APPENDIX A

GETTING STARTED WITH R

There are many excellent R primers and tutorials on the web. To learn more about R, the best starting place is the homepage of the R Project for Statistical Computing at http://www.r-project.org/. Go there to download the software. The site contains links to books, manuals, demonstrations, and other resources. For this introduction, we assume that you have R up and running on your computer.

1. **R as a calculator**

   When you bring up R the first window you see is the R console. You can type directly on the console, using R like a calculator.

```
> 1+1
[1] 2
> 2-2
[1] 0
> 3*3
[1] 9
> 4/4
[1] 1
> 5^5
[1] 3125
```

Introduction to Stochastic Processes with R, First Edition. Robert P. Dobrow.

R has many built-in math functions. Their names are usually intuitive. R is case sensitive. All commands are in lowercase letters. The # symbol is for comments. Anything appearing after # is ignored by R.

```
> pi
[1] 3.141593
> cos(pi)
[1] -1
> exp(1)   # e^1
[1] 2.718282
> abs(-6)  # |-6|
[1] 6
> factorial(4)
[1] 24
> sqrt(9)
[1] 3
```

Exercises:

1.1. How many orderings are there of a standard 52-card deck?

1.2. An isosceles right triangle has two legs of length 5. Find the length of the hypotenuse.

1.3. Find the tangent of a 60-degree angle.

1.4. The *hyperbolic sine* is the function

$$h(x) = \frac{e^x - e^{-x}}{2}.$$

Evaluate the hyperbolic sine at $x = 1$. The R command for hyperbolic sine is `sinh(x)`. Verify that `sinh(1)` gives the same results.

2. **Navigating the keyboard**

Each command you type in R is stored in history. The up arrow (↑) key scrolls back along this history; the down arrow (↓) scrolls forward. These keystrokes are enormously helpful for navigating your R session.

Suppose you want to calculate log 8 and you type

```
> Log(8)
Error: could not find function "Log"
```

You got an error because R commands start with lower case letters. Rather than retype the command, hit the up arrow key and fix the previously entered entry.

```
> log(8)
[1] 2.079442
```

You can use the <ESC> (escape) key to interrupt a running calculation or output. If R is expecting more input it will preface a line at the console with the + symbol. You can use escape to start anew. In the following, we forget the end parentheses. Repeatedly hitting the return key generated the + symbol as R is expecting the right parentheses to finish the command. By finally typing <ESC>, the command is voided.

```
> exp(1
+
+
+
>
```

3. **Asking for help**

Prefacing any R command with ? will produce a help file in another window.

```
> ?log
```

Reading the help file for the log function you learn that the function has the form log(x, base = exp(1)). The base argument defaults to exp(1) = e. That is, log(x) returns the natural logarithm of x. For logarithms in other bases give a second argument to log.

```
> log(8,2)
[1] 3
> log(100,10)
[1] 2
```

Exercises:

3.1. Read the help file for the factorial command to find the R command for computing the binomial coefficient $\binom{n}{k}$, also called "n choose k." Now compute the number of 5-card Poker hands in a standard 52-card deck $\binom{52}{5}$.

3.2. Use the up arrow key, and replace the 5 in the previous exercise with 13 to find the number of 13-card bridge hands in a standard 52-card deck.

4. **Vectors**

The real power of R lies in its ability to work with lists of numbers, called vectors. If a and b are integers, the command a:b creates a vector of integers from a to b.

```
> 1:10
[1]  1  2  3  4  5  6  7  8  9 10
```

```
> -3:5
[1] -3 -2 -1  0  1  2  3  4  5
```

For more control generating vectors, use the sequence command `seq`.

```
> seq(1,10)
[1]  1  2  3  4  5  6  7  8  9 10
> seq(1,20,2)
[1]  1  3  5  7  9 11 13 15 17 19
> seq(20,1,-4)
[1] 20 16 12  8  4
```

Create and manipulate vectors using the `c` concatenate command.

```
> c(2,3,5,7,11)
[1]  2  3  5  7 11
```

