An R tutorial

Based on tutorials developed by Prof. David Hunter (Statistics) for Penn State Summer Schools in Statistics for Astronomers

Clip & paste R scripts at http://astrostatistics.psu.edu/su07/R

Descriptive statistics

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 tutorial on descriptive statistics serves a dual role as a brief introduction to R. When this tutorial is used online, the indented lines in non-proportional font

```
# like this one
```

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

Reading data into R

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.

```
Sys.info()
R.version
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:

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

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.

```
getwd()
list.files() # what's in this directory?
# The # the comment symbol.
```

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.

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

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.

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.

```
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 mean 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.

```
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:

```
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:

```
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?

```
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:

```
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.

Standard errors and confidence intervals

The standard error of an estimator is, by definition, an estimate of the standard deviation of that estimator. Let's consider an example.

Perhaps the most commonly used estimator is the sample mean (called a *statistic* because it depends only on the data), which is an estimator of the population mean (called a *parameter*). Assuming that our sample of data truly consists of independent observations

of a random variable X, the true standard deviation of the sample mean equals stdev(X)/sqrt (n), where n is the sample size. However, we do not usually know stdev(X), so we estimate the standard deviation of the sample mean by replacing stdev(X) by an estimate thereof.

If the Vmag column (the 2nd column) of our dataset may be considered a random sample from some larger population, then we may estimate the true mean of this population by

```
mean(hip[,2])
```

and the standard error of this estimator is

```
sd(hip[,2]) / sqrt(2719)
```

We know that our estimator of the true population mean is not exactly correct, so a common way to incorporate the uncertainty in our measurements into reporting estimates is by reporting a confidence interval. A confidence interval for some population quantity is always a set of "reasonable" values for that quantity. In this case, the Central Limit Theorem tells us that the sample mean has a roughly Gaussian, or normal, distribution centered at the true population mean. Thus, we may use the fact that 95% of the mass of any Gaussian distribution is contained within 1.96 standard deviations of its mean to construct the following 95% confidence interval for the true population mean of Vmag:

```
mean(hip[,2]) + c(-1.96,1.96)*sd(hip[,2]) / sqrt(2719)
```

In fact, many confidence intervals in statistics have exactly the form above, namely, (estimator) +/- (critical value) * (standard error of estimator).

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 accessed by typing, say,

```
?'+'
```

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:

```
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:

```
a <- c(TRUE,TRUE,FALSE,FALSE);b <- c
```

```
(TRUE,FALSE,TRUE,FALSE)
   !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".

```
   which.min(hip[,4])
   tmp <- (hip[,4]==min(hip[,4]))
   (1: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 this 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.

```
   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.

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).

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

```
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:

```
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:

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

We now 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'.

```
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:

```
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))
```

We can produce boxplots *and* save the output.

```
b <- boxplot(hip[,c(2,6,7,9)])
```

'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:

```
b$names[2]
b$out[b$group==2]
```

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.

Exploratory Data Analysis and Regression

Bivariate exploratory data analysis

We begin by loading the Hipparcos dataset used in the descriptive statistics tutorial, found at http://astrostatistics.psu.edu/datasets/HIP_star.html. Type

```
   hip  <-  read.table("http://astrostatistics.psu.edu/
datasets/HIP_star.dat",
      header=T,fill=T)
   names(hip)
   attach(hip)
```

In the descriptive statistics tutorial, we considered boxplots, a one-dimensional plotting technique. We may perform a slightly more sophisticated analysis using boxplots to get a glimpse at some bivariate structure. Let us examine the values of Vmag, with objects broken into categories according to the B minus V variable:

```
   boxplot(Vmag~cut(B.V,breaks=(-1:6)/2),
      notch=T, varwidth=T, las=1, tcl=.5,
      xlab=expression("B minus V"),
      ylab=expression("V magnitude"),
      main="Can you find the red giants?",
      cex=1, cex.lab=1.4, cex.axis=.8, cex.main=1)
   axis(2, labels=F, at=0:12, tcl=-.25)
   axis(4, at=3*(0:4))
```

The notches in the boxes, produced using "notch=T", can be used to test for differences in the medians (see boxplot.stats for details). With "varwidth=T", the box widths are proportional to the square roots of the sample sizes. The "cex" options all give scaling factors, relative to default: "cex" is for plotting text and symbols, "cex.axis" is for axis annotation, "cex.lab" is for the x and y labels, and "cex.main" is for main titles. The two axis commands are used to add an axis to the current plot. The first such command above adds smaller tick marks at all integers, whereas the second one adds the axis on the right.

