Caltech Department of Mathematics

Ma 3
Introduction to Probability and Statistics

R crib sheet*

Contents

1 R . . . 1
2 Installing R and RStudio . . . 2
2.1 Mac OS X . . . 2
3 Directories . . . 2
4 Reading data from a file . . . 3
4.1 Data "frames" . . . 3
5 Basic descriptive statistics . . . 3
6 Some simple plots . . . 4
6.1 Histograms and ecdfs . . . 4
6.2 Labeling plots and other fancy tricks . . . 5
6.3 Overlays . . . 6
6.4 Saving plots to files . . . 6
6.5 Q-Q plots . . . 6
7 Named distributions . . . 8
8 Built-in functions . . . 8
9 Some operations on arrays . . . 9
10 Defining functions . . . 10
11 Graphing functions . . . 10
12 Numerical one-dimensional optimization . . . 11
13 Off-the-shelf tests . . . 12
14 Linear regression . . . 15

1 R

R is an open source alternative to S, which was developed at Bell Labs starting in the 1960s [7, pp. 1–2]. These are the same folks who brought you C and UNIX, so the command names tend to be short and somewhat cryptic. These notes are not intended to be a general introduction to the R language, but rather just enough to do the assignments in this class. For a more detailed course in R I recommend either Härdle, Okhrin, and Okhrin [7] or Hothorn and Everitt [6].

Warning: I am not an R programer (I use MATHEMATICA out of habit), so there are probably better ways to do things, and I am open to suggestions.[1] Most of what I know I learned by Googling various questions. Also—typing `?command` will bring up help on the

*I thank Richard S. Border for his help in preparing these notes.

[1]I do know a post-doc in statistical genetics who acts as my unpaid R consultant.

command `command`, but this only works if you know the command name. The built-in help search function `help` does not always find what I am looking for.

The code that follows can be used in a command file typically given extension `.R`, or entered interactively. Note that R tends to ignore whitespace, so you can write `a=b` or `a = b`, depending on your sense of readability. R uses double quotes `" "` to delimit strings such as file names. Also, I use `=` to assign to variables, as most other languages do, but real R programmers use `<-` for assignment.

2 Installing R and RStudio

You can download an R binary for MacOS X, Linux, or Windows from cran.r-project.org. RStudio is a graphical front end for R. I got the free open-source RStudio Desktop 1.2.5033 from rstudio.com/products/rstudio/download/. Since I use MacOS X, I can explain how to set up R on a Mac. I will ask someone else to write up instructions for other platforms.

2.1 Mac OS X

First download the R binary (about 77 MB) from cran.r-project.org/bin/macosx/. If you are vigilant, you should verify the MD5 or SHA1 checksum. To do this, open a Terminal.app window and at the prompt type

```
shasum
```

with a space at the end, but do not hit the `Return` *key yet!* Next, from the Finder window, drag the downloaded file to the Terminal window and drop it. Upon dropping the dragged file, the Terminal should fill in the path to the file. (You could have typed it in yourself if you were so inclined.) *Now* hit the `Return` key. The Terminal window will show a long apparently random string, which should match the SHA1-hash listed on the download web site. Using the `md5` command instead of `shasum` will compute the MD5-hash.

After verifying the download's integrity, you may double-click the installer package, and follow the instructions. This will also install R.app, which is a console for running R. You should be able to run R from the Terminal command line by typing `R` at the prompt. The full path is /Library/Frameworks/R.framework/Resources/bin/R. Remember that `q()` quits the program. (Note the empty parentheses.)

You can also get RStudio from rstudio.com/products/rstudio/download/ (about 127 MB), verify the appropriate hash, and double-click the disk image. Drag the app to the Applications folder. Double-clicking the app should now open RStudio.

At least this worked for me.

3 Directories

First, use

```
setwd("your_data_pathname")
```

to change your working directory to the folder where the data file is. (Or else be prepared to use a full pathname to the file.) Note the quotation marks around the file name. You can use either of

```
getwd()
list.files()
```

to check that you are in the right place. Note the empty parentheses!

4 Reading data from a file

In this course much of the data you will use is in the form of a list of numbers, one per line, in a plain text file. To read the data from such a file into an array named `a`, use this:

```
a = as.matrix(read.table("Random32-2020.txt"))  # as.matrix is important!
```

(# is a comment delimiter.) Of course you should change `Random32-2020.txt` to the name of the file you really want.

