A brief introduction to R

ASTRO 585  Fall 2011
The R statistical computing environment

- R is the public-domain version of the commercial S-Plus statistical computing package. Integrates data manipulation, graphics and statistical analysis. Uniform documentation and coding standards.

- Fully programmable C-like language, similar to IDL. Specializes in vector or matrix inputs; not designed for maps, images or movies.

- Easily downloaded from http://www.r-project.org with Windows, Mac or UNIX binaries.

- Tutorials available in dozens of books (most since 2005) and on-line.

- >2600 user-provided add-on packages collected in Comprehensive R Archive Network http://www.cran.r-project.org.
Some functionalities of R

Base R
arithmetic & linear algebra, bootstrap resampling, empirical distribution tests, exploratory data analysis, generalized linear modeling, graphics, robust statistics, linear programming, local and ridge regression, maximum likelihood estimation, multivariate analysis, multivariate clustering, neural networks, smoothing, spatial point processes, statistical distributions & random deviates, statistical tests, survival analysis, time series analysis

Selected methods from Comprehensive R Archive Network (CRAN)
Bayesian computation & MCMC, classification & regression trees, genetic algorithms, geostatistical modeling, hidden Markov models, irregular time series, kernel-based machine learning, least-angle & lasso regression, likelihood ratios, map projections, mixture models & model-based clustering, nonlinear least squares, multidimensional anlaysis, multimodality test, multivariate time series, multivariate outlier detection, neural networks, non-linear time series analysis, nonparametric multiple comparisons, omnibus tests for normality, orientation data, parallel coordinates plots, partial least squares, periodic autoregression analysis, principal curve fits, projection pursuit, quantile regression, random fields, random forest classification, ridge regression, robust regression, self-organizing maps, shape analysis, space-time ecological analysis, spatial analysis & kriging, spline regressions (MARS, BRUTO), tessellations, three-dimensional visualization, wavelet toolbox
Interfaces: BUGS, C, C++, Fortran, Java, Perl, Python, Xlisp, XML

I/O: ASCII, binary, bitmap, cgi, FITS, ftp, gzip, HTML, SOAP, URL

Graphics & emulators: Grace, GRASS, Gtk, Matlab, OpenGL, Tcl/Tk, Xgobi

Math packages: GSL, Isoda, LAPACK, PVM

Text processor: LaTeX

Since c.2002, R has been the premier public-domain statistical computing package.
Growth of CRAN contributed packages

Sample R Scripts

# Read dataset of 120 SDSS quasar r magnitudes
Qso <- read.table("http://astrostatistics.psu.edu/datasets/SDSS QSO.dat",
dim(qso) ; names(qso) ; summary(qso)
Rmag <- qso[1:120,9]
Amag <- qso[1:120,17]

# Plot e.d.f. with confidence bands
install.packages('sfsmisc') ; library('sfsmisc')
ecdf.ksCI(rmag)

# Plot e.d.f.'s
wilcoxon.test(rmag,amag,conf.int=T)
Absmag=amag+44.7 # sets equal medians
plot(ecdf(rmag),pch=20,xlab="Magnitude")
plot (ecdf(Absmag),add=T) ; text(21,0.7,lab='r mag')
# Run e.d.f. 2-sample tests
ks.test(rmag, Absmag)
install.packages('cramer') ; library(cramer)
cramer.test(rmag, Absmag)

# Plot histograms and kernel density estimators
hist(rmag, breaks='scott') ; hist(rmag, breaks=30)
plot(density(rmag, bw=bw.nrd0(rmag)))

# Plot k.d.e. with condence bands
install.packages('sm') ; library(sm)
help('sm.density')
sm.density(rmag) ; tt <- sm.density(rmag)
lines(tt$eval.points, tt$upper, col=3) ;
lines(tt$eval.points, tt$lower, col=3)
A complicated R graphic
Selected books on R

Introductory Statistics with R  P. Dalgaard
Software for data analysis: Programming with R, J Chambers 2008
Bayesian Computation with R  J. Albert 2\textsuperscript{nd} ed, 2009
Introductory Time Series with R Cowpertwait & A. V. Metcalfe 2009
ggplot2: Elegant Graphics for Data Analysis H. Wickham 2\textsuperscript{nd} ed, 2009
Lattice: Multivariate Data Visualization with R  D. Sarkar 2008
The R Book, Michael Crawley 2007 (900p)
R in a nutshell: A desktop quick reference J Adler 2009 (600p)

+ ~1 additional arriving monthly at Penn State’s libraries
Part I: Introduction to R

R is the most comprehensive public-domain statistical software available today. An implementation of the S language for statistical analysis, it is increasingly popular in the statistical community. It takes only a few minutes to obtain: go to [http://www.R-project.org](http://www.R-project.org), choose a CRAN mirror site, and download the files appropriate for your operating system.

The following tutorial is a series of excerpts from a weeklong set of tutorials presented at the Summer School in Statistics for Astronomers and Physicists at Penn State University and at Kavalur, India. When this demo is viewed online, the bolded, indented lines

# like this one

are meant to be copied and pasted directly into R at the command prompt.

Enter R by typing "R" (UNIX) or double-clicking to execute Rgui.exe (Windows) or R.app (Mac). In the commands below, we start by extracting some system and user information, the R version you are using, and some of its capabilities, citation tells how to cite R in publications. R is released under the GNU Public Licence, as indicated by copyright. Typing a question mark in front of a command opens the help file for that command.

```r
Sys.info()
R.version
capabilities()
citation()
?copyright
```

The various capitalizations above are important as R is case-sensitive. When using R interactively, it is very helpful to know that the up-arrow key can retrieve previous commands, which may be edited using the left-and right-arrow keys and the delete key.

The last command above, ?copyright, is equivalent to help(copyright) or help("copyright"). However, to use this command you have to know that the function called "copyright" exists. Suppose that you knew only that there was a function in R that returned copyright information but you could not remember what it was called. In this case, the help.search function provides a handy reference tool:

```r
help.search("copyright")
```

Last but certainly not least, a vast array of documentation and reference materials may be accessed via a simple command:

```r
help.start()
```

The initial working directory in R is set by default or by the directory from which R is invoked (if it is invoked on the command line). It is possible to read and set this working directory using the getwd or setwd commands. A list of the files in the current working directory is given by list.files, which has a variety of useful options and is only one of several utilities interfacing to the computer's files. In the setwd command, note that in Windows, path (directory) names are not case-sensitive and may contain either forward slashes or backward slashes; in the latter case, a backward slash must be written as "\" when enclosed in quotation marks.

```r
getwd()
list.files() # what's in this directory?
# The # symbol means that the rest of that line is a comment.
```

Reading data into R

In the course of learning a bit about how to generate data summaries in R, one will inevitably learn some useful R syntax and commands. Thus, this first part on descriptive statistics serves a dual role as a brief introduction to R.

We wish to read an ASCII data file into an R object using the read.table command or one of its variants. Let's begin with a cleaned-up version of the Hipparcos dataset described above, a description of which is
given at [http://astrostatistics.psu.edu/datasets/HIP_star.html](http://astrostatistics.psu.edu/datasets/HIP_star.html). There are two distinct lines below that read the dataset and create an object named hip. The first (currently commented out) may be used whenever one has access to the internet; the second assumes that the HIP_star.dat file has been saved into the current working directory.

```r
#   hip  <-  read.table("http://astrostatistics.psu.edu/datasets/HIP_star.dat",
#      header=T,fill=T) # T is short for TRUE
hip  <-  read.table("HIP_star.dat", header=T,fill=T)
```

The "<-", which is actually "less than" followed by "minus", is the R assignment operator. Admittedly, this is a bit hard to type repeatedly, so fortunately R also allows the use of a single equals sign (=) for assignment.

Note that no special character must be typed when a command is broken across lines as in the example above. Whenever a line is entered that is not yet syntactically complete, R will replace the usual prompt, ">" with a + sign to indicate that more input is expected. The `read.table` function can refer to a location on the web, though a filename (of a file in the working directory) or a pathname would have sufficed. The "header=TRUE" option is used because the first row of the file is a header containing the names of the columns. We used the "fill=TRUE" option because some of the columns have only 8 of the 9 columns filled, and "fill=TRUE" instructs R to fill in blank fields at the end of the line with missing values, denoted by the special R constant `NA` ("not available"). **Warning**: This only works in this example because all of the empty cells are in the last column of the table. (You can verify this by checking the ASCII file `HIP_star.dat`.) Because the `read.table` function uses delimiters to determine where to break between columns, any row containing only 8 values would always put the `NA` in the 9th column, regardless of where it was intended to be placed. As a general rule, data files with explicit delimiters are to be preferred to files that use line position to indicate column number, particularly when missing data are present. If you must use line position, R provides the `read.fortran` and `read.fwf` functions for reading fixed width format files.

### Summarizing the dataset

The following R commands list the dimensions of the dataset and print the variable names (from the single-line header). Then we list the first row, the first 20 rows for the 7th column, and the sum of the 3rd column.

```r
dim(hip)
names(hip)
hip[1,]
hip[1:20,7]
sum(hip[,3])
```

Note that vectors, matrices, and arrays are indexed using the square brackets and that "1:20" is shorthand for the vector containing integers 1 through 20, inclusive. Even punctuation marks such as the colon have help entries, which may be accessed using `help(":")`.

Next, list the maximum, minimum, median, and median absolute deviation (similar to standard deviation) of each column. First we do this using a `for`-loop, which is a slow process in R. Inside the loop, `c` is a generic R function that combines its arguments into a vector and `print` is a generic R command that prints the contents of an object. After the inefficient but intuitively clear approach using a `for`-loop, we then do the same job in a more efficient fashion using the `apply` command. Here the "2" refers to columns in the x array; a "1" would refer to rows.

```r
for(i in 1:ncol(hip)) {
  print(c(max(hip[,i]), min(hip[,i]), median(hip[,i]), mad(hip[,i])))
}
apply(hip, 2, max)
apply(hip, 2, min)
apply(hip, 2, median)
apply(hip, 2, mad)
```

The curly braces `{}` in the for loop above are optional because there is only a single command inside. Notice that the output gives only NA for the last column's statistics. This is because a few values in this column are missing. We can tell how many are missing and which rows they come from as follows:

```r
sum(is.na(hip[,9]))
which(is.na(hip[,9]))
```

There are a couple of ways to deal with the NA problem. One is to repeat all of the above calculations on a
new R object consisting of only those rows containing no NAs:

```r
y <- na.omit(hip)
for(i in 1:ncol(y)) {
  print(c(max(y[,i]), min(y[,i]), median(y[,i]), mad(y[,i])))
}
```

Another possibility is to use the na.rm (remove NA) option of the summary functions. This solution gives slightly different answers from the the solution above; can you see why?

```r
for(i in 1:ncol(hip)) {
  print(c(max(hip[,i],na.rm=T), min(hip[,i],na.rm=T), median(hip[,i],na.rm=T), mad(hip[,i],na.rm=T)))
}
```

A vector can be sorted using the Shellsort or Quicksort algorithms; rank returns the order of values in a numeric vector; and order returns a vector of indices that will sort a vector. The last of these functions, order, is often the most useful of the three, because it allows one to reorder all of the rows of a matrix according to one of the columns:

```r
sort(hip[1:10,3])
hip[order(hip[1:10,3]),]
```

Each of the above lines gives the sorted values of the first ten entries of the third column, but the second line reorders each of the ten rows in this order. Note that neither of these commands actually alters the value of x, but we could reassign x to equal its sorted values if desired.

**More R syntax**

**Arithmetic** in R is straightforward. Some common operators are: + for addition, - for subtraction, * for multiplication, / for division, %/% for integer division, %% for modular arithmetic, ^ for exponentiation. The help page for these operators may be accessed by typing, say,

```r
?'+'
```

Some common built-in functions are exp for the exponential function, sqrt for square root, log10 for base-10 logarithms, and cos for cosine. The syntax resembles "sqrt(z)". Comparisons are made using < (less than), <= (less than or equal), == (equal to) with the syntax "a >= b". To test whether a and b are exactly equal and return a TRUE/FALSE value (for instance, in an "if" statement), use the command identical(a,b) rather a==b. Compare the following two ways of comparing the vectors a and b:

```r
a <- c(1,2);b <- c(1,3)
a==b
identical(a,b)
```

Also note that in the above example, 'all(a==b)' is equivalent to 'identical(a,b)'.

R also has other logical operators such as & (AND), | (OR), ! (NOT). There is also an xor (exclusive or) function. Each of these four functions performs elementwise comparisons in much the same way as arithmetic operators:

```r
!a
a & b
a | b
xor(a,b)
```

However, when 'and' and 'or' are used in programming, say in 'if' statements, generally the '&&' and '||' forms are preferable. These longer forms of 'and' and 'or' evaluate left to right, examining only the first element of each vector, and evaluation terminates when a result is determined. Some other operators are listed here.

The expression "y == x^2" evaluates as TRUE or FALSE, depending upon whether y equals x squared, and performs no assignment (if either y or x does not currently exist as an R object, an error results).

Let's continue with simple characterization of the dataset: find the row number of the object with the smallest value of the 4th column using which.min. A longer, but instructive, way to accomplish this task creates a long vector of logical constants (tmp), mostly FALSE with one TRUE, then pick out the row with
TRUE.

```r
which.min(hip[,4])
tmp <- (hip[,4]==min(hip[,4]))
{i:nrow(hip)}[tmp] # or equivalently,
which(tmp)
```

The `cut` function divides the range of `x` into intervals and codes the values of `x` according to which interval they fall. It is a quick way to group a vector into bins. Use the "breaks" argument to either specify a vector of bin boundaries, or give the number of intervals into which `x` should be cut. Bin string labels can be specified. `cut` converts numeric vectors into an R object of class "factor" which can be ordered and otherwise manipulated; e.g. with command `levels`. A more flexible method for dividing a vector into groups using user-specified rules is given by `split`.

```r
table(cut(hip[,"Plx"],breaks=20:25))
```

The command above uses several tricks. Note that a column in a matrix may be referred to by its name (e.g., "Plx") instead of its number. The notation '20:25' is short for 'c(20,21,22,23,24,25)' and in general, 'a:b' is the vector of consecutive integers starting with `a` and ending with `b` (this also works if `a` is larger than `b`). Finally, the `table` command tabulates the values in a vector or factor.

Although R makes it easy for experienced users to invoke multiple functions in a single line, it may help to recognize that the previous command accomplishes the same task as following string of commands:

```r
p <- hip[,"Plx"]
cuts <- cut(p,breaks=20:25)
table(cuts)
```

The only difference is that the string of three separate commands creates two additional R objects, `p` and `cuts`. The preferred method of carrying out these operations depends on whether there will later be any use for these additional objects.

### Univariate plots

Recall the variable names in the Hipparcos dataset using the `names` function. By using `attach`, we can automatically create temporary variables with these names (these variables are not saved as part of the R session, and they are superseded by any other R objects of the same names).

```r
names(hip)
attach(hip)
```

After using the `attach` command, we can obtain, say, individual `summaries` of the variables:

```r
summary(Vmag)
summary(B.V)
```

Next, summarize some of this information graphically using a simple yet sometimes effective visualization tool called a dotplot or dotchart, which lets us view all observations of a quantitative variable simultaneously:

```r
dotchart(B.V)
```

The shape of the distribution of the B.V variable may be viewed using a traditional histogram. If we use the `prob=TRUE` option for the histogram so that the vertical axis is on the probability scale (i.e., the histogram has total area 1), then a so-called kernel density estimate, or histogram smoother, can be overlaid:

```r
hist(B.V,prob=T)
d <- density(B.V,na.rm=T)
lines(d,col=2,lwd=2,lty=2)
```

The topic of density estimation will be covered later. For now, it is important to remember that even though histograms and density estimates are drawn in two-dimensional space, they are intrinsically univariate analysis techniques here: We are only studying the single variable B.V in this example (though there are multivariate versions of these techniques as well).

Check the help file for the `par` function (by typing "?par") to see what the `col`, `lwd`, and `lty` options accomplish in the `lines` function above.
A simplistic histogram-like object for small datasets, which both gives the shape of a distribution and displays each observation, is called a stem-and-leaf plot. It is easy to create by hand, but R will create a text version:

```r
stem(sample(B.V,100))
```

The sample command was used above to obtain a random sample of 100, without replacement, from the B.V vector.

Finally, we consider box-and-whisker plots (or "boxplots") for the four variables Vmag, pmRA, pmDE, and B.V (the last variable used to be B-V, or B minus V, but R does not allow certain characters). These are the 2nd, 6th, 7th, and 9th columns of hip:

```r
boxplot(hip[,c(2,6,7,9)])
```

Our first attempt above looks pretty bad due to the different scales of the variables, so we construct an array of four single-variable plots:

```r
par(mfrow=c(2,2))
for(i in c(2,6,7,9))
  boxplot(hip[,i],main=names(hip)[i])
par(mfrow=c(1,1))
```

The `boxplot` command does more than produce plots; it also returns output that can be more closely examined. Below, we produce boxplots and save the output.

```r
b <- boxplot(hip[,c(2,6,7,9)])
names(b)
'b' is an object called a list. To understand its contents, read the help for `boxplot`. Suppose we wish to see all of the outliers in the pmRA variable, which is the second of the four variables in the current boxplot:

```r
d turbhan:Ruser E5F> b$names[2]
b$out[b$group==2]
```

**Scatterplots**

Let us now examine the bivariate relationship between Vmag and B minus V using a scatter plot.

```r
plot(Vmag,B.V)
```

The above plot looks too busy because of the default plotting character, set let's use a different one:

```r
plot(Vmag,B.V,pch=".")
```

Let's now use exploratory scatterplots to locate the Hyades stars. This open cluster should be concentrated both in the sky coordinates RA and DE, and also in the proper motion variables pm_RA and pm_DE. We start by noticing a concentration of stars in the RA distribution:

```r
plot(RA,DE,pch=".")
```

See the cluster of stars with RA between 50 and 100 and with DE between 0 and 25?

```r
rect(50,0,100,25,border=2)
```

Let's construct a logical (TRUE/FALSE) variable that will select only those stars in the appropriate rectangle:

```r
filter1 <- (RA>50 & RA<100 & DE>0 & DE<25)
```

Next, we select based on the proper motions. (As our cuts through the data are parallel to the axes, this variable-by-variable classification approach is sometimes called Classification and Regression Trees or CART, a very common multivariate classification procedure.)

```r
plot(pmRA[filter1],pmDE[filter1],pch=20)
rect(0,-150,200,50,border=2)
```

Let's replot after zooming in on the rectangle shown in red.

