Loading, summarizing and visualizing data

Contents

1 Loading data

R allows loading data from files of many different formats. The file may reside locally or may be specified as a URL.

1.1 From a local text file

We shall consider part of a data set given in

> Distance to the Large Magellanic Cloud: The RR Lyrae Stars Gisella Clementini, Raffaele Gratton, Angela Bragaglia, Eugenio Carretta, Luca Di Fabrizio, and Marcella Maio Astronomical Journal 125, 1309-1329 (2003).

We have slightly doctored the data file to make it compatible with R. The file is called LMC.dat and resides in somewhere in the local machine. The data set has two columns with the headings `Method`, `Dist` and `Err`. Here are the first few lines of the file:

```
Method                   Dist        Err
"Cepheids: trig. paral."  18.70     0.16
"Cepheids: MS fitting"    18.55     0.06
"Cepheids: B-W"          18.55     0.10
```

Note the following points:

- The first line contains the variable names.
- Character strings with spaces in them are surrounded by quotes.
- There is a single case per line.
- Each line consists of the same number of fields.

For the next R command to succeed you need to download the file

`http://astrostatistics.psu.edu/datasets/LMCmod.txt`

onto your desktop.
The R command to load the data set is

```
LMC = read.table("v:/Desktop/LMCmod.txt", header=T)
```

Note the use of forward slash (`/`) even if you are working in Windows. Also the option `header=T` tells that the first line of the data file gives the names of the columns. Here we have used the *absolute path* of the data file. It is also possible to use relative paths.

```
dim(LMC)
names(LMC)
summary(LMC)
LMC
```

Read the help of `read.table` carefully to learn about the many options of this versatile function.
Here the object `LMC` is like a matrix (more precisely it is called a **data frame**). Each column stores the values of one variable, and each row stores a case. Its main difference with a matrix is that different columns can hold different types of data (for example, the `Method` column stores character strings, while the other two columns hold numbers). Otherwise, a data frame is really like a matrix.

1.2 Reading from a remote text file

Most of the data sets used in this tutorial resides in the internet. We can download them and then load them into R as local files. But a smarter way is to ask R to download the data set.
We shall read the SDSS quasar sample.

```
qso =
read.table("http://astrostatistics.psu.edu/datasets/SDSS_QSO.dat",
head=T) #oops!
```

R says:

```
Warning message:
In scan(file, what, nmax, sep, dec, quote, skip, nlines, na.strings,  :
  number of items read is not a multiple of the number of columns
```

The problem is that R expects the data set to be in a rectangular format (same number of values in each line). But thanks to missing values, this data set does not have this format. Here is an example of such a file.

```
X Y Z
1 2 3
2   5
2 4 5
2
  2 4
```

Ideally all missing values should be replaced by the symbol `NA` like this.

```
X Y   Z
1 2   3
2 NA  5
2 4   5
2 NA  NA
NA 2  4
```

Surely it is not always an easy task to do this by hand. In many situations, however, we just want to omit the lines containing any missing value. Then we can use the `fill=T` option:

```
qso =
read.table("http://astrostatistics.psu.edu/datasets/SDSS_QSO.dat",
head=T,fill=T)
```

This causes each deficient line to be padded with NA's *from the right*. Thus the example data set becomes

```
X Y   Z
1 2   3
2 5   NA
2 4   5
2 NA  NA
2  4 NA
```

This is not the same as the original data set, but if we remove the lines with NA's then the remaining part matches the corresponding part of the original data set. This removal is achieved with the function na.omit:

```
qso = na.omit(qso)
```

This is what we shall do most of the time in this course.

1.3 Reading files of other formats

1.3.1 Comma-Separated-Values (.csv)

R can also read from the comma-separated-values (csv) format. The example data set mentioned above will look like the following in the csv format.

```
X, Y,   Z
1, 2,   3
2,  ,   5
2, 4,   5
2,  ,
 , 2,   4
```

The function read.csv reads from a file of this format.

1.3.2 FITS

The FITSio package can read from files in the Flexible Image Transport System (FITS) format. FITS is a standard format to store the image obtained from a telescope. The package allows both reading and writing FITS files. Here is a typical example (adapted from the online help of the FITSio package):

```
install.packages('FITSio')
library(FITSio)

Z = matrix(1:15, ncol = 3) #Just create a matrix.
```

```
writeFITSim(Z, file = "test.fits") #Dump the matrix
                                   #as a FITS image file.
X = readFITS("test.fits") #Read the image back
X
```

You'll notice that `X` has much more information in it than `Z` ever had. This is because a FITS file has slots for many pieces of information, which R has filled in with default values. To understand the contents of `X` fully you need to know the FITS format.

