1976*	55.7	*1986*	27.6
1937	33.2	*1947*	54.5	*1957*	39.8	*1967*	52.4	*1977*	118.9	*1987*	128.5
1938	29.2	*1948*	30.0	*1958*	58.1	*1968*	27.8	*1978*	25.0	*1988*	24.7
1939	46.7	*1949*	30.0	*1959*	58.1	*1969*	23.3	*1979*	55.6		
1940	80.0	*1950*	30.0	*1960*	48.5	*1970*	80.0	*1980*	40.1		

The rainfall values have been saved in a MATLAB object called `rain`, which is stored in MATLAB as a vector of length 58.

1.4. Numerical summaries of quantitative data

First we discuss methods for summarizing and graphing quantitative data.

Mean

The *mean* of a sample is what many people would think of as the "average value". It gives a measure of where the data are centred. The sample mean of values $x_1,\ldots,x_n$ is written as $\bar{x}$ and is calculated as

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

You should all be familiar with the Greek sigma notation here: $\sum_{i=1}^{n} x_i$ denotes the sum of the sample values x_i over the range of values for *i* specified (here we sum all sample values). The MATLAB function `mean` will return you the sample mean of a data set:

```
» mean(rain)

ans =

   48.4397
```

Median

The *median* is another useful measure of location (that is, of where the data are centred). The median is the middle value in the data set. The first step in calculating the median is to order the sample values. Then if the sample size is odd, there will be a unique middle value with an equal number of values positioned higher and lower than this middle value. This value is the median. For instance, if our ordered sample consisted of the numbers 1,2,3,4,5, then the middle value or median is 3. If your sample size is even, there is no unique middle value, but rather there are two middle values: in this case, the median is the mean of the middle values. For instance, if our ordered sample consisted of the numbers 1,2,3,4,5,6, then the two centre values are 3 and 4 and the median is 3.5. The median is obtained for our example data set in MATLAB as follows:

```
» median(rain)

ans =

   39.9500
```

Note that this is a lot lower than the mean – can you suggest why this might be so?

Variance

The *sample variance* is a measure of the variability of the data about its sample mean. It is defined to be:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \bar{x})^2$$

You may wonder why we use (*n*-1) instead of (*n*) in the above expression. In large samples (that is, for large *n*) it does not matter very much whether we divide by *n* or *n*-1. The reason for using *n*-1 is explained later.

The sample variance of our rainfall sample is:

```
» var(rain)

ans =

  565.0098
```

Standard Deviation

The *sample standard deviation* is the square root of the variance. The sample standard deviation has the same units as the original data values, and for our rainfall sample is computed in MATLAB as follows:

```
» sqrt(var(rain))

ans =

   23.7699
```

or

```
» std(rain)

ans =

   23.7699
```

Inter-quartile range

While the standard deviation is the most common measure of variability you will encounter, the *inter-quartile range* (*IQR*) is another useful way to quantify variability. Roughly speaking, the quartiles of a sample split the ordered sample values into four equal parts. What this means is that if you rank the sample data points from the lowest to the highest, the first quartile will have nearly a quarter of the observations below it, and the third quartile will have nearly a quarter of the observations above it. The second quartile has approximately half the sample values above it and half the sample values below it.

To be precise, the second quartile is the median, the first quartile is the median of the ordered sample values with positions strictly less than the position of the median, and the third quartile is the median of the ordered sample values with positions strictly greater than the position of the median. The *inter-quartile range* is the difference between the third and the first quartiles. The IQR for our rainfall sample is:

```
» iqr(rain)

ans =

   28.1000
```

Coefficient of Skewness

The coefficient of skewness is a measure of asymmetry of the data. It is defined to be:

$$g = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^3}{ns^3}$$

where *s* is the sample standard deviation. The coefficient of skewness is not always easy to interpret. In general, it will give an indication of departures from symmetry. A positive value of *g* may indicate a long "right tail" in the data with values above the median tending to be further away from the median than values below the median. Negative skewness may indicate a long "left tail" with values below the median tending to be more extreme. We will see what skewness means graphically when we talk about graphical summaries of quantitative data. We compute the skewness for our rainfall data set as follows:

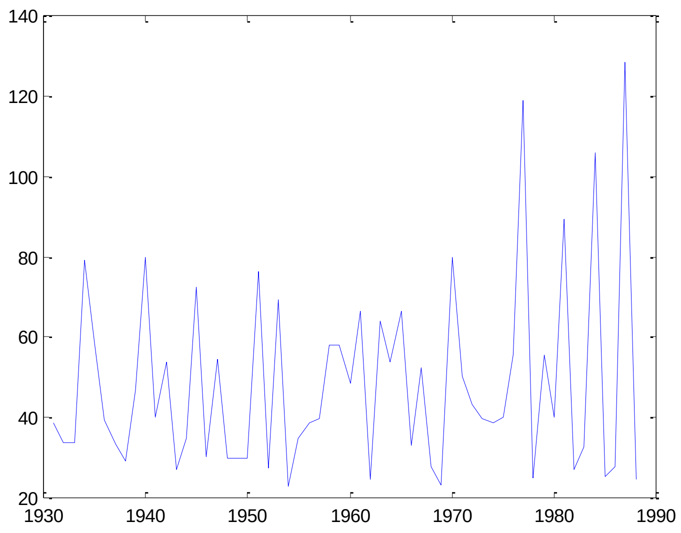
```
» skewness(rain)

ans =

    1.4267
```

1.5. Graphical summaries of quantitative data

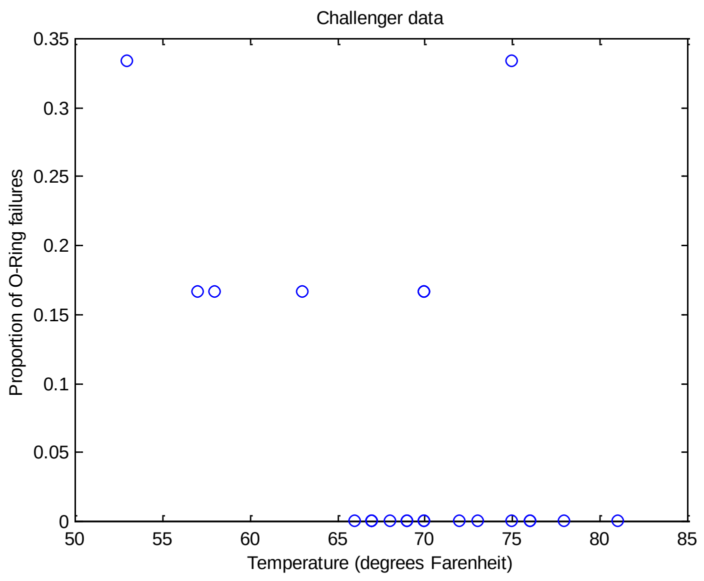
MATLAB has an extensive range of functions that allow you to visually examine data. Suppose we wish to plot the rainfall values in our data set as a *time-series*: this can be done here by typing

```
» plot(1931:1988,rain)
```

The first argument represents the x-values (years) and the second is the vector of corresponding rainfall values. The plot generated by this command is given below. Look in the help information for the `plot` function for options on adding labels, titles and so on.



As another example, consider the Challenger data set. For this data set we have temperature at take off for the 23 US space shuttle missions prior to the Challenger disaster, as well as the pressure at a pre-launch test and the number of O-Rings that failed (out of six). Below I have plotted the proportion of O-Rings failing for each mission (the number of O-Rings which fail divided by six) against the temperature at take off. A graph of two quantitative variables against each other like this is called a scatter plot. We can see here that it seems as though the risk of failure increases as the temperature decreases. The temperature at take off on the day of the Challenger disaster was 31 degrees Farenheit (well beyond the observed range of take off temperatures). This example shows that just graphically displaying carefully collected relevant data can sometimes be extremely helpful for making a decision (would you have had second thoughts about launching the Challenger if you had seen this graph?)



In MATLAB, with the proportions of failures in the vector `propfail` and the temperatures in the vector `temp`, we can generate the graph with the following commands:

```
» plot(temp,propfail,'o')
» xlabel('Temperature (degrees Farenheit)')
» ylabel('Proportion of O-Ring failures')
» title('Challenger data')
```

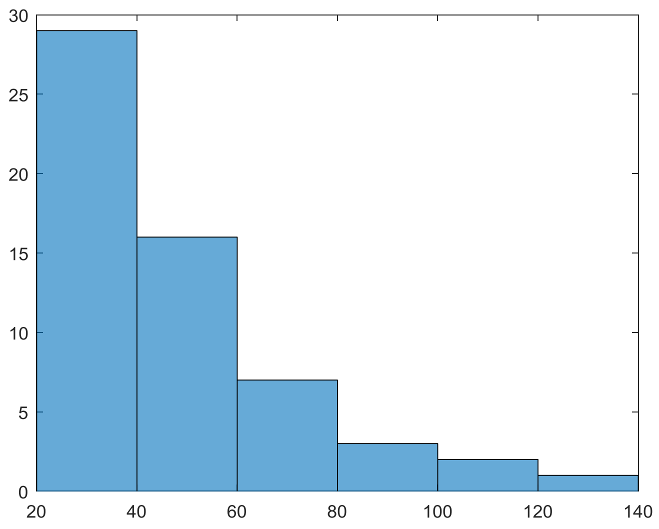
In this example, I have also illustrated how you can add labels to the *x* and *y* axes using the `xlabel` and `ylabel` commands, and a title using the `title` command. The `'o'` in the plot command tells MATLAB to use a circle as the plotting symbol in the graph (without this MATLAB will 'join the dots' with a line as in our previous example with the rainfall data).

Histogram

A *histogram* is a very convenient way of examining the variability of any data set. The sample data range is divided into a number of *bins*, and we count the number of observations falling in each bin. This gives the *frequency* of the data within the range specified by each bin. In a histogram we can also plot *relative frequencies* which are obtained by dividing the frequencies by the number of observations in the sample. The best way to understand histograms is to show an example. For our rainfall data, we obtain the histogram in MATLAB as follows:

```
» histogram(rain)
```

which results in the plot below.



The *x*-axis of the above plot represents the data values, and the *y*-axis the frequency corresponding to each bin. The width of each bin is chosen by the software using an appropriate default rule, but you can specify a different bin-width if you so desire.

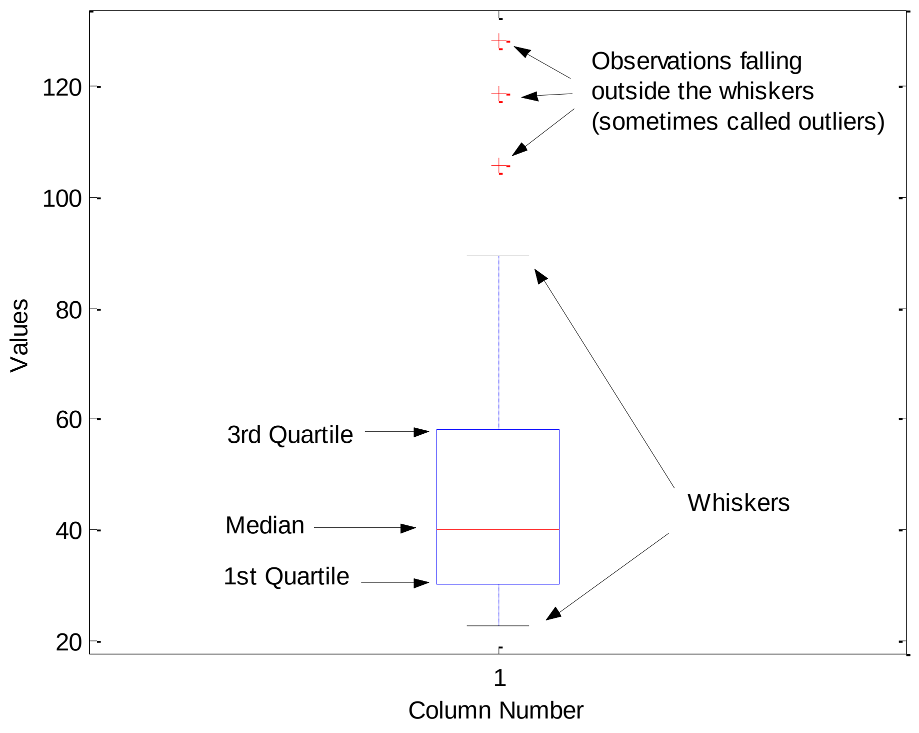
The histogram gives more information than you would get from looking at just a few summary statistics. We can get a fair idea of where the data are centred (and perhaps we can infer approximate values for the mean and median). We also get some impression of the scale of the data (a feature which is measured by summary statistics like the sample standard deviation and interquartile range) as well as whether or not the distribution exhibits skewness, multiple modes or peaks, and contains outliers. The histogram shows that the rainfall data are right skewed or right tailed with the large values in the sample further above the median than the small values are below. For a left skewed or left tailed distribution the small values in the sample are further below the median than the large values are above. The histogram suggests that the rainfall remains low most of the time (between 20-50mm each year) but does take some rather high values every now and then (maximum of nearly 130 mm).

An engineering question that might arise from this is to find a cutoff value for rainfall that would not be exceeded too frequently. This value could be used to estimate the resulting flood level in a stream that runs through that region, and zones where residences are safe to be built could be inferred. Finding this kind of cutoff value could have several other uses – for instance, it could be useful in trying to estimate the erosion that may occur on the hills the rainfall is falling on so as to apply appropriate slope-stability measures. If you were asked to decide this design rainfall cutoff value, what value would you use? Should it be the mean? That value will be exceeded quite frequently, so it will not be all that useful. Should it be the maximum value in the sample? Is that going to be safe enough for the years to come?

Boxplot

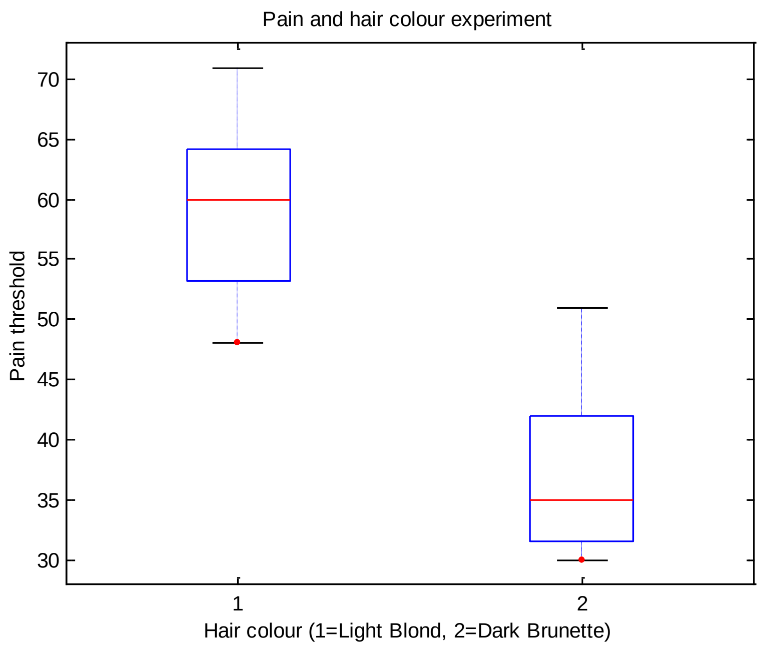
A *boxplot* is another useful way to graphically represent the location and variability of a sample. The boxplot for our rainfall data set is as follows:

```
» boxplot(rain)
```



The box in the boxplot extends to the first and third quartiles of the data (also called the *lower and upper quartiles*), the horizontal line in the middle representing the median. Two *whiskers* extend above and below the boxes to represent the variability of the data beyond the quartiles. The criteria for deciding the length of the whiskers may vary from one software package to another. All observations lying outside the whiskers are individually shown. They are sometimes called *outliers* indicating that they are unusually high or low values.