```r
plot(pmRA[filter1],pmDE[filter1],pch=20, xlim=c(0,200),ylim=c(-150,50))
```
Let's have a final look at the stars we have identified using the `pairs` command to produce all bivariate plots for pairs of variables. We'll exclude the first and fifth columns (the HIP identifying number and the parallax, which is known to lie in a narrow band by construction).

```r
pairs(hip[filter,-c(1,5)],pch=20)
```

Notice that indexing a matrix or vector using negative integers has the effect of excluding the corresponding entries.

We see that there is one outlying star in the e_Plx variable, indicating that its measurements are not reliable. We exclude this point:

```r
filter <- filter & (e_Plx<5)
pairs(hip[filter,-c(1,5)],pch=20)
```

How many stars have we identified? The filter variable, a vector of TRUE and FALSE, may be summed to reveal the number of TRUE's (summation causes R to coerce the logical values to 0's and 1's).

```r
sum(filter)
```

As a final look at these data, let's consider the HR plot of Vmag versus B.V but make the 92 Hyades stars we just identified look bigger (pch=20 instead of 46) and color them red (col=2 instead of 1). This shows the Zero Age Main Sequence, plus four red giants, with great precision.

```r
plot(Vmag,B.V,pch=c(46,20)[1+filter], col=1+filter,
xlim=range(Vmag[filter]), ylim=range(B.V[filter]))
```

---

### R scripts

While R is often run interactively, one often wants to carefully construct R scripts and run them later. A file containing R code can be run using the `source` command. In addition, R may be run in batch mode.

The editor Emacs, together with "Emacs speaks statistics", provides a nice way to produce R scripts.

---

### Part II: Hypothesis testing and estimation

This part demonstrates a few of the many statistical tests that R can perform. It is impossible to give an exhaustive list of such testing functionality, but we hope not only to provide several examples but also to elucidate some of the logic of statistical hypothesis tests with these examples.

---

### T tests

Earlier, we used exploratory techniques to identify 92 stars from the Hipparcos data set that are associated with the Hyades. We did this based on the values of right ascension, declination, principal motion of right ascension, and principal motion of declination. We then excluded one additional star with a large error of parallax measurement:

```r
hip <- read.table("http://astrostatistics.psu.edu/datasets/HIP_star.dat",
 header=T, fill=T)
#  hip <- read.table("HIP_star.dat", header=T, fill=T)
attach(hip)
filter1 <- (RA>50 & RA<100 & DE>0 & DE<25)
filter2 <- (pmRA>90 & pmRA<130 & pmDE>-60 & pmDE< -10)
filter <- filter1 & filter2 & (e_Plx<5)
sum(filter)
```

In this section, we will compare these Hyades stars with the remaining stars in the Hipparcos dataset on the basis of the color (B minus V) variable. That is, we are comparing the groups in the boxplot below:
color <- B.V
boxplot(color~filter,notch=T)

For ease of notation, we define vectors H and nH (for "Hyades" and "not Hyades") that contain the data values for the two groups.

H <- color[filter]
nH <- color[!filter & !is.na(color)]

In the definition of nH above, we needed to exclude the NA values.

A two-sample t-test may now be performed with a single line:

t.test(H,nH)

Because it is instructive and quite easy, we may obtain the same results without resorting to the \texttt{t.test} function. First, we calculate the variances of the sample means for each group:

\begin{verbatim}
v1 <- var(H)/92
v2 <- var(nH)/2586
c(var(H),var(nH))
\end{verbatim}

The t statistic is based on the standardized difference between the two sample means. Because the two samples are assumed independent, the variance of this difference equals the sum of the individual variances (i.e., $v_1 + v_2$). Nearly always in a two-sample t-test, we wish to test the null hypothesis that the true difference in means equals zero. Thus, standardizing the difference in means involves subtracting zero and then dividing by the square root of the variance:

\begin{verbatim}
tstat <- (mean(H)-mean(nH))/sqrt(v1+v2)
tstat
\end{verbatim}

To test the null hypothesis, this t statistic is compared to a t distribution. In a Welch test, we assume that the variances of the two populations are not necessarily equal, and the degrees of freedom of the t distribution are computed using the so-called Satterthwaite approximation:

\begin{verbatim}
(v1 + v2)^2 / (v1^2/91 + v2^2/2585)
\end{verbatim}

The two-sided p-value may now be determined by using the cumulative distribution function of the t distribution, which is given by the \texttt{pt} function:

\begin{verbatim}
2*pt(tstat,97.534)
\end{verbatim}

Incidentally, one of the assumptions of the t-test, namely that each of the two underlying populations is normally distributed, is almost certainly not true in this example. However, because of the central limit theorem, the t-test is robust against violations of this assumption; even if the populations are not roughly normally distributed, the sample means are.

In this particular example, the Welch test is probably not necessary, since the sample variances are so close that an assumption of equal variances is warranted. Thus, we might conduct a slightly more restrictive t-test that assumes equal population variances. Without going into the details here, we merely present the R output:

\begin{verbatim}
t.test(H,nH,var.equal=T)
\end{verbatim}

**Kernel density estimation redux**

Let's take another look at the color (B minus V) variable:

\begin{verbatim}
hist(color,prob=T)
d <- density(color,na.rm=T)
lines(d,col=2,lwd=2,lty=2)
\end{verbatim}

Now we'll use a contributed CRAN package to add confidence limits to the density estimate. The package is called "sm":

\begin{verbatim}
install.packages("sm")  # R will ask you to choose a mirror site
library(sm)
\end{verbatim}
We will use the "sm.density" function:

```r
?sm.density
bandwidth <- bw.nrd0(color[!is.na(color)])
d <- sm.density(color[!is.na(color)],
h=bandwidth, add=TRUE) # Adds to current plot
lines(d$eval.points, d$upper, col=3, lwd=2)
lines(d$eval.points, d$lower, col=3, lwd=2)
```

---

**Chi-squared tests for categorical data**

We begin by creating, albeit artificially, a second categorical variable ("filter", the Hyades indicator, is the first), which we will base on the color (B minus V) variable:

```r
bvcat <- cut(color, breaks=c(-Inf,.5,.75,1,Inf))
```

The cut values for bvcat are based roughly on the quartiles of the B minus V variable. Here is a summary of the dataset based only on these two categorical variables:

```r
table(bvcat,filter)
```

To perform a chi-squared test of the null hypothesis that the true population proportions falling in the four categories are the same for both the Hyades and non-Hyades stars, use the `chisq.test` function:

```r
chisq.test(bvcat,filter)
```

Since we already know these two groups differ with respect to the B.V variable, the result of this test is not too surprising. But it does give a qualitatively different way to compare these two distributions than simply comparing their means.

The p-value produced above is based on the fact that the chi-squared statistic is approximately distributed like a true chi-squared distribution (on 3 degrees of freedom, in this case) if the null hypothesis is true. However, it is possible to obtain exact p-values, if one wishes to calculate the chi-squared statistic for all possible tables of counts with the same row and column sums as the given table. Since this is rarely practical computationally, the exact p-value may be approximated using a Monte Carlo method. Such a method is implemented in the `chisq.test` function:

```r
chisq.test(bvcat,filter,sim=T,B=50000)
```

The two different p-values we just generated a numerically similar but based on entirely different mathematics. The difference may be summed up as follows: The first method produces the exact value of an approximate p-value, whereas the second method produces an approximation to the exact p-value!

The test above is usually called a chi-squared test of homogeneity. If we observe only one sample, but we wish to test whether the categories occur in some pre-specified proportions, a similar test (and the same R function) may be applied. In this case, the test is usually called a chi-squared test of goodness-of-fit.

---

**Maximum likelihood estimation in a simple case**

Let's now switch to a new dataset, one that comes from NASA's Swift satellite. The statistical problem at hand is modeling the X-ray afterglow of gamma-ray bursts. First, read in the dataset:

```r
grb <- read.table("http://astrostatistics.psu.edu/datasets/GRB_afterglow.dat",
header=T, skip=1)
# grb <- read.table("GRB_afterglow.dat", header=T, skip=1)
```

The skip=1 option in the previous statement tells R to ignore the first row in the data file. You will see why this is necessary if you look at the file.

For now, we just consider the second column, which are X-ray fluxes:

```r
flux <- grb[,2]
hist(flux)
```

The histogram suggests that the univariate distribution has roughly the shape of an exponential distribution (we'll speak more about what this means later). Let us replot these data in a particular (and particularly
common) manner besides the histogram that is also suggestive of an exponential distribution.

As a first step, let us calculate something akin to the \((x,y)\) coordinates of the empirical distribution function -
the function that has a jump of size \(1/n\) at every one of the sorted data points.

\[
\begin{align*}
n &\left< \text{length(flux)} \right. \\
xx &\left< \text{sort(flux)} \right. \\
yy &\left< \left(1:n\right)/n \right.
\end{align*}
\]

We could now obtain the empirical cdf by connecting the \((xx,yy)\) points using a stairstep pattern. However, we'll look at these points slightly differently.

The exponential distribution has a distribution function given by \(F(x) = 1 - \exp(-x/\mu)\) for positive \(x\), where \(\mu > 0\) is a scalar parameter equal to the mean of the distribution. This implies among other things that \(\log(1-F(x)) = -x/\mu\) is a linear function of \(x\) in which the slope is the negative reciprocal of the mean. Let us then look for the characteristic linear pattern if we plot \(\log(1-F(x))\) against \(x\) using the empirical distribution function for \(F\) (subtracting a small amount from \(yy\) avoids taking the log of zero):

\[
\begin{align*}
\text{plot}(xx, \; \log(1-yy+1/n), \; xlab\text{="flux"}, \\
\text{ylab\text{"log(1-F(flux))"}})
\end{align*}
\]

The plot certainly looks linear, so let us proceed on the assumption that the flux data are a sample from an exponential distribution with unknown parameter \(\mu\).

The overriding question of this section is this: How shall we estimate \(\mu\)?

As mentioned above, \(\mu\) is equal to the mean of this population. For a quick refresher on some probability theory, let us recall why this is so: The first step in going from the distribution function \(F(x) = 1 - \exp(-x/\mu)\) to the mean, or expectation, is to obtain the density function by differentiating:

\[
f(x) = \frac{\exp(-x/\mu)}{\mu}.
\]

Notice that we typically use \(F(x)\) to denote the distribution function and \(f(x)\) to denote the density function.

Next, we integrate \(x*f(x)\) over the interval 0 to infinity, which gives the mean, \(\mu\).

Since \(\mu\) is the population mean, it is intuitively appealing to simply estimate \(\mu\) using the sample mean. This method, in which we match the population moments to the sample moments and then solve for the parameter estimators, is called the method of moments. Though it is a well-known procedure, we focus instead on a much more widely used method (for good reason) called maximum likelihood estimation.

The first step in maximum likelihood estimation is to write down the likelihood function, which is nothing but the joint density of the dataset viewed as a function of the parameters. Next, we typically take the log, giving what is commonly called the loglikelihood function. Remember that all logs are natural logs unless specified otherwise.

The loglikelihood function in this case is (with apologies for the awkward notation)

\[
l(\mu) = -n \log(\mu) - x_1/\mu - ... - x_n/\mu
\]

A bit of calculus reveals that \(l(\mu)\) is therefore maximized at the sample mean. Thus, the sample mean is not only the method of moments estimator in this case but the maximum likelihood estimate as well.

In practice, however, it is sometimes the case that the linear-looking plot produced earlier is used to estimate \(\mu\). As we remarked, the negative reciprocal of the slope should give \(\mu\), so there is a temptation to fit a straight line using, say, least-squares regression, then use the resulting slope to estimate \(\mu\).

\[
\begin{align*}
\text{mean (flux)} &\left< \text{This is the MLE} \right. \\
m1 &\left< \text{lm(log(1-yy+1/n) \sim xx)} \right. \\
m1 &\left< -l/m1$coef[2] \right. \left< \text{An alternative estimator} \right.
\end{align*}
\]

There is a possible third method that I am told is sometimes used for some kinds of distributions. We start with a histogram, which may be viewed as a rough approximation of the density:

\[
h \left< \text{hist(flux)} \right.
\]

All of the information used to produce the histogram is now stored in the \(h\) object, including the midpoints of the bins and their heights on a density scale (i.e., a scale such that the total area of the histogram equals one).

To see how to use this information, note that the logarithm of the density function is \(\log f(x) = -\log(\mu) -

To see how to use this information, note that the logarithm of the density function is \( \log f(x) = -\log(\mu) - \frac{x}{\mu} \), which is a linear function of \( x \). Thus, plotting the logarithm of the density against \( x \) might be expected to give a line.

```r
counts <- h$counts
dens <- h$density[counts>0]
midpts <- h$mids[counts>0]
plot(midpts, log(dens))
```

When using linear regression to estimate the slope of the linear pattern just produced, I am told that it is standard to weight each point by the number of observations it represents, which is proportional to the reciprocal of the variance of the estimated proportion of the number of points in that bin. We can obtain both the weighted and unweighted versions here. We can then obtain an estimate of \( \mu \) using either the intercept, which is \(-\log(\mu)\), or the slope, which is \(-1/\mu\):

```r
m1 <- lm(log(dens) ~ midpts)
m2 <- lm(log(dens) ~ midpts, weights=counts[counts>0])
expm(-m1$coef[1]) # This is one estimate
-1/m1$coef[2] # This is another
expm(-m2$coef[1]) # Yet another
-1/m2$coef[2] # And another
```

We have thus produced no fewer than six different estimators of \( \mu \) (actually seven, except that the MLE and the method of moments estimator are the same in this case). How should we choose one of them?

There are a couple of ways to answer this question. One is to appeal to statistical theory. The method of maximum likelihood estimation is backed by a vast statistical literature that shows it has certain properties that may be considered optimal. The method of moments is also a well-established method, but arguably with less general theory behind it than the method of maximum likelihood. The regression-based methods, on the other hand, are all essentially ad hoc.

A second way to choose among estimators is to run a simulation study in which we repeatedly simulate datasets (whose parameters are then known to us) and test the estimators to see which seems to perform best. In order to do this, we will need to be able to generate random numbers, which is a topic covered below.

### Standard error of the MLE

Based on asymptotic theory (i.e., the mathematics of statistical estimators as the sample size tends to infinity), we know that for many problems, the sampling distribution of the maximum likelihood estimator (for \( \theta \)) is roughly normally distributed with mean \( \theta \) and variance equal to the inverse of the Fisher information of the dataset. For an i.i.d. sample of size \( n \), the Fisher information is defined to be \( n \) times the mean of the square of the first derivative of the log-density function. Equivalently, it is \(-n\) times the mean of the second derivative of the log-density function. In *many cases, it is easier to use the second derivative than the square of the first derivative.*

In the previous example, the log-density function is \(-\log(\mu) - \frac{x}{\mu}\). The second derivative with respect to the parameter is \( 1/\mu^2 - 2x/\mu^3 \). To find the Fisher information, consider \( x \) to be a random variable; we know its expectation is \( \mu \), so the expectation of the second derivative equals \(-1/\mu^2\). We conclude that the Fisher information equals \( n/\mu^2 \). Intuitively, this means that we get more "information" about the true value of \( \mu \) when this value is close to zero than when it is far from zero. For the exponential distribution with mean \( \mu \), this makes sense.

Earlier we calculated the MLE. Let's give it a name:

```r
mu.hat <- mean (flux) # The MLE found earlier
```

The standard error of \( \mu.hat \), based on the Fisher information, is (the square root of) the inverse of the F.I. evaluated at \( \mu.hat \):

```r
sqrt(mu.hat^2/n) # SE based on (expected) information
mu.hat + 1.96 * c(-1,1) * sqrt(mu.hat^2/n) # approx. 95% CI
```

The F.I. calculated above is sometimes called the *expected* information because it involves an expectation. As an alternative, we can use what is called the *observed* information, which is the negative second
derivative of the log-likelihood function. In this example, the loglikelihood function evaluated at the estimate \( \mu.hat \) is equal to

\[
-n \cdot \log(\mu.hat) - \text{sum(flux)} / \mu.hat
\]

and the negative second derivative of this function, evaluated at \( \mu.hat \), is

\[
-n / \mu.hat^2 + 2\cdot\text{sum(flux)} / \mu.hat^3 \quad \# \text{observed information at MLE}
\]

Notice in this case (though not in every model) that the observed information evaluated at the MLE is equal to the expected information evaluated at the MLE:

\[
n / \mu.hat^2 \quad \# \text{expected information at MLE}
\]

Do you see why?

**Part III: Regression**

This part demonstrates some of the capabilities of R for exploring relationships among two (or more) quantitative variables.

**Linear and polynomial regression**

Earlier, we used exploratory techniques to identify 92 stars from the Hipparcos data set that are associated with the Hyades. We did this based on the values of right ascension, declination, principal motion of right ascension, and principal motion of declination. We then excluded one additional star with a large error of parallax measurement:

```r
mainseqhyades <- filter & (Vmag>4 | B.V<0.2)
logL <- (15 - Vmag - 5 * log10(Plx)) / 2.5
x <- logL[mainseqhyades]
y <- B.V[mainseqhyades]
plot(x, y)
regline <- lm(y~x)
abline(regline, lwd=2, col=2)
summary(regline)
```

Note that the regression line passes exactly through the point (xbar, ybar):

```r
points(mean(x), mean(y), col=3, pch=20, cex=3)
```

Here is a quick example of linear regression relating BminusV to logL, where logL is the luminosity, defined to be \((15 - Vmag - 5 \cdot \log(\text{Plx})) / 2.5\). However, we'll use only the main-sequence Hyades to fit this model:

```r
mainseqhyades <- filter & (Vmag>4 | B.V<0.2)
logL <- (15 - Vmag - 5 * log10(Plx)) / 2.5
x <- logL[mainseqhyades]
y <- B.V[mainseqhyades]
plot(x, y)
regline <- lm(y~x)
abline(regline, lwd=2, col=2)
summary(regline)
```

```r
points(mean(x), mean(y), col=3, pch=20, cex=3)
```

Here is a regression of \( y \) on \( \exp(-x/4) \):

```r
newx <- exp(-x/4)
regline2 <- lm(y-newx)
xseq <- seq(min(x), max(x), len=250)
lines(xseq, regline2$coef %*% rbind(1, exp(-xseq/4)), lwd=2, col=3)
```

Let's reconsider the gamma ray burst dataset.