It is usually a good idea to check the length to make sure you got the data:

```
length(a)
```

4.1 Data "frames"

For a more complicated dataset, you can create what R calls a data *frame*. See [9]. A data frame is like a table, where each column corresponds to a different variate. If you have data arranged as a table in a file you can read it into a frame. Typically (for this course) the first line of the file is **header line** containing the name of each variate. Columns are separated by whitespace. (In this class we use ASCII TAB characters.) So for the file SimplifiedEarthquakeCatalog2020.txt, which has the headers `DateSerial`, `Magnitude`, `Latitude`, `Longitude`, `Date`, `Time`, you can import the data with:

```
quakes = read.table("SimplifiedEarthquakeCatalog2020.txt", header=T)
length(quakes)
```

The `T` in `header=T` is just an abbreviation of `TRUE`. The `length` command should report the number of columns in the data frame.

You may then refer to the arrays for each variable using R's `$` notation:

```
d = quakes$DateSerial
m = quakes$Magnitude
plot(d, m, type = "l")
```

This will plot the magnitude vs. time for the earthquake data. (For the earthquake dataset, the DateSerial is the time of occurrence in fractional days since 00:00 hours on Jan 1, 1933, the earliest date in the SCEDC catalog.)

5 Basic descriptive statistics

The mean and standard deviation of the sample in the array `a` are calculated by the functions

```
mean(a)
sd(a)
```

The maximum, minimum, mean, and quartiles, but not the standard deviation, are reported by

```
summary(a)
```

Aside: As you may know, most software calculates a sample standard deviation by dividing by $n-1$ (instead of n), as in

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

To see what R (or any software) does, try this:

```
a = c( -1, -1, 1, 1 )
sd(a)
```

The command `c` combines its arguments into a vector. This particular data vector has sample mean $= 0$, so the sum of squared deviations from the mean is 4. Dividing this by $n-1=3$ gives $4/3$, and its square root is $2/\sqrt{3} \approx 1.154701$, which is what R reports.

6 Some simple plots

R's `plot` command is similar to Mathematica's `DiscretePlot` command in that it plots lists of points. They can be connected with line segments, or left as just circles centered at the points. You can choose the style of a plot by using the `type="l"` or `type="p"` options. The `l` type draws a smooth curve through the data, while the default `p` type draws circles centered at each point; `type="b"` does both.

R's `curve` command is similar to Mathematica's `Plot` command in that you supply an interval over which to graph a function, and it will choose the points to create a smooth curve.

6.1 Histograms and ecdfs

There are two special kids of plots that have their own commands. The first is a histogram. For a list `a` of numbers,

```
hist(a)
```

creates a histogram, and if you are using RStudio, displays it. By default it uses the Sturges method of choosing bins, and the vertical axis measures counts. The `breaks` option allows you to choose your own bins. The `freq=FALSE` option changes the vertical axis to using the *density* of the data in a bin instead of the actual count.

So to create a histogram with bins of size 0.01, showing the density, use this:

```
bins = seq(0.0, 1.0, 0.01)
hist(a, breaks = bins, freq = FALSE)
```

The `seq(a, b, d)` function creates a vector (list) that starts at `a`, ends at `b`, in increments of `d`. There are more options.

Another thing you can do is assign your histogram **object** to a variable, like this:

```
h = hist(a, breaks = bins, freq = FALSE)
```

Then you can retrieve information on it, or plot it, with commands such as:

```
h
summary(h)
plot(h)
```

The second special plot is the empirical cdf. The following code saves the `ecdf` object in a variable named `g`, so you can plot it, or save it later to a file. Try this:

```
g = ecdf(a)
summary( g )
plot(g)
```

6.2 Labeling plots and other fancy tricks

Be nice to your benevolent TAs and label your plots. Here's how.

Axes labels are set with `xlab` or `ylab`. The main title is given by `main`. Here is an example of the syntax. (Note that you may use more than one line to enter a command, but there are some rules about it.)

```
g = ecdf(a)
plot(g, xlab="x",
    ylab="fraction", main="Empirical CDF for 32-bit Coin Toss Data")
```

You can also set the **color** of the plot using the `col` option. Let's try this out:

```
g=ecdf(a)
plot(g, xlab="x", ylab="fraction",
  main="Empirical CDF for 32-bit Coin Toss Data",
  col="red" )
```

See Figure 1.