As another example, consider data on hair colour and pain threshold. In an experiment conducted at the University of Melbourne, a number of subjects were divided into groups based on hair colour, and a pain threshold score was recorded for each subject. Below are boxplots showing the pain threshold scores for the light blond and dark brunette subjects. Boxplots are very useful when we plot several at once on the same scale like this in order to compare different samples. There were five light blond and five dark brunette subjects, and I entered their pain threshold scores in MATLAB into a 2 by 5 matrix `X`, where the first column held the light blond scores and the second column held the dark brunette scores. If we then type simply `boxplot(X)` in MATLAB, this will produce the plot below with separate boxplots for each column of `X` (the matrix `X` can have more than two columns). We can add labels on the axes and a title as for scatter plots using the `xlabel`, `ylabel` and `title` commands.



2. M files

In Section 1 we have shown how to use MATLAB interactively, by typing commands into the command window. Usually it is more convenient to store sequences of commands in a file that you can save and open again when needed. These are known as M files. To get a new M file, click the first icon on the toolbar, or use File/New/M-File from the menus. You can type MATLAB commands into this file and save them to a directory of your choice. To run the commands in an M file, first change the directory to the one where your M file is stored, either using the `cd` command, for example:

```
cd p:\math2089
```

or using the `Current Directory:` box on the toolbar.
Then simply type the name of the M file (without the `.m` extension) in the command window to execute the commands.

You can add comments to an M file using %: MATLAB ignores the remainder of a line after a % symbol.

3. Reading from and writing to files

Useful functions for reading text files include `load` and `textread`.
For example, suppose the file `rain.txt` in the directory `p:\math2089` contains annual observations of rainfall and runoff at Pontelagoscuro on the Po river in northeast Italy from 1918 to 1948 (we will use these data later when we look at regression in MATLAB). The file has no header record and the first four lines are:

```
1133 904
999 648
1501 1080
807 549
:
```

To read this file into a matrix called `rain`, use:

```
load p:\math2089\rain.txt
```

To read the file into variables called `rainfall` and `runoff` use:

```
[rainfall,runoff] = textread('p:\math2089\rain.txt')
```

To deal with a header record, suppose that the file `rainfall.txt` in the directory `p:\math2089` contains the same data as `rain.txt`, but there is a header record, as follows:

```
rainfall runoff
1133 904
999 648
1501 1080
807 549
:
```

To read this file into variables called `rainfall` and `runoff` use:

```
[rainfall,runoff] =
textread('p:\math2089\rainfall.txt','%n%n','headerlines',1)
```

The function `xlsread` can be used to read Excel files, for example:

```
rainx = xlsread('p:\math2089\rainfallx.xls')
```

As an alternative to using these functions, you can use the menus: `File/Import data` and follow the instructions.

Useful functions for writing data to a file include:

- `save`: e.g. `save('filename')` stores all workspace variables in the current directory in `filename.mat`, whereas `save('filename', 'var1','var2',...)` saves only the specified workspace variables in `filename.mat`
- `csvwrite`: e.g. `csvwrite('filename',M)` writes matrix `M` to comma-separated value file `filename`
- `dlmwrite`: e.g. `dlmwrite('filename',M,'D')` writes matrix `M` to file `filename` using delimiter `D` to separate values
- `csvwrite`: e.g. `csvwrite('filename',M)` writes matrix `M` to the Excel file `filename.`

See the help files for further details.

4. Statistical distributions in MATLAB

4.1. Available distributions and calculations

There is a wide variety of statistical distributions (such as normal and binomial) available in MATLAB, and a number of different quantities (such as probabilities and random samples) that can be calculated for each distribution. MATLAB uses a standard syntax for these functions: with a prefix identifying the distribution and a suffix identifying the quantity or quantities to be calculated. For example, `binocdf` calculates cumulative probabilities for the binomial distribution.

Table 4.1 shows the distributions available in MATLAB (although we will not cover all of these distributions in your course).

Table 4.1. Statistical distributions available in MATLAB

Distribution	**MATLAB prefix**
Beta	`beta`
Binomial	`bino`
Exponential	`exp`
Extreme value	`ev`
Gamma	`gam`
Log normal	`logn`
Normal	`norm`
Negative binomial	`nbin`
Poisson	`poiss`
Rayleigh	`rayl`
Uniform	`unif`
Weibull	`wbl`

Table 4.2 shows the quantities that can be calculated for each distribution.

Table 4.2. Computations for statistical distributions in MATLAB

MATLAB prefix	**Computes**	**Example**
`pdf`	probability or probability density function	`binopdf`
`cdf`	cumulative probability or cumulative distribution function	`binocdf`
`inv`	inverse distribution function	`binoinv`
`rnd`	random sample	`binornd`
`fit*`	parameter estimates and confidence intervals	`binofit`
`stat`	Mean and variance	`binostat`

* `unifit` rather than `uniffit` for the uniform distribution

The arguments to be supplied to the functions depend on the distribution: if you are unsure, check the help files.

Some examples of the use of these functions are given in the remainder of this section.

4.2. Computation of binomial probabilities

We now illustrate how to compute binomial probabilities from the binomial distribution function or probability function in an example.

Often in doing calculations with discrete distributions you will have to compute the probability that a discrete random variable *X* lies in some interval. A common mistake that students make in this situation involves not paying sufficient attention to what happens at the end points of the interval. This is illustrated in the example below.

Example: bacteria in water samples

Water samples are taken from 20 different streams and for each sample the presence or absence of a particular bacterium in the sample is recorded. Let X be the number of samples which contain the bacterium. If X~Bin(20,0.5), compute the following probabilities:

1. $P(X\leq 10)$
2. $P(X\geq 10)$
3. $P(X<10)$
4. $P(X>10)$
5. $P(5\leq X\leq 10)$

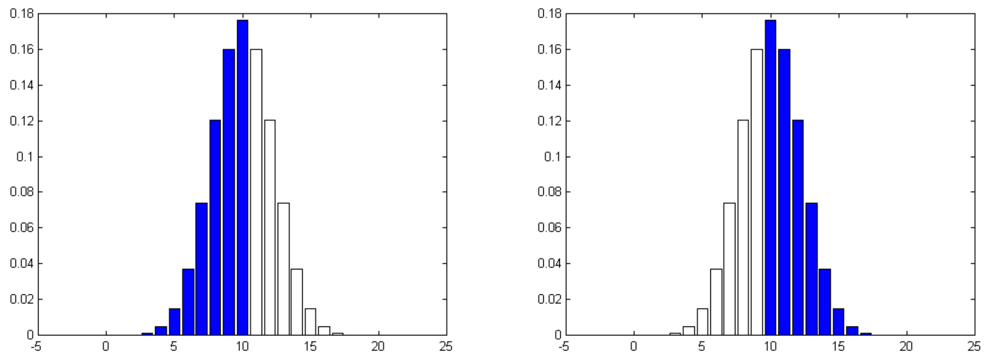


Figure 4.1: (Left) P(X<=10) is the sum of the heights of the solid bars. (Right) P(X>=10)=1-P(X<=9) (since the sum of heights of all bars is one). P(X>=10) is the sum of the heights of the solid bars, P(X<=9) is the sum of the heights of the unfilled bars.

Solution:

1. To compute $P(X\leq 10)$, we note that this is the value of the distribution function at 10, $F_X(10)$. If $p_X(x)$ denotes the probability function of X, then

$$F_X(10) = P(X \leq 10) = \sum_{x=0}^{10} P(X = x) = \sum_{x=0}^{10} p_X(x).$$

In Figure 4.1 on the left I have plotted the probability function for X, so that the distribution function value at 10, $P(X\leq 10)$ is just the sum of the heights of the solid bars. We compute a binomial cumulative distribution function value in MATLAB using the command `binocdf`. Typing `binocdf(10,20,0.5)` gives the value of the distribution function at x=10, with n=20 and p=0.5. Here the probability is 0.5881.

2. Computation of $P(X\geq 10)$ is illustrated on the right of Figure 4.1. Since the sum of the heights of all the bars in the figure is one, we have $P(X\geq 10)=1- P(X\leq 9)=1-0.4119=0.5881$ where in MATLAB we have obtained $P(X\leq 9)=0.4119$ by typing `binocdf(9,20,0.5)`.

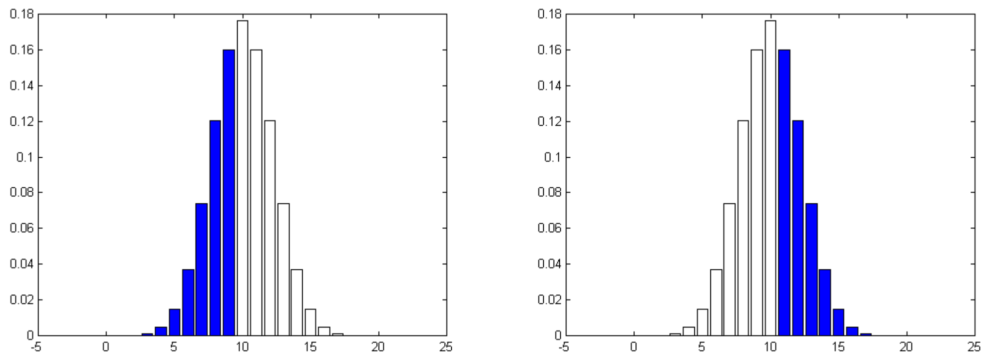


Figure 4.2: (Left) P(X<10)=P(X<=9) is the sum of the heights of the solid bars. (Right) P(X>10)=1-P(X<=10) (since the sum of the heights of all bars is one). P(X>10) is the sum of the solid bars, P(X<=10) is the sum of the heights of the unfilled bars.

3. Computation of P(X<10) is done in a similar way to the computation in 1. We note that $P(X<10)=P(X\leq 9)$ (since the largest possible value for X less than 10 is 9). Hence the required probability of less than ten samples containing the bacterium is the value of the distribution function at 9, which we have already computed as 0.4119. The left of Figure 4.2 illustrates the situation graphically.

4. To get $P(X>10)$ we note that this probability is 1-P($X\leq 10$) (one minus the value of the distribution function at 10) which is 1-0.5881=0.4119. The situation is illustrated graphically on the right of Figure 4.2.

5. Finally we are asked to compute $P(5\leq X\leq 10)$. Figure 4.3 shows what needs to be computed (the sum of the heights of the solid bars, which is the sum of the probability function values $p_X(x)$ for x=5,6,7,8,9,10). We note that this can be computed by getting the sum of the probability function values for x less than or equal to 10, and then subtracting the sum of the probability function values for x less than or equal to 4. So P($5\leq X\leq 10$)=P($X\leq 10$)-P($X\leq 4$). Using the `binocdf` command in MATLAB, we obtain $F_X(10)$=0.5881 and $F_X(4)$=0.0059, so that P($5\leq X\leq 10$)=0.5881-0.0059=0.5822.

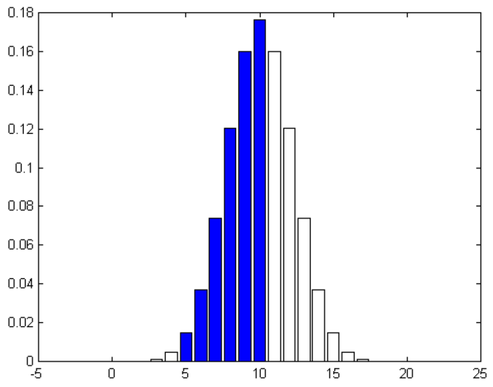


Figure 4.3: P(5<=X<=10) is the sum of the heights of the solid bars. We can compute this sum by getting the distribution function at 10 (sum of heights of bars for values less than or equal to 10) and subtracting the distribution function at 4 (sum of heights of bars for values less than or equal to 4).

4.3. Computation of Poisson probabilities

We illustrate computation of Poisson probabilities with an example.

Example: cars arriving at an intersection

Cars arrive at an intersection according to a Poisson process at a rate of 2 cars per minute. Let Y be the number of cars which arrive in a certain two minute period (so that Y~Po(4)). Compute the following probabilities:

1. $P(Y\leq 5)$
2. $P(Y\geq 5)$
3. $P(Y<5)$
4. $P(Y>5)$
5. $P(5\leq Y\leq 10)$

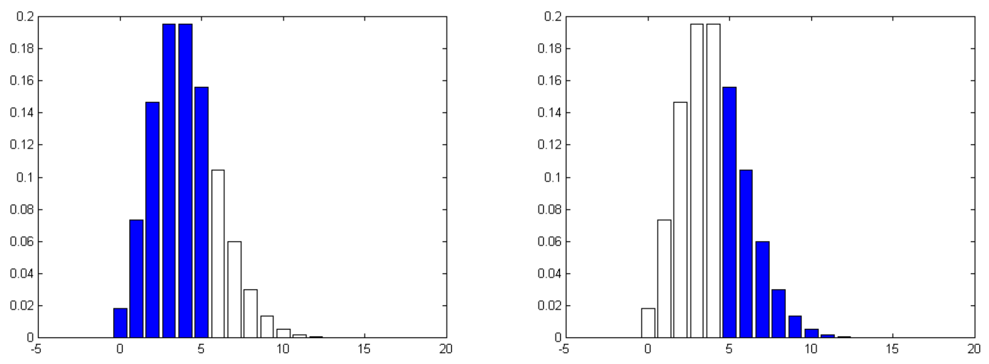


Figure 4.4: (Left) P(Y<=5) is the sum of the heights of the solid bars. (Right) P(Y>=5)=1-P(Y<=4) (since the sum of the heights of all bars is one). P(Y>=5) is the sum of the heights of the solid bars, P(Y<=4) is the sum of the heights of the unfilled bars.

Solution:

1. To compute $P(Y\leq 5)$, we note that if $p_Y(y)$ denotes the probability function of Y, then

$$F_Y(5) = P(Y \leq 5) = \sum_{y=0}^{5} P(Y = y) = \sum_{y=0}^{5} p_y(y).$$

In Figure 4.4 on the left I have plotted the probability function for Y, so that the distribution function value at 5, $P(Y\leq 5)$ is just the sum of the heights of the solid bars. Note that the plot is only over the range y=0,1,…,20, but a Poisson random variable can take on any non-negative integer value.

We compute a Poisson cumulative distribution function value in MATLAB using the command `poisscdf`. Typing `poisscdf(5,4)` gives the value of the distribution function at y=5 for λ=4. Here the probability of five or fewer arrivals is 0.7851.

2. Computation of $P(Y\geq 5)$ is illustrated on the right of Figure 4.4. Since the sum of the heights of all bars is one, and the required probability is the sum of the heights of the solid bars, the required probability is one minus the sum of the heights of the unfilled bars. But the sum of the heights of the unfilled bars is $P(Y\leq 4)$, which can be computed

in a similar way to before by typing `poisscdf(4,4)`. We get *P*(*Y*≤4)=0.6288, so that the required probability is *P*(*Y*≥5)=1- *P*(*Y*≤4)=1-0.6288=0.3712.