```r
grb <- read.table("http://astrostatistics.psu.edu/datasets/GRB_afterglow.dat",
header=T, skip=1)
```

```r
grb <- read.table("GRB_afterglow.dat", header=T, skip=1)
```
We will focus on the first two columns, which are times and X-ray fluxes:

```r
plot(grb[,1:2],xlab="time",ylab="flux")
```

This plot is very hard to interpret because of the scales, so let's take the log of each variable:

```r
x <- log(grb[,1])
y <- log(grb[,2])
plot(x,y,xlab="log time",ylab="log flux")
```

The relationship looks roughly linear, so let's try a linear model (lm in R):

```r
modell <- lm(y~x)
abline(modell, col=2, lwd=2)
```

The "response ~ predictor(s)" format seen above is used for model formulas in functions like lm.

The modell object just created is an object of class "lm". The class of an object in R can help to determine how it is treated by functions such as print and summary.

```r
modell # same as print(modell)
summary(modell)
```

Notice the sigma-hat, the R-squared and adjusted R-squared, and the standard errors of the beta-hats in the output from the summary function.

There is a lot of information contained in modell that is not displayed by print or summary:

```
names(modell)
```

For instance, we will use the modell$fitted.values and modell$residuals information later when we look at some residuals plots.

Notice that the coefficient estimates are listed in a regression table, which is standard regression output for any software package. This table gives not only the estimates but their standard errors as well, which enables us to determine whether the estimates are very different from zero. It is possible to give individual confidence intervals for both the intercept parameter and the slope parameter based on this information, but remember that a line really requires both a slope and an intercept. Since our goal is really to estimate a line here, maybe it would be better if we could somehow obtain a confidence "interval" for the lines themselves.

This may in fact be accomplished. By viewing a line as a single two-dimensional point in (intercept, slope) space, we set up a one-to-one correspondence between all (nonvertical) lines and all points in two-dimensional space. It is possible to obtain a two-dimensional confidence ellipse for the (intercept, slope) points, which may then be mapped back into the set of lines to see what it looks like.

Performing all the calculations necessary to do this is somewhat tedious, but fortunately, someone else has already done it and made it available to all R users through CRAN, the Comprehensive R Archive Network. The necessary functions are part of the "car" (companion to applied regression) package. There are several ways to install the car package, but perhaps the most straightforward is by using the install.packages function. Once the car package is installed, its contents can be loaded into the current R session using the library function:

```
install.packages("car")
library(car)
```

If all has gone well, there is now a new set of functions, along with relevant documentation. Here is a 95% confidence ellipse for the (intercept, slope) pairs:

```
confidence.ellipse(modell)
```

Remember that each point on the boundary or in the interior of this ellipse represents a line. If we were to plot all of these lines on the original scatterplot, the region they described would be a 95% confidence band for the true regression line. A graduate student, Derek Young, and I wrote a simple function to draw the borders of this band on a scatterplot. You can see this function at www.stat.psu.edu/~dhunter/R/confidence.band.r; to read it into R, use the source function:

```
source("http://www.stat.psu.edu/~dhunter/R/confidence.band.r")
confidence.band(modell)
```
In this dataset, the confidence band is so narrow that it's hard to see. However, the borders of the band are not straight. You can see the curvature much better when there are fewer points or more variation, as in:

```r
tmpx <- 1:10
tmpy <- 1:10+rnorm(10) # Add random Gaussian noise
certainty.band(lm(tmpy~tmpx))
```

Also note that increasing the sample size increases the precision of the estimated line, thus narrowing the confidence band. Compare the previous plot with the one obtained by replicating tmpx and tmpy 25 times each:

```r
tmpx25 <- rep(tmpx,25)
tmpy25 <- rep(tmpy,25)
certainty.band(lm(tmpy25~tmpx25))
```

A related phenomenon is illustrated if we are given a value of the predictor and asked to predict the response. Two types of intervals are commonly reported in this case: A prediction interval for an individual observation with that predictor value, and a confidence interval for the mean of all individuals with that predictor value. The former is always wider than the latter because it accounts for not only the uncertainty in estimating the true line but also the individual variation around the true line. This phenomenon may be illustrated as follows. Again, we use a toy data set here because the effect is harder to observe on our astronomical dataset. As usual, 95% is the default confidence level.

```r
certainty.band(lm(tmpy~tmpx))
predict(lm(tmpy~tmpx), data.frame(tmpx=7), interval="prediction")
text(c(7,7,7), .Last.value, "P",col=4)
predict(lm(tmpy~tmpx), data.frame(tmpx=7), interval="conf")
text(c(7,7,7), .Last.value, "C",col=5)
```

**Polynomial curve-fitting: Still linear regression!**

Because there appears to be a bit of a bend in the scatterplot, let's try fitting a quadratic curve instead of a linear curve. **Note: Fitting a quadratic curve is still considered linear regression.** This may seem strange, but the reason is that the quadratic regression model assumes that the response y is a linear combination of 1, x, and x^2. Notice the special form of the `lm` command when we implement quadratic regression. The `I` function means "as is" and it resolves any ambiguity in the model formula:

```r
plot(x,y,xlab="log time",ylab="log flux")
model2 <- lm(y~x+I(x^2))
summary(model2)
```

Here is how to find the estimates of beta using the closed-form solution:

```r
X <- cbind(1, x, x^2) # Create nx3 X matrix
solve(t(X) %*% X) %*% t(X) %*% y # Compare to the coefficients above
```

Plotting the quadratic curve is not a simple matter of using the `abline` function. To obtain the plot, we'll first create a sequence of x values, then apply the linear combination implied by the regression model using matrix multiplication:

```r
abline(model1, col=2, lwd=2)
xx <- seq(min(x),max(x),len=200)
yy <- model2$coef %*% rbind(1,xx,xx^2)
lines(xx,yy,lwd=2,col=3)
```

**Model selection using AIC and BIC**

Let's compare the AIC and BIC values for the linear and the quadratic fit. Without getting too deeply into details, the idea behind these criteria is that we know the model with more parameters (the quadratic model) should achieve a higher maximized log-likelihood than the model with fewer parameters (the linear model). However, it may be that the additional increase in the log-likelihood statistic achieved with more parameters is not worth adding the additional parameters. We may test whether it is worth adding the additional parameters by penalizing the log-likelihood by subtracting some positive multiple of the number of parameters. In practice, for technical reasons we take -2 times the log-likelihood, add a positive multiple of
the number of parameters, and look for the smallest resulting value. For AIC, the positive multiple is 2; for BIC, it is the natural log of \( n \), the number of observations. We can obtain both the AIC and BIC results using the **AIC** function. Remember that R is case-sensitive, so "AIC" must be all capital letters.

```r
AIC(model1)
AIC(model2)
```

The value of AIC for model2 is smaller than that for model1, which indicates that model2 provides a better fit that is worth the additional parameters. However, AIC is known to tend to overfit sometimes, meaning that it sometimes favors models with more parameters than they should have. The BIC uses a larger penalty than AIC, and it often seems to do a slightly better job; however, in this case we see there is no difference in the conclusion:

```r
n <- length(x)
AIC(model1, k=log(n))
AIC(model2, k=log(n))
```

---

**Other methods of curve-fitting**

Let's try a nonparametric fit, given by **loess** or **lowess**. First we plot the linear (red) and quadratic (green) fits, then we overlay the lowess fit in blue:

```r
plot(x,y,xlab="log time",ylab="log flux")
abline(model1, lwd=2, col=2)
lines(xx, yy, lwd=3, col=3)
npmodel1 <- lowess(y~x)
lines(npmodel1, col=4, lwd=2)
```

It is hard to see the pattern of the lowess curve in the plot. Let's replot it with no other distractions. Notice that the "type=n" option to **plot** function causes the axes to be plotted but not the points.

```r
plot(x,y,xlab="log time",ylab="log flux", type="n")
lines(npmodel1, col=4, lwd=2)
```

This appears to be a piecewise linear curve. An analysis that assumes a piecewise linear curve will be carried out on these data later in the week.

In the case of non-polynomial (but still parametric) curve-fitting, we can use **nls**. If we replace the response \( y \) by the original (nonlogged) flux values, we might posit a parametric model of the form \( \text{flux} = \exp(a+b*x) \), where \( x=\text{log}(\text{time}) \) as before. Compare a nonlinear approach (in red) with a nonparametric approach (in green) for this case:

```r
flux <- grb[,2]
nlsmo1 <- nls(flux ~ exp(a+b*x), start=list(a=0,b=0))
print(s <- summary(nlsmo1))
npmodel2 <- lowess(flux~x)
plot(x, flux, xlab="log time", ylab="flux")
lines(xx, exp(s$coef[1]+s$coef[2]*xx), col=2, lwd=2)
lines(npmodel2, col=3, lwd=2)
```

Interestingly, the coefficients of the nonlinear least squares fit are different than the coefficients of the original linear model fit on the logged data, even though these coefficients have exactly the same interpretation: If \( \text{flux} = \exp(a + b*\text{x}) \), then shouldn't \( \log(\text{flux}) = a + b*\text{x} \)? The difference arises because these two fitting methods calculate (and subsequently minimize) the residuals on different scales. Try plotting \( \exp(a + b*\text{x}) \) on the scatterplot of \( x \) vs. \( \text{flux} \) for both \( (a,b) \) solutions to see what happens. Next, try plotting \( a + b*\text{x} \) on the scatterplot of \( x \) vs. \( \log(\text{flux}) \) to see what happens.

If outliers appear to have too large an influence over the least-squares solution, we can also try resistant regression, using the **lqs** function in the MASS package. The basic idea behind lqs is that the largest residuals (presumably corresponding to "bad" outliers) are ignored. The results for our log(\( \text{flux} \)) vs. log(\( \text{time} \)) example look terrible but are very revealing. Can you understand why the output from lqs looks so very different from the least-squares output?

```r
library(MASS)
lqsmo1 <- lqs(y~x, method="lts")
plot(x,y,xlab="log time",ylab="log flux")
abline(model1,col=2)
abline(lqsmo1,col=3)
```
Let us now consider least absolute deviation regression, which may be considered a milder form of resistant regression than \textit{\textlqs}. In least absolute deviation regression, even large residuals have an influence on the regression line (unlike in \textit{\textlqs}), but this influence is less than in least squares regression. To implement it, we'll use a function called \textit{\textrq} (regression quantiles) in the "quantreg" package. Like the "car" package, this package is not part of the standard distribution of R, so we'll need to download it. In order to do this, we must tell R where to store the installed library using the \texttt{install.packages} function.

\begin{verbatim}
install.packages("quantreg")
library(quantreg)
\end{verbatim}

Assuming the quantreg package is loaded, we may now compare the least-squares fit (red) with the least absolute deviations fit (green). In this example, the two fits are nearly identical:

\begin{verbatim}
rgmodel1 <- rq(y~x)
plot(x,y,xlab="log time",ylab="log flux")
abline(model1,col=2)
abline(rqmodel1,col=3)
\end{verbatim}

We conclude by mentioning some well-studied and increasingly popular methods of penalized least squares. These include ridge regression (in which the penalty is proportional to the L\textsuperscript{2} norm of the parameter vector) and LASSO (which uses an L\textsuperscript{1} norm instead of an L\textsuperscript{2} norm). There are also other penalized least squares methods, perhaps most notably the SCAD (smoothly clipped absolute deviation) penalty. For an implementation of ridge regression, see the \texttt{lm.ridge} function in the MASS package. To use this function, you must first type \texttt{library(MASS)}.

\section*{Part IV: Techniques based on randomization}

This part deals with randomization and some techniques based on randomization. These include a simulation study whose goal is to evaluate several competing estimators of the same quantity and a method of approximating error distributions called bootstrapping.

\subsection*{Generating random numbers in R}

First, some semantics: "Random numbers" does not refer solely to uniform numbers between 0 and 1, though this is what "random numbers" means in some contexts. We are mostly interested in generating non-uniform random numbers here. If you are interested in the details of uniform (pseudo-)random number generation in R, check out the \texttt{?Random.seed}

\begin{verbatim}
?Random.seed
\end{verbatim}

R handles many common distributions easily. To see a list, type

\begin{verbatim}
help.search("distribution", package="stats")
\end{verbatim}

Let's consider the well-known normal distribution as an example:

\begin{verbatim}
?Normal
\end{verbatim}

The four functions \texttt{'rnorm', 'dnorm', 'pnorm', and 'qnorm'} give random normals, the normal density (sometimes called the differential distribution function), the normal cumulative distribution function (CDF), and the inverse of the normal CDF (also called the quantile function), respectively. Almost all of the other distributions have similar sets of four functions. The 'r' versions are \texttt{rbeta, rbinom, rcauchy, rchisq, rexp, rf, rgamma, rgeom, rhyper, rlogis, rnorm, rmultinom, rbinom, rnorm, rpois, rsignrank, rt, rufid, rweibull}, and \texttt{rwilcox} (there is no rtukey because generally only \texttt{ptukey} and \texttt{qtukey} are needed).

As an example, suppose we wish to simulate a vector of 10 independent, standard (i.e., mean 0 and standard deviation 1) normal random variables. We use the \texttt{rnorm} function for this purpose, and its defaults are mean=0 and standard deviation=1. Thus, we may simply type

\begin{verbatim}
rnorm(10)
\end{verbatim}
Suppose we wish to simulate a large number of normal random variables with mean 10 and standard deviation 3, then check a histogram against two normal density functions, one based on the true parameters and one based on estimates, to see how it looks. We'll use 'col=2, lty=2, lwd=3' to make the curve based on the true parameters red (color=2), dashed (line type=2), and wider than normal (line width=3). Also note that we are requesting 100 bins in the histogram (nclass=100) and putting it on the same vertical scale as the density functions (freq=FALSE).

```r
z <- rnorm(200000, mean=10, sd=3)
hist(z, freq=FALSE, nclass=100)
x <- seq(min(z), max(z), len=200)
lines(x, dnorm(x, mean=10, sd=3), col=2, lty=2, lwd=3)
lines(x, dnorm(x, mean=mean(z), sd=sqrt(var(z))))
```

We can find out what proportion of the deviates lie outside 3 standard deviations from the true mean, a common cutoff used by physical scientists. We can also see the true theoretical proportion:

```r
sum(abs((z-10)/3)>3)/length(z)
2*pnorm(-3)
```

In the first line above, we are using `sum` to count the number of TRUE's in the logical vector (`abs((z-10)/3)>3`). This works because logical values are coerced to 0's and 1's when necessary.

The function `dnorm` has a closed form: With mean=0 and sd=1, `dnorm(x)` equals `exp(-x^2/2)/sqrt(2*pi)`. By contrast, the CDF, given by `pnorm`, has no closed form and must be numerically approximated. By definition, `pnorm(x)` equals the integral of `dnorm(t)` as t ranges from minus infinity to x. To find a p-value (i.e., the probability of observing a statistic more extreme than the one actually observed), we use `pnorm`; to construct a confidence interval (i.e., a range of reasonable values for the true parameter), we use the inverse, `qnorm`.

```r
pnorm(1:3)-pnorm(-1:3)
qnorm(c(.05,.95))
```

The first line above summarizes the well-known 68, 95, 99.7 rule for normal distributions (these are the approximate proportions lying within 1, 2, and 3 standard deviations from the mean). The second line gives the critical values used to construct a 90% confidence interval for a parameter when its estimator is approximately normally distributed.

Let us now briefly consider an example of a discrete distribution, which means a distribution on a finite or countably infinite set (as opposed to a continuous distribution like the normal). The Poisson distribution, which has a single real-valued parameter lambda, puts all of its probability mass on the nonnegative integers. A Poisson distribution, often used to model data consisting of counts, has mean and variance both equal to lambda.

```r
k <- 0:10
dpois(k, lambda=2.5) # or equivalently,
exp(-2.5)*2.5^k/factorial(k)
```

Next, simulate some Poisson variables:

```r
x <- rpois(10000, lambda=2.5)
table(x)
mean(x)
var(x)
```

### The power-law or Pareto distribution

A commonly used distribution in astrophysics is the power-law distribution, more commonly known in the statistics literature as the Pareto distribution. There are no built-in R functions for dealing with this distribution, but because it is an extremely simple distribution it is easy to write such functions.

The density function for the Pareto is $f(x)=ab^a/x^{a+1}$ for $x>b$. Here, a and b are fixed positive parameters, where b is the minimum possible value. As an example, consider the log N = -1.5 * log S relationship, where S is the apparent brightness and N is the number of standard candles randomly located in transparent space. Thus, a Pareto distribution with $(a+1)=1.5$ is a reasonable, if simplistic, model for the brightness of observed standard candles in space. The b parameter merely reflects the choice of units of measurement.
As another example, consider the Salpeter function, the simple but widely known expression of the initial mass function (IMF), in which the mass of a randomly selected newly formed star has a Pareto distribution with parameter \( a=1.35 \).

It turns out that a Pareto random variable is simply \( b^a \exp(X) \), where \( X \) is an exponential random variable with rate \( a \) (i.e., with mean = \( 1/a \)). However, rather than exploiting this simple relationship, we wish to build functions for the Pareto distribution from scratch. Our default values, which may be changed by the user, will be \( a=0.5 \) and \( b=1 \).

\[
d_{\text{pareto}} \left( x, a=0.5, b=1 \right) = a b^a / x^{a+1}
\]

Next, we integrate the density function to obtain the distribution function, which is \( F(x) = 1 - (b/x)^a \) for \( x \geq b \) (and naturally \( F(x) = 0 \) for \( x < b \)).

\[
p_{\text{pareto}} \left( x, a=0.5, b=1 \right) = (x > b) \cdot (1 - (b/x)^a)
\]

Note that \((x > b)\) in the above function is coerced to numeric, either 0 or 1.

Inverting the distribution function gives the quantile function. The following simplistic function is wrong unless \( 0 < u < 1 \), so a better-designed function should do some error-checking.

\[
q_{\text{pareto}} \left( u, a=0.5, b=1 \right) = b / (1-u)^{1/a}
\]

Finally, to simulate random Pareto random variables, we use the fact that whenever the quantile function is applied to a uniform random variable, the result is a random variable with the desired distribution:

\[
r_{\text{pareto}} \left( n, a=0.5, b=1 \right) = q_{\text{pareto}}(\text{runif}(n), a, b)
\]

Creating functions in R, as illustrated above, is a common procedure. Note that each of the arguments of a function may be given a default value, which is used whenever the user calls the function without specifying the value of this parameter. Also note that each of the above functions consists of only a single line; however, longer functions may be created by enclosing them inside curly braces \{ \}.

### A few simple plots

The commands below create plots related to the four functions just created.