Assign a vector to a variable with the assignment operator `<-`. Here, the first five prime numbers are assigned to a variable called `primes`. The `primes` vector is then lengthened by concatenating three more primes to the original vector.

```
> primes <- c(2,3,5,7,11)
> primes
[1]  2  3  5  7 11
> primes <- c(primes,13,17,19)
> primes
[1]  2  3  5  7 11 13 17 19
```

Elements of vectors are indexed with brackets [ ]. Bracket arguments can be single integers or vectors.

```
> primes[4]
[1] 7
> primes[1:4]
[1] 2 3 5 7
> primes[c(1,4,5,10)]
[1]  2  7 11 NA
```

For the last command we asked for the first, fourth, fifth, and tenth element of `primes`. There is no tenth element so `R` returns an `NA`.

One often wants to find the elements of a vector that satisfy some property, such as those `primes` that are less than 10. This is done with the `which` command, which returns the indices of the vector that satisfy the given property.

```
> primes
[1]  2  3  5  7 11 13 17 19
> which(primes < 10)
[1] 1 2 3 4
> index <- which(primes < 10)
> primes[index]
[1] 2 3 5 7
```

Vectors can consist of numbers, characters, and even strings of characters.

```
> y <- c("Probability", "is", "very", "very",
  "cool")
> y
[1] "Probability" "is" "very" "very" "cool"
> y[c(1,2,5)]
[1] "Probability" "is" "cool"
```

When performing mathematical operations on vectors, the entire vector is treated as a single object.

```
> dog <- seq(0,30,4)
> dog
[1]  0  4  8 12 16 20 24 28
> dog+1
[1]  1  5  9 13 17 21 25 29
> dog*3
[1]  0 12 24 36 48 60 72 84
> 1/dog
[1]  Inf 0.25000 0.12500 0.08333 0.06250 0.05000
[7] 0.041667 0.035714
> cat <- dog+1
> cat
[1]  1  5  9 13 17 21 25 29
> dog*cat
[1]   0  20  72 156 272 420 600 812
```

Notice that when a single number is added to a vector, that number gets added to each element of the vector. But when two vectors of the same length are added together, then corresponding elements are added. This applies to most binary operations.

Many mathematical functions can take vector arguments, returning vectors as output.

```
> factorial(1:7)
[1]    1    2    6   24  120  720 5040
```

```
> sqrt(seq(0,900,100))  # √0, √100, √200, ..., √900
 [1]  0.000 10.000 14.142 17.321 20.000 22.361
 [7] 24.495 26.458 28.284 30.000
```

Here are some common, and intuitive, commands for working with vectors.

```
> x <- c(67.6, 68.7, 66.3, 66.2, 65.5, 70.2, 71.1)
> sum(x)
[1] 475.6
> mean(x)
[1] 67.943
> length(x)
[1] 7
> sort(x)
[1] 65.5 66.2 66.3 67.6 68.7 70.2 71.1
> sort(x,decreasing=T)
[1] 71.1 70.2 68.7 67.6 66.3 66.2 65.5
```

Exercises: Use vector operations:

4.1. Compute the squares of the first ten integers.

4.2. Compute the powers of 2 from 2^1 to 2^{20}.

4.3. For $n = 6$, use the `choose` command to compute the binomial coefficients $\binom{n}{k}$, for $k = 0, \ldots, n$.

4.4. Let $\boldsymbol{x}$ be an n-element vector defined by $x_k = 2\pi k/n$, for $k = 1, \ldots, n$. Find $(\cos x_1, \ldots, \cos x_n)$ for $n = 5$. Repeat for $n = 13$, using the up arrow key rather than retyping the full command.