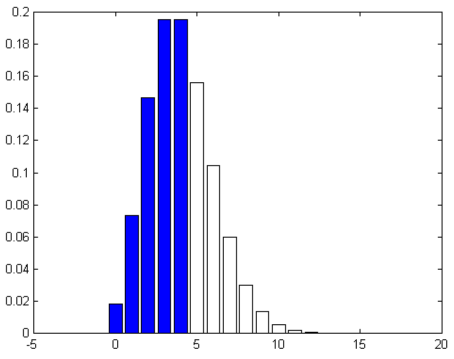
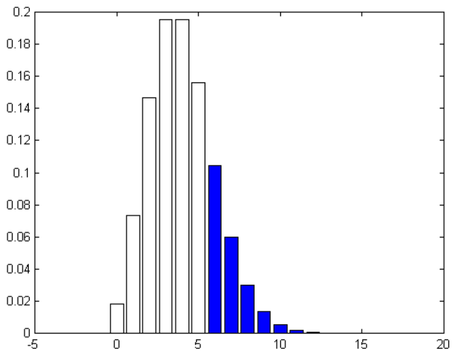



Figure 4.5: (Left) P(Y<5) =P(Y<=4) is the sum of the heights of the solid bars. (Right) P(Y>5)=1-P(Y<=5) (since the sum of the heights of all bars is one). P(Y>5) is the sum of the heights of the solid bars, P(Y<=5) is the sum of the heights of the unfilled bars.

3. Computation of P(*Y*<5) is done in a similar way to the computation in 1. We note that *P*(*Y*<5)=*P*(*Y*≤4) and hence the required probability of fewer than five cars arriving is the value of the distribution function at 4, which we have already computed as 0.6288. The left of Figure 4.5 illustrates the situation graphically.

4. To get *P*(*Y*>5) we note that this probability is 1-P(*Y*≤5) which is 1-0.7851=0.2149. The situation is illustrated graphically on the right of Figure 4.5.

5. Figure 4.6 illustrates computation of *P*(5≤*Y*≤10). The sum of the heights of the solid bars (the sum of the probability function values for y=5,6,7,8,9,10) can be obtained as the difference of the cumulative distribution function at 10 and the cumulative distribution function at 4. We obtain *P*(5≤*Y*≤10)=0.9972-0.6288=0.3683.

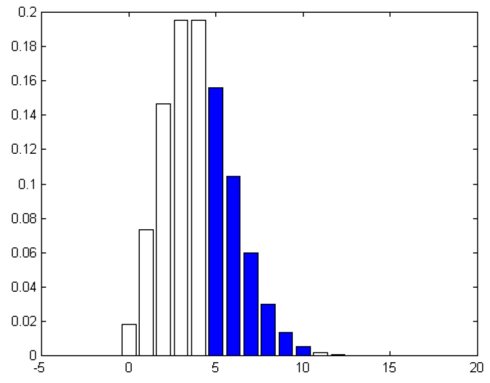


Figure 4.6: P(5<=Y<=10) is the sum of the heights of the solid bars. We can compute this sum by getting the distribution function at 10 (sum of heights of the bars for values less than or equal to 10) and subtracting the distribution function at 4 (sum of the heights of the bars for values less than or equal to 4).

4.4. Computation of normal probabilities

The best way to illustrate computation of normal probabilities in MATLAB is through an example.

Example: concrete density

The following example is based on Example 4.29 of Kottegoda and Rosso, “Statistics, Probability and Reliability for Civil and Environmental Engineers,” McGraw-Hill, 1997. The mean and standard deviation of the densities of concrete samples from a particular mix are 2445 and 16 N/mm^2. Assuming that a concrete sample X taken from this mix is normally distributed, compute the following:

1. $P(X\leq 2460)$
2. $P(X\geq 2460)$
3. $P(\mathrm{X}<2460)$
4. $P(X>2460)$
5. $P(2430\leq X\leq 2460)$

Solution:

1. To find $P(X\leq 2460)$ we simply need to compute the value of the cumulative distribution function at 2460 for a normal distribution with mean 2445 and standard deviation 16. This can be done in MATLAB by typing `normcdf(2460,2445,16)` to obtain 0.8257. The cumulative distribution function value is obtained as the integral of the density function over values less than or equal to 2460 (see Figure 4.7).

2. To find $P(X\geq 2460)$, simply observe that the complement of the event $\{X\geq 2460\}$ is $\{X<2460\}$, so that $P(X\geq 2460)=1-P(X<2460)$. How do we calculate $P(X<2460)$? The cumulative distribution function at 2460 is $P(X\leq 2460)$, which does not seem quite the same as $P(X<2460)$. However, for a continuous random variable like X, $P(X\leq 2460)=P(X<2460)$. Hence our required probability is $1-P(X\leq 2460)$, or $1-0.8257=0.1743$ (see Figure 4.7).

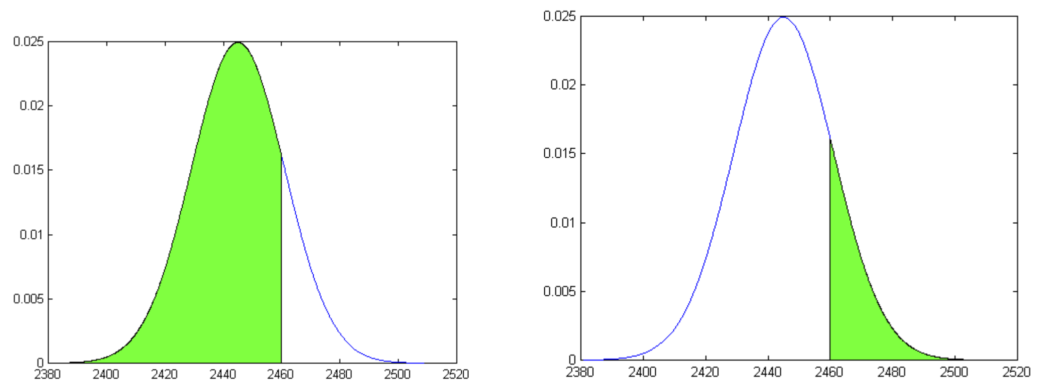


Figure 4.7: (Left) The shaded area is the value of the cumulative distribution function at 2460 (integral of the density function). (Right) The shaded area is P(X>=2460). P(X=2460)=0 since the line on the boundary of the shaded region has zero area. From this P(X>=2460)=1-P(X<2460)=1-P(X<=2460) and we can compute P(X>=2460) from the distribution function.

3. Computation of 3 is trivial, since $P(X<2460)=\mathrm{P}(\mathrm{X}\leq 2460)$ for a continuous random variable, and we have already computed this probability.

4. For a continuous random variable $P(X>2460)=P(X\geq 2460)$ and we have already computed this probability.

5. We have that $P(2430\leq X\leq 2460)=P(X\leq 2460)-P(X<2430)=P(X\leq 2460)-P(X\leq 2430)$. The computation is illustrated in Figure 4.8: the value of the distribution function at 2460 is the area under the density to the left of 2460, and the value of the distribution function at 2430 is the area under the density to the left of 2430, so that the shaded area is the difference of the two. We obtain $P(2430\leq X\leq 2460)=0.8257-0.1743=0.6514$.

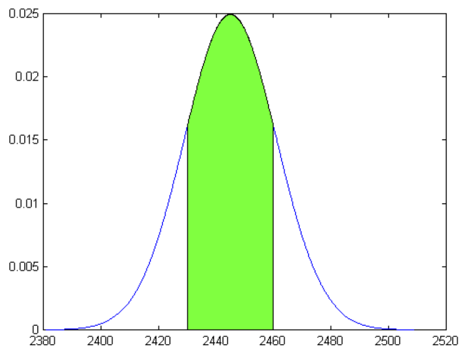


Figure 4.8: The shaded area is P(2430<=X<=2460). It can be obtained as the difference of the area under the curve to the left of 2460 and the area under the curve to the left of 2430 (that is, as a difference of distribution function values at 2460 and 2430).

4.5. Computing percentage points for the normal distribution

In computing confidence intervals for the mean, for example, it is necessary to be able to compute percentage points of a standard normal distribution. We can obtain percentage points of a standard normal distribution either from MATLAB or from a table.

Here we describe how to obtain percentage points of a standard normal distribution in MATLAB using the command `norminv`. Typing the command `norminv(p)` in MATLAB will return the value where the normal cumulative distribution function is equal to `p` (0<p<1). So if we want to obtain the upper 2.5 percentage point of the standard normal distribution, for instance, we type `norminv(0.975)`, since this will return the value which bounds an area of 0.025 in the upper tail of the standard normal density (we use 0.975 since 0.975=1-0.025). The upper 2.5 percentage point is 1.96.

4.6. Computing percentage points for the *t* distribution

As for the standard normal distribution, MATLAB can compute percentage points of the *t* distribution, enabling computation of confidence intervals, for example. The appropriate command to use in MATLAB is `tinv`. Typing the command `tinv(p,v)` in MATLAB will return the value where the distribution function of a *t* random variable with `v` degrees of freedom is equal to `p` (0<p<1). So if we want to obtain the upper 2.5 percentage point of a *t* distribution with 20 degrees of freedom, say, we type `tinv(0.975,20)`, since this will return the value which bounds an area of 0.025 in the upper tail of the *t*-density with 20 degrees of freedom.

5. Simulation in MATLAB

5.1. Introduction

We now study the important topic of *simulation*. Most statistical software packages have methods for generating samples from standard distribution functions like the binomial, Poisson and normal, and MATLAB is no exception.

Simulation is an extremely useful tool in engineering applications. It often happens in engineering applications that we can model the inputs to some system by a simple statistical model, but that the outputs of the system are a complicated function of the inputs. Working out the distribution of the outputs mathematically from the distribution of the inputs may be very difficult. However, if we can generate samples from the distributions for the inputs, we can work out the outputs of the system for these simulated inputs and by doing this repeatedly we gain some idea of what the distribution of the system outputs is like.

For instance, we might be interested in studying traffic flow at a busy intersection where the distribution of the number of cars queued at the lights is a function of the arrival times of cars at the intersection. We may be able to specify a simple stochastic model for car arrivals, but working out mathematically the distribution of queue length from this may not be simple. To consider another example, a geomatic engineer may be interested in what a model for uncertainty in a measurement implies about the uncertainty in a complicated function of the measurement.

As mentioned, statistical software packages like MATLAB have methods for simulating from standard distributions. Since you may sometimes need to simulate from distributions that are not from a standard parametric family, we give some discussion of basic methods for constructing simulation algorithms.

5.2. The uniform distribution

Given a certain distribution (for instance our model for arrival times of cars in the traffic flow example) how do we simulate from it? A simulation from any distribution can in fact be obtained from a simulation of a *uniform random variable* on the interval [0,1]. A random variable X is said to be uniformly distributed on the interval $[a,b]$ if it has the density function

$$f_X(x) = \begin{cases} (b-a)^{-1} & \text{if } x \in [a,b] \\ 0 & \text{otherwise} \end{cases}$$

If X is uniformly distributed on $[a,b]$ we write this as $X \sim U[a,b]$. A plot of the uniform density on [0,1] is shown in Figure 5.1.

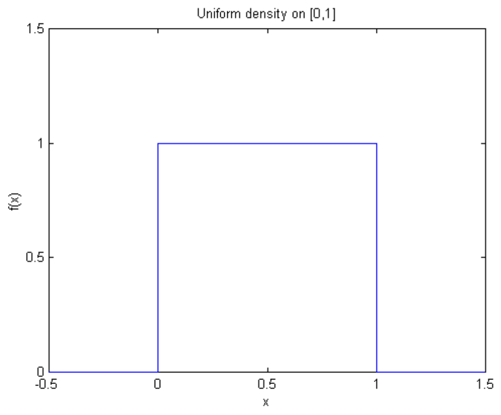


Figure 5.1: Probability density function of a uniform random variable on [0,1].

Statistical software packages like MATLAB contain algorithms for generating sequences of variables that behave like independent uniform random variables on [0,1]. The MATLAB command used for simulating from a uniform distribution is called `rand` (you could also use `unifrnd`, see Tables 4.1 and 4.2 and the help files). Typing the command `u=rand(m,n)` generates an `m` by `n` matrix `u` of simulated independent uniform random variables on [0,1].

Example: the central limit theorem in action

As an example of simulation of uniform random variables we illustrate the central limit theorem in action. Suppose we generate 1000 samples of size 50 from a uniform distribution on [0,1] by typing `u=rand(50,1000)`. Here `u` is a matrix with 50 rows and 1000 columns, and we can regard each column as being a random sample of size 50. We can compute the means of the 1000 samples (column means of `u`) and put the result in the vector `umeans` by typing `umeans=mean(u,1)`. The histogram of the sample means is shown in Figure 5.2. From the central limit theorem, we expect the distribution of the sample means to be approximately normal, and the shape of the histogram confirms that normality is a good approximation here.

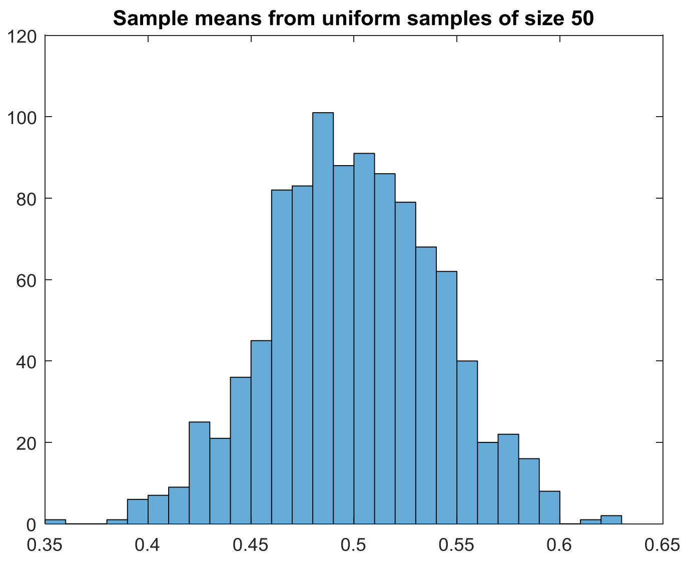


Figure 5.2: Histogram of 1000 sample means for samples of size 50 from a uniform distribution on [0,1].

5.3. Other distributions

I claimed above that we can simulate a random variable with any given distribution from a simulation of a uniform random variable. For simple discrete distributions, it is easy to see that this is the case.

Example: simulation of a Bernoulli random variable

Let X be a Bernoulli random variable with parameter p. Recall that such a random variable has a probability function defined on the values 0 and 1 with $p_X(0)=1-p$ and $p_X(1)=p$. A Bernoulli random variable is a binomial random variable with n=1. Now, suppose we have a random variable U~U[0,1]. Given U, is it possible to simulate from the Bernoulli distribution?

Consider the following algorithm. If U is less than or equal to p, set X=1, otherwise set X=0. The probability that U is less than or equal to p is the integral of the uniform density on [0,1] from 0 to p, which is p. Similarly, the probability that U is greater than p is 1-p. So our algorithm generates a random variable with the desired distribution.

From the above example, it is easy to see how we might simulate from more general discrete distributions using simulated uniform random variables on [0,1] (as an exercise, think about how you might do this).

As discussed in Section 4, MATLAB also has commands for simulating from common parametric families of discrete distributions. For instance, to simulate a matrix of independent binomial random variables with parameters n and p having r rows and c columns we type `binornd(n,p,r,c)`. Similarly, to simulate a matrix of independent Poisson random variables with mean λ and having r rows and c columns we type `poissrnd(λ,r,c)`.