```r
par(mfrow=c(2,2))
x <- seq(1,50,len=200)
plot(x,dpareto(x),type="l")
plot(x,ppareto(x),type="l",lty=2)
u <- seq(.005,.9,len=200)
plot(u,qpareto(u),type="l",col=3)
z <- rpareto(200)
dotchart(log10(z), main="200 random logged Pareto deviates ",
cex.main=.7)
par(mfrow=c(1,1))
```

The above commands illustrate some of the many plotting capabilities of R. The `par` function sets many graphical parameters, for instance, `mfrow=c(2,2)`, which divides the plotting window into a matrix of plots, set here to two rows and two columns. In the `plot` commands, `type` is set here to "l" for a line plot; other common options are "p" for points (the default), "b" for connected dots, and "n" for nothing (to create axes only). Other options used: 'lty' sets the line type (1=solid, 2=dashed, etc.), 'col' sets color (1=black, 2=red, 3=green, etc.), 'main' puts a string into the plot title, and 'cex.main' sets the text magnification. Type '?' `par` to see a list of the many plotting parameters and options.

### A simulation study

Let us posit that the random variable \( X \) has a Pareto distribution with parameters \( a=1.35 \) and \( b=1 \). We will simulate multiple datasets with this property and then apply several different estimation methods to each. To simplify matters, we will assume that the value of \( b \) is known, so that the goal is estimation of \( a \).

To evaluate the estimators, we will look at their mean squared error (MSE), which is just what it sounds
like: The average of the squared distances from the estimates to the true parameter value of \(a=1.35\).

To illustrate the estimators we'll evaluate, let's start by simulating a single dataset of size 100:

```r
d <- rpareto(100, a=1.35)
```

Here are the estimators we'll consider:

1. The **maximum likelihood** estimator. Since the density with \(b=1\) is given by \(f(x) = a/x^{a+1}\), the loglikelihood function is

\[
l(a) = n \log(a) - (a+1)(\log x_1 + \log x_2 + \ldots + \log x_n)\]

The maximizer may be found using calculus to equal \(n/(\log x_1 + \ldots + \log x_n)\). For our dataset, this may be found as follows:

\[
1/\text{mean}(\log(d))
\]

We used the sum of logarithms above where we could have used the equivalent mathematical expression given by the log of the product. Sometimes the former method gives more numerically stable answers for very large samples, though in this case "100/\log(prod(d))" gives exactly the same answer.

2. The **method of moments** estimator. By integrating, we find that the mean of the Pareto distribution with \(b=1\) is equal to \(a/(a-1)\). (This fact requires that \(a\) be greater than 1.) Setting \(a/(a-1)\) equal to the sample mean and solving for \(a\) gives \(1/(1-1/\text{mean}(d))\) as the estimator.

\[
1/(1-1/\text{mean}(d))
\]

3. What we'll call the **EDF (empirical distribution function)** estimator. Since \(\log(1-F(x)) = -a \log(x)\) when \(b=1\), by plotting the sorted values of \(\log(d)\) against \(\log(n/n), \log((n-1)/n), \ldots, \log(1/n)\), we should observe roughly a straight line. We may then use least-squares regression to find the slope of the line, which is our estimate of \(-a\):

```r
lse <- log(sort(d))
lseq <- log((100:1)/100)
plot(lse, lseq)
tmp <- lm(lseq ~ lse)
abline(tmp, col=2)
-tmp$coef[2]
```

4. What we'll call the **unweighted histogram estimator**. Since \(\log f(x) = \log(a) - (a+1) \log(x)\) when \(b=1\), if we plot the values of \(\log(d)\) against histogram-based estimates of the log-density function, we should observe roughly a straight line with slope \(-(a+1)\) and intercept \(\log(a)\). Let's use only the slope, since that is the feature that is most often the focus of a plot that is supposed to illustrate a power-law relationship.

```r
hd <- hist(d, nclass=20, plot=F)
counts <- hd$counts
ldens <- log(hd$density[counts>0])
lmidpts <- log(hd$mids[counts>0])
plot(lmidpts, ldens)
tmp <- lm(ldens ~ lmidpts)
abline(tmp, col=2)
-l-numeric(tmp$coef[2])
```

5. What we'll call the **weighted histogram estimator**. Exactly the same as the unweighted histogram estimator, but we'll estimate the slope using weighted least squares instead of ordinary least squares. The weights should be proportional to the bin counts.

```r
plot(lmidpts, ldens)
tmp <- lm(ldens ~ lmidpts, weights=counts[counts>0])
abline(tmp, col=2)
-l-numeric(tmp$coef[2])
```

Now let's write a single function that will take a vector of data as its argument, then return all five of these estimators.

```r
five <- function(d) {
  lsd <- log(sort(d))
  lse <- log((100:1)/100)
  lseq <- log((100:1)/100)
  lseq <- log((100:1)/100)
  plot(lsd, lseq)
  tmp <- lm(lseq ~ lsd)
  abline(tmp, col=2)
  -tmp$coef[2]
}
```

```r
lsd <- log(sort(d))
lseq <- log((100:1)/100)
plot(lsd, lseq)
tmp <- lm(lseq ~ lsd)
abline(tmp, col=2)
-tmp$coef[2]
```
The very last line of the function, "out", is the value that will be returned. (We could also have written "return(out)".) Let's test this function on our dataset:

```r
five(d)
```

There is no good way to compare these estimators based on a single sample like this. We now need to simulate multiple samples. Let's begin by taking n=100.

```r
n.100 <- NULL
for(i in 1:250) {
  dd <- rpareto(100, a=1.35)
  n.100 <- rbind(n.100, five(dd))
}
```

Now we can get estimates of the biases of the estimators (their expectations minus the true parameter) and their variances. Note that we'll use the biased formula for the variance (i.e., the one that uses n instead of n-1 in the denominator) for a technical reason explained below.

```r
bias.100 <- apply(n.100,2,mean) - 1.35
var.100 <- apply(n.100,2,var) * (249/250)
```

It is a mathematical identity that the mean squared error (MSE) equals the square of the bias plus the variance, as we may check numerically for (say) the first column of n.100. However, the identity only works if we use the biased formula for the variance, which is why we used the multiplier (249/250) above.

```r
mean((n.100[,1]-1.35)^2)
bias.100[1]^2 + var.100[1]
```

Thus, we can construct the MSEs and view the results as follows:

```r
mse.100 <- bias.100^2 + var.100
rbind(bias.100, var.100, mse.100)
```

Finally, let's repeat the whole experiment using samples of size 200.

```r
n.200 <- NULL
for(i in 1:250) {
  dd <- rpareto(200, a=1.35)
  n.200 <- rbind(n.200, five(dd))
}
```

```r
bias.200 <- apply(n.200,2,mean) - 1.35
var.200 <- apply(n.200,2,var) * (249/250)
mse.200 <- bias.200^2 + var.200
rbind(bias.200, var.200, mse.200)
```

**Nonparametric bootstrapping of regression standard errors**

This part deals with another randomization-based technique called bootstrapping. The term "bootstrapping" means something like improving oneself by one's own unaided efforts, and the statistical technique bearing the name has a bit of this flavor.

Earlier, we used exploratory techniques to identify 92 stars from the Hipparcos data set that are associated
with the Hyades. We then looked at a regression relationship between color and log-luminosity for the 88 main sequence stars:

```r
hip <- read.table("http://astrostatistics.psu.edu/datasets/HIP_star.dat", header=T, fill=T)
hip <- read.table("HIP_star.dat", header=T, fill=T)
attach(hip)
filter1 <- (RA>50 & RA<100 & DE>0 & DE<25)
filter2 <- (pmRA>90 & pmRA<130 & pmDE>-60 & pmDE< -10)
filter <- filter1 & filter2 & (e_Plx<5)
sum(filter)
mainseqhyades <- filter & (Vmag>4 | B.V<0.2)
logL <- (15 - Vmag - 5 * log10(Plx)) / 2.5
x <- logL[mainseqhyades]
y <- B.V[mainseqhyades]
plot(x,y,pch=20)
model1 <- lm(y ~ x)
abline(model1,lwd=2,col=2)
The red line on the plot is the usual least-squares line, for which estimation is easy and asymptotic theory gives easy-to-calculate standard errors for the coefficients:

```summary(model1)$coef
```

However, suppose we wish to use a resistant regression method such as `lqs`.

```r
library(MASS)
model2 <- lqs(y ~ x)
abline(model2, lwd=2, col=3)
model2
```

In this case, it is not so easy to obtain standard errors for the coefficients. Thus, we will turn to bootstrapping. In a standard, or nonparametric, bootstrap, we repeatedly draw samples of size 92 from the empirical distribution of the data, which in this case consist of the (DE, pmDE) pairs. We use `lqs` to fit a line to each sample, then compute the sample covariance of the resulting coefficient vectors. The procedure works like this:

```r
model2B <- matrix(0, 200, 2)
for (i in 1:200) {
  s <- sample(92, replace=T)
  model2B[i,] <- lqs(y[s]~x[s])$coef
}
```

We may now find the sample covariance matrix for model2B. The (marginal) standard errors of the coefficients are obtained as the square roots of the diagonal entries of this matrix:

```r
cov(model2B)
se <- sqrt(diag(cov(model2B)))
se
```

The logic of the bootstrap procedure is that we are estimating an approximation of the true standard errors. The approximation involves replacing the true distribution of the data (unknown) with the empirical distribution of the data. This approximation may be estimated with arbitrary accuracy by a Monte Carlo approach, since the empirical distribution of the data is known and in principle we may sample from it as many times as we wish. In other words, as the bootstrap sample size increases, we get a better estimate of the true value of the approximation. On the other hand, the quality of this approximation depends on the original sample size (92, in our example) and there is nothing we can do to change it.

An alternative way to generate a bootstrap sample in this example is by generating a new value of each response variable (y) by adding the predicted value from the original lqs model to a randomly selected residual from the original set of residuals. Thus, we resample not the entire bivariate structure but merely the residuals. As an exercise, you might try implementing this approach in R. Note that this approach is not a good idea if you have reason to believe that the distribution of residuals is not the same for all points. For instance, if there is heteroscedasticity or if the residuals contain structure not explained by the model, this residual resampling approach is not warranted.

**Using the boot package in R**

file:///Users/e5f/Desktop/R%20for%20Astronomers%202011/Rtutorial2011.html
There is a `boot` package in R, part of the base R distribution, that contains many functions relevant to bootstrapping. As a quick example, we will show here how to obtain the same kind of bootstrap example obtained above.

```r
library(boot)
mystat <- function(a,b)
  lqs(a[b,2]-a[b,1])$coef
model2B.2 <- boot(cbind(x,y),
  mystat, 200)
names(model2B.2)
```

As explained in the help file, the `boot` function requires as input a function that accepts as arguments the whole dataset and an index that references an observation from that dataset. This is why we defined the `mystat` function above. To see the output that is similar to that obtained earlier for the m2B object, look in `m2B2$t`:

```r
cov(model2B.2$t)
sqrt(diag(cov(model2B.2$t)))
```

Compare with the output provided by `print.boot` and the plot produced by `plot.boot`:

```r
model2B.2
plot(model2B.2)
```

Another related function, for producing bootstrap confidence intervals, is `boot.ci`.

### Parametric bootstrapping of regression standard errors

We now return to the regression problem studied earlier.

Sometimes, resampling is done from a theoretical distribution rather than from the original sample. For instance, if simple linear regression is applied to the regression of pmDE on DE, we obtain a parametric estimate of the distribution of the residuals, namely, normal with mean zero and standard deviation estimated from the regression:

```r
summary(model1)
```

Remember that model1 was defined above as `lm(y~x)`. We observe a residual standard error of 0.0649.

A parametric bootstrap scheme proceeds by simulating a new set of pmDE (or y) values using the model

```r
y <- 0.747 - 0.407*x + 0.0649*rnorm(92)
```

Then, we refit a linear model using y as the new response, obtaining slightly different values of the regression coefficients. If this is repeated, we obtain an approximation of the joint distribution of the regression coefficients for this model.

Naturally, the same approach could be tried with other regression methods such as those discussed above, but careful thought should be given to the parametric model used to generate the new residuals. In the normal case discussed here, the parametric bootstrap is simple, but it is really not necessary because standard linear regression already gives a very good approximation to the joint distribution of the regression coefficients when errors are heteroscedastic and normal. One possible use of this method is in a model that assumes the absolute residuals are exponentially distributed, in which case least absolute deviation regression as discussed earlier can be justified. The reader is encouraged to implement a parametric bootstrap using the `rq` function found in the "quantreg" package.

### Part V: EM algorithms

The class of algorithms called EM algorithms is enormously important in statistics. There are many, many different specific algorithms that can be called EM algorithms, but they have this in common: They seek to iteratively maximize a likelihood function in a situation in which the data may be thought of as incompletely observed.

The name "EM algorithm" has its genesis in a seminal 1977 paper by Dempster, Laird, and Rubin in the
Journal of the Royal Statistical Society, Series B. Many distinct algorithms published prior to 1977 were examples of EM, including the Lucy-Richardson algorithm for image deconvolution that is apparently quite well known in astronomy. The major contribution of Dempster et al was to unify these algorithms and prove certain facts about them. Interesting historical note: They even "proved" an untrue fact that was refuted six years (!) later (even thirty years ago, publications in statistics churned through the pipeline at a snail's pace).

An EM algorithm for right-censored measurements

We'll derive a simple EM algorithm on a toy example: We'll pretend that some of the gamma ray burst flux measurements were right-censored, as follows:

```r
grb <- read.table("http://astrostatistics.psu.edu/datasets/GRB_afterglow.dat", header=T, skip=1)  # grb <- read.table("GRB_afterglow.dat", header=T, skip=1)
flu <- grb[,2]
cflu <- flux
cflu[flu>=60] <- 60
n <- length(cflu)
yy <- (1:n)/n
plot(sort(cflu),log(1-yy+1/n))
```

The situation may be viewed like this: The complete dataset is a set of n observations from an exponential distribution with unknown mean $\mu$, say, $X_1, \ldots, X_n$. What we observe, however, is $Z_1, \ldots, Z_n$, where $Z_i$ is defined as $\min(X_i, 60)$. The loglikelihood for the observed data is as follows:

```r
m <- sum(flu>=60)
s <- sum(cflu)
loglik <- function(mu)
  -(n-m)*log(mu)-s/mu
```

As it turns out, this loglikelihood function can be maximized explicitly:

```r
mle <- s/(n-m)
```

However, we will construct an EM algorithm anyway for two reasons: First, it is instructive to see how the EM operates. Second, not all censoring problems admit a closed-form solution like this one does!

We start by writing down the complete-data loglikelihood for the sample. This is straightforward because the complete data are simply a random sample from an exponential distribution with mean $\mu$. Next, we pick a starting value of $\mu$, say $\mu_0$. Then comes the tricky part: We take the conditional expectation of the complete data loglikelihood, conditional on the observed data and assuming that $\mu_0$ is the correct parameter. (This will have to happen on the chalkboard!) The result is a function of both $\mu$ and $\mu_0$, and construction of this function is called the E (expectation) step. Next, we maximize this function over $\mu$. The result of the maximization becomes our next iterate, $\mu_1$, and the process repeats.

Let's start with $\mu_0=20$. Carrying out the calculations described above yields the following iterative scheme:

```r
mu <- 20
loglik(mu)
mu <- s/n + m*mu/n; loglik(mu)
mu <- s/n + m*mu/n; loglik(mu)
# repeat the last line a few times
```

Notice that the value of the (observed data) loglikelihood increases at each iteration. This is the fundamental property of any EM algorithm! In fact, it is very helpful when debugging computer code, since there must be a bug somewhere whenever the loglikelihood is ever observed to decrease. Notice also that the value of $\mu$ has converged to the true maximum likelihood estimator after a few iterations.

An EM algorithm for a mixture: A simple case

Let's try a two-component mixture of normals model on a quasar absorption line dataset. We only want to read 104 lines of the text file, starting with line 2, and we are only interested in the second of the two columns:
qso <- scan("http://www.astrostatistics.psu.edu/datasets/QSO_absorb.txt", skip=1, nlines=104)[2*(1:104)]
# qso <- scan("QSO_absorb.txt", skip=1, nlines=104)[2*(1:104)]
hist(qso) # Get a look at the dataset.

Here is a function to implement an EM algorithm for a two-component mixture of normals, where the variances of the two components are assumed to be the same. The function may be modified to allow for different variances by using the commented-out lines rather than the ones preceding them:

twocomponentnormalmixtureEM <- function(mu1, mu2, sigsqrds, lambda, x) {
    mx <- mean(x)
    nover2 <- length(x)/2
    dl <- 1
    l <- -Inf
    iterations <- 0
    a1 <- (x-mu1)^2; b1 <- lambda*exp(-a1/2/sigsqrds)
    a2 <- (x-mu2)^2; b2 <- -(1-lambda)*exp(-a2/2/sigsqrds)
    while (dl > 1e-4) {
        iterations <- iterations + 1
        postprobs <- b1/(b1+b2)
        lambda <- mean(postprobs)
        mu1 <- mean(postprobs*x)/lambda
        mu2 <- (mx - lambda*mu1)/(1-lambda)
        sigsqrds <- mean(postprobs*a1 + (1-postprobs)*a2)
        oldl <- l
        a1 <- (x-mu1)^2; b1 <- lambda*exp(-a1/2/sigsqrds)
        a2 <- (x-mu2)^2; b2 <- -(1-lambda)*exp(-a2/2/sigsqrds)
        l <- sum(log(b1+b2)) - nover2*log(sigsqrds)
        dl <- l - oldl
    }
    list(mu = c(mu1, mu2), variance = sigsqrds, lambda = lambda, loglik = l, iterations = iterations)
}

Now let's try it:

hist(qso, nclass=20)
twocomponentnormalmixtureEM (.6, 1, .1, .1, qso)

This seems to converge to a sensible solution. However, try some other starting values and see what happens. If you find a different solution (i.e., a different local maximum), how does its loglikelihood value compare to that of the first solution?
Multivariate Analysis and Data Mining for Astronomy

Eric Feigelson  SCMA tutorials 2011

Multivariate datasets in astronomy

*Whenever you have a table of rows and columns !!!*

Common, simple cases:
- Rows are stars, galaxies, planets, AGN, sources, locations, structures ...  
- Columns are properties: fluxes, line strengths, spectral indices, polarization fractions, ellipticities, eccentricities, sizes, ...
- Quantities can be real, integer, categorical or binary

Other cases:
- Images (fluxes vs. RA & Dec)
- Spectral datacubes [HET HETDEX] (fluxes vs. RA, Dec & wavelengths)
- Photon counting detectors [Chandra ACIS, Swift XRT] (photons vs. RA, Dec, wavelength, arrival time)
- Multiepoch image surveys [PTF, DeepSky, LSST] (fluxes vs. RA, Dec, time)
- Theoretical models [GADGET] (models or fitted data in parameter space; N-body simulations; hydrodynamical simulations)
Goals of multivariate analysis

1. quantify the location and distribution of the dataset in p-space (multivariate mean locations, variances and covariances, probability distributions and quantile functions)
2. achieve structural simplification by reducing the number of variables without significant loss of scientific information by combining or omitting variables
3. investigate the dependence between variables and seek relations between these correlations
4. pursue statistical inference of many types including testing hypotheses, applying and testing parametric regression models, and calculating confidence regions for estimated parameters
5. classify or cluster "similar" objects into homogeneous subsamples (next week)
6. predict the location of future objects (usually not important in astronomical research)

Structure in multivariate datasets

Multivariate analysis
Finding multivariate relationships using probabilistic models (1930s-60s). PCA, FA, CCA, MANOVA, multiple regression, etc. Early techniques based on least squares & maximum likelihood assuming multivariate normal distributions

Multivariate discrimination
Finding multivariate properties that discriminate previously known groups of objects. Linear discriminant analysis (1930s) based on multivariate normal distributions.