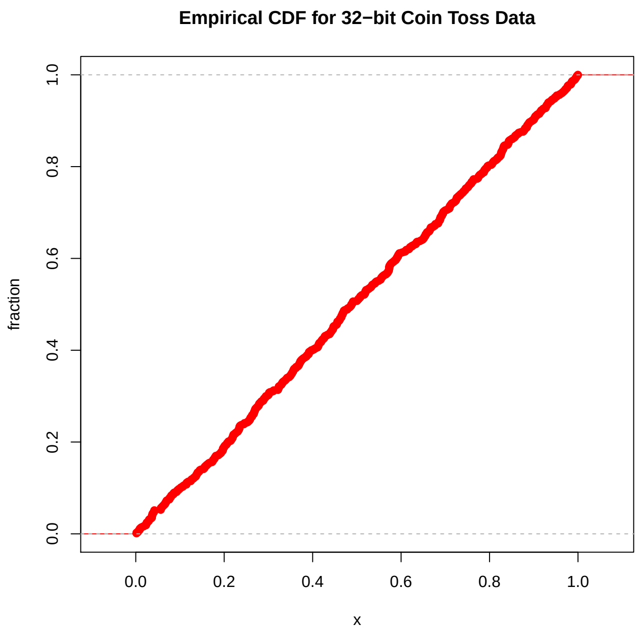

Figure 1: Color in an `ecdf` plot saved as a pdf file.

You can find more documentation at www.rdocumentation.org/packages/graphics/versions/3.6.2/topics/plot.

6.3 Overlays

For histograms and ecdfs we often wish to compare them to a known density or cdf. In Mathematica, the `Show[g1, g2]` command automagically combines the two graphics objects `g1` and `g2`. R is different. Let's overlay a plot of the uniform density over a histogram. See Figure 2.

```
s = seq(0, 1, .01)
hist(a, freq = FALSE)
points(s, dunif( s ), col="red", type="l", lwd=3)
```

You know what the `hist` and `seq` commands do. The `points` command *overlays* a `plot` of the points in the array `s` versus the values of the uniform density at these points. The array of values is created by `dunif(s)`. See Tables 1 and 2 for an explanation of the naming convention that gives us `dunif`. I've mentioned `col` and `type` above. The `lwd=3` option sets the linewidth to 3 times its default value. (I think R lines are too thin to be easily seen, but I'm old and in the way.)

You could also add a straight line with the `abline(a, b)` command. It draws a straight line with intercept `a` and slope `b`.

6.4 Saving plots to files

How did I get the plots into these notes? I saved each plot to a graphics file, and then used the LaTeX `graphicx` package's `\includegraphics` command. How do you save a plot to a file? You first have to create a graphics file *device*. To save a plot as a `pdf` file named HistogramSaveTest.pdf try this:

```
s = seq(0, 1, .025)

pdf("HistogramSaveTest.pdf")      # open the file device for writing

hist(a, breaks = s, freq = FALSE, xlab = "x", ylab = "density",
  main = "Histogram of 32-bit Coin Tossing Data")

points(s, dunif( s ), col="red", type="l", lwd=3)

dev.off()             # close the file.  This is crucial.
```

See Figure 2. **N.B.** The parentheses in `dev.off()` are crucial. Leaving them off generates a cryptic error message, and does not write the file. I found this out the hard way.

To save to a `png` file use `png("HistogramSaveTest.png")` for the first line. My R consultant deprecates the png format, saying that the pdf format gives a superior image. I have to agree with him.