What about simulation for continuous random variables? The following result is a special case of a more general result, and allows us to simulate from many continuous distributions.

Let $U \sim U[0,1]$ and suppose that $F(x)$ is a distribution function. Then if the inverse F^{-1} of F exists, $F^{-1}(U)$ is a random variable with the distribution $F(x)$.

Example: Simulation from general continuous distributions

Suppose we wish to simulate from the distribution function shown in Figure 5.3. Over the interval [0,1] this distribution function is given by $F(x)=x^2$. For negative x it is zero, and for x bigger than one it is one.

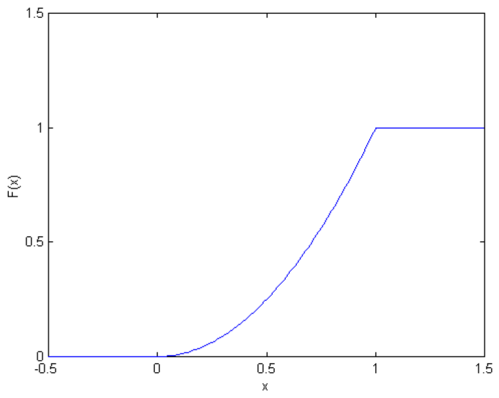


Figure 5.3: Distribution function for simulation example. $F(x)=x^2$ on the interval [0,1]. F(x)=0 for negative x, F(x)=1 for x>1.

To use the result we have stated above for simulation, we need to find the inverse of the distribution function over the range (0,1). Now, $F(x)=x^2$ on this interval. If we write $y=F(x)=x^2$, we obtain the inverse function here by solving for y. We get $x=y^{1/2}$. So we can simulate from the distribution function in Figure 11 simply by taking the square root of a uniform random variable on [0,1].

5.4. Simulation of normal and lognormal random variables

The MATLAB command used to simulate normal random variables is `normrnd`. Typing `r=normrnd(a,s,m,n)` where `a` and `s` are scalars will simulate an `m` by `n` matrix `r` of independent normal random variables with mean `a` and standard deviation `s`. There are various other ways to use the `normrnd` command (see help for details).

A distribution related to the normal is the *lognormal distribution*. A lognormal random variable Y can be represented as $Y=\exp(X)$ where X is normally distributed. In other words, $\log(Y)$ has a normal distribution. If $Y=\exp(X)$ where $X \sim N(\mu,\sigma^2)$ we write this as Y~lognormal(μ,σ^2). The lognormal distribution is often used for modelling positive measurements such as rainfall amounts or event interarrival times.

The MATLAB command used for simulating from a lognormal distribution is `lognrnd`. This command works in much the same way as the command `normrnd`. We can type `r=lognrnd(a,s,m,n)` to simulate an `m` by `n` matrix `r` of independent lognormal(`a, s`^{2}) random variables.

Example: storm rainfall

The following example is based on Kottegoda and Rosso, "Statistics, Probability and Reliability for Civil and Environmental Engineers," McGraw-Hill, 1998, Problem 8.5.

The total amount of water Z delivered by a storm in a given location is modelled as $Z=XY$ where X is the duration of the storm and Y is the rainfall rate (assumed constant for the duration of the storm). Assuming that X is lognormal(-1.447,1.805) and Y is lognormal(1.956,0.693) and that X and Y are independent, estimate the probability that the maximum amount of rainfall delivered in one hour by the storm (the hourly storm depth) is greater than 10mm based on 1000 simulations of the hourly storm depth. Also, plot histograms and boxplots showing the distribution of the hourly storm depth.

Solution:

Write W for the maximum amount of rainfall delivered in one hour by the storm. Then we have

$$W = \begin{cases} Y & \text{if } X \geq 1 \\ XY & \text{if } X < 1 \end{cases}$$

To see this, simply observe that if the duration of the storm is greater than or equal to one hour (X>1), then an amount of rainfall equal to the rate Y will be delivered in any hour of the storm. If the duration of the storm is less than one hour, then the maximum amount of rainfall that can be delivered in one hour is the total amount of rainfall, XY.

The following MATLAB commands generate a vector `w` containing 1000 independent realizations of W (the semicolons suppress screen printing).

```
>> x=lognrnd(-1.447,1.805,1000,1);
>> y=lognrnd(1.956,0.6931,1000,1);
>> b=(x>=1);
>> w=y.*b+x.*y.*(1-b);
```

The third command above `b=(x>=1)` creates a vector `b` of the same length as `x` with `b`$_i$=1 if `x`$_i$ is greater than or equal to 1 and `b`$_i$=0 otherwise. The fourth line `w=y.*b+x.*y.*(1-b)` creates a vector `w` of the same length as `b` and `x` with the ith element `w`$_i$ equal to `y`$_i$`*b`$_i$`+x`$_i$`*y`$_i$`*(1-b`$_i$`)`. Note that if `b`$_i$=1, then this expression is equal to `y`$_i$, whereas if `b`$_i$=0 it is equal to `x`$_i$`y`$_i$. Note that in general the expression `a.*b` in MATLAB where `a` and `b` are vectors of the same length evaluates to a vector of the same length as `a` and `b` with *i*th element given by a_ib_i. So the above lines of code simulate a vector `w` of length 1000 containing independent realizations of the random variable *W*. A histogram and boxplot of the values obtained when I typed the above commands in MATLAB are shown in Figure 5.4.

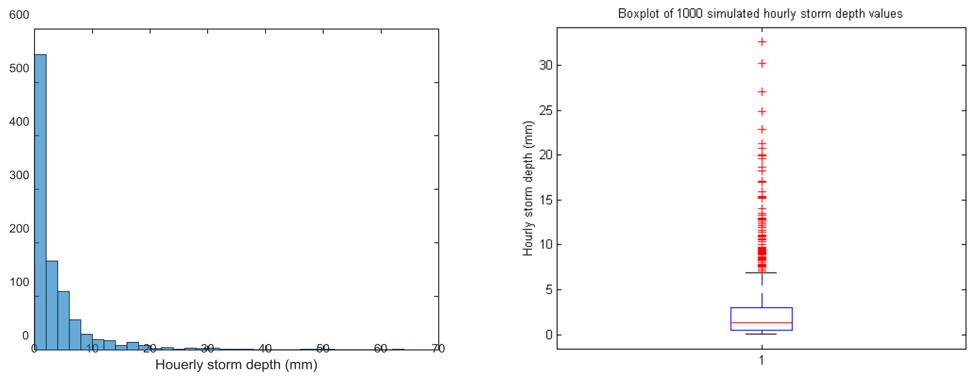


Figure 5.4: histogram and boxplot of 1000 simulated hourly storm depth values.

To obtain an estimate of the probability of an hourly storm depth greater than 10 millimetres, we can simply look at the proportion of the simulated values greater than 10 millimetres: here there are 44 such values, giving an estimated probability of 0.044.

This example illustrates the power of simulation. While this problem is probably still simple enough to enable a mathematical study of the distribution of interest (hourly rainfall depth in this case), a simulation based approach can easily handle additional complications.

For instance, suppose we are interested in the maximum rainfall delivered in one hour by all storms in a given year assuming that the number of storms is a Poisson random variable with mean 25 (assume that characteristics of storms are independent). To simulate a value for this maximum annual hourly storm depth, we simulate a realization of a Poisson random variable with mean 25, (*N* say) simulate hourly storm depth values for *N* different storms in the same way as we did above, and find the maximum of these. In MATLAB:

```
>> n=poissrnd(25)
>> x=lognrnd(-1.447,1.805,n,1);
>> y=lognrnd(1.956,0.6931,n,1);
>> b=(x>=1);
>> w=y.*b+x.*y.*(1-b);
>> m=max(w)
```

Here `m` will contain just one simulation of the annual maximum hourly depth. If we are interested in the distribution of annual maximum hourly depth we can iterate the above lines of code to obtain a sample from this distribution. I have simulated 1000 realizations of the annual maximum hourly depth in MATLAB: a histogram and boxplot of the resulting simulated values is shown in Figure 5.5.

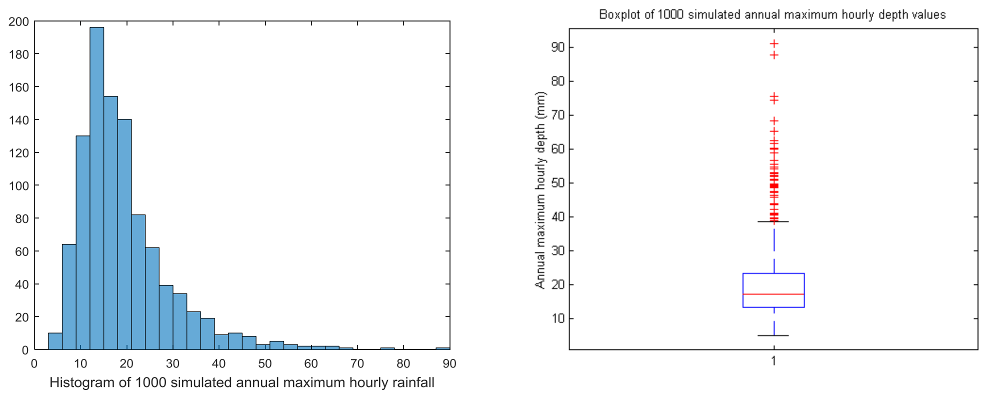


Figure 5.5: Histogram and boxplot of 1000 simulated annual maximum hourly rainfall depth values

Of course we can add still further complications to our model and to the questions we ask about it, and still study properties of the model using a simulation based approach.

5.5. Further applications of simulation

We give some further examples to illustrate the power of simulation as a tool for studying complex systems in engineering.

Example: wastewater treatment plant

The following example is from Kottegoda and Rosso, "Statistics, Probability and Reliability for Civil and Environmental Engineers," McGraw-Hill, 1998, Problem 8.8. An activated-sludge plant includes five serial processes: (1) coarse screening (2) grit removal (3) plain sedimentation (4) contact treatment, and (5) final settling. Let X_i denote the efficiency of the ith treatment, that is, the fraction of remaining pollutant removed by the ith serial treatment. For example, X_1 is the fraction of the pollutant removed by treatment process 1, X_2 is the fraction of the remaining pollutant removed by treatment process 2, and so on. The amount Q_{out} of pollutant in the effluent is given by

$$Q_{out} = (1-X_1)(1-X_2)(1-X_3)(1-X_4)(1-X_5)Q_{in}.$$

where Q_{in} denotes the amount of pollutant in the untreated inflow. A quality indicator of the performance of the plant is then defined as

$$Y = (1-X_1)(1-X_2)(1-X_3)(1-X_4)(1-X_5).$$

Consider a plant with the following single-process mean efficiencies in the removal of the 5-day 20°C biological oxygen demand (BOD):

$$\mu_1 = 0.05,\ \mu_2 = 0.05,\ \mu_3 = 0.20,\ \mu_4 = 0.70,\ \mu_5 = 0.10$$

where $\mu_i = E(X_i)$. Suppose that X_1, X_2, X_3 and X_5 are normal variates with standard deviations 0.01, 0.01, 0.04, and 0.02 respectively and that $X_4 \sim U[0.6,0.8]$. Simulate 100000 realizations from the distribution of Y, and plot a histogram of the simulated values. Also, give an estimate of the mean of Y based on your simulations.

Solution:

The following MATLAB code generates the required simulated values:

```
>> x1=normrnd(0.05,0.01,100000,1);
>> x2=normrnd(0.05,0.01,100000,1);
>> x3=normrnd(0.20,0.04,100000,1);
>> x4=unifrnd(0.6,0.8,100000,1);
>> x5=normrnd(0.10,0.02,100000,1);
>> y=(1-x1).*(1-x2).*(1-x3).*(1-x4).*(1-x5)
```

The command `unifrnd` in the fourth line above generates a sequence of uniform random variables on a given interval. Recall that we use the command `rand` to simulate uniform random variables from [0,1]. Typing `u=unifrnd(a,b,m,n)` generates an `m` by `n` matrix of independent uniform random variables on `[a,b]`. After typing the commands above, the 100000 required simulated values are in the vector `y`. A histogram of these values is shown in Figure 5.6.

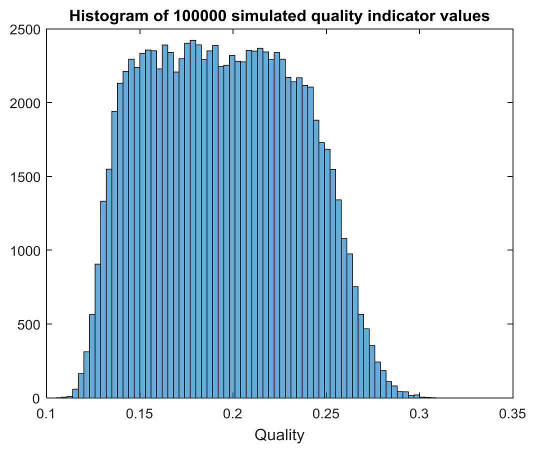


Figure 5.6: histogram of 100000 simulated quality indicator values for waste water treatment example.

An estimate of the mean of the distribution of the quality indicator is obtained by finding the sample mean of the simulated values: we have an estimated mean of 0.195 based on the simulation shown in the histogram.

Example: seismic hazard

The following example is based on Kottegoda and Rosso, "Statistics, Probability and Reliability for Civil and Environmental Engineers," McGraw-Hill, 1998, Problem 8.14. In a period of 600 years, about 330 earthquakes occurred in central Italy having an epicentral MCS intenstiy X exceeding 6. We assume for a given earthquake with epicentral MCS intensity greater than 6 that X is generated as X=6+Z where Z is a so-called exponential random variable with density function

$$f_Z(z) = \begin{cases} 0.91\exp(-0.91z) & \text{for } z > 0 \\ 0 & \text{otherwise} \end{cases}.$$

Generally, the form of the density of an exponential random variable is

$$f_X(x) = \lambda \exp\left(-\lambda x\right), \qquad x > 0.$$

where the parameter λ is positive. The mean of an exponential random variable with parameter λ is 1/λ.

Exponential random variables can be simulated in MATLAB using the command `exprnd`. The command `r=exprnd(μ,m,n)` simulates an `m` by `n` matrix `r` with entries which are independent exponential random variables with mean μ. For the density of Z given above, μ=1/0.91.

Seismic hazard at a specific site is represented by an MCS intensity Y related to the epicentral MCS intensity X by the following attenuation law:

$$Y = X - \frac{1}{\log \psi} \log \left[1 + \frac{\psi - 1}{\psi_0} \left(\frac{Z \varphi^{x_0 - x}}{z_0} - 1 \right) \right]$$

where Z denotes the distance from the epicenter, and z_0=9.5, x_0=10, ψ_0=1, ψ =1.5 and φ=1.3 are constants which are assumed to be known. (For further details see Grandori, G., Drei, A., Perotti, F. and Tagliani, A. (1991), "Macroseismic intensity versus epicentral distance: The case of central Italy," in: Stucchi, M., Postpischl, D., and D. Sleijko, eds., "Investigations of historical earthquakes in Europe," Technophysics, Vol. 193, pp. 165—181). Suppose that Z~U[3km,25km]. Approximate the distribution of Y by simulating 1000 values from its distribution and plotting a histogram.