4.5. Compute the sum of the cubes of the first 100 integers.

4.6. Among the cubes of the first 100 integers, how many are greater than 10,000? Use `which` and `length`.

5. **Generating random numbers**

The `sample` command generates a random sample of a given size from the elements of a vector. The syntax is `sample(vec, size, replace=, prob=)`. Samples can be taken with or without replacement (the default is without). A probability distribution on the elements of `vec` can be specified. The default is that all elements of `vec` are equally likely.

```
> sample(1:10,1)
[1] 8
> sample(1:4,4)
[1] 3 1 4 2
```

```
> sample(c(-8,0,1,4,60),6,replace=T)
[1] 60 -8  1  1  4 60
```

To simulate ten rolls of a six-sided die, type

```
> sample(1:6,10,replace=T)
 [1] 6 3 6 1 3 6 5 3 2 1
```

According to the Red Cross, the distribution of blood types in the United States is O: 44%, A: 42%, B: 10%, and AB: 4%. The following simulates a random sample of 10 people's blood types.

```
> sample(c("O","A","B","AB"),8,replace=T,
   prob=c(0.44,0.42,0.10,0.04))
 [1] "A"  "O"  "O"  "O"  "A"  "A"  "A"  "AB"
```

In a random sample of 50,000 people, how many have blood type B? Use the `which` command and the logical connective `==`.

```
> samp <- sample(c("O","A","B","AB"),50000,
    replace=T,prob=c(0.44,0.42,0.10,0.04))
> index <- which(samp=="B")
> length(index)
[1] 5042
```

Other logical connectives are given in Table A.1.

TABLE A.1 Logical Connectives

==	equal to	&	and
! =	not equal to	|	or
>	greater than	!	not
>=	greater than or equal to		
<	less than		
<=	less than or equal to		

In the sample of 50,000 people, how many are either type B or type AB?

```
> length(which(samp=="AB" | samp=="B"))
[1] 7085
```

What proportion of people in the sample are not type O?

```
> length(which(samp != "O"))/50000
[1] 0.56388
```

Exercises:

5.1. Consider a probability distribution on $\{0, 2, 5, 9\}$ with respective probabilities $0.1, 0.2, 0.3, 0.4$. Generate a random sample of size five.

5.2. For the above-mentioned distribution, generate a random sample of size one million and determine the proportion of sample values which are equal to 9.

5.3. Represent the cards of a standard 52-card deck with the numbers 1 to 52. Show how to generate a random five-card Poker hand from a standard 52-card deck. Let the numbers 1, 2, 3, and 4 represent the four aces in the deck. Write a command that will count the number of aces in a random Poker hand. Use `which` and `length`.

5.4. Generate 100,000 integers uniformly distributed between 1 and 20 and count the proportion of samples between 3 and 7.

6. **Probability distributions**

There are four commands for working with probability distributions in R . The commands take the root name of the probability distribution (see Table A.2) and prefix the root with `d`, `p`, `q`, or `r`. These give, respectively, continuous density or discrete probability mass function (`d`), cumulative distribution function (`p`), quantile (`q`), and random variable (`r`).

TABLE A.2 Probability Distributions in R

Distribution	Root	Distribution	Root
beta	beta	log-normal	lnorm
binomial	binom	multinomial	multinom
Cauchy	cauchy	negative binomial	nbinom
chi-squared	chisq	normal	norm
exponential	exp	Poisson	pois
F	*f*	Student's *t*	*t*
gamma	gamma	uniform	unif
geometric	geom	Weibull	weibull
hypergeometric	hyper		

Generate six random numbers uniformly distributed on $(0, 1)$.

```
> runif(6,0,1)
[1] 0.06300 0.44851 0.70231 0.20649 0.14377 0.74398
```

Generate 9 normally distributed variables with mean $\mu = 69$ and standard deviation $\sigma = 2.5$.

```
> rnorm(9,69,2.5)
 [1] 69.548 69.931 68.923 71.153 71.779 68.975
 [7] 67.429 65.621 70.148
```

Find $P(X \leq 2.5)$, where X has an exponential distribution with parameter $\lambda = 1$.

```
> pexp(2.5,1)
[1] 0.91792
```

Suppose SAT scores are normally distributed with mean $\mu = 500$ and standard deviation $\sigma = 100$. Find the 95th quantile of the distribution of SAT scores.

```
> qnorm(0.95,500,100)
[1] 664.49
```

Find $P(X = 10)$, where X has a binomial distribution with parameters $n = 20$ and $p = 0.5$.

```
> dbinom(10,20,0.5)
[1] 0.1762
```