6.5 Q-Q plots

A Q-Q plot plots the quantiles from a sample against the quantiles of another sample, or the quantiles of a distribution. R has a couple of commands for making Q-Q plots. If `y` is an array of data, the following will plot the quantiles of `y` (on the vertical axis) versus the quantiles of a standard normal random variable.

```
qqnorm(y)
```

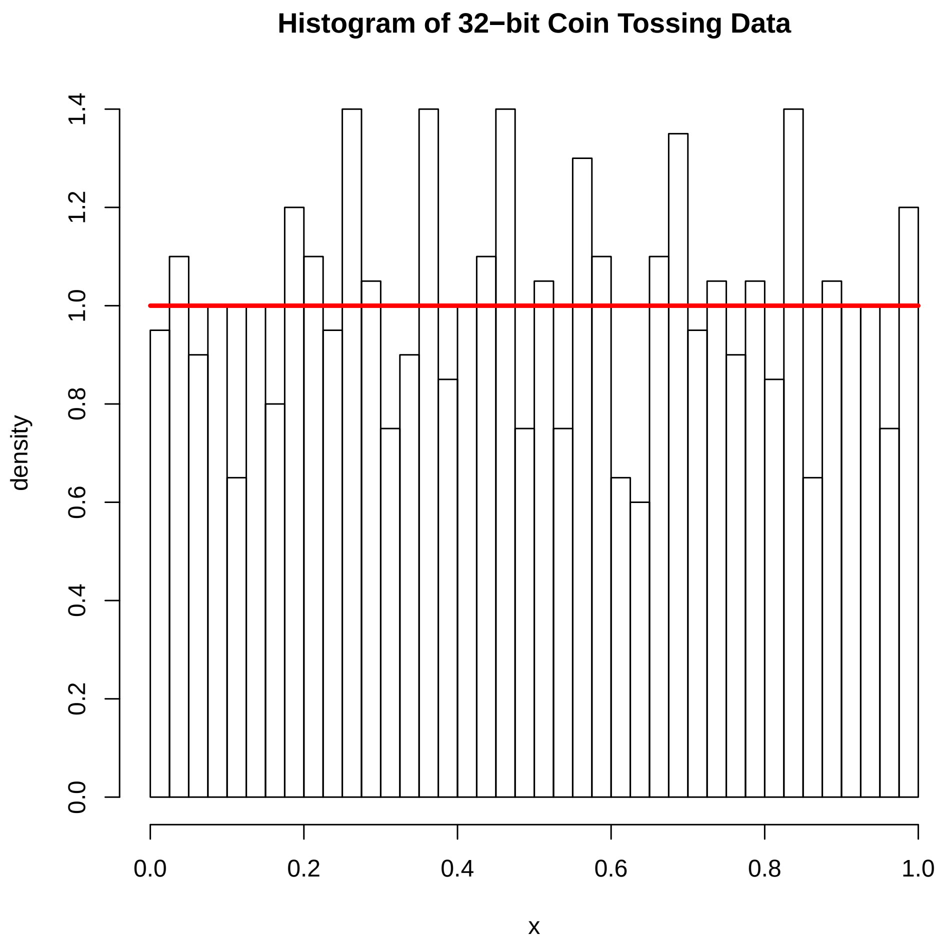

Figure 2: An R histogram with overlaid density, saved as a pdf file.

If `y` comes from a Normal(μ, σ^2) distribution, the points should lie on a line with slope σ and y-intercept μ. To overlay a line on the Q-Q plot use

```
qqline(y)
```

This will overlay a line that by default passes the through the first and third quartile points.

To create a Q-Q plot of a sample vs. the quantiles of a distribution other than the standard normal, you may specify one of R's named distributions (see Section 7).

To plot the quantiles of `y` against, say the Exponential(2) quantiles, you can use.

```
qqplot(qexp(ppoints(y), rate = 2), y,
       main = "Q-Q plot for Exponential, rate = 2")
qqline(y, distribution = function(p) qexp(p, rate = 2) )
```

Here `qexp` is the quantile function for the exponential family. The `ppoints` (used for *probability plotting*, scales the vector `y` to fit into $(0, 1)$, and `sort` sorts it. The specification of the distribution to the `qqline` command is rather awkward.

To visualize if two samples come from the same distribution, the `qqplot` command will make a Q-Q plot of the second sample (on the vertical axis) against the quantiles of the first (on the horizontal axis):

```
qqplot(x, y)
```

If `y` and `x` come from the same distribution, the points should lie on a line with slope 1. The `qqline` command does not work with two samples—it will only plot the first sample quartiles against the quantiles of a named distribution.

7 Named distributions

R has a consistent but not obvious convention for referring to probability densities, pmfs, and cdfs.

First you decide whether you want the cdf, the pmf/density, the quantile function, or a random sample for the distribution. Prefixes for these are listed in Table 1.