Solution:

The following code in MATLAB produces the required simulation.

```
>> x=6+exprnd(1/0.91,1000,1);
>> z=unifrnd(3,25,1000,1);
>> z0=9.5;
>> x0=10;
>> psi0=1;
>> psi=1.5;
>> phi=1.3;
>> y=x-1/log(psi)*log(1+(psi-1)/psi0*(z.*phi.^(x0-x)/z0-1))
```

Note that in MATLAB `a.^x` where `a` is a scalar and `x` is a vector evaluates to a vector of the same length as `x` where the ith element of the vector is given by `a`^xi. The histogram of the 1000 simulated values is shown in Figure 5.7.

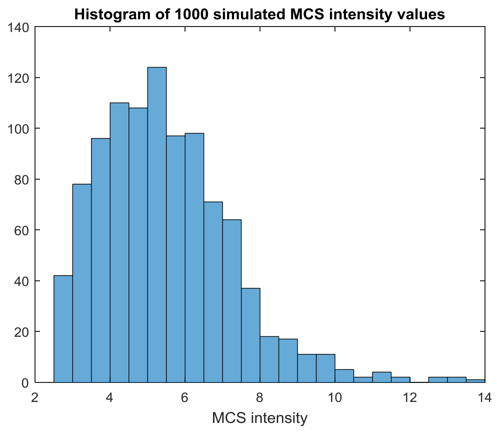


Figure 5.7: Histogram of 1000 simulated MCS intensity values.

6. Statistical tests in MATLAB

In your course you will be introduced to the rationale behind hypothesis testing, and you will be expected to perform the calculations required by hand. MATLAB does have some inbuilt functions for performing hypothesis tests, however, and we briefly summarise those functions in this section.

6.1. One-sample t-test

The MATLAB function `ttest` can be used to perform hypothesis tests regarding the mean of a normally distributed data vector with unknown variance. For example, entering the following command to MATLAB:

```
>> [h,p,ci,stats]=ttest(normrnd(.2,1,20,1))
```

performs a two-sided test of the null hypothesis that the data vector (here, a random sample of size 20 from a normal distribution with mean 0.2 and standard deviation 1) comes from a population with mean zero, using a 5% significance level. The following output is produced:

```
h =
     0

p =
    0.1268

ci =
   -0.0888
    0.6597

stats =
    tstat: 1.5966
       df: 19
       sd: 0.7997
```

A value of 0 for `h` indicates that the null hypothesis cannot be rejected. The *p*-value is 0.1268, and a 95% confidence interval for the mean is (-0.0888,0.6597). The t-statistic, which has a *t* distribution with 19 degrees of freedom under the null hypothesis, has an observed value of 1.5966. The estimated standard deviation is 0.7997.

If the population standard deviation is known, then the function `ztest` can be used. See the help file for further details.

6.2. Paired t-test

The function `ttest` can be also be used to perform paired t-tests. For example, entering the following command to MATLAB:

```
>> [h,p,ci,stats]=ttest(x,y)
```

performs a test of the hypothesis that the paired data vectors `x` and `y`, come from distributions with equal means, using a 5% significance level. The difference `x-y` is assumed to come from a normal distribution with unknown variance. `x` and `y` must have the same length.

6.3. Two-sample t-test

The function `ttest2` can be used to perform hypothesis tests of equality of means for two data vectors independently sampled from normal distributions with unknown variances. The syntax is:

```
>> ttest2(X,Y,ALPHA,TAIL,VARTYPE)
```

where `X` and `Y` are the data vectors, `ALPHA` specifies the desired significance level, `TAIL` is either `'both'`, `'right'` or `'left'` depending on whether we want a two-sided or upper or lower one-sided test, and `VARTYPE` is `'equal'` or `'unequal'` depending on whether or not we assume equal variances in the two populations.

6.4. The sign test

The sample median is an estimator of the population median, which is defined for a random variable *X* sampled from a continuous population to be the value $\tilde{\mu}$ satisfying

$$\Pr(X \leq \tilde{\mu}) = \Pr(X \geq \tilde{\mu}) = 0.5.$$

The sign test is a method for testing the null hypothesis that the median of a continuous population is equal to some nominal value against a one or two sided alternative. In the case of a population with a finite mean and where the population distribution is symmetric ($f_X(\tilde{\mu} - x) = f_X(\tilde{\mu} + x)$ for every x>0 where $f_X(x)$ is the density function defining the population distribution), the mean is equal to the median and so the sign test also provides a way of testing for the mean (for instance, for a normal population the mean and median are equal). We stress that the sign test provides a valid way of testing for the median for any continuous population, and that no assumption of normality is required. Tests like the sign test that do not assume that the population distribution is a member of some parametric family are sometimes called nonparametric tests or distribution free tests.

We wish to test H$_0$: $\tilde{\mu} = \mu_0$ against one of the alternatives $\tilde{\mu} \neq \mu_0, \tilde{\mu} < \mu_0$ or $\tilde{\mu} > \mu_0$ based on a sample $Y_1,\ldots,Y_n$ from the population. The test statisic used in the sign test is the number of sample values bigger than μ_0 (*U* say). If the null hypothesis is true, then $\Pr(Y_i \geq \mu_0) = 0.5$, i=1,…,n, and hence *U*~Bin(*n*,0.5). For the alternative H$_1$: $\tilde{\mu} \neq \mu_0$, we reject the null hypothesis if $U \leq c$ or $U \geq n-c$ where *c* is a constant chosen to achieve as nearly as possible a desired significance level. The form of the critical region here is intuitively sensible, since we reject the null hypothesis if either most of the sample values are bigger than μ_0 or most of the sample values are less than μ_0.

For the one-sided alternative H$_1$: $\tilde{\mu} < \mu_0$ the critical region is $U \leq c$, and for H$_1$: $\tilde{\mu} > \mu_0$ the form of the critical region is $U \geq c$ (where again *c* is chosen so that a desired significance level is achieved as nearly as possible). The constant *c* defining the critical region can be found for a given significance level from a table of cumulative binomial probabilities. However, we do not discuss finding these critical values from tables, since MATLAB has a command `signtest` that conducts the sign test and reports a *p*-value that can be compared with the significance level.

The MATLAB command signtest deals only with a two-sided alternative. If our sample values are contained in a vector `y`, then typing the command `signtest(y,m)` in MATLAB will report the *P*-value for testing the null hypothesis that the median is `m` against a two sided alternative.

6.5. The Wilcoxon signed-rank test

A better nonparametric test for the value of the population mean that applies when the population distribution is continuous and symmetric is the Wilcoxon signed-rank test. Note that the mean and median are equal when the mean exists and the population distribution is continuous and symmetric. In the Wilcoxon signed-rank test we test H_0: $\mu=\mu_0$ against a one or two sided alternative. If we have a sample $Y_1,\ldots,Y_n$ from the population, then the test statistic for the Wilcoxon signed-rank test is constructed by ranking the absolute differences $|Yi-\mu_0|$ and then finding the sum $R+$ of the ranks of the positive differences, and the sum $R-$ of the ranks of the negative differences. Under the assumption of a continuous and symmetric population distribution, we can work out the distribution of $R+$ and $R-$ under the null hypothesis, and these distributions depend only on the sample size *n*. For the two-sided alternative H_1:$\mu\neq\mu_0$ the Wilcoxon signed-rank test statistic is $R=\min(R+,R-)$ and the critical region takes the form $R\leq c$ where *c* is a constant chosen to achieve as nearly as possible a given significance level. The form of the critical region is quite intuitive. If most of the sample is below the hypothesized value for the mean then $R+$ will be small and *R* will be small. On the other hand, if most of the sample is above the mean, then $R-$ will be small and hence *R* will be small.

For the one-sided alternative H_1: $\mu<\mu_0$ the test statistic used is $R+$ and the form of the critical region is $R+\leq c$. For H_1: $\mu>\mu_0$ the test statistic used is $R-$ and the form of the critical region is $R-\leq c$. The critical values that define the critical region can be found from tables, although we do not discuss this. Instead, we discuss how to carry out the Wilcoxon rank-sum test using the MATLAB command signrank.

The MATLAB command signrank deals only with a two-sided alternative. If our sample values are contained in a vector `y`, and if the vector `x` is the same length as `y` and has entries which are all equal to μ_0, then typing signrank(x,y) in MATLAB will give the *p*-value for testing the null hypothesis that the mean is μ_0 against a two sided alternative.
As for the sign test, we can apply the Wilcoxon signed-rank test to paired data. Suppose we have pairs (X_1,Y_1), …, (X_n,Y_n) and that the distribution of pair differences is continuous and symmetric. We can apply the Wilcoxon signed-rank test to the pair differences to test if the mean difference is zero.

In MATLAB, if vectors `x` and `y` (of the same length) contain the pairs, we type `signrank(x,y)` to obtain the *P*-value of the test for a two-sided alternative.

7. Simple linear regression in MATLAB

7.1. Simple linear regression: fitting lines to data

In engineering applications we are often interested in statistical models that allow the distribution of a variable of interest to be described in terms of additional measured variables. The simple linear regression model allows us to fit lines to scatter plots in order to describe the relationship between two variables. The objective in doing this is usually to predict one variable (the response) in terms of the other (the predictor).

Example: predicting a rainfall-runoff relationship

(From Kottegoda and Rosso, 1997). The following table shows rainfall and runoff measurements at Pontelagoscuro on the Po river in northeast Italy, for the 31 years 1918 to 1948.

Year	Rainfall	Runoff	Year	Rainfall	Runoff	Year	Rainfall	Runoff
1918	1133	904	1928	1171	810	1938	940	517
1919	999	648	1929	876	490	1939	1196	801
1920	1501	1080	1930	1159	747	1940	1046	607
1921	807	549	1931	993	531	1941	1218	837
1922	1051	481	1932	1112	639	1942	948	522
1923	969	576	1933	1128	589	1943	896	444
1924	997	630	1934	1345	922	1944	950	407
1925	1090	688	1935	1290	787	1945	846	412
1926	1356	918	1936	1259	1039	1946	1011	679
1927	1133	733	1937	1529	958	1947	1096	585
						1948	1100	724

Figure 7.1 shows a scatter plot of rainfall against runoff. It is of interest to predict runoff in terms of rainfall in future years. If a method for predicting future annual rainfall is available, then a predictive relationship between rainfall and runoff may be useful for prediction of future runoff. These predictions of runoff are important to hydrologists involved in water resources management. In a simple linear regression model for these data, the response (variable to be predicted or explained) is runoff, and the predictor (thought to be useful for explaining variation in the response) is rainfall. We see from the plot that runoff seems to increase linearly as rainfall increases, and it appears that rainfall is a useful predictor of runoff. Of course, there are other predictors apart from rainfall that could be used to predict runoff, and statistical rainfall-runoff models used in practice incorporate many predictors, not just one predictor such as rainfall.

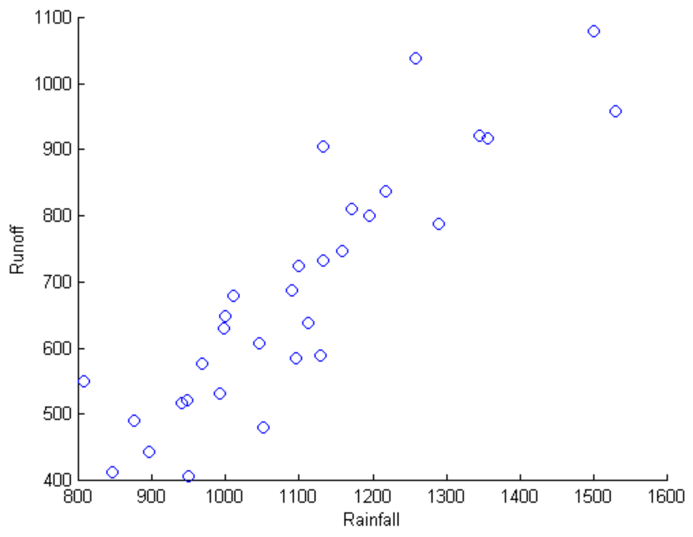


Figure 7.1: Scatter plot of runoff against rainfall for Pontelagoscuro on the Po river in northeast Italy for the period 1918-1948.

7.2. Fitting the simple linear regression model in MATLAB

Suppose we have n measurements $y=(y_1,\ldots,y_n)$ of a response variable and corresponding measurements $x=(x_1,\ldots,x_n)$ of a predictor. The simple linear regression model can be written as

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

where $\varepsilon_1,\ldots,\varepsilon_n$ are independent normal random errors with zero mean and variance σ^2. The term $\beta_0 + \beta_1 x_i$ is the systematic part of the model, or the component of the response that can be explained in terms of the predictor, whereas ε_i is a random error term representing variation in the response that cannot be explained by the predictor. We can think of the errors as being the "scatter" about the linear trend in a scatter plot like the one shown in Figure 7.2.

The method commonly used for estimation of β_0 and β_1 in the simple linear regression model is called least squares. Suppose we wanted to use a simple linear regression model for prediction of the response variable. If we are given x_i and some guesses α_0 and α_1 for β_0 and β_1 we would predict Y_i by $\alpha_0 + \alpha_1 x_i$. The error of prediction is $y_i - \alpha_0 - \alpha_1 x_i$. The idea of least squares estimation for the parameters β_0 and β_1 is to find values b_0 and b_1 which minimize with respect to α_0 and α_1 the sum of squared prediction errors

$$R(\alpha_0,\alpha_1) = \sum_i (y_i - \alpha_0 - \alpha_1 x_i)^2$$

In a sense this will give the best fitting line to the data. To find the least squares estimates b_0 and b_1 of β_0 and β_1 we minimize $R(\alpha_0,\alpha_1)$ by finding partial derivatives with respect to α_0 and α_1 and setting these to zero. Solving for α_0 and α_1 gives the least squares estimates b_0 and b_1. As shown in lectures, the solutions can be written:

$b_1 = \dfrac{S_{xy}}{S_{xx}}$ and $b_0 = \bar{y} - b_1\bar{x}$ where $\bar{y} = \dfrac{\sum_i y_i}{n}$, $\bar{x} = \dfrac{\sum_i x_i}{n}$, and

$$S_{xy} = \sum_i x_i y_i - \frac{\left(\sum_i x_i\right)\left(\sum_i y_i\right)}{n}$$

$$S_{xx} = \sum_i x_i^2 - \frac{\left(\sum_i x_i\right)^2}{n}$$

Example: rainfall-runoff model

We return to the example of the rainfall-runoff data for Pontelagoscuro on the Po river in northeast Italy. We have 31 measurements corresponding to the years 1918-1948. The response y is runoff and the predictor x is rainfall.

Figure 7.2 shows a scatter plot of runoff against rainfall with the fitted least squares regression line $b_0 + b_1 x$ superimposed.

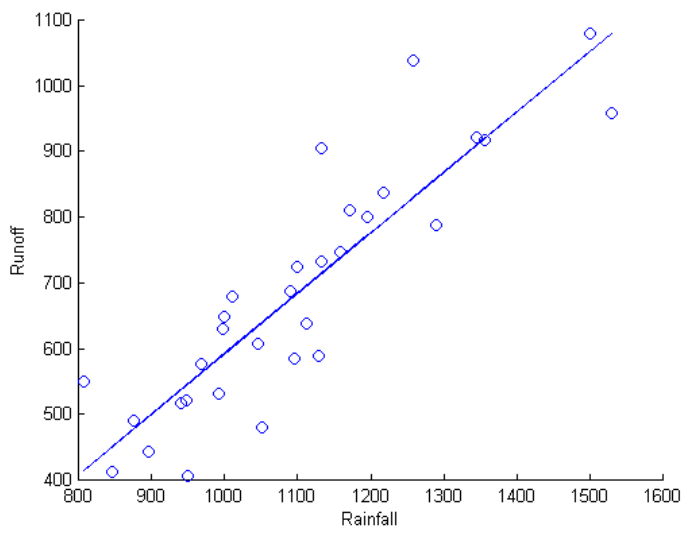


Figure 7.2: Scatter plot of runoff against rainfall for Pontelagoscuro, northeast Italy, 1918-1948 with least squares regression line superimposed.