Multivariate clustering
Finding distinct groups of objects in multivariate databases without prior knowledge of their number or location. Techniques include hierarchical clustering, k-means, multidimensional scaling (1960-80s).

Multivariate classification
Finding distinct groups of objects in multivariate databases based on training sets.
Means and variances

The sample covariance matrix $S$ is crucial in multivariate analysis: diagonal elements measure the spread in each variable, and off-diagonal elements measure linear association between variables. For large-$n$ and homogeneous samples, the sample covariance matrix converges to the population covariance matrix $\Sigma$ (Central Limit Theorem).

\[ s_{jk} = \frac{1}{n} \sum_{i=1}^{n} (x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k). \]

Distances in multivariate analysis

What is the ‘distance’ between two data points in $p$-space?

In 1-dimension, the distance is simply $d_1 = x_1 - x_j$. In $p$-dimensions, the Euclidean distance applies if the units of the dimensions are the same, $d_j = (\sum (x_{ij}^2))^{1/2}$.

But what if the units of the multivariate dataset are RA, Dec, km/s, pc, metallicity, solar masses, Type I vs. II, and redshift? The meaning of a $p$-dimensional ‘distance’ is completely unclear … the astronomer must make an arbitrary definition of distance.

Unit-free variables

The most common approach is to transform variables to a unit-free form, conduct a multivariate analysis, and transform back for interpretation. The most common transformation is standardization: subtract the mean and divide by the standard deviation, treating each variable separately. The covariance matrix is standardized using elements of the bivariate linear correlation coefficient. The Mahalonobis distance between two standardized data points. Astronomers often reduce range, and incidentally remove units, with a logarithmic transformation. The choice of variable transformation is arbitrary.
What is Data Mining?

Data Mining is a loosely-defined suite of approaches to understanding very large multivariate datasets: “pattern recognition”, “machine learning”, “knowledge discovery in databases” (KDD). While often based on multivariate probabilistic approaches, they are algorithms typically designed by computer scientists (rather than statisticians) working in Artificial Intelligence for efficient detection of structure and groupings in datasets. Techniques include:

- Artificial neural networks (ANN, 1950s–)
- Classification and Regression Trees (CART, 1960s–) with bagging & boosting (1990s–)
- Support Vector Machines (SVM, 1990s–)
- Nearest neighbor methods (k-NN, 1990s–)
- Genetic programming (1990s–)
- Bayesian classifiers (1990s–)

Applications include: handwriting, speech & biometric recognition; robotics; business decisions; data compression; image processing; scientific research

Data Mining in R

R is the largest public domain statistical software system. High level language similar to IDL. Base R (c.2005) has a few hundred statistical functions plus publication-quality graphics. The Comprehensive R Analysis Network (CRAN) is an exponentially growing collection of user-supplied R packages with thousands of functionalities:

>60 packages for machine learning

http://cran.r-project.org/web/views/MachineLearning.html

>120 packages for multivariate clustering

http://cran.r-project.org/web/views/Cluster.html

>130 packages for multivariate statistics

http://cran.r-project.org/web/views/Multivariate.html
Modern Statistical Methods
for Astronomy
with R applications

DRAFT January 30 2011 DRAFT

Eric D. Feigelson & G. Jogesh Babu

Not for distribution
CHAPTER 8. MULTIVARIATE ANALYSIS

8.6 Recommended reading

A collection of review articles on modern data visualization techniques. Topics include static, interactive and linked graphs; histograms; scatterplots; graph-theoretic approaches; data glyphs; dendrograms; mosaic plots; trellis plots; parallel coordinate plots; projection pursuit; and multivariate tours.

A comprehensive, advanced text that integrates classical with modern multivariate methods. Coverage includes: nonparametric density estimation; multiple and multivariate regression, principal component analysis; projection pursuit; multidimensional scaling; independent component analysis; principal curves; nonlinear manifold learning; and correspondence analysis.

A widely used and respected undergraduate textbook. Topics include the matrix algebra, multivariate normal distribution, inferences and comparisons of multivariate means, linear regression models, principal components analysis, factor analysis, structural equation models, canonical correlation analysis, linear discrimination and classification, hierarchical clustering, and multidimensional scaling.

A well-written and useful book for chemists with extensive applications in R and CRAN. Coverage includes multivariate data, principal component analysis, ordinary and partial least squares regression, robust regression, variable selection, classification (linear discriminant analysis, trees, neural networks, Support Vector Machines), clustering (hierarchical clustering, partitioning, model-based clustering).

8.7 R applications

To illustrate a range of multivariate analysis methods, we examine a small dataset presenting global properties of Galactic globular clusters (Webbink 1985) described in Appendix C. This dataset has 147 data points with 20 variables. In Chpt. we examined their univariate magnitude distribution, here we examine a variety of properties concerning their location in the Galaxy, photometry, metallicity, and dynamical properties such as core radius, tidal radius, crossing timescales, escape
8.7. **R Applications**

velocities, and eccentricity.

### 8.7.1 Univariate tests of normality

We first read the ASCII table using `read.table` with options for a header line with variable names and filling in incomplete rows with `NA` values. A useful early step in multivariate studies is to examine the distribution of each variable, both its range and its similarity to a normal distribution. Strong differences in either property will affect results of many multivariate analyses. R’s `summary` function gives the minimum, 25% quartile, median, mean, 75% quartile, and maximum for each variable. We construct a display of univariate histograms, `manyhist`, using R’s `function` function (Figure ??). The number of panels is evaluated using `n2mfrow` from the dimensionality of the dataset. R! `n2mfrow`

```
# Overview of Webbink globular cluster properties

GC = read.table("http://astrostatistics.psu.edu/MSMA/datasets/Webbink_GC_tab.txt",header=T,fill=T)
summary(GC)
manyhist <- function(x) {
  par(mfrow=n2mfrow(dim(x)[2]))
  for (i in 1:dim(x)[2]) {
    name=names(x)[i]
    hist(x[,i], main='', breaks='FD', ylab='', xlab=name)
  }
  par(mfrow=c(1,1))
  manyhist(GC[,-1])
}

Figure ??a shows boxplots for three of the variables. The boxplot displays show that Galactic longitude has a range of $0 - 360$ (in units of degrees) while core radius ranges over $0 - 25$ (in parsecs) and central surface brightness ranges over $5 - 19$ (magnitudes per square arcminute). The box-whisker shapes are also very different, indicating heavy tails in Galactic longitudes, strong skewed tails in core radius, and a near-normal distribution in central surface brightness.

More detail information is provided by normal quantile-quantile plots produced by function `qqnorm` (Figure ??b). A variable with approximately normal distribution will appear as a straight line in these plots; the `qqline` function draws a line through the first and third quartiles. We see that the globular cluster Galactic longitudes are strongly concentrated towards the Galactic Center which appears both near $0^\circ$ and $360^\circ$, causing most points to lie near the tails of the distribution. Most of the core radius are concentrated around small values around $0 - 2$ pc, but a long tail to larger values. The central surface brightnesses is closest to normal, but shows a heavy tail at low values.

```
# Boxplots and normal quantile-quantile plots for three variables

par(mfrow=c(1,3))
```
Figure 8.1: Histograms of all variables in Webbink’s globular cluster dataset.
8.7. R APPLICATIONS

boxplot(GC[,2], main='Galactic longitude', pars=list(cex=1.5, cex.main=2, cex.lab=2))
boxplot(GC[,8], main='Core radius', pars=list(cex=1.5, cex.main=2, cex.lab=2))
boxplot(GC[,20], main='Surf brightness', pars=list(cex=1.5, cex.main=2, cex.lab=2))

qqnorm(GC[,2], main='Galactic longitude', pch=20, cex=1.5, cex.lab=1.5, cex.main=2)
qqnorm(GC[,8], main='Core radius', pch=20, cex=1.5, cex.lab=1.5, cex.main=2)
qqnorm(GC[,20], main='Surf brightness', pch=20, cex=1.5, cex.lab=1.5, cex.main=2)
qqline(GC[,20])
par(mfrow=c(1,1))

8.7.2 Preparing the dataset

We now prepare the dataset for multivariate analysis in several ways. Not all steps are needed in later operations, but we proceed in a uniform fashion for simplicity. First, to compute variances, we remove objects with empty ('NA') cells using R function `na.omit`; this reduces the sample from 147 to 113 globular clusters.

Second, the inconsistent ranges of the different variables hinders many analyses. For example, the first component of a PCA applied to the original dataset is dominated by Galactic longitude because it has a wider range ($0^\circ - 360^\circ$) than other variables. We therefore standardize the variables to unit variance using function `stdize` in CRAN package `pls` (partial least squares) that was written for chemometrics (Filzmoser & Varmuza 2010). Since the `stdize` function produces an R list rather than an R data frame, as revealed using the structure function `str`, we convert format using function `data.frame`.

Third, we find it scientifically advantageous to divide the variables into two groups: dynamical variables that describe the internal properties of each globular cluster, and locational properties describing where it lies in the Galaxy. Note the syntax `Dataset[,,-c(1:4)]` that removes columns from a multivariate dataset.

Fourth, we examine the dataset for outliers which can strongly affect some analyses. From the univariate distributions of the standardized variables, we find two discrepant points: #12 has a very large tidal radius due to its location in the outer Galactic halo, and #63 has an unusually high absorption due to its location near the Galactic Center. These have $6\sigma$ deviations from the mean in one variable. Although these outliers can be scientifically important, here we artificially remove these points using R function `which.max` that identifies the row of the maximum value for a chosen variable. To treat outliers in an automated and algorithmic fashion, robust methods should be used. Robust estimation of covariance matrices, ANOVA, multivariate regression, and PCA is provided by CRAN package `robust` using the MCD and other methods. Chemometrics package `pcaPP` provides a robust PCA calculation.

# Prepare the data: remove missing data, standardize, create variable groups, remove outliers

dim(GC) ; GC1=na.omit(GC) ; dim(GC1)
Figure 8.2: Univariate distributions of three variables of Galactic globular clusters: (a) boxplots; and (b) normal quantile-quantile plots.
install.packages('pls') ; library(pls)
GC2 <- stdize(GC1)
str(GC2)
GC2 <- data.frame(GC2[])

GCloc <- GC2[,+c(1:4)]
GCdyn <- GC2[-c(1:4)]

GCDyn1 <- GCDyn[-c(which.max(GCdyn[,4]), which.max(GCdyn[,11])),]
GCloc1 <- GCloc[-c(which.max(GCdyn[,4]), which.max(GCdyn[,11])),]

8.7.3 Bivariate relationships

A common and useful early procedure in multivariate analysis, providing the dimensionality is not too large, is to examine pairwise scatter plots of the dataset. This is provided by R’s pairs function (Figure 8.3), and the significance of correlations can be assessed using Kendall’s $\tau$ with the cor function. The significance of a single bivariate relationship is calculated using cor.test. The R functions var and cov both give the covariance matrix of a multivariate dataset.

Concentrating on the dynamical variables, we first visually examine bivariate relationships from a pairs plot. We show both a standard pairs plot, and one with a variety of useful annotations: leastsquares regression line, local regression curve, one-dimensional ‘rug’ plot, and a two-dimensional contour map of a normal kernel density estimator.

A variety of bivariate relationships, both correlations and anticorrelations, is seen in the globular cluster dynamical variables. Distributions are often not do not have the MVN shapes and relationships are sometimes nonlinear. Linear procedures based on MVN distributions will thus not fully model the structure in the dataset, but traditional methods often provide useful insight. Much of the bivariate structure is statistically significant: over half of the bivariate relationships are significant at a level $|\tau| > 0.20$, equivalent to $P < 0.003$ for $n = 111$ or $> 3 \sigma$ deviation assuming a normal distribution. Some correlations show little scatter, while others are broad. Some show threshold effects.

# Bivariate relationships

cor(GCdyn, method='kendall') ; var(GCdyn)
pairs(GCdyn1,pch=20,cex=0.1)
cor(GCdyn1,method='kendall')
var(GCdyn1)

# Annotated pairs plot
Figure 8.3: Bivariate distributions of Galactic globular clusters dynamical variables. Top: Standard pairwise scatter plot. Bottom: Annotated pairs plot.
library(MASS)
pairs(GCdyn1[,4:7], panel=function(x,y) {abline(lsfit(x,y)$coef, lwd=2)
    lines(lowess(x,y), lty=2, lwd=2)
    points(x, y, pch=20)
    rug(jitter(x, factor=2),side=1)
    rug(jitter(y, factor=2),side=2)
    contour(kde2d(x,y)$x, kde2d(x,y)$y, kde2d(x,y)$z,drawlabels=F, add=T) })

8.7.4 Principal components analysis

The PCA solution is obtained using R’s `princomp` function, and various results are presented using `summary`, `loadings`, `plot` and `biplot`. R’s plot function for PCA produces a ‘scree plot’, shown in Figure 8.4a, which presents the variance explained by each component. The biplot shows the location of each data point in a scatter plot of the first and second principal components, and displays arrows relating to the strength of each variable’s loading in each component. There are many variants of biplots as detailed by Gower and Hand (1996).

Here we see some relationships expected from our astrophysical knowledge of globular clusters. $S_0$ and $V_{esc}$ both measure stellar velocity dispersion, and central star density is inversely correlated with core radius. But the associations of $[Fe/H]$ metallicity index with absorption $E(B−V)$ and of luminosity $M_V$ with ellipticity are less obvious and may be valuable scientific results.

Finally, we add the principal component values for each data point into a multivariate dataset for later analysis using R’s `data.frame` function. For example, if the dataset is heterogeneous, clustering and classification procedures (Chpt. 9) are often valuable performed on the components rather than the original variables.

# PCA for dynamical variables.

PCdyn <- princomp(GCdyn1)
plot(PCdyn, main='')
summary(PCdyn)
loadings(PCdyn)
biplot(PCdyn)

# Add principal component values into data frame

PCdat <- data.frame(names=row.names(GCdyn1), GCdyn1, PCdyn$scores[,1:4])

Table 8.7.4 shows the loadings of the first three components which together account for 73% of the sample variance in the standardized variables. The first component is dominated by the expected close relationship between core radius, con-
Figure 8.4: PCA results for 15 dynamical variables of Galactic globular clusters: (a) scree plot showing the variance reduction by the first 10 components; and (b) biplot of the first two components showing individual cluster locations and variable relationships.
centrination, central star density, and escape velocity. Less obvious is the inclusion of total luminosity ($M_V$) and ellipticity in this component. The second component is dominated by measures relating to cluster metallicity: the metal line strengths ([Fe/H]), color indices $E(B - V)$ and $B - V$, and subtle measure of the height of the horizontal branch in the Hertzsprung-Russell diagram. The third component links combination of dynamical, luminosity and metallicity variables in ways that are difficult to interpret. This component may be dominated by noise rather than astrophysically meaningful relationships.

### Principal components analysis of Galactic globular clusters

<table>
<thead>
<tr>
<th>Property</th>
<th>Comp.1</th>
<th>Comp.2</th>
<th>Comp.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variance</td>
<td>44%</td>
<td>19%</td>
<td>11%</td>
</tr>
<tr>
<td>[Fe/H]</td>
<td>...</td>
<td>-0.27</td>
<td>-0.35</td>
</tr>
<tr>
<td>$M_V$ (mag)</td>
<td>0.30</td>
<td>-0.32</td>
<td>0.33</td>
</tr>
<tr>
<td>$r_{core}$ (pc)</td>
<td>0.31</td>
<td>0.24</td>
<td>-0.18</td>
</tr>
<tr>
<td>$r_{tidal}$ (pc)</td>
<td>...</td>
<td>0.31</td>
<td>...</td>
</tr>
<tr>
<td>Concentration</td>
<td>-0.32</td>
<td>...</td>
<td>0.20</td>
</tr>
<tr>
<td>$\log t_{rad}$ (s)</td>
<td>0.25</td>
<td>0.37</td>
<td>-0.35</td>
</tr>
<tr>
<td>$\log \rho_{cent}$ (stars/pc$^3$)</td>
<td>-0.38</td>
<td>-0.20</td>
<td>...</td>
</tr>
<tr>
<td>$S_0$ (km/s)</td>
<td>-0.35</td>
<td>0.11</td>
<td>-0.38</td>
</tr>
<tr>
<td>$V_{esc}$ (km/s)</td>
<td>-0.36</td>
<td>0.11</td>
<td>-0.33</td>
</tr>
<tr>
<td>Vertical Horizontal Branch (mag)</td>
<td>0.13</td>
<td>-0.16</td>
<td>-0.31</td>
</tr>
<tr>
<td>E(B-V) (mag)</td>
<td>...</td>
<td>-0.39</td>
<td>-0.22</td>
</tr>
<tr>
<td>B-V (mag)</td>
<td>...</td>
<td>-0.42</td>
<td>-0.30</td>
</tr>
<tr>
<td>Ellipticity</td>
<td>0.28</td>
<td>-0.30</td>
<td>...</td>
</tr>
<tr>
<td>$V_t$ (mag)</td>
<td>-0.12</td>
<td>...</td>
<td>0.11</td>
</tr>
<tr>
<td>Central surface brightness</td>
<td>0.39</td>
<td>-0.10</td>
<td>-0.10</td>
</tr>
</tbody>
</table>

Some common characteristics of PCA are seen here. First, PCA has effectively reduced the dimensionality of the globular cluster dynamical dataset from $p = 15$ to $p \sim 2 - 3$. This is achieved both by combining variables that measure almost the same physical property, and by combining independent variables that are linearly correlated. Second, the component loadings are sometimes readily interpretable scientifically, but in other cases are difficult to understand. Third, unless a distinct ‘elbow’ appears in the scree plot, it is often unclear how many principal components should be used for further analysis and scientific interpretation. Components contributing only small fractions of the total variance in the sample are likely fitting noise as well as real structure.

We applied canonical correlations, R’s `cancor` function, to see whether linear
correlations in the Galactic location variables might explain correlations in the dynamical variables. However, the results did not have any obvious astronomical interpretation.

8.7.5 Multiple regression and MARS

While PCA is designed to find the intrinsic relationships between globular cluster properties, regression tells the dependencies between a chosen ‘response’ variable and other dynamical properties. Here we investigate the central surface brightness (CSB) as our variable of interest. R’s linear modeling function \texttt{lm} gives unweighted least-squares regression coefficients using QR matrix decomposition. The output file, here called \texttt{CSB.fit1} contains a variety of useful results and diagnostics. The resulting linear regression formula is

\[
CSB = -0.94 \log \rho_{cen} + 0.33V_{esc} - 0.33S_0 - 0.18VHB + 0.33E(B - V) + 0.31(B - V) + 0.24Ellipt + 0.09M_V - 0.09[Fe/H] + 0.07Conc + 0.07 \log t_{rad}
\]

with negligible dependence on \(r_{core}\) and \(V_t\). Recall that the units of CSB are magnitudes arcmin\(^{-2}\) so that negative coefficients represent positive correlations between the covariate and the central surface brightness. If we restrict the regression to the six variables with coefficients more important than 0.1, the result is

\[
CSB = -0.96 \log \rho_{cen} + 0.62V_{esc} - 0.61S_0 + 0.59E(B - V) - 0.23VHB + 0.30Ellipt
\]

# Multiple regression to predict globular cluster central surface brightnesses