Scatterplots

The boxplots in the plot above are telling us something about the bivariate relationship between the two variables. Yet it is probably easier to grasp this relationship by producing a scatter plot.

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

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

```
   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:

```
   plot(RA,DE,pch=".")
```
See the cluster of stars with RA between 50 and 100 and with DE between 0 and 25?
```
   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:
```
   filter1 <-  (RA>50 & RA<100 & DE>0 & DE<25)
```

Next, we select in 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.)
```
   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.
```
   plot(pmRA[filter1],pmDE[filter1],pch=20, xlim=c
(0,200),ylim=c(-150,50))
   rect(90,-60,130,-10,border=2)
   filter2 <- (pmRA>90 & pmRA<130 & pmDE>-60 & pmDE< -10) #
Space in 'pmDE< -10' is necessary!
   filter  <-  filter1 & filter2
```

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

Linear and polynomial regression

Here is how one may reproduce the output seen in the regression lecture, i.e., a linear regression relating BminusV to logL, where logL is the luminosity, defined to be (15 - Vmag - 5 log(Plx)) / 2.5. However, we'll use a different subset of the data than the one seen in the lecture, namely, the main-sequence Hyades:

```
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):

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

For an implementation of the ridge regression technique mentioned in lecture, see the lm.ridge function in the MASS package. To use this function, you must first type library (MASS). For a package that implements LASSO (which uses an L1-penalty instead of the L2-penalty of ridge regression), check out, e.g., the lasso2 package on CRAN.

There is a lot of information contained in regline produced by the lm function that is not displayed by print or summary:

```
names(regline)
```

For instance, the regline$fitted.values and regline$residuals are useful for 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.

Hypothesis testing and bootstrapping

This tutorial demonstrates some of the many statistical tests that R can perform.

T tests

In the exploratory data analysis and regression tutorial, 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:

```
   hip <- read.table("http://astrostatistics.psu.edu/
datasets/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 of the tutorial, 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 t.test function. First, we calculate the variances of the sample means for each group:

```
v1 <- var(H)/92
v2 <- var(nH)/2586
c(var(H),var(nH))
```

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., v1+v2). 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:

```
tstat <- (mean(H)-mean(nH))/sqrt(v1+v2)
tstat
```

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:

```
(v1 + v2)^2 / (v1^2/91 + v2^2/2585)
```

The two-sided p-value may now be determined by using the cumulative distribution function of the t distribution, which is given by the pt function.

```
2*pt(tstat,97.534)
```

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:

```
t.test(H,nH,var.equal=T)
```

Empirical distribution functions

Suppose we are curious about whether a given sample comes from a particular distribution. For instance, how normal is the random sample 'tlist' of t statistics obtained under the null hypothesis in the previous example? How normal (say) are the colors of 'H' and 'nH'?

A simple yet very powerful graphical device is called a Q-Q plot, in which some quantiles of the sample are plotted against the same quantiles of whatever distribution we have in mind. If a roughly straight line results, this suggests that the fit is good. Roughly, a pth quantile of a distribution is a value such that a proportion p of the distribution lies below that value.

A Q-Q plot for normality is so common that there is a separate function, qqnorm, that implements it. (Normality is vital in statistics not merely because many common populations are normally distributed -- which is actually not true in astronomy -- but because the central limit theorem guarantees the approximate normality of sample means.)

```
par(mfrow=c(2,2))
qqnorm(tlist,main="Null luminosity t statistics")
abline(0,1,col=2)
qqnorm(H,main="Hyades")
qqnorm(nH,main="non-Hyades")
```

Not surprisingly, the tlist variable appears extremely nearly normally distributed (more precisely, it is nearly *standard* normal, as evidenced by the proximity of the Q-Q plot to the line x=y, shown

in red). As for H and nH, the distribution of B minus V exhibits moderate non-normality in each case.

Related to the Q-Q plot is a distribution function called the *empirical (cumulative) distribution function*, or EDF. (In fact, the EDF is almost the same as a Q-Q plot against a uniform distribution.) The EDF is, by definition, the cumulative distribution function for the discrete distribution represented by the sample itself -- that is, the distribution that puts mass 1/n on each of the n sample points.

While it is generally very difficult to interpret the EDF directly, it is possible to compare an EDF to a theoretical cumulative distribution function or two another EDF. Among the statistical tests that implement such a comparison is the Kolmogorov-Smirnov test, which is implemented by the R function ks.test.