Second, you need to specify the distribution. Table 2 has a selection of named distributions. There are lots more.

Third, you need to know about parameters and options. These often have default values. Use the `help` function to find out.

For instance, to find the value of the cdf at the point `x` for a general normal, use

```
pnorm(x, mean = 2, sd = 3)
```

To get a sample of 100 independent draws from a standard normal distribution you can use any of these:

```
rnorm(100, mean = 0, sd = 1)
rnorm(100, 0, 1)
rnorm(100)
```

8 Built-in functions

In addition to the distribution-related functions, R has a number of built-in mathematical functions for numerical calculations. Table 3 has a useful selection; there are lots more.

function	prefix
cdf	`p`
pmf/density	`d`
quantile	`q`
random sample	`r`

Table 1: Prefixes for distributions.

distribution	R name	defaults
uniform(a, b)	`unif(x, min =` a`, max =` b`)`	`min` $= 0$, `max` $= 1$
	`unif(x,` a`,` b`)`	
normal(μ, σ)	`norm(x, mean =` μ`, sd =` σ`)`	`mean` $= 0$, `sd` $= 1$
	`norm(x,` μ`,` σ`)`	
binomial(n, p)	`binom(x, size =` n`, prob =` p`)`	
	`binom(x,` n`,` p`)`	
exponential(λ)	`exp(x, rate =` λ`)`	`rate` $= 1$
	`exp(x,` λ`)`	
chi-square(n)	`chisq(x, df =` n`)`	
	`chisq(x,` n`)`	
Student-t(n)	`t(x, df =` n`)`	
	`t(x,` n`)`	
(Snedecor) F(n, m)	`f(x, df1 =` n`, df2 =` m`)`	
	`f(x,` n`,` m`)`	

Table 2: R's names for some distributions, with named parameters and with parameter names omitted. Italicized or Greek parameters should be replaced by numerical values. Parameters with default values may be omitted.

9 Some operations on arrays

If `a` is an array of numbers, then adding a scalar to `a` adds it to each component, multiplying `a` by a scalar multiplies each component, and `a * a` squares each component of `a`. Here is a sample of input and output

```
> a = seq(1, 5)
> a
[1] 1 2 3 4 5
> a + 3
[1] 4 5 6 7 8
> 3 + a
[1] 4 5 6 7 8
> 2 * a
[1]  2  4  6  8 10
> a * 2
[1]  2  4  6  8 10
> a * a
[1]  1  4  9 16 25
```

You've already seen `mean` and `sd`. The `sum` function sums the components of `a`:

function	R name
natural log, $\ln x$	`log(x)`
exponential, $\exp x$ or e^x	`exp(x)`
square root, $\sqrt{x}$	`sqrt(x)`
x^a	`x ** a` or `x^a`
factorial, $n!$	`factorial(n)`
binomial coefficient, $\binom{n}{k}$	`choose(n, k)`
Gamma function, $\Gamma(x)$	`gamma(x)`
beta function, $B(r, s)$	`beta(r, s)`

Table 3: Some of R's built-in mathematical functions.

```
> sum(a)
[1] 15
```

Suppose you have a (sorted) list of times/dates at which earthquakes occurred, and you want to find the time intervals between them. This is what the `diff` command does. For instance,

```
> t = c(1, 2, 6, 11, 27)
> t
[1]  1  2  6 11 27
> diff(t)
[1]  1  4  5 16
```

The `sort` command sorts an array:

```
> t = c(1, 3, -2, 0, 5)
> t
[1]  1  3 -2  0  5
> sort(t)
[1] -2  0  1  3  5
```

10 Defining functions

To define a function of variables p, k, and n, say the binomial log-likelihood function,

$$f(p,k,n) = \log\left(\binom{n}{k} p^k (1-p)^{n-k}\right)$$

use

```
f <- function (p, k, n) log( choose(n, k) * p^k * (1-p)^(n-k) )
```

Note that if you intend to use this function with the `optimize` command, the *first* argument should be the maximand.