```
attach(GCdyn)
CSB_fit1 <- lm(Cent.surf.bright~.-Cent.surf.bright, data=GCdyn) ; CSB_fit1
CSB_fitt <- lm(Cent.surf.bright~., data=GCdyn[,c(7:11,13)]) ; CSB_fit2
str(CSB_fit2)
par(mfrow=c(2,1))
plot(CSB_fit2$fitted.values, Cent.surf.bright, pch=20)
qqnorm(CSB_fit2$residuals, pch=20, main='')
qqline(CSB_fit2$residuals)
summary(CSB_fit2) ; sd(CSB_fit2$residuals)
par(mfrow=c(1,1))
```

# MARS nonlinear regression

```
install.packages('earth') ; library(earth)
CSB_fit3 <- earth(Cent.surf.bright~.-Cent.surf.bright, data=GCdyn) ; CSB_fit3
sd(CSB_fit3$residuals)
qqnorm(CSB_fit2$residuals) ; qqline(CSB_fit2$residuals)
```
Figure 8.5: Multiple linear regression for predicting globular cluster central surface brightness involving six covariates. Top: Comparison of observed and predicted values. Bottom: Normal quantile-quantile plot of residuals showing the quality of the fit.
This regression fit is excellent; the predicted values are very close to the observed values, as shown in Figure 8.5. The standard deviation of the residuals is only 5% of the original scatter in the central surface brightness values, and the distribution of residuals from R’s qqnorm function show only small deviations from normality. Qualitatively, the dependencies found in the regression are readily seen on the pairwise bivariate scatter plots (Figure 8.3). Some are easily understood astrophysically — the central surface brightness of a cluster scales with the central star density and with the cluster gravitational potential measured by two nearly-redundant variables, \( V_{\text{esc}} \) and \( S_0 \). The inverse dependency on absorption, measured by \( E(B - V) \) and \( (B - V) \) may reflect the dynamical expansion of the cluster from tidal stresses near the Galactic Center. Perhaps for similar reasons, highly elliptical clusters have lower central surface brightnesses than spherical clusters.

In light of curved and thresholded relationships in the pairs plots, we examine the performance of the MARS nonlinear regression method. It is implemented in the CRAN package earth with function earth (Milborrow 2010). The regression relationships are considerably different than in the linear regression. Seven terms are obtained involving only three covariates: \( \log \rho_{\text{cen}} \), \( E(B - V) \), and \( \log t_{\text{rad}} \). But the performance of the fit is very similar with standard deviation of the residuals 6% of the original scatter. The quantile-quantile plot of residuals now shows no deviation from normality. We thus find that a nonlinear prediction of globular cluster central surface brightness need not depend on dynamical escape velocity, vertical height of the horizontal branch, or ellipticity. As we do not want our astrophysical interpretation to depend on whether the analysis used linear or nonlinear models, further investigation is needed before a clear understanding of the dataset is needed.

### 8.7.6 Scope of R and CRAN

As with most statistical packages, base R has extensive coverage of multivariate analysis, particularly in its MASS library. The cancor and corresp functions give canonical correlations between two datasets, cmdscale gives multidimensional scaling, cor and cor.test give correlation matrices, cov and cov.wt give unweighted and weighted covariance matrices, dist and mahalanobis compute the distance and Mahalanobis matrices, factanal provides a maximum-likelihood factor analysis, gam fits generalized additive models, glm fits generalized linear models, lda computes a linear discriminant analysis, lm fits linear models, lqs and rlm provides robust regression estimators, manova computes multivariate analysis of variance tests, mvrnorm simulates from a MVN distribution, nls gives a weighted nonlinear least squares regression, parcoord provides a parallel coordinates plot, ppr fits projection pursuit regression, princomp and pcomp give principal components analyses, proj returns projections of a matrix onto a linear model, qda gives a quadratic discriminant analysis, and sammon provides a non-metric multidimensional scaling. Many of these functions are accompanied by specialized summary, plot and print functions to assist in understanding the results.
Many CRAN packages provide specialized treatments of multivariate data; an updated annotated list is available at http://cran.r-project.org/web/views/Multivariate.html. The mvnormtest package gives a nonparametric test for multivariate normality. The dr package assists with reduction of dimensionality for large-dimensionality problems. The mvoutlier package provides a suite of robust methods including outlier identification, robust estimates of location and scatter, and robust Mahalanobis distances. Attractive biplots for PCA are given by bpcal. The glmnet, lars and lasso2 packages compute lasso and LARS linear regression models.

CRAN’s chemometrics package contains a useful collection of functions for physical scientists from the text of Varmuza and Filzmoser (2009). Several functions divide the sample into learning and test subsets to compute cross-validation estimators for multiple regression, Lasso regression, partial least squares, and ridge regression. PCA is computed using an efficient NIPALS algorithm with cross-validation for selecting significant components. Helpful graphical outputs are provided. A wide range of multivariate techniques for ecology and environmetrics are described at http://cran.r-project.org/web/views/Environmetrics.html.

R has a wide range of visualization capabilities for three or more dimensions of multivariate datasets. While base R has simple display functions, the most effective tools provide user interaction that are available when powerful graphical interfaces are linked to R. Packages with advanced capabilities — latticist, rggobi, and rattle — are presented in Chpt. ?? We mention a few other multivariate graphics capabilities here.

The lattice package incorporated into base R includes cloud, a static display function for 3-dimensional data points. The CRAN package rgl is a three-dimensional visualization device for R based on the well-established OpenGL application programming interface. The rgl rendering environment is widely used in other CRAN packages. It displays a cloud of 3-dimensional data points or surface, permitting interactive changes in aspects and scale, but does not have a brushing capability. Outputs from cloud and rgl are shown in Figure 8.6. CRAN’s scatterplot3d also gives a variety of 3-dimensional plots.

The parallel coordinate plot (PCP, Inselberg 2009) is implemented with the R functions parallel and parcoord. R’s lattice package includes the splom package that gives an informative combination of conditional scatter plots and PCPs. The variables are represented as a series of evenly spaced parallel lines, each a real line representing the range of a single variable. Each p-dimensional data point is represented as a line segment with a node at each of the lines. The result is that the multivariate behavior of a dataset can be compactly represented on a single large graph. The tool is particularly effective for identifying clusters and outliers, and is less effective for showing multivariate relationships because the sequence of the variables is not determined by any statistical process. The PCP for the globular cluster dynamical variables is shown in Figure 8.6. Other PCP tools are provided in CRAN’s iplots and rggobi packages.

Finally, the CRAN package abind provides the useful service of binding together
two matrices of arbitrary dimensions, generalizing *R*'s `cbind` and `rbind` functions.

```r
# Multivariate display techniques:
# interactive 3-dim scatter plot; 4-dim bubble plot; parallel coordinates plot

library(rgl)
open3d() ; plot3d(GCdyn[,5:7]) ;
snapshot3d('/Users/e5f/Desktop/CASt/Book/13_Multivariate_analysis/Figures/GCdyn3dsnap.png')

library(lattice)
cloud(GCdyn[,5]~GCdyn[,6]*GCdyn[,7], screen=list(z=60,x=45,y=20),
     xlab='log.t.rad', ylab='log.rho.cen', zlab='conc', col=1, pch=1, cex=GCdyn[,8]+1)

parallel(~GCdyn, col=rainbow(50))
```
Figure 8.6: Various multivariate displays of globular cluster dynamical variables: (a) ‘Cloud’ scatter plot of three variables with symbol size scaled to a fourth variable; (b) snapshot from the *rgl* 3-dimensional real-time rendering device; and (c) 1+1 dimensional parallel coordinates plot of all variables.
Multivariate clustering and classification using R/CRAN
for astronomical photometry

Eric Feigelson

Unsupervised clustering of COMBO-17 galaxy photometry

We illustrate unsupervised clustering algorithms using a two-dimensional color-magnitude diagram constructed from the COMBO-17 (‘Classifying Objects by Medium-Band Observations in 17 Filters’) photometric survey of normal galaxies (Wolf et al. 2003). The R script below starts with the which function to filter the dataset, keeping only low-redshift galaxies with z < 0.3 and remove a few points with bad data values. Most of the original 65 variables are ignored, and we keep only the galaxy absolute magnitude in the blue band, M_B, and the ultraviolet-to-blue color index, M_{280} - M_B. The resulting color-magnitude diagram (left panel) shows the well-known concentrations of luminous red galaxies around (M_B,M_{280-M_B} \simeq (-16,-0.2) and fainter blue galaxies around (-13,-0.9).

The clusters are more clearly seen after smoothing the point process with a two-dimensional kernel density estimator using kde2d in R's MASS library. The blue galaxies are spirals and irregular galaxies that have experienced recent active star formation, while the red galaxies are mostly ellipticals that have only older stars formed early in the Universe's history. Note that many galaxies have properties distributed around the major concentrations. For example, a few extremely luminous and red galaxies are seen around (-20,1.5); these are nearby examples of the 'luminous red galaxies' that are very important in cosmological studies (Eisenstein et al. 2005).

While the kernel density estimator provides a valuable visualization of the clustering pattern, it does not assign individual galaxies to specific clusters. We illustrate unsupervised clustering of the dataset using three methods in the following R script. For the nonparametric procedures where we assume a Euclidean distances between points in the 2-space, we first standardize the variables by removing the means and dividing by the standard deviations.

# Color-magnitude diagram for low-redshift COMBO-17 galaxies

COMBO=read.table('http://astrostatistics.psu.edu/MSMA/datasets/COMBO17.dat',header=T,fill=T)
dim(COMBO) ; names(COMBO)
loz_index=which((COMBO[,6]<0.3) & (COMBO[,12]<0) &
(COMBO[,28]<0))
COMBO_loz=cbind(COMBO[loz_index,12],COMBO[loz_index,28] -
COMBO[loz_index,12])
names(COMBO_loz)=c('M_B','M_280') ; dim(COMBO_loz) ; summary(COMBO_loz)
par(mfrow=c(1,2))
plot(COMBO_loz,pch=20,cex=0.5,xlim=c(-22,-7), ylim=c(-2,2.5),xlab='M_B (mag)',ylab='M_280 - M_B (mag)',main='COMBO-17 galaxies (z<0.3)')

# Two-dimensional kernel-density estimator
library(MASS)
COMBO_loz_sm=kde2d(COMBO_loz[,1],COMBO_loz[,2],
h=c(1.6,0.4),lims=c(-22,-7,-2,2.5), n=500)
image(COMBO_loz_sm, col=grey(13:0/15), xlab='M_B (mag)', ylab='M_280 - M_B (mag)',
, xlim=c(-22,-7), ylim=c(-2,2.5), xaxp=c(-20,-10,2))
par(mfrow=c(1,1))

R and CRAN have a variety of agglomerative hierarchical clustering algorithms. We start with the most commonly used procedure, function \texttt{hclust} in base-R. The procedure runs on the matrix of pairwise distances between points constructed using the function \texttt{dist}. As the structures in the smoothed distribution seem roughly spherical, we choose the ‘complete linkage’ definition of group locations.

# Standardize variables
Mag_std=(COMBO_loz[,1]-mean(COMBO_loz[,1]))/sd(COMBO_loz[,1])
Color_std=(COMBO_loz[,2]-mean(COMBO_loz[,2]))/sd(COMBO_loz[,2])
COMBO_std=cbind(Mag_std,Color_std)
plot(COMBO_std)

# Hierarchical clustering
COMBO_dist=dist(COMBO_std)
COMBO_hc=hclust(COMBO_dist, method='complete')
COMBO_coph=cophenetic(COMBO_hc)
cor(COMBO_dist,COMBO_coph)
plot(COMBO_dist,COMBO_coph,pch=20,cex=0.3)

# Cutting the tree at $k=5$ clusters
plclust(COMBO_hc,label=F)
COMBO_hc5a=rect.hclust(COMBO_hc,k=5,border='black')
str(COMBO_hc5a)
COMBO_hc5b=cutree(COMBO_hc,k=5)
plot(COMBO_loz,pch=(COMBO_hc5b),cex=1.0,xlab='M_B (mag)',
ylab='M_280 - M_B (mag)',main='COMBO-17 hier clustering (k=5)',
cex.lab=1.3,cex.axis=1.3)

There is no formal procedure to select branches of the dendrogram as physically valid clusters. The cophenetic correlation coefficient, a measure of the similarity of the hierarchical structure and the data, is 0.52 using functions \texttt{cophenetic} and \texttt{cor}, but this can not readily be converted to a probability. We investigated the tree by trial-and-error, and found that ‘cutting the tree’ at $k=5$ clusters provides a useful result. Two procedures are shown here: \texttt{rect.hclust} that shows rectangles in the dendrogram (top panel), and \texttt{cutree} which gives an output with individual galaxy memberships of the five clusters. These are shown as different symbols in the color-magnitude diagram of the bottom panel; the open triangles show the red galaxies and the open circles show the blue galaxies. These clusters include many outlying galaxies, and examination of smaller clusters in the hierarchy does not cleanly discriminate the cluster extents seen in the smoothed distribution seen in the earlier figure (left panel).

Our second clustering method attempts to alleviate this problem with the hierarchical clustering results by using the density as a starting point for the clustering algorithm. We use the DBSCAN (density-based cluster analysis) in CRAN package \texttt{fpc} (‘fixed point clusters’) which implements the procedure of Ester et al. (1996). DBSCAN is widely used, particularly for problems where compact clusters of interest are embedded in multiscale structure. The \texttt{dbscan} function
requires user input of two parameters: the minimum number of points within a radius (or ‘reach’) associated with the clusters of interest. By trial-and-error, we found that a minimum of 10 points within 0.3 standardized magnitude units provided a useful result as shown in the figure. Here only the fraction of galaxies lying within the regions satisfying this local density criterion are classified; red and blue galaxy groups are clearly discriminated, and intermediate galaxies are not classified.

# Density-based clustering algorithm

```r
install.packages('fpc') ; library(fpc)
COMBO_dbs =
dbscan(COMBO_std,eps=0.1,MinPts=5,method='raw')
print.dbscan(COMBO_dbs) ; COMBO_dbs$cluster
plot(COMBO_loz[COMBO_dbs$cluster==0,],pch=20,cex=0.7,xlab='M_B (mag)',ylab='M_280 - M_B (mag)',main='COMBO-17 density wt clustering',cex.lab=1.3,cex.axis=1.3)
points(COMBO_loz[COMBO_dbs$cluster==2,],pch=2,cex=1.0)
points(COMBO_loz[COMBO_dbs$cluster==1 | COMBO_dbs$cluster==3,],pch=1,cex=1.0)
```

Our third clustering method is the well-respected parametric `mclust` (‘model-based clustering’) package in CRAN that fits a multivariate normal (MVN) mixture model by maximum-likelihood estimation using the EM Algorithm with Bayesian regularization (Fraley & Raftery 2002, 2007). We run an unsupervised procedure, but the calculation can be initialized with the output of MVN hierarchical clustering and the user can specify conjugate priors for the means and variances. In function `mclustBIC`, the ’VVV’ model name specifies multivariate ellipsoidal Gaussians with arbitrary orientations. Model selection is performed by maximizing the Bayesian Information Criterion (BIC) for different number of clusters.

The model-based clustering algorithm is shown in the figure. The likelihood for the COMBO-17 color-magnitude diagram is maximized for three clusters, two of which distinguish the red and blue galaxy sequences. Detailed results are provided by the `summary.mclustBIC` function including the probabilities of cluster membership for each galaxy and the uncertainties to these probabilities.

Points lying between two clusters can be investigated using a visualization tool known as ‘shadow’ and ‘silhouette’ plots coupled to centroid-based partitioning cluster analysis (Leisch 2009). Each data point has a shadow value equal to two-times the distance to the closest cluster centroid divided by the sum of distances to closest and second-closest centroids. Points with shadow values near unity lie equidistant from the two clusters. Silhouette values measure the difference between the average dissimilarity of a point to all points in its own cluster to the smallest average dissimilarity to the points of a different cluster. Small values again indicate points with ill-defined cluster memberships. These plots can be constructed using CRAN’s `flexclust` package.

# Model-based clustering

```r
library(mclust)
COMBO_mclus=mclustBIC(COMBO_loz,modelNames='VVV')
plot(COMBO_mclus,col='black')
COMBO_sum_mclus=summary.mclustBIC(COMBO_mclus,COMBO_loz,3)
COMBO_sum_mclus$parameters ;
COMBO_sum_mclus$classification
COMBO_sum_mclus$z ; COMBO_sum_mclus$uncertainty
plot(COMBO_loz,pch=(19+COMBO_sum_mclus$classification),cex=1.0,xlab='M_B (mag)',ylab='M_280 - M_B (mag)',main='COMBO-17 MVN model clustering (k=3)',cex.lab=1.3,cex.axis=1.3)
```
Supervised classification of SDSS point sources

The Sloan Digital Sky Survey (SDSS) has produced some of the most impressive photometric catalogs in modern astronomy. A selection of 17,000 SDSS point sources, along with training sets for three spectroscopically confirmed classes (main sequence plus red giant stars, quasars, and white dwarfs). These are 4-dimensional datasets with variables representing the ratios of brightness in the five SDSS photometric bands (u-g, g-r, r-i, and i-z). The resulting color-color scatterplots show distributions that cannot be well-modeled by multinormal distributions, and distributions that are distinct in some variables but overlapping in others. The analysis here starts with the SDSS_train and SDSS_test obtained using the following R script.