1.3.3 Excel, SPSS, SAS etc

Sometimes we may need to import files that are saved by some standard statistical softwares like MS-Excel, SPSS, SYSTAT or SAS. These files are typically binary in nature, and use proprietary formats. The `foreign` package of R (included in the core distribution) has functions suitable to read from these files. We shall not need them in this course.

2 Summarizing data

Now that we know how to load data into R, it is time to learn how to quickly summarize the data.
We shall illustrate this with the LMC data set that we loaded just now.
We can find the mean of the variable `Dist` like this:

```
mean(LMC[,2])
mean(LMC,"Dist"])
```

Note that each column of the `LMC` matrix is a variable, so it is tempting to write

```
mean(Dist) #Oops!
```

but this will not work, since `Dist` is inside `LMC`. There are a couple of ways to "bring it out". A neat way is

```
with(LMC, mean(Dist))
```

A more messy way (requiring less typing, though) is

```
attach(LMC)
```

This causes each column of `LMC` to become a separate variable. Now the command

```
mean(Dist)
```

works perfectly. The irritating aspect of `attach` is that it floods the workspace with a large number of variables, which may often "mask" older variables with the same name. A typical scenario is where we work with a data set, and then work with a subset of the same data set.

```
LMC1 = LMC[1:5,] #Subset consisting of the first 5 cases
attach(LMC1)
```

R says:

```
The following object(s) are masked from 'LMC':

    Dist, Err, Method
```

One way to avoid the mess is to use the `detach` function to cancel the effect of an attach.
All the values of the `Dist` variable are different measurements of the same distance. So it is only natural to use the average as an estimate of the true distance. But the `Err` variable tells us that not all the measurements are equally reliable. So a better estimate might be a weighted mean, where the weights are inversely proportional to the errors.
We can use R as a calculator to directly implement this formula:

```
sum(Dist/Err)/sum(1/Err)
```

or you may want to be a bit more explicit

```
wt = 1/Err
sum(Dist*wt)/sum(wt)
```

Actually there is a smarter way than both of these.

```
weighted.mean(Dist, 1/Err)
```

It is quite easy to compute standard deviation, variance, covariance and correlation between numeric variables.

```
median(Dist)

sd(Dist)
var(Dist)
var(Err)
mad(Dist)

cov(Dist,Err)
cor(Dist,Err)
```

3 Visualizing data

3.1 1-dimensional

Let us load an asteroids data set.

```
asteroids = read.table(
"http://astrostatistics.psu.edu/MSMA/datasets/asteroid_dens.dat",
header=T)
astnames = asteroids[,1]
dens = asteroids[,2]
err = asteroids[,3]
```

Next we make a `dotchart` and a `boxplot`. A boxplot gives an idea about the centre, spread as well as skewness of a sample. It also highlights outlying points.

```
dotchart(dens, labels=astnames, cex=0.9, xlab='Density (g/cm^3)')

boxplot(asteroids[,2:3], varwidth=T,
        xlab="Asteroids", ylab="Density",
        pars=list(boxwex=0.3,
                  boxlwd=1.5,
                  whisklwd=1.5,
                  staplelwd=1.5,
```

```
                outlwd=1.5,
                font=2))
```

A simple histogram created by the `hist` function is another useful plot.

```
# Read Milky Way Galaxy and M 31 globular cluster K magnitudes
GC_MWG = read.table(
'http://astrostatistics.psu.edu/MSMA/datasets/GlobClus_MWG.dat',
header=T)
KGC_MWG = GC_MWG[,2] ;
kseq = seq(-15.0, -5.0, 0.25)

hist(KGC_MWG, breaks=kseq, ylim=c(0,10), main='',
    xlab='K mag', ylab='N')
```

We shall talk more about histograms later.

3.1.1 Saving a plot

The Windows version of R allows the user to save a plot by right-clicking on a graphics window, and selecting "Save as..". R allows saving in different formats:

- image format like png, jpg etc,
- vector formats like postscript (version 3), pdf,
- editable formats like xfig.

It is also possible to save an image from the R prompt (this works for all platforms).

```
postscript("path/to/myplot.ps") #both absolute and relative
                                #paths are allowed.
hist(KGC_MWG) #No plot is shown on screen.
dev.off() #This line is important:
          #It completes and closes the postscript file,
          #and directs subsequent plots back to the screen.
```

There is a function called `dev.copy2eps` that copies the plot screen to an Encapsulated Postscript (eps) file. It works well on Mac, but fails to copy the entire plot in Windows or Linux.