7. **Plots and graphs**

To graph functions and plot data, use `plot`. This workhorse command has enormous versatility. Here is a simple plot comparing two vectors.

```
> radius <- 1:20
> area <- pi*radius^2
> plot(radius,area, main="Area as
function of radius")
```

To graph smooth functions, you can also use the `curve` command. See Figures A.1 and A.2 . The syntax is `curve(expr,from=,to=)`, where `expr` is an expression that is a function of x.

```
> curve(pi*x^2,1,20,xlab="radius",ylab="area",
+ main="Area as function of radius")
```

For displaying data, a histogram is often used, obtained with the command `hist(vec)`. The default is a frequency histogram of counts. A density histogram has bars whose areas sum to one and is obtained using the `hist` command with parameter `freq=F`. A continuous density curve can be overlaid on the histogram by including the parameter `add=T`.

For example, male adult heights in the United States are normally distributed with mean 69 inches and standard deviation 2 inches. Here, we simulate 1,000

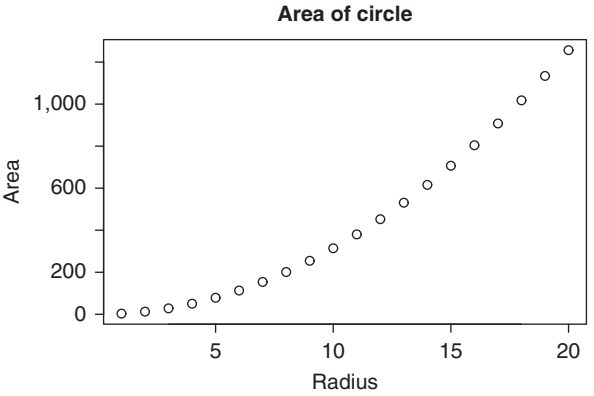

Figure A.1

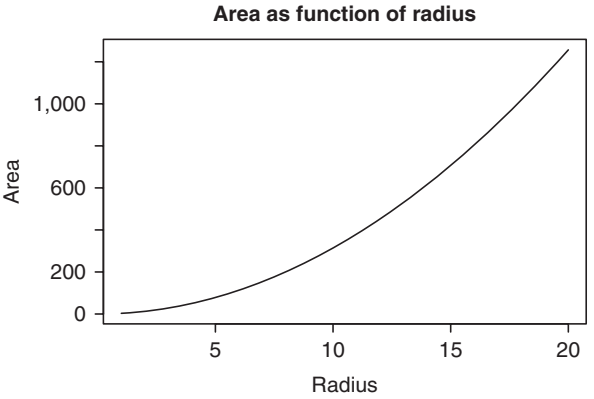

Figure A.2

observations from such a distribution and plot the resulting data. The histogram is first graphed with counts, then with relative frequencies, and then overlaid with a normal density curve. See Figure A.3.

```
> heights <- rnorm(1000,69,2.0)
> hist(heights,main="Distribution of heights")
> hist(heights,main="Distribution of heights",
  freq=F)
> hist(heights,main="Distribution of heights",
  freq=F)
> curve(dnorm(x,69,2),60,80,add=T)
```

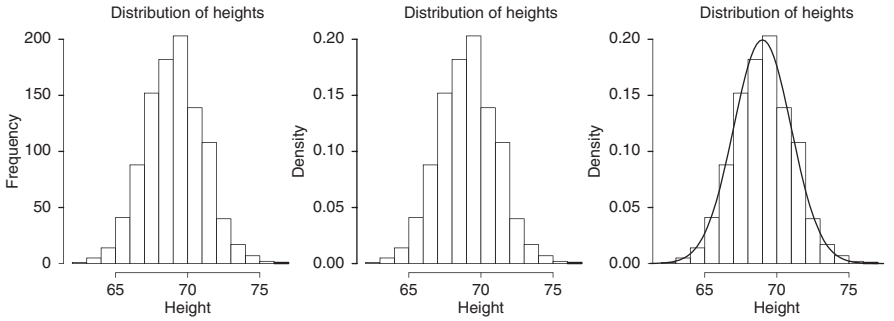

Figure A.3

8. **Script files**

   When you have many R commands to manage, it is useful to keep your work in a script file. A script is a plain text file that contains R commands. The file can be saved and edited and you can execute the entire file or portions of it as you please. Many of the R examples in this book are contained in script files that you can download, execute, and modify.