Although we could find the least squares estimates for β_0 and β_1 using the formulae above, it is much easier to use the MATLAB command `fitlm`.

If the rainfall measurements are held in a row vector `rainfall` and the runoff measurements in a row vector `runoff` (the same length as `rainfall`) then we can run a linear regression using the `fitlm` command in MATLAB.

```
RainMod=fitlm(rainfall,runoff)
RainMod =
Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate       SE        tStat       pValue
                   ________    ________    _______    __________

    (Intercept)    -327.12       99.568    -3.2854     0.0026651
    x1             0.91946     0.089285     10.298    3.3758e-11


Number of observations: 31, Error degrees of freedom: 29
Root Mean Squared Error: 86.7
R-squared: 0.785,  Adjusted R-Squared 0.778
F-statistic vs. constant model: 106, p-value = 3.38e-11
```

We get some useful output by default, but the object `RainMod` stores just about anything you might want to know about your model. In the output, the first value under Estimate is the estimated intercept b_0 and the second value is the estimated slope b_1 (compare with the values obtained previously).

You might wonder what's actually happening inside the `fitlm` function. Firstly the function creates a design matrix `X`. This matrix has ones in the first column and the values of `rainfall` in the second column. To see why it does this, just observe that if β is the vector $(\beta_0, \beta_1)^T$ and if $y=(y_1,\ldots,y_n)^T$ and if $\varepsilon=(\varepsilon_1,\ldots,\varepsilon_n)^T$ then we can write the simple linear regression model in matrix language as

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

or

$$y = X\beta + \varepsilon.$$

X is called the design matrix and expressing the model in terms of the design matrix will be worth the effort when we go on to consider more complex models than the simple linear regression model. The `fitlm` function then estimates the model parameters via least squares.

7.3. Estimation of σ^2

As discussed in lectures, an unbiased estimator of σ^2 is

$$S^2 = \frac{\sum_i (y_i - b_0 - b_1 x_i)^2}{n-2}$$

An alternative expression for this estimator is

$$S^2 = \frac{S_{yy} - b_1^2 S_{xx}}{n-2}$$

where

$$S_{yy} = \sum_i y_i^2 - \frac{\left(\sum_i y_i\right)^2}{n}.$$

Example: rainfall-runoff model

We can read this directly from the default output, the mean square error is 86.7.

7.4. Decomposing variation

We give a basic result expressing a decomposition of variation in the response. This result also leads to a measure of the amount of variation explained in a simple linear regression model by the predictor.

Write $\hat{y}_i = b_0 + b_1 x_i$ for the fitted value corresponding to y_i and let $e_i = y_i - \hat{y}_i$ (the residuals or raw residuals). Then the following identity can be proven.

$$\sum (y_i - \bar{y})^2 = \sum_i (\hat{y}_i - \bar{y})^2 + \sum_i (y_i - \hat{y}_i)^2.$$

We call

$$SST = \sum_i (y_i - \bar{y})^2$$

the total sum of squares,

$$SSR = \sum_i (\hat{y}_i - \bar{y})^2$$

the regression sum of squares and

$$SSE = \sum_i (y_i - \hat{y}_i)^2$$

the residual or error sum of squares. With this notation we can write the above identity as *SST=SSR+SSE*. *SST* is the sum of squared deviations of the responses about their mean (the total variation of the responses), *SSR* is the sum of squared deviations of the fitted values about the mean of the responses (the amount of variation explained by the regression) and *SSE* is the sum of the squared residuals (amount of variation left unexplained by the regression). The proportion of total variation explained by the regression is called the coefficient of determination:

$$R^2 = \frac{SSR}{SST}.$$

Since *SSR=SST-SSE* from the above identity, we can also write

$$R^2 = 1 - \frac{SSE}{SST}.$$

The value of R^2 always lies between zero and one, with a value close to one indicating that the simple linear regression on the predictor explains most of the variation in the response.

Example: rainfall-runoff model

We can read this directly from the default output, the $R^{2=}$ 0.785.

7.5. Different kinds of residuals

The residuals we have defined above are often called the raw residuals (or simply residuals). However, there are other kinds of residuals which may be more useful than the raw residuals for some purposes. One problem with the raw residuals is that they will not have constant variance, even if the assumptions of the model hold and the errors ε_i do have constant variance. In particular, the variance of the *i*th raw residual e_i will depend on where x_i is in relation to the other predictor values.

It can be shown for the simple linear regression model that the variance of the *i*th residual is

$$Var(e_i) = \sigma^2 \left(1 - \left(\frac{1}{n} + \frac{(x_i - \bar{x})^2}{S_{xx}} \right) \right).$$

As the sample size n increases the variance of the residual will tend to σ^2. However, for small samples, the variance may be much less than σ^2, and the variance of the ith residual will depend on how far x_i is from the mean of the predictor values. Roughly speaking, an observation for an x_i a long way from the mean of the predictors is very influential and can pull the fitted least squares regression line towards itself, reducing the variance of the corresponding residual.

The above considerations lead to the idea of a standardized residual, where we divide the raw residual by an estimate of its standard deviation in order to adjust for the fact that the raw residual has a variance depending on the predictor value. We define the ith standardized residual as

$$r_i = \frac{e_i}{s\sqrt{1 - \left(\frac{1}{n} + \frac{(x_i - \bar{x})^2}{S_{xx}}\right)}}$$

where s is the estimated standard deviation of the errors. Plots of standardized residuals against the predictor values or the fitted values are better than the corresponding plots of the raw residuals for many purposes. For instance, it will be easier to assess the reasonableness of the constancy of variance assumption for the errors using the standardized residuals. It may also be easier to detect observations which do not fit the pattern of the rest of the data using the standardized residuals.

It can be shown that

$$\frac{e_i}{\sigma\sqrt{1 - \left(\frac{1}{n} + \frac{(x_i - \bar{x})^2}{S_{xx}}\right)}}$$

has a standard normal distribution if the assumptions of the model hold. The standardized residuals r_i are obtained by replacing σ in the above expression by s, and on the basis of our previous work we might expect that this results in the standardized residual having a t distribution. This is in fact not the case, but we can define another kind of residual, the studentized residual, which is t-distributed under the model assumptions. The ith studentized residual is

$$t_i = \frac{e_i}{s_{-i}\sqrt{1 - \left(\frac{1}{n} + \frac{(x_i - \bar{x})^2}{S_{xx}}\right)}}$$

where s_{-i} is the estimated standard deviation of the errors obtained when the regression model is fitted to the data with the ith observation excluded. The studentized residual has a t distribution with n-3 degrees of freedom for the simple linear regression model if the model assumptions hold.

This distributional result about the studentized residual gives us a way of detecting outlying observations: we can compare the ith studentized residual to the percentage points of a t-distribution to determine if an observation might be thought of as an outlier.

Suppose that the ith error ε_i has mean Δ (not necessarily zero) and that the assumptions of the linear regression model hold for the remaining observations. One formal way of detecting whether or not the *i*th observation is an outlier is to test the hypothesis

$$H_0: \Delta=0$$

against the alternative

$$H_1: \Delta\neq 0.$$

We can use as the test statisic the *i*th studentized residual t_i, which has a t_{n-3} distribution under the null hypothesis. The critical region for a test at level α is $t_i < -t_{\alpha/2;n-3}$ or $t_i > t_{\alpha/2;n-3}$, where $t_{\alpha/2;n-3}$ is the upper 100α/2 percentage point of the *t* distribution with *n*-3 degrees of freedom.
We must be careful about applying the above outlier test to all observations in a large data set. In a large data set, it is very likely that there will be some large studentized residuals purely by chance. The test above should be used as an informal diagnostic for detecting potential outliers which can then be further investigated in the scientific or engineering context of the problem.

Example: green liquor Na_2S concentration and paper machine production

(From Montogmery and Runger, 1999). An article in the Tappi Journal (March, 1986) presented data on green liquor Na_2S concentration and paper machine production available in dataset `paper`. A scatter plot of the data is shown in Figure 7.3.

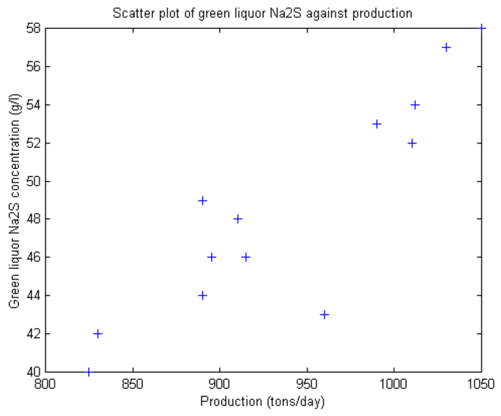


Figure 7.3: Scatter plot of green liquor Na_2S concentration (g/l) against production (tons/day).

When we fit the model in MATLAB using the `fitlm` command, the studentized residuals are stored inside the `paperMod` object. We can then plot the residuals using the `plotResiduals` command with the `paperMod` object.

```
PaperMod=fitlm(production,concentration);
plotResiduals(PaperMod,'fitted','ResidualType','Studentized')
```

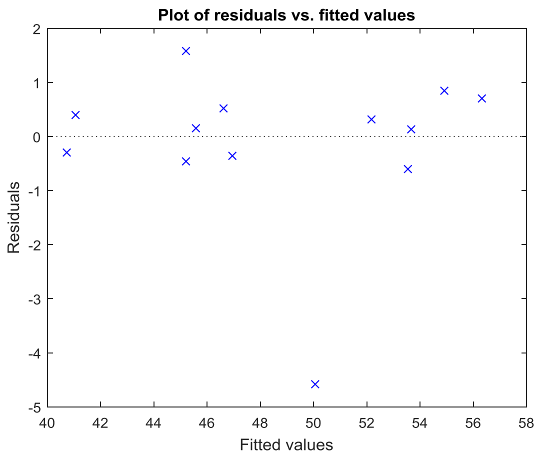


Figure 7.4: Plot of studentized residuals against fitted values for data on green liquor Na_2S concentration and paper machine production.

There is one studentized residual here that is much larger in magnitude than the others (a studentized residual of –4.58, corresponding to a production of 960 tons/day. Suppose we had a prior reason for thinking that this observation was not reliable. In this case it would be interesting to conduct the outlier test described above in order to decide whether this observation should be discarded when fitting the model.

8. Multiple linear regression in MATLAB

8.1. The multiple linear regression model

The general multiple linear regression has observations y_i, i=1,…,n, of a response variable which we wish to predict or explain in terms of corresponding measurements of k predictor variables $x_{i1},\dots,x_{ik}$, i=1,…,n. The observations are assumed to come from the model

$$y_i = \beta_0 + \beta_1 x_{i1} + \dots + \beta_k x_{ik} + \varepsilon_i$$

where $\beta_0,\beta_1,\dots,\beta_k$ are unknown parameters and the ε_i are zero mean independent random errors which are normally distributed with a common variance σ^2.

The multiple linear regression model can be written in matrix form. Let $y=(y_1,\dots,y_n)^T$ be the vector of the responses, let $\beta = (\beta_0,\dots,\beta_k)^T$ and let $\varepsilon = (\varepsilon_1,\dots \varepsilon_n)^T$. Also, let X be the matrix

$$X = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1k} \\ \vdots & & & \vdots \\ 1 & x_{n1} & \cdots & x_{nk} \end{bmatrix}.$$

Thus X is the matrix with entries in its first column equal to 1, and the remaining k columns given by the k vectors of predictor values. X is often called the design matrix. We can write the multiple linear regression model as

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1k} \\ \vdots & & & \vdots \\ 1 & x_{n1} & \cdots & x_{nk} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{bmatrix}.$$

or

$$y = X\beta + \varepsilon.$$

8.2. Estimation of model parameters

As in the simple linear regression model, we can estimate the parameters β using least squares. Let $\alpha = (\alpha_0, \ldots \alpha_k)^T$ be some guess for the parameter vector β. If we predict the responses by their means assuming β=α, the sum of squared prediction errors is

$$R(\alpha) = (y - X\alpha)^T (y - X\alpha) = \sum_{i=1}^{n} (y_i - \alpha_0 - \alpha_1 x_{i1} - \ldots - \alpha_k x_{ik})^2 .$$

The least squares estimate *b* of β is the value that minimizes *R*(α) with respect to α. To obtain an expression for *b*, we can follow a similar argument to the one we gave for the simple linear regression model. The least squares estimate *b* can be written in matrix language as

$$b = \left(X^T X\right)^{-1} X^T y$$

where X^T denotes the transpose of the matrix *X* and $(X^T X)^{-1}$ is the inverse of $X^T X$.

We also need to be able to estimate the variance of the errors σ^2. It can be shown that an unbiased estimator of the error variance is

$$S^2 = \frac{(y - Xb)^T (y - Xb)}{n - k - 1} = \frac{\sum_{i=1}^{n} (y_i - b_0 - b_1 x_{i1} - \ldots - b_k x_{ik})^2}{n - k - 1}.$$

Note that in the case of the simple linear regression model (one predictor) this reduces to the expression we used previously.

Example: predicting 28-day concrete strength

The following example is from Metcalfe, “Statistics in Civil Engineering,” Arnold Publishers, 1997.

For fourteen concrete cubes from the same batch of cement the density, 7-day strength and 28-day strength are measured. We want to be able to predict 28-day strength from information available at seven days. The data are shown in the table below: here we have written *y* for the 28-day strength, x_1 for the 7-day strength and x_2 for the density.

y=28-day strength	x_1=7-day strength	x_2=Density
46.2	40.3	2058
43.2	38.1	2074
42.5	36.0	2045
47.8	40.5	2085
42.5	34.9	2073
42.5	33.2	2087

42.0	34.2	2069
39.2	32.5	2055
44.3	38.8	2087
56.7	44.4	2412
56.8	45.2	2409
52.1	37.6	2418
55.3	42.7	2406
56.8	43.8	2415

We again use the `fitlm` command, this time to fit a multiple linear regression model to the concrete strength data using MATLAB (assuming this model is appropriate). First we create a matrix which contains both the predictors, `Strenght7` and `Density`, then we use them to predict `Strength28`.

```
>> X=[Strength7,Density];
>> concreteMod=fitlm(X,Strength28)
Linear regression model:
    y ~ 1 + x1 + x2

Estimated Coefficients:
                   Estimate       SE         tStat       pValue
                   ________    _________    ________    __________

    (Intercept)     -29.698       2.9617    -10.027     7.1942e-07
    x1              0.70751     0.079246      8.928     2.2673e-06
    x2             0.022808    0.0019773     11.535     1.7442e-07


Number of observations: 14, Error degrees of freedom: 11
Root Mean Squared Error: 0.827
R-squared: 0.986,  Adjusted R-Squared 0.984
F-statistic vs. constant model: 392, p-value = 5.95e-11
```

Most of the key information can be gained from the default output. In the output, the first value under `Estimate` is the estimated intercept b_0 and the second value is the estimated slope b_1. The mean square error is 0.827 and $R^{2=}$ 0.986.