Exercise 1: Look up the helps of the functions `postscript`, `pdf`, `png` and `xfig` to get an idea of the various options. In particular, learn how one can create multiple image files with a single call to these functions. ■

3.1.2 Multiple plots per page

```
GC_M31 = read.table(
'http://astrostatistics.psu.edu/MSMA/datasets/GlobClus_M31.dat',
header=T)

KGC_M31 = GC_M31[,2]-24.44 #The 24.44 is just a shift to make
                           #KGC_M31 comparable to GC_M31. We
                           #shall learn later how such shifts
                           #are estimated statistically.

par(mfrow=c(1,2))
hist(KGC_MWG, breaks=kseq, ylim=c(0,10), main='',
    xlab='K mag', ylab='N')
hist(KGC_M31, breaks=kseq, ylim=c(0,50), main='',xlab='K
mag',ylab='N')
par(mfrow=c(1,1))
```

Here we have used the `par` function, which sets global graphics parameters in R. There is a long list of such parameters to produces widely different types of customized plots. Spending time to carefully go through the list is essential before producing a plot for publication.

Exercise 2: Thoroughly read the help page for the `par` function. ■

3.1.3 Multiple plot windows

When we issue a command like `plot` R checks if there is a graphics window already open. If it is, then the plot is displayed there, else a new graphics window is created for the plot. It is possible for the user to explicitly create a new graphics window. The command is

```
dev.new()
```

Only one of the graphics windows is active at a time, and all subsequent plotting commands direct their output to it. By default, the most recently created graphics window is the active one. However we can use the function `dev.set` to activate some other graphics window.

```
hist(KGC_MWG, breaks=kseq, ylim=c(0,10), main='',
    xlab='K mag', ylab='N')
dev.set(??) #Replace ?? by the no. of the first window.
```

```
hist(KGC_M31, breaks=kseq, ylim=c(0,50), main='',xlab='K
mag',ylab='N')
```

3.2 2-dimensional

The simplest form of 2d plot is a scatterplot.

```
# Construct and plot data set of SDSS quasars with 18<i<22
qso = read.table(
'http://astrostatistics.psu.edu/MSMA/datasets/SDSS_17K.dat',
header=T,fill=T)
qso = na.omit(qso)
summary(qso)
qso1 = qso[((qso[,6]<22) & (qso[,6]>18)),]
dim(qso1) ; summary(qso1) ; attach(qso1)
attach(qso1)
Err_z[which(Err_z<=0.03)] = 0.03
plot(i_mag, z_mag, pch=20, cex=0.1, col=grey(0.5),
   xlab="SDSS i (mag)", ylab="SDSS z (mag)")
```

Sometimes we need error bars. These may be added as segments.

```
errplot = function(x,y,xerrlo,yerrlo,xerrhi=xerrlo,yerrhi=yerrlo,...) {
   plot(x,y,xlim=range(x-xerrlo,x+xerrhi),
            ylim=range(y-yerrlo,y+yerrhi),...)
   segments(x,y-yerrlo,x,y+yerrhi)
   segments(x-xerrlo,y,x+xerrhi,y)
}
```

Let's use it.

```
errplot(i_mag, z_mag, Err_i, Err_z, pch=20, cex=0.1, col=grey(0.5),
   xlab="SDSS i (mag)", ylab="SDSS z (mag)")
```

Exercise 3: Can you modify the function so that we can suppress either the horizontal or vertical errorbars? ■

The `segments` function allows addition of line segments to a plot. There are a variety of similar functions that embellish an existing plot. We discuss these next.

3.2.1 Embellishing a plot

There are basically two types of plot commands:

- those that create a fresh plot
- those that add things to the current plot

Commands like `plot`, `boxplot`, `hist` belong to first category, while the function `segments` is a member of the second. Some examples of the second category are

- `lines`: draws a polyline. Useful for overlaying a graph on an existing plot.
- `segments`: draws disjoint line segments.
- `abline`: draws a single straight line given either as $y = a + bx$ or a horizontal/vertical line. The line goes from margin to margin. Useful for linear regression.
- `points`: adds some points.
- `arrows`: draws a sequence of arrows.
- `polygon`: like the `lines` function, joins the two extremes. The region inside the polygon can be shaded. Useful for shading parts under a graph.
- `rect`: draws a rectangle. Useful for highlighting a region in a scatterplot.
- `text`: adds a label to any given point inside a plot. Don't use for creating title or labelling the axes, though. Use the `title` function for those purposes.
- `legend`: adds a legend.
- `title`: adds a headings and axis labels.