```
ks.test(H,nH)
```

Whereas the first result above gives a surprisingly small p-value, the second result is not surprising; we already saw that H and nH have statistically significantly different means. However, if we center each, we obtain

```
ks.test(H-mean(H),nH-mean(nH))
```

In other words, the Kolmogorov-Smirnov test finds no statistically significant evidence that the distribution of B.V for the Hyades stars is anything other than a shifted version of the distribution of B.V for the other stars.

Chi-squared tests for categorical data

We begin with a plot very similar to one seen in the exploratory data analysis and regression tutorial:

```
bvcat <- cut(color, breaks=c(-Inf,.5,.75,1,Inf))
boxplot(Vmag~bvcat, varwidth=T,
   ylim=c(max(Vmag),min(Vmag)),
   xlab=expression("B minus V"),
   ylab=expression("V magnitude"),
   cex.lab=1.4, cex.axis=.8)
```

The cut values for bvcat are based roughly on the quartiles of the B minus V variable. We have created, albeit artificially, a second categorical variable ("filter", the Hyades indicator, is the first). Here is a summary of the dataset based only on these two variables:

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

Note that the Vmag variable is irrelevant in the table above.

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:

```
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 (just as we did earlier for the permutation test). Such a method is implemented in the

chisq.test function:

```
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.

Nonparametric bootstrapping of regression standard errors

Let us consider a linear model for the relationship between DE and pmDE among the 92 Hyades stars:

```
x <- DE[filter]
y <- pmDE[filter]
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.

```
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:

```
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:

```
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

There is a boot package in R 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 (for the lqs model of pmDE regressed on DE for the Hyades stars.)

```
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:

```
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:

```
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:

```
summary(model1)
```

Remember that model1 was defined above as lm(y~x). We observe a residual standard error of 4.449.

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

```
y <- 21.9 - 3.007*x + 4.449*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 in the EDA and regression tutorial, 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 in the EDA and regression tutorial can be justified. The

reader is encouraged to implement a parametric bootstrap using the rq function found in the "quantreg" package.

Kolmogorov-Smirnov: Bootstrapped p-values

Earlier, we generated 5000 samples from the null distribution of the t-statistic for testing the null hypothesis that the distribution of MWG luminosities is the same as the distribution of M31 luminosities, shifted by 24.44. Let's do a similar thing for the color comparison between Hyades and non-Hyades:

```
tlist2 <- NULL
all <- c(H,nH)
for(i in 1:5000) {
   s <- sample(2586,92) # choose a sample
   tlist2 <- c(tlist2, t.test(all[s],all[-s],
      var.eq=T)$stat) # add t-stat to list
}
```

Let's look at two different ways of assessing whether the values in the tlist2 vector appear to be a random sample from a standard normal distribution. The first is graphical, and the second uses the Kolmogorov-Smirnov test.

```
plot(qnorm((2*(1:5000)-1)/10000), sort(tlist2))
abline(0,1,col=2)
ks.test(tlist2, "pnorm")
```

The graphical method does not show any major deviation from standard normality, but this graphical "test" is better able to detect departures from an overall normal shape than to detect, say, a shift of the mean away from zero. The following procedures illustrate this fact:

```
plot(qnorm((2*(1:5000)-1)/10000), sort(tlist2)-mean
(tlist2))
abline(0,1,col=2)
ks.test(tlist2-mean(tlist2), "pnorm")
```

The graphical plot shows no discernable difference from before, but we see a vast difference in the new p-value returned by the

Kolmogorov-Smirnov test (!!).

However, let us now consider the p-value returned by the last use of ks.test above. It is not quite valid because the theoretical null distribution against which we are testing depends upon an estimate (the mean) derived from the data. To get a more accurate p-value, we may use a bootstrap approach.

First, obtain the Kolmogorov-Smirnov test statistic from the test above:

```
obs.ksstat <- ks.test(tlist2-mean(tlist2),
   "pnorm")$stat
```

Now we'll generate a new bunch of these statistics under the null hypothesis that tlist2 really represents a random sample from *some* normal distribution with variance 1 and unknown mean:

```
random.ksstat <- NULL
for(i in 1:1000) {
   x <- rnorm(5000)
   random.ksstat <- c(random.ksstat,
      ks.test(x,pnorm,mean=mean(x))$stat)
}
```

Here is a histogram of the test statistics and an estimate a p-value:

```
hist(random.ksstat,nclass=40)
abline(v=obs.ksstat,lty=2,col=2)
mean(random.ksstat>=obs.ksstat)
```

Note that the approximate p-value above is smaller than the original p-value reported by newkstest, though it is still not small enough to provide strong evidence that the tlist2 sample is not normal with unit variance.

The bootstrap procedure above relied on multiple resamples with replacement. Since these samples were drawn from a theoretical population (in this case, a normal distribution with parameters that might be determined by the data), it is considered a *parametric* bootstrap procedure.