   Script files can be written in your favorite text editor. They can also be managed directly in your R environment.

   In the Mac environment, click on **File: New Document** to bring up a new window for typing your R commands. To execute a portion of your code, highlight the text that you want to execute, and then hit **Command-Return**. The highlighted code will appear in the R console and be executed.

   In the PC environment, click on **File: New Script** to bring up a new window for a script file. To execute a portion of your code, highlight the text. Under the task bar, press the **Run line or selection** button (third from the left) to execute your code.

   Open the R script file **scriptsample.R**. The file contains two blocks of code. The first block contains commands for plotting the area of a circle as a function of radius for radii from 1 to 20. Highlight the three lines of R code in the first part of the file. Execute the code and you should see a plot similar to that in Figure A.1.

```
# scriptsample.R
### Area of circle
radius <- 1:20
area <- pi*radius^2
plot(radius,area, main="Area as function of radius")
```

```
### Coin flips
n <- 1000 # Number of coin flips
coinflips <- sample(0:1,n,replace=TRUE)
heads <- cumsum(coinflips)
prop <- heads/(1:n)
plot(1:n,prop,type="l",xlab="Number of coins",
  ylab="Running average",
  main="Running proportion of heads in 1000
  coin flips")
abline(h=0.5)
```

The second block of code simulates flipping 1,000 coins, with 1 representing heads and 0 representing tails. The running proportion of heads is plotted from 1 to 1,000. The vector `coinflips` contains the outcome of 1,000 flips. The `cumsum` command generates a cumulative sum and stores the resulting vector in `heads`. The kth element of `heads` gives the number of heads in the first k coin flips. The proportion of heads is then computed and stored in `prop`. Finally, the running proportions are plotted as in Figure A.4. We see that the proportion of heads appears to converge to 1/2, an illustration of the law of large numbers.

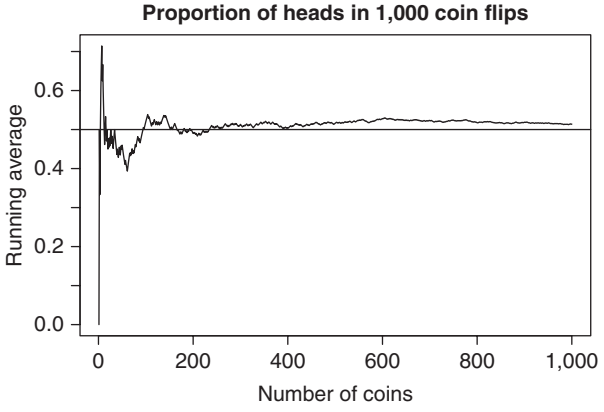

Figure A.4

Exercises:

8.1. Modify the **scriptsample.R** file to compute the volume of a sphere. Plot the volume as a function of radius for radii from 1 to 100.

8.2. Modify the **scriptsample.R** file so that the coin flips are biased and the probability of heads is 0.51. What happens to the resulting plot?

8.3. Given a list of numbers $x_1, \ldots, x_n$, the *sample standard deviation* is

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

where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the average of the x_i, obtained in R with the `mean` command. Write a command to compute the sample standard deviation for the integers from 1 to n. Use a script file. Compute the standard deviation for $n = 2, 10, 10000$. The R command for the standard deviation is `sd(x)`, where `x` is a vector. Compare the results of your command with that of the R command.