We shall see some of these in action later. Let us see how we can overlay a curve on a histogram.

```
hist(KGC_MWG, breaks=kseq, main='', xlab='K mag',ylab='N',
     col=gray(0.5),prob=T)
lines(kseq, dnorm(kseq, mean=-10.32,sd=1.79))
```

Exercise 4: Just as the help for the `par` function contains an annotated list of useful (and not so useful) graphical parameters, the help for the `points` function has a list of types of point markers. The developers of R has the not-too-commendable habit of using numbers to denote various markers (instead of mnemonics like 'circle', 'diamond' etc). Read through the help for `points` to familiarize yourself with the list. ■

There is a function `curve` for plotting a mathematical curve:

```
curve(sin,-pi,pi)
```

One nice feature about this is that it may be used to create either a fresh plot or add to an existing one.

```
curve(cos(x),-pi,pi,col='red',add=T)
```

R allows Greek symbols in titles and labels via the `expression` function:

```
curve(cos(x),-pi,pi,xlab=expression(theta),
                    ylab=expression(paste("cos ",theta)))
```

3.2.2 The `pairs` plot

To visualize a multivariate data set it often helps to see the scatterplots of all possible pairs of variables in a single page. Repeatedly invoking the `plot` function is a tedious way to achieve this. A niftier way is via the `pairs` function.

```
# Overview of Webbink globular cluster properties
GC = read.table(
"http://astrostatistics.psu.edu/MSMA/datasets/Webbink_GC_tab.txt",
header=T,fill=T)

GC = na.omit(GC)
GCd = GC[,-(1:4)] #get rid of the first 4 columns,
                      #which store location info

GC = scale(GC) #subtract mean and divide by sd (separately for
                #each variable).



# Bivariate relationships
pairs(GC,pch=20,cex=0.1)
```

Well there is one very large value in each of columns 4 and 11, which dwarfs the remaining values. So let's get rid of these maximums and produce a fresh `pairs` plot.

```
GC1 = GC[-c(which.max(GC[,4]), which.max(GC[,11])),]
pairs(GC1,pch=20,cex=0.1)
```

3.2.3 Interactivity

Sometimes we need to select points falling inside a rectangle of our choice from a bivariate data. Imagine picking the Hyades stars from the Hipparcos data set using the RA and declination values.
The `locator` function is ideally suited for this. It is the only function in R that recognizes the mouse. The function changes the cursor to a cross-hair, waits for mouse clicks, and returns the clicked coordinates.

```
pickRect = function(coords) {
  plot(coords)
  cat("Pick a rectangle.\n")
  cat("First click on its bottom left corner.\n")
  bl = locator(1) #Get 1 click
  cat("Now click on its top right corner.\n")
  tr = locator(1) #Get another click
  rect(bl$x,bl$y,tr$x,tr$y)
  ind = coords[,1] > bl$x &
        coords[,1] < tr$x &
        coords[,2] > bl$y &
        coords[,2] < tr$y
  ind
}
```

Let's now put it to use.

```
hip = read.table(
'http://astrostatistics.psu.edu/MSMA/datasets/HIP.dat',
head=T,fill=T)
hip = na.omit(hip)
names(hip)
ind = with(hip,pickRect(cbind(RA,DE)))
ind
sum(ind) #number of stars selected
```

Exercise 5: What happens if you invoke `locator()` without any argument? See the answer if you get "stuck"! ■

3.2.4 Images and contour plots

Here is an example of a more exotic plot. A scatterplot gives a rough idea about the distribution of the points. But if there are too many points then the plot just becomes a big black blot of ink. The next example shows a way out: by binning the data and then plotting the frequencies of the bin as an image. We show this with the SDSS quasar data set.

```
qso = na.omit(
  read.table(
   "http://astrostatistics.psu.edu/datasets/SDSS_QSO.dat",
   head=T,fill=T))
q1 = qso[qso[,10] < 0.3,] #We do some filtering
q1 = q1[q1[,12]<0.3,]

dim(q1) ; names(q1) ; summary(q1)

r_i = with(q1, r_mag - i_mag) #Create the variables
z = q1[,4]
```

Next we need to install and load an external package called `ash`. No, this has nothing to do with the residue from a recent burning. Here `ash` stands for **Average Shifted Histogram**. To understand this imagine drawing a histogram with bin widths equal to 1, say. Let's say the bins are [0,1], [1,2] etc. Now construct a new histogram from the same data but with shifted bins [0.1,1.1], [1.1, 2.1] etc. Repeat this shifting process 9 times (if you shift one more time then you basically get back the original histogram). If you now average all these histograms, you get a smoothed histogram-like object which we call an average shifted histogram.
The same idea can also be carried out in 2 dimensions. Both the 1- and 2-dimensional versions are implemented in the package `ash`. We shall illustrate the 2-dimensional version here.

```
install.packages('ash')
library(ash)
```