11 Graphing functions

To graph a function, the `curve` command assumes the argument of the function is named x. To plot it over an interval (a, b), use the option `xlim=c(a, b)`. Axes labels are set with `xlab` or `ylab`. The main title is given by `main`. Here is an example of the syntax. (Again, note that you may use more than one line to enter a command in R.)

```
curve( f(7, 4, x), xlim=c(0, 1), xlab="p",
  ylab="Log Likelihood",
  main="Binomial Log Likelihood Function")
```

See Figure 3.

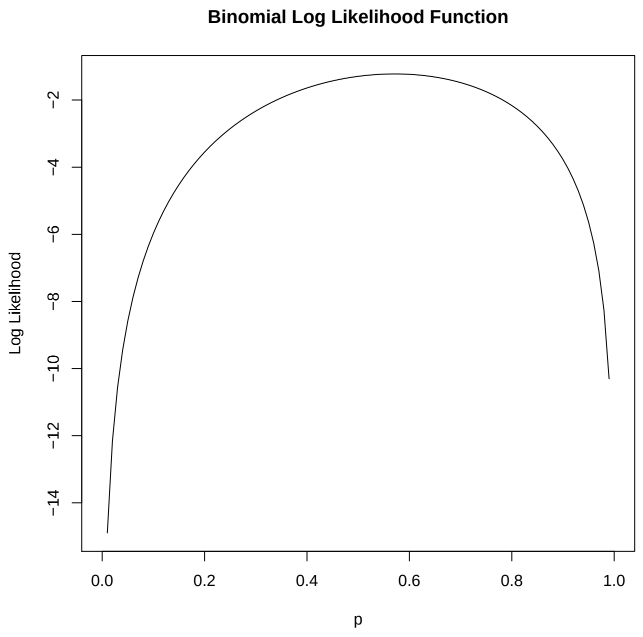

Figure 3: Output from the `curve` command saved as a pdf file.

12 Numerical one-dimensional optimization

To maximize a function f of one variable over the interval (a, b) use one of the following.

```
optimize( f, interval = c(a, b), maximum = TRUE )
optimize( f, c(a, b), maximum = TRUE )
```

The `optimize` command *minimizes* f if the `maximum=TRUE` option is omitted. Be aware that if f is ill-behaved, then this may not work, so examine your results carefully. Consider maximizing the logarithm of a likelihood function instead of the likelihood. It is typically better behaved numerically.

If f has more than one argument, the `optimize` command will maximize it with respect to its *first* argument. If additional arguments are needed, they should be supplied (in order) as below.

```
optimize( f, interval = c(a, b), arg2, arg3, ..., argn,  maximum = TRUE )
```

For instance, in assignment 5 you are asked to maximize (wrt p) the World Series likelihood, which is proportional to

$$\tilde{L}(p; N_0, N_1, N_2, N_3) = \prod_{k=0}^{3} \left[p^4(1-p)^k + p^k(1-p)^4 \right]^{N_k}.$$

As of the 2019 series, the values for $N_0, \dots, N_3$ are $(21, 26, 24, 40)$. Use the following R code.

```
g <- function(p, k, n) ( ( p^4 * (1-p)^k + p^k * (1-p)^4 )^n )

likelihood <- function(p, n0, n1, n2, n3) ( g(p, 0, n0) * g(p, 1, n1) *
  g(p, 2, n2) * g(p, 3, n3) )

optimize( likelihood, c(.5,1), 21, 26, 24, 40, maximum=TRUE )
```

[My R consultant tells me that there are more R-like ways to define the likelihood function, using `mapply`, but he doesn't recommend trying to teach them in a stats class.]

13 Off-the-shelf tests

The `help` function can be used to find out how to perform these tests. I will soon add more details.

- One-sample t-test: For a one-sample test, if your sample is in the array `a`, the command

  ```
  t.test(a, mu=m, alternative="two.sided")
  ```

  creates a t-test object that returns a detailed report on the two-sided test of the hypothesis $\mu = m$ including a confidence interval for μ. If you omit, `mu=m`, the test uses the default `mu=0`. If you do not want a two-sided test (the default), use either `alternative="less"` or `alternative="greater"`.

  Here is a sample:

  ```
  set.seed(314159)
  a = rnorm(100)
  t.test(a)
  ```

  produces the following output:

  ```
  One Sample t-test

  data:  a
  t = -0.57906, df = 99, p-value = 0.5639
  alternative hypothesis: true mean is not equal to 0
  95 percent confidence interval:
   -0.2458422  0.1347676
  sample estimates:
    mean of x
  -0.05553729
  ```

The `set.seed(314159)` command sets the seed for the random number generator. You don't need to do this, but I did it so you can replicate my results. (There is no special reason to set the value to 314159, I just like it.)