9. **Functions**

You can create your own functions in R . The syntax is

```
> name <- function(arg1, arg2, . . . ) expression
```

A function can have one, several, or no arguments. Here is a function to compute the area of a circle of given radius.

```
> area <- function(radius) pi*radius^2
```

The name of the function is `area`. It is a function of one variable.

```
> area(1)
[1] 3.141593
> area(7.5)
[1] 176.7146
```

The function `cone` computes the volume of a cone of height h and circular base of radius r.

```
> cone <- function(r,h) (1/3)*pi*r^2*h
> cone(1,1)
[1] 1.047198
> cone(3,10)
[1] 94.24778
```

If the function definition contains more than one line of code, enclose the code in curly braces {}. The function `allsums` takes a vector `vec` as its input and outputs the sum, sum of squares, and sum of cubes of the elements of `vec`.

```
> allsums <- function(vec) {
 s1 <- sum(vec)
 s2 <- sum(vec^2)
 s3 <- sum(vec^3)
 c(s1,s2,s3) }
> allsums(1:5)
[1]  15  55 225
> allsums(-5:5)
[1]   0 110   0
```

Exercises:

9.1. Write a function that takes as input the lengths of two sides of a right triangle and returns the length of the hypotenuse.

9.2. A tetrahedron die is a four-sided die with labels $\{1, 2, 3, 4\}$. Write a function with argument n that rolls n tetrahedron dice and computes their average value. Implement your function for $n = 1$, 1000, and 1,000,000.

10. **Other useful commands**

```
> table(c(0,1,1,1,1,1,2,2,6,6,6,6,6,6,6))
   # create a frequency table
0 1 2 6
1 5 2 7

> min(0:20)    # minimum element
[1] 0

> max(0:20)    # maximum element
[1] 20

> round(pi,3)  # rounds to a given number of
  decimal places
[1] 3.142
```

The `replicate` command is powerful and versatile. The syntax is `replicate(n,expr)`. The expression `expr` is repeated n times and output as a vector.

```
> replicate(6,2)
 [1]  2 2 2 2 2 2
```

Choose 1,000 numbers uniformly distributed between 0 and 1 and find their mean. Then repeat the experiment five times.

```
> replicate(5,mean(runif(1000,0,1)))
```

```
[1] 0.51168 0.50591 0.48371 0.49162 0.50879
```

A Poisson distribution with parameter λ has mean $\mu = \lambda$ and standard deviation $\sigma = \sqrt{\lambda}$. To illustrate the central limit theorem we generate 80 observations $X_1, \ldots, X_{80}$ from a Poisson distribution with parameter $\lambda = 4$ and compute the normalized sum

$$\frac{(X_1 + \cdots + X_n)/n - \mu}{\sigma/\sqrt{n}} = \frac{(X_1 + \cdots + X_{80})/80 - 4}{2/\sqrt{80}}.$$

The simulation is repeated 100,000 times and the resulting data are graphed as a histogram together with a standard normal density curve. See Figure A.5.

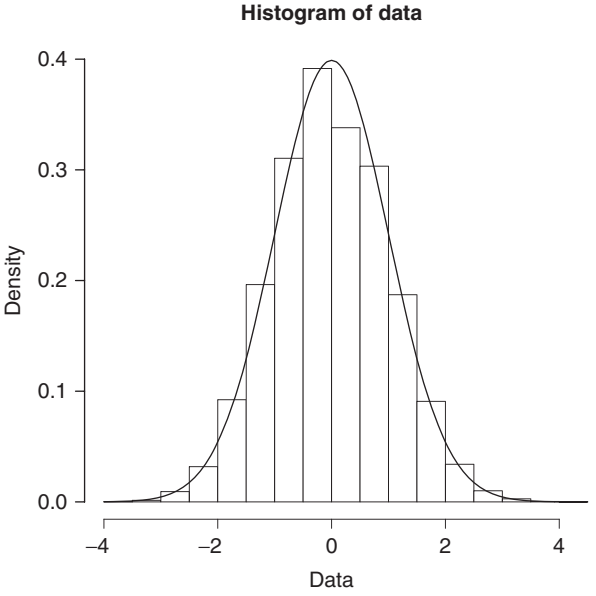

Figure A.5

```
> normsum <- function() (mean(rpois(80,4))-4)/
  (2/sqrt(80))
> normsum()
[1] -0.50312
> normsum()
[1] 1.062132
> data <- replicate(100000,normsum())
> hist(data,freq=F)
> curve(dnorm(x),-4,4,add=T)
```