8.3. Inference for model coefficients

In addition to obtaining point estimates of model parameters it is important to be able to derive interval estimates that give a range of plausible values for the parameters. In lectures we stated some distributional results that allow construction of confidence intervals:

A 100(1-α)% confidence interval for β_i is given by

$$b_i \pm t_{\alpha/2;n-k-1}\hat{s}e(b_i).$$

We can also test hypotheses on individual coefficients:

To test the hypothesis

$$H_0: \beta_i = \beta^*$$

against a one or two sided alternative we use the test statistic

$$T = \frac{b_i - \beta^*}{\hat{s}e(b_i)}$$

which has a t_{n-k-1} distribution under the null hypothesis. For a test with significance level α and the two-sided alternative $H_1: \beta_i \neq \beta^*$, the critical region is

$$T < -t_{\alpha/2;n-k-1} \text{ or } T > t_{\alpha/2;n-k-1}.$$

For the one-sided alternatives $H_1: \beta_i < \beta^*$ and $H_1: \beta_i > \beta^*$ the critical region is modified to $T < -t_{\alpha;n-k-1}$ or $T > t_{\alpha;n-k-1}$ respectively. These hypothesis tests are sometimes called partial *t*-tests.

Example: concrete strength data

To can obtain the confidence intervals for the concrete strength data using the `coefCI` function on the `concreteMod` object.

```
>> X=[Strength7,Density];
>> concreteMod=fitlm(X,Strength28)
>> coefCI(concreteMod)
  -36.2162  -23.1790
    0.5331    0.8819
    0.0185    0.0272
```

This output gives us a matrix with two columns, the lower and upper bounds of the 95% confidence intervals for β_0, β_1, and β_2 respectively.

We can add a second argument to `coefCI` to control the level of confidence for the intervals. Typing `coefCI(concreteMod,alpha)` where `alpha` is some constant between zero and one produces 100(1-`alpha`)% confidence intervals. With `alpha` equal to 0.05 we obtain a 95% confidence interval.

Example: Big Mac data

Suppose we want to construct 99% confidence intervals for the regression coefficients for the Big Mac data. T vectors `engsal` and `engtax` are average salary of an electrical engineer and tax rate paid by engineers and the vector `bigmacindex` contains minutes of labour required by an average worker to buy a Big Mac and French fries.

```
In MATLAB, we type:

>> X=[engsal,engtax];
bigmacMod=fitlm(X,bigmacindex);
coefCI(bigmacMod,0.01)

ans =

   65.8158  133.9485
   -2.4638   -0.7387
   -1.3310    1.5613
```

This gives us the 99% confidence intervals (we set the second argument of `coefCI` equal to 0.01 here to get 99% confidence intervals).

8.4. Confidence intervals for the mean and prediction intervals

Confidence intervals for the mean response

If we write x_0 for a vector of new predictor values $(1,x_{01},\ldots,x_{0k})^T$ then the estimated mean response when the predictors take the values $x_{01},\ldots,x_{0k}$ is

$$\hat{y}(x_0) = x_0^T b = b_0 + b_1 x_{01} + \ldots + b_k x_{0k} .$$

We have the following result:

A 100(1-α)% confidence interval for the mean response at x_0 is

$$\left(x_0^T b - st_{\alpha/2;n-k-1}\sqrt{x_0^T\left(X^TX\right)^{-1}x_0}\,,\, x_0^T b + st_{\alpha/2;n-k-1}\sqrt{x_0^T\left(X^TX\right)^{-1}x_0} \right).$$

Prediction intervals

In addition to giving a confidence interval for the mean at x_0, we need to be able to quantify our uncertainty about a future observation Y_0 observed at the predictor values in x_0.

A 100(1-α)% prediction interval for Y_0 is given by

$$\left(x_0^T b - st_{\alpha/2;n-k-1}\sqrt{1+x_0^T\left(X^TX\right)^{-1}x_0}\,,\, x_0^T b + st_{\alpha/2;n-k-1}\sqrt{1+x_0^T\left(X^TX\right)^{-1}x_0} \right).$$

Example: concrete strength data

We illustrate the construction of confidence intervals and prediction intervals in MATLAB using the concrete strength data. Suppose we are interested in predicting 28-day strength when 7-day strength is 36 and density is 2050. The following commands and output in MATLAB give a 95% confidence interval for the mean and a 95% prediction interval for these values of the predictors.

```
>> X=[Strength7,Density];
>> concreteMod=fitlm(X,Strength28)
>> [ypred,yci] = predict(concreteMod,[36,2050]) ;
>> yci =

   41.8856   43.1739

>> [ypred,ypi]=
predict(concreteMod,[36,2050],'Prediction','observation');
ypi

ypi =

   40.5994   44.4601
```

Hence a 95% confidence interval for the mean when 7-day strength is 36 and density is 2050 is (41.8856,43.1739) and a 95% prediction interval is (40.5994,44.4601).

8.5. Assessing overall model adequacy

In our discussion of the simple linear regression model, we gave a decomposition of total variation into variation explained by the regression and residual variation. A similar decomposition holds for the multiple linear regression model. The coefficient of determination (which is a measure of the proportion of variation in the response explained by the regression) is defined as for the simple linear regression model:

$$\boxed{R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}.}$$

It follows from the identity $SST=SSR+SSE$ that R^2 always lies between zero and one, with a value close to one indicating that most of the variation in the response can be explained by the predictors, and a value close to zero indicating that a large proportion of variation is left unexplained by the predictors.

One thing we would like to do in assessing the overall adequacy of a multiple linear regression model is to test the hypothesis that all the coefficients of the predictors in the model are zero against the alternative that at least one of the coefficients is nonzero. This is a test of whether the predictors can help to explain a significant amount of variation in the response.

More precisely, we test

$$H_0: \beta_i=0,\ i=1,\ldots,k$$

against the alternative

$$H_1: \text{Not all } \beta_i=0,\ i=1,\ldots,k.$$

The test statisic for this hypothesis test is

$$F = \frac{SSR\ /k}{SSE\ /(n-k-1)}$$

The distribution of F under the null hypothesis can be calculated: it has a distribution belonging to the F family of distributions. A distribution in the F family has two parameters (called the degrees of freedom). The statistic given above has an F distribution with k and n-k-1 degrees of freedom under the null hypothesis, and we write $F \sim F_{k,n-k-1}$. If we write $F_{\alpha;k,n-k-1}$ for the upper 100α percentage point of an F distribution with k and n-k-1 degrees of freedom (that is, the value that bounds an area of α in the upper tail of the $F_{k,n-k-1}$ density) then the critical region for the above hypothesis test is

$$F > F_{\alpha;k,n-k-1}.$$

We can summarize the testing procedure as follows.

To test

$$H_0: \beta_i=0,\ i=1,\dots,k$$

against the alternative

$$H_1: \text{Not all } \beta_i=0,\ i=1,\dots,k.$$

we use the test statistic

$$F = \frac{SSR\ /\ k}{SSE\ /(n-k-1)}$$

which has an $F_{k,n-k-1}$ distribution under H_0. We reject H_0 at significance level α if

$$F > F_{\alpha;k,n-k-1}.$$

In most statistical packages the decomposition of variation $SST=SSR+SSE$ and the above overall test of model significance is displayed in an analysis of variance (ANOVA) table. The form of the ANOVA table for the multiple linear regression model is shown below.

Source	DF	SS	MS	F	P
Regression Error	k n-k-1	SSR SSE	SSR/k $SSE/(n$-k-$1)$	$\frac{SSR\ /\ k}{SSE\ /(n-k-1)}$	p-value
Total	n-1	SST			

The first column "Source" describes the nature of the variation considered in the remainder of each row (we are interested in the decomposition of total variation into variation explained by the regression and unexplained or error variation). The third column "SS" (which stands for sums of squares) gives SSR, SST and SSE. The second column "DF" (for degrees of freedom) gives the values we divide the sums of squares by in computing the test statistic F. The fourth column "MS" (which stands for mean squares) gives the numerator and denominator for the F statistic used in the global test of model significance. The fifth column, labelled "F" gives the test statistic for the test of model significance, and the final column gives the p-value of the test.

Example: concrete strength data

The F statistic and associated p-value are given in the last line of the default output from the `fitlm` object.

```
>> X=[Strength7,Density];
>> concreteMod=fitlm(X,Strength28)
>> concreteMod

concreteMod = 


Linear regression model:
    y ~ 1 + x1 + x2

Estimated Coefficients:
                   Estimate       SE          tStat       pValue   
                   ________    _________     _______    __________

    (Intercept)     -29.698       2.9617     -10.027    7.1942e-07
    x1              0.70751     0.079246       8.928    2.2673e-06
    x2             0.022808    0.0019773      11.535    1.7442e-07


Number of observations: 14, Error degrees of freedom: 11
Root Mean Squared Error: 0.827
R-squared: 0.986,  Adjusted R-Squared 0.984
F-statistic vs. constant model: 392, p-value = 5.95e-11
```

So from the *p*-value, at the 5% level (or the 1% level) we reject the null hypothesis that the coefficients of all the predictors are zero in favour of the alternative that at least one of these coefficients is nonzero (since the *p*-value for the test is zero to four decimal places). The coefficient of determination is 0.986.

Example: Big Mac data

As another example, consider the Big Mac data set available in `bigmac.csv`. Our responses are in the vector `bigmacindex`, and the predictors are `engsal` and `elgtax`. We type the following

```
>> X=[engsal,engtax];
bigmacMod=fitlm(X,bigmacindex)
```

```
bigmacMod = 

Linear regression model:
    y ~ 1 + x1 + x2

Estimated Coefficients:
                   Estimate      SE        tStat       pValue  
                   ________    _______    _______    __________

    (Intercept)     99.882      12.626     7.9107    7.4597e-10
    x1             -1.6012     0.31968    -5.0089    1.0341e-05
    x2             0.11515     0.53598    0.21483       0.83094


Number of observations: 45, Error degrees of freedom: 42
Root Mean Squared Error: 33.7
R-squared: 0.467,  Adjusted R-Squared 0.442
F-statistic vs. constant model: 18.4, p-value = 1.79e-06
```

So if we are testing at the 5% level, since the *p*-value is less than 0.05, we conclude that the predictors do contain useful information for explaining variation in the response. The coefficient of determination is 0.467.

8.6. Stepwise approaches to model selection

Sequential methods start with an initial model and then make a sequence of additions or deletions of predictors, attempting to improve the fit of the model at each step. We do a search through the model space, visiting only a small subset of possible models, ending our search when some stopping rule is satisfied.

Sequential methods are often used in practice and you need to know about them. However, with modern computing power, they are not needed unless the number of predictors is very large. Furthermore, even if the number of predictors is very large, often the experience of the scientist or engineer in the context of the problem can reduce the set of potential predictors to a smaller set, and then some other model selection criterion can be applied. It should be emphasized that we do not always want to select just a single “best” model in any case, but may wish to identify a number of good models for further investigation.

There are three basic sequential methods (and many variants of these): forward selection, backward selection and stepwise.

Forward selection

In forward selection, we start with some initial model (typically the model including no predictors) and then attempt to improve on the current model by adding the predictor which most improves the fit until some stopping rule is satisfied.

Our measure of the improvement of fit upon addition of a predictor is the absolute value of the partial *t*-statistic for testing whether the predictor coefficient is zero. That is, we use

$$|t_i| = \left| \frac{b_i}{\hat{s}e(b_i)} \right|$$

The predictor *i* not currently in the model for which $|t_i|$ is largest is considered the best to add.

The algorithm is as follows:

1. Fit the model involving just an intercept (no predictors)
2. Repeat:
   a. Considering all possible one variable additions to the current model, find the predictor *i* for which $|t_i|$ is largest and add it to the model if $|t_i|>T_{in}$, where T_{in} is some cutoff value.
   b. If no variable could be added in a., then stop.

In order to implement the above algorithm, the cutoff value T_{in} must be specified. A common choice for T_{in} is 2.0. Unfortunately, differences in the final model selected depending on the cutoff value chosen do occur.

Example: concrete strength data

With 28-day strength as the response and 7-day strength and density as predictors, there are four possible models (include neither predictor, both predictors, just density or just 7-day strength).

We start with the model with just an intercept, and consider models obtained by adding one variable. We can use the `stepwiselm` command to carry out model selection. To start with the model with no predictors we use the option '`constant`'.

```
>> X=[Strength7,Density];
>> stepwiselm(X,Strength28,'constant')
1. Adding x2, FStat = 93.219, pValue = 5.2265e-07
2. Adding x1, FStat = 79.7099, pValue = 2.26728e-06

ans =

Linear regression model:
    y ~ 1 + x1 + x2

Estimated Coefficients:
                   Estimate        SE          tStat        pValue
                   ________    _________      _______     __________

    (Intercept)     -29.698       2.9617      -10.027     7.1942e-07
    x1              0.70751     0.079246        8.928     2.2673e-06
    x2             0.022808    0.0019773       11.535     1.7442e-07


Number of observations: 14, Error degrees of freedom: 11
Root Mean Squared Error: 0.827
R-squared: 0.986,  Adjusted R-Squared 0.984
F-statistic vs. constant model: 392, p-value = 5.95e-11
```

From the output we can see `x2 (Density)` was added in the first step followed by `x1 (Strenght7)`. The final model contains both the predictors.

8.7. Residuals and influence in multiple linear regression

As for simple linear regression, the basis for model criticism in multiple linear regression is usually some kind of residual analysis.

Let y_i be the value of the ith response, i=1,…,n, and x_{i1},…,x_{ik} values of k predictor variables. The responses y_i are assumed to follow the model,

$$y_i = \beta_0 + \beta_1 x_{i1} + ... + \beta_{ik} + \varepsilon_i$$

where as usual the ε_i are zero mean independent normally distributed errors with variance σ^2. In matrix notation we write

$$y = X\beta + \varepsilon$$

where y is the vector of the responses, X is the design matrix, $\beta = (\beta_0, \dots, \beta_k)^T$ is the vector of unknown parameters in the mean response and ε is the vector of random errors. We write b for the least squares estimate of β, and $\hat{y}_i$ for the fitted value at the predictor values x_{i1},…,x_{ik},

$$\hat{y}_i = b_0 + b_1 x_{i1} + ... + b_k x_{ik}$$

If we let x_i be the vector formed from the ith row of the design matrix, $x_i=(1,x_{i1},\dots,x_{ik})^T$, then we can write the fitted value in matrix notation as

$$\hat{y}_i = x_i^T b.$$

and the ith raw residual e_i is defined to be $e_i = y_i - \hat{y}_i$.

We can use plots of residuals (against fitted values or the predictors) to check some of the multiple linear regression model assumptions. The interpretation of residual plots is much the same as for the simple linear regression case. In particular, a trend in the mean level of the residuals as the fitted values or predictors increase suggests that the mean of the responses is not correctly specified, whereas a trend in the variability of the residuals casts doubt on the constancy of variance assumptions for the errors. Residual plots can also be used to identify outliers that do not fit the trend of the rest of the data.

While plots of the raw residuals can be very useful, as for the simple linear regression case there are other kinds of residuals that may be more informative for some purposes. A drawback with the raw residuals is that they do not have the same variance, even if the assumptions of the model hold. So we may wish to standardize the raw residuals by dividing by an estimate of their standard deviation to obtained a standardized residual. We write r_i for the ith standardized residual.