By default, the command reports a 95% confidence interval for the mean. You can change this with the `conf.level` option. There are other options and defaults as well. Check the documentation.

You can assign the *t*-test object to a variable:

```
tobj = t.test(a)
```

You can then retrieve specific parts of the output by referring to the components of the object. Here's an example:

```
ci = tobj$conf.int
```

retrieves the confidence interval's endpoints:

```
> ci[1]
[1] -0.2458422
> ci[2]
[1] 0.1347676
```

Or, you could skip the intermediate assignments and write `t.test(a)$conf.int[1]`, but this is cumbersome if you specify a lot options.

See the documentation for `t.test` under the heading Value for more components of the *t*-test object.

- Paired two-sample *t*-test: Supply two data vectors, `x, y`, of the same length and use the `paired=TRUE` option.

```
set.seed(314159)
x=rnorm(12)
y=rnorm(12)
t.test(x ,y, paired=TRUE)
```

Here's some sample output:

```
Paired t-test

data:  x and y
t = -0.38833, df = 11, p-value = 0.7052
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.2923235  0.9046922
sample estimates:
mean of the differences
             -0.1938156
```

See the documentation for additional details.

- Unpaired two-sample *t*-test: For a two-sample model, with potentially different variances, use `t.test(data1, data2)`. See Dytham [4, pp. 103–110].

  ```
  a2 = rnorm(50)
  t.test(a, a2)
  ```

  produces the following output:

  ```
  Welch Two Sample t-test

  data:  a and a2
  t = -0.43415, df = 95.517, p-value = 0.6652
  alternative hypothesis: true difference in means is not equal to 0
  95 percent confidence interval:
   -0.4101209  0.2629252
  sample estimates:
    mean of x   mean of y
  -0.05553729  0.01806055
  ```

- Chi-squared test: `chisq.test`

- Fisher's exact test: `fisher.test`

- Kolmogorov–Smirnov test: `ks.test` (See the documentation or Dytham [4, pp. 86–89].) To test the null hypothesis that the data vector `y` is drawn from a given distribution, use

  ```
  ks.test(y, "YourDistribution", YourParameters)
  ```

  where, of course, `YourDistribution` is replaced by the name of a continuous cdf, such as `pnorm` or `pexp` for normal and exponential; and `YourParameters` is replaced by the parameters of the named distribution. For example,

  ```
  ks.test(y, "pnorm", mu = 0, sd = 1)
  ks.test(y, "pexp", rate = 3)
  ```

  The output includes the value of the test statistic D and its corresponding p-value. Here is a sample

  ```
  > ks.test(times, "pexp", rate=3)

  One-sample Kolmogorov-Smirnov test

  data:  times
  D = 0.096407, p-value = 0.5984
  alternative hypothesis: two-sided
  ```

  Using the `exact=TRUE` option reports a p-value that is in better agreement with the report from Mathematica.

- Mann–Whitney and Wilcoxon test: `wilcox.test`

- Kruskal–Wallis rank sum test: `kruskal.test`

14 Linear regression

R is an object-oriented language, and regression is carried out via an `lm` (for linear model) object. The distinctive feature of an `lm` object is its `formula` property. A typical `formula` specification looks like `formula = y  x`, which means to regress y on x and a constant. (Constants are supplied by default. To get rid of the constant you could say `formula = y  -1 + x`, but you almost never want to get rid of the constant.) The `lm` object also requires a `data` specification. The are other properties that could be set, but the default values are adequate for most of our assignments.

First you read in the data, as usual. You may need to supply a full path or use `setwd` to set the working directory to where you data files are kept. Let's call the data `a`, as in Anscombe.

```
a = read.table("Anscombe1", header=TRUE)
```

You can use `sapply` to calculate the means and standard deviations of all the variates in the dataset. (`sapply` applies a function to a list and returns a vector.)

```
sapply(a, mean)
sapply(a, sd)
```

Now you can create a linear model object. Let's call it `model`. Because we read in the data with headers, R knows what X and Y are.

```
model = lm(formula=y~x, data=a)
```

Much useful output is obtained by simply asking for a summary:

```
summary(model)
```

This should have provided you with estimated coefficients, t-statistics, the R^2, the F-statistic for the regression, and some other stuff.