11. **Working with matrices**

The `matrix` command is used to create matrices from vectors. Specify the number of rows or columns. The default is to fill the matrix by columns. To fill the entries of a matrix from left to right, top to bottom, add the parameter `byrow=T`.

```
> matrix(1:9,nrow=3)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> matrix(1:9,nrow=3,byrow=T)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

If x is a scalar, then $x + A$ results in adding x to each component of A. If A and B are matrices, then $A + B$ adds entries componentwise. The same is true for all binary operators.

```
> A <- matrix(c(1,3,0,-4),ncol=2,byrow=T)
> B <- matrix(1:4,ncol=2,byrow=T)
> A
     [,1] [,2]
[1,]    1    3
[2,]    0   -4
> B
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> 2+A
     [,1] [,2]
[1,]    3    5
[2,]    2   -2
> 3*B
     [,1] [,2]
[1,]    3    6
[2,]    9   12
> A+B
     [,1] [,2]
[1,]    2    5
[2,]    3    0
```

```
> A-B
     [,1] [,2]
[1,]    0    1
[2,]   -3   -8
> 2^B
     [,1] [,2]
[1,]    2    4
[2,]    8   16
> A/B
     [,1] [,2]
[1,]    1  1.5
[2,]    0 -1.0
```

For matrix multiplication, or matrix–vector multiplication use the matrix multiplication operator `%*%`.

```
> A %*% A
     [,1] [,2]
[1,]    1   -9
[2,]    0   16
> x <- c(2,1)
> A %*% x
     [,1]
[1,]    5
[2,]   -4
> y <- matrix(c(1,3),nrow=2)
y %*% x
     [,1] [,2]
[1,]    2    1
[2,]    6    3
```

Surprisingly, R does not have a primitive or built-in command for taking matrix powers. Here is a function that will take integer powers of a matrix.

```
matrixpower <- function(mat,k) {
  out <- mat
  for (i in 2:k) {
    out <- out %*% mat
    }
 out
    }

> mat <-matrix(c(0.1,0.9,0.6,0.4),nrow=2,byrow=T)
> mat
```

```
     [,1] [,2]
[1,]  0.1  0.9
[2,]  0.6  0.4
> matrixpower(mat,2)
     [,1] [,2]
[1,] 0.55 0.45
[2,] 0.30 0.70

> matrixpower(mat,10)
          [,1]      [,2]
[1,] 0.4005859 0.5994141
[2,] 0.3996094 0.6003906
> matrixpower(mat,50)
     [,1] [,2]
[1,]  0.4  0.6
[2,]  0.4  0.6
```

For $\boldsymbol{A}^T$, the transpose of $\boldsymbol{A}$, type `t(A)`.

```
> A <- matrix(c(3,1,-2,0,0,4),nrow=2,byrow=T)
> A
     [,1] [,2] [,3]
[1,]    3    1   -2
[2,]    0    0    4
> t(A)
     [,1] [,2]
[1,]    3    0
[2,]    1    0
[3,]   -2    4
> A %*% t(A)
     [,1] [,2]
[1,]   14   -8
[2,]   -8   16
```

To solve the linear system $\boldsymbol{Ax} = \boldsymbol{b}$, type `solve(A,b)`. If $\boldsymbol{A}$ is invertible, typing `solve(A)` without the second argument returns the inverse $\boldsymbol{A}^{-1}$.