Now comes the actual step.

```
nbin = c(500, 500) #500 bins along either axis
ab = matrix(c(0.0,-0.5,5.5,2.), 2,2) #range of axes
bins = bin2(cbind(z,r_i), ab, nbin) #find the bins
f = ash2(bins, c(5,5)) #find frequencies
```

```
names(f)
```

You'll notice that `f` has three fields called `x`, `y` and `z`. The first two identify a bin, and the last is the frequency of the bin. Here the frequencies have a wide range of variation (some bins are overcrowded, while some have hardly any point). So we take log to reduce the disparity.

```
logfz = log10(f$z) #taking log reduces the scale
```

There are various ways to depict a large matrix like this: as an image or as a contour or as a surface.

```
image(f$x, f$y, logfz, col=gray(seq(0.2,0.5,by=0.05)), zlim=c(-1.8,3.0),
   main="SDSS quasars", xlab="Redshift", ylab="r-i")
dev.new()
contour(f$x, f$y, logfz, zlim=c(0.1,5) ,nlevels=5)
```

The third method (plotting a surface) takes us into 3-dimensional visualization of data.

3.3 3-dimensional

R's graphical capabilities are mostly suited for 2-dimensional visualization. For 3-dimensional depiction of data we have to mostly rely on external packages. Before discussing them we shall talk some simple techniques available in R itself.

3.4 Simple techniques

The `persp` function plots 3-dimensional surfaces. We could replace the `image` or `contour` plots in our example above with this function as follows.

```
persp(f$x,f$y,f$z)
```

Well, this does not look too inviting! You may try playing with the parameters `phi` and `theta` to rotate the plot.
A simple way is to make a 2-dimensional plot, where the size of the symbol is scaled according to the 3-rd dimension. We illustrate the technique with a galaxy redshift survey of the Shapley Supercluster region (Drinkwater et al. 2004). The data set has 4215 galaxies over about 100 square degrees with five variables:

- two spatial dimensions, right ascension and declination
- optical magnitude which is an inverse logarithmic measure of a galaxy's mass
- radial velocity (equivalent to redshift) which is a distorted measure of the third dimension, distance
- the measurement error of the radial velocity.

First we load the data in R:

```
shap=read.table(
'http://astrostatistics.psu.edu/MSMA/datasets/Shapley_galaxy.dat',
header=T,fill=T)
shap = na.omit(shap)
dim(shap)
summary(shap)
```

Now let's do the plotting.

```
with(shap,plot(R.A.,Dec.,cex=3*Vel/(max(Vel)-min(Vel))))
```

A much better solution is provided by the `scatterplot3d` package.

```
install.packages('scatterplot3d')
library(scatterplot3d)
scatterplot3d(shap[,c(1,2,4)], pch=20, cex.symbols=0.7, type='p', angl=40)
```

However, the `scatterplot3d` package provides no interactivity. The package that we discuss next is free of this stigma.

3.5 The `rgl` package

This package is based on openGL.

```
install.packages('rgl')
library(rgl)
rgl.open() #A little square grey window opens
with(shap,plot3d(scale(R.A.),scale(Dec.), scale(Vel)))
```

Notice how we are using the `scale` function to subtract the mean and divide by the standard deviation. The resulting plot can be rotated with the left mouse button, and zoomed with the middle one. We can save a 2-dimensional snapshot of the plot as follows.

```
rgl.snapshot('shaprgl.png')
rgl.close()
```

4 Hints for selected exercises

3.

```
errplot =
function(x,y,
         xerrlo=NULL,yerrlo=NULL,
         xerrhi=xerrlo,yerrhi=yerrlo,...) {
   if(!is.null(xerrlo)) {
     xrange = range(x-xerrlo,x+xerrhi)
   }
   else {
     xrange = range(x)
   }

   if(!is.null(yerrlo)) {
     yrange = range(y-yerrlo,y+yerrhi)
   }
   else {
     yrange = range(y)
   }

   plot(x,y,xlim=xrange, ylim=yrange,...)
   if(!is.null(yerrlo)) segments(x,y-yerrlo,x,y+yerrhi)
   if(!is.null(xerrlo)) segments(x-xerrlo,y,x+xerrhi,y)
}
errplot(i_mag, z_mag, yerrlo=Err_z)
errplot(i_mag, z_mag, xerrlo=Err_i)
```

5. It goes on registering mouse clicks indefinitely. To stop it you have to right-click on the graphics window, and select "Stop". Then it will return an $n \times 2$

matrix, where n is the number of clicks. The first column stores the x-values, the second column the y-values.