There are other useful things in the object. I suggest you try:

```
coef(model)
residuals(model)
```

The sum of squared residuals is not automatically reported, but it's not hard to calculate.

```
e = residuals(model)
ssr = sum(e*e)
ssr
```

You can compute the s^2 statistic $\dfrac{e'e}{T-K}$, and take its square root:

```
T = length(a$x)
K = length(coef(model))
sqrt(ssr/(T-K))
```

How do you create a scatter plot with the regression line? Here's one way:

```
plot(a)
abline(model)
```

And here's a normal Q-Q plot of the residuals, with a Kolmogorov–Smirnov test for normality thrown in for good measure.

```
e = residuals(model)
qqnorm(e)
abline(a=0, b=1)
ks.test(e, pnorm)
```

Some useful resources

[1] Adler, J. 2012. *R in a nutshell*, 2d. ed. Sebastopol, California: O'Reilly Media.

[2] Chang, W. 2013. *R graphics cookbook.* Sebastopol, California: O'Reilly Media.

[3] Dalgaard, P. 2008. *Introductory statistics with R*, 2d. ed. Statistics and Computing. New York: Springer Science+Business Media.

[4] Dytham, C. 2011. *Choosing and using statistics: A biologist's guide*, 3d. ed. Wiley–Blackwell.

[5] Ekstrøm, C. T. 2011. *R primer.* Boca Raton, Florida: Chapman & Hall/CRC Press.

[6] Hothorn, T. and B. S. Everitt. 2014. *A handbook of statistical analysis using R*, third ed. Boca Raton, Florida: CRC Press.

[7] Härdle, W. K., O. Okhrin, and Y. Okhrin. 2017. *Basic elements of computatinal statisitcs.* Statistics and Computing. Cham, Switzerland: Springer Nature.

[8] Teetor, P. 2011. *R cookbook.* Sebastopol, California: O'Reilly Media.

[9] Verzani, J. 2014. *Using R for introductory statistics.* The R Series. Boca Raton, Florida: CRC Press.

[10] Wickham, H. 2009. *ggplot2: Elegant graphics for data analysis.* Dordrecht: Springer.

[11] Zuur, A. F., E. N. Ieno, and E. H. W. G. Meesters. 2009. *A beginner's guide to R.* New York: Springer Science+Business Media.

Index of R commands and options

*, multiplication, 9, 15
**, exponentiation, 10
<-, assignment operator, 2
?, help function, 1
$, 3, 13
^, exponentiation, 10

abline, 6, 15, 16
alternative, 12
as.matrix, 3

beta(r, s), 10
breaks, 4

c, 4
chisq.test, 14
choose(n, k), 10
coeff, 15
col, 5, 6
color, col, 5
curve, 4, 10

data, 15
dev.off(), 6
diff, 10
dunif, 6
dunif, uniform density, 6

ecdf, 4, 5
exact=TRUE, 14
exp(x), 10

factorial(n), 10
fisher.test, 14
formula, 15
freq=FALSE, 4
function, 10, 12

gamma(x), 10

header=T, 3
help, 2, 8, 12
hist, 4, 6

kruskal.test, 14
ks.test, 14, 16

length, 3, 15
linewidth, lwd, 6
lm, 15
log, 10
log(x), 10
lwd, 6

main, 5, 11
mapply, 12
maximum=TRUE, 11
mean, 3, 9

optimize, 10, 11

paired=TRUE, 13
pdf, 6
pexp, 14
plot, 3–6, 15
png, 6
pnorm, 14
points, 6
ppoints, 8

q(), 2
qexp, 8
qqline, 8
qqnorm, 8, 16
qqplot, 8

read.table, 3, 15
residuals, 15
rnorm, normal random sample, 13

sapply, 15
sd, 3, 4, 9
seq, 4, 6
set.seed, 13
setwd, 15
sort, 8, 10
sqrt(x), 10
sum, 9
summary, 4, 5, 15

t.test, 12, 13
type, 6

wilcox.test, 14

xlab, 5, 10
xlim, 10

ylab, 5, 11