```
> A <- matrix(c(2,0,1,1,-1,4,3,1,0),nrow=3,byrow=T)
> A
     [,1] [,2] [,3]
[1,]    2    0    1
[2,]    1   -1    4
[3,]    3    1    0
> b <- c(0,1,1)
```

```
> solve(A,b)
[1] -0.5  2.5  1.0
> A %*% c(-.5,2.5,1)
     [,1]
[1,]    0
[2,]    1
[3,]    1

> solve(A)
     [,1]  [,2]  [,3]
[1,]    1 -0.25 -0.25
[2,]   -3  0.75  1.75
[3,]   -1  0.50  0.50
> A %*% solve(A)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

For integer n, `diag(n)` gives the $n \times n$ identity matrix. If $\boldsymbol{x}$ is a vector, `diag(x)` returns a diagonal matrix whose diagonal elements are $\boldsymbol{x}$. If $\boldsymbol{A}$ is a matrix, `diag(A)` gives the diagonal elements of $\boldsymbol{A}$.

```
> diag(3)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
> A <- diag(c(1,4,0,2))
> A
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    4    0    0
[3,]    0    0    0    0
[4,]    0    0    0    2
> diag(A)
[1] 1 4 0 2
```

For an $n \times n$ matrix A, the command `eigen(A)` returns the eigenvalues and eigenvectors of A in a two-component list. The first component of the list `eigen(A)$values` is a vector containing the n eigenvalues. The second component `eigen(A)$vectors` is an $n \times n$ matrix whose columns contain the corresponding eigenvectors. To illustrate, we diagonalize the matrix $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$. That is, we find an invertible matrix S, and a diagonal matrix D

such that $A = SDS^{-1}$. The diagonal elements of D are the eigenvalues of A. The columns of S are the corresponding eigenvectors.

```
> A <- matrix(replicate(9,1),nrow=3)
> A
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
[3,]    1    1    1
> eigen(A)$values
[1] 3 0 0
> eigen(A)$vectors
           [,1]        [,2]        [,3]
[1,] 0.57735027  0.70710678  0.40824829
[2,] 0.57735027 -0.70710678  0.40824829
[3,] 0.57735027  0.00000000 -0.81649658
> D <- diag(eigen(A)$values)
> D
     [,1] [,2] [,3]
[1,]    3    0    0
[2,]    0    0    0
[3,]    0    0    0
> S <- eigen(A)$vectors
> S
           [,1]        [,2]        [,3]
[1,] 0.57735027  0.70710678  0.40824829
[2,] 0.57735027 -0.70710678  0.40824829
[3,] 0.57735027  0.00000000 -0.81649658
> S %*% D %*% solve(S)  # SDS^-1 = A
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
[3,]    1    1    1
```

Exercises:

11.1. Let $A = \begin{pmatrix} 1 & 3 & -1 \\ 2 & 1 & 0 \\ 4 & -2 & 3 \end{pmatrix}$. Find the following:

(i) A^3

(ii) A^T

(iii) A^{-1}

(iv) $p(A)$, where $p(x) = x^5 - 3x^2 + 7$.

11.2. Construct a random 4×4 matrix each of whose elements is uniformly distributed on (0, 1). Find the eigenvalues.

11.3. Solve the linear system

$$\begin{aligned} 2x + 3y - \ z &= 1, \\ x + \ y + \ z &= 2, \\ x + 2y - 3z &= 3. \end{aligned}$$

TABLE A.3 Summary of R Commands for Matrix Algebra

Command	Description
`matrix(vec)`	Creates a matrix from a vector
`A+B`	Matrix addition
`s*A`	Scalar multiplication
`A*B`	Elementwise multiplication
`A %*% B`	Matrix multiplication
`x %*% y`	If $\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors, returns dot product
`solve(A)`	Inverse $\boldsymbol{A}^{-1}$
`solve(A,b)`	Solves the linear system $\boldsymbol{Ax} = \boldsymbol{b}$.
`t(A)`	Transpose $\boldsymbol{A}^T$
`det(A)`	Determinant of A
`diag(s)`	If s is a scalar, creates the $s \times s$ identity matrix
`diag(x)`	If $\boldsymbol{x}$ is a vector, creates diagonal matrix whose diagonal elements are $\boldsymbol{x}$
`diag(A)`	If $\boldsymbol{A}$ is a matrix, returns the diagonal elements
`eigen(A)`	Eigenvalues and eigenvectors of $\boldsymbol{A}$ `eigen(A)$values` is vector of eigenvalues `eigen(A)$vectors` is matrix of eigenvectors