```r
# SDSS point sources dataset, N=17,000 (mag<21, point sources, hi-qual)
SDSS = read.csv('http://astrostatistics.psu.edu/MSMA/datasets/SDSS_test.csv', header=T)
dim(SDSS); summary(SDSS)
SDSS_test = data.frame(cbind((SDSS[,1]-SDSS[,2]),(SDSS[,2]-SDSS[,3]),(SDSS[,3]-SDSS[,4]),(SDSS[,4]-SDSS[,5])))
names(SDSS_test) = c('u_g','g_r','r_i','i_z')
str(SDSS_test)

par(mfrow=c(1,3))
plot(SDSS_test[,1],SDSS_test[,2],xlim=c(-0.7,3),ylim=c(-0.7,1.8),pch=20,
cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='u-g (mag)',ylab='g-r (mag)')
plot(SDSS_test[,2],SDSS_test[,3],xlim=c(-0.7,1.8),ylim=c(-0.7,1.8),pch=20,
cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='g-r (mag)',ylab='r-i (mag)')
plot(SDSS_test[,3],SDSS_test[,4],xlim=c(-0.7,1.8),ylim=c(-1.1,1.3),pch=20,
cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='r-i (mag)',ylab='i-z (mag)')
par(mfrow=c(1,1))

# Quasar training set, N=2000 (Class 1)
temp1 = read.table('http://astrostatistics.psu.edu/MSMA/datasets/SDSS_QSO.dat', header=T)
dim(temp1); summary(temp1)
qso = cbind(temp1[,c(5,7,9,11,13,6,8,10,12,14,2,3)]) # set same variables in both datasets
bad_phot_qso = which(qso[,1:6] > 21.0 | qso[,9]==0)
qso1 = qso[-bad_phot_qso,]
qso2 = qso1[1:2000,]
qso3 = cbind(qso2[,1]-qso2[,2],qso2[,2]-qso2[,3],qso2[,3]-qso2[,4],qso2[,4]-qso2[,5])
qso_train = data.frame(cbind(qso3,rep(1,length(qso2[,1]))))
names(qso_train) = c('u_g','g_r','r_i','i_z','Class')
dim(qso_train); summary(qso_train)

# Star training set, N=5000 (Class 2)
temp2 = read.csv('http://astrostatistics.psu.edu/MSMA/datasets/SDSS_stars.csv', header=T)
```

file:///Users/e5f/Desktop/R%20for%20Astronomers%202011/ClusClass.html
dim(temp2); summary(temp2)

star = cbind((temp2[,1]-temp2[,2]),(temp2[,2]-temp2[,3]),(temp2[,3]-temp2[,4]),(temp2[,4]-temp2[,5]))

star_train = data.frame(cbind(star, rep(2, length(star[,1]))))

names(star_train) = c('u_g', 'g_r', 'r_i', 'i_z', 'Class')
dim(star_train); summary(star_train)

# White dwarf training set, N=2000 (Class 3)

temp3 = read.csv('http://astrostatistics.psu.edu/MSMA/datasets/SDSS_wd_Eisenstein06.csv', header=T)
dim(temp3); summary(temp3)
temp3 = na.omit(temp3)
wd_train = data.frame(cbind(wd, rep(3, length(wd[,1]))))

names(wd_train) = c('u_g', 'g_r', 'r_i', 'i_z', 'Class')
dim(wd_train); summary(wd_train)

# Combined training set (9000 objects)

SDSS_train = data.frame(rbind(qso_train, star_train, wd_train))

names(SDSS_train) = c('u_g', 'g_r', 'r_i', 'i_z', 'Class')

str(SDSS_train)

par(mfrow=c(1,3))

plot(SDSS_train[,1], SDSS_train[,2], xlim=c(-0.7, 1.8), ylim=c(-0.7, 1.8), pch=20, col=SDSS_train[,5], cex=0.6, cex.lab=1.5, cex.axis=1.5, main='', xlab='u-g (mag)', ylab='g-r (mag)')

legend(-0.5, 1.7, c('QSO', 'MS + RG', 'WD'), pch=20, col=c('black', 'red', 'green'), cex=1.8)

plot(SDSS_train[,2], SDSS_train[,3], xlim=c(-0.7, 1.8), ylim=c(-0.7, 1.8), pch=20, col=SDSS_train[,5], cex=0.6, cex.lab=1.5, cex.axis=1.5, main='', xlab='g-r (mag)', ylab='r-i (mag)')

plot(SDSS_train[,3], SDSS_train[,4], xlim=c(-0.7, 1.8), ylim=c(-0.7, 1.8), pch=20, col=SDSS_train[,5], cex=0.6, cex.lab=1.5, cex.axis=1.5, main='', xlab='r-i (mag)', ylab='i-z (mag)')

par(mfrow=c(1,1))

Unsupervised clustering fails to recover the known distributions in the SDSS photometric distribution. We show, for example, the result of a k-means partitioning in the figure. The k-means partitioning with divides the main sequence into segments, even though there are no gaps in the distribution. Even a supervised k-means partitioning with three initial cluster roughly centered on the three training classes does not lead to a correct result. Similar problems arise when unsupervised hierarchical clustering (R function hclust) or model-based clustering (function Mclust in package mclust) are applied to the SDSS distributions.

# Unsupervised k-means partitioning

SDSS_kmean = kmeans(SDSS_test, 6)

plot(SDSS_test[,1], SDSS_test[,2], xlim=c(-0.5, 3), ylim=c(-0.5, 2), pch=20, col=SDSS_kmean$cluster, cex=0.6, col=SDSS_kmean$cluster, cex=0.6,
Discrimination analysis and k-nn classification

The classification of SDSS objects is much improved when the training set used. We show here the result of linear discriminant analysis (LDA) using function `lda` in base-R's MASS package. The LDA classification from the training set is applied to the test set using R's `predict` function. The figure (top left panel) shows the result for the test sample; a similar plot can be inspected for the training sample.

Discriminant analysis gives a reasonable classification of stars, quasars and white dwarfs, with no difficulty following the elongated and curved distributions in 4-space. However, some classification errors are evident: a few main sequence stars are mislabeled as quasars (black dots), and the white dwarf class (green dots) is truncated by the quasar distribution. The closely related quadratic discriminate analysis using function `qda` has additional problems, classifying some main sequence and red giant stars as white dwarfs.

CRAN package `class` implements a k-nearest neighbors classifier where a grid of classifications is constructed from the training set. The application to the SDSS test set is shown in the figure (top right panel) and shows good performance. Here we use k=4 neighbors, but the result is not sensitive to a range of k values.

We consider two ways to examine the reliability of the these classifiers by applying it to the training set. First, a cross-validation experiment can be made (e.g., using function `knn.cv`) where leave-one-out resamples of the test dataset give posterior probabilities for the classification of each object. Second, the class obtained by the classified can be plotted against the true class known for the training set objects. We show this in the bottom panels of the figure; the LDA clearly makes more misclassifications than the k-nn algorithm. For k-nn, misclassification of stars (Class 2) is rare (0.1%) but confusion between quasars (Class 1) and white dwarfs (Class 3) occurs in about 2% of cases. This is understandable given the overlap in their distributions in color-color plots. Note the use of R's `jitter` function to facilitate visualization of categorical data for scatterplots.

# Linear discriminant analysis

```r
library(MASS)
SDSS_lda=lda(SDSS_train[,1:4],as.factor(SDSS_train[,5]))
SDSS_train_lda=predict(SDSS_lda,SDSS_train[,1:4])
SDSS_test_lda=predict(SDSS_lda,SDSS_test[,1:4])
par(mfrow=c(1,2))
plot(SDSS_test[,1],SDSS_test[,2],xlim=c(-0.7,3),ylim=c(-0.7,1.8),pch=20, col=SDSS_test_lda$class, cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='u-g (mag)',ylab='g-r (mag)')
```

# k-nn classification

```r
install.packages('class') ; library(class)
SDSS_knn4 = knn(SDSS_train[,1:4],SDSS_test,SDSS_train[,5],k=4,prob=T)
plot(SDSS_test[,1],SDSS_test[,2],xlim=c(-0.7,3),ylim=c(-0.7,1.8),pch=20, col=SDSS_knn4, cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='u-g (mag)',ylab='g-r (mag)')
par(mfrow=c(1,1))
```

# Validation of k-nn classification
Machine learning classifiers perform well for this problem. In the following R script, we apply CART using \texttt{rpart} (acronym for \textquote{recursive partitioning and regression trees}) in base-R's \texttt{rpart} library, and the Support Vector Machine \texttt{svm} implemented in CRAN's \texttt{e1071} package. The procedure for running these and similar classifiers is straightforward. The \textquote{model}'s produced by \texttt{rpart} or \texttt{svm} with a formula like \textquote{Known_classes ~ .} to the training set. Examining the model using \texttt{summary} and \texttt{str} shows that the classifier output can be quite complicated; e.g., CART will give details on the decision tree nodes while SVM will give details on the support vectors. But the model predictions can be automatically applied to the training and test datasets using R's \texttt{predict} function without understanding these details.

We plot the predicted classes against the known classes for the training set in the figure. CART does not perform as well as the k-nn shown above, but the SVM classifier does a better job. The figure shows the CART tree with the splits labeled, and the next figure shows how much of the variance is reduced by each split of the data.

Considering the SVM classification as the best available, we show the final classifications of the test SDSS sample in the figure, and write them to an ASCII output file \texttt{SDSS\_test\_svm.out}. Note that R's \texttt{write} function produces tables that are difficult to read; we use the \texttt{format} function and other options in \texttt{write} to improve the appearance of the ASCII output.

```r
library('rpart')
SDSS_rpart_mod = rpart(SDSS_train[,5] ~ ., data=SDSS_train[,1:4])
SDSS_rpart_test_pred = predict(SDSS_rpart_mod, SDSS_test)
SDSS_rpart_train_pred = predict(SDSS_rpart_mod, SDSS_train)
summary(SDSS_rpart_mod) ; str(SDSS_rpart_mod)
par(mfrow=c(1,2))
plot(jitter(SDSS_rpart_train_pred,factor=5),jitter(as.numeric(SDSS_train[,5]),factor=0.5),pch=20,cex=0.3,cex.axis=1.5, cex.lab=1.5,xlab='CART class',ylab='True
class',xaxp=c(1,3,2),yaxp=c(1,3,2))
plot(SDSS_test[,1],SDSS_test[,2],xlim=c(-0.7,3),ylim=c(-0.7,1.8),pch=20, col=round(SDSS_rpart_test_pred), cex=0.6,cex.lab=1.5,cex.axis=1.5,main=' ',xlab='u-g (mag)',ylab='g-r (mag)')
par(mfrow=c(1,1))
```
plot(SDSS_rpart_mod); text(SDSS_rpart_mod)
plotcp(SDSS_rpart_mod,lwd=2,cex.axis=1.3,cex.lab=1.3)

# Support Vector Machine model, prediction and validation

install.packages('e1071'); library(e1071)
SDSS_svm_mod = svm(SDSS_train[,5] ~.,data=SDSS_train[,1:4],cost = 100, gamma = 1)
summary(SDSS_svm_mod); str(SDSS_svm_mod)
SDSS_svm_test_pred = predict(SDSS_svm_mod, SDSS_test)
SDSS_svm_train_pred = predict(SDSS_svm_mod, SDSS_train)
par(mfrow=c(1,2))
plot(SDSS_svm_train_pred,jitter(SDSS_train[,5]),pch=20,cex=0.3,cex.axis=1.5,
cex.lab=1.5,xlab='SVM class',ylab='True class',yaxp=c(1,3,2))
plot(SDSS_test[,1],SDSS_test[,2],xlim=c(-0.7,1.8),ylim=c(-0.7,1.8),pch=20, col=round(SDSS_svm_test_pred),
cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='u-g (mag)',ylab='g-r (mag)'
)

# Final SVM classification of the test set

par(mfrow=c(1,3))
plot(SDSS_test[,1],SDSS_test[,2],xlim=c(-0.7,1.8),ylim=c(-0.7,1.8),pch=20,
cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='u-g (mag)',ylab='g-r (mag)'
)
plot(SDSS_test[,2],SDSS_test[,3],xlim=c(-0.7,1.8),
col=round(SDSS_svm_test_pred),ylim=c(-0.7,1.8),pch=20,
cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='g-r (mag)',ylab='r-i (mag)'
)
plot(SDSS_test[,3],SDSS_test[,4],xlim=c(-0.7,1.8),
col=round(SDSS_svm_test_pred),ylim=c(-1.1,1.3),pch=20,
cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='r-i (mag)',ylab='i-z (mag)'
)
par(mfrow=c(1,1))

SDSS_test_svm_out =
cbind(SDSS[,6],SDSS[,7],SDSS_test,SDSS_svm_test_pred)
names(SDSS_test_svm_out)[c(1,2,7)] = c('R.A.', 'Dec', 'SVM Class')
# write.table(format(SDSS_test_svm_out),
file='SDSS_test_svm.out',sep='\t',quote=F)
Censoring and truncation in astronomical surveys

R for Astronomers
2011

Some astronomical surveys
Statistical treatment of censoring
Statistical treatment of truncation

Star counts: The first flux limited surveys

For a uniform population of objects distributed randomly in transparent space:

\[ S = \frac{L}{4 \pi D^2} \]
\[ V = \frac{4}{3} \pi D^3 \]
\[ N \sim V \sim S^{-3/2} \]

William Herschel (1785) used deviations from this prediction to infer that the Universe (now known as our Milky Way Galaxy) is limited in extent (~1 kpc) and is elongated in shape.
Selection bias in flux-limited surveys

Astronomers often struggle to detect faint characteristics of celestial populations and fail. Many surveys are flux-limited \((F=L/4\pi d^2)\), and are limited to detecting the closer and/or more luminous members of a population. This leads to biased samples: at large distances, high-luminosity objects are over-represented (e.g., the majority of \(V<2\) stars are giants and supergiants). Two types of bias in flux-limited surveys:

1. A 'blind' astronomical survey of a portion of the sky is thus truncated at the sensitivity limit, where truncation indicates that the undetected objects, even the number of undetected objects, are entirely missing from the dataset.

2. In a 'supervised' astronomical survey where a particular property (e.g., IR emission with Herschel, HCO\(^+\) line emission with ALMA, redshifted Ly\(\alpha\) emission with HETDEX) of a previously defined sample of objects is sought, some objects in the sample may be too faint to detect. This gives upper limits or left-censored datapoints.

Statistical challenges of censoring & truncation

In a truncated or censored sample, neither the same mean nor the median converge to the population values. The sample distribution (e.g., a Pareto luminosity function \(N(L) \sim L^{-\alpha}\)) will not converge to the population distribution because faint objects are underrepresented. Relationships among variables (e.g., \(L_{\text{opt}} \sim L_{\text{radio}}^{-\alpha}\)) may be less affected, but sample correlations will be biased unless nondetections are adequately treated.
Survival analysis
A large field of applied statistics called **survival analysis** developed
during 1950-80s to treat right-censoring in several applications:

1. **Life insurance** To calculate annuities, Edmund Halley (1692)
constructed ‘life tables’ from birth/death records in a city. But some
people leave the city; this leads to right-censoring in their survival time.
(Need: Univariate distribution function)

2. **Industrial reliability** GE manufactures Widget Mark IV. To find
improvement over Mark III, operate 100 widgets until 20% fail. From
failure times, engineers find 50% improved lifetime. (Need: 2-sample test)

3. **Biometrics** Reynolds Tobacco wants to test the effect of smoking on
cancer rates. To compare longevities, 100 rats are given smoke at 0-5
cigarette packs/day. After 1 year, the experiment is stopped but some
rats are still alive with right-censored survivals. (Need: Regression)

4. **Astronomy** The EVLA seeks 21cm hydrogen line emission from starburst
galaxies. Due to low star formation rate and/or large distance, half are
not detected. Compare to LINERs and infrared dust emission. (Need:
distribution function, 2-sample test, regression)

---

**Statistical foundations of survival analysis**

The **survival function** $S(x)$ gives the probability that an object has
a value of $X$ above a specified value $x$ (the inverse of the e.d.f.):

$$S(x) = \text{Prob}(X > x) = \frac{\# \text{observation} \geq x}{n}$$

$$= 1 - F(x) = 1 - \int_0^x f(s)ds.$$  

The **hazard rate** gives the probability that an object will have a specified value $x$:

$$h(x) = \frac{f(x)}{S(x)} = \frac{d \ln S(x)}{dx}$$

(Nota the p.d.f. is the product of the survival function and hazard rate.)

**Example: Pareto distribution**

$$f(x) = \frac{\theta \lambda^\theta}{x^{\theta+1}}$$

$$S(x) = \frac{\lambda^\theta}{x^\theta}$$

$$h(x) = \frac{\theta}{x^{\theta+1}}$$
Kaplan-Meier estimator

Likelihoods can be constructed for detected, censored and truncated samples:

\[ L \propto \prod_{\text{det}} f(x_i) \prod_{\text{cens}} (1 - S(x_i)) \prod_{\text{trunc}} f(x_i)/S(x_{\text{trunc}}) \]

For a randomly censored univariate X, the Kaplan-Meier estimator is the unique unbiased nonparametric maximum likelihood estimator of the survival function is

\[ \hat{S}_{KM}(x) = \prod_{x_i \geq x} \left(1 - \frac{d_i}{N_i}\right) \]

where \( d_i \) is the number of occurrences at \( x_i \) (\( d_i = 1 \) if no ties are present) and \( N_i \) is the number of 'at risk' objects left in the sample. An intuitive procedure: construct the e.d.f. of detected points, but increase the step size at low values by redistributing the upper limits to the left.

The Kaplan-Meier estimator is asymptotically normal with variance

\[ \text{Var}(\hat{S}_{KM}) = \hat{S}_{KM}^2 \sum_{x_i \geq x} \frac{d_i}{N_i(N_i - d_i)} \]

Example of univariate survival analysis (Kaplan-Meier)

Feigelson & Nelson 1986
Example:
X-ray luminosity function of quasars from ROSAT All-Sky Survey

R script:
```r
survobj <- Surv(XLum, Xstatus)
KM.XLF <- survfit(survobj ~ 1, conf.int=0.95)
plot(log10(-KM.XLF$time), log10(KM.XLF$surv),
    log-log=TRUE, las=1, xlab="log L_x", ylab="log L_x")
lines(log10(-KM.XLF$time), KM.XLF$upper, lty=2)
lines(log10(-KM.XLF$time), KM.XLF$lower, lty=2)
```

Two-sample tests & regression

Several generalizations of 1930s nonparametric 2-sample tests (e.g. Wilcoxon) were developed in 1960-70s that treat censored values in reasonable fashions:
Gehan, logrank, Peto-Peto, Peto-Prentice tests

Linear regressions developed during 1970-80s:
- Maximum likelihood line assuming Gaussian residuals (using EM Algorithm)
- Buckley-James line assuming nonparametric KM residuals
- Cox regression for multivariate independent variables
- Akritas-Thiel-Sen semi-parametric line
An example of astronomical censored data

Heckman et al. 1989 A millimeter-wave survey of CO emission in Seyfert galaxies

Gehan’s survival 2-sample hypothesis test

What is the chance that two censored datasets do not arise from the same underlying distribution? $H_0 : S_1(x) = S_2(x)$

The Gehan test, developed in 1965 as a generalized Wilcoxon test for survival data, is perhaps the simplest of the censored two-sample tests. For a left-censored Sample 1 with $n$ objects, $x_1^1, x_2^1, \ldots, x_n^1$, and Sample 2 with $m$ objects, $x_1^2, x_2^2, \ldots, x_m^2$, compute the pairwise quantity

\[
U_{ij} =
\begin{cases}
  +1 & \text{if } x_i^1 < x_j^2 \text{ (where } x_i^1 \text{ may be censored)} \\
  -1 & \text{if } x_i^1 > x_j^2 \text{ (where } x_j^2 \text{ may be censored)} \\
  0 & \text{if } x_i^1 = x_j^2 \text{ or if the relationship is ill–determined.}
\end{cases}
\] (10.20)

Gehan’s test statistic is

\[ W_{\text{Gehan}} = \sum_{i=1}^{n} \sum_{j=1}^{m} U_{ij}. \] (10.21)

$W_{\text{Gehan}}$ is asymptotically normal with zero mean. A common estimate of its variance of based on permutations is

\[ \overline{\text{Var}}(W_{\text{Gehan}}) = \frac{mn \sum_{i=1}^{n} U_i^2}{(n + m)(n + m - 1)} \] (10.22)
Bivariate correlation for censored data

Consider a nonparametric hypothesis test for correlation. Helsel proposes a generalizing Kendall’s coefficient based on pairwise comparison of data points, \((x_i, y_i) - (x_j, y_j)\).

\[
\tau_H = \frac{n_c - n_d}{\sqrt{\left( \frac{n(n-1)}{2} - n_{t,x} \right) \left( \frac{n(n-1)}{2} - n_{t,y} \right)}}
\]

where \(n_c\) is the number of pairs with a positive slope in the \((x, y)\) diagram, \(n_d\) is the number of pairs with negative slopes, \(n_{t,x}\) and \(n_{t,y}\) are the number of ties or indeterminate relationships in \(x\) and \(y\) respectively. As the censoring fraction increases, fewer points contribute to the numerator of \(\tau_H\), but the denominator measuring the number of effective pairs in the sample also decreases. So \(\tau_H\) depends on the detailed locations of the censored points.