For assessing whether an observation fits the pattern of the rest of the data, it is helpful to know the distribution of the residuals so that it can be determined if an observation is unusually large or small. Again following the simple linear regression case, we can define studentized residuals (similar to standardized residuals but employing an estimator of the error variance which excludes the ith observation) that have a t-distribution if the model assumptions hold.

In particular, the ith studentized residual t_i has a t_{n-k-2} distribution. We can test whether an observation is an outlier using the studentized residual as the test statistic: as in the simple linear regression case, this test should be considered an informal diagnostic to highlight observations that require further examination in the context of the problem. We should not apply such an outlier test blindly to all obseravtions in the data set and exclude observations which fail the test, since some large studentized residuals are likely to occur purely by chance in a large data set.

Example: concrete strength data

We can plot residuals using the `plotResiduals` command.

```
>> X=[Strength7,Density];
>> concreteMod=fitlm(X,Strength28)
>> plotResiduals(concreteMod,'fitted','ResidualType','Studentized')
```

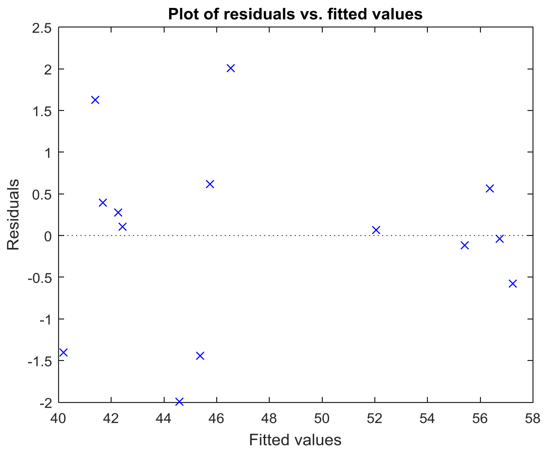


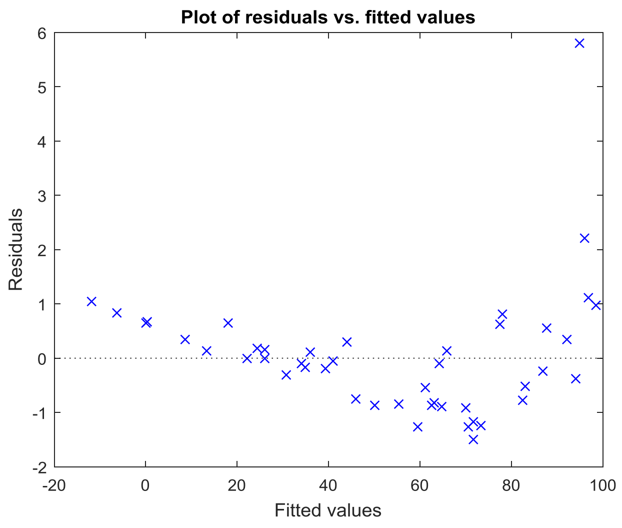
Figure 8.1: Plot of the studentized residuals against fitted values

From the graph of studentized residuals against fitted values it seems that the constancy of variance assumption might be violated: the studentized residuals for large fitted values tend to be smaller in absolute value than those for smaller values.

Example: Big Mac data set

We return to the Big Mac data set. The responses are in a vector `bigmacindex`, and the predictors are `engsal` (average salary of an electrical engineer in US dollars) and `elgtax`. (the tax rate for engineers).

```
>> X=[engsal,engtax];
>> bigmacMod=fitlm(X,bigmacindex);
>> plotResiduals(bigmacMod,'fitted','ResidualType','Studentized')
```



What do you conclude from this plot? Do you think this model is appropriate for these data? Why, or why not?

9. ANOVA in MATLAB

9.1. One way ANOVA

The two sample *t*-test gives us a test for the equality of two population means. We now discuss briefly a method for comparing means for more than two populations, and describe the connection between this method and multiple linear regression.

Suppose that we have *k* different populations, and that we take a random sample of n_i observations from the *i*th population, i=1,…,k. Write y_{ij} for the *j*th observation from the *i*th population, i=1,..,k, j=1,…,n_i. Also, write μ_i for the population mean of the *i*th population, i=1,..,k, and suppose that the population variances are all equal. We wish to determine whether all population means are equal, or whether there are differences among the means of the populations.

The above experimental situation where population means are to be compared based on random samples from the populations is referred to as a completely randomized single-factor experiment, and the analysis to determine differences among population means is referred to as a single factor or one-way analysis of variance (ANOVA). The following example gives one illustration of a situation where a one-way ANOVA might be of interest.

Example: compressive strength of concrete

(From Montgomery and Runger, 1999). The compressive strength of concrete is being studied, and four different mixing techniques are being investigated. The following data have been collected.

Mixing Technique				
1	3129	3000	2865	2890
2	3200	3300	2975	3150
3	2800	2900	2985	3050
4	2600	2700	2600	2765

In each row there are four observations of compressive strength for one of the mixing techniques. The question of interest here is: do mixing techniques have an effect on the compressive strength of the concrete? We will return to this example later.

We can write our model for the data here in a form similar to the multiple linear regression model. We assume that the observations y_{ij} follow the model

$$y_{ij} = \mu_i + \varepsilon_{ij}$$

where the ε_{ij} are zero mean independent normally distributed errors with variance σ^2. Now define variables x_{ijm}=1 if $m=i$ and zero otherwise. Think of $x_{ij1},\ldots,x_{ijk}$ as being k predictor values associated with the response y_{ij}: of these k predictor values, x_{iji} is one, and the remaining predictors are zero. Hence we can write

$$\begin{aligned} y_{ij} &= \mu_i + \varepsilon_{ij} \\ &= \mu_1 x_{ij1} + \ldots + \mu_k x_{ijk} + \varepsilon_{ij} \end{aligned}$$

which is in the form of a multiple linear regression model with no intercept, parameters $\mu_1,\ldots,\mu_k$ in the mean response, and predictors $x_{ij1},\ldots,x_{ijk}$ for the observation y_{ij}. As a matter of fact, the mathematical framework for estimating parameters in the model here and conducting tests of hypotheses is the same as that used in the study of the multiple linear regression model. We can estimate parameters, estimate our uncertainty about parameters, obtain predictions and so forth as before.

One result that carries over from multiple linear regression is that there is a decomposition of total variation into variation explained by the predictors (variation which can be explained in terms of membership of the different population groups) and variation unexplained. It can be shown that the usual decomposition of variation in multiple linear regression $SST=SSR+SSE$ can be written here as

$$\sum_i \sum_j \left(y_{ij} - \bar{y}_{..}\right)^2 = \sum_i n_i \left(\bar{y}_{i.} - \bar{y}_{..}\right)^2 + \sum_i \sum_j \left(y_{ij} - \bar{y}_{i.}\right)^2$$

where we have written $\bar{y}_{..}$ for the mean of all the observations, and $\bar{y}_{i.}$ for the mean of the observations from the ith population.

We can test the hypothesis

$$H_0: \mu_1 = \ldots = \mu_k$$

against the alternative

$$H_1: \text{Not all } \mu_i \text{ are equal, } i=1,\ldots,k.$$

Using the test statistic

$$F = \frac{SSR \;/(k-1)}{SSE \;/(n-k)}$$

which has an $F_{k-1,n-k}$ degrees of freedom under the null hypothesis (where n is the total number of observations). The critical region for the test when the significance level is α is $F > F_{\alpha;k-1,n-k}$.

Example: compressive strength of concrete

We illustrate testing for the equality of means in a completely randomized experiment with a single factor using the compressive strength of concrete data and MATLAB's `anova1` command. We have the same number of observations in each population (mixing method) in this example, and suppose we have stored the observations in a matrix `X` in MATLAB, with different columns corresponding to different mixing types:

```
» X
X =

          3129          3200          2800          2600
          3000          3300          2900          2700
          2865          2975          2985          2600
          2890          3150          3050          2765
```

We can obtain the *p*-value for the test that the four population means are equal (mixing type has no effect) by typing `anova1(X)` in MATLAB. The number which appears in the command window is the *p*-value. The `anova1` command also produces an analysis of variance table which gives the decomposition $SST=SSR+SSE$ and the calculations leading to the F statistic used in the test for equality of means, as well as a side by side boxplot of the observations for the four mixing types (not very helpful here with only four observations in each group). This will be demonstrated in lectures.

Our *p*-value here is approximately 4.8×10^{-4} so that we reject the null hypothesis of equality of population means for the mixing types at the 5% level (or the 1% level). There is a form of the `anova1` command that can be used when there are differing numbers of observations from the populations to be compared (see the next example).

Example: blondes are tougher

Studies conducted at the University of Melbourne indicate that there may be a difference between the pain thresholds of blonds and brunettes. Men and women of various ages were divided into four categories, according to hair colour (recorded as Light Blond, Dark Blond, Light Brunette or Dark Brunette). For each person a pain threshold score (Pain) was measured.

The data are shown in the table below.

Hair Colour	Pain Score
Light Blond	62
Light Blond	60
Light Blond	71
Light Blond	55
Light Blond	48
Dark Blond	63

Dark Blond	57
Dark Blond	52
Dark Blond	41
Dark Blond	43
Light Brunette	42
Light Brunette	50
Light Brunette	41
Light Brunette	37
Dark Brunette	32
Dark Brunette	39
Dark Brunette	51
Dark Brunette	30
Dark Brunette	35

There are 19 rows in the table above (19 subjects) and suppose we enter the data in MATLAB in the form of two vectors: a vector `group` taking values of 1, 2, 3 or 4 (where 1 indicates a light blond subject, 2 a dark blond, 3 a light brunette and 4 a dark brunette) and a vector `pain` containing the pain threshold measurements for the 19 subjects (the values in the second column of the table).

Then we can do the one way analysis of variance by typing `anova1(pain,group)`. MATLAB gives the *p*-value for testing equality of means for the hair colour groups as 0.0041. Since this is less than 0.05, at the 5% level we reject the null hypothesis of equality of means for the hair colour groups.

9.2. One way ANOVA as a regression model

We discussed above how to write a general completely randomized single factor experiment as a multiple linear regression with a suitable design matrix *X*. The X matrix will contain indicator variables, which for each observation have a 1 in the column corresponding to the group that observation is in, and 0's everywhere else. We can then write the model in matrix form:

$$\begin{bmatrix} y_{11} \\ y_{12} \\ y_{13} \\ y_{14} \\ y_{21} \\ y_{22} \\ y_{23} \\ y_{24} \\ y_{31} \\ y_{32} \\ y_{33} \\ y_{34} \\ y_{41} \\ y_{42} \\ y_{43} \\ y_{44} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \end{bmatrix} + \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{12} \\ \varepsilon_{13} \\ \varepsilon_{14} \\ \varepsilon_{21} \\ \varepsilon_{22} \\ \varepsilon_{23} \\ \varepsilon_{24} \\ \varepsilon_{31} \\ \varepsilon_{32} \\ \varepsilon_{33} \\ \varepsilon_{34} \\ \varepsilon_{41} \\ \varepsilon_{42} \\ \varepsilon_{43} \\ \varepsilon_{44} \end{bmatrix}$$

or

$$y = X\mu + \varepsilon.$$

MATLAB's `fitlm` function does this internally, and we don't need to worry about it. All we have to do is tell `fitlm` which of the predictors are categorical. We can use the output object from `fitlm` command to estimate parameters, compute confidence intervals for parameters, find residuals and so on. We can carry out an `anova` using the `fitlm` function just as we did for linear models, and indicating which variable(s) are categorical. You can have multiple categorical variables to do two-way or three-way ANOVAs.

```
>> painMod=fitlm(group,pain,'CategoricalVars',[1]);
>> anova(painMod)

ans = 

                  SumSq     DF    MeanSq      F         pValue  
                  ______    __    ______    ______    _________

    x1            1360.7     3    453.58    6.7914    0.0041142
    Error         1001.8    15    66.787                        
```

The main advantage of using linear models to do an ANOVA allows us to check model assumptions using the same methods we used for linear models.

Residual analysis

Plotting of the raw residuals against an index of the population groups or the fitted values can help to detect violations of assumptions. In particular, we might look for violations of the constancy of variance assumption, outliers, or obvious departures from normality.

Example: compressive strength of concrete

We illustrate residual analysis for the completely randomized one factor experiment with the compressive strength of concrete data.

```
>> strengthMod=fitlm(technique,strength,'CategoricalVars',[1]);
>> plotResiduals(strengthMod,'fitted','ResidualType','raw')
```

Figure 9.1 shows plots of the raw residuals against the fitted values.

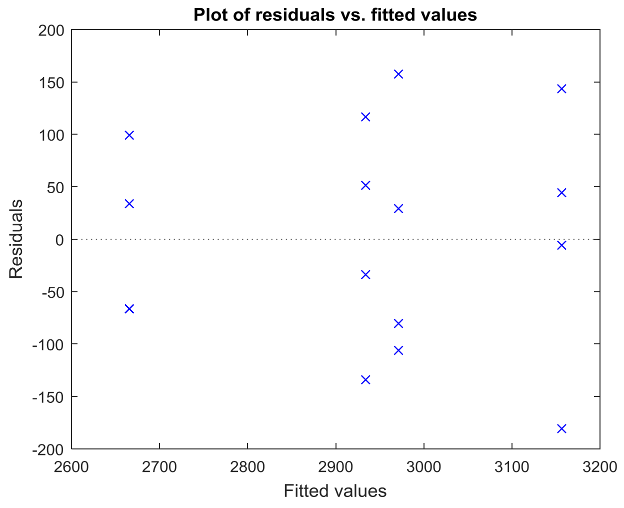


Figure 9.1: Plot of raw residuals against mixing technique (left) and against fitted values (right).

There doesn't seem to be any reason to question model assumptions here. With only four observations from each population it is difficult to say too much from the plots.

Example: blonds are tougher

Consider the pain and hair colour dataset, where a number of subjects were divided into groups based on hair colour (light blond, dark blond, light brunette or dark brunette). Then a pain threshold score was measured for each subject. We did a one way analysis of variance for these data and concluded that there were significant differences between means for some of the groups (pain threshold and hair colour seem to be related).

The plots below show the residuals against group (1=light blond, 2=dark blond, 3=light brunette and 4=dark brunette) and residuals against fitted values for this example. What are your conclusions? Are the assumptions we made in conducting the one way analysis of variance reasonable?

```
>> painMod=fitlm(group,pain,'CategoricalVars',[1]);
>> plotResiduals(painMod,'fitted','ResidualType','Studentized')
```

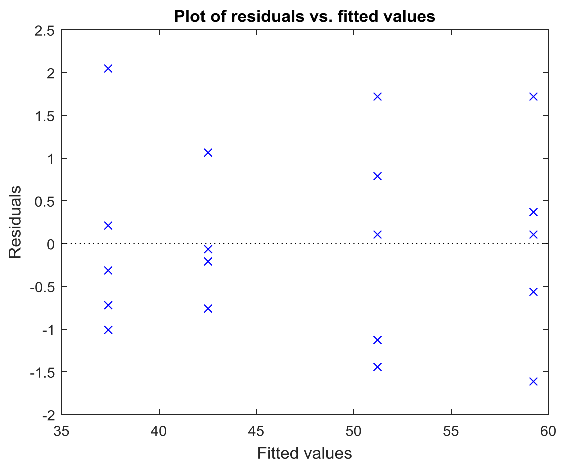


Figure 4: Plot of raw residuals against hair colour (left) and fitted values (right) for pain and hair colour example.