```
install.packages('NADA'); library(NADA)
cenken_out <- cenken(logN_Bol, Be_con, logN_Li1, Li_con)
abline(a = cenken_out$intercept, b = cenken_out$slope, lwd = 2)
```
Linear regression: Several approaches

1. Iterative least squares, the familiar linear regression model $y = \alpha + \beta x + \epsilon$ where $\epsilon = N(0, \sigma^2)$. (Industrial reliability)
2. Accelerated failure-time model $\log y = \alpha + \beta x + \epsilon$. (Industrial reliability)
3. Tobit regression. (Econometrics)
4. Proportional hazards model (Cox regression) where the hazard rate has an exponential dependence on the covariates, $h(y|x) = h_0(y)e^{\beta x}$. MLE estimation and inference. (Biometrics)
5. Buckley-James line permitting non-Gaussian residuals around the line. $\epsilon$ is estimated in local regions of $x$ using the Kaplan-Meier estimator. (Biometrics)
6. Akritas-Thiel-Sen line for doubly censored data shown in figure above. (Astronomy)

Test of bivariate correlation/regression for simulated flux-limited surveys

Here is a simulation of uncorrelated X-ray and radio luminosities of a hypothetical sample of galaxies. The left panel has infinite sensitivity in the X-ray band, while the right panel shows finite sensitivity giving upper limits. A spurious correlation is found if only detections are considered (dashed line), but no correlation is found if survival methods (e.g., Cox regression) are used.

Isobe, Feigelson & Nelson 1986

Unbiased Buckley-James line including nondetections
Software for survival calculations

Our group produced a stand-alone Fortran 77 code, Astronomy SURVival analysis (ASURV), which has been widely used. Most of its functionalities are in R/CRAN, and we recommend future work be conducted within R. ASURV implements:

Univariate distribution function Kaplan-Meier estimator with confidence limits & quantiles. Assumes random censoring.

Univariate two-sample tests Gehan, logrank, and Peto-Prentice tests. Can treat unusual censoring patterns.

Bivariate correlation coefficient Generalized Kendall’s τ. Can treat censoring in both variables.

Bivariate linear regression MLE assuming normal residuals (EM Algorithm), Buckley-James line treating non-normal residuals, Schmitt’s binned regression.

But we now encourage use of R/CRAN rather than the old ASURV !!

Truncation in astronomical surveys:
The case of the galaxy luminosity function
A recent SDSS study
Volume-limited survey of 28,089 galaxies with d<150 Mpc

Blanton et al. 2005 The Properties and Luminosity Function of Extremely Low Luminosity Galaxies
Normal galaxy luminosity function

Parametric model:
Schechter function related to gamma distribution

\[ \frac{dN}{dL} \sim \Phi \sim L^{-\alpha} e^{-L/L^*} \]

Note that the Schechter function is just the gamma distribution introduced by Pearson (1901) based on the gamma function of Euler & Lagrange.

Nonparametric methods:
two used here (\(1/V_{\text{max}}\), stepwise max. likelihood)

(Many technical issues concerning correction for missing low-surface brightness galaxies, K-correction and evolution-corrections to luminosity, double Schechter fits, etc.)

Nonparametric estimators to LFs

1. Classic estimator:

\[ \Phi(L) = \frac{N(L)}{V} \]

where \(N\) is the number of stars/galaxies/AGNs in surveyed volume \(V\). A biased estimator.

2. Schmidt estimator (~40 citations/yr):

\[ \Phi(L) = \sum \frac{1}{V_{\text{max}}(L_i)} \]

where \(V_{\text{max}}\) is the maximum volume within which an object of the observed flux could have been seen given the survey’s sensitivity limit. Unbiased estimator but with high variance.

Schmidt 1968 Space Distribution and Luminosity Functions of Quasi-Stellar Radio Sources
Felten 1976 On Schmidt’s \(V_m\) estimator and other estimators of luminosity function
Stepwise maximum-likelihood estimator (~10 citations/yr):

$$\ln L = \sum_{i=1}^{K} \sum_{j=1}^{K} W(M_x - M_j) \ln \phi_x - \ln \left[ \sum_{j=1}^{K} \phi_j H(M_{\text{ind}}(z_j) - M_j) \Delta M \right].$$ (11)

where

$$W(M_x - M) = \begin{cases} 1 & \text{for } M_x - \frac{\Delta M}{2} \leq M_x \leq M_x + \frac{\Delta M}{2}, \\ 0 & \text{otherwise.} \end{cases}$$ (8)

$$H(M_{\text{ind}}(z) - M) = \begin{cases} 1 & M_{\text{ind}}(z) - \frac{\Delta M}{2} > M, \\ \frac{M_{\text{ind}}(z) - M + \frac{1}{2}}{\Delta M} & M_{\text{ind}}(z) - \frac{\Delta M}{2} \leq M < M_{\text{ind}}(z) + \frac{\Delta M}{2}, \\ 0 & M_{\text{ind}}(z) + \frac{\Delta M}{2} \leq M, \end{cases}$$ (10)

Set $dL/d\phi = 0$ to obtain

$$\phi_x \Delta M = \sum_{i=1}^{K} W(M_k - M_j) \times \left[ \sum_{j=1}^{K} \phi_j H(M_{\text{ind}}(z_j) - M_j) \Delta M \right]^{-1}. \quad (11)$$

Efstathiou et al. 1988 Analysis of a complete galaxy redshift survey. II - The field-galaxy luminosity function.

Lynden-Bell-Woodrooffe MLE (~5 citations/yr):

Recurison relation:

$$\phi(M) \propto \sum_{k=1}^{M_k < M} \psi_k = \psi_1 \prod_{k=1}^{M_k < M} C_k + 1. \quad (11)$$

where $C_k$ is the number of stars/galaxies/AGNs in the $k$-th rectangle in the luminosity-distance diagram.

Lynden-Bell 1971 A method of allowing for known observational selection in small samples applied to 3CR quasars
Woodroofe 1985 Estimation of a distribution function with truncated data
Takeuchi et al. (2000) apply these LF estimators to Monte Carlo simulations of small galaxy catalogs (N=100 and 1000). All perform well when spatial distribution is homogeneous. But for spatially clustered distributions, the $1/V_{\text{max}}$ estimator is badly affected.

There has been no analytical evaluation of the mathematical properties of these estimators since study of $1/V_{\text{max}}$ by Felten (1976).

The Lynden-Bell-Woodroofe estimator for randomly truncated univariate data is the best choice: unbiased, unbinned, nonparametric maximum likelihood, asymptotically normal. Analog of the Kaplan-Meier estimator for randomly censored univariate data.

---

Methodological work needed for astronomical nondetections

1. Investigate methods for astronomical censoring patterns
   Type 1 censoring in $F=L/4pd^2$ gives nonrandom censoring in $L$

2. Develop multivariate analysis for censoring in all variables
   Clustering, regression/PCA, MANOVA, MARS & neural nets, etc.

3. Calculate nonlinear regression & goodness-of-fit

4. Treat simultaneous censoring & truncation

5. Treat simultaneous censoring & measurement error
   Unlike biometrical and industrial reliability testing environment where survival times are measured precisely, in astronomy the censored value is typically set at the $3\sigma$ upper limit where $\sigma$ is the known noise level. A new statistical approach is needed that treats all observations (detected or not) with measurement errors. See B.C. Kelly (ApJ, 2007) for the beginning of a MLE approach.
6. Development of suite of tools based on the Lynden-Bell-Woodroofe estimator for truncated data

   Two-sample tests, correlation coefficients (Efron-Petrosian), linear regressions, multivariate, ...

References


Spatial Point Processes

R for Astronomers
2011

Astronomical context

Spatial data: data in p-dimensions (typically p=2-3) either a point process (random variables) or a representing a continuous distribution grid (fixed variables)

Spatial point process
• Stars or galaxies in the sky (galaxy clustering ~ Large Scale Structure)
• Orbital elements of asteroids in the Kuiper Belt
• Locations of star streams in the Galactic halo
• Photons in an X-ray or gamma-ray imaging detector

Continuous spatial process
• Velocity distribution across an elliptical galaxy
• Velocity distribution in a turbulent molecular cloud
• Plasma temperature distribution within a rich galaxy cluster
• Fluctuations of the cosmic microwave background

Spatial statistics overlaps other fields like multivariate clustering, image processing, and stochastic processes
**Concepts of spatial point processes**

**Stationarity:** A point process is stationary if its properties are invariant under spatial translation

**Isotropy:** A point process is isotropic if its properties are invariant under rotation

**Intensity:** The intensity of a point process \( \lambda \) is the expected number \( n \) of points per unit volume \( v \) in p-space: \( \mathbb{E}[n] = \lambda \cdot v \).

**Poisson process:** \( n \sim \text{Pois}(\lambda) \)

**Complete spatial randomness (CSR):** A stationary Poisson process. Many astronomical problems can be modeled as a nonstationary (inhomogeneous) Poisson process where the intensity varies with location, \( \lambda(x,y,z) \).

**History and bibliography**

*Most methods of spatial point processes were developed in the context of Earth sciences: trees in a woods (forestry), roads on a map (geography), soils in a terrain (remote sensing), and rock layers underground (geology and mining)*

Example: the spatial interpolation procedure called ‘kriging’ was developed in 1951 by the South African mining engineer Daniel G. Krige.

**Useful references:**

- A. S. Fotheringham & Rogerson (eds.) *The SAGE Handbook of Spatial Analysis* [review articles for geographers]
**Statistics of a stationary point process**

**Mean & variance:**
The unbiased MLE of the intensity of a CSR process is (as expected) the number of points divided by the volume. Mean and variances of locations are the same as for regular variables.

**Global measures of spatial autocorrelation:**
Moran’s I and Geary’s c (1950s) are extensions of Pearson’s linear correlation coefficient to measure departures from CSR. The data are grouped in to \( m \) spatial bins with \( c \) counts weighted by some kernel \( w(r) \):

\[
I(m) = \frac{n}{\sum_{j=1}^{m} (c_j - \bar{c})^2} \sum_{j=1}^{m} \sum_{k=1}^{m} w_{jk} (c_j - \bar{c})(c_k - \bar{c})
\]

Plots of \( I(m) \) are called correlograms; they show the spatial scales where high spatial variance is present. Moran’s I can be measured locally for nonstationary point processes.

The **variogram** (or semi-variogram) is a similar, but unbinned, measure of spatial variance (or second moment) summed over all pairs of points:

\[
\gamma(d) = \frac{1}{2m(d)} \sum_{j=1}^{m(d)} [z(x_j) - z(x_j + d)]^2
\]

For a stationary, isotropic, Gaussian point process, the variogram simplified to the sum of the **nugget** \( \sigma^2 \) (variance at zero distance attributable to measurement error or intrinsic small-scale variations) and the **correlation function** \( \rho \):

\[
V(d) = \sigma^2 [1 - \rho(d)] \quad \rho = \frac{\text{Cov}(S(x), S(x'))}{\sigma^2}
\]

An elaborate toolkit for parametric modeling of variograms is widely used.

**Tobler’s First Law of Geography**
‘Everything is related to everything else, but near things are more related than distant things.’
Variogram of a galaxy redshift survey

Spatial interpolation

If a spatial point process is an observable representation of an underlying continuous distribution, then interpolation is an effective approach to estimating this distribution. The underlying quantity of interest could be the density of objects, where the spatial positions suffice for interpolation.

But often we seek the spatial variation of a mark variable associated with each point. Here interpolation is made between measurements at distinct locations of mass, velocity, temperature, abundance, gravitational potential, or other astrophysical variable.

Inverse distance weighted interpolation

\[
\hat{z}(X_0) = \frac{\sum_{i=1}^{n} z(X_i) d_i^{-\alpha}}{\sum_{i=1}^{n} d_i^{-\alpha}}
\]
Kriging is a suite of linear least-squares, minimum variance methods for density estimation in two or three dimensions with the goal of localizing features of inhomogeneous structures rather than globally characterize homogeneous distributions. The continuous surfaces are derived from discrete data, either evenly or unevenly spaced, by interpolation using the variogram. For a stationary Gaussian spatial process, kriging gives the minimum mean square error (MSE) predictor for the unobserved continuous distribution.
Global measures of clustering

For a stationary and isotropic, but not homogeneous (intensity $\lambda$ is spatial variable, non-CSR), spatial point process, we can quantify the nature of the clustering without identifying individual clusters.

\[
\begin{align*}
G(d) &= \frac{\# \{d_{NN}(s_i) < d \}}{n} \\
F(d) &= \frac{\# \{d_{NN}(p_i, S) < d \}}{n}
\end{align*}
\]

are the cumulative distribution (e.d.f.'s) of nearest neighbor (NN) distances computed either at the data points (G) or random (empty) locations (F). Baddeley’s $J$ is highly resistant to edge effects that tend to underestimate the number of points at large $r$.

\[
J(r) = \frac{1 - G(r)}{1 - F(r)}
\]
Ripley’s $K$ is the average number of points within distance $d$ of the observed points:

$$K(d) = \frac{1}{\lambda n} \sum_{i=1}^{n} \# [ S \in C(s_i, d) ]$$

Since $K(d)$ increases \(^\sim\) quadratically, Besag’s $L^*$ function with stabilize variance is often used:

$$L^*(d) = \sqrt{\frac{K(d)}{\pi}} - d$$

For CSR distributions, the expected values are:

\[
\begin{align*}
E[G(d)] &= E[F(d)] = 1 - e^{-\lambda \pi d^2} \\
E[K(d)] &= \pi d^2 \\
E[L^*(d)] &= 0.
\end{align*}
\]

\[Var[K(d)] = \frac{2\pi d^2}{\lambda^2 A}\]

The $K$ and $L$ functions for low density portion of the Shapley Supercluster (2-dim spatial variables only)

Dashed: CSR
Dotted: Function
Solid: Function with edge correction
The astronomers’ 2-point correlation function

The two-point correlation function counts the number of objects in annuli around each point rather than counts within circles around each point, and is thus related to the differential of the K function.

The joint probability that two points like in small volumes at distance $d$ is

$$dP_{12} = \rho^2 [1 + \xi(d)] dV_1 dV_2$$

For a stationary and isotropic process,

$$\xi(d) = \frac{\lambda_2(d)}{\rho^2} - 1$$

Astronomers call $\xi$ the two-point correlation function, and statisticians call $\xi + 1$ the pair correlation function. Astronomers have several variants that treat edge effects differently using ratios of the observed pair distribution to a CSR distribution in the observed volume.
The relationship between $\xi$ and Ripley’s $K$ is similar to that between a p.d.f. and an e.d.f.:

$$\xi(d) + 1 = \frac{K'(d)}{2\pi d}$$

The two-point correlation function is not widely used by statisticians or scientists in other fields due to difficulties in computation: choice of binning, differentiating $K$, and estimation at small distances.